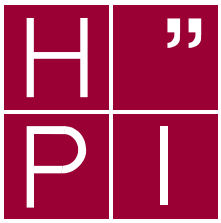
HASSO-PLATTNER-INSTITUT
für Softwaresystemtechnik an der Universität Potsdam



Java Language Conversion Assistant

An Analysis

Stefan Richter
Stefan Henze
Eiko Büttner
Steffen Bach
Andreas Polze
(eds.)

Technische Berichte
des Hasso-Plattner-Instituts
für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik
an der Universität Potsdam

4

Java Language Conversion Assistant
An Analysis

Stefan Richter, Stefan Henze, Eiko Büttner, Steffen Bach, Andreas Polze (eds.)

April 21, 2004

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber:	Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam
Redaktion	Stefan Richter, Stefan Henze, Eiko Büttner, Steffen Bach, Andreas Polze
EMail	{stefan.richter, stefan.henze, eiko.buettner, steffen.bach, andreas.polze}@hpi.uni-potsdam.de
Vertrieb:	Universitätsverlag Potsdam Postfach 60 15 53 14415 Potsdam Fon +49 (0) 331 977 4517 Fax +49 (0) 331 977 4625 e-mail: ubpub@rz.uni-potsdam.de http://info.ub.uni-potsdam.de/verlag.htm
Druck	allprintmedia gmbH Blomberger Weg 6a 13437 Berlin email: info@allprint-media.de

Heft 4 (2004)
ISBN 3-937786-10-4
ISSN 1613-5652

Contents

chapter 1 Introduction

This document is an analysis of the *Java Language Conversion Assistant* . It will also cover a language analysis of the *Java Programming Language* as well as a survey of related work concerning Java and C# interoperability on the one hand and language conversion in general on the other.

Part I deals with language analysis.

Part II covers the JLCA tool and tests used to analyse the tool. Additionally, it gives an overview of the above mentioned related work.

Part III presents a complete project that has been translated using the JLCA.

Within the sector of language conversion and interoperability, a broad spectrum of tools and techniques have been developed. Different representations of programs and methods of interoperability build up a tree that is shown in figure 1.1.

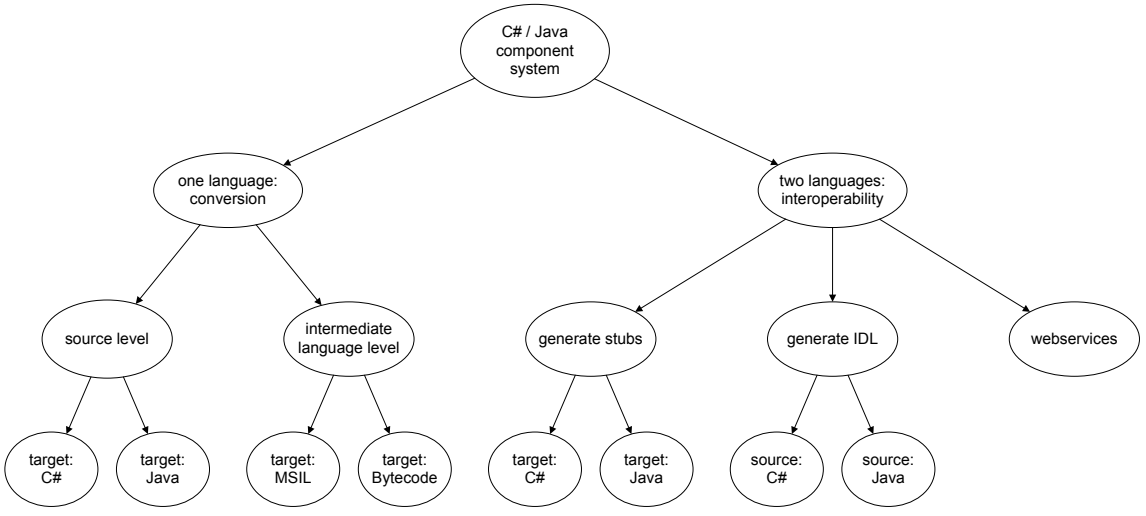


Figure 1.1: Tree of Representation and Interoperability

Now, it would be desirable to convert from one representation to another or to connect interoperability endpoints in different occurences. Some tools can convert or compile whole programs whereas others integrate Java and C# programs by

translating their respective communication mechanisms or via Corba. Figure 1.2 shows a matrix that displays if and with which tool the different representations may be converted or how they may interoperate.

Source \ Target		Sourcecode		Bytecode		CORBA				RPC	
		C#	Java	MSIL	Java	C# stubs	Java stubs	OMG IDL	SOAP	Java	.NET
Sourcecode	C#	n/a		csc		n/a	n/a	IIOP.NET	MS		
	Java	JLCA	n/a	GCC	javac	n/a	n/a				
Bytecode	MSIL			n/a		n/a	n/a				
	Java				n/a	n/a	n/a				
CORBA	C# stubs	n/a	n/a	n/a	n/a	n/a					
	Java stubs	n/a	n/a	n/a	n/a	JLCA / Janeva	n/a				
	OMG IDL	IIOP.NET	CORBA	n/a	n/a	IIOP.NET / Janeva		n/a			
	SOAP	MS	J2EE	n/a	n/a				n/a		
RPC	Java									Java	Ja.NET
	.NET									Ja.NET	.NET

Figure 1.2: Conversion and Interoperability

I
part

Language Analysis

chapter 2 Overview

Before being able to evaluate a language conversion tool satisfyingly, one must gain a good knowledge of both, source and target language. Simply said, it is necessary to know how each construct of the source language can be expressed in the target language.

As *Java* is not only a programming language but also an execution environment it is sensible to have a comparative look at the Java programming language and the C# programming language as well as to compare the Java execution environment to the .NET execution environment including the most basic libraries.

2.1 Java Programming Language

The Java programming language and the C# programming language are syntactically similar as their syntaxes are both derived from the C/C++ family. Still, there are some differences concerning all kinds of language constructs.

These constructs may be divided into:

- Entity declarations and modifications
- Statements
- Expressions

Each Java program consists of a set of entities that define the environment[1] in which the program's code will run. These entities are packages, types, type members, methods, local variables, method parameters and return values. All these entities are related to each other in some way or the other. Entities and their relationships are subject to chapter 3 on page 15.

One kind of them, namely methods, contains the program's code which is a sequence of statements intermingled with expressions. Chapter 4 on page 23 focuses on statements and expressions.

[1] This environment should not be confused with the Execution Environment.

The following list shows Java language constructs according to *The Java Language Specification* and marks if they are treated by this report. If so, the column *C&E*[2] shows how large the conversion effort is when using the JLCA.

Issue	**C&E**[2]
Lexical Structure	todo
Character sets	ok
Identifiers	ok
Keywords	3
Literals	ok
Comments	ok
Operators	todo
Type System	todo
strongly typed	ok
primitive types	ok
integral types (range, operators, casting)	ok
float types (format)	ok
boolean	ok
reference types	todo
class types	ok
interface types	todo
array types	todo
null as special type	ok
String	2
Object	todo
Variables	todo
variable types (local, field)	ok
modifiers (final)	todo
lifetime	todo
default values	ok
Conversions	todo
Names	–
Scope of declarations	–
Solving ambiguous names	–
Packages	–
Members	–
Compilation units	–
Importing	–
Classes	todo
Declaration	1
Modifiers	ok

[2]Footnote here

Issue	**C&E**[2]
Inner Classes and Enclosing Instances	1
Superclasses, Subclasses, Superinterfaces	ok
Class Body and Member Declarations	ok
Class Members	todo
Inheritance with public, protected, private, default	1
Accessing Members of Inaccessible Classes	todo
Field Declarations	ok
Field Modifiers	ok
Initialization of Fields	ok
Method Declarations	no
Formal Parameters	ok
Method Signature	ok
Method Modifiers	ok
Method Throws	no
Method Body	ok
Inheritance, Overriding, and Hiding	ok
Overloading	ok
Member Type Declarations	ok
Instance Initializers	ok
Static Initializers	ok
Constructor Declarations	no
Formal Parameters	ok
Constructor Signature	ok
Constructor Modifiers	ok
Constructor Throws	no
Constructor Body	ok
Explicit Constructor Invocations	ok
Constructor Overloading	ok
Default Constructor	todo
Preventing Instantiation of a Class	todo
Interfaces	–
Interface Declarations	–
Interface Modifiers	–
Superinterfaces and Subinterfaces	–
Interface Body and Member Declarations	–
Access to Interface Member Names	–
Interface Members	–
Field (Constant) Declarations	–
Initialization of Fields in Interfaces	–
Abstract Method Declarations	–
Inheritance and Overriding	–

Issue	**C&E[2]**
Overloading	–
Member Type Declarations	–
Arrays	–
Array Types	–
Array Variables	–
Array Creation	–
Array Access	–
Array Initializers	–
Array Members	–
Class Objects for Arrays	–
An Array of Characters is Not a String	–
Array Store Exception	–
Exceptions	todo
Causes of Exceptions	todo
Compile-Time Checking of Exceptions	todo
Handling of an Exception	todo
Exception Hierarchy	todo
Blocks and Statements	–
Blocks	ok
Local Class Declarations	–
Local Variable Declaration Statements	–
Statements	todo
The Empty Statement	ok
Labeled Statements	no
Expression Statements	ok
The if Statement	ok
The if-then Statement	ok
The if-then-else Statement	ok
The switch Statement	ok
The while Statement	ok
The do Statement	ok
The for Statement	ok
The break Statement	ok
The continue Statement	ok
The return Statement	ok
The throw Statement	ok
The synchronized Statement	todo
The try statement	ok
Unreachable Statements	todo
Expressions	todo
Evaluation, Denotation, and Result	todo

Issue	C&E[2]
Variables as Values	todo
Type of an Expression	todo
FP-strict Expressions	no
Expressions and Run-Time Checks	todo
Normal and Abrupt Completion of Evaluation	todo
Evaluation Order	todo
Primary Expressions	todo
Lexical Literals	todo
Class Literals	todo
this	todo
Qualified this	todo
Parenthesized Expressions	todo
Class Instance Creation Expressions	todo
Array Creation Expressions	todo
Run-time Evaluation of Array Creation Expressions	todo
Field Access Expressions	todo
Method Invocation Expressions	todo
Compile-Time Step 1: Determine Class or Interface to Search	todo
Compile-Time Step 2: Determine Method Signature	todo
Compile-Time Step 3: Is the Chosen Method Appropriate?	todo
Runtime Evaluation of Method Invocation	todo
Array Access Expressions	todo
Postfix Expressions	ok
Postfix Increment Operator ++	ok
Postfix Decrement Operator –	ok
Unary Operators	ok
Prefix Increment Operator ++	ok
Prefix Decrement Operator –	ok
Unary Plus Operator +	ok
Unary Minus Operator -	ok
Bitwise Complement Operator	ok
Logical Complement Operator !	ok
Cast Expressions	todo
Multiplicative Operators	ok
Multiplication Operator *	ok
Division Operator /	ok
Remainder Operator %	ok
Additive Operators	ok
String Concatenation Operator +	ok
Additive Operators (+ and -) for Numeric Types	ok

Issue	C&E[2]
Shift Operators	ok
Relational Operators	ok
Numerical Comparison Operators <, <=, >, and >=	ok
Type Comparison Operator instanceof	ok
Equality Operators	3
Numerical Equality Operators == and !=	ok
Boolean Equality Operators == and !=	ok
Reference Equality Operators == and !=	3
Bitwise and Logical Operators	ok
Integer Bitwise Operators &, ^, and |	ok
Boolean Logical Operators &, ^, and |	ok
Conditional Operators &	ok
Conditional-And Operator &&	ok
Conditional-Or Operator ||	ok
Conditional Operator ? :	ok
Assignment Operators	todo
Simple Assignment Operator =	ok
Compound Assignment Operators	todo
Expression	todo
Constant Expression	todo
Definite Assignment	todo
Short summary as C# and Java should not differ that much and the matter is not really easy	todo

2.2 Java Execution Environment

Because of the huge amount of standard libraries that are shipped with the Java Execution Environment and that greatly support software development, it is not possible to examine them comprehensively in this report.

It is rather a good idea to pick out some mechanisms that seem worth an inspection in detail. This might be justified by either technical or commercial interest. Thus, technical interest would be the main reason for the examination of the Java Server Pages while reflections on Graphical User Interfaces would get their legitimation from the software industry's need to convert their Java Swing front-end client applications into Microsoft .NET Windows applications.

The libraries that would be most interesting for an inspection in detail are shown in the list.

Issue	**C&E**[2]
Core	todo
`java.lang`	todo
`java.lang.ref`	todo
`java.lang.reflection` (just a comment that mapping reflection is a hard task)	todo
`java.applet`	todo
`java.text`	todo
`java.math`	todo
Utilities	–
`java.io`	–
`java.net`	–
`java.nio`	–
`java.nio.channels`	–
`java.nio.channels.spi`	–
`java.nio.charset`	–
`java.nio.charset.spi`	–
`java.util`	–
`java.util.jar`	–
`java.util.logging`	–
`java.util.prefs`	–
`java.util.regex`	–
`java.util.zip`	–
`javax.net`	–
`javax.net.ssl`	–
GUI	–
`java.awt`	–
`java.awt.color`	–
`java.awt.datatransfer`	–
`java.awt.dnd`	–
`java.awt.event`	–
`java.awt.font`	–
`java.awt.geom`	–
`java.awt.im`	–
`java.awt.im.spi`	–
`java.awt.image`	–
`java.awt.image.renderable`	–
`java.awt.print`	–
`java.beans`	–
`java.beans.beancontext`	–
`javax.accessibility`	–
`javax.swing`	–
`javax.swing.border`	–

Issue	C&E[2]
javax.swing.colorchooser	–
javax.swing.event	–
javax.swing.filechooser	–
javax.swing.plaf	–
javax.swing.plaf.basic	–
javax.swing.plaf.metal	–
javax.swing.plaf.multi	–
javax.swing.table	–
javax.swing.text	–
javax.swing.text.html	–
javax.swing.text.html.parser	–
javax.swing.text.rtf	–
javax.swing.tree	–
javax.swing.undo	–
org.eclipse.swt.*	–
Media Utilities	–
javax.imageio	–
javax.imageio.event	–
javax.imageio.metadata	–
javax.imageio.plugins.jpeg	–
javax.imageio.spi	–
javax.imageio.stream	–
javax.print	–
javax.print.attribute	–
javax.print.attribute.standard	–
javax.print.event	–
javax.sound.midi	–
javax.sound.midi.spi	–
javax.sound.sampled	–
javax.sound.sampled.spi	–
Security	–
java.security	–
java.security.acl	–
java.security.cert	–
java.security.interfaces	–
java.security.spec	–
javax.crypto	–
javax.crypto.interfaces	–
javax.crypto.spec	–
javax.security.auth	–
javax.security.auth.callback	–
javax.security.auth.kerberos	–
javax.security.auth.login	–

Issue	**C&E**[2]
javax.security.auth.spi	–
javax.security.auth.x500	–
org.ietf.jgss	–
XML	–
javax.xml.parsers	–
javax.xml.transform	–
javax.xml.transform.dom	–
javax.xml.transform.sax	–
javax.xml.transform.stream	–
org.w3c.dom	–
org.xml.sax	–
org.xml.sax.ext	–
org.xml.sax.helpers	–
CORBA and RMI	–
java.rmi	–
java.rmi.activation	–
java.rmi.dgc	–
java.rmi.registry	–
java.rmi.server	–
javax.naming	–
javax.naming.directory	–
javax.naming.event	–
javax.naming.ldap	–
javax.naming.spi	–
javax.rmi	–
javax.rmi.CORBA	–
javax.transaction	–
javax.transaction.xa	–
org.omg.CORBA	–
org.omg.CORBA.DynAnyPackage	–
org.omg.CORBA.ORBPackage	–
org.omg.CORBA.portable	–
org.omg.CORBA.TypeCodePackage	–
org.omg.CORBA_2_3	–
org.omg.CORBA_2_3.portable	–
org.omg.CosNaming	–
org.omg.CosNaming.NamingContextExtPackage	–
org.omg.CosNaming.NamingContextPackage	–
org.omg.Dynamic	–
org.omg.DynamicAny	–
org.omg.DynamicAny.DynAnyFactoryPackage	–
org.omg.DynamicAny.DynAnyPackage	–
org.omg.IOP	–

Issue	**C&E**[2]
org.omg.IOP.CodecFactoryPackage	–
org.omg.IOP.CodecPackage	–
org.omg.Messaging	–
org.omg.PortableInterceptor	–
org.omg.PortableInterceptor.ORBInitInfoPackage	–
org.omg.PortableServer	–
org.omg.PortableServer.CurrentPackage	–
org.omg.PortableServer.POAManagerPackage	–
org.omg.PortableServer.POAPackage	–
org.omg.PortableServer.portable	–
org.omg.PortableServer.ServantLocatorPackage	–
org.omg.SendingContext	–
org.omg.stub.java.rmi	–
Database Access	–
java.sql	–
javax.sql	–

Entity Declarations and Modifiers

chapter 3

Writing programs is defining a sequence of instructions that operate on memory locations. Many high-level programming languages coerce the programmer to define these locations' structure and to modularize the sequence of instructions. Thus, high-level programming is mainly structuring of code and data by declaring[1] and defining[2] entities. This chapter compares the principal approaches of Java and C# concerning this structuring.

3.1 Code Structuring

3.1.1 Java

In the Java programming language on the most basic level, code is modularized into methods and constructors[3], i.e. each instruction must be within a method or constructor, or more precisely: each instruction has a unique position in the (one and only) code block that is associated with a method. Each method is associated with exactly one class while a class may contain an arbitrary number of methods.

A class must be contained in a package or another class. This package or class is the container of the class. Beside its class name, a class also has a full name, which is the name of the container followed by `.` and the class name. References to other classes can always be expressed with the full name but classes in the same package can be referred to by ommitting the package name. Using the keyword `import` followed by a package name allows to omit that package's name, too. Additionally, access to a class depends on the containers of the accessing and the accessed class and the latter one's modifiers.

Note, that each class that is directly contained in a package must be completely declared and defined in one file that has the same name as the class plus the file

[1] saying that an entity must exist

[2] saying how the entity works and/or looks like

[3] Most of the time, the term *method* includes constructors, too. However, in certain contexts it should come out clearly that *method* only refers to methods.

ending `java` for source files or `class` for binaries respectively. Additionally, the file system path of the file must correspond to the package. Thus, the binaries have the same file system structure as the source files, except for inner classes (i.e. classes that are contained in another class). These classes are declared and defined in the same file as their parent class but their binaries are separate files in the same directory, prefixed with the parent class's name plus `$`.

Packages are the only entities that are not contained in other entities. Even if it might seem that for example the package `java.lang.reflection` is part of the package `java.lang` it is not. This confusion exists because in the file system, each part of a package is a subdirectory of the former part.[4] If a class does not specify explicitly its package, its package is the (implicitly defined) *default* package that has no name and is located in the program's root directory.

Finally, each program must have an entrypoint. In Java, a program can share parts of its code with other programs by using the same classes. Therefore, the entrypoint of a program is defined as a method with the signature `public static void main(String[] args)`. As each class may have at most one method with the same name and parameter signature, each class may have at most one entrypoint.

3.1.2 C#

The basic code structuring in C# is similar to the one in Java: code is modularized into methods and classes. However, the idea about what a program is differs in both languages (and the execution environments).

First, the declaration and definition of a code is nearly completely independent of the file system structure. Still, all of a class's declarations and definitions must be contained in one file. But that file can contain other classes and thus may have an arbitrary name and location.

Furthermore, binaries and source files do not share a common file system structure. Instead, binaries are organized in assemblies which may consist of one or more files.[5] These files do not have to be located at certain places in the file system.[6] From the programming language's point of view, an assembly is the output of one compiler session.

A program or application is an assembly which has an entrypoint. This entrypoint must be a static method named `Main` that can have a single parameter of type `String[]` (or not) and whose return type must be either `int` or `void`. Each application must have exactly one entry point. For that reason, Java applications

[4] The package `this.is.a.package` has the (local) path `/this/is/a/package` or `\this\is\a\package`.

[5] However, tools like Microsoft Visual Studio do not support multifile assemblies, yet.

[6] Advanced concepts like the Global Assembly Cache are out of scope of this report.

that have more than one entrypoint must be converted into several assemblies. Unfortunately, this can lead to further problems because of code dependencies and code access restrictions between the applications. In the worst case, the entrypoint structure of such applications must be redesigned.

Concerning accessibility, assemblies play a similar role as packages do in Java. However, this is not true for full names of classes. This aspect of Java packages is realized by namespaces. Similar to packages, namespaces do not contain each other[7] but only classes and each class is in a namespace or in another class (which is its container). Thus, the full name of a class in C# is the class name preceded by the container's name. Namespaces have no special relation to assemblies, i.e. namespaces can span several assemblies while assemblies can contain more than one namespace.

3.2 Data Structuring

3.2.1 Memory Allocation and Deallocation

All the memory that can be used by a Java or C# program is allocated either in the form of an object or as the set of local variables and arguments for a method call. Each time a method is called, the necessary amount of memory for all local variables and arguments must be allocated. That local variables memory will be released after the method call has returned. In general, local variables memory is allocated on the stack.

In contrast, the memory for an object is allocated if and only if the `new` instruction is executed.[8] That piece of memory is released after the execution environment is sure that the object will never be used again in the program. I.e. there is no possibility to release the memory occupied by an object. The object memory is not allocated on the stack but has its own, method independent space.

There are no other ways to tell the compiler to allocate memory. However, the compiler can collect all necessary information about the needed memory when looking at the definitions of methods and classes. The memory needed by a method call is the sum of the memory needed by each local variable and argument. The memory occupied by an object is the sum of all its member fields and methods (for inheritance).

Local variables, arguments and fields have in common that they are typed. This

[7] And similar to packages one might think that they contain each other due to the syntax: `namespace two { namespace name_spaces {}}` defines two namespaces `two` and `two.name_spaces`.

[8] In fact, the boxing mechanism in C# could be seen as another possibility to allocate memory. But as boxing is nothing more than allocating an appropriate object and assigning that value to it, it will not be considered separately as all the more Java cannot box.

type specifies which kind of data may only be written into that piece of memory and how the data is interpreted when reading it. A type may either be a primitive type or a reference type.

3.2.2 Primitive Types

Java knows the primitive types listed below (with the respective value set):

`boolean`	true, false
`byte`	8 bit signed integer
`short`	16 bit signed integer
`int`	32 bit signed integer
`long`	64 bit signed integer
`char`	16 bit Unicode (or 16 bit unsigned)
`float`	32 bit floating point
`double`	64 bit floating point

In contrast, C# can cope with:

`bool`	true, false
`byte`	8 bit unsigned integer
`sbyte`	8 bit signed integer
`ushort`	16 bit unsigned integer
`short`	16 bit signed integer
`uint`	32 bit unsigned integer
`int`	32 bit signed integer
`ulong`	64 bit signed integer
`long`	64 bit signed integer
`char`	16 bit Unicode (or 16 bit unsigned)
`float`	32 bit floating point
`double`	64 bit floating point
`decimal`	precise decimal with 28 significant digits

Despite minor differences, it is clear that C# comprises Java's set of primitive types.

3.2.3 Reference Types

All other data types in Java are reference types. This means, that a variable of such a type always holds a reference to an object of that type or the special value represented by the literal `null`. On the other hand, each object of that type can only be referenced to by a variable of that type or an ancestor type. Reference types can be divided into classes and interfaces. In addition to those reference types, C# knows value types which are called structs and which are a conceptual mix of classes and primitive types.

Classes

A class is not only a container for methods and other classes but also a template for the creation of objects. Therefore, classes can contain typed fields.

In Java, each object consists of the sufficient amount of memory for each member field and a slot for a reference to each member method. This slot is necessary for object oriented concepts like polymorphism. Moreover, classes can contain fields, that do not belong to a certain object and therefore exist only once for all objects of this class. These fields are called static. Similar, static methods are methods, that have no slot in each object but that belong to the class. C# adds methods, that are not static but do not need a slot for each method. All this will be explained in section 3.3.

Interfaces

Interfaces are some kind of tags that are associated with types. They express that objects of such a type have methods.

Java interfaces can consist of a set of member methods and a set of constant static fields. Each type that implements such an interface must provide for methods with the same signature. Contrary, the static fields only exist in the interface.

Unfortunately, C# splits this up: interfaces must not contain fields. Therefore, a Java interface with fields and methods must be represented by two entities in C#: A class that contains the static fields and an interface that contains the member methods. Alternatively, C# provides for enumerations, that are essentially classes that only contain constant static fields.

Structs

Structs are structured value types whose main difference towards classes is that they are not reference types but some kind of user defined primitive types.[9] Therefore, instances of structs are created as local variables. If a struct consists of two `int`s, a declaration of a local variable of that struct would reserve the memory for two `int`s in the local variables memory. In contrast, the declaraion of a local variable whose type is a class with two `int`s would reserve the memory for two `int`s in the object memory and a reference to that memory in the local variables memory.

Furthermore, an argument that is of a struct type will be copied in its whole each time the method is called. Analogously, the same is true for return values that are

[9] Although Java does not know such types it is necessary to have some facts about them in mind because great parts of the .NET framework use them in signatures. Thus, unless you want to rewrite that functionality, the C# mapping of your Java code must use structs.

structs.

Since structs cannot be referenced, they can never have the value `null`.

3.3 Object Model

Basically, the object models of Java and C# are very similar. First, objects are created using a constructor and the `new` operator. Then, the program will call methods on the objects, and finally objects will be destroyed automatically, once the execution environment has found out, that they cannot be used any more.

3.3.1 Inheritance

A class C can be derived from at most one class P by writing `class C extends P` in Java or `class C : P` in C#. Objects of type C will then have at least the same slots(member fields and references to member methods), i.e. each object of type C can be used as an object of type P.

