

Annotated Solution Guide for

# Thinking

in

# Java

Fourth Edition

TIJ4 published February, 2006

Solution Guide published September 2007

**Bruce Eckel**

President, MindView, Inc.

**Ervin Varga**

Ph.D. Computer Science,  
University of Novi Sad, Serbia;  
Faculty of Technical Sciences

©2007, MindView, Inc.  
All Rights Reserved



# Copyright & Disclaimer

**This *Annotated Solution Guide for Thinking in Java, Fourth Edition* is not freeware.** You cannot post it on any website, reproduce or distribute it, display it publicly (such as on overhead slides), or make it the basis of any derivative work. Copyrighted by MindView, Inc., this publication is sold only at *www.MindView.net*.

The Source Code in this book is provided without express or implied warranty of any kind, including any implied warranty of merchantability, fitness for a particular purpose or non-infringement. MindView, Inc. does not warrant that the operation of any program that includes the Source Code will be uninterrupted or error-free. MindView, Inc. makes no representation about the suitability of the Source Code or of any software that includes the Source Code for any purpose. The entire risk as to the quality and performance of any program that includes the Source Code is with the user of the Source Code. The user understands that the Source Code was developed for research and instructional purposes and is advised not to rely exclusively for any reason on the Source Code or any program that includes the Source Code. Should the Source Code or any resulting software prove defective, the user assumes the cost of all necessary servicing, repair, or correction.

IN NO EVENT SHALL MINDVIEW, INC., OR ITS PUBLISHER BE LIABLE TO ANY PARTY UNDER ANY LEGAL THEORY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS, OR FOR PERSONAL INJURIES, ARISING OUT OF THE USE OF THIS SOURCE CODE AND ITS DOCUMENTATION, OR ARISING OUT OF THE INABILITY TO USE ANY RESULTING PROGRAM, EVEN IF MINDVIEW, INC., OR ITS PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. MINDVIEW, INC. SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOURCE CODE AND DOCUMENTATION PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, WITHOUT ANY ACCOMPANYING SERVICES FROM MINDVIEW, INC., AND MINDVIEW, INC. HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.



# About this Document

This is the annotated solution guide for *Thinking in Java, Fourth Edition*. *Thinking in Java, Fourth Edition* is available in print from Prentice Hall and for sale electronically from [www.mindview.net](http://www.mindview.net). This solution guide is only available online as a PDF document along with the associated source code (if printed in the same format as *Thinking in Java*, this guide would be almost 900 pages).

You can download a free sample of this solution guide, up to and including the solutions for the chapters *Everything is an Object* and *Operators*, from <http://www.mindviewinc.com/Books/TIJ4/Solutions>. You should do this before buying the guide to make sure that you can properly install, compile and run the solutions on your system.

The complete version of this book including all exercises is only available electronically, for US \$25 (credit cards or PayPal), from <http://www.mindviewinc.com/Books/TIJ4/Solutions/>.

## Unpacking the Distribution

The *Annotated Solutions Guide* is distributed as a zipped file containing the book in Adobe Acrobat PDF format, along with a source-code tree in the **code.zip** file. **Please make sure your computer can unzip files before purchasing the guide.** There are free unzip utilities for virtually every platform, available by searching on the Internet.

**Linux/Unix (including Mac OSX) Users:** Unzip the file using Info-Zip (pre-installed on many Linux distributions, or available at <http://www.info-zip.org/>). Unzip the package for Linux/Unix using the **-a** option to correct for the difference between DOS and Unix newlines, like this:

```
unzip -a TIJ4-solutions.zip
```

**All Users:** This guide uses the *Ant* build system, and includes Ant **build.xml** files in each subdirectory of the code tree, which compile and run the code examples using the **javac** compiler in the Sun JDK (available at <http://java.sun.com/j2se/>). Ant, the standard build tool for Java projects, is an open-source tool. The full download, installation and configuration instructions, along with the Ant executable and documentation are available at <http://ant.apache.org/>.

Once you install and configure Ant on your computer, you can type **ant** at the command prompt of the guide's source-code root directory to build and test all the code. You can also choose to build and test the code for a particular chapter. For example, to build and test the code from the *Polymorphism* chapter, enter the **polymorphism** sub-directory and type **ant** from there.

Full detailed instructions for installation can be found after the table of contents.

## No Exercises in Chapter 1

The chapter *Introduction to Objects* has no exercises.

## Additional exercises

This guide features additional exercises not included in *Thinking in Java*, for which solutions are not provided, as a challenge to the reader.

# Contents

Copyright & Disclaimer	i
About this Document	iii
Unpacking the Distribution.....	iii
No Exercises in Chapter 1..	iv
Additional exercises.....	iv
Installing the Code	1
Using Eclipse.....	3
Packages & IDEs	5
Left to the Reader	7
Everything is an Object	9
Exercise 1.....	9
Exercise 2 .....	9
Exercise 3 .....	10
Exercise 4 .....	10
Exercise 5 .....	11
Exercise 6 .....	11
Exercise 7 .....	12
Exercise 8 .....	13
Exercise 9 .....	13
Exercise 10 .....	14
Exercise 11 .....	16
Exercise 12 .....	16
Exercise 13.....	17
Exercise 14 .....	17
Exercise 15.....	18
Exercise 16 .....	18
Operators	21
Exercise 1.....	21
Exercise 2 .....	21
Exercise 3 .....	22
Exercise 4 .....	23
Exercise 5 .....	24
Exercise 6 .....	24
Exercise 7 .....	26

Exercise 8 .....	27
Exercise 9 .....	27
Exercise 10 .....	28
Exercise 11 .....	29
Exercise 12 .....	31
Exercise 13.....	33
Exercise 14 .....	33

## **Controlling Execution 35**

Exercise 1.....	35
Exercise 2 .....	35
Exercise 3 .....	37
Exercise 4.....	38
Exercise 5 .....	39
Exercise 6 .....	41
Exercise 7 .....	42
Exercise 8 .....	43
Exercise 9 .....	45
Exercise 10 .....	46

## **Initialization & Cleanup 49**

Exercise 1.....	49
Exercise 2 .....	49
Exercise 3 .....	50
Exercise 4.....	51
Exercise 5 .....	51
Exercise 6 .....	52
Exercise 7 .....	53
Exercise 8 .....	54
Exercise 9 .....	54
Exercise 10 .....	55
Exercise 11 .....	55
Exercise 12 .....	56
Exercise 13.....	57
Exercise 14 .....	58
Exercise 15.....	58
Exercise 16 .....	59
Exercise 17.....	60
Exercise 18 .....	61
Exercise 19 .....	61
Exercise 20.....	62
Exercise 21 .....	63
Exercise 22 .....	63



