* **JDK consists of a set of separate programs for developing and testing Java programs, each of which is invoked from a command line.**
* **Besides JDK, there are more than a dozen Java development tools on the market today. Three major development tools are:**
* **JBuilder by Borland (**[**http://www.borland.com/**](http://www.borland.com/)**)**
* **NetBeans Open Source by Sun (**[**http://www.netbeans.org/**](http://www.netbeans.org/)**)**
* **Eclipse Open Source by IBM (**[**http://www.eclipse.org/**](http://www.eclipse.org/)**)**
	+ - * **ALL these are IDEs**
* **A Java program can be written in many ways.**
	+ **Java applications,**
	+ **applets, and**
	+ **servlets.**
1. **Applications are standalone programs that can be executed from any computer with a JVM.**
2. **Applets** **are special kinds of Java programs that run from a Web browser.**
3. **Servlets are special kinds of Java programs that run from a Web server to generate dynamic Web contents.**
* **Let us begin with a simple Java program that displays the message "Welcome to Java!" on the console. The program is shown in** **Listing 1.1****.**
* **Listing 1.1. Welcome.java**
* **Every Java program must have at least one class.**
* **A class is a construct that defines data and methods.**
* **Each class has a name.**
* **By convention, class names** **start with an uppercase letter.**
* **In this example, the class name is Welcome.**
* **In order to run a class, the class must contain a method named main.**
* **The JVM executes the program by invoking the** **main method.**
* **A method is a construct that contains statements. The main method in this program contains the System.out.println statement. This statement prints a message "Welcome to Java!" to the console.**
* **This file must end with the extension .java and**
* **must have the exact same name as the public class name**
* **Java source programs are case-sensitive. It would be wrong, for example, to replace main in the program with Main**

**Displaying Text in a Message Dialog Box**

* **use the showMessageDialog method in the JOptionPane class**

**import javax.swing.JOptionPane;**

**public class WelcomeInMessageDialogBox {**

 **public static void main(String[] args) {**

**//System.out.println(“Welcome to Java”);**

**JOptionPane.showMessageDialog(null, "Welcome to Java!", "Display Message", OptionPane.INFORMATION\_MESSAGE);**

 **}**

 **}**

* **Java's predefined classes are grouped into packages.**
* **JOptionPane is in the javax.swing package.**
* **JOptionPane is imported to the program using the import** **statement in line 4 so that the compiler can locate the class.**
* **If you replace JOptionPane on line 9 with javax.swing.JOptionPane****, you don't need to import it in line 4. javax.swing.JOptionPane is the full name for the JOptionPane class**

**2.2. Writing Simple Programs**

* **Writing a program involves**
	+ **designing simple program and**
	+ **data structures, as well as**
	+ **translating algorithms into programming code.**
* **An** ***algorithm*** **describes how a problem is solved in terms of the actions to be executed, and it specifies the order in which the actions should be executed.**
* **Algorithms can help the programmer plan a program before writing it in a programming language.**
* **The algorithm for a program can be described as follows:**
	+ **Read in the radius.**
	+ **Compute the area using the following formula:**
		- **area = radius x radius x p**
	+ **Display the area.**

* **Data structures involve data representation and manipulation.**
* **Java provides data types for representing**
	+ **integers,**
	+ **floating-point numbers** **(i.e., numbers with a decimal point),**
	+ **characters, and**
	+ **Boolean types.**
	+ **These types are known as** ***primitive data types*****.**
	+ **Java also supports**
		- **array and**
		- **string types as objects.**
* **Some advanced data structures, such as**
	+ **stacks,**
	+ **sets, and**
	+ **lists, have built-in implementation in Java.**

**Now the outline of the program would look like this:**

**As you know, every application must have a main** **method where program execution begins. So the program is expanded as follows:**

**public class ComputeArea {**

 **public static void main(String[] args) {**

 **// Step 1: Read in radius**

 **// Step 2: Compute area**

 **// Step 3: Display the area**

 **}**

**}**

****

* **Since Java is case-sensitive, X and x are different identifiers**

**2.6. Constants**

* **final datatype CONSTANTNAME = VALUE;**
* **Example:**

**final double PI = 3.14159;**

**2.7. Numeric Data Types and Operations**

| **Table 2.1. Numeric Data Types** |
| --- |
| **Name** | **Range** | **Storage Size** |
| **byte** | **-27 (-128) to 27 - 1(127)** | **8-bit signed** |
| **short** | **-215 (-32768) to 215 - 1(32767)** | **16-bit signed** |
| **int** | **-231 (-2147483648) to 231 - 1(2147483647)** | **32-bit signed** |
| **long** | **-263 to 263 – 1** | **64-bit signed** |
|  | **(i.e., -9223372036854775808 to 9223372036854775807)** |  |
| **float** | **Negative range: -3.4028235E + 38 to -1.4E-45** | **32-bit IEEE 754** |
|  | **Positive range: 1.4E-45 to 3.4028235E + 38** |  |
| **double** | **Negative range: -1.7976931348623157E+308 to -4.9E-324** | **64-bit IEEE 754** |
|  | **Positive range: 4.9E-324 to 1.7976931348623157E+308** |  |

 **2.7.1. Numeric Operators**

 **+,-,%,\***

**The result of integer division is an integer. The fractional part is truncated. For example, 5/2 yields 2, not 2.5, and —5 / 2 yields —2, not —2.5**