Moreover, class C can declare additional fields and methods as well as override methods defined in P. Overriding means first, that C provides a definition for a method M of P that has the same signature, and second, that objects of type C hold a reference to C.M instead of P.M. Thus, a method call on a variable of type P could result in a method call to P.M or C.M depending on the object that is referred to by the variable. In Java, methods are overridden by defining M in C the same way as if it would not exist in P. There is no way to prevent this overriding by the derived class (e.g. if the programmer of C does not have a good knowledge of P and does not want to override any methods by accident). That's why in C#, each method in C that has the same signature as a method in P must be declared as either `override` or `new`.

To forbid the overriding of a method, the method can be declared as `final` in Java or `sealed` in C#. If these modifiers appear in the declaration of a class, no classes can be derived from that class.

Another effect of overriding is the hiding of the name of the overridden method. As long as a method is not overridden, each occurence of its name refers to it, even in derived types. But when a method is overridden, its name refers to the method defined by the overriding type when used in that overriding type and all of its subtypes. However, sometimes it is necessary to refer to the original method. In Java, this can be done using the keyword `super`, in C# it is `base`.

If a class is not derived explicitly from another class, it is derived implicitly from `java.lang.Object` (Java) or `System.Object` (C#). These classes form the root of the class inheritance tree in their respective environment.

3.3.2 Abstract Classes and Interfaces

If a class declares a method without providing a definition, this method must be declared as `abstract` in both languages. A class that contains abstract methods must be itself declared as `abstract`. This is also true, if the class inherits abstract methods, that are not implemented. Non-abstract classes that inherit from abstract classes must implement all abstract methods, overriding them.

Interfaces contain method declarations, but never method definitions. Thus, class C can implement an arbitrary number of interfaces. This must be declared in the class's signature: `class C implements A, B, C` in Java and `class C : A, B, C` in C#. If C inherits from P, P must be the first identifier immediately after `:` in C#. Implemented methods do not have to be declared as `override`.

3.3.3 Inner Classes

Inner classes are classes that are contained in another class. Java knows two types of inner classes: static and non-static inner classes. Except for not being contained in a package, static inner classes do not differ from other classes but they must be declared with the keyword `static`.

Instances of non-static inner classes can only be created and referred to in non-static contexts (i.e. member methods and constructors) of objects of the outer class because they hold an implicit reference to their encapsulating object.

In C#, there are only static inner classes. Unfortunately, they are not marked as `static`. Non-static classes' behaviour must be emulated by explicitly defining a member field of the outer type that contains a reference to the encapsulating object. This reference must be an argument of all constructors.

There is also a possibility of defining anonymous inner classes in Java which is not possible in C#. But this can easily be fixed by naming the C# counterparts of Java's anonymous classes.

3.4 Accessibility

As a means of structuring data and code and avoiding programming errors, Java allows to restrict code to access certain entities. In this section, we will concentrate on the most basic concepts which can be expressed by using language means. The more sophisticated accessibility and security concepts are subject to examinig the appropriate code libraries.

Access to classes and their members can be restricted. Therefore, there is a set of modifiers that express these restrictions. If a certain method or field can be accessed

depends its class's modifier and its own modifier as well as the relative location of the method that wants to access it and the class inheritance hierarchy.

3.4.1 Content of Packages/ Assemblies

In Java, classes and interfaces that are contained in a package can be declared as `public`. They can then be referred to from other packages; otherwise, they can only be used by code inside the same package.

In C#, types that are not part of another class can be declared as `public` or `internal` (if none of those is given, `internal` is assumed). If a type is `internal`, it can only be accessed by methods that are defined inside the same assembly; otherwise, also methods defined in other assemblies can access the type.

3.4.2 Content of Classes

If a class is accessible by a method, not all of its members[10] have to be necessarily accessible. But conversely, it is not possible that a member is accessible while its containing class is not. Instead, each member can be modified by `private`, `protected`, `public` in Java and additionally `internal` in C#. Except for the combination `protected internal`, only one access modifier can be specified.

A member that is declared as `public` can be accessed by everyone else; `private` members are only accessible by methods of the containing class (and classes that are inside that class). `protected` members are like `private` members but can be accessed also by methods of derived classes. In C#, `internal` members are accessible by methods that are defined in the same assembly; a method can access a `protected internal` member if it could access the member if the member was `protected` or `internal`.

[10]The statements in this section apply to everything inside a class: static methods, inner types, ...

Expressions and Statements

chapter 4

4.1 Expressions

An expression represents a value that must be calculated if necessary. A value is everything that can be stored in a variable. The name of a variable is therefore one of the most simple expressions indicating the value stored in that variable.

Other simple expressions are *literals*. A literal is a value that is constant in the whole program and known at compile time as for example `"This is a string literal."`, `9879` or `'a'`. Literals are `null`, `true` and `false`, too.

More complex expressions can be constructed out of simple expressions, operators and method calls, for example `123 + 987` or `54 + calculate()`. Since method calls are some kind of statement, too, they will be covered in section 4.2.

4.1.1 operators

In Java, operators are predefined while in C#, it is possible to redefine (also known as overload) operators. However, this possibility is out of scope of this report. In general, the operators work the same way in both languages. The main differences can be found when looking at the `==` operator concerning strings.

While in Java, the `==` operator always compares references when used with reference types, the same operator in C# compares the values of the strings. Normally, this difference does not come out clearly in Java programs because of a mechanism called string interning. This allows to call the method `intern()` on a string. The runtime then tries to find a string that has the same value in its internal string table. If it does not find one the reference to that string will be stored in the table and returned by `intern()`. Otherwise the reference to the string in the table will be returned. String literals are automatically interned as well as many Java libraries intern strings automatically. Thus, reference comparison is often sufficient. However, the correct translation to C# must cast the `String`s to `Object`s and compare these because objects are compared by reference. This effect is due to operator overloading in C#.

Additionally, the `instanceof` operator of Java has been replaced by the `is` operator in C#.

Most interestingly, there is a suffix `.class` which is used in Java to form the literal that denotes the type at runtime (i.e. whose value is a reference to the object of type `Class` that represents the class). This literal mechanism finds its counterpart in the C# operator `typeof()`.

4.2 Statements

A statement is an instruction that has no value and rather controls the flow of execution. However, as mentioned above, method calls can have an expression like character as they return a value. But if the return value of a method is neglected, the method call cannot be an expression. As with expressions, most of the statements in Java and C# are identical. These are the `if`, `then`, `else`, `for`, `while`, `do`, and `return` statements.

The statements that deal with excpetion handling (`throw`, `try`, `catch`, `finally`) will be covered in the appropriate section 5.1

Note that there are three ways to terminate a statement. First, the statement can be processed completely, which is called *to complete normally*. Second, a `return`, `break` or `continue` statement (additionally the `goto` statements in C#) can transfer the flow of execution out of the statement without completing normally. Third, an exception (or more precisely an object of type `Throwable` in Java or of type `Exception` in C#) might occur. These both ways are called *to complete abruptly*.

4.2.1 The `switch` Statement

In Java, `switch` statements can be used for multiple decisions. The same statement exists for the same purpose in C# but the `case` substatements work a little bit different.

While in Java, the flow of execution falls through the labels denoted with `case` unless it completes abruptly, C# requires that no `case` section has a reachable endpoint. Therefore, each section must complete abruptly.[1] However, to allow fall-through behaviour, the `goto case` statement followed by a label or the `goto default` statement can be used.

[1] Several labels can form one `case` section if their declarations follow immediately: `case label1: case label2: method_call(); break;`.

4.2.2 The `break` and `continue` Statements

The unlabeled version of these statements has the same effect in both languages: they transfer the control out of a `for`, `while`, or `do` loop (unless `break` is used as part of the `switch` statement). `break` completes abruptly the loop, while `continue` completes abruptly the iteration and therefore makes the loop check if another iteration must be made.

Statements can be prefixed with labels. If a loop is prefixed by a label L, a `break` statement followed by L will complete normally that loop in Java. This can be used to break out of nested loops, as it would else not be possible to break an outer loop from an inner loop without using state variables. Similar, the labeled `continue` statement completes the respective loop iteration.

In C#, there are no labeled versions of `break` and `continue`. Instead, the `goto` statement can be used to transfer control out of loops but not to transfer control into loops. The label immediately after `goto` can be declared at any legal position inside the same method or constructor.

However, this is not an equivalent translation. To transform `break` statements, the label in the C# program must be declared immediately after the loop that bears the label in the Java program. In contrast, `goto` can be used in the C# program like the labeled `continue` in Java. In the C# program, one could as well set the label followed by an empty statement (`label:;`) at last position inside the loop that bears the label in Java.

4.2.3 The `synchronized` Statement and Modifier

The `synchronized` statement opens a block of code that is intended for use with concurrent programming and is equivalent to the `lock` statement in C#. Both statements require an object to lock on and guarantee the correct locking and unlocking, no matter if the code block completes abruptly or normally.

However, and little bit off topic of this section, there is a `synchronized` modifier for methods in Java, too. This modifier expresses that all methods in the class, that have this modifier, share the same locking object and synchronize their whole code. This modifier does not exist in C# but can be easily emulated. Therefore, each class, that has `synchronized` methods must have one lock for all static methods and one lock for each instance for all instance methods in C#. Note that this second lock should be accessible in a protected field as all instance methods in derived types must use the same lock. Furthermore, the whole code of each `synchronized` method must be contained in one single `lock` statement in C# that locks on the respective lock object.

chapter 5 Execution Environment

This chapter is a very short overview of the execution environment. It contains a section about exception handling and a section about some basic mappings between types.

5.1 Exception Handling

The concepts of exceptions are very similar in both languages, although there are two major differences: First, methods must declare which exceptions they throw using the keyword `throws` or catch every exception that methods which are called by them have declared to throw unless these exception belong to the category of runtime exceptions. Moreover and more important, the exception type hierarchies differ.

As it would be rather complicated to enforce the programmer to make her methods declare the possibly thrown exceptions[1] and as it does not affect the program's behaviour, this topic can be neglected when talking about conversion of complete programs.

Unfortunately, the differences in the exception type hierarchies can lead to worse and subtile problems. In Java, every object of type `Throwable` can be thrown by the statement `throw`. There are two predefined subclasses of this class: `Error` and `Exception`. `Error`s are supposed not to be caught by a reasonable application except for very rare cases.

In C#, the exception type hierarchy starts with `Exception`, i.e. an object can be thrown by the `throw` statement iff it is of type `Exception`. Therefore, it seems reasonable to map Java's `Exception` on C#'s `Exception`. But what would then be the mapping of `Throwable`, `Error` and application defined exceptions that are derived from `Throwable`?

This question suggests, that it could possibly be impossible to map the sophisticated

[1] One way would be to define an attribute in C# whose parameters would be the exceptions. But still without means of modifying the compiler, this does not enforce the programmer to do so.

exception handling structure of some Java application to an appropriate C# program if the application uses standard libraries that throw exceptions and handles these exceptions together with their own exceptions.

There are also some minor differences concerning the syntax. While the use of the `throw`, `try`, and `finally` statements is the same, `catch` can be used slightly different: In Java, `catch` must always be followed by a variable in brackets that is of a type derived from `Throwable`. In C#, there are shortcuts. Thus, it is sufficient to specify only the class without a variable if the variable is not needed in the `catch` block. If even the class is omitted the `catch` block catches all exceptions (like `catch (System.Exception)`).

5.2 Basic Mappings

Note that these mappings are just a rough orientation. Often, methods are not mappable easily.

`java.lang.Object`	`System.Object`
`java.lang.Class`	`System.Type`
`java.lang.Math`	`System.Math`
`java.lang.String`	`System.String`
`java.lang.Thread`	`System.Threading.Thread`
`java.util.ArrayList`	`System.Collections.ArrayList`
`java.util.Calendar`	`System.Globalization.Calendar`
`java.util.Date`	`System.DateTime`
`java.util.Hashtable`	`System.Collections.Hashtable`
`java.ref.WeakReference`	`System.WeakReference`

II part

Java Language Conversion Assistant

Based on the language survey in Part I, this part examines the Java Language Conversion Assistant (JCLA).
First of all, the JLCA is introduced in chapter 6. Afterwards, chapters 7 and 8 deal with tools that are needed for the conversion of large Java projects. Chapter 9 examines how the JLCA performs when converting selected Java programs. Chapter 10 briefly introduces a number of tools that have similar purposes as the JLCA.

chapter 6 Introduction

The Java Language Conversion Assistant (JLCA) is a tool that helps with the automatic conversion of Java source code to C# source code. The JLCA is able to automatically convert most of Java's syntactic constructs[1] as well as the usage of many classes of the Java class library.

The JLCA comes in two different flavors - as a command line tool and as a plug-in for Visual Studio .NET 2003. Both versions seem to be built on top of the same core and thus offer the same functionality.

The JLCA can handle normal console and GUI programs as well as web applications that are usually deployed in a Java Application Server[2].

For each set of Java source files the JLCA converts, a new Visual Studio project is created. Additionally, the tool writes a conversion report that contains a list of all problems encountered during the conversion.

Currently, it is not directly possible to convert an application step by step. Even applications consisting of many loosely coupled modules must be converted in one step. This is due to the fact that the JLCA does not know how to resolve references to libraries that are not part of the Java class library. If an application was converted in multiple steps several class libraries would normally be created all of which would be used by a main module. The JLCA would be unable to automatically resolve references from the main module to the previously converted class libraries. Chapter 8 deals with the Extensibility Kit. This kit, which allows the definition of custom conversion rules, can be employed to solve the above-mentioned problem.

Furthermore, the JLCA can currently not be used to repeatedly convert Java projects that are still under development. Usually, manual changes have to be applied to the C# code that is created by the JLCA. These changes would have to be re-applied again and again after each conversion as the JLCA always makes the same errors. Chapter 7 deals with some shell scripts that provide a solution to this problem.

---

[1] In fact, the only construct that is not automatically converted is Java's `assert` keyword. Furthermore, Java and C# sometimes treat identifiers differently. In rare cases, this may lead to problems as well.

[2] This document deals with console and GUI applications only.

6.1 Conversion Example

In the following, a small example will illustrate how the JLCA can be used to convert Java source code to C# source code. Suppose we have a very simple Java source file that contains nothing but an empty class declaration.

```
1 public class EmptyClass{
2 }
```

Listing 6.1: EmptyClass.java

When the JLCA converts this class it creates a C# source file that contains exactly the same empty class definition.

```
1 using System;
2 public class EmptyClass
3 {
4 }
```

Listing 6.2: EmptyClass.cs

Furthermore, the JLCA creates a number of other files:

AssemblyInfo.cs This file contains information about the .NET assembly that is created when the C# project is compiled.

EmptyClass.csproj This is the Visual Studio .NET 2003 project file.

EmptyClass.xml This file contains an XML representation of all problems the JLCA encountered during the conversion process. It contains the same information as the conversion report.

_ConversionSummary.txt This file contains a short summary of what has been converted, namely how many files have been converted and what sizes these files have.

_ConversionReport.htm This file contains an HTML representation of the conversion report. It can be viewed with any modern web browser.

_ConversionReport_Files This directory contains some auxiliary files for the conversion report.

For the above example the conversion report contains only one issue which states that the "interaction between members of a class may differ because their execution sequence is different." This issue is automatically added to each conversion report. It does not indicate any real problems and should be considered as a general warning that the JLCA can never make sure that a converted program has the same behavior as the original program. After all, the A in JLCA stands for "assistant".

Obviously, this small example does not show much of what the JLCA is able to do or not to do. Chapters 7, 8 and 9 provide more insight into the capabilities of the JLCA.

chapter 7 Diff-Patch Tool

This chapter deals with a tool suite that was developed in the course of the JLCA evaluation. This suite helps to solve problems that arise when the JLCA is repeatedly used to convert a software project that is still under development[1].

7.1 Motivation for the Diff-Patch Tool

Imagine the following situation: The source code of a large Java application is to be converted to C#. After the conversion, a lot of manual changes have to be applied to the C# code in order to make it compilable and correct. However, as software projects are never really completed the Java sources are changed afterwards and the conversion has to be performed a second time. If the JLCA alone was used for this purpose all changes would have to be made a second time. All the manual work that was carried out during the first conversion would be lost.

The tool suite solves this issue by remembering the changes that are applied to the C# code after each conversion. Each time the Java sources are converted anew, the tool suite tries to apply these changes to the new version of the C# code. For accomplishing these tasks, the tool suite uses the diff and patch tools that are well-known from Unix platforms. Currently, the tool suite consists of some shell scripts that run under the Cygwin environment on any Win32 platform. However, it should be very easy to adapt the scripts so they run on the Interix platform for example.

7.2 The Tool Suite

The tool suite consists of the following shell scripts and related files:

jlca.bash This is the main script of the tool suite. It starts the JLCA and automatically patches the C# code after a conversion.

[1]Please note that the current version of the tool suite is no more than a proof a concept. It is not very flexible yet and does not feature a very convenient user interface.

When started, this script looks for a directory named “java” where it expects to find the Java sources that are to be converted. During its execution, two directories are created - “jlca_orig” and “jlca_corrected”. The files in “jlca_orig” should not be modified. They contain the unpatched C# sources which are needed for the diff tool. The second directory contains the patched C# code. All changes to the C# code have to be made in this directory.

jlca.bash will only start the JLCA if the Java sources have been modified since the last conversion.

makediff.bash This script is used to start the diff tool. It creates the file “diff.txt” which contains information about all changes that were applied to the C# code. The file is located in a directory named “diff”. The input for the diff tool are the directories “jlca_orig” and “jlca_corrected”.

applydiff.bash This script is used to patch the C# code after a conversion. It uses the file “diff.txt” in the directory “diff”. During its execution, the script copies the directory “jlca_orig” to the directory “jlca_corrected” which is the directory the patch is applied on.

header.bash This shell script cannot be started directly. Instead, it is a configuration script that is used by all other shell scripts. It contains variable definitions related to the directories the other scripts use, command line settings for the JLCA, and global procedures. The values of the variables in this script can be modified to make the other scripts use different directories.

diffExcludedFiles.txt This file contains a list of all files that the diff tool must ignore. For example, it makes no sense that the diff tool compares conversion reports.

Figure 7.1 on the facing page shows the development workflow that is associated with the tool suite. Steps one and two are handled by jlca.bash. Internally, jlca.bash uses applydiff.bash for carrying out step two. Step four is performed by makediff.bash.

7.3 Diff-Patch Tool Example

The following example is going to illustrate the work with the Diff-Patch tool.

Listing 7.1 shows a small Java program that the JLCA cannot fully convert. Instead of printing the message “Hello from Java.” the converted program should print the message “Hello from C#.”

```
1 public class MainClass {
2     public static void main(String[] args) {
3         System.out.println("Hello from Java.");
4     }
5 }
```

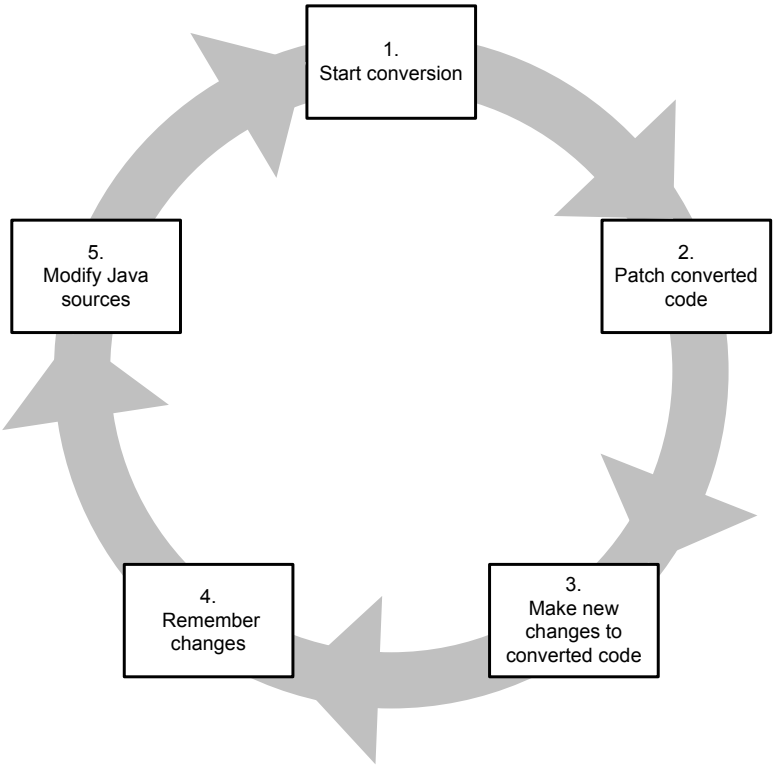


Figure 7.1: Development workflow with the Diff-Patch tool

Listing 7.1: java/MainClass.java

When jlca.bash is executed the JLCA is started and converts the Java program, producing the C# code shown in listing 7.2. As this is the first conversion the contents of the directories "jlca_orig" and "jlca_corrected" is identical.

```
1 using System;
2 public class MainClass
3 {
4     public static void  Main(System.String[] args)
5     {
6         System.Console.Out.WriteLine("Hello from Java.");
7     }
8 }
```

Listing 7.2: jlca_orig/MainClass.cs

At this point the file "jlca_corrected/MainClass.cs" should be modified and the script makediff.bash should be executed to store the modifications. Listing 7.3 shows the new C# code.

```
1 using System;
2 public class MainClass
3 {
4     public static void  Main(System.String[] args)
5     {
6         System.Console.Out.WriteLine("Hello from C#.");
7     }
8 }
```

Listing 7.3: jlca_orig/MainClass.cs

Now, whenever the Java sources are converted anew the modifications to the file "jlca_corrected/MainClass.cs" will be carried out automatically.

However, there is also a problem coming with the usage of the diff and patch tools. If the Java sources are changed in unfavorable places the patch tool is unable to find the lines it has to modify. This is due to the fact that the patch tool uses a combination of line numbers and line contents to find these lines. Even though it employs some heuristics the patch tool fails if too much of this information is corrupted or lost, e.g. because certain lines have been moved, deleted or modified.

Listings 7.4 and 7.5 give examples of how the patch tool works if the Java sources are modified in certain ways. The modifications in listing 7.4 corrupt too much of the information needed by the patch tool. The modifications in listing 7.5 on the other hand cause no problems.

```
1 public class MainClass {
2     public static void main(String[] args) {
3         System.out.println("Hello from Java.");
4         System.out.println("Hello again.");
5     }
6 }
```

Listing 7.4: MainClass.java with unfavorable modifications

```
1 public class MainClass {
2     public static void main(String[] args) {
3         System.out.println("Hello from Java.");
4     }
5
6     public void test() {
7     }
8 }
```

Listing 7.5: MainClass.java with neutral modifications

chapter 8 Extensibility Kit

This chapter gives an overview of the Extensibility Kit (EK) that is shipped with recent versions of the Java Language Conversion Assistant (JLCA)[1]. The Extensibility Kit is an extension to the JLCA which lets users add new conversion rules to the rule data base of the JLCA. This helps to solve the problem that the JLCA is unable to automatically convert the usage of classes for which it knows no such conversion rules. Additionally, the Extensibility Kit allows the redefinition of the built-in conversion rules of the JLCA.

8.1 Extensibility Kit Basics

New conversion rules have to be defined in a special pattern-based mapping language which allows the definition of rules for two distinct scenarios. On the one hand, rules can be defined that specify how the usage of a class member is mapped from Java to C#. This includes method invocations and read and write accesses to fields. On the other hand, the language provides means for specifying how overrides of class members have to be converted. Moreover, the mapping language allows the definition of custom error messages that can be included into the converted source code and into the conversion report. Further down in this chapter there are some examples which help to get a better understanding of the capabilities of the mapping language.

Fortunately, the EK ships with a convenient Visual Studio .NET 2003 integration. This integration features Visual Studio mapping language projects, syntax highlighting, keyword completion and the integration of the mapping language compiler.

Mapping language files are compiled into .NET assemblies. These assemblies, which seem to contain conversion rules in a special JLCA format, must be placed into the "Maps" subdirectory of the installation directory of the JLCA. This is done automatically during the compilation of a mapping language project. The "Maps" directory also contains the built-in conversion rules of the JLCA.

[1]Please note that at the writing of this document the release modalities for the EK were still unclear.

When the Extensibility Kit is used for converting Java programs the JLCA needs the compiled Java class files in addition to the source files. (Without the EK the JLCA does not seem to need any class files.) For finding these files the JLCA uses approximately the same rules as a typical Java virtual machine. It looks into the current directory and into all directories that are enumerated in the `CLASSPATH` environment variable. It also looks into all JAR files that are either specified directly in the `CLASSPATH` environment variable or that are located in one of the directories that are listed in this variable. *Please note*: The JLCA fails if it cannot find the required class files. Neither does it apply the custom conversion rules nor does it provide an error message with an indication of the problem.

8.2 Basic Mapping Definitions

The first example illustrates how custom conversion rules are defined with the mapping language of the Extensibility Kit. Listings 8.1, 8.2 and 8.3 on the facing page contain the Java source code that is to be converted as well as the mapping language code.