<b>Access Control</b>	<b>65</b>
Exercise 1.....	65
Exercise 2 .....	65
Exercise 3 .....	66
Exercise 4 .....	67
Exercise 5 .....	68
Exercise 6 .....	70
Exercise 7 .....	70
Exercise 8.....	71
Exercise 9 .....	75
<b>Reusing Classes</b>	<b>77</b>
Exercise 1.....	77
Exercise 2 .....	78
Exercise 3 .....	79
Exercise 4 .....	80
Exercise 5 .....	81
Exercise 6 .....	81
Exercise 7 .....	82
Exercise 8 .....	83
Exercise 9 .....	84
Exercise 10 .....	85
Exercise 11 .....	86
Exercise 12 .....	87
Exercise 13.....	89
Exercise 14 .....	90
Exercise 15.....	91
Exercise 16 .....	92
Exercise 17.....	92
Exercise 18 .....	93
Exercise 19 .....	94
Exercise 20.....	95
Exercise 21 .....	96
Exercise 22 .....	96
Exercise 23 .....	97
Exercise 24.....	98
<b>Polymorphism</b>	<b>99</b>
Exercise 1.....	99
Exercise 2 .....	100
Exercise 3 .....	102
Exercise 4 .....	104
Exercise 5 .....	105
Exercise 6 .....	106

Exercise 7 .....	108
Exercise 8 .....	109
Exercise 9 .....	111
Exercise 10 .....	113
Exercise 11 .....	114
Exercise 12 .....	115
Exercise 13.....	116
Exercise 14 .....	118
Exercise 15.....	121
Exercise 16 .....	122
Exercise 17.....	123

<b>Interfaces</b>	<b>125</b>
-------------------	------------

Exercise 1.....	125
Exercise 2 .....	126
Exercise 3 .....	127
Exercise 4 .....	128
Exercise 5 .....	129
Exercise 6 .....	130
Exercise 7 .....	130
Exercise 8 .....	132
Exercise 9 .....	133
Exercise 10 .....	135
Exercise 11 .....	136
Exercise 12 .....	138
Exercise 13.....	139
Exercise 14 .....	140
Exercise 15.....	142
Exercise 16 .....	143
Exercise 17.....	145
Exercise 18 .....	146
Exercise 19 .....	147

<b>Inner Classes</b>	<b>149</b>
----------------------	------------

Exercise 1.....	149
Exercise 2 .....	149
Exercise 3 .....	150
Exercise 4 .....	151
Exercise 5 .....	152
Exercise 6 .....	152
Exercise 7 .....	153
Exercise 8 .....	154
Exercise 9 .....	155
Exercise 10 .....	156

Exercise 11 .....	157
Exercise 12 .....	158
Exercise 13.....	158
Exercise 14 .....	159
Exercise 15.....	160
Exercise 16 .....	161
Exercise 17.....	162
Exercise 18 .....	163
Exercise 19 .....	164
Exercise 20.....	165
Exercise 21 .....	166
Exercise 22 .....	167
Exercise 23.....	168
Exercise 24.....	170
Exercise 25 .....	174
Exercise 26.....	175

## **Holding Your Objects      179**

Exercise 1.....	179
Exercise 2 .....	180
Exercise 3 .....	180
Exercise 4.....	181
Exercise 5 .....	183
Exercise 6 .....	185
Exercise 7 .....	187
Exercise 8.....	188
Exercise 9 .....	189
Exercise 10 .....	190
Exercise 11 .....	191
Exercise 12 .....	192
Exercise 13.....	193
Exercise 14 .....	197
Exercise 15.....	197
Exercise 16 .....	198
Exercise 17.....	199
Exercise 18 .....	200
Exercise 19 .....	201
Exercise 20.....	202
Exercise 21 .....	203
Exercise 22 .....	204
Exercise 23 .....	206
Exercise 24.....	208
Exercise 25 .....	209
Exercise 26.....	210

Exercise 27 .....	212
Exercise 28 .....	214
Exercise 29 .....	214
Exercise 30 .....	215
Exercise 31 .....	217
Exercise 32 .....	219

## **Error Handling with Exceptions** **221**

Exercise 1 .....	221
Exercise 2 .....	221
Exercise 3 .....	222
Exercise 4 .....	223
Exercise 5 .....	224
Exercise 6 .....	225
Exercise 7 .....	226
Exercise 8 .....	227
Exercise 9 .....	228
Exercise 10 .....	229
Exercise 11 .....	229
Exercise 12 .....	230
Exercise 13 .....	231
Exercise 14 .....	232
Exercise 15 .....	233
Exercise 16 .....	234
Exercise 17 .....	235
Exercise 18 .....	237
Exercise 19 .....	238
Exercise 20 .....	239
Exercise 21 .....	241
Exercise 22 .....	242
Exercise 23 .....	243
Exercise 24 .....	245
Exercise 25 .....	246
Exercise 26 .....	247
Exercise 27 .....	248
Exercise 28 .....	248
Exercise 29 .....	249
Exercise 30 .....	250

## **Strings** **253**

Exercise 1 .....	253
Exercise 2 .....	254
Exercise 3 .....	255

Exercise 4 .....	256
Exercise 5 .....	257
Exercise 6 .....	260
Exercise 7 .....	261
Exercise 8 .....	262
Exercise 9 .....	262
Exercise 10 .....	263
Exercise 11 .....	266
Exercise 12 .....	267
Exercise 13.....	268
Exercise 14 .....	270
Exercise 15.....	270
Exercise 16 .....	272
Exercise 17.....	273
Alternative A.....	274
Alternative B.....	275
Exercise 18 .....	278
Alternative A.....	278
Alternative B.....	279
Exercise 19 .....	280
Alternative A.....	280
Alternative B.....	281
Exercise 20 .....	282

<b>Type Information</b>	<b>285</b>
-------------------------	------------

Exercise 1.....	285
Exercise 2 .....	286
Exercise 3 .....	288
Exercise 4 .....	288
Exercise 5 .....	289
Exercise 6 .....	290
Exercise 7 .....	293
Exercise 8 .....	294
Exercise 9 .....	295
Exercise 10 .....	298
Exercise 11 .....	299
Exercise 12 .....	303
Exercise 13.....	303
Exercise 14 .....	304
Exercise 15.....	306
Exercise 16 .....	309
Exercise 17.....	312
Exercise 18 .....	313
Exercise 19 .....	315

Exercise 20.....	316
Exercise 21 .....	317
Exercise 22 .....	318
Exercise 23 .....	320
Exercise 24 .....	321
Exercise 25 .....	324
Exercise 26 .....	325

<b>Generics</b>	<b>329</b>
-----------------	------------

Exercise 1.....	329
Exercise 2 .....	329
Exercise 3 .....	330
Exercise 4 .....	331
Exercise 5 .....	332
Exercise 6 .....	333
Exercise 7 .....	334
Exercise 8 .....	335
Exercise 9 .....	337
Exercise 10 .....	338
Exercise 11 .....	338
Exercise 12 .....	339
Exercise 13.....	340
Exercise 14 .....	341
Exercise 15.....	342
Exercise 16 .....	343
Exercise 17.....	344
Exercise 18 .....	346
Exercise 19 .....	347
Exercise 20.....	349
Exercise 21 .....	350
Exercise 22 .....	351
Exercise 23 .....	352
Exercise 24 .....	353
Exercise 25 .....	354
Exercise 26 .....	355
Exercise 27 .....	356
Exercise 28.....	356
Exercise 29 .....	357
Exercise 30.....	359
Exercise 31.....	360
Exercise 32 .....	360
Exercise 33 .....	361
Exercise 34.....	363
Exercise 35 .....	364

