Chapter 1

1. TIP: For answers to common Java questions, turn to one of the Java FAQ(frequently asked question) lists on the Web—see <http://www.apl.jhu.edu/~hall/java/FAQs-and-Tutorials.html>

Chapter 3

3.2 Comments in Java, like comments in most programming languages, do not show up to the executable program. Thus, you can add as many comments as needed without frar of bloating the code.

3.3.2 In particular, you cannot test

if ( x = Double.NaN ) // is never true

to check whether a particular result equals Double.NaN. All “not a number” values are considered distinct. However, you can use the Double.isNaN method:

if( Double.isNaN( x) ) //check whether x is “ not a number “

Floating-point numbers are not suitable for financial calculation in which roundoff errors cannot be tolerated. For example, the command System.out.println (2.0 – 1.1 ) prints 0.899999999999, not 0.9 as you would expect. Such roundoff errors are caused by the fact that floating-point numbers are represented in the binary number system. There is no precise binary representation of the fraction 1/10, just as there is no accurate representation of the fraction 1/3 in the decimal system. If you need precise numerical computations without roundoff errors, use the BigDecimal class, which is introduced later in this chapter.

3.3.3 Our strong recommendation is not to use the char type in your programs unless you are actually manipulating UTF-16cod units. You are almost always better off treating strings (which we will discuss in the section “Strings” on page 53) as abstract data types.

**3.4 If you are really curious as to what Unicode characters are “letters” as far as Java is concerned, you can use the isJavaIdentifierStart and isJavaIdentifierPart methods in the Character class to check**

~~3.4.1 In general, you should not have two names that only differ in their letter case.~~

~~3.4.2~~

~~In Java, it is considered good style to declare variables as closely as possible to the point where they are first used.~~

3.5

|  |
| --- |
| public class ZeroDivisionWillNotThrowError {  public static void main(String[] args) {  float f = 2.33f;  System.out.println(f / 0);// will print "Infinity(无穷大)"  }  } |

However, methods tagged with the strictfp keyword must use strict floating-point operations that yield reproducible results. For example, you can tag main as

public static strictfp void main(String[] args)

Then all instructions inside the main method use strict floating-point computations. If you tag a class as strictfp, then all of its methods use strict floating-point computations

3.5.1

Because these operators change the value of a variable, they cannot be applied to numbers themselves. For example, 4++ is not a legal statement

3.5.4

The functions in the Math class use the routines in the computer’s floating-point unit for fastest performance. If completely predictable results are more important than fast performance, use the StrictMath class instead. It implements the algorithms from the “Freely Distributable Math Library” fdlibm, guaranteeing identical results on all platforms.

3.5.5

int🡪float, long🡪float, int🡪double are arrow(箭头) denote(指示) conversions that may lose precision. For example, a large integer such as 123456789 has more digits than the float type can represent. When the integer is converted to a float , the resulting value has the correct magnitude but it loses some precision.

int n = 123456789;

float f = n; // f is 1.23456792E8

3.5.6

CAUSION: if you try to cast a number of one type to another that is out of the range for the rarget type, the result will be a truncated number that has a different value. For example, (byte)300 is actually 44.

3.6.1

The second parameter of substring is the first position that you don’t want to get from current string.

3.6.3

Overall(总的来说), the designers of Java decided that the efficiency of sharing outweighs the inefficiency of string editing by extracting substrings and concatenating. Look at your own programs; we suspect(猜想，怀疑，不可信的) that most of the time, you don’t change strings—you just compare them.

3.6.4

If the virtual machine would always arrange for equal strings to be shared, then you could use the == operator for testing equality. But only string constants are shared, not strings that are the result of operations like + or substring. Therefore, never use == to compare strings lest(以免) you end up with a program with the worst kind of bug—an intermittent one that seems to occur randomly.

3.6.6

String API:

int compareTo(String other)

returns a negative value (don’t say it’s -1) if the string comes before other in dictionary order, a positive value(don’t say it’s 1 ) if the string comes after other in dictionary order, or 0 if the strings are equals

3.6.8

NOTE: if all string editing happens in a single thread ( which is usually the case), you should use StringBuilder instead. The APIs of both Stringbuffer and StringBuilder classes are identical(同一的；完全相同的).

3.7.3

To read from a file, construct(建造，构造；创立) a Scanner object from a File object, like this:

Scanner in = new Scanner(new File(“myfile.txt”));

CAUSION: You can construct a Scanner with a string parameter, but the scanner interprets the string as data, not a file name. For example, if you call

Scanner in = new Scanner(“myFile.txt”); //ERROR??

then the scanner will see ten characters of data: ‘m’, ’y’, ’f’, and so on .That is probably not what was intended in this case.