`MainClass` uses the custom Java class `de.hpi.Test` for which the JLCA does not know any conversion rules. `Test` features a method `test` that is overloaded several times, a setter and a getter method as well as a public field. The code in `MainClass` makes use of all these members.

```
1  import de.hpi.Test;
2
3  public class MainClass {
4      public static void main(String[] args) {
5          Test t = new Test();
6          t.test();
7          t.test("s");
8          t.test(123);
9          t.test("s",123);
10         String s = t.getProp();
11         t.setProp("s");
12         int i = t.i;
13         t.i = 1;
14     }
15 }
```

Listing 8.1: MainClass.java

```
1  package de.hpi;
2
3  public class Test {
4      public void test() {}
```

```
5       public void test(String s) {}
6       public void test(int i) {}
7       public void test(String s, int i) {}
8
9       public String getProp() {
10          return "";
11      }
12
13      public void setProp(String s) {}
14
15      public int i;
16  }
```

Listing 8.2: Test.java

The mapping file "Test.emap" contains the conversion rules for the class `de.hpi.Test`. These rules specify that the class `de.hpi.Test` is to be mapped onto the .NET class `CsTest`. There are mappings for all `test` methods as well as for the getter and the setter methods and the public field. Note how the getter and the setter methods and the public field are mapped onto C# properties. Further note how the order of the arguments for the method `test(String s, int i)` is exchanged in line 13.

```
1  package de.hpi
2
3      class Test : CsTest
4
5          Test
6              b() -> new CsTest();
7          endmap
8
9          test
10             a.b() -> a.Test();
11             a.b(p1:int) -> a.Test(p1);
12             a.b(p1:java.lang.String) -> a.Test(p1);
13             a.b(p1:java.lang.String, p2:int) -> a.Test(p2
                   ,p1);
14         endmap
15
16         getProp
17             a.b() -> a.Prop;
18         endmap
19
20         setProp
21             a.b(p:java.lang.String) -> a.Prop = p;
22         endmap
23
```

```
24          i
25              a.b -> a.I;
26              a.b = c -> a.I = c;
27          endmap
28
29      endclass
30
31  endpackage
```

Listing 8.3: Test.emap

Listing 8.4 contains the C# code that the JLCA produces with the custom conversion rules. Apparently, the Java class `de.hpi.Test` is successfully mapped onto the .NET class `CsTest`.

```
1  using System;
2  using Test = de.hpi.Test;
3
4  public class MainClass
5  {
6      [STAThread]
7      public static void  Main(System.String[] args)
8      {
9          CsTest t = new CsTest();
10         t.Test();
11         t.Test("s");
12         t.Test(123);
13         t.Test(123, "s");
14         System.String s = t.Prop;
15         t.Prop = "s";
16         int i = t.I;
17         t.I = 1;
18     }
19 }
```

Listing 8.4: MainClass.cs

8.3 Overriding Built-In Conversion Rules

This example shows that the EK can be used to replace selected default conversion rules.

The listings 8.5 on the facing page and 8.6 on the next page show the Java source code and the mapping language code respectively. The custom conversion rules specify that the class `java.util.Date` is to be mapped onto the C# class `MyCsDate`. The parameterless constructor `Date()` is mapped onto `MyCsDate()` and the

method `toString()` is mapped onto `MyToString()`. All other default conversion rules remain intact.

```
import java.util.Date;

public class MainClass {
    public static void main(String[] args) {
        Date d = new Date();
        System.out.println(d.toString());
        System.out.println(d.getTime());
    }
}
```

Listing 8.5: MainClass.java

```
package java.util

    class Date : MyCsDate

        Date
            b() -> new MyCsDate();
        endmap

        toString
            a.b() -> a.MyToString();
        endmap

    endclass

endpackage
```

Listing 8.6: Date.emap

Listing 8.7 shows the C# code that the JLCA produces with the custom conversion rules. Obviously, the constructor and the method `toString()` are converted according to the specifications of the mapping file. The method `getTime()` on the other hand is converted with the default rules of the JLCA. It is mapped onto the `Ticks` property of the .NET class `System.DateTime`.

```
using System;

public class MainClass
{
    [STAThread]
    public static void  Main(System.String[] args)
    {
        MyCsDate d = new MyCsDate();
        System.Console.Out.WriteLine(d.MyToString());
```

```
10          //UPGRADE_TODO: Method 'java.util.Date.getTime' was converted to '
                System.DateTime.Ticks' which has a different behavior.
11          System.Console.Out.WriteLine(d.Ticks);
12      }
13  }
```

Listing 8.7: MainClass.cs

8.4 Declaration Mappings

This example shows the usage of so-called declaration mappings. This kind of mapping is used for the conversion of inheritance relations where overwritten Java methods and fields must be mapped onto adequate C# methods, fields and properties.

Listings 8.8, 8.9 and 8.10 on the next page show the Java source code for this example. The class `Test` uses the classes `de.hpi.BaseClass` and `DerivedClass`. The class `DerivedClass` inherits from the class `de.hpi.BaseClass`. `DerivedClass` overwrites all methods and hides the original field `s`.

```
1  public class Test {
2      public static void main(String[] args) {
3          de.hpi.BaseClass bc = new de.hpi.BaseClass();
4          bc.test();
5          bc.test2();
6          bc.test3();
7          System.out.println(bc.s);
8
9          DerivedClass dc = new DerivedClass();
10         dc.test();
11         dc.test2();
12         dc.test3();
13         System.out.println(dc.s);
14     }
15 }
```

Listing 8.8: Test.java

```
1  public class DerivedClass extends de.hpi.BaseClass {
2      public void test() {
3          System.out.println("Hello from DerivedClass.");
4      }
5
6      public void test2() {
7          System.out.println("Hello again from DerivedClass
               .");
```

```
8       }
9
10      public void test3() {
11          System.out.println("Last hello from DerivedClass.
                ");
12      }
13
14      public String s = "DerivedClassString";
15  }
```

Listing 8.9: DerivedClass.java

```
1   package de.hpi;
2
3   public class BaseClass {
4
5       public void test() {
6           System.out.println("Hello from BaseClass.");
7       }
8
9       public void test2() {
10          System.out.println("Hello again from BaseClass.")
                ;
11      }
12
13      public void test3() {
14          System.out.println("Last hello from BaseClass.");
15      }
16
17      public String s = "BaseClassString";
18  }
```

Listing 8.10: BaseClass.java

Listing 8.11 contains the custom conversion rules for the class `de.hpi.BaseClass`. The method `test` has only an invocation mapping. `test2` has an invocation mapping and explicitly no declaration mapping. `test3` finally has both mappings. Moreover, there is a mapping for reading accesses to the `s` field.

```
1   package de.hpi
2
3       class BaseClass : CisBaseClass
4
5           BaseClass
6               b() -> new CisBaseClass();
7           endmap
8
```

```
9           test
10              a.b() -> a.Test();
11          endmap
12
13          test2
14              a.b() -> a.Test2();
15              decl() -> notmap;
16          endmap
17
18          test3
19              a.b() -> a.Test3();
20              decl() -> funcdecl public void Test3();
21          endmap
22
23          s
24              a.b -> a.S;
25          endmap
26
27      endclass
28
29  endpackage
```

Listing 8.11: BaseClass.emap

Listings 8.12 and 8.13 on the facing page show the results of the conversion with the JLCA. In "Test.cs" all member usages are converted as could be expected. Note the difference between the lines 11 and 17. `CisBaseClass` has a field `S` (upper case letter) and `DerivedClass` has a field `s` (lower case letter). This is due to the fact that no declaration mapping is specified for this field.

"DerivedClass.cs" shows the effects of the declaration mappings. Only the overwritten method `test3` is correctly mapped to the corresponding method `Test3`. For `test` there is no declaration mapping. For `test2` there is an upgrade note that corresponds to the `notmap` of the custom conversion rules.

```
1  using System;
2  public class Test
3  {
4      [STAThread]
5      public static void  Main(System.String[] args)
6      {
7          CisBaseClass bc = new CisBaseClass();
8          bc.Test();
9          bc.Test2();
10         bc.Test3();
11         System.Console.Out.WriteLine(bc.S);
12
13         DerivedClass dc = new DerivedClass();
```

```
14          dc.Test();
15          dc.Test2();
16          dc.Test3();
17          System.Console.Out.WriteLine(dc.s);
18      }
19  }
```

Listing 8.12: Test.cs

```
1  using System;
2  public class DerivedClass:CisBaseClass
3  {
4      public override void  test()
5      {
6          System.Console.Out.WriteLine("Hello from
               DerivedClass.");
7      }
8
9      //UPGRADE_NOTE: The equivalent of method 'DerivedClass.test2' is not an
           override method.
10     public void  test2()
11     {
12         System.Console.Out.WriteLine("Hello again from
               DerivedClass.");
13     }
14
15     public override void  Test3()
16     {
17         System.Console.Out.WriteLine("Last hello from
               DerivedClass.");
18     }
19
20     new public System.String s = "DerivedClassString";
21 }
```

Listing 8.13: DerivedClass.cs

8.5 Custom Error Messages

This example illustrates how custom error messages can be defined with the mapping language. This is very useful if no custom conversion rules can be specified. In these cases, error messages can point programmers to the places in the converted source code where manual work is required after the conversion. In the mapping language the abbreviation EWI is used for such messages. EWI stands for “errors, warnings and issues”.

Listings 8.14 and 8.15 show the Java source code for this example. The class `Test` both uses and extends the class `de.hpi.EWI`.

```
1  import de.hpi.EWI;
2
3  public class Test extends EWI {
4
5      public void methodWithoutMapping() {
6      }
7
8      public void methodWithoutInheritanceMapping() {
9      }
10
11     public static void main(String[] args) {
12         EWI e = new EWI();
13         e.methodWithoutMapping();
14         e.methodWithoutInheritanceMapping();
15
16         Test t = new Test();
17         t.methodWithoutMapping();
18         t.methodWithoutInheritanceMapping();
19     }
20 }
```

Listing 8.14: Test.java

```
1  package de.hpi;
2
3  public class EWI {
4      public void methodWithoutMapping() {
5      }
6
7      public void methodWithoutInheritanceMapping() {
8      }
9  }
```

Listing 8.15: EWI.java

Listing 8.16 shows the custom conversion rules for this example. The Java class `EWI` is mapped onto the .NET class `CisEWI`. The method `methodWithoutMapping` explicitly has no invocation mapping. The method `methodWithoutInheritanceMapping` has an invocation but no declaration mapping.

Furthermore, the mapping file includes a definition file. This file, which is shown in listing 8.17 on the facing page, contains the custom EWI.

```
1  using "Declarations.def";
```

```
2
3  package de.hpi
4
5      class EWI : CisEWI
6
7          EWI
8              b() -> new CisEWI();
9          endmap
10
11         methodWithoutMapping
12             a.b() -> notmap;
13         endmap
14
15         methodWithoutInheritanceMapping
16             a.b() -> a.MethodWithoutInheritanceMapping();
17             decl() -> notmap(noDeclarationMapping);
18         endmap
19
20     endclass
21
22 endpackage
```

Listing 8.16: EWI.emap

```
1 ewi noDeclarationMapping : "There is no declaration
     mapping for this method", "http://myUrl";
```

Listing 8.17: Declarations.def

Listing 8.18 shows the resulting C# code. The custom EWI appears in line 11 to warn users that the declaration of the method `methodWithoutInheritanceMapping` could not be converted. In lines 20 and 25 there are default messages that correspond to the `notmap` in "EWI.emap" in line 12.

```
1  using System;
2  using EWI = de.hpi.EWI;
3
4  public class Test:CisEWI
5  {
6
7      public override void  methodWithoutMapping()
8      {
9      }
10
11     //There is no declaration mapping for this method 'http://myUrl'
12     public void  methodWithoutInheritanceMapping()
```

```
13      {
14      }
15
16      [STAThread]
17      public static void  Main(System.String[] args)
18      {
19          CisEWI e = new CisEWI();
20          //UPGRADE_ISSUE: Method 'de.hpi.EWI.methodWithoutMapping' was not
                converted.
21          e.methodWithoutMapping();
22          e.MethodWithoutInheritanceMapping();
23
24          Test t = new Test();
25          //UPGRADE_ISSUE: Method 'Test.methodWithoutMapping' was not
                converted.
26          t.methodWithoutMapping();
27          t.MethodWithoutInheritanceMapping();
28      }
29  }
```

Listing 8.18: Test.cs

8.6 The Extensibility Kit and The Diff-Patch Tool

The last example shows how the conversion of the usage of a small custom library can be handled with the Extensibility Kit. As the JLCA and the EK are not able to correctly convert the example the Diff-Patch tool is used to remember the changes that have to be applied to the C# code after the conversion.

8.6.1 The Java Library

The following listings contain the source code of the Java library that is used in this example. This library provides some functions for changing and querying properties of Win32 console windows, such as the text attributes and the console title. This library is based on a DLL that conforms to the JNI[2] specification and that provides the low-level functions for the communication with the Win32 platform.

Obviously, the usage of this library cannot be automatically converted. First of all, it is a custom library for which the JLCA has no built-in conversion rules. Secondly, it uses the JNI which is not available on the .NET platform.

```
1  package de.hpi.coloredConsole;
```

[2]Java Native Interface.

```
2
3  public class Console {
4
5      private static short _consoleHandle;
6      private static short _textAttributes = (short)(Colors
           .FOREGROUND_BLUE | Colors.FOREGROUND_GREEN |
           Colors.FOREGROUND_RED);
7
8      static {
9          System.loadLibrary("console_jni");
10         _consoleHandle = _getStdHandle(Handles.
               STD_OUTPUT_HANDLE);
11         try {
12             setTextAttributes(_textAttributes);
13         } catch (ConsoleException e) {
14             System.err.println("The console does not
                   support extended functions.");
15         }
16     }
17
18     private static native short _getStdHandle(short
           handleType);
19
20     private static native boolean _setTextAttributes(
           short handle, short attributes);
21
22     private static native boolean _setConsoleTitle(String
           title);
23     private static native String _getConsoleTitle();
24
25     public static int getStdHandle(short handleType) {
26         return _getStdHandle(handleType);
27     }
28
29     public static void setTextAttributes(short attributes
           ) throws ConsoleException {
30         boolean success = _setTextAttributes(
               _consoleHandle, attributes);
31
32         if (!success)
33             throw new ConsoleException("Could not set
                   text attributes.");
34
35         _textAttributes = attributes;
36     }
37
38     public static short getTextAttributes() {
```

```
39          return _textAttributes;
40      }
41
42      public static void setConsoleTitle(String title)
           throws ConsoleException {
43          boolean success = _setConsoleTitle(title);
44          if (!success)
45              throw new ConsoleException("Could not set
                   console title.");
46      }
47
48      public static String getConsoleTitle() {
49          return _getConsoleTitle();
50      }
51  }
```

Listing 8.19: Console.java

```
1  package de.hpi.coloredConsole;
2
3  public interface Colors {
4      public static short FOREGROUND_BLUE      = 0x01;
5      public static short FOREGROUND_GREEN     = 0x02;
6      public static short FOREGROUND_RED       = 0x04;
7      public static short FOREGROUND_INTENSITY = 0x08;
8      public static short BACKGROUND_BLUE      = 0x10;
9      public static short BACKGROUND_GREEN     = 0x20;
10     public static short BACKGROUND_RED       = 0x40;
11     public static short BACKGROUND_INTENSITY = 0x80;
12 }
```

Listing 8.20: Colors.java

```
1  package de.hpi.coloredConsole;
2
3  public interface Handles {
4
5      //Copied from WinBase.h
6      public static short STD_INPUT_HANDLE  = -10;
7      public static short STD_OUTPUT_HANDLE = -11;
8      public static short STD_ERROR_HANDLE  = -12;
9
10 }
```

Listing 8.21: Handles.java

8.6.2 The Java Program

Listing 8.22 contains the Java program that is to be converted. This program uses the methods provided by the console library in order to query and set the console title and to print some colored text.

```
1  import de.hpi.coloredConsole.Console;
2  import de.hpi.coloredConsole.Handles;
3  import de.hpi.coloredConsole.Colors;
4  import de.hpi.coloredConsole.ConsoleException;
5
6  public class ConsoleTest {
7
8      public static void main(String[] args) throws
           Exception {
9          System.out.println("Current console title: " +
               Console.getConsoleTitle() + ".");
10         Console.setConsoleTitle("Console Test");
11         System.out.println("New console title: " +
               Console.getConsoleTitle() + ".");
12
13         Console.setTextAttributes(Colors.FOREGROUND_RED);
14         System.out.println("red");
15         Console.setTextAttributes(Colors.FOREGROUND_GREEN
               );
16         System.out.println("green");
17         Console.setTextAttributes(Colors.FOREGROUND_BLUE)
               ;
18         System.out.println("blue");
19
20         Console.setTextAttributes((short)(Colors.
               FOREGROUND_RED | Colors.FOREGROUND_INTENSITY))
               ;
21         System.out.println("light red");
22         Console.setTextAttributes((short)(Colors.
               FOREGROUND_GREEN | Colors.FOREGROUND_INTENSITY
               ));
23         System.out.println("light green");
24         Console.setTextAttributes((short)(Colors.
               FOREGROUND_BLUE | Colors.FOREGROUND_INTENSITY)
               );
25         System.out.println("light blue");
26
27         for (short i = 0; i < 256; i++) {
28             Console.setTextAttributes(i);
29             System.out.print("#");
30         }
```

```
31
32          Console.setTextAttributes((short)(Colors.
               FOREGROUND_RED | Colors.FOREGROUND_GREEN |
               Colors.FOREGROUND_BLUE));
33          Thread.sleep(3000);
34      }
35  }
```

Listing 8.22: ConsoleTest.java

8.6.3 The Custom Conversion Rules

The following listings contain the custom conversion rules for the console library. Please note that some methods are mapped onto C# properties. Further note that the Java interfaces `de.hpi.coloredConsole.Colors` and `de.hpi.coloredConsole.Handles` are converted to C# enums.

```
1  package de.hpi.coloredConsole
2
3      class Console : DE.HPI.ColoredConsole.Console
4
5          getStdHandle
6              a.b(handleType : short) -> a.GetStdHandle(
                   handleType);
7          endmap
8
9          setConsoleTitle
10             a.b(title : java.lang.String) -> a.
                   ConsoleTitle = title;
11         endmap
12
13         getConsoleTitle
14             a.b() -> a.ConsoleTitle;
15         endmap
16
17         setTextAttributes
18             a.b(attributes : short) -> a.TextAttributes
                   = attributes;
19         endmap
20
21         getTextAttributes
22             a.b() -> a.TextAttributes;
23         endmap
24
25     endclass
```

```
26
27 endpackage
```

Listing 8.23: Console.emap

```
1  package de.hpi.coloredConsole
2
3      class Colors : DE.HPI.ColoredConsole.Color
4
5          FOREGROUND_RED
6              a.b      -> a.FGRed;
7              a.b = c -> notmap;
8          endmap
9
10         FOREGROUND_GREEN
11             a.b      -> a.FGGreen;
12             a.b = c -> notmap;
13         endmap
14
15         FOREGROUND_BLUE
16             a.b      -> a.FGBlue;
17             a.b = c -> notmap;
18         endmap
19
20         FOREGROUND_INTENSITY
21             a.b      -> a.FGIntensity;
22             a.b = c -> notmap;
23         endmap
24
25         BACKGROUND_RED
26             a.b      -> a.BGRed;
27             a.b = c -> notmap;
28         endmap
29
30         BACKGROUND_GREEN
31             a.b      -> a.BGGreen;
32             a.b = c -> notmap;
33         endmap
34
35         BACKGROUND_BLUE
36             a.b      -> a.BGBlue;
37             a.b = c -> notmap;
38         endmap
39
40         BACKGROUND_INTENSITY
41             a.b      -> a.BGIntensity;
42             a.b = c -> notmap;
```

```
43          endmap
44
45      endclass
46
47  endpackage
```

Listing 8.24: Colors.emap

```
1  package de.hpi.coloredConsole
2
3      class Handles : DE.HPI.ColoredConsole.Handles
4
5          STD_INPUT_HANDLE
6              a.b -> a.StandardInput;
7              a.b = c -> notmap;
8          endmap
9
10         STD_OUTPUT_HANDLE
11             a.b -> a.StandardOutput;
12             a.b = c -> notmap;
13         endmap
14
15         STD_ERROR_HANDLE
16             a.b -> a.StandardError;
17             a.b = c -> notmap;
18         endmap
19
20     endclass
21
22 endpackage
```

Listing 8.25: Handles.emap

8.6.4 The C# Library

The listings in this section show the C# library the converted code must use. Wherever possible, this library replaces Java getters and setters with C# properties and static Java interface members with C# enums.

Unlike the Java library, this library uses the .NET platform invoke mechanisms.

```
1  using System;
2  using System.Runtime.InteropServices;
3  using System.Text;
4
5  namespace DE.HPI.ColoredConsole
```

```
6  {
7      public class Console
8      {
9          private static Color _textAttributes = Color.
               FGRed | Color.FGGreen | Color.FGBlue;
10         private static IntPtr _handle;
11
12         static Console()
13         {
14             _handle = GetStdHandle(Handles.StandardOutput
                   );
15             TextAttributes = _textAttributes;
16         }
17
18         [DllImport("kernel32.dll", EntryPoint = "
               SetConsoleTitle", SetLastError = true)]
19         private static extern bool _setConsoleTitle(
               String title);
20
21         [DllImport("kernel32.dll", EntryPoint = "
               GetConsoleTitle", SetLastError = true)]
22         private static extern uint _getConsoleTitle(
               StringBuilder buffer, uint size);
23
24         public static String ConsoleTitle
25         {
26             get
27             {
28                 StringBuilder buffer = new StringBuilder
                       (256);
29                 uint size = _getConsoleTitle(buffer, 256)
                       ;
30                 if (size > 0)
31                 {
32                     return buffer.ToString(0, (int)size);
33                 }
34                 else
35                     throw new ConsoleException("Could not
                           get console title. Win32 error: "
                           + Marshal.GetLastWin32Error() + "
                          .");
36             }
37             set
38             {
39                 if (!_setConsoleTitle(value))
40                     throw new ConsoleException("Could not
                           set console title. Win32 error: "
```

```
                            + Marshal.GetLastWin32Error() + "
                           .");
41              }
42          }
43
44          [DllImport("kernel32.dll", EntryPoint = "
               GetStdHandle", SetLastError = true)]
45          private static extern IntPtr _getStdHandle(
               Handles handleType);
46
47          public static IntPtr GetStdHandle(Handles
               handleType)
48          {
49              return _getStdHandle(handleType);
50          }
51
52          [DllImport("kernel32.dll", EntryPoint = "
               SetConsoleTextAttribute", SetLastError = true)
               ]
53          private static extern bool
               _setConsoleTextAttribute(IntPtr handle, Color
               color);
54
55          public static Color TextAttributes
56          {
57              get
58              {
59                  return _textAttributes;
60              }
61              set
62              {
63                  if (!_setConsoleTextAttribute(_handle,
                       value))
64                      throw new ConsoleException("Could not
                           set text attribute. Win32 error:
                          " + Marshal.GetLastWin32Error() +
                          ".");
65                  _textAttributes = value;
66              }
67          }
68      }
69  }
```

Listing 8.26: Console.cs

```
1  using System;
2
```

```
3  namespace DE.HPI.ColoredConsole
4  {
5      [Flags]
6      public enum Color : ushort
7      {
8          FGBlue = 0x01,
9          FGGreen = 0x02,
10         FGRed = 0x04,
11         FGIntensity = 0x08,
12         BGBlue = 0x10,
13         BGGreen = 0x20,
14         BGRed = 0x40,
15         BGIntensity = 0x80
16     }
17 }
```

Listing 8.27: Color.cs

```
1  using System;
2
3  namespace DE.HPI.ColoredConsole
4  {
5      public enum Handles : int
6      {
7          StandardInput = -10,
8          StandardOutput = -11,
9          StandardError = -12,
10     }
11 }
```

Listing 8.28: Handles.cs

8.6.5 The Converted Program

Listing 8.29 on the next page shows the C# code the JLCA produces with the help of the custom conversion rules. Even though rules have been specified for all elements of the Java library this code still contains some errors.

First of all, the JLCA has converted the `import` statements in lines one to four in "ConsoleTest.java" to useless `using` statements. The reason for this kind of behavior is still unclear.

Moreover, the Java programming language does not know constructs like C# enums. In the Java library, all constants have the short data type. This is why in "ConsoleTest.java" type casts have to be performed in lines 20, 22 and 24. The counter

variable of the for loop does not have to be type-casted as it already has the right data type.

As stated before, the C# library uses enums as a replacement for Java's static interface constants. This has some implications on the way type casts have to be performed in the converted program. Unfortunately, the JLCA seems to be unaware of this issue and produces uncompilable source code. The type casts that are performed in lines 24, 26 and 28 in "jlca_orig/ConsoleTest.cs" are wrong and thus lead to compilation errors. Additionally, the converted C# code lacks a required type cast in the for loop. The counter variable, which has the short data type, cannot be assigned to the `TextAttributes` property as the latter has an enumerated type.

```
1  using System;
2  using Console = de.hpi.coloredConsole.Console;
3  using Handles = de.hpi.coloredConsole.Handles;
4  using Colors = de.hpi.coloredConsole.Colors;
5  using ConsoleException = de.hpi.coloredConsole.
       ConsoleException;
6
7  public class ConsoleTest
8  {
9
10     [STAThread]
11     public static void  Main(System.String[] args)
12     {
13         System.Console.Out.WriteLine("Current console
               title: " + DE.HPI.ColoredConsole.Console.
               ConsoleTitle + ".");
14         DE.HPI.ColoredConsole.Console.ConsoleTitle = "
               Console Test";
15         System.Console.Out.WriteLine("New console title:
               " + DE.HPI.ColoredConsole.Console.ConsoleTitle
                + ".");
16
17         DE.HPI.ColoredConsole.Console.TextAttributes = DE
               .HPI.ColoredConsole.Color.FGRed;
18         System.Console.Out.WriteLine("red");
19         DE.HPI.ColoredConsole.Console.TextAttributes = DE
               .HPI.ColoredConsole.Color.FGGreen;
20         System.Console.Out.WriteLine("green");
21         DE.HPI.ColoredConsole.Console.TextAttributes = DE
               .HPI.ColoredConsole.Color.FGBlue;
22         System.Console.Out.WriteLine("blue");
23
24         DE.HPI.ColoredConsole.Console.TextAttributes = (
               short) (DE.HPI.ColoredConsole.Color.FGRed | DE
               .HPI.ColoredConsole.Color.FGIntensity);
```

```
25          System.Console.Out.WriteLine("light red");
26          DE.HPI.ColoredConsole.Console.TextAttributes = (
               short) (DE.HPI.ColoredConsole.Color.FGGreen |
               DE.HPI.ColoredConsole.Color.FGIntensity);
27          System.Console.Out.WriteLine("light green");
28          DE.HPI.ColoredConsole.Console.TextAttributes = (
               short) (DE.HPI.ColoredConsole.Color.FGBlue |
               DE.HPI.ColoredConsole.Color.FGIntensity);
29          System.Console.Out.WriteLine("light blue");
30
31          for (short i = 0; i < 256; i++)
32          {
33              DE.HPI.ColoredConsole.Console.TextAttributes
                    = i;
34              System.Console.Out.Write("#");
35          }
36
37          DE.HPI.ColoredConsole.Console.TextAttributes = (
               short) (DE.HPI.ColoredConsole.Color.FGRed | DE
               .HPI.ColoredConsole.Color.FGGreen | DE.HPI.
               ColoredConsole.Color.FGBlue);
38          //UPGRADE_TODO: Method 'java.lang.Thread.sleep' was converted to '
               System.Threading.Thread.Sleep' which has a different behavior.
39          System.Threading.Thread.Sleep(new System.TimeSpan
               ((System.Int64) 10000 * 3000));
40      }
41  }
```

Listing 8.29: jlca_orig/ConsoleTest.cs

Listing 8.30 shows the corrected version of the C# program. The useless `using` statement have been commented out and type casts have been removed and inserted as required.

