Chapter 2

Creating Classes

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Chapter 2 discusses the concepts needed to create classes. The first section focuses on the various primitive data types available in Java programs. The arithmetic operators that can be used to perform calculations on numerical values are introduced. Students will learn the precedence and associativity for these operators. As discussed in Chapter 1, a class consists of two sections: the data declaration section and the method definition section. The next two sections of the chapter focus on these in turn. Students will also learn to use assignment operators. The final section focuses on the initial modeling stage of program development. In this stage, the objects that will be used in a program are identified in a model.

Chapter Objectives

Students should be able to describe:

- Data Values and Arithmetic Operations
- Constructing a Data Declaration Section: Variables
- Completing the Class: Methods
- Assignment Operations
- Program Design and Development: Object Identification
- Common Programming Errors

Instructor Notes

Data Values and Arithmetic Operations

Java data types are divided into two general categories: primitive and reference types. Primitive types are used to represent numbers, character data, Booleans. Reference types are associated with classes, arrays, and interfaces. The primitive data types are shown in Figure 2.1, reference types in Figure 2.2. A literal is a value that explicitly identifies itself such as 1 or “Hello World.”

Integer Literals

An integer is a whole number without a decimal point. In Java, integer literals range between -2147483648 and +2147483647. They are stored in 4 bytes. Java also defines a long integer type that is stored in 8 bytes and ranges between -9223372036854775808 and +9223372036854775807. These numbers are written by adding an L to the end of the number: 8976929L. Short and byte integer types can also be used for variables, but not literals. Integer data types are summarized in Table 2.1.

Floating-Point Values

A floating-point value is a real number, positive or negative, with a decimal point. Java defines two data types that store floating-point values: `double` and `float`. A float number is stored in 4 bytes and a double in 8. Table 2.2 summarizes floating-point data types. Floating-point numbers can be expressed in exponential notation as shown on page 64.

Character Values

Character values are letters and numbers. Character literals are represented by placing a character inside single-quotes, such as 'a.' Java stores characters using Unicode encoding as unsigned integers. Character strings are reference types of the class `String`.

Teaching Tip	Students can learn more about Unicode from the site: www.unicode.org/

Escape Sequences

Escape sequences are created by placing a \ in front of one or more characters. This indicates to the compiler that special processing is required. Table 2.4 lists escape sequences.

Boolean Values

Boolean data must have a value of either true or false. Boolean variables are used for conditional processing.

Arithmetic Operations

Java supports the standard mathematical operators +, -, *, /, and %. A simple arithmetic expression combines two operands with one operator. An example of such an expression is 1 + 1. When evaluating a simple arithmetic expression, the rules listed in the middle of page 68 govern the data type of the result.

Arithmetic expressions can be used as parameters to methods such as those that output results. For example, the expression `System.out.println(1+1)` will output the result 2. Special caution should be used when combining string data with numbers in an output statement. The + symbol is the string concatenation operator, and is evaluated before the addition operator.

QuickTest Program 2.1 outputs arithmetic operation results using `System.out.println()`.

Teaching Tip	Students should create their own programs to perform arithmetic calculations. They should note how the rules of precedence apply by creating equations that exercise these rules.

An overloaded operator is an operator that acts differently depending on the operands it is working on. In the case of the + operator, the operator works differently based on whether integers, floating-point numbers, or even strings are the operands.

Teaching Tip	Overloading is an important concept in object-oriented programming and will be covered more fully later in the book.

Integer Division

Integer division results in dropping the fractional part of the result. Therefore, 15/2 = 7.

The Modulus Operator

The modulus operator is represented using the % symbol. This operator returns the remainder of a division. For example, 15 % 4 = 3.

Negation

Negation is a unary operator that reverses the sign of the operand. The symbol for negation is the same as that for subtraction: -.

