1 C# and the .NET Framework

C# (pronounced: *see sharp*) is a programming language developed by Microsoft for the .NET platform. Although .NET programs can be written in many different languages (including C++, Visual Basic, Java, Fortran, Cobol and Eiffel), Microsoft developed this new "in-house" language in order to exploit the full potential of .NET. C# is an object-oriented language that looks like Java at first sight but goes beyond Java in its capabilities. It has all the features required to develop applications making full use of the latest in software technology.

C# is not a revolutionary language. It is more a combination of Java, C++ and Visual Basic. The aim has been to adopt the best features of each of these languages while avoiding more complex features. C# has been developed in a careful and controlled way by a small team lead by *Anders Hejlsberg* ([HWG04], [C#Std]). Hejlsberg is an experienced language expert. At Borland he was the chief designer of Delphi. He has a reputation for designing his languages with the needs of practitioners in mind.

This chapter is intended as a summary of the main features of C#. Because C# is so similar to Java, we have decided to compare the features of the two languages, assuming that the reader is already familiar with a language like Java or C++. We briefly discuss .NET at the end of the this chapter, because any C# developer needs to know the basic concepts of .NET.

1.1 Similarities between C# and Java

At first sight, C# programs look much like Java programs. Any Java programmer should be able to read them. As well as having almost identical syntax, the following concepts have been carried across from Java:

- *Object-orientation.* Like Java, C# is an object-oriented language with single inheritance. Classes can inherit from just one base class but can implement several interfaces.
- *Type safety*. C# is a type-safe language. Programming errors that arise from incompatible types in statements and expressions are trapped by the compiler. In applications, arbitrary pointer arithmetic or unchecked type casts

are not allowed as they are in C++. At run time there are checks to ensure that array indices lie in the appropriate range, that objects are not referenced via uninitialized pointers and that a type cast leads to a well-defined result.

- *Garbage collection*. Dynamically allocated objects are not released by the programmer, but are automatically disposed of by a garbage collector as soon as they are no longer referenced. This eliminates many awkward errors that can occur, for example, in C++ programs.
- *Namespaces*. What Java calls packages, C# calls namespaces. A namespace is a collection of type declarations. It allows the same names to be used in different contexts.
- *Threads*. C# supports lightweight parallel processes in the form of threads. As in Java, there are mechanisms for synchronization and communication between threads.
- *Reflection*. As in Java, type information about a program can be accessed at run time, classes can be loaded dynamically, and it is even possible to compose object programs at run time.
- *Libraries*. Many types in the C# library resemble those in the Java library. There are familiar classes such as Object, String, ICollection and Stream, often even with the same methods as in Java.

Various features are also taken from C++, for example operator overloading, pointer arithmetic in system-level classes (which must be marked as *unsafe*) as well as some syntactical details, for example in connection with inheritance. The foreach loop is an example of a feature taken from Visual Basic.

1.2 Differences between C# and Java

Beside these similarities, C# has several characteristics that go beyond Java. Most of them also apply to the other .NET languages:

- *Reference parameters*. Parameters can be passed not only *by value*, as in Java, but also *by reference*. This means that input parameters can be used as well as output and transient parameters.
- *Objects on the stack*. Whereas in Java all objects are kept on the heap, in C# an object can also be stored on the method call stack. Such objects are lightweight, that is, they make no demands of the garbage collector.
- *Block matrices*. The Java storage model for multidimensional arrays is not efficient enough for numerical applications. C# allows the programmer to choose whether to have a matrix laid out as in Java (that is, as an array of arrays) or as a compact block matrix, as in C, Fortran or Pascal.
- *Enumerations*. As in Pascal or C, there are enumeration types with values denoted by names.

- *goto statement*. The much-maligned goto statement has been reintroduced in C#, but with restrictions that make it scarcely possible to misuse it.
- *Uniform type system*. In C#, all types are derived from the object type. In contrast to Java, numbers or character values can also be stored in object variables. The C# mechanism for this is called *boxing*.
- *Attributes*. The programmer can attach metadata to classes, methods or fields. This information can be accessed at run time by means of reflection. .NET uses this mechanism, for example, for serializing data structures.
- *Versioning*. Libraries are given a version number during compilation. This means that a library can be available in several versions at the same time. Each application uses the version of the library used for compilation and testing.