For remembering all these changes the Diff-Patch tool can be employed. In this way, all changes are automatically applied to the converted source code should the original Java program ever be converted again. This even works if the Java program is slightly changed, e.g. because it is further developed.

```
1  using System;
2  //using Console = de.hpi.coloredConsole.Console;
3  //using Handles = de.hpi.coloredConsole.Handles;
4  //using Colors = de.hpi.coloredConsole.Colors;
5  //using ConsoleException = de.hpi.coloredConsole.ConsoleException;
6
7  public class ConsoleTest
8  {
9
```

```
10      [STAThread]
11      public static void  Main(System.String[] args)
12      {
13          System.Console.Out.WriteLine("Current console
               title: " + DE.HPI.ColoredConsole.Console.
               ConsoleTitle + ".");
14          DE.HPI.ColoredConsole.Console.ConsoleTitle = "
               Console Test";
15          System.Console.Out.WriteLine("New console title:
               " + DE.HPI.ColoredConsole.Console.ConsoleTitle
                + ".");
16
17          DE.HPI.ColoredConsole.Console.TextAttributes = DE
               .HPI.ColoredConsole.Color.FGRed;
18          System.Console.Out.WriteLine("red");
19          DE.HPI.ColoredConsole.Console.TextAttributes = DE
               .HPI.ColoredConsole.Color.FGGreen;
20          System.Console.Out.WriteLine("green");
21          DE.HPI.ColoredConsole.Console.TextAttributes = DE
               .HPI.ColoredConsole.Color.FGBlue;
22          System.Console.Out.WriteLine("blue");
23
24          DE.HPI.ColoredConsole.Console.TextAttributes = (
               DE.HPI.ColoredConsole.Color.FGRed | DE.HPI.
               ColoredConsole.Color.FGIntensity);
25          System.Console.Out.WriteLine("light red");
26          DE.HPI.ColoredConsole.Console.TextAttributes = (
               DE.HPI.ColoredConsole.Color.FGGreen | DE.HPI.
               ColoredConsole.Color.FGIntensity);
27          System.Console.Out.WriteLine("light green");
28          DE.HPI.ColoredConsole.Console.TextAttributes = (
               DE.HPI.ColoredConsole.Color.FGBlue | DE.HPI.
               ColoredConsole.Color.FGIntensity);
29          System.Console.Out.WriteLine("light blue");
30
31          for (short i = 0; i < 256; i++)
32          {
33              DE.HPI.ColoredConsole.Console.TextAttributes
                    = (DE.HPI.ColoredConsole.Color)i;
34              System.Console.Out.Write("#");
35          }
36
37          DE.HPI.ColoredConsole.Console.TextAttributes = (
               DE.HPI.ColoredConsole.Color.FGRed | DE.HPI.
               ColoredConsole.Color.FGGreen | DE.HPI.
               ColoredConsole.Color.FGBlue);
38          //UPGRADE_TODO: Method 'java.lang.Thread.sleep' was converted to '
```

```
                System.Threading.Thread.Sleep' which has a different behavior.
39              System.Threading.Thread.Sleep(new System.TimeSpan
                    ((System.Int64) 10000 * 3000));
40          }
41      }
```

Listing 8.30: jlca_corrected/ConsoleTest.cs

Conversion Tests with JLCA

chapter 9

9.1 Sample Report

This section shows a sample test report as it is provided for each test the JLCA has to manage.

Summary

At the beginning of each test a summary shows a brief overview of the tested aspect and the result. It may look like the following one.

Test: Sample report summary

```
public class Sample {
    public String test() {
        return "This is a sample";
    }
}
```

Results:

supported	issues	difficulty
no	4	5

This summary shows which characteristic or Java class is tested. In this case it is the *language element "assert"*. Additionally a short example is given. This mostly is just a code fragment while the code listed below will be more complete.

The second part of the summary concerns the results of the test. The color of the results indicates the success: green means that the tested item was processed without any problems. If the table is yellow, some modification to the code had to be made. Finally, if it is red, no suitable conversion was possible. The table consists of three columns:

supported indicates if the tested item is supported by the JLCA at all,

issues shows the number of problems that occurred during conversion. This number may be smaller than the number of issues that really occurred for only issues

that depend on the tested characteristic are counted here;

difficulty shall give an idea of how much work has to be done to make the conversion work. The given number may be from 1 to 5, where 1 means *easy* and 5 means *very difficult*.

Introduction

After the summary a preface to the test is given. Aspects and specialities concerning the test are mentioned as well as expected difficulties.

Java Source Code

The first code shown in each test is the Java source code that has been converted.

```
1 public class Sample {
2     public String test() {
3         return "This is a sample";
4     }
5 }
```

Conversion Results

This paragraph contains the results generated by the JLCA-Tool. A sample result is shown below

Filename.java

Conversion Issues for
ClassName.methodSignature(...)

#	Type	Severity	Description
1	ToDo	2	Description for the issue.

Conversion Issues for
ClassName.methodSignature(...)

#	Type	Severity	Description
1	Compile	1	Description for the issue.
2	Compile	1	Description for the issue.

Converted C# Code

The conversion results are followed by the generated C# code.

```
1  //This would be a converted file
2  public class Sample
3  {
4      public string test()
5      {
6          return "This is a sample";
7      }
8
9      lines that were commented out from the original converted file are shown
           this way
10     lines that are added to the original converted file
           are displayed this way
11 }
```

When a line has to be removed (i.e. commented out) from the file the JLCA generated, they are composed as shown in line 9. On the other hand, lines that are added to the code are shown as in line 10.

Analysis

The last but most important paragraph is the analysis of the problem, the conversion results and the consequences that follow from these results. Workarounds to upcoming problems are given or even suggestions for improvements of the conversion tool. The JLCA extensibility kit will be kept in mind for this analysis.

Visualisation

If the test contains a visual representation (i.e. for GUI tests), screenshots are shown here.

9.2 Hello World

Summary

Test: Hello World program

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World.");
    }
}
```

Results:

supported	issues	difficulty
yes	0	0

Introduction

The first converted Java program is the classical Hello-World example. As one could expect, the Java Language Conversion Assistant encounters no errors during the conversion of this program. The converted C# program yields the same results as the original.

Java Source Code

Listing 9.2 shows the "minimal" Hello-World Java program.

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World.");
4     }
5 }
```

Conversion Results

HelloWorld.java

There are no issues for this file.

Converted C# Code

Listing 9.2 shows the converted Hello-World program.

```
1 using System;
2 public class HelloWorld
3 {
4     [STAThread]
5     public static void  Main(System.String[] args)
6     {
7         System.Console.Out.WriteLine("Hello World.");
8     }
9 }
```

Except for the `STAThread` attribute, this program is identical to the original Java program.

Analysis

Even though the Java Language Conversion Assistant does not have any real problems when converting the Hello-World class it still reports one global warning. This warning is included in all conversion reports and does normally not indicate any serious problems.

9.3 Anonymous Classes

Summary

Test: Anonymous Classes

```
public class MainClass {

    private int member = 0;

    public static void main(String[] args) {
        new MainClass().test();
    }

    private void test() {
        ITest t1 = new ITest() {
            private int member = 1;
            public void test() {
                System.out.println("Member of
                    anonymous class: " + member);
                System.out.println("Member of
                    MainClass: " + MainClass.this.
                    member);
            }
        };
        t1.test();

        ITest t2 = new ITest() {
            private int member = 3;
            public void test() {
                System.out.println("Member of
                    anonymous class: " + member);
                System.out.println("Member of
                    MainClass: " + MainClass.this.
                    member);
            }
        };
        t2.test();
    }
}
```

Results:

supported	issues	difficulty
yes	0	0

Introduction

This example shows how anonymous classes are translated. As anonymous classes do not exist in C# they are converted to nested classes.

Java Source Code

```
1  public class MainClass {
2
3      private int member = 0;
4
5      public static void main(String[] args) {
6          new MainClass().test();
7      }
8
9      private void test() {
10         ITest t1 = new ITest() {
11             private int member = 1;
12             public void test() {
13                 System.out.println("Member of anonymous
                       class: " + member);
14                 System.out.println("Member of MainClass:
                       " + MainClass.this.member);
15             }
16         };
17         t1.test();
18
19         ITest t2 = new ITest() {
20             private int member = 3;
21             public void test() {
22                 System.out.println("Member of anonymous
                       class: " + member);
23                 System.out.println("Member of MainClass:
                       " + MainClass.this.member);
24             }
25         };
26         t2.test();
27     }
28 }
```

In the main method, the program creates an object of type `MainClass` and executes its `test` method. In this method, two objects are created which both are of an anonymous class each. Then the method `test` is called on each object to put out some text.

Conversion Results

ITest.java

There are no issues for this file.

MainClass.java

There are no issues for this file.

Converted C# Code

```
1  using System;
2  public class MainClass
3  {
4      //UPGRADE_NOTE: Field 'EnclosingInstance' was added to class '
           AnonymousClassITest' to access its enclosing instance.
5      private class AnonymousClassITest : ITest
6      {
7          public AnonymousClassITest(MainClass
               enclosingInstance)
8          {
9              InitBlock(enclosingInstance);
10         }
11         private void  InitBlock(MainClass
               enclosingInstance)
12         {
13             this.enclosingInstance = enclosingInstance;
14         }
15         private MainClass enclosingInstance;
16         public MainClass Enclosing_Instance
17         {
18             get
19             {
20                 return enclosingInstance;
21             }
22
23         }
24         private int member = 1;
25         public virtual void  test()
26         {
27             System.Console.Out.WriteLine("Member of
                   anonymous class: " + member);
28             System.Console.Out.WriteLine("Member of
                   MainClass: " + Enclosing_Instance.member);
29         }
```

```
30      }
31      //UPGRADE_NOTE: Field 'EnclosingInstance' was added to class '
           AnonymousClassITest1' to access its enclosing instance.
32      private class AnonymousClassITest1 : ITest
33      {
34          public AnonymousClassITest1(MainClass
               enclosingInstance)
35          {
36              InitBlock(enclosingInstance);
37          }
38          private void  InitBlock(MainClass
               enclosingInstance)
39          {
40              this.enclosingInstance = enclosingInstance;
41          }
42          private MainClass enclosingInstance;
43          public MainClass Enclosing_Instance
44          {
45              get
46              {
47                  return enclosingInstance;
48              }
49
50          }
51          private int member = 3;
52          public virtual void  test()
53          {
54              System.Console.Out.WriteLine("Member of
                   anonymous class: " + member);
55              System.Console.Out.WriteLine("Member of
                   MainClass: " + Enclosing_Instance.member);
56          }
57      }
58
59      private int member = 0;
60
61      [STAThread]
62      public static void  Main(System.String[] args)
63      {
64          new MainClass().test();
65      }
66
67      private void  test()
68      {
69          ITest t1 = new AnonymousClassITest(this);
70          t1.test();
71
```

```
72          ITest t2 = new AnonymousClassITest1(this);
73          t2.test();
74      }
75  }
```

Because there are no anonymous classes in C# the code mainly consists of constructs that emulate the Java behaviour of anonymous classes.

Analysis

The JLCA creates two inner classes `AnonymousClassITest` and `AnonymousClassITest1`. Both classes are constructed similar to the inner class example. This is necessary because anonymous classes are always non-static and thus have a reference to their enclosing instance.

9.4 Inner Classes

Summary

Test: Inner Classes

```
class MainClass {

    private int member = 0;

    void test() {
        new InnerClass().test();
        new InnerClass2().test();
    }

    public static void main(String[] args) {
        new MainClass().test();
    }

    //The inner classes...

    class InnerClass {
        void test() {
            System.out.println("Value of member: "
                + member);
        }
    }

    class InnerClass2 {
        private int member = 1;

        void test() {
            System.out.println("Value of local
               member: " + member);
            System.out.println("Value of parent
               member: " + MainClass.this.member);
        }
    }
}
```

Results:

supported	issues	difficulty
yes	0	0

Introduction

Inner classes are different in Java and C#. While inner classes in Java can be static (using the modifier `static`)or non-static (without the modifier `static`), there is only one type of inner classes in C#. Confusingly, inner classes in C# behave as static inner classes in Java but are declared without that modifier.

Static inner classes are simply classes that do not lie in a package but in a class. They do not have any relation to objects of their enclosing class. Except for accessing private (static) members of the enclosing class, declaring static inner classes is equivalent to declaring a class in a package (or similar structures in C#).

In contrast, instances of non-static inner classes are always associated with an object of their enclosing class (called the enclosing instance). The execution environment automatically provides this association that has no direct representation in Java code.

This example shows how non-static inner classes are converted to C# by using static inner classes.

Java Source Code

```
1  class MainClass {
2
3      private int member = 0;
4
5      void test() {
6          new InnerClass().test();
7          new InnerClass2().test();
8      }
9
10     public static void main(String[] args) {
11         new MainClass().test();
12     }
13
14     //The inner classes...
15
16     class InnerClass {
17         void test() {
18             System.out.println("Value of member: " +
                   member);
19         }
20     }
21
22     class InnerClass2 {
23         private int member = 1;
24
25         void test() {
```

```
26              System.out.println("Value of local member: "
                    + member);
27              System.out.println("Value of parent member: "
                    + MainClass.this.member);
28          }
29      }
30  }
```

There are two non-static inner classes. The first one, `InnerClass`, contains only one method `test` which simply puts out the value of the field `member` that is declared in `MainClass`. The second inner class `InnerClass2` also contains a method `test` and additionally a field `member`. The method prints the value of both fields, the `member` of `MainClass` and the `member` of `InnerClass2`.

Conversion Results

MainClass.java

There are no issues for this file.

Converted C# Code

```
1  using System;
2  class MainClass
3  {
4
5      private int member = 0;
6
7      internal virtual void  test()
8      {
9          new InnerClass(this).test();
10         new InnerClass2(this).test();
11     }
12
13     [STAThread]
14     public static void  Main(System.String[] args)
15     {
16         new MainClass().test();
17     }
18
19     //UPGRADE_NOTE: Field 'EnclosingInstance' was added to class '
           InnerClass' to access its enclosing instance.
20     //The inner classes...
```

```
21
22      internal class InnerClass
23      {
24          public InnerClass(MainClass enclosingInstance)
25          {
26              InitBlock(enclosingInstance);
27          }
28          private void  InitBlock(MainClass
                enclosingInstance)
29          {
30              this.enclosingInstance = enclosingInstance;
31          }
32          private MainClass enclosingInstance;
33          public MainClass Enclosing_Instance
34          {
35              get
36              {
37                  return enclosingInstance;
38              }
39
40          }
41          internal virtual void  test()
42          {
43              System.Console.Out.WriteLine("Value of member
                    : " + Enclosing_Instance.member);
44          }
45      }
46
47      //UPGRADE_NOTE: Field 'EnclosingInstance' was added to class '
            InnerClass2' to access its enclosing instance.
48      internal class InnerClass2
49      {
50          public InnerClass2(MainClass enclosingInstance)
51          {
52              InitBlock(enclosingInstance);
53          }
54          private void  InitBlock(MainClass
                enclosingInstance)
55          {
56              this.enclosingInstance = enclosingInstance;
57          }
58          private MainClass enclosingInstance;
59          public MainClass Enclosing_Instance
60          {
61              get
62              {
63                  return enclosingInstance;
```

```
64                }
65
66            }
67            private int member = 1;
68
69            internal virtual void  test()
70            {
71                System.Console.Out.WriteLine("Value of local
                      member: " + member);
72                System.Console.Out.WriteLine("Value of parent
                       member: " + Enclosing_Instance.member);
73            }
74        }
75    }
```

Most code in the converted source is due to the lack of non-static inner classes in C#.

Analysis

Each non-static inner class is converted to a static inner class that contains a private field `enclosingInstance`, a public read-only property `Enclosing_Instance` and a constructor with a parameter `enclosingInstance` of type `MainClass`. These entities are used to emulate Java's reference to the enclosing instance of a non-static inner class.

The different constructor declaration leads to a change in the code: whenever an object of the inner class is created the enclosing instance must explicitly be provided.

9.5 Identifier Scope

Summary

Test: Identifier Scope

```
public void hello(String text) {
    System.out.println("Hello " + text);
}

public void test() {
    hello("world!");
    String hello = "you!";
    hello(hello);
}
```

Results:

supported	issues	difficulty
no	0	2

Introduction

Even though Java and C# are quite similar languages they differ in some points. One such point is the slightly different treatment of identifiers and identifier scopes. The purpose of this test is to find out whether the JLCA is prepared to cope with Java programs that use identifiers in a fashion that is not permitted in C#.

Java Source Code

```
1  public class MainClass {
2      public void hello(String text) {
3          System.out.println("Hello " + text);
4      }
5
6      public void test() {
7          hello("world!");
8          String hello = "you!";
9          hello(hello);
10     }
11
12     public static void main(String[] args) {
13         new MainClass().test();
14     }
15 }
```

The Java source code for this test is very simple. `MainClass` has a method `hello` which just prints a short message to the console. The method `test` uses this `hello` method in order to print two messages to the console. Furthermore, a local variable `hello` is declared in this method. Even though the identifier `hello` denotes two different things the Java compiler is still able to compile this program by looking at the contexts this identifier is used in.

Conversion Results

As the conversion report shows, the JLCA cannot find any problems. However, as we are going to see later, the C# program it produces is not compilable.

MainClass.java

There are no issues for this file.

Converted C# Code

```
1  using System;
2  public class MainClass
3  {
4      public virtual void  hello(System.String text)
5      {
6          System.Console.Out.WriteLine("Hello " + text);
7      }
8
9      public virtual void  test()
10     {
11         hello("world!");
12         System.String hello = "you!";
13         hello(hello);
14     }
15
16     [STAThread]
17     public static void  Main(System.String[] args)
18     {
19         new MainClass().test();
20     }
21 }
```

The C# program is almost identical to the Java program. Nevertheless, it cannot be compiled.

Analysis

The JLCA does not pay attention to the way the `hello` identifier is used in the Java source code. Thus, the same identifier usage appears in the C# program. However, the C# compiler does not allow this identifier overloading and issues the following error messages during the compilation of the converted source code.

```
MainClass.cs(12,17): error CS0136: A local variable named
    'hello' cannot be
        declared in this scope because it would give a
           different meaning to
        'hello', which is already used in a 'parent or
           current' scope to denote
        something else
MainClass.cs(12,17): error CS0654: Method 'MainClass.
   hello(string)' referenced
        without parentheses
MainClass.cs(13,9): error CS0654: Method 'MainClass.hello
   (string)' referenced
        without parentheses
```

The JLCA completely fails to notice this issue. Neither does it rename either the `hello` method or the local variable nor does it warn the user about this problem in the conversion report.

A possible reason for the inability of the C# compiler to find out what exactly `hello` denotes is the C# syntax for delegates. In Java, method identifiers are always followed by parentheses. In C# on the other hand, method identifiers may appear without any parentheses when they are used in the constructors of delegates.

9.6 Assert Keyword

Summary

This test shows the conversion of the `assert`-keyword. This keyword does not exist in C#. It is not converted at all.

Test: `assert` Keyword

```
assert 1+1 == 2;
```

Results:

supported	issues	difficulty
no	1	4[a]

[a] An implementation of the *assert* feature has to be used as it is not part of the C# language.

Java Source Code

```
1 public class MainClass {
2     public static void main(String[] args) {
3         assert 1+1 == 2;
4         System.out.println("Asserted that 1+1 equals 2.")
              ;
5     }
6 }
```

Conversion Results

MainClass.java

Conversion Issues for **MainClass.main(java.lang.String[])**

#	Type	Severity	Description
1	Compile	1	The following fragment of code could not be parsed and was not converted.

Converted C# Code

```
1  using System;
2  public class MainClass
3  {
4      [STAThread]
5      public static void  Main(System.String[] args)
6      {
7          //UPGRADE_ISSUE: The following fragment of code could not be
               parsed and was not converted.
8          assert 1 + 1 == 2;
9          System.Console.Out.WriteLine("Asserted that 1+1
               equals 2.");
10     }
11 }
```

Analysis

Assertions are possible and used in C# but are not directly supported by the language. However there are solutions in C# using libraries. Hence it should be no problem to support this keyword by using a *support class*.

9.7 Exception Hierarchy

Summary

Test: Exception Hierarchy

```
try {
    throw new MyException("Test1");
} catch (Exception e) { .. }

try {
    throw new MyError("Test2");
} catch (Error e) { .. }
```

Results:

supported	issues	difficulty
no	2	5

Introduction

In Java there are two different types of faults: Errors and Exceptions. These are treated differently and are on the same level. In C# there is only one exception branch. Errors cannot be mapped naturally.

Java Source Code

In this class an exception is thrown and caught, then an error is thrown and caught and finally an error is thrown but an exception is caught.

```
1  public class MainClass {
2      public static void main(String[] args) {
3          try {
4              throw new MyException("Test1");
5          } catch (Exception e) {
6              System.out.println("Caught Exception: " + e);
7          }
8
9          try {
10             throw new MyError("Test2");
11         } catch (Error e) {
12             System.out.println("Caught Error: " + e);
13         }
14
15         try {
16             throw new MyError("Test3");
```

```
17          } catch (Exception e) {
18              System.out.println("This code should not be
                    executed.");
19          }
20      }
21  }
```

```
1  public class MyException extends Exception {
2      public MyException(String message) {
3          super(message);
4      }
5  }
```

```
1  public class MyError extends Error {
2      public MyError(String message) {
3          super(message);
4      }
5  }
```

Conversion Results

The conversion results mention different behavior of the `toString()` method which is not relevant here.

MainClass.java

Conversion Issues for
MainClass.main(java.lang.String[])

#	Type	Severity	Description
1	ToDo	2	The equivalent in .NET for method 'java.lang.Throwable.toString' may return a different value.
2	ToDo	2	The equivalent in .NET for method 'java.lang.Throwable.toString' may return a different value.

MyError.java

There are no issues for this file.

MyException.java

There are no issues for this file.

Converted C# Code

This code shows that exceptions are derived from the exception class as in Java.

```
1 using System;
2 [Serializable]
3 public class MyException:System.Exception
4 {
5     public MyException(System.String message):base(
          message)
6     {
7     }
8 }
```

But this code turns out that the mapping of errors is wrong because it is mapped to `ApplicationException` which is a subclass of the `Exception` class.

```
1 using System;
2 [Serializable]
3 public class MyError:System.ApplicationException
4 {
5     public MyError(System.String message):base(message)
6     {
7     }
8 }
```

Because of the different exception hierarchy, the test starting in line 27 fails. An error is thrown. But this error is derived from `ApplicationException` an therefore is a subclass of `Exception` and hence caught in line 31. This is not the same behavior as in Java.

```
1 using System;
2 public class MainClass
3 {
4     [STAThread]
5     public static void  Main(System.String[] args)
6     {
7         try
8         {
9             throw new MyException("Test1");
10        }
11        catch (System.Exception e)
12        {
```

```
13                  //UPGRADE_TODO: The equivalent in .NET for method 'java.lang
                        .Throwable.toString' may return a different value.
14                  System.Console.Out.WriteLine("Caught
                        Exception: " + e);
15              }
16
17              try
18              {
19                  throw new MyError("Test2");
20              }
21              catch (System.ApplicationException e)
22              {
23                  //UPGRADE_TODO: The equivalent in .NET for method 'java.lang
                        .Throwable.toString' may return a different value.
24                  System.Console.Out.WriteLine("Caught Error: "
                        + e);
25              }
26
27              try
28              {
29                  throw new MyError("Test3");
30              }
31              catch (System.Exception e)
32              {
33                  System.Console.Out.WriteLine("This code
                        should not be executed.");
34              }
35          }
36      }
```

Analysis

The mapping of the exception hierarchy of Java in wrong. In the converted code, errors may be caught by catching exceptions. A solution was the introduction of `JavaException` and `JavaError` and the construction of a totally separate exception hierarchy. This may not be wanted for code maintenance.

The developer should keep in mind that the usage of errors will make problems in the converted code. Unfortunately this is not mentioned in the conversion report generated by the JLCA.

9.8 java.lang.reflect.Modifier

Summary

Test: `java.lang.reflect.Modifier`

```
        Class cls = mc.getClass();

        int modifiers = cls.getModifiers();
```

Results:

supported	issues	difficulty
no	2	5

Introduction

This test shows how the `java.lang.reflect.Modifier` class is converted by the JLCA. This class belongs to the Java reflection system. It contains static methods and constants to decode class and member access modifiers. Converting such kind of classes is very difficult for the JLCA as C# and Java have quite different reflection systems.

Java Source Code

The Java source code for this test is very simple. In the first step, a Class object is retrieved. In the second step, the modifiers of this class object are queried and printed to the console.

```
1  import java.lang.reflect.*;
2
3  public class MainClass {
4
5      public static void main(String[] args) {
6          MainClass mc = new MainClass();
7
8          Class cls = mc.getClass();
9
10         int modifiers = cls.getModifiers();
11         System.out.println(Modifier.toString(modifiers));
12     }
13 }
```

Conversion Results

As the conversion results indicate, neither the `getModifiers()` method nor the `Modifier.toString()` method are converted.