Operator Precedence and Associativity

The rules for operator precedence are listed on page 72. Parentheses are used to indicate that certain operations should be performed first. Table 2.7 summarizes these rules. Note that the rules for Java operator precedence are the standard rules used in mathematics.

String Concatenation

The string concatenation operator joins two strings into one. It is represented using the + symbol. Note that when performing string concatenation, it is important to include a space character in one of the strings (if a space between words is desired).

Quick Quiz

1. What are the integer data types?
   Answer: `byte`, `short`, `int`, and `long`

2. Provide an example of a mixed-mode expression.
   Answer: 1.2 + 3

3. What is the result of 5/2?
   Answer: 2

4. What operator is used to perform string concatenation?
   Answer: +

Constructing a Data Declaration Section: Variables

Integer, floating-point, and character data is stored in and retrieved from the computer's memory. Memory is divided into addressed locations, and the data that is stored in memory is accessed by address. Figure 2.4 illustrates this concept. A variable is a name given by a programmer to a memory address, and is used to reference the data stored in that addressed space. The rules for selecting variable names are the same as for other identifiers such as class and method names.

Assignment statements, such as `num1 = 4`, are used to assign values to the memory locations referenced by variables. An assignment statement can either assign a literal value to a variable or the value of one variable to another. The = symbol is the assignment operator.

Declaration Statements

A declaration statement specifies the data type and name of a variable. The format for a Java declaration statement is:

```
optionalAccessSpecifier dataType variableName;
```

The optional access specifier is public, private, protected, or left blank. The int data type is used to store an integer. The float and double data types store floating-point values. An initial value for a variable can be assigned by combining the variable declaration with an assignment statement in the form:

```
private int num1 = 15;
```

Constructing a Data Declaration Section

A variable's classification depends on where in the class it is declared and the modifiers used in the declaration. Table 2.8 explains the four variable classifications: instance, class, local, and parameter. The table indicates where each type of variables is declared within a class.

Instance variables are the most commonly used. Instance variables are declared at the beginning of a class in the data declaration section. An example class definition with the data declaration section completed is:

```
public class RoomType
{
   // data declarations section
   private double length; // declare length as a double variable
   private double width; // declare width as a double variable
   //method definitions section
}
```

Creating Objects

Java objects are created using the new operator. The format for using this operator is:

```
variableName = new ClassName();
```

Before an object is created, a variable of that object type must be declared. As with primitive variable declarations, the declaration and initialization of a reference variable can be completed in one statement:

```
ClassName variableName = new ClassName();
```

An object can be declared as an instance variable, as a class variable, or as a local variable. A local variable is declared within a method and the declaration does not include a visibility modifier. All local variables are private.

When a variable representing an object is declared, its value is initialized to null. Null is a special value that indicates that an object is uninitialized. Once an object is created using the new operator space for that object, including memory locations for each instance variable, are created. This is illustrated in Figure 2.8. The new operator is also referred to as the dynamic memory allocation operator. The process of creating a class is also called instantiating a class.

An important difference between a primitive data type (such as an integer) and a reference data type (such as a class) is that the primitive variable identifier refers directly to a memory address, while the reference variable identifier points to a memory location where the object is stored.

Each instance of a class that is created has its own instance variables. Figure 2.9 illustrates this concept.

Notice the private modifier given to the length and width instance variables in the text in Program 2.2 (and above). This means that these variables can only be accessed from within the class. In order for other classes to reference these variables, methods must be developed for the class. This technique enforces data security.

Cleansing Memory

When a reference variable is reassigned to another location, the memory location that it used to point to remains occupied with data, but there is no way to reference this data. This is illustrated in Figure 2.10. Eventually, memory becomes filled up with such useless data. This is called a memory leak. In older programming languages, programmers were responsible for releasing memory locations that were no longer used. Sloppy programming could lead to memory leaks. The Java language frees programmers from this concern through a process known as garbage collection. The JVM periodically cleans up memory locations that are no longer referenced by the program without any need for the programmer to do anything.