**5.0/2.0 = 2.5 Decimal division**

**5/2 = 2 Integer division**

**5 / 9 = 0**

* **The %** **operator is often used for positive integers but also can be used with negative integers and floating-point values. The remainder is negative only if the dividend is negative. For example,**
	+ - **-7 % 3 yields -1,**
		- **-12 % 4 yields 0,**
		- **-26 % -8 yields -2, and**
		- **20 % -13 yields 7.**
* **Suppose today is Saturday, you and your friend are going to meet in 10 days. What day is in 10** **days? You can find that day is Tuesday using the following expression:**

****

* **Another example, 500** **seconds contains 8 minutes and 20 seconds.**

****

**Integer Literals**

**The statement System.out.println (2147483648), for example, would cause a compilation error, because 2147483648 is too long as an int value.**

**Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,**

**System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);**

**displays 0.5000000000000001, not 0.5, and**

**System.out.println(1.0-0.9);**

**displays 0.09999999999999998, not 0.1****. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.**

**To denote an integer literal of the long type, append the letter L to it (e.g., 2147483648L).**

**Floating-Point Literals**

* **By default, a floating-point literal is treated as a double type value.**
* **For example, 5.0 is considered a double value, not a float value.**
* **You can make a number a float by appending the letter f or F, and**
* **you can make a number a double** **by appending the letter d or D.**
* **For example, you can use**
	+ **100.2f or 100.2F for a float number, and**
	+ **100.2d or 100.2D for a double number.**
* **The double** **type values are more accurate than the float type values.**
* **For example,**
	+ **System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);**
	+ **displays 1.0 / 3.0 is 0.3333333333333333.**
* **System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);**
	+ **displays 1.0F / 3.0F is 0.33333334.**

**2.7.4. Shorthand Operators**

| **Table 2.4. Increment and Decrement Operators** |
| --- |
| **Operator** | **Name** | **Description** |
| **++var** | **preincrement** | **The expression (++var) increments var by 1 and evaluates to the new value in var after the increment.** |
| **var++** | **postincrement** | **The expression (var++) evaluates to the original value in var and increments var by 1.** |
| **--var** | **predecrement** | **The expression (––var) decrements var by 1 and evaluates to the new value in var after the decrement.** |
| **var--** | **postdecrement** | **The expression (var––) evaluates to the original value in var and decrements var by 1.** |

**Example: i=10;**

 **int x = 10 \* i++ here the value of x is 100, however if**

 **int x = 10\*++1 here x is 110**

**Example:**

**double x = 1.0;**

**double y = 5.0;**

**double z = x–– + (++y);**

 **What is the value of z, y and x (ans:7,6,0)**

**2.8. Numeric Type Conversions**

1. **If one of the operands is double, the other is converted into double.**
2. **Otherwise, if one of the operands is float, the other is converted into float.**
3. **Otherwise, if one of the operands is long, the other is converted into long.**
4. **Otherwise, both operands are converted into int.**

**For example, the result of 1 / 2 is 0, because both operands int values. The result of 1.0 / 2 is 0.5, because 1.0 is double and 2 is converted to 2.0**

**Type Casting (Narrowing or widening)**

**float f = (float)10.1;**

**int i = (int)f; Here i is 10**

**Listing 2.4. SalesTax.java**

|  |
| --- |
|  **1 public class SalesTax {** **2 public static void main(String[] args) {** **3 double purchaseAmount = 197.55;** **4 double tax = purchaseAmount \* 0.06;** **5 System.out.println((int)(tax \* 100) / 100.0);** **6 }** **7 }** |

**the tax is evaluated as 11.853 (line 4). The statement in line 5 displays the tax 11.85 with two digits after the decimal point. Note that (int)(tax \* 100) is 1185, so (int)(tax \* 100) / 100.0 is 11.85.**

**2.9.3. Casting Between char and Numeric Types**

**When a floating-point value is cast into a char, the integral part of the floating-point value is cast into a char.**

**char c = (char)65.25; // decimal 65 is assigned to t**

**System.out.println(c); // c is character A**

**When a char** **is cast into a numeric type, the character's Unicode is cast into the specified numeric type.**

**int i = (int)'A'; // the Unicode of character A is assigned to i**

**System.out.println(i); // i is 65**

**Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used. For example, since the Unicode of 'a' is 97****, which is within the range of a byte, these implicit castings are fine:**

**byte b = 'a';**

**int i = 'a';**

**But the following casting is incorrect, because the Unicode \uFFF4 cannot fit into a byte:**

**byte b = '\uFFF4';**

**More examples:**

**int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51**

**System.out.println("i is " + i);**

**int j = 2 + 'a'; // (int)'a' is 97**

**System.out.println("j is " + j);**

**System.out.println(j + " is the Unicode for character " + (char)j);**

**System.out.println("Chapter" + '2');**

**display**

**i is 101**

**j is 99**

**99 is the Unicode for character c**

**Chapter 2**

**2.11. Getting Input from Input Dialogs**

****

#### 2.11.1. Converting Strings to Numbers

**The input returned from the input dialog box is a string. SoYou have to convert a string into a number to obtain the input as a number.**

