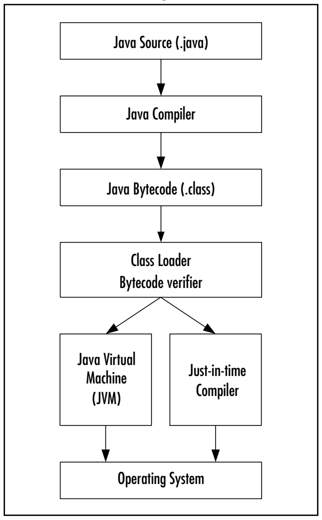
Contents

Java Code Cycle



Chapter 1 The .NET Philosophy 1
Introduction 2
Overview of the .NET Platform 2
Examining the .NET Framework Features 4
  Multilanguage Development 5
  Platform and Processor Independence 6
  Automatic Memory Management 7
  Versioning Support 7
  Support for Open Standards 8
  Easy Deployment 9
  Interoperability with Unmanaged Code 10
  Providing Security 11
Understanding the .NET Architecture 13
  The Common Language Runtime 14
  The .NET Framework Class Library 15
  The Microsoft Intermediate Language (MSIL) 17
  Just-In-Time Compilation 17
Following .NET Code from Source to Binary 18
Summary 21
Solutions Fast Track 22
Frequently Asked Questions 24

Chapter 2 Introducing C# 27
Introduction 28
The C# Language 28
  Similarities with Java 29
  Differences with Java 30
Getting Started 32
  Installing the .NET Framework SDK 32

Creating Your First C# Program 33
Compiling and Running the Program 34
Using the Debugger Tool 37
Using Different IDEs 40
Visual Studio.NET 41
Other IDEs 44
A Stroll through C# 45
Creating the Media Player Application 45
Rapid Application Development with Visual Studio.NET 54
Summary 60
Solutions Fast Track 60
Frequently Asked Questions 62

Chapter 3 Language Fundamentals 63

Introduction 64
Main() Method 64
Command Line Arguments 66
Return Values 69
Single-Line and Multiline Comments 72
XML Documentation Comments 74
Data Types and the Common Type System 82
Variables 85
Constants 87
Assignment Statements 87
Conversions between Data Types 90
Operators 92
Mathematical Operators 92
Assignment Operators 93
Increment and Decrement Operators 95
Relational Operators 96
Logical Operators 97
Bitwise Operators 98
Ternary Operator 99
Operator Precedence 100
Preprocessor Directives 100
#define and #undef 102

Mathematical Operators

Operator	Definition
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus

#if, #elif, #else, and #endif 102
#error and #warning 106
#region and #endregion 107
#line 108
Namespaces 110
Summary 112
Solutions Fast Track 112
Frequently Asked Questions 115

Chapter 4 Programming Structures 117

Introduction 118
Strings 118
The WriteLine Method 118
Creating Strings 119
Verbatim String Literal 123
The *StringBuilder* Class 123
Using Regular Expressions 126
Flow Control 129
Branch Statements 129
The *if/else* Statement 129
The *switch* Statement 130
Iteration Statements 132
The *while* Loop 133
The *do-while* Loop 133
The *for* Loop 133
Jump Statements 134
The *goto* Statement 134
The *break* Statement 135
The *continue* Statement 135
The *return* Statement 136
Arrays 137
Declaring and Initializing Arrays 137
Using the *params* Keyword 140
Multidimensional Arrays 141
Rectangular Arrays 141
Jagged Arrays 144

Arrays

- Standard C# arrays are identical to their Java counterparts.
- C# provides two different kinds of multidimensional arrays, rectangular and jagged.
- A rectangular array has equal dimensions, a jagged array does not.
- The *params* keyword can be used to specify that an array of unknown dimensions will be passed to a method.

The foreach Statement 147
Indexers 150
Implementing an Indexer 151
Multiple Indexers 155
Multiparameter Indexers 157
Collections 159
Collection Interfaces 163
Exceptions 164
Catching Exceptions 164
The *try*, *catch*, *finally* Blocks 164
Throwing Exceptions 167
Creating New Exceptions 170
Rethrowing Exceptions 173
Summary 175
Solutions Fast Track 175
Frequently Asked Questions 178

Chapter 5 Objects and Classes 179
Introduction 180
Using Classes 180
Access Control 181
Class Modifiers 182
Abstract Classes 186
Sealed Classes 187
Instance Variables 187
Static Variables 189
Constants as Static Members 191
Using Methods 191
Access Modifiers 194
Method Parameters 195
The *ref* and *out* Method Parameters 197
Overloaded Methods 199
Creating Objects 202
Constructors 202
Overloading Constructors 204
Static Constructors 206
Destroying Objects 208

NOTE

Only nested classes permit the use of the *new* keyword. The *new* modifier specifies that the class hides an inherited member by the same name. Inner classes and Inheritance will be discussed in Chapter 6.