NOTE: when you specify a relative file name, such as “myfile.txt” , “myfirectory/myfile.txt”, or “../myfile.txt” , the file is located relative to the directory in which the Java virtual machine was started. If you launched your program from a command shell , by executing

java MyProg

then the starting directory is the current directory of the command shell. However, if you use an integrated development enviroment , the starting directory is controlled by the IDE. You can find the directory location with this call:

String dir = System.getProperty(“use.dir”);

If you run into grief with locating files, consider using absolute path names such as “/home/me/mydirectory/myfile.txt”

3.8.1

However, you may not declare identically named variables in two nested blocks. For example, the following is an error and will not compile:

public static void main(String[] args){

int n;

{

int k;

int n; // error – can’t redefine n in inner block

}

}

C++ can, but Java can’t

3.8.4

CAUTION: Be careful about testing for equality of floating-point numbers in loops. A for loop that looks like

for(double x=0; x != 10; x+=0.1) may never end. Because of round off errors, the final value may not be reached exactly. For example, in the loop above, x jumps from 9.99999999999998 to 10.09999999999998 because there is no exact binary representation for 0.1

3.8.6

Here’s an example that shows the break statement at work. Notice that the label must precede the outermost loop out of which you want to break. It also must be followed by a colon.

|  |
| --- |
| public class BreakTagInForloop {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  read\_data**:** while (true) {  System.out.println("while loop is called.");  for (int i = 1; i != 0;) {  System.out.println("Enter a number >= 0 :");  i = in.nextInt();  if (i < 0) {  break **read\_data**;  }  }  }  }  } |

3.9

BigInteger and BigDecimal: Unfortunately, you cannot use the familiar mathematical operators such as + and \* to combine big numbers .Instead, you must use methods such as add and multiply in the big number classes.

BigInteger c = a.add(b) ; // c= a+ b

BigInteger d = c.multiply(b.add(BigInteger.valueOf(2))); // d = c \* (b + 2)

3.10.1

The enhanced for loop

for( variable : collection ) statement

sets the given variable to each element of the collection and the executes the statement( which , of course, may be a block). The collection expression must be an array or an object of a class that implements the Iterable interface, such as ArrayList.

3.10.6

NOTE: To print out a quick and dirty list of the elements of a two-dimensional array, call

System.out.println(Arrays.deepToString(a));

The output is formatted like this:

[[14,23,4],[3,54,90],[38,23,5]]

4.2

Using **Predefined** Classes(使用现有类)

However, not all of these show off the typical features of object orientation. Take, for example, the Math class.

4.2.1

Variables are not automatically initialized to null. You must initialize them, either by calling new or by setting them to null.

4.2.2

You can construct a calendar object for midnight on a specific date by supplying year, month, and day:

new GregorianCalendar(1999,11,31)

Somewhat curiously, the months are counted from 0. therefore, 11 is December. For greater clarity, there are constants like Calendar.DECEMBER.

4.3.6

CAUTION: Be careful not to write accessor method that return references to mutable objects. We violated that rule in our Employee class in which the getHireDay method returns an object of class Date:

class Employee

{

public Date getHireDay(){

return hireDay;

}

}

This breaks the encapsulation! Consider the following rogue code:

Employee harry = ..;

Date d = harry.getHireDay();

double tenYearsInMilliSeconds = 10 \* 365.25 \* 24 \* 60 \* 60 \* 1000;

d.setTime(d.getTime() – (long) tenYearsInMilliSeconds);

//let’s give Harry ten years added seniority

The reason is subtle. Both d and harry.hireDay refer to the same object (see figure 4-5). Applying mutable methods to d automatically changes the private state of the employee object!

If you need to return a reference to a mutable object, you should clone it first. A clone is an exact copy of an object that is stored in a new location. We discuss cloning in detail in Chapter 6. Here is the corrected code:

Class Employee{

public Date getHireDay(){

return (Date) hireDay.clone();

}

}

As a rule of thumb, always use clone whenever you need to return a copy of a mutable data field.

4.5

Let’s try to write a method that swaps two employee objects:

public static void swap( Employee x , Employee y) // doesn’t work

{

Employee temp = x;

x = y;

y = temp;

}

4.6.2

If you don’t set a field explicitly in a constructor, it is automatically set to a default value: numbers to 0, Boolean values to false, and object references to null. But it is considered poor programming practice to rely on this. Certainly, it makes it harder for someone to understand your code if fields are being initialized invisibly.