**To convert a string into an int value, use the parseInt method in the Integer class, as follows:**

**int intValue = Integer.parseInt(intString);**

**To convert a string into a double value, use the parseDouble method in the Double class, as follows:**

**double doubleValue = Double.parseDouble(doubleString);**

**where doubleString is a numeric string such as "123.45".**

**The Integer and Double classes are both included in the java.lang package, and thus are automatically imported**

**2.12.1. Example: Computing Loan Payments**

**Quickly write the following program and test it.**

****

**Here are the algorithm in developing the program:**

1. **Prompt the user to enter the annual interest rate, number of years, and loan amount.**
2. **Obtain the monthly interest rate from the annual interest rate.**
3. **Compute the monthly payment using the preceding formula.**
4. **Compute the total payment, which is the monthly payment multiplied by 12 and multiplied by the number of years.**
5. **Display the monthly payment and total payment in a message dialog**

**Listing 2.6. ComputeLoan.java**

|  |
| --- |
|  **1 import javax.swing.JOptionPane;** **2** **3 public class ComputeLoan {** **4 /\*\* Main method \*/** **5 public static void main(String[] args) {** **6 // Enter yearly interest rate** **7 String annualInterestRateString = JOptionPane.showInputDialog(** **8 "Enter yearly interest rate, for example 8.25:");** **9****10 // Convert string to double****11 double annualInterestRate =****12 Double.parseDouble(annualInterestRateString);****13****14 // Obtain monthly interest rate****15 double monthlyInterestRate = annualInterestRate / 1200;****16****17 // Enter number of years****18 String numberOfYearsString = JOptionPane.showInputDialog(****19 "Enter number of years as an integer, \nfor example 5:");****20****21 // Convert string to int****22 int numberOfYears = Integer.parseInt(numberOfYearsString);****23****24 // Enter loan amount****25 String loanString = JOptionPane.showInputDialog(****26 "Enter loan amount, for example 120000.95:");****27****[Page 48]****28 // Convert string to double****29 double loanAmount = Double.parseDouble(loanString);****30****31 // Calculate payment****32 double monthlyPayment = loanAmount \* monthlyInterestRate / (1****33 – 1 / Math.pow(1 + monthlyInterestRate, numberOfYears \* 12));****34 double totalPayment = monthlyPayment \* numberOfYears \* 12;****35****36 // Format to keep two digits after the decimal point****37 monthlyPayment = (int)(monthlyPayment \* 100) / 100.0;****38 totalPayment = (int)(totalPayment \* 100) / 100.0;****39****40 // Display results****41 String output = "The monthly payment is " + monthlyPayment +****42 "\nThe total payment is " + totalPayment;****43 JOptionPane.showMessageDialog(null, output);****44 }****45 }** |

**3.2. boolean Data Type and Operations**

| **Table 3.1. Comparison Operators** |
| --- |
| **Operator** | **Name** | **Example** | **Answer** |
| **<** | **less than** | **1 < 2** | **true** |
| **<=** | **less than or equal to** | **1 <= 2** | **true** |
| **>** | **greater than** | **1 > 2** | **false** |
| **>=** | **greater than or equal to** | **1 >= 2** | **false** |
| **==** | **equal to** | **1 == 2** | **false** |
| **!=** | **not equal to** | **1 != 2** | **true** |

| **Table 3.2. Boolean Operators** |
| --- |
| **Operator** | **Name** | **Description** |
| **!** | **not** | **logical negation** |
| **&&** | **and** | **logical conjunction** |
| **||** | **or** | **logical disjunction** |
| **^** | **exclusive or** | **logical exclusion** |

| **Table 3.3. Truth Table for Operator !** |
| --- |
| **p** | **!p** | **Example** |
| **true** | **false** | **!(1 > 2) is true, because (1 > 2) is false.** |
| **false** | **true** | **!(1 > 0) is false, because (1 > 0) is true.** |

| **Table 3.4. Truth Table for Operator &&** |
| --- |
| **p1** | **p2** | **p1 && p2** | **Example** |
| **false** | **false** | **false** | **(2 > 3) && (5 > 5) is false, because either (2 > 3) or (5 > 5) is false.** |
| **false** | **true** | **false** |  |
| **true** | **false** | **false** | **(3 > 2) && (5 > 5) is false, because (5 > 5) is false.** |  |
| **true** | **true** | **true** | **(3 > 2) && (5 >= 5) is true, because (3 > 2) and (5 >= 5) are both true.** |  |

| **Table 3.5. Truth Table for Operator ||** |
| --- |
| **p1** | **p2** | **p1 || p2** | **Example** |
| **false** | **false** | **false** | **(2 > 3)||(5 > 5) is false, because (2 > 3) and (5 > 5) are both false.** |
| **false** | **true** | **true** |  |
| **true** | **false** | **true** | **(3 > 2)||(5 > 5) is true, because (3 > 2) is true.** |  |
| **true** | **true** | **true** |  |

**The exclusive or (^) of two Boolean operands is true** **if and only if the two operands have different Boolean values.**

| **Table 3.6. Truth Table for Operator ^** |
| --- |
| **p1** | **p2** | **p1^p2** | **Example** |
| **false** | **false** | **false** | **(2 > 3)^(5 > 1) is true, because (2 > 3) is false and (5 > 1) is true.** |
| **false** | **true** | **true** |  |
| **true** | **false** | **true** | **(3 > 2)^(5 > 1) is false, because both (3 > 2) and (5 > 1) are true.** |  |
|  |  |  |  |