MainClass.java

Conversion Issues for **MainClass.main(java.lang.String[])**

#	Type	Severity	Description
1	Compile	1	Method 'java.lang.Class.getModifiers' was not converted.
2	Compile	1	Method 'java.lang.reflect.Modifier.toString' was not converted.

Converted C# Code

The JLCA mappes the first part of the Java code correctly. The second part, however, is left completely untouched.

```
1  using System;
2
3  public class MainClass
4  {
5
6      [STAThread]
7      public static void  Main(System.String[] args)
8      {
9          MainClass mc = new MainClass();
10
11         System.Type cls = mc.GetType();
12
13         //UPGRADE_ISSUE: Method 'java.lang.Class.getModifiers' was not
               converted.
14         //int modifiers = cls.getModifiers();
15         //UPGRADE_ISSUE: Method 'java.lang.reflect.Modifier.toString' was
               not converted.
16         //System.Console.Out.WriteLine(Modifier.toString(modifiers));
17         Console.Out.WriteLine(cls.Attributes);
18     }
19 }
```

Analysis

As expected the JLCA does not succeed in converting the `Modifier` class correctly. In fact, it does not convert any calls related to this class at all. This is quite understandable, considering the different reflection systems the C# and Java languages offer to programmers.

One possible way of converting the `Modifier` class in an adequate way would be to use the `Attributes` property offered by the .NET `System.Type` class. The following example illustrates this:

```
Console.Out.WriteLine(cls.Attributes);
```

9.9 java.lang.reflect.Proxy

Summary

Test: `java.lang.reflect.Proxy`

```
//Create a dynamic proxy for the TestEventListener
   interface...
TestEventListener proxyListener = (
   TestEventListener)Proxy.newProxyInstance(
    TestEventListener.class.getClassLoader()
       ,
    new Class[] { TestEventListener.class },
    myInvokationHandler
);
```

Results:

supported	issues	difficulty
no	8	5

Introduction

The main objective of this test is to find out whether the JLCA is capable of converting the usage of the `java.lang.reflect.Proxy` class. This class belongs to the Java reflection system. As C# and Java have quite different reflection systems the automatic conversion of this class is rather difficult.

`Proxy` provides static methods for creating dynamic proxy classes and instances, and it is also the superclass of all dynamic proxy classes created by those methods. A dynamic proxy class is a class that implements a list of interfaces specified at runtime when the class is created.

Additionally, this test shows how a number of other Java API calls are treated by the JLCA. For example, the `getClassLoader()` method and the `java.lang.reflect.InvocationHandler` interface are tested as well as the `java.util.EventListener` interface and the `java.util.EventListenerList` class.

Java Source Code

This test uses three different classes. `MainClass` contains the main logic of the test. In its `main` method, a `TestEventDispatcher` is instantiated. Then, an ordinary event listener implementing the `TestEventListener` interface and a dynamic proxy instance are added to the listener list of the dispatcher. Finally, with the invocation of the `dispatchTestEvent()` method, the dispatcher sends

a message to all its listeners.

```
1  import java.lang.reflect.InvocationHandler;
2  import java.lang.reflect.Method;
3  import java.lang.reflect.Proxy;
4
5  public class MainClass {
6
7      public static void main(String[] args) {
8
9          TestEventDispatcher ted = new TestEventDispatcher
               ();
10
11         //Add an ordinary event listener...
12         ted.addTestEventListener(new TestEventListener()
               {
13             public void somethingHappened(String
                  description) {
14                 System.out.println("Something happened: "
                      + description);
15             }
16         });
17
18         //Create an invokation handler for a dynamic proxy...
19         InvocationHandler myInvokationHandler = new
              InvocationHandler() {
20             public Object invoke(Object proxy, Method
                  method, Object[] args) throws Throwable {
21                 System.out.println("invoke called:");
22                 System.out.println("\tMethod: " + method)
                      ;
23                 System.out.println("\tArgs:");
24                 for (int i = 0; i < args.length; i++) {
25                     System.out.println("\t " + (i+1) + "
                          : " + args[i]);
26                 }
27                 return null;
28             }
29         };
30
31         //Create a dynamic proxy for the TestEventListener interface...
32         TestEventListener proxyListener = (
              TestEventListener)Proxy.newProxyInstance(
33             TestEventListener.class.getClassLoader(),
34             new Class[] { TestEventListener.class },
35             myInvokationHandler
36         );
```

```
37
38          //Add the dynamic proxy to the listerners list of the test event
                dispatcher...
39          ted.addTestEventListener(proxyListener);
40
41          ted.dispatchTestEvent("but what?!!");
42      }
43
44  }
```

```
1  import java.util.EventListener;
2  import javax.swing.event.EventListenerList;
3
4  public class TestEventDispatcher {
5
6      private EventListenerList _listenerList = new
           EventListenerList();
7
8      public void addTestEventListener(TestEventListener
           listener) {
9          _listenerList.add(TestEventListener.class,
               listener);
10     }
11
12     public void removeTestEventListener(TestEventListener
           listener) {
13         _listenerList.remove(TestEventListener.class,
               listener);
14     }
15
16     public void dispatchTestEvent(String description) {
17         EventListener[] listeners = _listenerList.
               getListeners(TestEventListener.class);
18         for (int i = 0; i < listeners.length; i++) {
19             ((TestEventListener)listeners[i]).
                   somethingHappened(description);
20         }
21     }
22 }
```

```
1  import java.util.EventListener;
2
3  public interface TestEventListener extends EventListener
       {
4      public void somethingHappened(String description);
5  }
```

Conversion Results

The conversion report shows that the JLCA has great difficulties with the conversion of the test program. In fact, it is not able at all to create an adequate C# program.

MainClass.java

Conversion Issues for **MainClass.invoke(java.lang.Object,java.lang.reflect.Method,java.lang.Object[])**

#	Type	Severity	Description
1	ToDo	2	The equivalent in .NET for method 'java.lang.Object.toString' may return a different value.

Conversion Issues for **MainClass.main(java.lang.String[])**

#	Type	Severity	Description
1	Compile	1	Interface 'java.lang.reflect.InvocationHandler' was not converted.
2	Compile	1	Interface 'java.lang.reflect.InvocationHandler' was not converted.
3	Compile	1	Method 'java.lang.reflect.Proxy.newProxyInstance' was not converted.
4	Compile	1	Method 'java.lang.Class.getClassLoader' was not converted.

TestEventDispatcher.java

Conversion Issues for **TestEventDispatcher.dispatchTestEvent(java.lang.String)**

#	Type	Severity	Description
1	Compile	1	Interface 'java.util.EventListener' was not converted.
2	Compile	1	Method 'javax.swing.event.EventListenerList.getListeners' was not converted.

TestEventListener.java

Conversion Issues for
TestEventListener.Declarations

#	Type	Severity	Description
1	Compile	1	Interface 'java.util.EventListener' was not converted.

Converted C# Code

```
1  using System;
2
3  public class MainClass
4  {
5      private class AnonymousClassTestEventListener :
           TestEventListener
6      {
7          public virtual void  somethingHappened(System.
               String description)
8          {
9              System.Console.Out.WriteLine("Something
                   happened: " + description);
10         }
11     }
12     //UPGRADE_ISSUE: Interface 'java.lang.reflect.InvocationHandler' was
           not converted.
13     private class AnonymousClassInvocationHandler :
           InvocationHandler
14     {
15         public virtual System.Object invoke(System.Object
               proxy, System.Reflection.MethodInfo method,
               System.Object[] args)
16         {
17             System.Console.Out.WriteLine("invoke called:"
                   );
18             System.Console.Out.WriteLine("\tMethod: " +
                   method);
19             System.Console.Out.WriteLine("\tArgs:");
20             for (int i = 0; i < args.Length; i++)
21             {
22                 //UPGRADE_TODO: The equivalent in .NET for method 'java
                       .lang.Object.toString' may return a different
```

```
                    value.
23                  System.Console.Out.WriteLine("\t " + (i
                        + 1) + ": " + args[i]);
24              }
25              return null;
26          }
27      }
28
29      [STAThread]
30      public static void  Main(System.String[] args)
31      {
32
33          TestEventDispatcher ted = new TestEventDispatcher
                ();
34
35          //Add an ordinary event listener...
36          ted.addTestEventListener(new
                AnonymousClassTestEventListener());
37
38          //Create an invokation handler for a dynamic proxy...
39          //UPGRADE_ISSUE: Interface 'java.lang.reflect.InvocationHandler'
                was not converted.
40          InvocationHandler myInvokationHandler = new
                AnonymousClassInvocationHandler();
41
42          //Create a dynamic proxy for the TestEventListener interface...
43          //UPGRADE_ISSUE: Method 'java.lang.reflect.Proxy.newProxyInstance'
                was not converted.
44          //UPGRADE_ISSUE: Method 'java.lang.Class.getClassLoader' was not
                converted.
45          TestEventListener proxyListener = (
                TestEventListener) Proxy.newProxyInstance(
                typeof(TestEventListener).getClassLoader(),
                new System.Type[]{typeof(TestEventListener)},
                myInvokationHandler);
46
47          //Add the dynamic proxy to the listerners list of the test event
                dispatcher...
48          ted.addTestEventListener(proxyListener);
49
50          ted.dispatchTestEvent("but what?!!");
51      }
52  }
```

```
1   using System;
2
3   public class TestEventDispatcher
```

```
{

    private System.Collections.Hashtable _listenerList =
       new System.Collections.Hashtable();

    public virtual void  addTestEventListener(
       TestEventListener listener)
    {
        _listenerList.Add(listener, typeof(
           TestEventListener));
    }

    public virtual void  removeTestEventListener(
       TestEventListener listener)
    {
        _listenerList.Remove(listener);
    }

    public virtual void  dispatchTestEvent(System.String
       description)
    {
        //UPGRADE_ISSUE: Interface 'java.util.EventListener' was not
           converted.
        //UPGRADE_ISSUE: Method 'javax.swing.event.EventListenerList.
           getListeners' was not converted.
        EventListener[] listeners = _listenerList.
           getListeners(typeof(TestEventListener));
        for (int i = 0; i < listeners.Length; i++)
        {
            ((TestEventListener) listeners[i]).
               somethingHappened(description);
        }
    }
}
```

```
using System;

//UPGRADE_ISSUE: Interface 'java.util.EventListener' was not converted.
public interface TestEventListener:EventListener
{
    void  somethingHappened(System.String description);
}
```

Analysis

As the conversion report and the C# code illustrate the JLCA is unable to convert the most interesting parts of the test program. Neither does it convert the usage of the `java.lang.reflect.Proxy` and `java.lang.reflect.InvocationHandler` classes and the `getClassLoader()` method, nor is it able to handle the `java.util.EventListener` and `java.util.EventListenerList` classes.

JLCA's incapability of converting the reflection related API calls of this test is due to the fact that the .NET framework simply does not offer the creation of dynamic proxies. JLCA's help system proposes the use of the .NET `System.Runtime.Remoting.Proxies.RealProxy` class for replacing the `java.lang.reflect.InvocationHandler` interface. This, however, does not solve the problem of creating dynamic proxy classes at runtime.

The problems JLCA reports for the conversion of the event handling related API calls are unexpected. Especially the `java.util.EventListenerList` class could be covered a lot better, for instance by using support classes. One of the reasons for JLCA's bad performance in this area might be the different approaches Java and C# take on event handling. While Java uses listener classes that follow the Adapter design pattern for implementing event handling C# uses delegates. Moreover, there is no direct equivalent for the `java.util.EventListenerList` in C# as delegates handle listener lists automatically.

9.10 java.lang.ref.WeakReference

Summary

Test: `java.lang.ref.WeakReference`

```
MainClass mc = new MainClass();
WeakReference wr = new WeakReference(mc,
   _weakReferenceQueue);
```

Results:

supported	issues	difficulty
no	12	5

Introduction

This test is to demonstrate how the JLCA handles the automatic conversion of the `java.lang.ref.WeakReference` class. Additionally, the handling of the `java.lang.ref.ReferenceQueue` class is tested.

Weak references are references that do not prevent their referents from being made finalizable, finalized, and then reclaimed. Reference queues are special queues to which the garbage collector appends registered objects as soon as it detects changes in their reachability.

The automatic conversion of code using these and similar classes is rather difficult as these classes depend on specific behavior of the Java virtual machine and its garbage collector.

Java Source Code

```
1  import java.lang.ref.*;
2
3  public class MainClass {
4
5      private static ReferenceQueue _weakReferenceQueue =
          new ReferenceQueue();
6
7      public static void main(String[] args) {
8          System.out.println("WeakReference test.");
9
10         MainClass mc = new MainClass();
11         WeakReference wr = new WeakReference(mc,
              _weakReferenceQueue);
12
```

```
13          Thread referenceQueueReader = new Thread(new
               Runnable() {
14              public void run() {
15                  try {
16                      Reference r = null;
17                      r = _weakReferenceQueue.remove();
18                      System.out.println("Removed reference
                            from reference queue.");
19                      Object o = r.get();
20                      System.out.println("Referent of weak
                           reference: " + o);
21                  } catch (InterruptedException e) {
22                      System.err.println("Caught
                           InterruptedException:");
23                      e.printStackTrace();
24                  }
25              }
26          });
27
28          referenceQueueReader.setDaemon(true);
29          referenceQueueReader.start();
30
31          System.out.println("Giving up reference to object
               .");
32          mc = null;
33
34          System.out.println("Referent of weak reference: "
                + wr.get());
35          System.out.println("Starting GC.");
36          System.gc();
37          System.out.println("Referent of weak reference: "
                + wr.get());
38
39          try {
40              Thread.sleep(1000);
41          } catch (InterruptedException e) {
42                  System.err.println("Caught
                       InterruptedException:");
43                  e.printStackTrace();
44          }
45      }
46
47      protected void finalize() throws Throwable {
48          super.finalize();
49          System.out.println("Finalized MainClass object.")
               ;
50      }
```

```
51
52 }
```

The Java source code for this test consist of a single class. First, a new `MainClass` object is created. For this object a new `WeakReference` is created. Moreover, the object is registered with the static reference queue that belongs to the `MainClass` class.

Afterwards, a thread is started. This thread continuously removes all references the garbage collector places into the reference queue and checks whether they are still reachable.

At the end, the reference to the `MainClass` object is given up and the garbage collector is invoked directly in order to observe the behavior of the weak reference and the reference queue.

The output of the program is as follows:

```
WeakReference test.
Giving up reference to object.
Referent of weak reference: MainClass@17182c1
Starting GC.
Finalized MainClass object.
Removed reference from reference queue.
Referent of weak reference: null
Referent of weak reference: null
```

This means that the `MainClass` object is still reachable after all strong references have been given up. However, as soon as the garbage collector is started, this object is immediately finalized and thus no longer reachable. Additionally, the garbage collector places the weak reference into the reference queue where it is removed by the reader thread that was previously started.

Conversion Results

As the conversion results show the JLCA does not succeed in converting the usage of the `java.lang.ref.WeakReference` and `java.lang.ref.ReferenceQueue` classes.

MainClass.java

Conversion Issues for
MainClass.run()

#	Type	Severity	Description
1	Compile	1	Class 'java.lang.ref.Reference' was not converted.

2	Compile	1	Method 'java.lang.ref. ReferenceQueue.remove' was not converted.
3	Compile	1	Method 'java.lang.ref.Reference.get' was not converted.
4	ToDo	2	The equivalent in .NET for method 'java.lang.Object.toString' may return a different value.

Conversion Issues for **MainClass.main(java.lang.String[])**

#	Type	Severity	Description
2	Compile	1	Constructor 'java.lang.ref. WeakReference.WeakReference' was not converted.
4	Compile	1	Method 'java.lang.ref.Reference.get' was not converted.
6	Compile	1	Method 'java.lang.ref.Reference.get' was not converted.
1	ToDo	2	Method 'java.lang.Thread.sleep' was converted to 'System.Threading. Thread.Sleep' which has a different behavior.
3	ToDo	2	The equivalent in .NET for method 'java.lang.Object.toString' may return a different value.
5	ToDo	2	The equivalent in .NET for method 'java.lang.Object.toString' may return a different value.

Conversion Issues for **MainClass.Declarations**

#	Type	Severity	Description
1	Compile	1	Class 'java.lang.ref.ReferenceQueue' was not converted.
2	Compile	1	Constructor 'java.lang.ref. ReferenceQueue.ReferenceQueue' was not converted.

Converted C# Code

An interesting aspect of the C# code is how the `java.lang.Runnable` interface and the `java.lang.Thread` class are handled. The JLCA creates a support class for Java's `Thread` class in order to provide similar functionality in C#.

The rest of the code shows that the JLCA is unable to convert the `java.lang.ref` classes.

```
1  using System;
2
3  public class MainClass
4  {
5      private class AnonymousClassRunnable :
           IThreadRunnable
6      {
7          public virtual void  Run()
8          {
9              try
10             {
11                 //UPGRADE_ISSUE: Class 'java.lang.ref.Reference' was
                       not converted.
12                 Reference r = null;
13                 //UPGRADE_ISSUE: Method 'java.lang.ref.ReferenceQueue.
                       remove' was not converted.
14                 r = MainClass._weakReferenceQueue.remove
                       ();
15                 System.Console.Out.WriteLine("Removed
                       reference from reference queue.");
16                 //UPGRADE_ISSUE: Method 'java.lang.ref.Reference.get'
                       was not converted.
17                 System.Object o = r.get_Renamed();
18                 //UPGRADE_TODO: The equivalent in .NET for method 'java
                       .lang.Object.toString' may return a different
                       value.
19                 System.Console.Out.WriteLine("Referent of
                        weak reference: " + o);
20             }
21             catch (System.Threading.
                   ThreadInterruptedException e)
22             {
23                 System.Console.Error.WriteLine("Caught
                       InterruptedException:");
24                 SupportClass.WriteStackTrace(e, Console.
                       Error);
25             }
26         }
```

```
27      }
28
29      //UPGRADE_ISSUE: Class 'java.lang.ref.ReferenceQueue' was not converted
            .
30      //UPGRADE_ISSUE: Constructor 'java.lang.ref.ReferenceQueue.
            ReferenceQueue' was not converted.
31      private static ReferenceQueue _weakReferenceQueue =
            new ReferenceQueue();
32
33      [STAThread]
34      public static void  Main(System.String[] args)
35      {
36          System.Console.Out.WriteLine("WeakReference test.
                ");
37
38          MainClass mc = new MainClass();
39          //UPGRADE_ISSUE: Constructor 'java.lang.ref.WeakReference.
                WeakReference' was not converted.
40          System.WeakReference wr = new WeakReference(mc,
                _weakReferenceQueue);
41
42          SupportClass.ThreadClass referenceQueueReader =
                new SupportClass.ThreadClass(new System.
                Threading.ThreadStart(new
                AnonymousClassRunnable().Run));
43
44          referenceQueueReader.IsBackground = true;
45          referenceQueueReader.Start();
46
47          System.Console.Out.WriteLine("Giving up reference
                 to object.");
48          mc = null;
49
50          //UPGRADE_TODO: The equivalent in .NET for method 'java.lang.
                Object.toString' may return a different value.
51          //UPGRADE_ISSUE: Method 'java.lang.ref.Reference.get' was not
                converted.
52          System.Console.Out.WriteLine("Referent of weak
                reference: " + wr.get_Renamed());
53          System.Console.Out.WriteLine("Starting GC.");
54          System.GC.Collect();
55          //UPGRADE_TODO: The equivalent in .NET for method 'java.lang.
                Object.toString' may return a different value.
56          //UPGRADE_ISSUE: Method 'java.lang.ref.Reference.get' was not
                converted.
57          System.Console.Out.WriteLine("Referent of weak
                reference: " + wr.get_Renamed());
```

```
58
59          try
60          {
61              //UPGRADE_TODO: Method 'java.lang.Thread.sleep' was converted
                    to 'System.Threading.Thread.Sleep' which has a different
                    behavior.
62              System.Threading.Thread.Sleep(new System.
                    TimeSpan((System.Int64) 10000 * 1000));
63          }
64          catch (System.Threading.
                ThreadInterruptedException e)
65          {
66              System.Console.Error.WriteLine("Caught
                    InterruptedException:");
67              SupportClass.WriteStackTrace(e, Console.Error
                    );
68          }
69      }
70
71      ~MainClass()
72      {
73          //UPGRADE_NOTE: Call to 'super.finalize()' was removed.
74          System.Console.Out.WriteLine("Finalized MainClass
                object.");
75      }
76  }
```

Analysis

As the .NET framework and the Java runtime system have different APIs for handling weak references the JLCA does not succeed in performing automatic conversions when the `java.lang.ref.WeakReference` and `java.lang.ref.ReferenceQueue` classes are used. Especially for the `java.lang.ref.ReferenceQueue` class there seems to be no real equivalent in the .NET framework.

However, the .NET framework features a `System.WeakReference` class that offers some of the functionality required for automatic conversions. The JLCA could perform better if it used this class. Furthermore, the employment of support classes could facilitate the automatic conversion of usages of the `java.lang.ref.ReferenceQueue` class.

Similar tests have been performed with the `java.lang.ref.SoftReference` and the `java.lang.ref.PhantomReference` classes. These tests have produced results that are mostly equivalent to the results of this test.

9.11 java.lang.Runtime.addShutdownHook

Summary

Test: `java.lang.Runtime.addShutdownHook`

```
Runtime.getRuntime().addShutdownHook(new
   Thread(
    new Runnable() {
        public void run() {
            System.out.println("Running
               shutdown hook.");
        }
    }
));
```

Results:

supported	issues	difficulty
no	1	3

Introduction

The purpose of this test is to show whether the JLCA supports the automatic conversion of the `java.lang.Runtime.addShutdownHook` method. This method lets applications register threads that are started when the Java virtual machine shuts down. Thus, this method can be used to assure that special cleanup procedures are always carried out.

Java Source Code

```
1  import java.lang.Thread.*;
2
3  class MainClass {
4
5      public static void main(String[] args) {
6          System.out.println("Adding shutdown hook.");
7
8          Runtime.getRuntime().addShutdownHook(new Thread(
9              new Runnable() {
10                 public void run() {
11                     System.out.println("Running shutdown
                           hook.");
12                 }
13             }
14         ));
```

```
15
16          System.out.println("Exiting...");
17      }
18  }
```

The Java source code for this test is fairly simple. Only one shutdown hook thread is registered. This thread, when run, just prints a short message to the standard output stream and then exits.

The program produces the following console output:

```
Adding shutdown hook.
Exiting...
Running shutdown hook.
```

Conversion Results

The conversion results show that the JLCA does not support the automatic conversion of the `java.lang.Runtime.addShutdownHook` method.

MainClass.java

Conversion Issues for **MainClass.main(java.lang.String[])**

#	Type	Severity	Description
1	Compile	1	Method 'java.lang.Runtime.addShutdownHook' was not converted.

Converted C# Code

```
1  using System;
2
3  class MainClass
4  {
5      private class AnonymousClassRunnable :
           IThreadRunnable
6      {
7          public virtual void  Run()
8          {
9              System.Console.Out.WriteLine("Running
                   shutdown hook.");
```

```
10          }
11      }
12
13      [STAThread]
14      public static void  Main(System.String[] args)
15      {
16          System.Console.Out.WriteLine("Adding shutdown
                hook.");
17
18          //UPGRADE_ISSUE: Method 'java.lang.Runtime.addShutdownHook' was
                not converted.
19          System.Diagnostics.Process.GetCurrentProcess().
                addShutdownHook(new SupportClass.ThreadClass(
                new System.Threading.ThreadStart(new
                AnonymousClassRunnable().Run)));
20
21          System.Console.Out.WriteLine("Exiting...");
22      }
23  }
```

Analysis

While the JLCA is able to convert the creation of the shutdown hook thread it does not succeed in mapping the call to the `java.lang.Runtime.addShutdownHook` method. Interestingly, the `java.lang.Runtime` class is mapped to the `System.Diagnostics.Process` class. While this may actually work for other methods associated with the `Runtime` class this mapping is not correct for the `addShutdownHook` method.

A far better mapping for the `addShutdownHook` method is the `ProcessExit` event of the `System.AppDomain` class. This event occurs on the default application domain when the default application domain's parent process exits. Thus, handlers for this event can carry out cleanup procedures almost in the same way as Java shutdown hook threads. The following C# listing shows a small program that behaves in the same way as the original Java program.

```
1  using System;
2
3  namespace AppDomainEvents
4  {
5      public class MainClass
6      {
7
8          public static void Main()
9          {
```

```
10              Console.WriteLine("Registering for
                    ProcessExit event.");
11              AppDomain.CurrentDomain.ProcessExit += new
                    EventHandler(CurrentDomain_ProcessExit);
12              Console.WriteLine("Exiting.");
13          }
14
15          private static void CurrentDomain_ProcessExit(
                object sender, EventArgs e)
16          {
17              Console.WriteLine("Running ProcessExit event
                    handler.");
18          }
19      }
20  }
```

However, there is still a difference between the semantics of the `addShutdownHook` method and the `ProcessExit` event. In Java, all shutdown hook threads are executed concurrently. The .NET runtime environment normally calls event handlers sequentially. This means that all event handlers for the `ProcessExit` event are executed one after the other.

For overcoming this difference a support class could be implemented. This support class could map the behavior of the `addShutdownHook` method, e.g. by providing a means for registering threads. For starting all these threads when the .NET runtime environment shuts down this support class could make use of the `ProcessExit` event.

9.12 Swing: Hello World

Summary

This example tests a simple Java program using the *Java Swing* technology. Detailed tests to specific features of Swing will follow. It turns out that the conversion of such a simple application works quite well and just one line of code has to be added to the generated source.

Test: `javax.swing` in general

```
//Create a JFrame...
JFrame mainFrame = new JFrame("Test");
mainFrame.setDefaultCloseOperation(JFrame.
   EXIT_ON_CLOSE);
mainFrame.setSize(128,128);
mainFrame.getContentPane().setLayout(new
   FlowLayout());

//Add a label...
JLabel helloLabel = new JLabel("Hello World!
   ");
mainFrame.getContentPane().add(helloLabel);
```

Results:

supported	issues	difficulty
yes	10	1

Introduction

This program just creates a label and a button that may be used to quit the application. Additionally it uses the *FlowLayout* to arrange the elements. Layout managers are not supported in C# and must have been solved with an own implementation inside a support class.

