Chapter 1

Review of Java Fundamentals

Lecture 2

Jenny Walter

Fall 2008

Language Basics

- Java application
  - Collection of classes
    - One class contains the *main* method = Point of entry for program execution
- Java programs can also be written as applets, in which case they do not have a main method

Classes

- Simply stated, *a class defines a data type*
- Defines data fields and methods available for instances of the class
- An *object* in Java is an *instance* of a class
- Class definition line includes
  - Optional access modifier (`public` or `private`)
  - Keyword `class`
  - Optional `extends` clause
  - Optional `implements` clause
  - Class body: The part inside the curly braces that contains all data fields and method definitions

Class Structure

```
package helloprogram;
/*
 * File: HelloProgram.java
 * -----------------------
 * This program displays "hello, world" on the screen.
 */

import acm.graphics.*;
import acm.program.*;

public class HelloProgram extends GraphicsProgram {

   public HelloProgram() {
      this.start();
      this.add( new GLabel("hello, world",100,75));
   }
}
```

package name (opt)

header comment (opt)

import statements (opt)

class definition line

constructor

Class Hierarchy

Every Java class is a subclass of either

- Another Java class (uses keyword **`extends`**)
- The **`Object`** class: Class at the root of Java class hierarchy; every class extends **`Object`** implicitly

`new` operator

Creates an object or instance of a class

Packages (aka libraries)

Mechanism for grouping related classes

`package` statement

Type this statement outside class curly braces to indicate a class is part of a package

Java assumes all classes in a particular package are contained in the same directory

Java API consists of many predefined packages

Importing Packages

- `import` statement
  - Allows use of classes contained in packages without the inconvenient "fully qualified name"
  - Included outside class curly braces, usually at top of file
- Package `java.lang` is implicitly imported to all Java code
- Other commonly imported packages include
  - java.util, java.io, java.awt, javax.swing
  - acm.graphics, acm.program

Data Fields

- Class members that are either variables or constants
- Declared within class curly braces, but not within the curly braces of any method.
- Data field declarations can contain
  - Access modifiers (`public`, `private`, …)
  - Use modifiers (`static`)

Local Variables

- Declared within method curly braces.
- Not visible (or usable) outside method braces.
- Local variable declarations contain no access or use modifiers.
- Never need to use a dot operator as a prefix to a local variable.

Methods

- Implement tasks
- Declared within class curly braces
- Each method should perform one well-defined task and *no more.*
- Method modifiers
  - Access modifiers and use modifiers
- Void method
  - Returns nothing but has a *side effect*
- Valued method
  - Returns a value
  - Body must contain **`return`** `expression;`
  - Return type must be declared in method definition (method header line)

Side Effects

- Any value returned from a method must be explicitly specified in a **`return`** statement
- Side effects include actions such as:
  - Input and Output (I/O)
  - Changing values of instance variables
  - Calling other methods

Methods

- Syntax of a method declaration:

```
access-modifier use-modifiers return-type
          method-name (formal-parameter-list)
{
  method-body
}
```

- Multiple methods of the same name can be defined
  - method overloading
  - methods with same names must have different number or type of parameters
- Primitive type arguments are passed by value
  - For objects and arrays, a reference value is copied instead

Methods

- Syntax of a method call:

```
<objectName or ClassName>.method-name(argument-list)
```

- Method calls are analogous to messages being sent between objects or classes
- Method calls should always be preceded by the object or class receiving the message followed by the "dot" operator
- I encourage you to use the keyword "this" to precede the dot operator when calling an inherited method or an instance method

Methods

- Constructor
  - Special kind of method
  - Has the same name as the class, no return type, no modifiers
  - Executed only when an object is created
- A class can contain multiple constructors (*overloaded* constructors)

How to Access *public* Members of an Object

- Data fields and methods not declared static
  - Use the *object* name, followed by a period, followed by member name
- Data fields and methods declared *static*
  - Use the *class* name, followed by a period, followed by member name

public vs. *private*

- Public class members of Class X are accessible by any class Y that has privilege to access class X
- Private class members are accessible only within the class curly braces

Static Class Members

- Technically, data fields and methods declared with use modifier **`static`** are "class members"
- Static members are invoked independently of any instance of the class, using the class name followed by the dot operator, followed by the member name

Comments

- Comment line
  - Begins with two slashes (//)
  - Continues until the end of the line
- Multiple-line comment
  - Begins with /* and ends with */
  - Useful for debugging; "commenting out" code
  - You can't nest multiple-line comments
- `javadoc` comments
  - Begin with /** and end with */

Identifiers and Keywords

- Identifier
  - Sequence of letters, digits, underscores, and dollar signs (no other symbols allowed)
  - Must begin with either a letter or underscore
  - Used to name members of the program
  - Java distinguishes between uppercase and lowercase letters
- Keywords
  - Java reserved identifiers

Naming Conventions

- keywords, variable names, and method names start with a lower case letter and every word after the first is capitalized.
- Class names differ only in that they start with a capital letter
- Constants are written in all capital letters with underscores separating words.
- If you adhere to these naming conventions, your code will be much easier for other experienced programmers to understand

Java Keywords (Reserved Words)

abstract
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
extends
false
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
true
try
void
volatile
while

Variable

- Name for a memory location
- Contains a value of primitive type or a reference
- Its name is a Java identifier
- Declared by preceding variable name with data type

```
double radius; // radius of a sphere
String name; // reference to a String object
```

Literal Constants

- Indicate particular values (e.g., numbers, characters, strings of characters) within a program
- Used to initialize the value of a variable and discouraged from use most other places in a program
- Decimal integer constants
  - Do not use commas, decimal points, or leading zeros
  - Default data type is either `int` or `long`
- Floating constants
  - Written using decimal points
  - Default data type is `double`