**a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3****, and whether a number is divisible by 2 or 3 but not both:**

**Listing 3.1. TestBoolean.java**

|  |
| --- |
|  **1 import javax.swin ptionPane;** **2** **3 public class TestBoolean {** **4 public static void main(String[] args) {** **5 int number = 18;** **6** **7 JOptionPane.showMessageDialog(null,** **8 "Is " + number +** **9 "\n divisible by 2 and 3? " +****10 (number % 2 == 0 && number % 3 == 0)****11 + "\n divisible by 2 or 3? " +****12 (number % 2 == 0 || number % 3 == 0) +****13 "\n divisible by 2 or 3, but not both? "****14 + (number % 2 == 0 ^ number % 3 == 0));****15 }****16 }** |

#### 3.2.2. Example: Determining Leap Year

**This section presents a program that lets the user enter a year in a dialog box and checks whether it is a leap year.**

**A year is a leap year if it is divisible by 4 but not by 100 or if it is divisible by 400****. So :**

**(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)**

**Listing 3.2. LeapYear.java**

|  |
| --- |
| **1 import javax.swing.JOptionPane;** **2** **3 public class LeapYear {** **4 public static void main(String args[]) {** **5 // Prompt the user to enter a year****6 String yearString = JOptionPane.showInputDialog("Enter a year");** **7** **8 // Convert the string into an int value** **9 int year = Integer.parseInt(yearString);****10****11 // Check if the year is a leap year****12 boolean isLeapYear =****13 (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);****14****15 // Display the result in a message dialog box****16 JOptionPane.showMessageDialog(null,****17 year + " is a leap year? " + isLeapYear);****18 }****19 }** |

**3.2.3. Example: A Simple Math Learning Tool :The program randomly generates two single-digit integers number1 and number2**

**Listing 3.3. AdditionTutor.java**

|  |
| --- |
|  **1 import javax.swing.\*;** **2** **3 public class AdditionTutor {** **4 public static void main(String[] args) {** **5 int number1 = (int)(System.currentTimeMillis() % 10);** **6 int number2 = (int)(System.currentTimeMillis() \* 7 % 10);** **7** **8 String answerString = JOptionPane.showInputDialog** **9 ("What is " + number1 + " + " + number2 + "?");****10****11 int answer = Integer.parseInt(answerString);****12****13 JOptionPane.showMessageDialog(null,** **number1 + " + " + number2 + " = " + answer + " is " +(number1 + number2 == answer));****16 }****17 }** |

**3.3. if Statements**

**if (radius >= 0) {**

 **area = radius \* radius \* PI;**

 **System.out.println("The area for the circle of radius " +**

 **radius + " is " + area);**

**}**

**The booleanExpression is enclosed in parentheses for all forms of the if** **statement.**

****

**The following statement determines whether a number is even or odd:**

**// Prompt the user to enter an integer**

**String intString = JOptionPane.showInputDialog(**

 **"Enter an integer:");**

**// Convert string into int**

**int number = Integer.parseInt(intString);**

**if (number % 2 == 0)**

 **System.out.println(number + " is even.");**

**if (number % 2 != 0)**

 **System.out.println(number + " is odd.");**

**3.3.2. if ... else Statements**

**if (radius >= 0) {**

 **area = radius \* radius \* PI;**

 **System.out.println("The area for the circle of radius " +**

 **radius + " is " + area);**

**}**

**else {**

 **System.out.println("Negative input");**

**}**

**3.3.3. Nested if Statements**

**if (i > k) {**

 **if (j > k)**

 **System.out.println("i and j are greater than k");**

**}**

**else**

 **System.out.println("i is less than or equal to k");**

**Often new programmers write the code that assigns a test condition to a boolean** **variable like the code in (a):**

****

**To test whether a boolean variable is true or false** **in a test condition, it is redundant to use the equality comparison operator like the code in (a):**

****

**3.3.4. Example: Computing Taxes**

| **Table 3.7. 2002 U.S. Federal Personal Tax Rates** |
| --- |
| **Tax rate** | **Single filers** | **Married filing jointly or qualifying widow/widower** | **Married filing separately** | **Head of household** |
| **10%** | **Up to $6,000** | **Up to $12,000** | **Up to $6,000** | **Up to $10,000** |
| **15%** | **$6,001–$27,950** | **$12,001–$46,700** | **$6,001–$23,350** | **$10,001–$37,450** |
| **27%** | **$27,951–$67,700** | **$46,701–$112,850** | **$23,351–$56,425** | **$37,451–$96,700** |
| **30%** | **$67,701–$141,250** | **$112,851–$171,950** | **$56,426–$85,975** | **$96,701–$156,600** |
| **35%** | **$141,251–$307,050** | **$171,951–$307,050** | **$85,976–$153,525** | **$156,601–$307,050** |
| **38.6%** | **$307,051 or more** | **$307,051 or more** | **$153,526 or more** | **$307,051 or more** |

**Your program computes the tax for the taxable income based on the filing status. The filing status can be determined using if statements outlined as follows**