Java Source Code

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class SwingHelloWorld {
6
7     public static void main(String[] args) {
8         //Create a JFrame...
```

```
9           JFrame mainFrame = new JFrame("Test");
10          mainFrame.setDefaultCloseOperation(JFrame.
               EXIT_ON_CLOSE);
11          mainFrame.setSize(128,128);
12          mainFrame.getContentPane().setLayout(new
               FlowLayout());
13
14          //Add a label...
15          JLabel helloLabel = new JLabel("Hello World!");
16          mainFrame.getContentPane().add(helloLabel);
17
18          //Add a button...
19          JButton exitButton = new JButton("Exit");
20          exitButton.addMouseListener(new MouseAdapter() {
21              public void mouseClicked(MouseEvent e) {
22                  System.exit(0);
23              }
24          });
25          mainFrame.getContentPane().add(exitButton);
26
27          //Show the JFrame...
28          mainFrame.show();
29      }
30  }
```

Conversion Results

SwingHelloWorld.java

Conversion Issues for **SwingHelloWorld.main(java.lang.String[])**

#	Type	Severity	Description
1	ToDo	2	Method 'java.awt.Component.setSize' was converted to 'System.Windows.Forms.Control.Size' which has a different behavior.
2	ToDo	2	Method 'javax.swing.JFrame.getContentPane' was converted to 'System.Windows.Forms.Form' which has a different behavior.
3	ToDo	2	Constructor 'java.awt.FlowLayout.FlowLayout' was converted to 'System.Object[]' which has a different behavior.

4	ToDo	2	Method 'java.awt.Container.add' was converted to 'System.Windows.Forms.ContainerControl.Controls.Add' which has a different behavior.
5	ToDo	2	Method 'javax.swing.JFrame.getContentPane' was converted to 'System.Windows.Forms.Form' which has a different behavior.
6	ToDo	2	Method 'java.awt.Container.add' was converted to 'System.Windows.Forms.ContainerControl.Controls.Add' which has a different behavior.
7	ToDo	2	Method 'javax.swing.JFrame.getContentPane' was converted to 'System.Windows.Forms.Form' which has a different behavior.
8	ToDo	2	'System.Windows.Forms.Application.Run' must be called to start a main form.

Converted C# Code

The anonymous class for event handling in Java is converted in a private class called `AnonymousClassMouseAdapter`. Another event handler for the `mouseDown` event is registered but just sets a class-wide static field `state2` that is never used again.

Line 51 had to be added to start the application while line 50 was commented out.

```
1  using System;
2
3  public class SwingHelloWorld
4  {
5      static private System.Int32 state2;
6      private class AnonymousClassMouseAdapter
7      {
8          public void  mouseClicked(System.Object
               event_sender, System.EventArgs e)
9          {
10             System.Environment.Exit(0);
11         }
12     }
```

```
13      private static void  mouseDown(System.Object
           event_sender, System.Windows.Forms.MouseEventArgs
           e)
14      {
15          state2 = ((int) e.Button | (int) System.Windows.
               Forms.Control.ModifierKeys);
16      }
17
18      [STAThread]
19      public static void  Main(System.String[] args)
20      {
21          //Create a JFrame...
22          System.Windows.Forms.Form mainFrame =
               SupportClass.FormSupport.CreateForm("Test");
23          mainFrame.Closing += new System.ComponentModel.
               CancelEventHandler(SwingHelloWorld.
               SwingHelloWorld_Closing_EXIT_ON_CLOSE);
24          //UPGRADE_TODO: Method 'java.awt.Component.setSize' was converted
               to 'System.Windows.Forms.Control.Size' which has a different
               behavior.
25          mainFrame.Size = new System.Drawing.Size
               (128, 128);
26          //UPGRADE_TODO: Method 'javax.swing.JFrame.getContentPane' was
               converted to 'System.Windows.Forms.Form' which has a different
               behavior.
27          //UPGRADE_TODO: Constructor 'java.awt.FlowLayout.FlowLayout' was
               converted to 'System.Object[]' which has a different behavior.
28          ((System.Windows.Forms.ContainerControl)
               mainFrame).Tag = new System.Object[]{(int)
               System.Drawing.ContentAlignment.TopCenter
               , 5, 5};
29          ((System.Windows.Forms.ContainerControl)
               mainFrame).Layout += new System.Windows.Forms.
               LayoutEventHandler(SupportClass.
               FlowLayoutResize);
30
31          //Add a label...
32          System.Windows.Forms.Label temp_label;
33          temp_label = new System.Windows.Forms.Label();
34          temp_label.Text = "Hello World!";
35          System.Windows.Forms.Label helloLabel =
               temp_label;
36          //UPGRADE_TODO: Method 'javax.swing.JFrame.getContentPane' was
               converted to 'System.Windows.Forms.Form' which has a different
               behavior.
37          //UPGRADE_TODO: Method 'java.awt.Container.add' was converted to '
               System.Windows.Forms.ContainerControl.Controls.Add' which has a
```

```
                different behavior.
38          ((System.Windows.Forms.ContainerControl)
               mainFrame).Controls.Add(helloLabel);
39
40          //Add a button...
41          System.Windows.Forms.Button exitButton =
               SupportClass.ButtonSupport.
               CreateStandardButton("Exit");
42          exitButton.MouseDown += new System.Windows.Forms.
               MouseEventHandler(SwingHelloWorld.mouseDown);
43          exitButton.Click += new System.EventHandler(new
               AnonymousClassMouseAdapter().mouseClicked);
44          //UPGRADE_TODO: Method 'javax.swing.JFrame.getContentPane' was
               converted to 'System.Windows.Forms.Form' which has a different
               behavior.
45          //UPGRADE_TODO: Method 'java.awt.Container.add' was converted to '
               System.Windows.Forms.ContainerControl.Controls.Add' which has a
                different behavior.
46          ((System.Windows.Forms.ContainerControl)
               mainFrame).Controls.Add(exitButton);
47
48          //Show the JFrame...
49          //UPGRADE_TODO: 'System.Windows.Forms.Application.Run' must be
               called to start a main form.
50          //mainFrame.Show();
51          System.Windows.Forms.Application.Run(mainFrame);
52      }
53      private static void
            SwingHelloWorld_Closing_EXIT_ON_CLOSE(System.
           Object sender, System.ComponentModel.
           CancelEventArgs  e)
54      {
55          e.Cancel = true;
56          SupportClass.CloseOperation((System.Windows.Forms
               .Form) sender, 3);
57      }
58  }
```

Analysis

The standard swing components are converted fairly naturally.

The flow layout was managed using a quite complicated support class. An event handler is registered to the layout event to rearrange the elements during layouting. This works even better than the original Java version for it refreshes the layout

continuously while resizing the window while in Java the refreshing is performed after releasing the button.

A somewhat strange effect is the generation of a obviously useless event handler. This may be used to get the state of the button but this is not a deal of this example.

To start the application the `Application.Run` method must be called. This is not performed by the generator but a comment in line 49 points to this fact.

The visual conversion of swing to C# works quite well. The following images show the initial windows and resized versions.

Visualisation

Java:

C#:

9.13 Swing: Simple menu example

Summary

Test: `javax.swing.JMenu, JMenuBar, JMenuItem` (simple)

```
        menuBar = new JMenuBar();

        menu = new JMenu("A Menu");
        menuBar.add(menu);

        menuItem = new JMenuItem("An item");
        menu.add(menuItem);
```

Results:

supported	issues	difficulty
yes	2	0

Introduction

Simple menus are treated nearly the same way in Java an C#. Problems may occur with specialties like checkboxes which will be examined later in this document.

Java Source Code

The relevant function here is `createMenuBar()` in line 8. A `JMenuBar` is created and a menu and an item are added.

```
1  import javax.swing.JMenu;
2  import javax.swing.JMenuItem;
3  import javax.swing.JMenuBar;
4  import javax.swing.JFrame;
5
6  public class MenuSimple {
7
8      public JMenuBar createMenuBar() {
9          JMenuBar menuBar;
10         JMenu menu;
11         JMenuItem menuItem;
12
13         menuBar = new JMenuBar();
14
15         menu = new JMenu("A Menu");
16         menuBar.add(menu);
17
```

```
18          menuItem = new JMenuItem("An item");
19          menu.add(menuItem);
20
21          return menuBar;
22      }
23
24      private static void createAndShowGUI() {
25          JFrame frame = new JFrame("Menu1");
26          frame.setDefaultCloseOperation(JFrame.
               EXIT_ON_CLOSE);
27
28          MenuSimple demo = new MenuSimple();
29          frame.setJMenuBar(demo.createMenuBar());
30
31          frame.setSize(450, 260);
32          frame.setVisible(true);
33      }
34
35      public static void main(String[] args) {
36          createAndShowGUI();
37      }
38  }
```

Conversion Results

MenuSimple.java

Conversion Issues for **MenuSimple.createMenuBar()**

#	Type	Severity	Description
1	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenuBar.add' may return a different value.
2	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.

Conversion Issues for **MenuSimple.createAndShowGUI()**

#	Type	Severity	Description
1	ToDo	2	Method 'java.awt.Component.setSize' was converted to 'System.Windows.Forms.Control.Size' which has a different behavior.

2	ToDo	2	Method 'java.awt.Component.setVisible' was converted to 'System.Windows.Forms.Control.Visible' which has a different behavior.
3	ToDo	2	'System.Windows.Forms.Application.Run' must be called to start a main form.

Converted C# Code

The converted code looks nearly the same as the Java code. The converted code works as expected but is not most efficient or even not as a human would write it: The creation of `menu` and `menuItem` is done in two steps. First the objects are created and the title is given in a second line of code while both classes support giving a title inside the constructor.

```
1  using System;
2
3  public class MenuSimple
4  {
5
6      public virtual System.Windows.Forms.MainMenu
           createMenuBar()
7      {
8          System.Windows.Forms.MainMenu menuBar;
9          System.Windows.Forms.MenuItem menu;
10         System.Windows.Forms.MenuItem menuItem;
11
12         menuBar = new System.Windows.Forms.MainMenu();
13
14         menu = new System.Windows.Forms.MenuItem();
15         menu.Text = "A Menu";
16         menuBar.MenuItems.Add(menu);
17
18         menuItem = new System.Windows.Forms.MenuItem();
19         menuItem.Text = "An item";
20         menu.MenuItems.Add(menuItem);
21
22         return menuBar;
23     }
24
25     private static void  createAndShowGUI()
```

```
26      {
27          System.Windows.Forms.Form frame = SupportClass.
               FormSupport.CreateForm("Menu1");
28          frame.Closing += new System.ComponentModel.
               CancelEventHandler(MenuSimple.
               MenuSimple_Closing_EXIT_ON_CLOSE);
29
30          MenuSimple demo = new MenuSimple();
31          frame.Menu = demo.createMenuBar();
32
33          //UPGRADE_TODO: Method 'java.awt.Component.setSize' was converted
               to 'System.Windows.Forms.Control.Size' which has a different
               behavior.
34          frame.Size = new System.Drawing.Size(450, 260);
35          //UPGRADE_TODO: Method 'java.awt.Component.setVisible' was
               converted to 'System.Windows.Forms.Control.Visible' which has a
                different behavior.
36          //UPGRADE_TODO: 'System.Windows.Forms.Application.Run' must be
               called to start a main form.
37          frame.Visible = true;
38          System.Windows.Forms.Application.Run(frame);
39      }
40
41      [STAThread]
42      public static void  Main(System.String[] args)
43      {
44          createAndShowGUI();
45      }
46      private static void  MenuSimple_Closing_EXIT_ON_CLOSE
           (System.Object sender, System.ComponentModel.
           CancelEventArgs  e)
47      {
48          e.Cancel = true;
49          SupportClass.CloseOperation((System.Windows.Forms
               .Form) sender, 3);
50      }
51  }
```

9.14 Swing: Menu with Submenu

Summary

Test: `javax.swing.JMenu` (used as submenu)

```
menu = new JMenu("A Menu");
submenu = new JMenu("A submenu");
...
menu.add(submenu);
```

Results:

supported	issues	difficulty
yes	5	0

Introduction

In this test a menu containing a submenu item is created. This functionality is available both in Java and C#.

Java Source Code

The relevant function here is `createMenuBar()` in line 8.

```
1  import javax.swing.JMenu;
2  import javax.swing.JMenuItem;
3  import javax.swing.JMenuBar;
4  import javax.swing.JFrame;
5
6  public class MenuSubmenu {
7
8      public JMenuBar createMenuBar() {
9          JMenuBar menuBar;
10         JMenu menu, submenu;
11         JMenuItem menuItem;
12
13         menuBar = new JMenuBar();
14         menu = new JMenu("A Menu");
15         menuBar.add(menu);
16
17         menuItem = new JMenuItem("An item");
18         menu.add(menuItem);
19
20         menu.addSeparator();
21
```

```
22          //a submenu
23          submenu = new JMenu("A submenu");
24
25          menuItem = new JMenuItem("An item in the submenu"
                );
26          submenu.add(menuItem);
27
28          menuItem = new JMenuItem("Another item");
29          submenu.add(menuItem);
30
31          menu.add(submenu);
32
33          return menuBar;
34      }
35
36      private static void createAndShowGUI() {
37          JFrame frame = new JFrame("Submenu demo");
38          frame.setDefaultCloseOperation(JFrame.
                EXIT_ON_CLOSE);
39
40          MenuSubmenu demo = new MenuSubmenu();
41          frame.setJMenuBar(demo.createMenuBar());
42
43          frame.setSize(450, 260);
44          frame.setVisible(true);
45      }
46
47      public static void main(String[] args) {
48          createAndShowGUI();
49      }
50
51  }
```

Conversion Results

There are fife Issues in the relevant method. But all issues concern the same method: `add` which may return a different value. This value is not used in the given example and will be rarely used in other applications.

MenuSubmenu.java

Conversion Issues for
MenuSubmenu.createMenuBar()

#	Type	Severity	Description

1	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenuBar.add' may return a different value.
2	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.
3	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.
4	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.
5	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.

Conversion Issues for **MenuSubmenu.createAndShowGUI()**

#	Type	Severity	Description
1	ToDo	2	Method 'java.awt.Component.setSize' was converted to 'System.Windows.Forms.Control.Size' which has a different behavior.
2	ToDo	2	Method 'java.awt.Component.setVisible' was converted to 'System.Windows.Forms.Control.Visible' which has a different behavior.
3	ToDo	2	'System.Windows.Forms.Application.Run' must be called to start a main form.

Converted C# Code

The conversion again worked correctly. An interesting line is line 21. The `menu.addSeparator()` command has changed to a menu item containing a `-` as its title which is the usual way to add separators in C#.

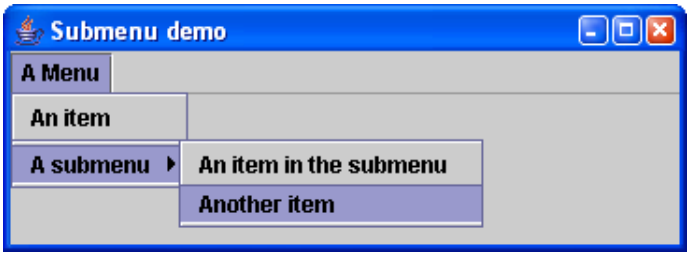
```
1 using System;
2
3 public class MenuSubmenu
4 {
```

```
5
6       public virtual System.Windows.Forms.MainMenu
           createMenuBar()
7       {
8           System.Windows.Forms.MainMenu menuBar;
9           System.Windows.Forms.MenuItem menu, submenu;
10          System.Windows.Forms.MenuItem menuItem;
11
12          menuBar = new System.Windows.Forms.MainMenu();
13          menu = new System.Windows.Forms.MenuItem();
14          menu.Text = "A Menu";
15          menuBar.MenuItems.Add(menu);
16
17          menuItem = new System.Windows.Forms.MenuItem();
18          menuItem.Text = "An item";
19          menu.MenuItems.Add(menuItem);
20
21          menu.MenuItems.Add(new System.Windows.Forms.
               MenuItem("-"));
22
23          //a submenu
24          submenu = new System.Windows.Forms.MenuItem();
25          submenu.Text = "A submenu";
26
27          menuItem = new System.Windows.Forms.MenuItem();
28          menuItem.Text = "An item in the submenu";
29          submenu.MenuItems.Add(menuItem);
30
31          menuItem = new System.Windows.Forms.MenuItem();
32          menuItem.Text = "Another item";
33          submenu.MenuItems.Add(menuItem);
34
35          menu.MenuItems.Add(submenu);
36
37          return menuBar;
38      }
39
40      private static void  createAndShowGUI()
41      {
42          System.Windows.Forms.Form frame = SupportClass.
               FormSupport.CreateForm("Submenu demo");
43          frame.Closing += new System.ComponentModel.
               CancelEventHandler(MenuSubmenu.
               MenuSubmenu_Closing_EXIT_ON_CLOSE);
44
45          MenuSubmenu demo = new MenuSubmenu();
46          frame.Menu = demo.createMenuBar();
```

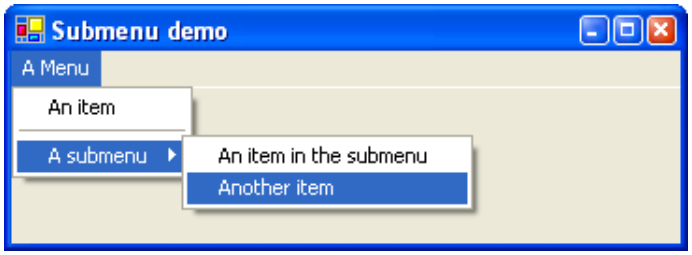
```
47
48          //UPGRADE_TODO: Method 'java.awt.Component.setSize' was converted
                to 'System.Windows.Forms.Control.Size' which has a different
                behavior.
49          frame.Size = new System.Drawing.Size(450, 260);
50          //UPGRADE_TODO: Method 'java.awt.Component.setVisible' was
                converted to 'System.Windows.Forms.Control.Visible' which has a
                 different behavior.
51          //UPGRADE_TODO: 'System.Windows.Forms.Application.Run' must be
                called to start a main form.
52          frame.Visible = true;
53          System.Windows.Forms.Application.Run();
54      }
55
56      [STAThread]
57      public static void  Main(System.String[] args)
58      {
59          createAndShowGUI();
60      }
61      private static void
            MenuSubmenu_Closing_EXIT_ON_CLOSE(System.Object
           sender, System.ComponentModel.CancelEventArgs  e)
62      {
63          e.Cancel = true;
64          SupportClass.CloseOperation((System.Windows.Forms
                .Form) sender, 3);
65      }
66  }
```

Visualisation

Java:



C#:


Submenu demo
A Menu
An item
A submenu
An item in the submenu
Another item

9.15 Swing: Menu with Checkbox and Radiobuttons

Summary

Test: `javax.swing.JRadioButtonMenuItem, JCheckBoxMenuItem`

```
rbMenuItem = new JRadioButtonMenuItem("A
   radio button menu item");
rbMenuItem.setSelected(true);
group.add(rbMenuItem);
menu.add(rbMenuItem);
```

Results:

supported	issues	difficulty
yes	11	4

Introduction

Checked menus are supported by both frameworks. Hence a correct conversion was expected.

Java Source Code

In Java there is a special menu item called `JRadioButtonMenuItem` for alternative selection and `JCheckBoxMenuItem` for single decision selection. The radiobuttons are added to a group to determine the alternatives.

```
1  import javax.swing.JMenu;
2  import javax.swing.JMenuBar;
3  import javax.swing.JFrame;
4  import javax.swing.JCheckBoxMenuItem;
5  import javax.swing.JRadioButtonMenuItem;
6  import javax.swing.ButtonGroup;
7
8  public class MenuCheckbox {
9
10     public JMenuBar createMenuBar() {
11         JMenuBar menuBar;
12         JMenu menu;
13         JRadioButtonMenuItem rbMenuItem;
14         JCheckBoxMenuItem cbMenuItem;
15
```

```
16          menuBar = new JMenuBar();
17          menu = new JMenu("A Menu");
18          menuBar.add(menu);
19
20          //a group of radio button menu items
21          ButtonGroup group = new ButtonGroup();
22
23          rbMenuItem = new JRadioButtonMenuItem("A radio
               button menu item");
24          rbMenuItem.setSelected(true);
25          group.add(rbMenuItem);
26          menu.add(rbMenuItem);
27
28          rbMenuItem = new JRadioButtonMenuItem("Another
               one");
29          group.add(rbMenuItem);
30          menu.add(rbMenuItem);
31
32          menu.addSeparator();
33
34          //a group of check box menu items
35          cbMenuItem = new JCheckBoxMenuItem("A check box
               menu item");
36          menu.add(cbMenuItem);
37
38          cbMenuItem = new JCheckBoxMenuItem("Another one")
               ;
39          menu.add(cbMenuItem);
40
41          return menuBar;
42      }
43
44      private static void createAndShowGUI() {
45          JFrame frame = new JFrame("Checkbox and
               Radiobutton demo");
46          frame.setDefaultCloseOperation(JFrame.
               EXIT_ON_CLOSE);
47
48          MenuCheckbox demo = new MenuCheckbox();
49          frame.setJMenuBar(demo.createMenuBar());
50
51          frame.setSize(450, 260);
52          frame.setVisible(true);
53      }
54
55      public static void main(String[] args) {
56          createAndShowGUI();
```

```
57         }
58 }
```

Conversion Results

MenuCheckbox.java

Conversion Issues for
MenuCheckbox.createMenuBar()

#	Type	Severity	Description
1	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenuBar.add' may return a different value.
2	ToDo	2	Class 'javax.swing.ButtonGroup' was converted to 'System.Collections. ArrayList' which has a different behavior.
3	ToDo	2	Class 'javax.swing.ButtonGroup' was converted to 'System.Collections. ArrayList' which has a different behavior.
4	ToDo	2	Constructor 'javax.swing. JRadioButtonMenuItem. JRadioButtonMenuItem' was converted to 'System.Windows.Forms.MenuItem' which has a different behavior.
5	ToDo	2	The equivalent in .NET for method 'javax.swing.ButtonGroup.add' may return a different value.
6	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.
7	ToDo	2	Constructor 'javax.swing. JRadioButtonMenuItem. JRadioButtonMenuItem' was converted to 'System.Windows.Forms.MenuItem' which has a different behavior.
8	ToDo	2	The equivalent in .NET for method 'javax.swing.ButtonGroup.add' may return a different value.
9	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.

10	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.
11	ToDo	2	The equivalent in .NET for method 'javax.swing.JMenu.add' may return a different value.

Conversion Issues for **MenuCheckbox.createAndShowGUI()**

#	Type	Severity	Description
1	ToDo	2	Method 'java.awt.Component.setSize' was converted to 'System.Windows.Forms.Control.Size' which has a different behavior.
2	ToDo	2	Method 'java.awt.Component.setVisible' was converted to 'System.Windows.Forms.Control.Visible' which has a different behavior.
3	ToDo	2	'System.Windows.Forms.Application.Run' must be called to start a main form.

Converted C# Code

The converted code does not behave like the Java source. The visual representation is correct but there is no built in behavior for check boxes or radio buttons as in Java menus.

In line 87 and line 100 there are two event handlers that had to be created manually. These handlers deal with click events and manage the checking and unchecking of the menu items.