Exercise 36 .....	365
Exercise 37 .....	367
Exercise 38 .....	368
Exercise 39 .....	370
Exercise 40 .....	370
Exercise 41 .....	372
Exercise 42 .....	373

<b>Arrays</b>	<b>377</b>
---------------	------------

Exercise 1.....	377
Exercise 2 .....	378
Exercise 3 .....	378
Exercise 4 .....	380
Exercise 5 .....	383
Exercise 6 .....	384
Exercise 7 .....	384
Exercise 8 .....	385
Exercise 9 .....	386
Exercise 10 .....	387
Exercise 11 .....	387
Exercise 12 .....	388
Exercise 13.....	388
Exercise 14 .....	389
Exercise 15.....	390
Exercise 16 .....	392
Exercise 17.....	395
Exercise 18 .....	396
Exercise 19 .....	397
Exercise 20 .....	398
Exercise 21 .....	399
Exercise 22 .....	401
Exercise 23 .....	402
Exercise 24 .....	403
Exercise 25 .....	404

<b>Containers in Depth</b>	<b>407</b>
----------------------------	------------

Exercise 1.....	407
Exercise 2 .....	408
Exercise 3 .....	409
Exercise 4 .....	409
Exercise 5 .....	409
Exercise 6 .....	411
Exercise 7 .....	413
Exercise 8 .....	414

Exercise 9 .....	418
Exercise 10 .....	419
Exercise 11 .....	424
Exercise 12 .....	425
Exercise 13.....	426
Exercise 14 .....	428
Exercise 15.....	429
Exercise 16 .....	430
Exercise 17.....	434
Exercise 18 .....	434
Exercise 19 .....	435
Exercise 20.....	436
Exercise 21 .....	439
Exercise 22 .....	440
Exercise 23 .....	441
Exercise 24.....	443
Exercise 25 .....	445
Exercise 26 .....	449
Exercise 27 .....	451
Exercise 28.....	453
Exercise 29 .....	458
Exercise 30.....	462
Exercise 31.....	464
Exercise 32 .....	466
Exercise 33 .....	467
Exercise 34 .....	472
Exercise 35 .....	474
Exercise 36.....	476
Exercise 37 .....	481
Exercise 38.....	484
Exercise 39.....	486
Exercise 40.....	489
Exercise 41 .....	492
Exercise 42.....	494

I/O	497
-----	-----

Exercise 1.....	497
Exercise 2 .....	498
Exercise 3 .....	499
Exercise 4.....	500
Exercise 5 .....	501
Exercise 6 .....	502
Exercise 7 .....	503
Exercise 8.....	504



Exercise 9 .....	505
Exercise 10 .....	505
Exercise 11 .....	506
Exercise 12 .....	511
Exercise 13.....	512
Exercise 14 .....	513
Exercise 15.....	514
Exercise 16 .....	515
Exercise 17.....	517
Exercise 18 .....	518
Exercise 19 .....	520
Exercise 20.....	521
Exercise 21 .....	522
Exercise 22 .....	522
Exercise 23 .....	524
Exercise 24.....	525
Exercise 25 .....	525
Exercise 26 .....	530
Exercise 27 .....	531
Exercise 28.....	533
Exercise 29 .....	535
Exercise 30.....	537
Exercise 31.....	540
Exercise 32 .....	543
Exercise 33 .....	544

<b>Enumerated Types</b>	<b>547</b>
-------------------------	------------

Exercise 1.....	547
Exercise 2 .....	548
Exercise 3 .....	549
Exercise 4.....	550
Exercise 5 .....	552
Exercise 6 .....	554
Exercise 7 .....	554
Exercise 8 .....	555
Exercise 9 .....	559
Exercise 10 .....	562
Exercise 11 .....	567

<b>Annotations</b>	<b>575</b>
--------------------	------------

Exercise 1.....	575
Exercise 2 .....	578
Exercise 3 .....	581
Exercise 4 .....	584

Exercise 5 .....	585
Exercise 6 .....	585
Exercise 7 .....	586
Exercise 8 .....	587
Exercise 9 .....	588
Exercise 10 .....	589
Exercise 11 .....	590

<b>Concurrency</b>	<b>597</b>
--------------------	------------

Exercise 1.....	597
Exercise 2 .....	598
Exercise 3 .....	599
Exercise 4 .....	600
Exercise 5 .....	601
Exercise 6 .....	603
Exercise 7 .....	604
Exercise 8 .....	605
Exercise 9 .....	605
Exercise 10 .....	607
Exercise 11 .....	609
Exercise 12 .....	611
Exercise 13.....	612
Exercise 14 .....	613
Exercise 15.....	614
Exercise 16 .....	617
Exercise 17.....	621
Exercise 18 .....	623
Exercise 19 .....	623
Exercise 20 .....	625
Exercise 21 .....	626
Exercise 22 .....	628
Exercise 23 .....	630
Exercise 24 .....	632
Exercise 25 .....	635
Exercise 26 .....	637
Exercise 27 .....	640
Exercise 28.....	643
Exercise 29 .....	645
Exercise 30 .....	650
Exercise 31.....	652
Exercise 32 .....	655
Exercise 33 .....	657
Exercise 34.....	662
Exercise 35 .....	664

Exercise 36 .....	668
Exercise 37 .....	676
Exercise 38 .....	682
Exercise 39 .....	687
Exercise 40 .....	690
Exercise 41 .....	692
Exercise 42 .....	694

## Graphical

### User Interfaces 697

Exercise 1.....	697
Exercise 2 .....	697
Exercise 3 .....	698
Exercise 4 .....	699
Exercise 5 .....	700
Exercise 6 .....	701
Exercise 7 .....	702
Exercise 8 .....	704
Exercise 9 .....	705
Exercise 10 .....	708
Exercise 11 .....	709
Exercise 12 .....	710
Exercise 13.....	712
Exercise 14 .....	714
Exercise 15.....	715
Exercise 16 .....	716
Exercise 17.....	718
Exercise 18 .....	719
Exercise 19 .....	720
Exercise 20.....	724
Exercise 21 .....	726
Exercise 22 .....	728
Exercise 23 .....	729
Exercise 24 .....	731
Exercise 25 .....	734
Exercise 26 .....	737
Exercise 27 .....	738
Exercise 28 .....	740
Exercise 29 .....	742
Exercise 30 .....	743
Exercise 31.....	744
Exercise 32 .....	745
Exercise 33 .....	746
Exercise 34 .....	749

Exercise 35 .....	751
Exercise 36 .....	751
Exercise 37 .....	752
Exercise 38 .....	753
Exercise 39 .....	753
Exercise 40 .....	754
Exercise 41 .....	754
Exercise 42 .....	755
Exercise 43 .....	758

# Installing the Code

Detailed instructions for installing, configuring and testing the source code.