**Listing 3.4. ComputeTaxWithSelectionStatement.java**

**1 import javax.swing.JOptionPane;**

 **2**

 **3 public class ComputeTaxWithSelectionStatement {**

 **4 public static void main(String[] args) {**

 **5 // Prompt the user to enter filing status**

 **6 String statusString = JOptionPane.showInputDialog(**

 **7 "Enter the filing status:\n" +**

 **8 "(0-single filer, 1-married jointly,\n" +**

 **9 "2-married separately, 3-head of household)");**

**10 int status = Integer.parseInt(statusString);**

**11**

**12 // Prompt the user to enter taxable income**

**13 String incomeString = JOptionPane.showInputDialog(**

**14 "Enter the taxable income:");**

**15 double income = Double.parseDouble(incomeString);**

**16**

**17 // Compute tax**

**18 double tax = 0;**

**19**

**20 if (status == 0) { // Compute tax for single filers**

**21 if (income <= 6000)**

**22 tax = income \* 0.10;**

**23 else if (income <= 27950)**

**24 tax = 6000 \* 0.10 + (income - 6000) \* 0.15;**

**25 else if (income <= 67700)**

**26 tax = 6000 \* 0.10 + (27950 - 6000) \* 0.15 +**

**27 (income - 27950) \* 0.27;**

**28 else if (income <= 141250)**

**29 tax = 6000 \* 0.10 + (27950 - 6000) \* 0.15 +**

**30 (67700 - 27950) \* 0.27 + (income - 67700) \* 0.30;**

**31 else if (income <= 307050)**

**32 tax = 6000 \* 0.10 + (27950 - 6000) \* 0.15 +**

**33 (67700 - 27950) \* 0.27 + (141250 - 67700) \* 0.30 +**

**34 (income - 141250) \* 0.35;**

**35 else**

**36 tax = 6000 \* 0.10 + (27950 - 6000) \* 0.15 +**

**37 (67700 - 27950) \* 0.27 + (141250 - 67700) \* 0.30 +**

**38 (307050 - 141250) \* 0.35 + (income - 307050) \* 0.386;**

**39 }**

**40 else if (status == 1) { // Compute tax for married file jointly**

**41 // Left as exercise**

**42 }**

**43 else if (status == 2) { // Compute tax for married separately**

**44 // Left as exercise**

**45 }**

**46 else if (status == 3) { // Compute tax for head of household**

**47 // Left as exercise**

**48 }**

**49 else {**

**50 System.out.println("Error: invalid status");**

**51 System.exit(0);**

**52 }**

**53**

**54 // Display the result**

**55 JOptionPane.showMessageDialog(null, "Tax is " +**

**56 (int)(tax \* 100) / 100.0);**

**57 }**

**58 }**

**3.6. Formatting Console Output and Strings**

| **Table 3.8. Frequently Used Specifiers** |
| --- |
| **Specifier** | **Output** | **Example** |
| **%b** | **a boolean value** | **true or false** |
| **%c** | **a character** | **'a'** |
| **%d** | **a decimal integer** | **200** |
| **%f** | **a floating-point number** | **45.460000** |
| **%e** | **a number in standard scientific notation** | **4.556000e+01** |
| **%s** | **a string** | **"Java is cool"** |

| **Table 3.9. Examples of Specifying Width and Precision** |
| --- |
| **Example** | **Output** |
| **%5c** | **Output the character and add four spaces before the character item.** |
| **%6b** | **Output the boolean value and add one space before the false value and two spaces before the true value.** |
| **%5d** | **Output the integer item with width at least 5****. If the number of digits in the item is <5****, add spaces before the number. If the number of digits in the item is >5****, the width is automatically increased.** |
| **%10.2f** | **Output the floating-point item with width at least 10** **including a decimal point and two digits after the point. Thus there are 7** **digits allocated before the decimal point. If the number of digits before the decimal in the item is <7****, add spaces before the number. If the number of digits before the decimal in the item is >7, the width is automatically increased.** |
| **%10.2e** | **Output the floating-point item with width at least 10** **including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10****, add spaces before the number.** |
| **%12s** | **Output the string with width at least 12 characters. If the string item has less than 12** **characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.** |

**You can put the minus sign (–****) in the specifier to specify that the item is left-justified in the output within the specified field. For example, the following statement**

**System.out.printf("%8d%-8s\n", 1234, "Java");**

**System.out.printf("%-8d%-8s\n", 1234, "Java");**

**displays**

 **1234Java**

**1234 Java**

**To display formatted output in a message dialog box use the static format** **method in the String** **class to create a formatted string**

**String s = String.format("count is %d and amount is %f", 5, 45.56));**

**creates a formatted string "count is 5 and amount is 45.560000".**

**JOptionPane.showMessageDialog(null, String.format("Sales tax is %1.2f", 24.3454));**

****

**Chapter 4. Loops**

**4.2. The while Loop**

**int count = 0;**

**while (count < 100) {**

 **System.out.println("Welcome to Java!");**

 **count++;**

**}**

#### 4.2.1. Example: An Advanced Math Learning Tool

**a program that generates ten questions and reports the number of correct answers after a student answers all ten questions. The program also displays the time spent on the test and lists all the questions**

