8354
Java from the Very Beginning, Part III

Steve Ryder
JSR Systems, Austin, Texas
Stephen Pipes
IBM Hursley Park Labs, United Kingdom

Objectives

- Learn how to implement methods
- Manage source and class files
- Understand how to use Java documentation
- Look at common problems and their solutions

Agenda

- Part II recap
- Methods
- Managing source files
- Javadoc
- Common problems

Recap

```
for ( int i = 0; i < 2; i++) {
   for ( int j = 0; j < 3; j++) {
      if ( i == j ) {
        continue;
      }
      System.out.println( "i = " + i + " j = " + j );
   }
}
```

Which lines are part of the output?

A. i = 0 j = 0
B. i = 0 j = 1
C. i = 0 j = 2
D. i = 1 j = 0
E. i = 1 j = 1
F. i = 1 j = 2

Recap

```
outer: for ( int i = 0; i < 2; i++) {
   for ( int j = 0; j < 3; j++) {
      if ( i == j ) {
        continue outer;
      }
      System.out.println( "i = " + i + " j = " + j );
   }
}
```

Which lines are part of the output?

A. i = 0 j = 0
B. i = 0 j = 1
C. i = 0 j = 2
D. i = 1 j = 0
E. i = 1 j = 1
F. i = 1 j = 2

Agenda

- Part II recap
- ➡ Methods
- Managing source files
- Javadoc
- Common problems

Methods

- A method is a named block of code
- It may take arguments (parameters)
- It may return a value
- A method consists of:
  1. A method declaration
  2. A method body

Method Declaration

- A method declaration contains:
  - the name of the method
  - the return type
  - the number and type of arguments passed to the method
  - details of which other classes and objects can call the method
  - details of any errors or exceptions which can occur in the method
  - other *modifiers*

Method Declaration

accessLevel modifiers **returnType methodName(** paramList **)** *throws exceptions*

accessLevels:

➡ **private**	class
➡ *"friendly" or "package"*	same package
➡ **protected**	package, subclass
➡ **public**	world

Method Declaration

accessLevel modifiers **returnType methodName(** paramList **)** *throws exceptions*

modifiers:

- **static** — class method
- **abstract** — no method body!
- **final** — can't be changed
- **native** — another language
- **synchronized** — needs a monitor

Method Declaration

accessLevel modifiers **returnType methodName(** paramList **)** *throws exceptions*

return type:

- any valid Java type
- if your method does not return a value, use **void**

Method Declaration

accessLevel modifiers **returnType methodName(** paramList **)** *throws exceptions*

method name:

- any valid Java identifier
- a class may have several methods with the SAME NAME, as long as those methods are differentiated by the number or type or their arguments
- called *overloading*

Overloading Example

```
class DataRenderer {

 public void draw(String s) {
    ...
 }

 public void draw(int i) {
    ...
 }

 public void draw(float f) {
    ...
 }

}
```

Method Declaration

accessLevel modifiers **returnType methodName(** paramList **)** *throws exceptions*

parameter list:

- a comma delimited list of variable declarations

  type *name*, type *name*, ...
- *name* is used within the method body to refer to the argument
- arguments are **passed by value**

Method Declaration

accessLevel modifiers **returnType methodName(** paramList **)** *throws exceptions*

throws list:

- a comma delimited list of exceptions that may be thrown by the method

A method must either

1. Catch any exceptions that may occur within its body by providing an exception handler, or
2. Specify that it may throw those exceptions

Method Declaration

- The simplest possible method declaration:
  - void doNothing()
- Method returning an integer:
  - int getInteger()
- Method to square a number:
  - int square(int aNumber)
- Method to sum two numbers:
  - int sum(int a, int b)

Method Body

- Follows method declaration
- Enclosed in curly braces: { }
- May contain local variable declarations
  - exist only within scope of the method
- Refer to arguments by name
- Unless return type is declared void, must contain a return statement.

Method Body

```
/**
 * Method to sum two numbers and return the result
 * @param a first number
 * @param b second number
 * @return a + b
 */
int sum(int a, int b) {
  int returnValue;
  returnValue = a + b;
  return returnValue;
}
```

Calling Methods

- Within the same class:
  - methodName(args)
- A static method of another class:
  - className.methodName(args)
- A method of an object:
  - objectName.methodName(args)

Exercise

- Write a program to compute the value of a degree 2 polynomial

A Solution

- In main...

```
int polynomialAtX;
for(int x = 0; x <= 10; x++) {
    polynomialAtX = polynomial(x);
    System.out.println(x + "\t\t" + polynomialAtX);
}
```

- Polynomial function:

```
/**
 * Calculate the polynomial function at x
 */
public static int polynomial(int x) {
  int returnValue;
  returnValue = ( ( ( a * x ) + b ) * x ) + c;
  return returnValue;
}
```

Agenda

- Part II recap
- Methods
- Managing source files
- Javadoc
- Common problems

Managing Source Files

- To make classes easier to find and use, to avoid naming conflicts, and to control access, programmers bundle groups of related classes into packages.
- A package is a collection of related classes and interfaces that provides access protection and namespace management.