```
1 using System;
2
3 public class MenuCheckbox
4 {
5     //moved from createMenuBar()
6     private System.Collections.ArrayList _buttonGroup =
          new System.Collections.ArrayList();
7
```

```
8       public virtual System.Windows.Forms.MainMenu
           createMenuBar()
9       {
10          System.Windows.Forms.MainMenu menuBar;
11          System.Windows.Forms.MenuItem menu;
12          System.Windows.Forms.MenuItem rbMenuItem;
13          System.Windows.Forms.MenuItem cbMenuItem;
14
15          menuBar = new System.Windows.Forms.MainMenu();
16          menu = new System.Windows.Forms.MenuItem();
17          menu.Text = "A Menu";
18          menuBar.MenuItems.Add(menu);
19
20          //a group of radio button menu items
21          //UPGRADE_TODO: Class 'javax.swing.ButtonGroup' was converted to '
               System.Collections.ArrayList' which has a different behavior.
22          //moved to top as instance variable
23          //System.Collections.ArrayList group = new System.Collections.
               ArrayList();
24
25          System.Windows.Forms.MenuItem temp_menuitem;
26          //UPGRADE_TODO: Constructor 'javax.swing.JRadioButtonMenuItem.
               JRadioButtonMenuItem' was converted to 'System.Windows.Forms.
               MenuItem' which has a different behavior.
27          temp_menuitem = new System.Windows.Forms.MenuItem
               ();
28          temp_menuitem.RadioCheck = true;
29          temp_menuitem.Text = "A radio button menu item";
30          rbMenuItem = temp_menuitem;
31          rbMenuItem.Checked = true;
32          _buttonGroup.Add((System.Object) rbMenuItem);
33          rbMenuItem.Click += new EventHandler(
               rbMenuItem_Click);
34          menu.MenuItems.Add(rbMenuItem);
35
36          System.Windows.Forms.MenuItem temp_menuitem2;
37          //UPGRADE_TODO: Constructor 'javax.swing.JRadioButtonMenuItem.
               JRadioButtonMenuItem' was converted to 'System.Windows.Forms.
               MenuItem' which has a different behavior.
38          temp_menuitem2 = new System.Windows.Forms.
               MenuItem();
39          temp_menuitem2.RadioCheck = true;
40          temp_menuitem2.Text = "Another one";
41          rbMenuItem = temp_menuitem2;
42          _buttonGroup.Add((System.Object) rbMenuItem);
43          rbMenuItem.Click += new EventHandler(
               rbMenuItem_Click);
```

```
44          menu.MenuItems.Add(rbMenuItem);
45
46          menu.MenuItems.Add(new System.Windows.Forms.
               MenuItem("-"));
47
48          //a group of check box menu items
49          cbMenuItem = new System.Windows.Forms.MenuItem("A
                check box menu item");
50          cbMenuItem.Click += new EventHandler(
               cbMenuItem_Click);
51          menu.MenuItems.Add(cbMenuItem);
52
53          cbMenuItem = new System.Windows.Forms.MenuItem("
               Another one");
54          cbMenuItem.Click += new EventHandler(
               cbMenuItem_Click);
55          menu.MenuItems.Add(cbMenuItem);
56
57          return menuBar;
58      }
59
60      private static void  createAndShowGUI()
61      {
62          System.Windows.Forms.Form frame = SupportClass.
               FormSupport.CreateForm("Checkbox and
               Radiobutton demo");
63          frame.Closing += new System.ComponentModel.
               CancelEventHandler(MenuCheckbox.
               MenuCheckbox_Closing_EXIT_ON_CLOSE);
64
65          MenuCheckbox demo = new MenuCheckbox();
66          frame.Menu = demo.createMenuBar();
67
68          //UPGRADE_TODO: Method 'java.awt.Component.setSize' was converted
               to 'System.Windows.Forms.Control.Size' which has a different
               behavior.
69          frame.Size = new System.Drawing.Size(450, 260);
70          //UPGRADE_TODO: Method 'java.awt.Component.setVisible' was
               converted to 'System.Windows.Forms.Control.Visible' which has a
                different behavior.
71          //UPGRADE_TODO: 'System.Windows.Forms.Application.Run' must be
               called to start a main form.
72          frame.Visible = true;
73          System.Windows.Forms.Application.Run();
74      }
75
76      [STAThread]
```

```
77      public static void  Main(System.String[] args)
78      {
79          createAndShowGUI();
80      }
81      private static void
            MenuCheckbox_Closing_EXIT_ON_CLOSE(System.Object
           sender, System.ComponentModel.CancelEventArgs  e)
82      {
83          e.Cancel = true;
84          SupportClass.CloseOperation((System.Windows.Forms
               .Form) sender, 3);
85      }
86
87      private void cbMenuItem_Click(object sender,
           EventArgs e)
88      {
89          System.Windows.Forms.MenuItem item = (System.
               Windows.Forms.MenuItem)sender;
90          if (item.Checked)
91          {
92              item.Checked = false;
93          }
94          else
95          {
96              item.Checked = true;
97          }
98      }
99
100     private void rbMenuItem_Click(object sender,
           EventArgs e)
101     {
102         foreach (System.Windows.Forms.MenuItem item in
               _buttonGroup)
103         {
104             if (sender == item)
105             {
106                 item.Checked = true;
107             }
108             else
109             {
110                 item.Checked = false;
111             }
112         }
113     }
114 }
```

Analysis

The Java behavior could have been adapted with a simple event handler. By adding several handlers to a single event, the programmer does not have to deal with these handlers performing the selection but only has to deal with a handler that manages the semantic of the menu item.

9.16 Swing: FileChooser

Summary

Test: `javax.swing.JFileChooser, javax.swing.filechooser.FileFilter`

```
JFileChooser fc = new JFileChooser();
fc.addChoosableFileFilter(new MyFileFilter("txt"));
```

Results:

supported	issues	difficulty
yes	11	2

Introduction

This test demonstrates the conversion of the `FileChooser` dialog box. It is a simple Application that pops up a *Save File Dialog* when clicking on the button. In Java it is necessary to add `FileFilter` objects. Those filter do not exist in C#. While the file filters are very powerful and may do nearly anything to decide whether a file should be displayed or not, in C# only filtering by file extensions is possible. In this example file extension filtering is used that can be mapped to the C# dialog.

Java Source Code

In line 9 the file chooser is created and from line 13 on configured with choosable file filters that result in the drop down box to let the user choose the extension.

The class `MyFileFilter` (line 41)is used to filter the file extensions.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.filechooser.FileFilter;
5  import java.io.File;
6
7  public class FileChooser {
8
9      JFileChooser fc = new JFileChooser();
10
11     FileChooser() {
12         //Configure JFileChooser
13         fc.addChoosableFileFilter(new MyFileFilter("txt")
               );
```

```
14          fc.addChoosableFileFilter(new MyFileFilter("doc")
               );
15          fc.addChoosableFileFilter(new MyFileFilter("jpg")
               );
16          fc.addChoosableFileFilter(new MyFileFilter("gif")
               );
17
18          //Create a JFrame...
19          JFrame mainFrame = new JFrame("FileChooser Demo")
               ;
20          mainFrame.setDefaultCloseOperation(JFrame.
               EXIT_ON_CLOSE);
21          mainFrame.setSize(128,128);
22          mainFrame.getContentPane().setLayout(new
               FlowLayout());
23
24          //Add a button...
25          JButton button = new JButton("FileChooser");
26          button.addMouseListener(new MouseAdapter() {
27              public void mouseClicked(MouseEvent e) {
28                  fc.showOpenDialog(null);
29              }
30          });
31          mainFrame.getContentPane().add(button);
32
33          //Show the JFrame...
34          mainFrame.show();
35      }
36
37      public static void main(String[] args) {
38          FileChooser demo = new FileChooser();
39      }
40
41      class MyFileFilter extends FileFilter {
42
43          String extension;
44
45          public MyFileFilter(String extension) {
46              this.extension = extension;
47          }
48
49          public boolean accept(File f) {
50              if (f.isDirectory()) {
51                  return true;
52              }
53
54              String e = Utils.getExtension(f);
```

```
55              if (e != null) {
56                  if (e.equals(this.extension)) {
57                      return true;
58                  } else {
59                      return false;
60                  }
61              }
62
63              return false;
64          }
65
66          public String getDescription() {
67              return "*."+this.extension;
68          }
69      }
70  }
```

An additional class `Utils` is used to query file extensions from files. Since file filtering works totally different in C#, this class is no longer necessary in the converted version and therefore not shown here.

Conversion Results

FileChooser.java

Conversion Issues for **FileChooser.mouseClicked(java.awt.event.MouseEvent)**

#	Type	Severity	Description
1	ToDo	2	The equivalent in .NET for method 'javax.swing.JFileChooser. showOpenDialog' may return a different value.

Conversion Issues for **FileChooser.FileChooser()**

#	Type	Severity	Description
1	Compile	1	Method 'javax.swing.JFileChooser. addChoosableFileFilter' was not converted.
2	Compile	1	Method 'javax.swing.JFileChooser. addChoosableFileFilter' was not converted.

3	Compile	1	Method 'javax.swing.JFileChooser.addChoosableFileFilter' was not converted.
4	Compile	1	Method 'javax.swing.JFileChooser.addChoosableFileFilter' was not converted.
5	ToDo	2	Method 'java.awt.Component.setSize' was converted to 'System.Windows.Forms.Control.Size' which has a different behavior.
6	ToDo	2	Method 'javax.swing.JFrame.getContentPane' was converted to 'System.Windows.Forms.Form' which has a different behavior.
7	ToDo	2	Constructor 'java.awt.FlowLayout.FlowLayout' was converted to 'System.Object[]' which has a different behavior.
8	ToDo	2	Method 'java.awt.Container.add' was converted to 'System.Windows.Forms.ContainerControl.Controls.Add' which has a different behavior.
9	ToDo	2	Method 'javax.swing.JFrame.getContentPane' was converted to 'System.Windows.Forms.Form' which has a different behavior.
10	ToDo	2	'System.Windows.Forms.Application.Run' must be called to start a main form.

Conversion Issues for **FileChooser.MyFileFilter.Declarations**

#	Type	Severity	Description
3	Compile	1	Class 'javax.swing.filechooser.FileFilter' was not converted.
1	Compile	2	Class hierarchy differences between 'javax.swing.JFileChooser' and 'System.Windows.Forms.FileDialog' may cause compilation errors.
2	ToDo	2	Constructor may need to be changed depending on function performed by the 'System.Windows.Forms.FileDialog' object.

Utils.java

There are no issues for this file.

Converted C# Code

JLCA does not convert the file filtering at all. As shown from line 41 is just tells the user that the lines were not converted. The solution to this issue is the insertion of line 50. C# uses pairs of description and extension devided by |. Those pairs again are devided by |.

```
1  using System;
2
3  public class FileChooser
4  {
5      static private System.Int32 state2;
6      //UPGRADE_NOTE: Field 'EnclosingInstance' was added to class '
           AnonymousClassMouseAdapter' to access its enclosing instance.
7      private class AnonymousClassMouseAdapter
8      {
9          public AnonymousClassMouseAdapter(FileChooser
               enclosingInstance)
10         {
11             InitBlock(enclosingInstance);
12         }
13         private void  InitBlock(FileChooser
               enclosingInstance)
14         {
15             this.enclosingInstance = enclosingInstance;
16         }
17         private FileChooser enclosingInstance;
18         public FileChooser Enclosing_Instance
19         {
20             get
21             {
22                 return enclosingInstance;
23             }
24
25         }
26         public void  mouseClicked(System.Object
               event_sender, System.EventArgs e)
27         {
```

```
28                  Enclosing_Instance.fc.ShowDialog(null);
29              }
30          }
31          private static void  mouseDown(System.Object
               event_sender, System.Windows.Forms.MouseEventArgs
               e)
32          {
33              state2 = ((int) e.Button | (int) System.Windows.
                   Forms.Control.ModifierKeys);
34          }
35
36          //UPGRADE_TODO: Constructor may need to be changed depending on function
                performed by the 'System.Windows.Forms.FileDialog' object.
37          internal System.Windows.Forms.FileDialog fc = new
               System.Windows.Forms.OpenFileDialog();
38
39          internal FileChooser()
40          {
41              //Configure JFileChooser
42              //UPGRADE_ISSUE: Method 'javax.swing.JFileChooser.
                   addChoosableFileFilter' was not converted.
43              fc.addChoosableFileFilter(new MyFileFilter(this, "txt"));
44              //UPGRADE_ISSUE: Method 'javax.swing.JFileChooser.
                   addChoosableFileFilter' was not converted.
45              fc.addChoosableFileFilter(new MyFileFilter(this, "doc"));
46              //UPGRADE_ISSUE: Method 'javax.swing.JFileChooser.
                   addChoosableFileFilter' was not converted.
47              fc.addChoosableFileFilter(new MyFileFilter(this, "jpg"));
48              //UPGRADE_ISSUE: Method 'javax.swing.JFileChooser.
                   addChoosableFileFilter' was not converted.
49              fc.addChoosableFileFilter(new MyFileFilter(this, "gif"));
50              fc.Filter = "Alle Dateien|*.*|*.txt|*.txt|*.doc
                   |*.doc|*.jpg|*.jpg|*.gif|*.gif";
51
52              //Create a JFrame...
53              System.Windows.Forms.Form mainFrame =
                   SupportClass.FormSupport.CreateForm("
                   FileChooser Demo");
54              mainFrame.Closing += new System.ComponentModel.
                   CancelEventHandler(this.
                   FileChooser_Closing_EXIT_ON_CLOSE);
55              //UPGRADE_TODO: Method 'java.awt.Component.setSize' was converted
                   to 'System.Windows.Forms.Control.Size' which has a different
                   behavior.
56              mainFrame.Size = new System.Drawing.Size
                   (128, 128);
57              //UPGRADE_TODO: Method 'javax.swing.JFrame.getContentPane' was
```

```
            converted to 'System.Windows.Forms.Form' which has a different
            behavior.
58          //UPGRADE_TODO: Constructor 'java.awt.FlowLayout.FlowLayout' was
            converted to 'System.Object[]' which has a different behavior.
59          ((System.Windows.Forms.ContainerControl)
            mainFrame).Tag = new System.Object[]{(int)
            System.Drawing.ContentAlignment.TopCenter
            , 5, 5};
60          ((System.Windows.Forms.ContainerControl)
            mainFrame).Layout += new System.Windows.Forms.
            LayoutEventHandler(SupportClass.
            FlowLayoutResize);
61
62          //Add a button...
63          System.Windows.Forms.Button button = SupportClass
            .ButtonSupport.CreateStandardButton("
            FileChooser");
64          button.MouseDown += new System.Windows.Forms.
            MouseEventHandler(FileChooser.mouseDown);
65          button.Click += new System.EventHandler(new
            AnonymousClassMouseAdapter(this).mouseClicked)
            ;
66          //UPGRADE_TODO: Method 'javax.swing.JFrame.getContentPane' was
            converted to 'System.Windows.Forms.Form' which has a different
            behavior.
67          //UPGRADE_TODO: Method 'java.awt.Container.add' was converted to '
            System.Windows.Forms.ContainerControl.Controls.Add' which has a
             different behavior.
68          ((System.Windows.Forms.ContainerControl)
            mainFrame).Controls.Add(button);
69
70          //Show the JFrame...
71          //UPGRADE_TODO: 'System.Windows.Forms.Application.Run' must be
            called to start a main form.
72          mainFrame.Show();
73          System.Windows.Forms.Application.Run(mainFrame);
74      }
75
76      [STAThread]
77      public static void  Main(System.String[] args)
78      {
79          FileChooser demo = new FileChooser();
80      }
81
82
83      private void  FileChooser_Closing_EXIT_ON_CLOSE(
            System.Object sender, System.ComponentModel.
```

```
            CancelEventArgs  e)
84      {
85          e.Cancel = true;
86          SupportClass.CloseOperation((System.Windows.Forms
                .Form) sender, 3);
87      }
88  }
```

Analysis

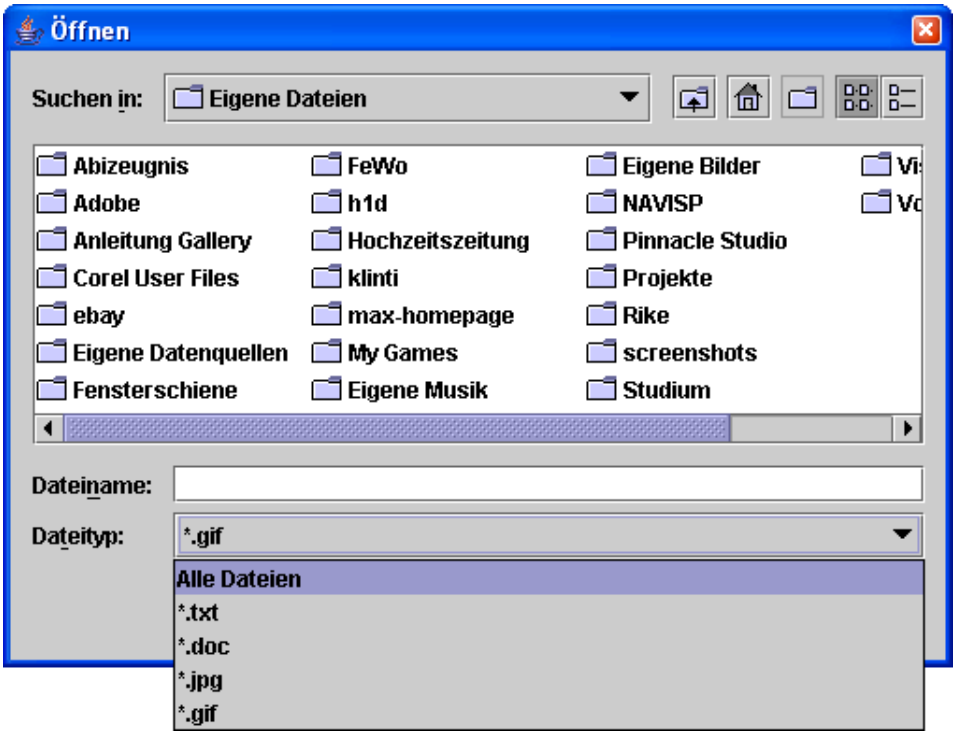
The pure conversion of the file dialog works with one problem: Java uses the user's home directory as starting directory while C# starts in the current working directory (i.e. the starting directory of the application). JLCA does not adjust this difference. Another test has to show the behavior when a starting directory is given explicitly.

The conversion of Java's powerful file filters fails completely although it is possible to express the same semantics in case of file extension filtering. Admittedly this was a very hard task for the JLCA to recognize whether file extension filtering is implemented in the file filters or something more complicated.

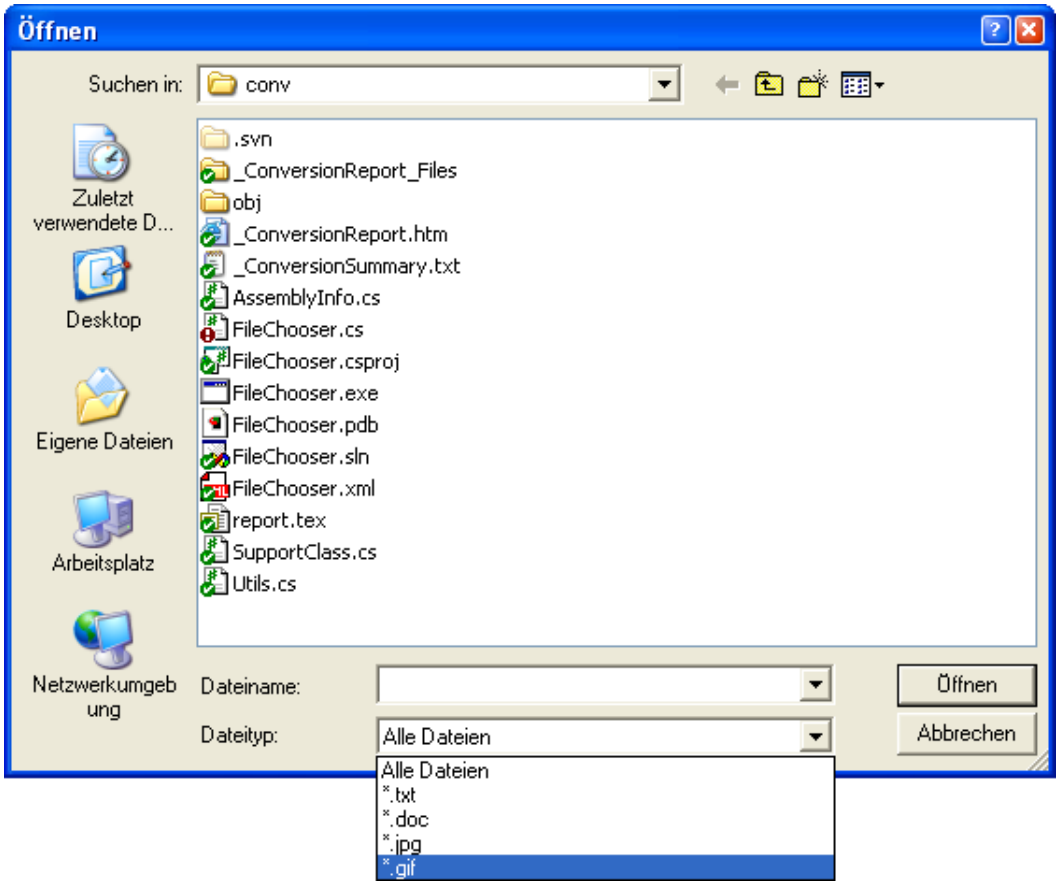
The appearance of the dialog box in C# is – as expected – windows style and will the current windows style on each version.

Visualisation

Java:


Öffnen
Suchen in:
Eigene Dateien
Abizeugnis
Adobe
Anleitung Gallery
Corel User Files
ebay
Eigene Datenquellen
Fensterschiene
FeWo
h1d
Hochzeitszeitung
klinti
max-homepage
My Games
Eigene Musik
Eigene Bilder
NAVISP
Pinnacle Studio
Projekte
Rike
screenshots
Studium
Dateiname:
Dateityp:
*.gif
Alle Dateien
*.txt
*.doc
*.jpg
*.gif

C#:


Öffnen
Suchen in:
conv
Zuletzt verwendete D...
Desktop
Eigene Dateien
Arbeitsplatz
Netzwerkumgebung
.svn
_ConversionReport_Files
obj
_ConversionReport.htm
_ConversionSummary.txt
AssemblyInfo.cs
FileChooser.cs
FileChooser.csproj
FileChooser.exe
FileChooser.pdb
FileChooser.sln
FileChooser.xml
report.tex
SupportClass.cs
Utils.cs
Dateiname:
Dateityp:
Alle Dateien
Öffnen
Abbrechen
Alle Dateien
*.txt
*.doc
*.jpg
*.gif

chapter 10 Related Work

10.1 Borland Janeva

The goal of Borland Janeva is to provide a high performance connectivity between the Microsoft .NET Framework on the one side and the J2EE and CORBA environments on the other. With the help of the Janeva runtime, this tool allows .NET-based applications to access heterogeneous J2EE and CORB server-side components via the Internet Inter-ORB Protocol (IIOP), thus avoiding many problems Web Services and many other proprietary bridging solutions bring with them. Janeva is also tightly integrated with popular Microsoft .NET Framework development environments,including Borland C#-Builder and Microsoft Visual Studio .NET.

Janeva consists of 2 parts: a set of code generators that produce the .NET stubs needed to connect to J2EE and CORBA servers during development and the afore mentioned runtime, which is embedded directly within the deployed applications.

10.1.1 Janeva compilers

Within Janeva, two compilers are included. One is for connecting to J2EE servers; the other is for connecting to CORBA servers. Although the J2EE-based compiler reads interfaces specified in Java RMI and the CORBA-based compiler reads interfaces specified in the OMG IDL, both compilers produce .NET stubs. These stubs, which can optionally be configured to use the Microsoft .NET Remoting Framework, are packaged in to .NET Assemblies that can then be imported into a development tool supporting .NET. Because the stubs do not target a specific language, but rather the Common Type System (CTS), they are directly accessible in any .NET language and can be inspected by any tool supporting .NET assemblies.

The generated stubs consist of two layers, the ease-of-use layer and the raw stub layer. The bottom layer of the assembly is the raw stub layer. This layer uses the Janeva runtime to access the low-level server technology, such as the Naming Service, the Transaction Service, and the EJB Home and Remote objects. The .NET client can access this raw layer for a direct client mapping of the J2EE server technologies.

The ease-of-use layer on the other hand provides a .NET-oriented mapping of the raw J2EE server technologies. It maps the J2EE design patterns into the corresponding design patterns of the Microsoft .NET Framework, for example the Naming Service style lookups of J2EE are converted to URL style lookups of .NET Remoting and calls to J2EE Home objects are converted to .NET Remoting instantiation calls.

10.1.2 Janeva runtime

The Janeva runtime is an implementation of Borland Enterprise Server client-side functionality and therefore supports a rich set of middleware capabilities, including Marshaling, Connection Management and so on. The only significant capability that is not included in full is the Portable Object Adapter, which is required to implement CORBA servers. The Janeva runtime is compliant with the Microsoft CLR and is therefore code that is executed and managed within the .NET Framework itself.

10.1.3 Conclusion

As already mentioned, Janeva's goal is not to convert whole J2EE or CORBA applications to any .NET Language, but to generate a link between these two environments. This is accomplished without any modifications done on the server side, but with the generation of .NET stubs from the J2EE and CORBA interfaces, so that you only have to write a new client to your application but don't have to touch your (working) servers. Therefore it may be best suited in situations where companies want to keep their current servers for a number of reasons and just want to open their applications to .NET clients (for the new market, for better GUIs or for other reasons).

10.2 Ja.NET

Ja.NET is a bridge between Java and Microsoft .NET. With its help it is possible to write clients for Enterprise Java Beans in any language supported by the CLR, access .NET components from any type of Java object and therefore mix and match web server technologies with other middleware technologies (for example you can use ASP.NET with EJBs or Java Servlets with .NET components) without re-writing any code. Using this tool, Java components appear to be CLR components and CLR components appear to be Java components.

To make this happen, Ja.Nets developer Intrinsyc states that their tool leverages .NET Remoting. Therefore it has the usual remoting advantages over any Web Service based bridging solution (for example support for callbacks, activation and lifetime control of remote objects by the client and so on). As you can configure

and extend the transport protocol and data formatting in .NET Remoting, Ja.NET is open to many protocols and data formattings. Currently it supports both HTTP and TCP/IP transport protocols, and either SOAP or binary data formatting.

Ja.NET consists of 5 components, the Ja.NET Runtime, the Janetor configuration tool, used to create a configuration file that configures the Ja.NET runtime with core details such as the location, type and channel of remote objects and so on, the GenNet development tool, which is used to generate .NET proxies from Java classes, the GenJava development tool, used to generate Java proxies from .NET classes and the GenService windows service. This service service, which is invoked by GenNet or GenJava to read (in the case of GenJava) or write (in the case of GenNet) .NET assemblies, is the only non-Java component of Ja.NET and is only deployable on Windows platforms. All other components are 100

Capabilities (Version 1.5)

- Support for .NET Framework 1.1 and 1.0
- Supported JVM versions 1.2.2, 1.3.1 and 1.4.0
- see http://j-integra.intrinsyc.com/ja.net/doc/release.html for explicit details

10.2.1 Example

In order to make Ja.NET work you have to install and use a number of tools. If you want to access an EJB from .NET for example, you first have to start the GenNet tool that generates a .NET component containing proxies for the EJB client-side classes. Then you have to let Janetor generate a Web application archive (WAR) file containing all the files to be deployed in the Web server. The WAR file can be then easily deployed in any Web server that supports servlets. The CLR client, which can be written in any .NET language, can access the EJBs as if it were accessing local CLR components.

10.2.2 Conclusion

As it is hard to get any deeper insights into Ja.NET without testing it, it wouldn't make much sense to give any conclusion until we finish testing.

10.3 IIOP.NET

IIOP.NET is basically a .NET remoting channel based on the IIOP protocol and converts the .NET type system to CORBA and vise versa. It is an open-source

project hosted on sourceforge (http://iiop-net.sourceforge.net/). It works with many ORBs and J2EE servers.

10.3.1 Components

This tool consists basically of three parts:

1. a .NET remoting channel to communicate with ORBs via IIOP (`IIOPChannel`)
2. the `CLSToIDLGenerator`, a tool to generate IDL descriptions from .NET assemblies
3. the `IDLToCLSCompiler`, a compiler to create assemblies from IDL files that can be used to access CORBA services with .NET.