**Listing 4.1. SubtractionTutorLoop.java**

|  |
| --- |
| **1 import javax.swing.JOptionPane;****3 public class SubtractionTutorLoop {****4 public static void main(String[] args) {****5 int correctCount = 0; // Count the number of correct answers****6 int count = 0; // Count the number of questions****7 long startTime = System.currentTimeMillis();****8 String output = "";**10 while (count < 10) {11 // 1. Generate two random single-digit integers12 int number1 = (int)(Math.random() \* 10);13 int number2 = (int)(Math.random() \* 10);15 // 2. If number1 < number2, swap number1 with number216 if (number1 < number2) {17 int temp = number1;18 number1 = number2;19 number2 = temp;20 } String answerString = JOptionPane.showInputDialog24 ("What is " + number1 + " - " + number2 + "?");25 int answer = Integer.parseInt(answerString);27 // 4. Grade the answer and display the result28 String replyString;29 if (number1 - number2 == answer) {30 replyString = "You are correct!";31 correctCount++;32 }33 else34 replyString = "Your answer is wrong.\n" + number1 + " - "35 + number2 + " should be " + (number1 - number2);36 JOptionPane.showMessageDialog(null, replyString);38 // Increase the count39 count++;41 output += "\n" + number1 + "-" + number2 + "=" + answerString +42 ((number1 - number2 == answer) ? " correct" : " wrong");43 }45 long endTime = System.currentTimeMillis();46 long testTime = endTime - startTime;48 JOptionPane.showMessageDialog(null,49 "Correct count is " + correctCount + "\nTest time is " +50 testTime / 1000 + " seconds\n" + output);51 }52 } |

****

**4.2.2. Controlling a Loop with a Confirmation Dialog**

**int option = 0;**

**while (option == JOptionPane.YES\_OPTION) {**

 **System.out.println("continue loop");**

 **option = JOptionPane.showConfirmDialog(null, "Continue?");**

**}**

****

**Question: rewrite** **Listing 4.1** **using a confirmation dialog to let the user decide whether to continue the next question.**

**4.2.3. Controlling a Loop with a Sentinel Value**

**Listing 4.2. SentinelValue.java**

|  |
| --- |
|  **1 import javax.swing.JOptionPane;** **2** **3 public class SentinelValue {** **4 /\*\* Main method \*/** **5 public static void main(String[] args) {** **6 // Read an initial data** **7 String dataString = JOptionPane.showInputDialog(** **8 "Enter an int value:\n(the program exits if the input is 0)");** **9 int data = Integer.parseInt(dataString);****10****11 // Keep reading data until the input is 0****12 int sum = 0;****13 while (data != 0) {****14 sum += data;****15****16 // Read the next data****17 dataString = JOptionPane.showInputDialog(****18 "Enter an int value:\n(the program exits if the input is 0)");****19 data = Integer.parseInt(dataString);****20 }****21****22 JOptionPane.showMessageDialog(null, "The sum is " + sum);****23 }****24 }** |

**4.3. The do-while Loop**

**The do-while loop is a variation of the while loop. Its syntax is given below:**

**Listing 4.3. TestDo.java**

|  |
| --- |
|  **1 import javax.swing.JOptionPane;** **2** **3 public class TestDoWhile {** **4 /\*\* Main method \*/** **5 public static void main(String[] args) {** **6 int data;** **7 int sum = 0;** **8** **9 // Keep reading data until the input is 0****10 do {****11 // Read the next data****12 String dataString = JOptionPane.showInputDialog(null,****13 "Enter an int value:\n(the program exits if the input is 0)",****14 "TestDo", JOptionPane.QUESTION\_MESSAGE);****15****16 data = Integer.parseInt(dataString);****17****18 sum += data;****19 } while (data != 0);****20****21 JOptionPane.showMessageDialog(null, "The sum is " + sum,****22 "TestDo", JOptionPane.INFORMATION\_MESSAGE);****23 }****24 }** |

**4.4. The for Loop**

**int i;**

**for (i = 0; i < 100; i++) {**

 **System.out.println("Welcome to Java!");**

**}**

**The initial-action in a for** **loop can be a list of zero or more comma-separated variable declaration statements or assignment expressions. For example,**

**for (int i = 0, j = 0; (i + j < 10); i++, j++) {**

 **// Do something**

**}**

**If the loop-continuation-condition in a for loop is omitted, it is implicitly true****. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:**

****

**Chapter 5. Methods**

**A method is a collection of statements that are grouped together to perform an operation**

****

**The returnValueType** **is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword void**

**A return statement using the keyword return is required** **for a nonvoid method to return a result**

**The method name** **and the parameter list together constitute the** ***method signature*****. Parameters are optional; that is, a method may contain no parameters.**

**5.3. Calling a Method**

 **For non-void methods, the example to call a method is**

 **int larger = max(3, 4);**

 **For void methods,**

 **System.out.println("Welcome to Java!");**

**Listing 5.1. TestMax.java**

|  |
| --- |
|  **1 public class TestMax {** **2 /\*\* Main method \*/** **3 public static void main(String[] args) {** **4 int i = 5;** **5 int j = 2;** **6 int k = max(i, j);** **7 System.out.println("The maximum between " + i +** **8 " and " + j + " is " + k);** **9 }****10****11 /\*\* Return the max between two numbers \*/****12 public static int max(int num1, int num2) {****13 int result;****14****15 if (num1 > num2)****16 result = num1;****17 else****18 result = num2;****19****20 return result;** **21 }****22 }** |

**Figure 5.3. When the max** **method is invoked, the flow of control transfers to the max method. Once the max** **method is finished, it returns the control back to the caller.**

****

**The statements in main** **may invoke other methods that are defined in the class that contains the main** **method or in other classes**