Creating a package

- To create a package, simply put a class or interface into it
- To put a class or interface in a package, put a package statement at the top of the source file:

```
package graphics;
class Circle {
    ...
}
```

- Include the package statement at the top of every file containing a class to be included in the package

Packages

- The full name of a class includes its package
  - The full name of class Circle in package graphics is: graphics.Circle
- By convention, use your company's reversed internet domain name in your package names
  - IBM (ibm.com), creates packages beginning com.ibm, e.g. com.ibm.mq

Using Packages

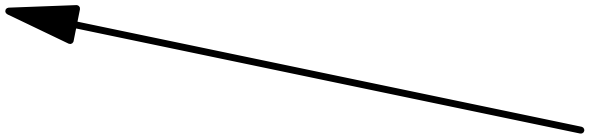
- To use a public package member from outside its package you must either:
  1. Use its full (disambiguated) name e.g. java.io.FileReader
  2. Import the package member (import java.io.FileReader;)
  3. Import the whole package (import java.io.*;)
- Import statements are placed after the package statement, and before any class or interface definitions

Using Packages

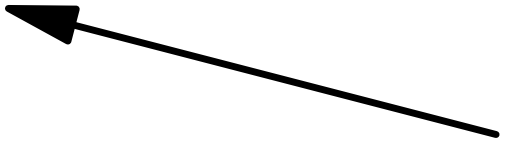
```
package com.ibm.graphics;
import java.awt.*;

public class Circle {
    ...
}
```

package statement



one or more import statements



class or interface declaration(s)

Naming and Placing Java files

- A public class Foo must be placed in a file Foo.java.
- The source file must be placed in a directory whose name maps the package name
  - e.g. class Foo in package com.ibm.mq would be com\ibm\mq\Foo.java
  - Your CLASSPATH must point to the root of this tree (ie. the directory with "com" in it)
  - By default, "." is in the CLASSPATH

Agenda

- Part II recap
- Methods
- Managing source files
- ➡ Javadoc
- Common Problems

JavaDoc

- The documentation format for the Java class libraries
- A tool for documenting your own programs
- Uses special comment blocks that begin with "/**" and end with "*/"

JavaDoc

- Any leading spaces or asterisks are stripped from each line before processing
- A doc comment may contain HTML tags, but should avoid structural tags such as <H2>
- The first sentence should be a summary sentence, suitable for display on its own.

JavaDoc - Class

```
/**
 * Simple class to represent a Circle.
 * @author Adrian Colyer
 * @version 1.0
 */
public class Circle { ... }
```

@author --> Author: entry

@version --> Version: entry

JavaDoc - Method

```
/**
 * Calculate the circumference of the circle
 * @return the circle's circumference
 * @see Circle#area
 */
public float circumferece( ) { ... }
```

@return --> Returns: entry

@see --> See also: entry

JavaDoc - Method

```
/**
 * Set the centre of the Circle
 * @param x the x co-ordinate of the centre
 * @param y the y-co-odinate of the centre
 */
public void setCentre( int x, int y ) { ... }
```

@param --> adds the parameter and its description to the "Parameters:" section.

JavaDoc - Field

```
/**
 * The radius of the circle.
 */
public int radius;
```

JavaDoc

javadoc [-d directory -author -version ...] filenames

or...

javadoc [-d directory -author -version ...] packagename

Exercise

- Run the javadoc tool

Agenda

- Part II recap
- Methods
- Managing source files
- Javadoc
- ➡ Common problems

Common Problems

- The compiler complains that it can't find a class
  - Make sure you've imported the class or its package.
  - Unset the CLASSPATH environment variable, if it's set.
  - Make sure you're spelling the class name exactly the same way as it is declared. Case matters!
  - If your classes are in packages, make sure that they appear in the correct subdirectory as outlined in Managing Source and Class Files.
  - Also, some programmers use different names for the class name from the .java filename. Make sure you're using the class name and not the filename. In fact, make the names the same and you won't run into this problem for this reason.

Common Problems

- The interpreter complains that it can't find a class
  - Make sure you specified the class name--not the class file name--to the interpreter.
  - Unset the CLASSPATH environment variable, if it's set.
  - If your classes are in packages, make sure that they appear in the correct subdirectory as outlined in Managing Source and Class Files.
  - Make sure you're invoking the interpreter from the directory in which the .class file is located.

Common Problems

- My program doesn't work!
  - Did you forget to use break after each case statement in a switch statement?
  - Did you use the assignment operator = when you really wanted to use the comparison operator ==?
  - Are the termination conditions on your loops correct? Make sure you're not terminating loops one iteration too early or too late. That is, make sure you are using < or <= and > or >= as appropriate for your situation.
  - Remember that array indices begin at 0

Common Problems

- My program doesn't work!
  - Are you comparing floating-point numbers using ==? Remember that floats are approximations of the real thing. The greater than and less than (> and <) operators are more appropriate when conditional logic is performed on floating-point numbers.
  - Are you trying to change the value of an argument from a method? Arguments in Java are passed by value and can't be changed in a method.
  - Did you inadvertently add an extra semicolon (;), thereby terminating a statement prematurely?

Common Problems

- My program doesn't work!
  - Are you using the correct conditional operator? Make sure you understand && and || and are using them appropriately.
  - Do you use the negation operator ! a lot? Try to express conditions without it. Doing so is less confusing and error-prone.
  - Are you using a do-while? If so, do you know that the loop executes one more time than a similar while loop?

Summary

- Methods are identified by name and arguments
- Arguments are passed by value
- Group classes into packages
- Javadoc is used to document the Java class libraries and your own code