These instructions describe a Windows installation, but they will also act as a guide for OSX and Linux installations.

These instructions also work with the free demo version of the guide.

1. Create a directory called **C:\TIJ4-Solutions\code**.
2. Using WinZip or some other zip utility (if one is not preinstalled, search the web for a free utility), extract the zip file that you received when you purchased the guide. Unzip it into the **C:\TIJ4-Solutions\code** directory. When you're done, you should see numerous subdirectories in the **C:\TIJ4-Solutions\code** directory, including subdirectories corresponding to the chapters in the solution guide.
3. Install the Java SE Development Kit (JDK), version 5 or newer, from the download site at Sun (<http://java.sun.com/javase/downloads/index.jsp>). You'll also eventually want the documentation, which is available from the same site.
4. Set the CLASSPATH in your computer's environment. For Windows machines, right-click on the "My Computer" icon and select "Properties." Then select the "Advanced" tab and click the "Environment Variables" button at the bottom. Under "System Variables," look to see if there's already a "CLASSPATH" variable. If there is, double click it and add **.;.;C:\TIJ4-Solutions\code;** to the end of the current entry.

If there is no "CLASSPATH" variable, click the "New" button and enter **CLASSPATH**

In the "Variable name" box, and

**.;.;C:\TIJ4-Solutions\code;**

In the "Variable value" box, then click "OK". To verify that your classpath has been set, start a command prompt (see below), then enter **set** and look for the **CLASSPATH** information in the output.

5. Using the same technique as in Step 4, but for PATH instead of CLASSPATH, add the **bin** directory from your Java installation into your

system's PATH environment variable. On Windows, the default JDK installation path is under "C:\Program Files" and because this has spaces, you must quote that directory when adding it to the PATH:  
**C:"Program Files"\Java\bin;**

6. Create a directory called **C:\jars**. Place the following files into this directory:
  - **javassist.jar** (download here: [http://sourceforge.net/project/showfiles.php?group\\_id=22866](http://sourceforge.net/project/showfiles.php?group_id=22866); you may need to search for it).
  - **swt.jar** from the Eclipse SWT library (<http://download.eclipse.org/eclipse/downloads/>). Click on the most recent build number, then scroll down to "SWT Binary and Source" and select the file corresponding to your platform. Further details about finding the jar file are in *Thinking in Java, 4<sup>th</sup> Edition*, under the heading "Installing SWT."
  - **tools.jar**, which is actually part of the JDK, but you must explicitly add it to your classpath. You'll find it in the **lib** directory wherever you installed the JDK on your machine. (The default is **C:"Program Files"\Java\lib**).
  - **javaws.jar**, also part of the JDK, in the **/jre/lib/** directory.
  - **xom.jar**, available from <http://www.cafeconleche.org/XOM/>.
7. You must explicitly add each of the Jar files to your CLASSPATH, following the directions in Step 4. However, *you must also include the name of the Jar file in the CLASSPATH entry*. For example, after you put the **javassist.jar** file into the **C:\jars\** directory, the associated CLASSPATH entry is **C:\jars\javassist.jar;**
8. Install the **Ant** 1.7 (or newer) build tool by following the instructions you will find in the Ant download at <http://ant.apache.org/>.  
**Note:** **Ant** is required in order to compile the examples in the book. Once you successfully run '**ant build**' in the root directory, you can also compile each example individually (once you have the CLASSPATH set, as described in Step 4) using the **javac** command-line compiler that was installed when you completed the steps 3 and 5. To compile a file called **MyProgram.java**, you type **javac MyProgram.java**.
9. Start a command prompt in the **C:\TlJ4- Solutions\code** directory. To do this in Windows, press the "Start" button, then select "Run" and type "**cmd**" and press "OK." then type

**cd C:\TIJ4-Solutions\code**  
into the resulting command window.

10. At the prompt, type  
**ant build**  
The build should successfully compile all the chapters in the solution guide.
11. Once you've run **ant build** in the root directory, you can also move into individual chapters and type **ant** (to compile and execute the code in that chapter) or **ant build** (to compile the code only).
12. This code is designed to work without an IDE, but it has also been tested with Eclipse (free at <http://www.eclipse.org/>); see the following section for instructions on how to use the code with Eclipse.

If you want to use this code inside other IDEs you might need to make appropriate adjustments. Different IDEs have different requirements and you might find it's more trouble than it's worth right now; instead, you may want to begin with a more basic editor like JEdit (free at <http://www.jedit.org/>).

13. **Note:** The output for the programs has been verified for Java 6. Certain programs (primarily those that use hashing) can produce different output from one version to the next.

## Using Eclipse

Once you've followed the above instructions, you can use the code inside the Eclipse development environment as follows:

1. Install Eclipse from <http://www.eclipse.org/downloads/>; choose a version for Java developers and follow the installation instructions.
2. Start Eclipse, then choose File | New | Java Project from the main menu.
3. In the ensuing dialog box, under "Contents," select "Create Project from Existing Source." Press the "Browse" button and navigate to **C:\TIJ4-Solutions\code**. Enter "TIJ4-Solutions" as the project name and press the "Finish" button.

**Note:** If you are installing the demo version of the solution guide, you do not need to perform any of the following steps.

4. Eclipse will work for awhile and then present you with a “problems” pane containing a lot of errors and warnings. We’ll remove the errors in the following steps.
5. In the “Package Explorer” pane, right click on “TIJ4-Solutions” and select “Properties” (at the bottom). In the left column of the ensuing dialog box, select “Java Build Path.”
6. Select the “Source” tab. The default Eclipse configuration may have chosen to exclude and include some files. Click on “Included” and press the “Remove” button, then click on “Excluded” and press “Remove.”
7. Click on “Excluded” and press “Edit.” Under “Exclusion Patterns,” add the following files, which are not intended to compile. After you add the files, press the “Finish” button.
  - access/E04\_ForeignClass.java
  - arrays/E11\_AutoBoxingWithArrays.java
  - interfaces/E02\_Abstract.java
  - reusing/E06\_ChessWithoutDefCtor.java
  - reusing/E20\_OverrideAnnotation.java
  - reusing/E21\_FinalMethod.java
  - reusing/E22\_FinalClass.java
8. Click on the “Libraries” tab, then the “Add External Jars” button. Add **tools.jar**, **swt.jar**, **javassist.jar** and **xom.jar** that are described in step 6 of the previous section.
9. When you press OK to close the dialog box, the project should rebuild without any errors. The warnings that you see refer to code that is intentional for those solutions, in order to demonstrate features and issues of the language.