Teaching Tip	Read more about garbage collection: www.javacaps.com/scjp_gar.html

Specifying Storage Allocation

There are several reasons for specifying variable names and data types in a data declaration section. The first reason is that it provides the programmer a convenient list of the variables that will be used in a program. The second is that it protects the programmer against spelling errors in variable names that might cause logical program errors. Thirdly, Java can allocate the correct amount of space for a variable based on its data type.

Quick Quiz

1. How would you declare a private instance variable named sum with an int data type?
   Answer: `private int sum;`

2. Which operator is used to create an instance of a class?
   Answer: `new`

3. True or False: Java does not have the same potential for memory leak problems that older programming languages do.
   Answer: True

Completing the Class: Methods

Table 2.9 lists three types of methods that all classes should have. Generally, classes also provide other methods to implement specific functionality. Each method consists of a method header and a method body. The format for a method header is shown in Figure 2.13.

Constructor Methods

When the new operator is used to create an instance of a class, the constructor method for the class is called by the JVM. The purpose of a constructor is to initialize the instance variables of the class. As shown in Figure 2.14, the name of a constructor is the same as the name of the class, and the declaration does not include a return type. A class can have no constructor, or more than one constructor. If more than one constructor exists, each one must differ in the number and/or type of parameters. If no constructor is provided, each instance variable is initialized to the default values listed in Table 2.10. A constructor that does not define any parameters is called a default constructor.

An example of a constructor for the RoomType class is:

```
public RoomType()
{
    length = 0.0;
    width = 0.0;
}
```

Accessor Methods

An **accessor method** is used to access the value of an instance variable (or information derived from instance variables). Another name for an accessor is a `get()` method. It is common to name an accessor method `getX()` where X is the name of the referenced variable.

Mutator Methods

The purpose of a mutator method is to change the value of an instance variable. Another name for a mutator method is a `set()` method. It is common to name a mutator method `setX()` where X is the name of the referenced variable.

Teaching Tip	The concept of encapsulation or data hiding is very important in object-oriented programming. An article on this subject can be found here: www.developer.com/java/other/article.php/3387591

In addition to constructors, accessors, and mutators, methods for a class provide the behavior for the class. For example, a method `calculateArea()` for the `RoomType` class would print the product of the length and width as shown on page 95.

The full listing for the `RoomType` class appears on page 96. Program 2.3 on page 97 makes use of the `RoomType` class in a Java program. When calling methods belonging to a class from another object, it is necessary to specify the variable name that refers to the instance of the class. The format for doing so is:

```
instanceName.methodName();
```

When calling a method from within a class, the method name alone can be used. As an alternative the `this` keyword can be used as the instance name:

```
this.methodName();
```

Convenience

For the purpose of convenience, classes may be combined into a single file in one of two ways. As shown on pages 100-101, the `main()` method from one class may become a method of the other class in order to test the functionality of the class. An alternative is to write both classes in a single .java file, but remove the public modifier from the class that does not declare a `main()` method.

Assignment Operations

Assignment statements are used to assign values to variables. The assignment operator most commonly used in Java is the = symbol. The operand that appears to the right of the = operator is a literal value, a variable, or an expression. An expression is any combination of constants and variables that can be evaluated to yield a result. Several examples of assignment statements are shown at the bottom of page 103.

QuickTest Program 2.6 uses assignment statements to calculate the area of a rectangle. QuickTest Program 2.7 presents the same information in a dialog.

Multiple Declarations

Declarations of variables with the same data type can be grouped together. For example:

```
double length, width, area;
```

Coercion

When assignment statements mix data types, data type conversion is sometimes performed. A coercion automatically occurs only when a smaller range numerical data type is assigned to a variable of a larger range type. An example of this would be assigning an integer value to a double variable. It is not possible to perform the reverse operation since data would be lost.