Finally there are many features of C# that are convenient to use, although they do not really increase the power of the language. They can be described as "*syntactic sugar*", because things that can be done in other languages are more straightforward and elegant in C#. They include the following:

- *Properties* and *events*. These features facilitate component technology. Properties are special fields of an object. When they are accessed the system automatically calls getter and setter methods. Events can be declared and triggered by components and handled by other components.
- *Indexers*. As with arrays, an index operator can be declared for custom collections via getter and setter methods.
- *Delegates*. Delegates are essentially the same as *procedure variables* in Pascal and *function pointers* in C. However, they are more powerful. For example, several methods can be stored in a delegate variable at the same time.
- *foreach loop*. This loop statement is a convenient way of iterating through arrays, lists or sets.
- *Boxing/unboxing*. Values such as numbers or characters can be assigned to variables of type object. To do this they are automatically wrapped into an auxiliary object (*boxing*). When they are assigned to a number or character variable, they are automatically unwrapped again (*unboxing*). This feature allows the construction of generic container types.

1.3 The .NET Framework

Anyone programming in C# will eventually need to get to grips with the .NET Framework, the Windows component for which C# was specifically developed. It is a kind of layer on top of Windows (and perhaps other operating systems in the future, see Fig. 1.1), with two main components:

- A **run-time environment** (the *common language runtime*), providing automatic *garbage collection*, security, versioning and, above all, interoperability between programs written in different languages.
- An **object-oriented class library**, providing a rich set of functionality for graphical user interfaces (*Windows forms*), web interfaces (*ASP.NET*), database connectivity (*ADO.NET*), collections, threads, reflection and much more. In many cases it replaces the current Windows API and goes beyond it.

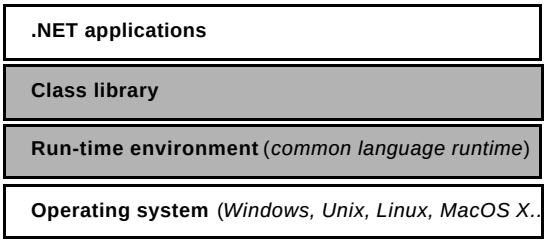

Fig. 1.1 *Outline of the .NET Framework architecture*

Although .NET was developed by Microsoft, it is based on open standards. For example, the ECMA-335 standard [CLI] defines the common language runtime and parts of the class library, the ECMA-334 standard [C#Std] describes the C# language, and general standards such as SOAP, WSDL or UDDI are used for web services. As part of an open source project ([Mono]), the .NET Framework is currently being ported to Linux, and even Microsoft is publishing large parts of the CLR source code via [SSCLI] ([SNS03]).

This section summarizes the main components of the .NET Framework. For a more detailed description, please refer to [MBBW04], [Rich02] or [Rob01]. Parts of the class library are described in Chapter 18.

Common Language Runtime

The common language runtime (CLR) is the run-time environment under which .NET programs are executed, supporting features like garbage collection, security and interoperability [MR04].

Like the Java environment, the CLR is based on a *virtual machine*, with its own instruction set (CIL *common intermediate language*) into which programs written in all .NET languages are translated. Just before they are run (*just in time*) CIL programs are converted into the native machine language (e.g. Intel code) (see Fig. 1.2). The CIL code guarantees interoperability between different languages as well as code portability, while JIT compilation (*just-in-time compilation*), ensures that programs are executed efficiently.

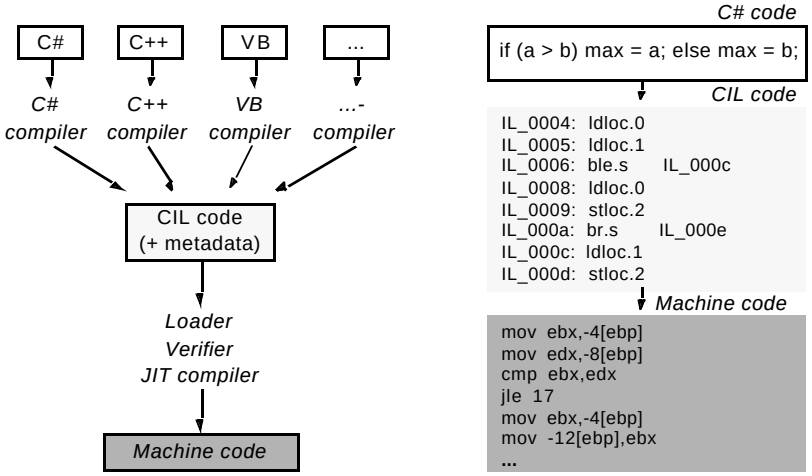

Fig. 1.2 *Source code, CIL code and machine code*

Unfortunately, though, for different languages to be able to cooperate it is not enough for them just to be translated into CIL. They must also use the same sort of data types. This is why the CLR defines a *common type system* (CTS) defining how classes, interfaces and other types are represented. The CTS not only allows a class implemented in C# to be used in a Visual Basic program, but it even allows the C# class to be extended by a subclass written in Visual Basic. Similarly, an *exception* raised by a C# program can be handled by a program written in any other .NET language.