# Packages & IDEs

When Java first appeared there was no *Integrated Development Environment* (IDE) support, so you typically used a text editor and the command-line compiler. Over the years, IDE support has gotten so good (and many prevalent IDEs are free) that it's less and less likely that you'll develop in Java – or even learn the language – without an IDE.

There's a conflict, however, between IDEs and the way that *Thinking in Java* attempts to teach the language: One step at a time, using language features only after they've been introduced.

An IDE like *Eclipse* (from [www.Eclipse.org](http://www.Eclipse.org)) likes to have all its code in packages (later versions have become more tolerant of unpackaged code, but it still prefers packages). Packages, however, are not introduced until the *Access Control* chapter.

Because of the prevalence of IDEs, we have chosen to include **package** statements for all the code in this book, even for chapters before *Access Control*. If you have solved the problems in those chapters without using **package** statements, your solutions are still correct.



# Left to the Reader

We have left only a few exercises to the reader. For these, the solution typically requires some configuration on your own computer.

The exercises left to the reader include:

- Exercises 12 & 13 from the chapter *Everything Is an Object*
- Exercise 13 from the chapter *Initialization & Cleanup*
- Exercise 2 from the chapter *Access Control*
- Exercise 15 from the chapter *Generics*
- Exercise 8 from the chapter *Arrays*
- Exercise 21 from the chapter *Containers in Depth*
- Exercise 35, 36, 38, 39, and 43 from the chapter *Graphical User Interfaces*



# Everything is an Object

To satisfy IDEs like *Eclipse*, we have included **package** statements for chapters before *Access Control*. If you have solved the problems in this chapter without using **package** statements, your solutions are still correct.

## Exercise 1

```
//: object/E01_DefaultInitialization.java
/***** Exercise 1 *****/
 * Create a class containing an int and a char
 * that are not initialized. Print their values
 * to verify that Java performs default
 * initialization.
 *****/
package object;

public class E01_DefaultInitialization {
    int i;
    char c;
    public E01_DefaultInitialization() {
        System.out.println("i = " + i);
        System.out.println("c = [" + c + ']');
    }
    public static void main(String[] args) {
        new E01_DefaultInitialization();
    }
} /* Output:
i = 0
c = [ ]
*///:~
```

When you run the program you'll see that both variables are given default values: 0 for the **int**, and a "space" for the **char**.

## Exercise 2

```
| //: object/E02_HelloWorld.java
```

```

/***** Exercise 2 *****/
* Follow the HelloDate.java example in this
* chapter to create a "hello, world" program that
* simply displays that statement. You need only a
* single method in your class (the "main" one that
* executes when the program starts). Remember
* to make it static and to include the argument
* list (even though you don't use it).
* Compile the program with javac and run it using
* java. If you are using a different development
* environment than the JDK, learn how to compile
* and run programs in that environment.
*****/
package object;

public class E02_HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
} /* Output:
Hello, world!
*///:~

```

## Exercise 3

```

//: object/E03_ATypeName.java
/***** Exercise 3 *****/
* Turn the code fragments involving ATypeName
* into a program that compiles and
* runs.
*****/
package object;

public class E03_ATypeName {
    public static void main(String[] args) {
        E03_ATypeName a = new E03_ATypeName();
    }
} //:~

```

## Exercise 4

```

//: object/E04_DataOnly.java
/***** Exercise 4 *****/
* Turn the DataOnly code fragments into a
* program that compiles and runs.

```

```

*****/
package object;

public class E04_DataOnly {
    int i;
    double d;
    boolean b;
    public static void main(String[] args) {
        E04_DataOnly d = new E04_DataOnly();
        d.i = 47;
        d.d = 1.1;
        d.b = false;
    }
} ///:~

```

## Exercise 5

```

//: object/E05_DataOnly2.java
/***** Exercise 5 *****/
* Modify Exercise 4 so the values
* of the data in DataOnly are assigned to and
* printed in main().
*****/
package object;

public class E05_DataOnly2 {
    public static void main(String[] args) {
        E04_DataOnly d = new E04_DataOnly();
        d.i = 47;
        System.out.println("d.i = " + d.i);
        d.d = 1.1;
        System.out.println("d.d = " + d.d);
        d.b = false;
        System.out.println("d.b = " + d.b);
    }
} /* Output:
d.i = 47
d.d = 1.1
d.b = false
*///:~

```

## Exercise 6

```

//: object/E06_Storage.java
/***** Exercise 6 *****/

```

```

* Write a program that includes and calls the
* storage() method defined as a code fragment in
* this chapter.
*****/
package object;

public class E06_Storage {
    String s = "Hello, World!";
    int storage(String s) {
        return s.length() * 2;
    }
    void print() {
        System.out.println("storage(s) = " + storage(s));
    }
    public static void main(String[] args) {
        E06_Storage st = new E06_Storage();
        st.print();
    }
} /* Output:
storage(s) = 26
*///:~

```

## Exercise 7

```

//: object/E07_Incrementable.java
/***** Exercise 7 *****/
* Turn the Incrementable code fragments into a
* working program.
*****/
package object;

class StaticTest {
    static int i = 47;
}

public class E07_Incrementable {
    static void increment() { StaticTest.i++; }
    public static void main(String[] args) {
        E07_Incrementable sf = new E07_Incrementable();
        sf.increment();
        E07_Incrementable.increment();
        increment();
    }
} ///:~

```



You can call **increment()** by itself, because a **static** method (**main()**, in this case) can call another **static** method without qualification.

## Exercise 8

```
//: object/E08_StaticTest.java
/***** Exercise 8 *****/
* Write a program to demonstrate that no
* matter how many objects you create of a
* particular class, there is only one instance
* of a particular static field in that class.
*****/
package object;

public class E08_StaticTest {
    static int i = 47;
    public static void main(String[] args) {
        E08_StaticTest st1 = new E08_StaticTest();
        E08_StaticTest st2 = new E08_StaticTest();
        System.out.println(st1.i + " == " + st2.i);
        st1.i++;
        System.out.println(st1.i + " == " + st2.i);
    }
} /* Output:
47 == 47
48 == 48
*///:~
```

The output shows that both instances of **E08\_StaticTest** share the same **static** field. We incremented the shared field through the first instance and the effect was visible in the second instance.

## Exercise 9

```
//: object/E09_AutoboxingTest.java
/***** Exercise 9 *****/
* Write a program to demonstrate that
* autoboxing works for all the primitive types
* and their wrappers.
*****/
package object;

public class E09_AutoboxingTest {
    public static void main(String[] args) {
```

```

    Byte by = 1;
    byte bt = by;
    System.out.println("byte = " + bt);
    Short sh = 1;
    short s = sh;
    System.out.println("short = " + s);
    Integer in = 1;
    int i = in;
    System.out.println("int = " + i);
    Long lo = 1L;
    long l = lo;
    System.out.println("long = " + l);
    Boolean bo = true;
    boolean b = bo;
    System.out.println("boolean = " + b);
    Character ch = 'x';
    char c = ch;
    System.out.println("char = " + c);
    Float fl = 1.0f;
    float f = fl;
    System.out.println("float = " + f);
    Double db = 1.0d;
    double d = db;
    System.out.println("double = " + d);
}
} /* Output:
byte = 1
short = 1
int = 1
long = 1
boolean = true
char = x
float = 1.0
double = 1.0
*///:~

```

The terms *Autoboxing* and *Autounboxing* appear often in the literature. The only difference is the direction of the conversion: autoboxing converts from the primitive type to the wrapper object, and autounboxing converts from the wrapped type to the primitive type.