4.6.6

NOTE: Here is a Java trivia fact to amaze your fellow Java coders: You can write a “Hello, World” program in Java without ever writing a main method.

public class Hello{

static{

System.out.println(“Hello, World”);

}

}

When you invoke the class with java Hello, the class is loaded, the static initialization block prints “Hello, World” and only then do you get an ugly error message that main is not defined. You can avoid that blemish by calling System.exit(0) at the end of the static initialization block.

4.6.8

You can add a finalize method to any class. The finalize method will be called before the garbage collector sweeps away the object. In practice, do not rely on the finalize method for recycling any resources that are in short supply—you simply cannot know when this method will be called.

4.7.2

Cumbersome constants: If you use lots of constants with tedious names, you will welcome static import. For example,

if(d.get(DAY\_OF\_WEEK) == MONDAY)

is easier on the eye than

if(d.get(Calendar.DAY\_OF\_WEEK) == Calendar.MONDAY)

4.7.4

Starting with version 1.2, the JDK implementers rigged the class loader to explicitly disallow loading of user-defined classes whose package name starts with “java.”!

5.1

If the subclass constructor does not call a super class constructor explicitly, then the default (no-parameter) constructor of the super class is invoked.

5.1.3

NOTE: The return type is not part of the signature. However, when you override a method, you need to keep the return type compatible. Prior to Java SE5.0, the return types had to be identical. However, it is now legal for the subclass to change the return type of an overridden method to a subtype of the original type.

5.1.3

It would be time consuming to carry out this search every time a method is called. Therefore, the virtual machine pre-computes for each class a method table that lists all method signatures and the actual methods to be called. When a method is actually called, the virtual machine simply makes a table lookup.

5.1.4

NOTE: Recall that fields can also be declared as final. A final field cannot be changed after the object has been constructed. However, if a class is declared as final, only the methods, not the fields, are automatically final.

5.2.2

Here is a recipe for writing the perfect equals method:

1. Name the explicit parameter otherObject—later, you need to cast it to another variable that should call other.
2. Test whether this happen to be identical to otherObject:

if( this == otherObject ) return true;

This statement is just an optimization. In practice, this is a common case. It is much cheaper to check for identity than to compare the fields.

1. Test whether otherObject is null and return false if it is. This test is required.

if(otherObject == null) return false;

1. Compared the classes of this and otherObject. If the semantics of equals can change in subclasses, use the getClass test:

if( getClass() != otherObject.getClass()) return false;

1. Cast otherObject to a variable of your class type:

ClassName other = (ClassName)otherObject;

1. Now compare the fields, as required by your nothion of equality. Use == for primitive type fields, equals for object fields. Return true if all fields match, false otherwise.

return field1 == other.field1 && field2.equals(other.fields2)&&..;

If you redefine equals in a subclass , include a call to super.equals(other);

5.2.3

Note that the String s = “OK” and String t = new String(“OK”) have the same hash code because, for strings, the hash codes are derived from their contents.

You definitions of equals and hashCode must be compatible: if x.equals(y) is true, then x.hashCode() must be the same value as y.hashCode(). For example, if you define Employee.equals to compare employee IDs, the hashCode method needs to hash the IDs, not employee names or memory address.

5.2.4

Most(but not all) toString methods follow this format: the name of the class, followed by the field values enclosed in square brackets. Here is an implementation of the toString method for the Employee class:

public String toString(){

return “Employee[name=” + name + “, salary=”+salary + “]”;

}

CAUTION: annoyingly, arrays inherit the toString method from Object, with the added twist that the array type is printed in an archaic format. For example,

int[] luckyNumbers = {1,3,4,5};

String s = “” + luckNumbers;

yields the string “[I@1a46e30”. (The prfix [I denotes an array of integers.] The remedy is to call the static Arrays.toString method instead. The code

String s = Array.toString(luckyNumbers);

yields the string “[1,3,4,5]”

to correctly print multidimensional arrays (that is , arrays of arrays), use Arrays.deepToString.

5.3

If you call add and the internal array is full, the array list automatically creates a bigger array and copies all the objects from the smaller to the bigger array.

If you already know, or have a good guess, how many elements you want to store, then call the ensureCapacity method before filling the array list:

staff.ensureCapacity(100);

That call allocates an internal array of 100 objects. Then, the first 100 calls to add do not involve any costly reallocation.