Memory Management and Garbage Collection 208
C#'s Destructor 209
The *Finalize*, *Dispose*, and *Close* Methods 210
The *using* Statement 211
Summary 214
Solutions Fast Track 214
Frequently Asked Questions 215

Chapter 6 Object-Oriented Programming 217
Introduction 218
Inheritance 218
Defining a Base Class 221
Calling Base Class Constructors 222
Polymorphism 226
Abstract Classes 227
The *abstract* Modifier 232
The *virtual* Modifier 234
The *override* Modifier 237
The *new* Modifier 241
Inner Classes 244
Using Interfaces 246
Creating an Interface 247
Declaring Interfaces 248
Implementing Interfaces 249
The *is* Operator 253
The *as* Operator 258
Explicit Interface Implementation 258
Implementation Hiding 260
Summary 262
Solutions Fast Track 263
Frequently Asked Questions 264

Chapter 7 Other C# Features 267
Introduction 268
Properties 268
Read-Only Fields 271

Frequently Asked Questions

Q: Does C# support multiple inheritance?

A: Yes and no—just like Java, C# allows single inheritance of classes and multiple inheritance of interfaces.

Q: Does Java support inner classes?

A: Yes. C# supports only one kind of inner class compared to Java's four.

Unboxing

Unboxing is the act of converting an object back into a value type. The syntax for this process looks very similar to explicit casting in Java, as the following C# code demonstrates:

```
int x = 29;

object xObj = x; //
Boxing

int x1 = (int)xObj;
// Unboxing
```

Delegates

- Delegates are similar to C/C++ function pointers.
- Delegates reference a method.
- Delegates are object-oriented, type-safe, and secure.

Enumerations 272
Boxing and Unboxing 276
    Boxing 277
    Unboxing 278
Operator Overloading 279
    Unary Operator Overloading 283
    Binary Operator Overloading 287
    Equals() Method 287
User-Defined Conversions 288
    The *implicit* Operator 289
    The *explicit* Operator 291
Structs 293
    Defining Structs 294
    Using Structs 297
Summary 301
Solutions Fast Track 301
Frequently Asked Questions 304

Chapter 8 Delegates and Events 305

Introduction 306
Delegates 306
Using Delegates as Callbacks 307
    Declaration 308
    Instantiation 309
    Invocation 311
    Implementing Callbacks 312
Using Delegates for Event Handling 316
    Event Handling in Java 317
    Event Handling in C# 319
        Using System.EventArgs 320
        Creating and Handling Events 321
Multicasting 328
        Order of Operations in Multicasting 331
Advanced Delegate Usage 331
    Declaring Delegates as Static Members 332
    Delegates and Thread Creation 333

Summary 335
Solutions Fast Track 335
Frequently Asked Questions 337

Chapter 9 Attributes and Assemblies 339
Introduction 340
Working with Attributes 340
Using Attributes 341
Creating Custom Attributes 346
Defining the *AttributeUsage* Attribute 347
Declaring an Attribute Class 348
Declaring Attribute Class Constructors and Properties 349
Using Custom Attributes 351
Using Reflection 352
Creating Assemblies 360
Manifest Data 361
Building Assemblies 361
Multiple Module Assembly 362
Versioning 364
Creating Versioned Assemblies 365
Summary 369
Solutions Fast Track 370
Frequently Asked Questions 371

Chapter 10 Multithreading 373
Introduction 374
Threads 374
Creating Threads 375
Managing Threads 380
Using *Suspend()/Resume()* and *Abort()* 382
Scheduling Threads 384
Synchronizing Threads 389
Using the *lock* Statement 389
Using the Monitor Class 391
Avoiding Deadlock and Starvation 395

Creating Assemblies

- Assemblies are the C# equivalent to Java's packages and are used to segment namespaces.
- Assemblies in the .NET architecture can be written and compiled in different languages, and still work together.
- All information about an assembly is stored in the assembly manifest.

Developing & Deploying…

Monitor.Wait() Parameters

The *Wait()* method can take on a variety of parameters, including an integer specifying the number of milliseconds to wait as well as a *TimeSpan* structure. In the event that the specified time expires before it is notified by a corresponding *Pulse()*, *Wait()* returns a *boolean* value of *false*.