**One of the benefits of methods is for reuse.**

* **The max method can be invoked from any class besides TestMax.**
* **If you create a new class, Test, you can invoke the max method using ClassName.methodName (i.e., TestMax.max).**

**5.4. void Method Example**

**How to declare and invoke a void method**

**Listing 5.2. TestVoidMethod.java**

|  |
| --- |
|  **1 public class TestVoidMethod {** **2 public static void main(String[] args) {** **3 printGrade(78.5);** **4 }** **5** **6 public static void printGrade(double score) {** **7 if (score >= 90.0) {** **8 System.out.println('A');** **9 }****10 else if (score >= 80.0) {****11 System.out.println('B');****12 }****13 else if (score >= 70.0) {****14 System.out.println('C');****15 }****16 else if (score >= 60.0) {****17 System.out.println('D');****18 }****19 else {****20 System.out.println('F');****21 }****22 }****23 }** |

**A return statement is not needed for a void** **method, but it can be used for terminating the method and returning to the method's caller.**

**For example, the following code has a return statement to terminate the function when the score is invalid.**

**public static void printGrade(double score) {**

 **if (score < 0 || score > 100)**

 **System.out.println("Invalid score");**

 **return;**

 **}**

 **if (score >= 90.0) {**

 **System.out.println('A');**

 **}**

 **else if (score >= 80.0) {**

 **System.out.println('B');**

 **}**

 **else if (score >= 70.0) {**

 **System.out.println('C');**

 **}**

 **else if (score >= 60.0) {**

 **System.out.println('D');**

 **}**

 **else {**

 **System.out.println('F');**

 **}**

**}**

**5.5. Passing Parameters by Values**

**When you invoke a method with a parameter, the value of the argument is passed to the parameter.**

* **This is referred to as** ***pass-by-value*****.**
* **If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter.**
* **The variable is not affected, regardless of the changes made to the parameter inside the method.**
* **Listing 5.3. TestPassByValue.java**

|  |
| --- |
|  **1 public class TestPassByValue {** **3 public static void main(String[] args) {** **5 int num1 = 1;** **6 int num2 = 2;** **7** **8 System.out.println("Before invoking the swap method, num1 is " +** **9 num1 + " and num2 is " + num2);****10****12 swap(num1, num2);****13****14 System.out.println("After invoking the swap method, num1 is " +****15 num1 + " and num2 is " + num2);****16 }****17****18 /\*\* Swap two variables \*/****19 public static void swap(int n1, int n2) {****20 System.out.println("\tInside the swap method");****21 System.out.println("\t\tBefore swapping n1 is " + n1****22 + " n2 is " + n2);****24 // Swap n1 with n2****25 int temp = n1;****26 n1 = n2;****27 n2 = temp;****28****29 System.out.println("\t\tAfter swapping n1 is " + n1****30 + " n2 is " + n2);****31 }****32 }** |

* **it makes no difference whether the parameter and the argument have the same name**

**5.6. Overloading Methods**

* **The max method that was used earlier works only with the int** **data type.**
* **But what if you need to find which of two floating-point numbers has the maximum value?**
* **The solution is to create another method with the same name but different parameters, as shown in the following code:**

**public static double max(double num1, double num2) {**

 **if (num1 > num2)**

 **return num1;**

 **else**

 **return num2;**

**}**

* **If you call max with int parameters, the max method that expects int parameters will be invoked;**
* **if you call max with double parameters, the max method that expects double** **parameters will be invoked.**
* **This is referred to as** ***method overloading*****; that is, two methods have the same name but different parameter lists within one class.**
* **The Java compiler determines which method is used based on the method signature**

**Listing 5.4. TestMethodOverloading.java**

|  |
| --- |
| **1 public class TestMethodOverloading {** **2 /\*\* Main method \*/** **3 public static void main(String[] args) {** **4 // Invoke the max method with int parameters** **5 System.out.println("The maximum between 3 and 4 is "** **6 + max(3, 4));** **7** **8 // Invoke the max method with the double parameters** **9 System.out.println("The maximum between 3.0 and 5.4 is "****10 + max(3.0, 5.4));****11****12 // Invoke the max method with three double parameters****13 System.out.println("The maximum between 3.0, 5.4, and 10.14 is "****14 + max(3.0, 5.4, 10.14));****15 }****16****17 /\*\* Return the max between two int values \*/****18 public static int max(int num1, int num2) {****19 if (num1 > num2)****20 return num1;****21 else****22 return num2;****23 }****24****25 /\*\* Find the max between two double values \*/****26 public static double max(double num1, double num2) {****27 if (num1 > num2)****28 return num1;****29 else****30 return num2;****31 }****32****33 /\*\* Return the max among three double values \*/****public static double max(double num1, double num2, double num3) {** **return max(max(num1, num2), num3);** **}****}** |

**Can you invoke the max method with an int value and a double value, such as max(2, 2.5)? If so, which of the max** **methods is invoked?**

* **Overloaded methods must have different parameter lists. You cannot overload methods based on different modifiers or return types**
* **Avoid ambiguous overloading such as this**
* **Both max(int, double) and max(double, int) are possible candidates to match max(1, 2)****.**
* **Since neither of them is more specific than the other, the invocation is ambiguous, resulting in a compilation error.**