## Exercise 10

```

//: object/E10_ShowArgs.java
// {Args: A B C}
/***** Exercise 10 *****/

```

```

* Write a program that prints three arguments
* taken from the command line.
* You'll need to index into the command-line
* array of Strings.
*****/
package object;

public class E10_ShowArgs {
    public static void main(String[] args) {
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
    }
} /* Output:
A
B
C
*///:~

```

Remember, when you want to get an argument from the command line:

- Arguments are provided in a **String** array.
- **args[o]** is the first command-line argument and *not* the name of the program (as it is in C).
- You'll cause a runtime exception if you run the program without enough arguments.

You can test for the length of the command-line argument array like this:

```

//: object/E10_ShowArgs2.java
// {Args: A B C}
package object;

public class E10_ShowArgs2 {
    public static void main(String[] args) {
        if(args.length < 3) {
            System.err.println("Need 3 arguments");
            System.exit(1);
        }
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
    }
} /* Output:
A
B

```

```
C
*///:~
```

**System.exit()** terminates the program and passes its argument back to the operating system as a status code. (With most operating systems, a non-zero status code indicates that the program execution failed.) Typically, you send error messages to **System.err**, as shown above.

## Exercise 11

```
//: object/E11_AllTheColorsOfTheRainbow.java
/***** Exercise 11 *****/
 * Turn the AllTheColorsOfTheRainbow example into
 * a program that compiles and runs.
 *****/
package object;

public class E11_AllTheColorsOfTheRainbow {
    int anIntegerRepresentingColors;
    void changeTheHueOfTheColor(int newHue) {
        anIntegerRepresentingColors = newHue;
    }
    public static void main(String[] args) {
        E11_AllTheColorsOfTheRainbow all =
            new E11_AllTheColorsOfTheRainbow();
        all.changeTheHueOfTheColor(27);
    }
} ////:~
```

## Exercise 12

```
//: object/E12_LeftToReader.java
/***** Exercise 12 *****/
 * Find the code for the second version of
 * HelloDate.java, the simple comment-
 * documentation example. Execute Javadoc on the
 * file and view the results with your Web browser.
 *****/
package object;

public class E12_LeftToReader {
    public static void main(String args[]) {
        System.out.println("Exercise left to reader");
    }
} ////:~
```

Note that **Javadoc** doesn't automatically create the destination directory. Consult the **Javadoc** reference in the JDK documentation to learn the many uses of **Javadoc**.

## Exercise 13

```
//: object/E13_LeftToReader.java
/***** Exercise 13 *****/
 * Run Documentation1.java, Documentation2.java,
 * and Documentation3.java through Javadoc. Verify
 * the resulting documentation with your Web
 * browser.
 *****/
package object;

public class E13_LeftToReader {
    public static void main(String args[]) {
        System.out.println("Exercise left to reader");
    }
} ///:~
```

## Exercise 14

```
//: object/E14_DocTest.java
/***** Exercise 14 *****/
 * Add an HTML list of items to the documentation
 * in Exercise 13.
 *****/
package object;

/** A class comment
 * <pre>
 * System.out.println(new Date());
 * </pre>
 */
public class E14_DocTest {
    /** A variable comment */
    public int i;
    /** A method comment
     * You can <em>even</em> insert a list:
     * <ol>
     * <li> Item one
     * <li> Item two
     * <li> Item three
     */
}
```

```

    * </ol>
    */
    public void f() {}
} ///:~

```

We simply added the HTML code fragments from the chapter examples.

## Exercise 15

```

//: object/E15_HelloWorldDoc.java
/***** Exercise 15 *****/
* Add comment documentation to the program in Exercise 2.
* Extract it into an HTML file using Javadoc
* and view it with your Web browser.
*****/
package object;

/** A first example from <i>STIJ4</i>.
 * Demonstrates the basic class
 * structure and the creation of a
 * <code>main()</code> method.
 */
public class E15_HelloWorldDoc {
    /** The <code>main()</code> method which is
     called when the program is executed by saying
     <code>java E15_HelloWorldDoc</code>.
     @param args array passed from the command-line
     */
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
} /* Output:
Hello, world!
*///:~

```

## Exercise 16

```

//: object/E16_OverloadingDoc.java
/***** Exercise 16 *****/
* In the Initialization and Cleanup chapter, add
* Javadoc documentation to the Overloading.java example.
* Extract it into an HTML file using Javadoc
* and view it with your Web browser.
*****/
package object;

```

```

/** Model of a single arboreal unit. */
class Tree {
    /** Current vertical aspect to the tip. */
    int height; // 0 by default
    /** Plant a seedling. Assume height can
        be considered as zero. */
    Tree() {
        System.out.println("Planting a seedling");
    }
    /** Transplant an existing tree with a given height. */
    Tree(int i) {
        System.out.println("Creating new Tree that is "
            + i + " feet tall");
        height = i;
    }
    /** Produce information about this unit. */
    void info() {
        System.out.println("Tree is " + height + " feet tall");
    }
    /** Produce information with optional message. */
    void info(String s) {
        System.out.println(s + ": Tree is "
            + height + " feet tall");
    }
}

/** Simple test code for Tree class */
public class E16_OverloadingDoc {
    /** Creates <b>Tree</b> objects and exercises the two
        different <code>info()</code> methods. */
    public static void main(String[] args) {
        for(int i = 0; i < 5; i++) {
            Tree t = new Tree(i);
            t.info();
            t.info("overloaded method");
        }
        // Overloaded constructor:
        new Tree();
    }
}
/* Output:
Creating new Tree that is 0 feet tall
Tree is 0 feet tall
overloaded method: Tree is 0 feet tall
Creating new Tree that is 1 feet tall
Tree is 1 feet tall
overloaded method: Tree is 1 feet tall

```

```
Creating new Tree that is 2 feet tall
Tree is 2 feet tall
overloaded method: Tree is 2 feet tall
Creating new Tree that is 3 feet tall
Tree is 3 feet tall
overloaded method: Tree is 3 feet tall
Creating new Tree that is 4 feet tall
Tree is 4 feet tall
overloaded method: Tree is 4 feet tall
Planting a seedling
*///:~
```

The one-argument constructor does not check the input argument, which should be greater than zero. Statements that control the execution flow of the program appear in a later chapter of *TIJ4*.