The CLR offers mechanisms for making .NET programs safer and more robust. These include the *garbage collector*, which is responsible for reclaiming the space used by objects once they are no longer needed. In older languages such as C and C++, it was up to the programmer to release object space. This meant that the space was sometimes released by mistake when it was still in use by other objects. This left the other objects with nowhere to go, potentially destroying unrelated storage areas. Similarly, a programmer could forget to release an object even though it was no longer being referenced. This then remained in the memory as a *memory leak*, wasting space. Such errors are hard to identify and locate, but thanks to the garbage collector, they cannot occur in .NET.

When a program is loaded and compiled into machine code, the CLR uses a *verifier* to check that the type rules of the CTS have not been violated. For example, it is illegal to treat a number as an address and use it to access storage areas that belong to other programs.

Assemblies

.NET supports component-based software development. The components are called *assemblies* and are the smallest units that can be individually deployed. An assembly is a collection of classes and other resources (for example, images). It is stored either as an executable EXE file or as a DLL file (*dynamic link library*) (see Fig. 1.3). In some cases an assembly is even made up of multiple files.

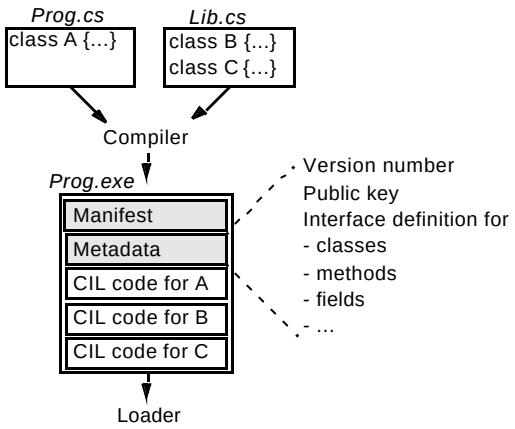

Fig. 1.3 *Prog.exe assembly generated by compiler*

Each assembly also contains *metadata* in addition to code. The metadata holds the interface definition of the classes, fields, methods and other program elements in the assembly. An assembly also contains a *manifest*, which can be thought of as a table of contents. The manifest renders assemblies self-describing and can be inspected and used by loaders, compilers and other tools, by means of *reflection*.

Assemblies are also used for version control. Each has a multi-level version number that also applies to all the classes within the assembly. When a class is compiled, the version numbers of classes referenced from other assemblies are recorded in its object code. The loader then asks for those classes (i.e. assemblies) that correspond to the expected version numbers. In .NET, several DLLs with the same name but different version numbers can coexist without conflicting with each other (*side-by-side execution*). This spells the end of the "DLL hell" in Windows, where the installation of new software could cause old DLLs to be overwritten by new ones with the same names, causing the existing software to suddenly stop working.

Another advantage is that assemblies no longer have to be recorded in the Windows registry. They are simply copied into the application directory or into the so-called *global assembly cache*, and they are equally easy to remove when they are no longer needed.

Assemblies are effectively the successors of COM components. Unlike COM objects (*component object model*), assemblies do not need to be described by an IDL (*interface definition language*) because they contain comprehensive metadata that has been gathered by the compiler from the source code. The common type system guarantees that software written in different .NET languages uses the same sort of metadata and is thus binary compatible. However, investment in COM components is not lost. It is still possible to use COM components from .NET classes and vice versa (see Chapter 17).

ADO.NET

ADO.NET comprises all the classes of the .NET library that are concerned with accessing databases and other data sources (such as XML files). It has a predecessor technology called ADO (*ActiveX Data Objects*), with which it only shares a name. ADO.NET is object-oriented and therefore more structured and straightforward to use

ADO.NET supports the relational data model, with transactions and locking mechanisms. Therefore it is independent of different data providers and database architectures. The differences between concrete data sources like MS SQL Server, OLE DB (*Object Linking and Embedding Database*) and ODBC (*Open Database Connectivity*) are abstracted away with common interfaces.