Summary	401
Solutions Fast Track	401
Frequently Asked Questions	403
Chapter 11 Working with I/O Streams	**405**
Introduction	406
File System	406
Directories	406
Files	411
Streams	415
Stream	416
FileStream	418
MemoryStream	419
BufferedStream	423
Encoding Data Types	423
Text	428
StreamReader and *StreamWriter*	429
StringReader and *StringWriter*	430
Network I/O	431
Server Side	432
Client Side	435
Synchronous vs. Asynchronous	438
Web Streams	441
Serialization	443
Creating a Serializable Object	443
Serializing an Object	444
Deserializing an Object	445
Transient Data	446
Deserialization Operations	446
Summary	450
Solutions Fast Track	451
Frequently Asked Questions	453
Chapter 12 Creating User Interfaces with Windows Forms	**455**
Introduction	456
Windows Form Classes	456
Windows Form Class Hierarchy	457

Debugging…

The Directory Separator

One of the most frequent bugs when programming with the file system is the backslash used to identify directory structures. Notice the need to use two backslashes in the preceding example. This is because the backslash is an escape character, so it is necessary to nullify the first by using two backslashes. An even better solution is to indicate a verbatim string literal by placing the @ symbol in front of the string, as follows:

```
String filename =
@"c:\Program Files";
```

Financial Calculator

Creating a Windows Form Application 458
Using Controls 460
  Adding Controls 461
  Basic Controls 462
    Buttons 462
    Textboxes 464
    Labels 464
Handling Events 465
Using a Text Editor 467
Using Visual Studio.NET 470
  Creating a File Browser 473
Summary 476
Solutions Fast Track 476
Frequently Asked Questions 478

Chapter 13 Web Development with C# 479

Introduction 480
Web Services Overview 480
  Using SOAP 481
Creating Web Services 482
  Connecting to a Database 483
  Building a Web Service 486
  Running and Testing Your Web Service 489
  Creating the Books Web Service Using VS.NET 491
Consuming Web Services 495
  Web Service Description Language 496
  Creating Proxy Objects 498
    Using the *wsdl.exe* Utility 498
    Using Visual Studio.NET 499
Web Forms Overview 503
  Differences between HTML and Web Controls 504
    Understanding HTML Controls 505
    Understanding Web Controls 507
Using Web Form Controls 507
  HTML Page Access Web Controls 509

Creating Proxy Objects

To interact with a Web service you will need to create a proxy object that will act as the middleman between your application and the service. The proxy object can be generated from the WSDL file in two ways:

- Using the *wsdl.exe* command line utility
- Using Visual Studio.NET

Data Enabling Controls 509
DataGrid 510
DropDownList 510
Validation Control 511
Required Field Validator 511
RegularExpressions Validator 512
Complex Web Controls 513
Using the AdRotator Control 513
Using the Calendar Control 514
Creating a Web Form Application 515
Summary 522
Solutions Fast Track 522
Frequently Asked Questions 524

Chapter 14 Working with ActiveX, COM, and Unmanaged Code 527

Introduction 528
Working with Unmanaged Code 529
Interoperability with Unmanaged Code 530
Managed Code Calling an Unmanaged COM DLL Function 531
Managed Code Calling an Unmanaged COM Object or an ActiveX Control 531
Unmanaged COM DLL Calling Managed .NET Code 531
Working with the Platform Invocation Utility 532
Working with COM Components 535
Creating a Simple COM Component 536
Runtime Callable Wrappers 539
Creating a Runtime Callable Wrapper for a COM Component 541
Building a Client for the RCW 544
Examining Runtime Callable Wrapper Properties 546
Using Late Binding RCWs 548
Limitations of Using RCWs 551

Unmanaged Code

The interoperability services in .NET could be categorized into the following scenarios:

- .NET assembly (managed) calling a single COM DLL (unmanaged)
- .NET assembly (managed) calling a COM object or an ActiveX control (unmanaged)
- COM DLL (unmanaged) calling a .NET assembly (managed)

Working with ActiveX Controls 552
Differences between ActiveX Controls and Windows Forms Controls 552
Using the ActiveX Control Importer Utility (*AxImp.exe*) 553
Using Visual Studio .NET to Import ActiveX Controls 555
Working with Pointers 559
Unsafe Code 559
The *unsafe* Keyword 560
The *fixed* Keyword 561
Summary 564
Solutions Fast Track 564
Frequently Asked Questions 566

Chapter 15 Microsoft Says JUMP—Java User Migration Path 569
Introduction 570
What Is J#? 571
Features of Visual J# 572
Using Visual J# 573
Creating a Simple Visual J# Application 578
Summary 592
Solutions Fast Track 592
Frequently Asked Questions 593

Appendix A C# Keywords and Java Equivalents 595

Index 601

What Is J#?

J# is a complete implementation of the Java language specification. J# allows the majority of existing Java applications to run after recompilation or after binary conversion.