# Operators

To satisfy IDEs like *Eclipse*, we have included **package** statements for chapters before *Access Control*. If you have solved the problems in this chapter without using **package** statements, your solutions are still correct.

## Exercise 1

```
//: operators/E01_PrintStatements.java
/***** Exercise 1 *****/
 * Write a program that uses the "short" and
 * normal form of print statement.
 *****/
package operators;
import java.util.Date;
import static net.mindview.util.Print.*;

public class E01_PrintStatements {
    public static void main(String[] args) {
        Date currDate = new Date();
        System.out.println("Hello, it's: " + currDate);
        print("Hello, it's: " + currDate);
    }
} /* Output: (Sample)
Hello, it's: Wed Mar 30 17:39:26 CEST 2005
Hello, it's: Wed Mar 30 17:39:26 CEST 2005
*///:~
```

## Exercise 2

```
//: operators/E02_Aliasing.java
/***** Exercise 2 *****/
 * Create a class containing a float and use it to
 * demonstrate aliasing.
 *****/
package operators;
import static net.mindview.util.Print.*;

class Integral {
    float f;
}
```

```

public class E02_Aliasing {
    public static void main(String[] args) {
        Integral n1 = new Integral();
        Integral n2 = new Integral();
        n1.f = 9f;
        n2.f = 47f;
        print("1: n1.f: " + n1.f + ", n2.f: " + n2.f);
        n1 = n2;
        print("2: n1.f: " + n1.f + ", n2.f: " + n2.f);
        n1.f = 27f;
        print("3: n1.f: " + n1.f + ", n2.f: " + n2.f);
    }
} /* Output:
1: n1.f: 9.0, n2.f: 47.0
2: n1.f: 47.0, n2.f: 47.0
3: n1.f: 27.0, n2.f: 27.0
*///:~

```

You can see the effect of aliasing after **n2** is assigned to **n1**: they both point to the same object.

## Exercise 3

```

//: operators/E03_Aliasing2.java
/***** Exercise 3 *****/
* Create a class containing a float and use it
* to demonstrate aliasing during method calls.
*****/
package operators;
import static net.mindview.util.Print.*;

public class E03_Aliasing2 {
    static void f(Integral y) { y.f = 1.0f; }
    public static void main(String[] args) {
        Integral x = new Integral();
        x.f = 2.0f;
        print("1: x.f: " + x.f);
        f(x);
        print("2: x.f: " + x.f);
    }
} /* Output:
1: x.f: 2.0
2: x.f: 1.0
*///:~

```

This exercise emphasizes that you're always passing references around, thus you're always aliasing. Even when you don't actually see changes being made to the code you're writing or the method you're calling, that code or method could be calling other methods that modify the object.

## Exercise 4

```
//: operators/E04_Velocity.java
// {Args: 30.5 3.2}
/***** Exercise 4 *****/
* Write a program that calculates velocity
* using a constant distance and a constant time.
*****/
package operators;

public class E04_Velocity {
    public static void main(String[] args) {
        if(args.length < 2) {
            System.err.println(
                "Usage: java E04_Velocity distance time");
            System.exit(1);
        }
        float distance = Float.parseFloat(args[0]);
        float time = Float.parseFloat(args[1]);
        System.out.print("Velocity = ");
        System.out.print(distance / time);
        // Change the next line if you want to use a different
        // unit for 'distance'
        System.out.println(" m/s");
    }
} /* Output:
Velocity = 9.53125 m/s
*///:~
```

Here we take the **distance** and **time** values from the command line. Arguments come in as a **String** array; if you need a **float** instead, use the **static `parseFloat()`** method of class **Float**. This can be difficult to locate using the JDK HTML documentation; you must remember either “parse” or that it’s part of class **Float**.

Note the difference between **System.out.print()** and **System.out.println()**; the latter terminates the current line by writing the line separator string.

## Exercise 5

```
//: operators/E05_Dogs.java
/***** Exercise 5 *****/
* Create a class called Dog with two Strings:
* name and says. In main(), create two dogs,
* "spot" who says, "Ruff!", and "scruffy" who
* says, "Wurf!". Then display their names and
* what they say.
*****/
package operators;

class Dog {
    String name;
    String says;
}

public class E05_Dogs {
    public static void main(String[] args) {
        Dog dog1 = new Dog();
        Dog dog2 = new Dog();
        dog1.name = "spot";      dog1.says = "ruff!";
        dog2.name = "scruffy";   dog2.says = "wurf!";
        System.out.println(dog1.name + " says " + dog1.says);
        System.out.println(dog2.name + " says " + dog2.says);
    }
} /* Output:
spot says ruff!
scruffy says wurf!
*///:~
```

This walks you through the basics of objects, and demonstrates that each object has its own distinct storage space.

## Exercise 6

```
//: operators/E06_DogsComparison.java
/***** Exercise 6 *****/
* Following Exercise 5 assign, a new Dog
* reference to spot's object. Test for comparison
* using == and equals() for all references.
*****/
package operators;
import static net.mindview.util.Print.*;
```

```

public class E06_DogsComparison {
    static void compare(Dog dog1, Dog dog2) {
        print("== on top references: " + (dog1 == dog2));
        print(
            "equals() on top references: " + dog1.equals(dog2)
        );
        print("== on names: " + (dog1.name == dog2.name));
        print(
            "equals() on names: " + dog1.name.equals(dog2.name)
        );
        print("== on says: " + (dog1.says == dog2.says));
        print(
            "equals() on says: " + dog1.says.equals(dog2.says)
        );
    }
    public static void main(String[] args) {
        Dog dog1 = new Dog();
        Dog dog2 = new Dog();
        Dog dog3 = dog1; // "Aliased" reference
        dog1.name = "spot";    dog1.says = "ruff!";
        dog2.name = "scruffy"; dog2.says = "wurf!";
        print("Comparing dog1 and dog2 objects...");
        compare(dog1, dog2);
        print("\nComparing dog1 and dog3 objects...");
        compare(dog1, dog3);
        print("\nComparing dog2 and dog3 objects...");
        compare(dog2, dog3);
    }
} /* Output:
Comparing dog1 and dog2 objects...
== on top references: false
equals() on top references: false
== on names: false
equals() on names: false
== on says: false
equals() on says: false

Comparing dog1 and dog3 objects...
== on top references: true
equals() on top references: true
== on names: true
equals() on names: true
== on says: true
equals() on says: true

Comparing dog2 and dog3 objects...
== on top references: false

```

```
equals() on top references: false
== on names: false
equals() on names: false
== on says: false
equals() on says: false
*///:~
```

Guess whether the following line compiles:

```
| print("== on top references: " + dog1 == dog2);
```

Why or why not? (Hint: Read the *Precedence* and *String operator + and +=* sections in *TIJ4*.) Apply the same reasoning to the next case and explain why the comparison always results in **false**:

```
| print("== on says: " + dog1.name == dog2.name);
```

## Exercise 7

```
//: operators/E07_CoinFlipping.java
/***** Exercise 7 *****/
 * Write a program that simulates coin-flipping.
 *****/
package operators;
import java.util.Random;

public class E07_CoinFlipping {
    public static void main(String[] args) {
        Random rand = new Random(47);
        boolean flip = rand.nextBoolean();
        System.out.print("OUTCOME: ");
        System.out.println(flip ? "HEAD" : "TAIL");
    }
} /* Output:
OUTCOME: HEAD
*///:~
```

This is partly an exercise in Standard Java Library usage. After familiarizing yourself with the HTML documentation for the JDK (downloadable from [java.sun.com](http://java.sun.com)), select “R” at the JDK index to see various ways to generate random numbers.