10.3.2 Conclusion

Like Janeva and some other tools, IIOP.NET tries to connect .NET programs with Java via CORBA. It may not be used to convert these Languages.

If a client to a distributed system is converted from Java to C# using the JLCA for a better look and feel and better support of MS Windows GUI capabilities, it may be connected to the original Java server application through CORBA by using one of these tools.

10.4 Automatic Language Conversion Case Study

This is based on a paper by Andrey A. Terekhov[1]. It describes the reengineering process of an application written in *HPS*Rules* to Visual Basic and COBOL as a case study. The application contains more than 1.5 million lines of code and consists of a server part (to be converted into COBOL) and two client parts (one to be converted into Visial Basic and the other into COBOL).

Rules is a programming language which is part of the *High Productivity System (HPS)*. Programs written in this language basically define business logic rules. On the ends of the business logic, there are data declarations (bindings) and GUI window descriptions.

In the opinion of the author, it will never be possible to convert 100% of an application and is focused on reeingeineering. While converting HPS*Rules to COBOL

[1] *Automatic Language Conversion: A Case Study*, Andrey A. Terekhov, see http://ddt.tepkom.ru/alc.pdf

an automation of 90% was achieved on the server side while it was ony 60% on the client side. Conversion to Visual Basic was even worse: only 30% could be converted automatically. This was traced back on the fact that the group that developed the conversion tool had little experience with Visual Basic. After improving the tool an amount of nearly 90% was possible.

This turns out that the development of a conversion tool requires well-founded knowledge of the source and target programming languages. Visual Basic and HPS*Rules are very similar in many ways. Anyhow the first conversion results have been unacceptable. This leads to an assumption that can also be noticed when looking at JLCA: Because of the similarity of the languages, the authors of the conversion tools as well as the users don't pay much attention on understanding both languages as a whole but instead they assume that one language and the programming experience can be mapped to the other one nearly directly.

According to the author, some of the factors affecting the automation level of language conversion are:

1. programming paradigms of the languages
2. expressive power of target and source languages
3. amount of interaction with the user.

While matters 1 and 2 may be marked as OK for Java and C#, the third point is a problem especially when converting *Java Swing* to *Windows Forms*.

III part

Project

This part of the document gives a report about the conversion of a complete project with the Java Language Conversion Assistant. First, the selected project is described and analyzed followed by an account of the issues that arose during the conversion of the application. Finally, there is a summary of the experiences gathered with the automatic conversion of large projects.

chapter 11 Selected Project

In addition to the already covered parts of the analysis of the Java Language Conversion Assistant, it also seemed desirable to examine how the JLCA handles applications that are significantly larger than average test cases. As the Java Swing technology proved to be an essential element of this analysis, the tested application is mostly based on this technology.

The open source project JHotDraw[1] is a Java GUI framework for technical and structured graphics. It has been developed as a "design exercise" but is already quite powerful. Its design relies heavily on some well-known design patterns. JHotDraw's original authors have been Erich Gamma and Thomas Eggenschwiler. The following sections comprise a short overview of JHotDraw, its package organization, and its structure.

11.1 Description of JHotDraw

Based upon Java Swing, JHotDraw defines a basic skeleton for a GUI editor with tools in a tool palette, different views, user-defined graphical figures, and support for saving, loading, and printing drawings. The framework can be customized using inheritance and combining components.

Besides the main drawing window, JHotDraw offers some support for different kinds of windows, such as text editors. With some basic knowledge of JHotDraw's structure, one can extend the framework to include missing functionality. If you run the examples included in the distribution, you can see what a typical, JHotDraw based application looks like. For example, JavaDraw is a standard drawing application that shows very well what is possible with JHotDraw.

For software engineering purposes, JHotDraw is interesting as well. Originally developed in Smalltalk by Kent Beck and Ward Cunningham, JHotDraw was one of the first software development projects explicitly designed for reuse and labeled a framework. It has also been documented very early in terms of design patterns [2],

[1] See http://jhotdraw.sourceforge.net for details.

[2] ftp://st.cs.uiuc.edu/pub/papers/HotDraw/documenting-frameworks.ps

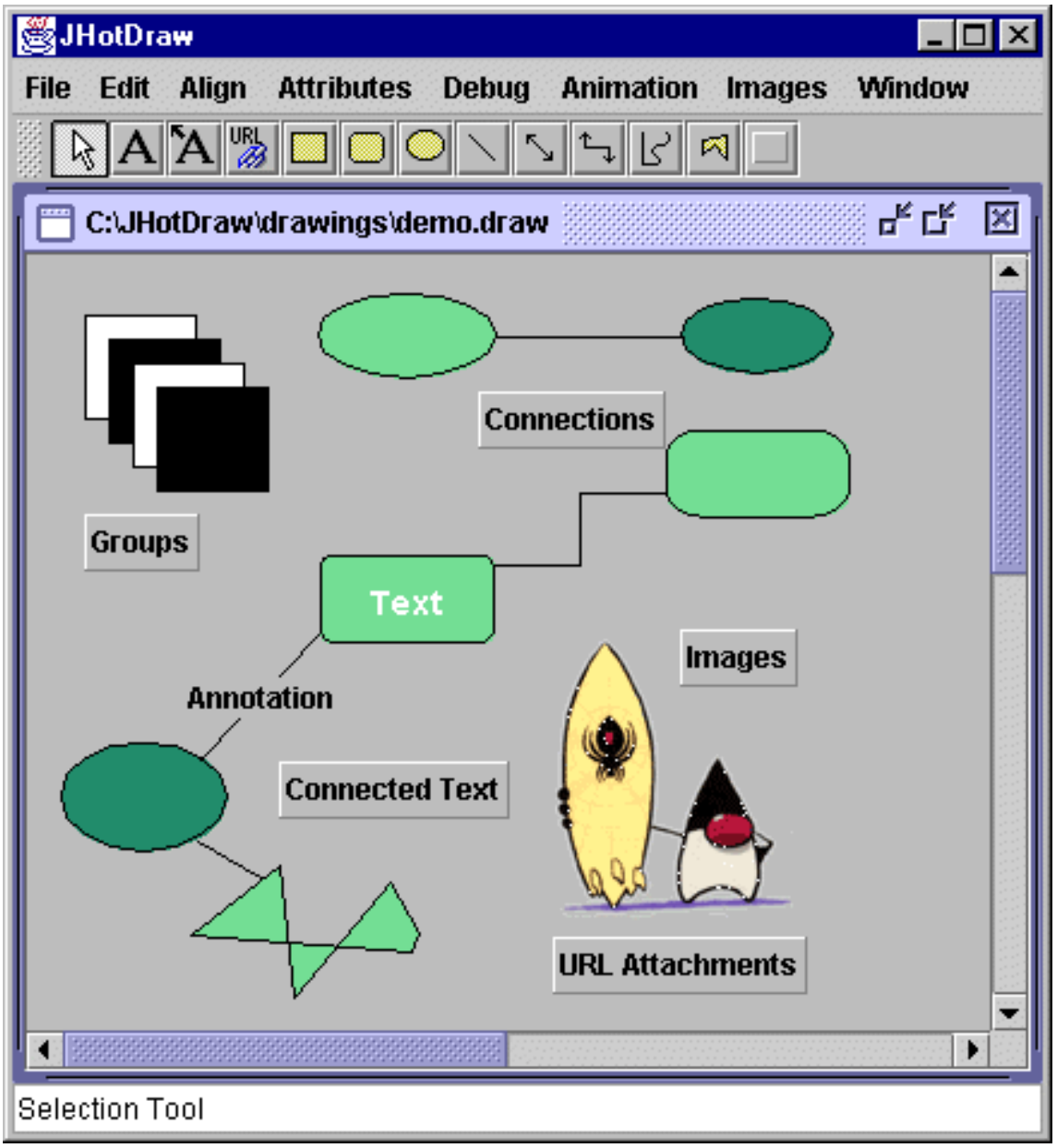


Figure 11.1: JavaDraw as a typical application of JHotDraw.

and was therefore very influential to the design pattern community. Erich Gamma and Thomas Eggenschwiler developed the first version of JHotDraw.

In JHotDraw version 5.2, the original AWT components have been replaced by their JFC Swing counterparts. It also supports new JFC Swing features like windows with several internal frames, split panes, scrollbars, toolbars, and pop-up menus. Therefore, JHotDraw (as an application-specific GUI framework) is based on the general-purpose GUI facilities that the Java Swing technology offers, but it adds its own features and functionalities. This report refers to this version.

11.2 Package Organization

All JHotDraw classes and interfaces are organized in packages according to their functionality. The package `CH.ifa.draw.framework` contains mostly interface definitions of core component requirements - their responsibility, functionality, and interoperation. You can find a standard implementation of these interfaces in `CH.ifa.draw.standard` and additional functionality in `CH.ifa.draw.figures` and `CH.ifa.draw.contrib`. Skeletons of an application and an applet are defined in `CH.ifa.draw.application` or `CH.ifa.draw.applet`, respectively.

11.3 Structure of JHotDraw

A more detailed look at the packages, in particular at the core framework package, reveals JHotDraw's structure and shows which role each of its components plays. (See figure 11.2 on the next page)

An application that uses JHotDraw has a window dedicated to drawing. This `DrawWindow` is the editor window and is a subclass of `javax.swing.JFrame`. It contains one or more internal frames, each associated with a drawing view. The `DrawingView`, a subclass of `javax.swing.JPanel`, is an area that can display a `Drawing` and accept user input. Changes in the `Drawing` are propagated to the `DrawingView` that is responsible for updating any graphics. The `Drawing` consists of `Figure`s, which in turn can be containers for other `Figure`s. Each `Figure` has `Handle`s, which define access points and determine how to interact with the `Figure` (e.g. how to connect the `Figure` to another `Figure`). In a `DrawingView`, you can select several figures and manipulate them. Usually, the `DrawWindow` itself has one active `Tool` from the tool palette, which operates on the `Drawing` associated with the current `DrawingView`.

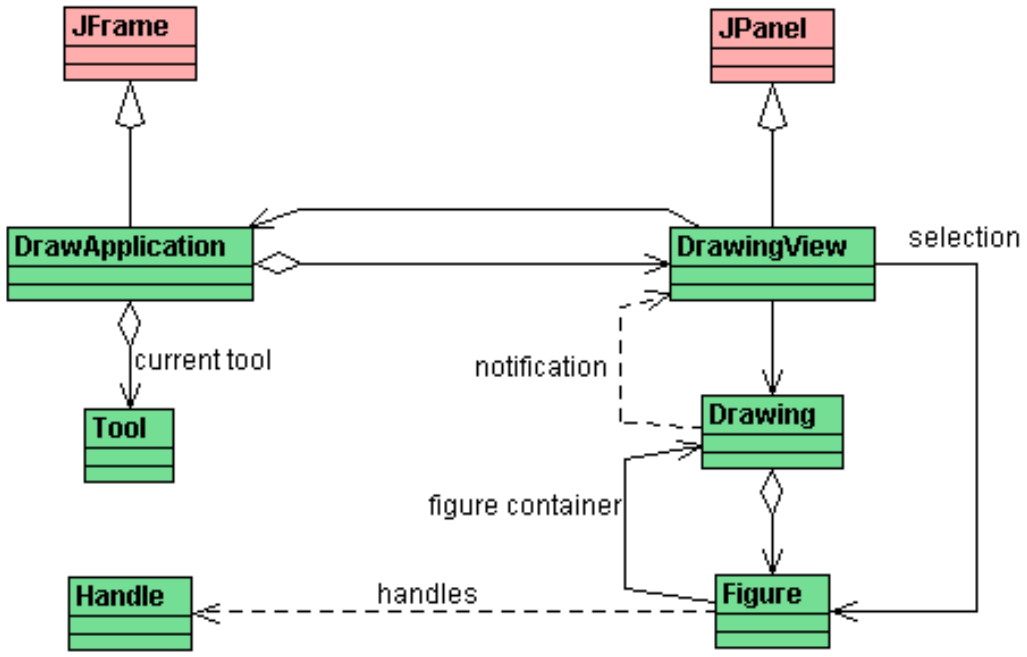


Figure 11.2: Basic components of JHotDraw's architecture.

chapter 12 Main Issues

After the conversion of all of JHotDraw's 160 classes, a total number of 696 issues were generated by the JLCA and could be seen in the Conversion Report. Of course, it would go beyond the scope of this analysis to explain every single reported issue. Furthermore, there are more issues, that have not been reported by the tool. Therefore, only those issues will be described, that could be common problems when converting large projects or had the most impact and caused the greatest workload in fixing or reprogramming.

12.1 Exceptions

Problem

As in the JHotDraw project, exception handling is essential in nearly all projects. The `StorableInput` class in the `CH.ifa.draw.util` package shall exemplify this.

Java Source Code

The relevant lines here are the `catch` blocks in line 6, 9, 11, and 13.

```
1 [...]
2     private Object makeInstance(String className) throws
         IOException {
3         try {
4             Class cl = Class.forName(className);
5             return cl.newInstance();
6         } catch (NoSuchMethodError e) {
7             throw new IOException("Class " + className
8                 + " does not seem to have a no-arg
                     constructor");
9         } catch (ClassNotFoundException e) {
```

```
10              throw new IOException("No class: " +
                    className);
11          } catch (InstantiationException e) {
12              throw new IOException("Cannot instantiate: "
                    + className);
13          } catch (IllegalAccessException e) {
14              throw new IOException("Class (" + className
                    + ") not accessible");
15          }
16      }
17  [...]
```

Converted C# Code

In general, the JLCA can handle the conversion of exceptions. However, it can happen that the JLCA converts an exception to a different type of exception. More often than not, the JLCA simply generalizes an unknown Java exception to a `System.Exception` in C#, as you can see in line 16 and 21.

The problem is that now *all* exceptions are caught, even those that were not caught before in line 16, and even if there is a specialized `catch` block for the exception that is thrown later in line 25 (e.g. `System.UnauthorizedAccessException`).

```
1  [...]
2  private System.Object makeInstance(System.String
       className)
3          {
4              try
5              {
6                  //UPGRADE_TODO: The differences in the format  of
                       parameters for method 'java.lang.Class.forName'
                        may cause compilation errors.
7                  System.Type cl = System.Type.GetType(
                       className);
8                  //UPGRADE_TODO: Method 'java.lang.Class.newInstance' was
                        converted to 'System.Activator.CreateInstance'
                       which has a different behavior.
9                  return System.Activator.CreateInstance(cl
                       );
10             }
11             catch (System.MethodAccessException e)
12             {
13                 throw new System.IO.IOException("Class "
                       + className + " does not seem to have
                       a no-arg constructor");
14             }
```

```
15                  //UPGRADE_NOTE: Exception 'java.lang.ClassNotFoundException'
                        was converted to 'System.Exception' which has different
                        behavior.
16                  catch (System.Exception e)
17                  {
18                      throw new System.IO.IOException("No class
                            : " + className);
19                  }
20                  //UPGRADE_NOTE: Exception 'java.lang.InstantiationException'
                        was converted to 'System.Exception' which has different
                        behavior.
21                  catch (System.Exception e)
22                  {
23                      throw new System.IO.IOException("Cannot
                            instantiate: " + className);
24                  }
25                  catch (System.UnauthorizedAccessException e)
26                  {
27                      throw new System.IO.IOException("Class ("
                            + className + ") not accessible");
28                  }
29              }
30  [...]
```

Analysis

There is not a single perfect solution to this problem, it always depends on the situation where it arises. For a start, you should move all specialized `catch` blocks (like the one in line 25) in front of the generalized catch blocks and after that remove any multiple `catch`es for the same exception as it is the case here (in line 16 and 21). Now, you have to consider your situation closely and decide whether self-made exceptions are an option – you have to rewrite this part of your application – or if the problem on hand is simply not that important and you can do without the specialized `catch` blocks.

12.2 Reference Paremeters

Problem

In Java, parameters with a type derived from `java.lang.Object` are passed by reference whereas simple values are passed by value. In C#, `struct`s are derived from `System.Object` but passed by value by default. As the JLCA tries to reuse

as much standard libraries as possible it maps some object types to structs. This could result in a different behavior of some methods. To overcome this, the keyword `ref` can be used and, more important, will be used by several standard libraries of C#. However, this can lead to problems.

For example, the `EllipseFigure` class in the `CH.ifa.draw.figures` package contains a parameterless constructor that calls the other constructor. This constructor has two parameters of type `Point` which should be mapped wisely onC#'s struct `Point`.

Java Source Code

The relevant line here is line 5.

```
1
2 public class EllipseFigure extends AttributeFigure {
3 [...]
4     public EllipseFigure() {
5         this(new Point(0,0), new Point(0,0));
6     }
7
8     public EllipseFigure(Point origin, Point corner) {
9         basicDisplayBox(origin,corner);
10    }
11
12  [...]
13
14    }
```

Converted C# Code

As you can see, the JLCA converts the `this(...)` call to the code in line 6. But this code is not compilable because the constructor that is called expects two reference parameters (line 15). That's why the constructor must be rewritten as the code in lines 7 to 12.

```
1     public class EllipseFigure:AttributeFigure
2     {
3         [...]
4
5         //UPGRADE_TODO: Parameter cannot be passed by reference.
6         public EllipseFigure():this(new System.Drawing.Point(0, 0), new
              System.Drawing.Point(0, 0))
7         public EllipseFigure()
8         {
```

```
9               System.Drawing.Point origin = new System.
                    Drawing.Point(0,0);
10              System.Drawing.Point corner = new System.
                    Drawing.Point(0,0);
11              basicDisplayBox(ref origin,ref corner);
12          }
13
14          //UPGRADE_NOTE: ref keyword was added to struct-type parameters.
15          public EllipseFigure(ref System.Drawing.Point
                origin, ref System.Drawing.Point corner)
16          {
17              //UPGRADE_NOTE: ref keyword was added to struct-type
                    parameters.
18              basicDisplayBox(ref origin, ref corner);
19          }
20          [...]
21
22      }
```

Analysis

There is an easy solution to the problem in this case, as there is only one method called within the constructor. However, in other cases, where more complicated operations and initializations are made within the constructor, or when the parameters are by default read only, there is not such an easy solutions and you have to determine an solution appropriate to the specific case. Possibly, problems resulting out of reference parameters can become very subtle and therefore hard to work around.

12.3 Double Names

Problem

In Java, method names and local variable names are completely separated from each other, i.e. a local variable may have the same as a method without hiding it. Contrary, in C# this is not possible, at least because methods can be referenced by name only (without a parameter list, e.g. when used as a parameter for delegates).

As obviously not being aware of this problem, the JLCA doesn't touch these names and translates them directly to the C# code. Since such a naming is not allowed in the C# language, the C# compiler issues an compiling error when encountering such an issue. The `ColorMap` class within the `CH.ifa.draw.util` package will do as an example.

Java Source Code

The relevant lines here are line 13 and 14, where the program checks if the *parameter* `color` equals the color that is returned when calling the *method* `color(String name)` with the string "None".

```
1  [...]
2
3      public static Color color(String name) {
4          for (int i = 0; i < fMap.length; i++)
5              if (fMap[i].fName.equals(name))
6                  return fMap[i].fColor;
7
8          return Color.black;
9      }
10
11 [...]
12
13     public static boolean isTransparent(Color color) {
14         return color.equals(color("None"));
15     }
16
17 [...]
```

Converted C# Code

Here you should closely examine line 16, where theoretically the same happens as in the Java context. Like already mentioned the C# compiler won't compile this file, because the name of the parameter hides the name of the method.

```
1  [...]
2
3          public static System.Drawing.Color color(System.
               String name)
4          {
5              for (int i = 0; i < fMap.Length; i++)
6                  if (fMap[i].fName.Equals(name))
7                      return fMap[i].fColor;
8
9              return System.Drawing.Color.Black;
10         }
11
12 [...]
13
14         public static bool isTransparent(ref System.
               Drawing.Color color)
```

```
15          {
16              return color.Equals(color("None"));
17          }
18
19  [...]
```

Analysis

The solution in such a case is very easy, renaming the local variable or parameter to a unique identifier will always do. Alternatively, the method call could be fully qualified. But this would not enhance readibility.

12.4 Interfaces

12.4.1 Image Observer

Problem

As an example of problems that arise when trying to map standard libraries on each other.

The interface `ImageObserver` is one of Java's standard library interfaces for which no direct equivalent in the .NET environment exists. It provides for information about an image that was previously requested using an asynchronous mechanism. A closer look to the `DrawingView` interface of the `CH.ifa.draw.framework` package will show what happens if reference to such an interface occurs.

Java Source Code

The relevant line here is line 1.

```
1  public interface DrawingView extends ImageObserver,
       DrawingChangeListener {
2      [...]
3  }
```

Converted C# Code

As you can see in line 2 the JLCA does not convert this interface and issues a warning.

```
1 //UPGRADE_ISSUE: Interface 'java.awt.image.ImageObserver' was not converted.
2     public interface DrawingView:ImageObserver,
          DrawingChangeListener
3     {
4         [...]
5     }
```

Analysis

In this special case there is not much of a problem since there is no other place in the JHotDraw project where the `Image Observer` interface is referenced. Therefore you can simply delete the interface and the application will run as intended.

However, in most cases, this is no solution. You then have to rebuild your project or to create such interfaces and to implement the appropriate mechanisms. One example in the selected project (and in many other applications using the Java Swing technology) is the `EventListener` class. Although this class is no interface but a "real" class, it is mainly used as an interface in the JHotDraw project and therefore provides a good example.

12.4.2 Event Listener

In Java, this class must be derived to create new event listeners. When extending this class, you have to specify that you are creating an event-listening interface to listen to your own custom events. In the .NET Framework, there is no direct equivalent. In contrast to the problem described above, the interface in this case is used in different places.

Java Source Code

This code is part of the `FigureChangeListener` interface within the `CH.ifa.draw.framework` package. The relevant line here is line 1.

```
1 public interface FigureChangeListener extends
      EventListener {
2     [...]
3 }
```

Converted C# Code

As you can see in line 2 the JLCA does not convert this interface and issues a warning.

```
1 public interface FigureChangeListener:EventListener
2     {
3         [...]
4     }
```

Analysis

The problem here is that the `EventListener` class is not only used as a base class in this interface (and therefore provides the interface of the `EventListener` class to the `FigureChangeListener` interface), but also in the `FigureChangeEventMulticaster` class in the `CH.ifa.draw.standard` package (see section 12.5 for the source code of this class). In this case, simply deleting the interface would solve one problem (the compilation error for the `FigureChangeListener` class), but the consructor and some methods of the `AWTEventMulticaster` class would still need parameters with the interface of the `EventListener` class. In cases such as this one, either you have to rewrite a big part of your application or to create an EventListener like interface manually. This interface should be placed in a project or namespace that all parts of your application can access.

12.5 AWTEventMulticaster

Problem

This issue stands for the many issues that arise when using the AWT package within an Java Swing application. In Java, this class implements efficient and thread-safe multi-cast event-dispatching for the AWT events defined in the `java.awt.event` package. It manages an immutable structure consisting of a chain of event listeners and dispatches events to those listeners. Since the structure is immutable, it is safe to use this class to add or remove listeners during an event dispatch operation. In the .NET Framework, there is no direct equivalent. All delegate classes derive from the `System.MulticastDelegate` class.

Java Source Code

The relevant lines here are line 1 and 4.

```
1 public class FigureChangeEventMulticaster extends
```

```
2     AWTEventMulticaster implements FigureChangeListener {
3
4     public FigureChangeEventMulticaster(EventListener a,
         EventListener b) {
5         super(a, b);
6     }
7
8     [...]
9
10 }
```

Converted C# Code

As you can see in line 2 the JLCA does not convert this interface and issues a warning. To give an example of the problems that arise with the "missing" class you should take a look at line 5, where a call to the non-existing base class is made.

```
1  public class FigureChangeEventMulticaster:
      AWTEventMulticaster, FigureChangeListener
2  {
3           //UPGRADE_ISSUE: Constructor 'java.awt.AWTEventMulticaster.
               AWTEventMulticaster' was not converted.
4           //UPGRADE_ISSUE: Interface 'java.util.EventListener' was not
               converted.
5           public FigureChangeEventMulticaster(EventListener
               a, EventListener b):base(a, b)
6           {
7           }
8
9           [...]
10
11 }
```

Analysis

One solution of this problem is a clear, but not very easy or fast one. You have to manually rebuild the `AWTEventMulticaster` class in C Sharp as a *Support Class* where you add handlers (listeners) to the first delegate, using the `System.Delegate.CombineImpl` method. It is not a big problem to create this class. However, as in most cases when there are issues with the AWT part of an application, you need some time to implement the missing classes or features.

Summary 13 chapter

After the conversion of the JHotDraw project and another, smaller project that cannot be presented here, there are several statements to give.

First and foremost, the conclusion is that the JLCA together with the Extensibility Kit is a powerful tool that can do a lot of typing work for you if you want to convert a project from Java to C#. However, there is no way that this tool will do *all* the work for you as the execution environments differ too much. Generally, the more complicated your application is and the more sophisticated concepts it uses, the more work goes to your hands.

This is especially true for Java Swing applications. Although it is said that the "JLCA can effortlessly convert Swing applications to C#" (and this statement is somewhat true as it is capable of translating simple Swing applications), you will hardly have no problems with the AWT part of your application, since Swing is based on AWT and the assistant is not in any way capable of converting AWT technology.

Thus, the more complicated your project gets and therefore the more AWT components you use (like the `AWTEventMulticaster` for example) or the more you use constructs like the `Event Listener` interface that don't exist in C#, the more problems you will have with the conversion.

Furthermore, it is noticeable that well written Java Code with well thought naming conventions is much easier to convert.For example the JLCA does not have the ability to detect if there are already classes with the same name when he automatically creates some needed classes, for example when he converts anonymous classes.

That beeing said, although the JHotDraw project is a somewhat complicated example of an Java Swing applications it was possible to solve nearly all of the generated issues of the conversion and to get it running as a C# project. The impression is that this method of first converting and then fixing the issues was still a lot faster than a complete new build in C#.

ISBN 3-937786-10-4
ISSN 1613-5652