Database access can be either connection-oriented or connectionless. In connection-oriented access, a permanent connection to a data source is established. In connectionless access, a snapshot of a part of the database is fetched into a DataSet object and then processed locally. In both cases, SQL statements (*Structured Query Language*) can normally be used to access the data.

ASP.NET

ASP.NET is the part of the .NET technology that deals with programming dynamic web pages. Its name is reminiscent of the predecessor technology ASP (*Active Server Pages*). However, the programming model is fundamentally different.

With ASP.NET, web pages are constructed dynamically on the server from current data and are sent to the client in the form of pure HTML, so that any web browser can display them. In contrast to ASP, ASP.NET uses an object-oriented model. Web pages, as well as the controls that appear in them, are objects whose fields and methods can be accessed in programs. All this is done in a compiled language such as C# or Visual Basic .NET and not, as in ASP, in an interpreted language such as JavaScript or VBScript. This means that web pages can take advantage of the entire class library of .NET.

User input is handled in an event-driven way. When a user fills out a text field, clicks a button or selects an item from a list, this raises an event that can be handled by code on the server side. Although the server is stateless—as is usual for the Internet—state information is retained automatically between individual user interactions, in fact in the HTML code itself. This represents a considerable simplification over the former programming model, where it was the programmer who was responsible for maintaining state information.

ASP.NET offers a rich library of controls that go far beyond what is supported by HTML, although all controls are eventually translated to HTML. Programmers can even build their own controls and thus adapt the user interface of their web pages to their particular needs. It is particularly straightforward to display the results of database queries as lists and tables, since ASP.NET has largely automated this. Validators are a further new feature. They allow user input to be checked for validity.

The Visual Studio .NET development environment allows the user interface of a web page to be built interactively, in a way familiar from the development of desktop applications. Controls can be dragged into windows with the mouse. Values of properties can be assigned using menus and property windows, and methods can be specified that will be called in response to user input. All this sweeps away the difference between programming desktop applications and web applications and simplifies the development of online stores and pages that show frequently updated information (for example, stock data). ASP.NET is explained in more detail in Section 19.3.

Web services

Web services are regarded as one of the core features of .NET technology, although they also exist outside .NET. They work via *remote procedure calls* using protocols such as HTTP and SOAP (an application of XML).

The Internet has proved itself to be tremendously powerful for accessing information and services distributed around the world. Currently, access is mainly through web browsers such as Internet Explorer or Netscape Navigator. Web services, on the other hand, allow a new style of cooperation between applications by making them communicate without web browsers. Ordinary desktop applications can fetch information such as current exchange rates or booking data from one or more web services that are running as methods of applications on other computers and which respond over the Internet.

The calls and their parameters are generally coded to conform to SOAP [SOAP], an XML-based standard that is supported by most large firms. Programmers need to know nothing of this. They call a web service in just the same way as a normal method and .NET takes care of translating the call into SOAP, sending it over the Internet and decoding it on the target machine. On the target machine,

the chosen method is invoked and its result is transmitted back to the caller transparently, again using SOAP. The caller and the callee can therefore be written in quite different languages and can run under different operating systems.

In order for .NET to be able to carry out the coding and decoding correctly, the web services, together with their parameters, are described in WSDL (*Web Services Description Language* [WSDL]). This is also done automatically by .NET. Web services are described further in Section 19.2.

1.4 Exercises[1]

1. **Suitability of C# for large-scale software projects.** To what extent do the features of C# help developers of large-scale software projects?
2. **Features of .NET.** What are the main features of the .NET Framework? Which of them resemble the Java environment, and which are new?
3. **Security**. Give reasons why C# is a safe language. Name the kind of programming errors or hazardous situations that would be trapped by the C# compiler or the CLR?
4. **Interoperability.** What makes .NET a platform on which programs written in different languages can cooperate seamlessly?
5. **Assemblies.** What makes .NET assemblies easier to install and de-install than COM objects?
6. **Internet resources**. Visit these web pages [MS], [MSDN], [GotD] and [Dev] for an overview of the .NET Framework and C#.
7. **Mono**. Visit the [Mono] web site to find out more about the project porting .NET to Linux.

1. Example solutions to the exercises can be found in [JKU].