You can also pass an initial capacity to the ArrayList constructor:

ArrayList<Employee> staff = new ArrayList<Employee>(100);

5.3.1

CAUTION: Do not call list.set(i, x) until the size of the array list is larger than i. For example, the following code is wrong:

ArrayList<Employee> staff = new ArrayList<Employee>(100);// capacity is 100, size 0;

list.set(0, x); // no element 0 yet

Use the add method instead of set to fill up an array, and use set only to replace a previously added element.

5.6

public enum Size{SAMLL, MEDIUM, LARGE, EXTRAL\_LARGE};

The type defined by this declaration is actually a class. The class has exactly four instances—it is not possible to construct new objects.

Therefore, you never to use equals for values of enumerated types. Simply use == to compare them.

SAMLL.ordinal():returns the zero-based positon of this enumerated constant in the enum declaration

5.7.1

The newInstance method calls the default constructor ( the one that takes no parameters) to initialize the newly created object. An exception is thrown if the class has no default constructor.

If you need to provide parameters for the constructor of a class you want to create by name in this manner, then you can’t use statements like the preceding. Instead, you must use the newInstance method in the Constructor class.

5.7.4

AccessibleObject.setAccessible(fields, true);

System.out.println(new ObjectAnalyzer().toString(yourArraylist));

You can use the generic toString method to implement the toString methods of your own classes, like this:

public String toString(){

return new ObjectAnalyzer().toString(this);

}

5.7.5

NOTE: We present this program to illustrate how to work with arrays through refection. If you just want to grow an array, use the copyOf method in the Arrays class.

Employee[] a = new Employee[100];

…

// array is full

a = Arrays.copyOf( a, a.length \* 11 / 10 + 10);

5.7.6

public Object invoke(Object implicitParameter, Object[] explicitParameters)

For static methods , pass null as the implicit parameter. Pass primitive type values by using wrappers. Primitive type return values must be unwrapped.

5.8

Design hints for inheritance

1. Place common operations and fields in the superclass.
2. Don’t use protected fields
3. Use inheritance to model the “is-a” relationship.
4. Don’t use inheritance unless all inherited methods make sense.
5. Don’t change the expected behavior when you override a method.
6. Use polymorphism, not type information.
7. Don’t overuse reflection.

Chapter 6

6.1

When calling x.compareTo(y), the compareTo method must actually be able to compare two objects and return an indication whether x or y is larger. Then method is supposed to return a negative number if x is smaller than y, zero if they are equal, and a positive number otherwise.

If there is a common algorithm for comparing subclass objects, simple provide a single compareTo method in the superclass and declare it as final.

6.1.1

It is legal to tag interface method as public, and fields as public static final. Some programmer do that, either out of habit or for greater clarify. However, the Java Language Specification recommends that the redundant keywords not be supplied, and we follow that recommendation.

6.3

Toolkit.getDefaultToolkit().beep();

6.4.2

public void actionPerformed(ActionEvent event){

if(TalkingClock.this.beep)Toolkit.getDefaultToolkit().beep();

}

TalkingClock jabberer = new TalkingClock(1000,true);

TalkingClock.TimerPrinter listener = jabberer.new TimePrinter();

6.4.5

Not only can they access the fields of their outer classes, they can even access local variables! However, those local variables must be declared final.

The final restriction is comewhat inconvenient. Suppose, for example, you eant to update a counter in the enclosing scope. Here, we want to count how often the compareTo method is called during sorting.

int counter = 0;

Date[] dates = new Date[100];

for(int i=0;i<dates.length;i++){

dates[i] = new Date(){

public int compareTo(Date other){

counter++ ; // error

return super.compareTo(other);

}

}

}

Arrays.sort(dates);

System.out.println(counter + “ comparisons…”);

You can’t declare sounter as final because you clearly need to update it. You can’t replace it with an Integer because Integer objects are immutable. The remedy is to use an array of length 1:

final int[] counter = new int[1];

Date[] dates = new Date[100];

for(int i=0;i<dates.length;i++){

dates[i] = new Date(){

public int compareTo(Date other){

counter[0]++ ;

return super.compareTo(other);

}

}

}

(They array variable is still declared as final, but that merely means that you can’t have it refer to a different array. You are free to mutate the array elements)

6.4.7

NOTE: You use a static inner class whenever the inner class does not need to access an outer class object. Some programmers use the term nested to describe static inner classes.