Literal Constants

- Character constants
  - Enclosed in single quotes (I.e., 'a')
  - Default data type is `char`
  - Literal character strings
    - Sequence of characters enclosed in double quotes (I.e., "This is a string of characters")

Named Constant

- Name for a memory location that cannot be changed after declared
- Contains a value of primitive or reference type
- Its name is a Java identifier and its value must be set on the same line it is declared
- Declared by preceding variable name with data type and the keyword `final`

```
public final double PI = 3.14;
public final String WEEK_DAY_1 = "Monday";
```

Primitive Data Types

- Organized into four categories
  - `boolean`
  - `char`
  - `int`
  - `double (float)`
- char and int types are called integral types
- Integral and floating-point types are called arithmetic types

Primitive Data Types

- Value of primitive type is not considered an object
  - `java.lang` provides wrapper classes for each of the primitive types
- Auto-boxing
  - Automatically converts from a primitive type to the equivalent wrapper class
- Auto-unboxing
  - Reverse process

Reference types

- Data type used to locate an object
- Java does not allow programmer to perform operations on the reference value
- Location of object in memory (on the heap) can be assigned to a reference variable

Assignments and Expressions

- Expressions
  - Combination of variables, constants, operators, and parentheses
  - Must be evaluated before value is known
- Assignment statement (= is assignment operator)
  - Example: `radius = r;`

Assignments and Expressions

- Other assignment operators

```
-=
*=
/=
%=
++
--
```

Statements

- Combination of expressions
- Each Java statement ends with a semicolon
- Every executable statement that is not a declaration should be contained within the curly braces of a method

Arithmetic Expressions

Combine variables and constants with arithmetic operators and parentheses

Arithmetic operators: *, /, %, +, -

Relational Expressions

- Combine variables and constants with relational (I.e. comparison) and equality operators and parentheses
    - Relational operators: <, <=, >=. >
    - Equality operators: ==, !=
  - All relational expressions evaluate to `true` or `false`

Logical Expressions

Combine variables and constants of arithmetic types in relational expressions and joins these expressions with logical operators

- Logical operators: &&, ||
- Evaluate to `true` or `false`
- Short-circuit evaluation
    - Evaluates logical expressions from left to right
    - Stops as soon as the value of expression is apparent

Assignments and Expressions

- Implicit type conversions
  - Occur during assignment and during expression evaluation
  - Right-hand side of assignment operator is converted to data type of item on left-hand side if possible, otherwise an error occurs
  - Floating-point values are truncated not rounded
  - Integral promotion
    - Values of type `byte`, `char`, or `short` are converted to `int`
  - Conversion hierarchy
    - `int` → `long` → `float` → `double`

Assignments and Expressions

- Explicit type conversions
  - Possible by means of a cast
  - Cast operator
    - Unary operator
    - Formed by enclosing the desired data type within parentheses
- Multiple assignments
  - Embed assignment expressions within assignment expressions
    - Example: `a = 5 + (b = 4)`
    - Evaluates to `9` while `b` is assigned `4`

Using the acm.jar

- Useful because
  - it creates new windows that display output that is more visually appealing than text showing up at the bottom of the screen in an IDE
  - it makes graphics programming code much more compact
- For a class that writes output on a console window:
  - import acm.program.*
  - extend ConsoleProgram
  - in the constructor, include a call to this.start() (opens the window)
  - System.out.println's become println's
  - reading int's goes from reading a String and converting it to an int to readInt("Prompt for user:")
- For a class that sets up a graphics canvas:
  - import acm.program.* and acm.graphics.* and java.awt.* and sometimes javax.swing.*
  - extend GraphicsProgram
  - in the constructor, include a call to this.start() (opens the window)

Class TestCircle

```
package testcircle;

/*
 * File: TestCircle.java
 * -----------------------
 * This program displays a circle on the canvas.
 */

public class TestCircle {

    public static void main(String[] args) {
        new Circle();
    }

}
```

Class Circle

```
package testcircle;
import acm.program.*;
import acm.graphics.*;
import java.awt.*;

public class Circle extends GraphicsProgram {

    public Circle() {
        this.start();
        GOval myCircle = new GOval(200,200,200,200);
        myCircle.setVisible(true);
        myCircle.setFilled(true);
        myCircle.setColor(Color.green);
        this.add(myCircle);
    }

}
```

Class SimpleSphere

```
package simplesphere;

public class SimpleSphere {
   private double radius;
   public static final double DEFAULT_RADIUS = 1.0;
   public SimpleSphere() {
      this.radius = this.DEFAULT_RADIUS;
   }  // end 0-parameter constructor
   public SimpleSphere(double r)  {
      this.radius = r;
   }  // end 1-parameter constructor
   public double getRadius() {
      return this.radius;
   }  // end getRadius
   public double getVolume() {
      double radiusCubed = radius * radius * radius;
      return 4 * Math.PI * radiusCubed / 3;
   }  // end getVolume
}  // end class SimpleSphere
```

Class TestSimpleSphere

```
package simplesphere;

public class TestSimpleSphere {
   public static void main (String[] args) {
      SimpleSphere ball;
      ball = new SimpleSphere(19.1);
      System.out.println("The volume of a sphere of "
         + "radius " + ball.getRadius() +" inches is "
         + (float)ball.getVolume()
         + " cubic inches\n");
   }   // end main
}  // end TestSimpleSphere
```

Class Circle

```
package testcircle;
import acm.program.*;
import acm.graphics.*;
import java.awt.*;

public class Circle extends GraphicsProgram {

    public Circle() {
        this.start();
        GOval myCircle = new GOval(200,200,200,200);
        myCircle.setVisible(true);
        myCircle.setFilled(true);
        myCircle.setColor(Color.green);
        this.add(myCircle);
    }

}
```