The program uses a ternary if-else operator to produce output. (See the *Ternary if-else Operator* section in *TIJ4* for more information.)

**NOTE:** You will normally create a **Random** object with no arguments to produce different output for each execution of the program. Otherwise it can

hardly be called a *simulator*. In this exercise and throughout the book, we use the seed value of 47 to make the output identical, thus verifiable, for each run.

## Exercise 8

```
//: operators/E08_LongLiterals.java
/***** Exercise 8 *****/
 * Show that hex and octal notations work with long
 * values. Use Long.toBinaryString() to display
 * the results.
 *****/
package operators;
import static net.mindview.util.Print.*;

public class E08_LongLiterals {
    public static void main(String[] args) {
        long l1 = 0x2f; // Hexadecimal (lowercase)
        print("l1: " + Long.toBinaryString(l1));
        long l2 = 0X2F; // Hexadecimal (uppercase)
        print("l2: " + Long.toBinaryString(l2));
        long l3 = 0177; // Octal (leading zero)
        print("l3: " + Long.toBinaryString(l3));
    }
} /* Output:
l1: 101111
l2: 101111
l3: 1111111
*///:~
```

Note that **Long.toBinaryString()** does not print leading zeroes.

## Exercise 9

```
//: operators/E09_MinMaxExponents.java
/***** Exercise 9 *****/
 * Display the largest and smallest numbers for
 * both float and double exponential notation.
 *****/
package operators;
import static net.mindview.util.Print.*;

public class E09_MinMaxExponents {
    public static void main(String[] args) {
        print("Float MIN: " + Float.MIN_VALUE);
        print("Float MAX: " + Float.MAX_VALUE);
    }
}
```

```

        print("Double MIN: " + Double.MIN_VALUE);
        print("Double MAX: " + Double.MAX_VALUE);
    }
} /* Output:
Float MIN: 1.4E-45
Float MAX: 3.4028235E38
Double MIN: 4.9E-324
Double MAX: 1.7976931348623157E308
*///:~

```

## Exercise 10

```

//: operators/E10_BitwiseOperators.java
/***** Exercise 10 *****/
* Write a program with two constant values, one
* with alternating binary ones and zeroes, with
* a zero in the least-significant digit, and the
* second, also alternating, with a one in the
* least-significant digit. (Hint: It's easiest to
* use hexadecimal constants for this.) Combine
* these two values every way possible using the
* bitwise operators. Display the results using
* Integer.toBinaryString().
*****/
package operators;
import static net.mindview.util.Print.*;

public class E10_BitwiseOperators {
    public static void main(String[] args) {
        int i1 = 0xaaaaaaaa;
        int i2 = 0x55555555;
        print("i1 = " + Integer.toBinaryString(i1));
        print("i2 = " + Integer.toBinaryString(i2));
        print("~i1 = " + Integer.toBinaryString(~i1));
        print("~i2 = " + Integer.toBinaryString(~i2));
        print("i1 & i1 = " + Integer.toBinaryString(i1 & i1));
        print("i1 | i1 = " + Integer.toBinaryString(i1 | i1));
        print("i1 ^ i1 = " + Integer.toBinaryString(i1 ^ i1));
        print("i1 & i2 = " + Integer.toBinaryString(i1 & i2));
        print("i1 | i2 = " + Integer.toBinaryString(i1 | i2));
        print("i1 ^ i2 = " + Integer.toBinaryString(i1 ^ i2));
    }
} /* Output:
i1 = 10101010101010101010101010101010
i2 = 1010101010101010101010101010101
~i1 = 1010101010101010101010101010101

```











# Exercise 13

```
//: operators/E13_BinaryChar.java
/***** Exercise 13 *****/
* Write a method to display char values in
* binary form. Demonstrate it using several
* different characters.
*****/
package operators;
import static net.mindview.util.Print.*;

public class E13_BinaryChar {
    public static void main(String[] args) {
        print("A: " + Integer.toBinaryString('A'));
        print("!: " + Integer.toBinaryString('!'));
        print("x: " + Integer.toBinaryString('x'));
        print("7: " + Integer.toBinaryString('7'));
    }
} /* Output:
A: 1000001
!: 100001
x: 1111000
7: 110111
*///:~
```

# Exercise 14

```
//: operators/E14_CompareStrings.java
/***** Exercise 14 *****/
* Write a method that compares two String arguments
* using all the Boolean comparisons and print the
* results. Perform the equals() test for the == and
* !=. In main(), call your method with different
* String objects.
*****/
package operators;

public class E14_CompareStrings {
    public static void p(String s, boolean b) {
        System.out.println(s + ": " + b);
    }
    public static void compare(String lval, String rval) {
        System.out.println("lval: " + lval + " rval: " + rval);
        //! p("lval < rval: " + lval < rval);
        //! p("lval > rval: " + lval > rval);
    }
}
```

```

    //! p("lval <= rval: " + lval <= rval);
    //! p("lval >= rval: " + lval >= rval);
    p("lval == rval", lval == rval);
    p("lval != rval", lval != rval);
    p("lval.equals(rval)", lval.equals(rval));
}
public static void main(String[] args) {
    compare("Hello", "Hello");
    // Force creation of separate object:
    String s = new String("Hello");
    compare("Hello", s);
    compare("Hello", "Goodbye");
}
} /* Output:
lval: Hello rval: Hello
lval == rval: true
lval != rval: false
lval.equals(rval): true
lval: Hello rval: Hello
lval == rval: false
lval != rval: true
lval.equals(rval): true
lval: Hello rval: Goodbye
lval == rval: false
lval != rval: true
lval.equals(rval): false
*///:~

```

The only comparisons that actually compile are `==` and `!=`. This (slightly tricky) exercise highlights the critical difference between the `==` and `!=` operators, which compare *references*, and `equals()`, which actually compares *content*.

Remember that quoted character arrays also produce references to **String** objects. In the first case, the compiler recognizes that the two strings actually contain the same values. Because **String** objects are immutable (you cannot change their contents), the compiler can merge the two **String** objects into one, so `==` returns **true** in that case. However, when you create a separate **String s** you also create a distinct object with the same contents, therefore the `==` returns **false**. *The only reliable way to compare objects for equality is with `equals()`.* Be wary of any comparison that uses `==`, which always and only compares two references to see if they are identical (that is, they point to the same object).