Assignment Variations

It is possible to use the same variable on both sides of an assignment statement. For example, `sum = sum + 10;` increments the value of the sum variable by 10. This type of assignment statement can be written using one of the shortcut assignment operators that are listed on page 108.

Accumulating

A common programming calculation is accumulating. This involves adding values to one variable, as shown on page 109. QuickTest Program 2.9 performs an accumulation. QuickTest Program 2.10 performs an accumulation on strings.

Counting

Another common calculation involves adding a fixed number to a variable repeatedly. The increment operator ++ is used to increment a number by 1. The -- operator is called the decrement operator and subtracts 1 from the number. These operators can be placed on either side of the variable as either a prefix or postfix operator. When used in a prefix position, the increment/decrement is performed before value is used in an assignment expression. When placed in a postfix position the increment/decrement is used after the assignment is made.

Program Design and Development: Object Identification

When creating an object-oriented program to solve a problem, a systematic series of steps can be followed to develop the program. The first step is to develop a model to represent the problem. Since the program will be object-oriented, the model must consist of one or more objects. Each object is identified by name, and in terms of its attributes and behaviors. An object diagram, such as that shown in Figure 2.22 identifies an object, its attributes, and its behaviors.

From Objects to Classes

The next step in the process is to create a Java class that matches the object diagram. The object attributes are translated into variables, while the behaviors become methods. An instance of this class is said to have a state. Practically, the state is defined by the values of each instance variable.

Procedural versus Object Orientations

Comparing Program 2.12 with QuickTest Program 2.13 shows the difference between object-orientation and procedure-orientation for a program. The two programs produce the same result, but the way they work is very different. Program 2.12 uses an object `RoomType` to encapsulate information for the needed calculations. QuickTest Program 2.13 simply performs the calculations using primitive values.

Quick Quiz

1. What is another name for an accessor method?
   Answer: A `get()` method

2. Provide an example of an assignment statement.
   Answer: `sum = 10;`

3. A(n) ____ automatically occurs only when a smaller range numerical data type is assigned to a variable of a larger range type.
   Answer: coercion

4. Provide an example of using the prefix decrement operator.
   Answer: --value;

5. What is the first step in creating an object-based model?
   Answer: Thinking in objects

Discussion Questions

- How do you choose what data type to use for a variable?
- What are the implications of changing a data type once a program has been written?
- What (if any) is a disadvantage of using private instance variables and accessor and mutator methods over public instance variables?
- Under what circumstances should a variable be local rather than an instance variable?

Key Terms

- **Accessor Method:** A member method that accesses a class's private data members for the purpose of returning individual values
- **Attribute:** Defines the properties of interest for an object
- **Behavior:** Defines how the object reacts to its environment
- **Boolean:** A data type restricted to one of two values: true or false
- **Char:** A data type, includes the letters of the alphabet (both uppercase and lowercase), the 10 digits 0 through 9, and special symbols such as 1 $. , - !
- **Constructor:** Any method that has the same name as its class
- **Double:** A floating-point value required to be stored using 8 bytes
- **Float:** A floating-point number stored using 4 bytes
- **Implicit (Implied) Object:** An object name that precedes the method name when the method is called
- **Instance Variable:** Any variable whose declaration statement is placed within a class's body and outside any method and does not contain the static reserved word
- **Int:** Any integer value within the range 22147483648 to 12147483647 that must be stored using 4 bytes
- **Integer:** A whole number
- **Model:** A representation of a problem
- **Object Diagram:** A diagram that identifies attributes and behavior of object
- **Operator Associativity:** Ordering of computation of operators
- **Operator Precedence:** Priority relative to all other operators
- **Primitive Data Types:** Numerical types known as integers and floating-point numbers and the character and Boolean types
- **Real Number:** A number with a decimal point
- **Reference Data Types:** Classes, arrays, interfaces
- **String Concatenation:** Adding two strings together to form one
- **This Reference:** An object that a method is to operate on
- **Variable:** Programmer-defined name for memory address whose value may vary during program execution