**5.7. Case Study: Computing Taxes with Methods---Listing 5.5. ComputeTaxWithMethod.java**

|  |
| --- |
|  **1 import javax.swing.JOptionPane;** **2** **3 public class ComputeTaxWithMethod {** **4 public static void main(String[] args) {** **5 // Prompt the user to enter filing status** **6 String statusString = JOptionPane.showInputDialog(** **7 "Enter the filing status:");** **8 int status = Integer.parseInt(statusString);****11 String incomeString = JOptionPane.showInputDialog(****12 "Enter the taxable income:");****13 double income = Double.parseDouble(incomeString);****16 JOptionPane.showMessageDialog(null, "Tax is " +****17 (int)(computeTax(status, income) \* 100) / 100.0);****18 }****19****20 public static double computeTax(double income,****21 int r1, int r2, int r3, int r4, int r5) {****22 double tax = 0;****24 if (income <= r1)****25 tax = income \* 0.10;****26 else if (income <= r2)****27 tax = r1 \* 0.10 + (income - r1) \* 0.15;****28 else if (income <= r3)****29 tax = r1 \* 0.10 + (r2 - r1) \* 0.15 + (income - r2) \* 0.27;****30 else if (income <= r4)****31 tax = r1 \* 0.10 + (r2 - r1) \* 0.15 +****32 (r3 - r2) \* 0.27 + (income - r3) \* 0.30;****33 else if (income <= r5)****34 tax = r1 \* 0.10 + (r2 - r1) \* 0.15 + (r3 - r2) \* 0.27 +****35 (r4 - r3) \* 0.30 + (income - r4) \* 0.35;****36 else****37 tax = r1 \* 0.10 + (r2 - r1) \* 0.15 + (r3 - r2) \* 0.27 +****38 (r4 - r3) \* 0.30 + (r5 - r4) \* 0.35 + (income - r5) \* 0.386;****40 return tax;****41 }****42****43 public static double computeTax(int status, double income) {****44 switch (status) {****45 case 0: return****46 computeTax(income, 6000, 27950, 67700, 141250, 307050);****47 case 1: return****48 computeTax(income, 12000, 46700, 112850, 171950, 307050);****49 case 2: return****50 computeTax(income, 6000, 23350, 56425, 85975, 153525);****51 case 3: return****52 computeTax(income, 10000, 37450, 96700, 156600, 307050);****53 default: return 0;****54 }****55 }****56 }** |

**6.2. Array Basics**

**The following code snippets are examples of this syntax:**

**double[] myList;**

**or**

**double myList[]; // This style is allowed, but not preferred**

**6.2.2. Creating Arrays**

**Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:**

**dataType[] arrayRefVar = new dataType[arraySize];**

**or**

**dataType arrayRefVar[] = new dataType[arraySize];**

**Here is an example of such a statement:**

**double[] myList = new double[10];**

#### 6.2.3. Array Size and Default Values

**The size of an array cannot be changed after the array is created. Size can be obtained using arrayRefVar.length. For example, myList.length is 10.**

**When an array is created, its elements are assigned the default value of**

* **0 for the numeric primitive data types,**
* **'\u0000' for char types, and**
* **false for boolean types.**

####

#### 6.2.4. Array Indexed Variables

**The array elements are accessed through the index. Array indices are 0-based****; that is, they start from 0 to arrayRefVar.length-1.**

**Each element in the array is represented using the following syntax, known as an** **indexed variable:**

**arrayRefVar[index];**

**The following loop assigns 0 to myList[0], 1 to myList[1], and 9 to myList[9]:**

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

 **myList[i] = i;**

**}**

#### 6.2.5. Array Initializers

**Java has a shorthand notation, known as the** ***array initializer*****, which combines declaring an array, creating an array, and initializing in one statement using the following syntax:**

**dataType[] arrayRefVar = {value0, value1, ..., valuek};**

**For example,**

**double[] myList = {1.9, 2.9, 3.4, 3.5};**

**This statement declares, creates, and initializes the array myList** **with four elements, which is equivalent to the statements shown below:**

**double[] myList = new double[4];**

**myList[0] = 1.9;**

**myList[1] = 2.9;**

**myList[2] = 3.4;**

**myList[3] = 3.5;**

**The new** **operator is not used in the array initializer syntax.**

**Thus the next statement is wrong:**

**double[] myList;**

**myList = {1.9, 2.9, 3.4, 3.5};**

6.2.6. Processing Arrays

* The following loop initializes the array myList with random values between 0.0 and 99.0:

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

 myList[i] = Math.random() \* 100;

}

* To print an array, you have to print each element in the array using a loop like the one shown below.

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

 System.out.print(myList[i] + " ");

}

* **Char arrays**

char[] city = {'D', 'a', 'l', 'l', 'a', 's'};

System.out.println(city);

* **Or you can print using** one print statement

char[] city = {'D', 'a', 'l', 'l', 'a', 's'};

System.out.println(city);

* **Summing all elements)**

double total = 0;

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

 total += myList[i];

}

**Finding the largest element**

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

 if (myList[i] > max) max = myList[i];

}

**Finding the smallest index of the largest element**

Suppose the array myList is {1, 5, 3, 4, 5, 5}.

double max = myList[0];

int indexOfMax = 0;

for (int i = 1; i < myList.length; i++) {

 if (myList[i] > max) {

 max = myList[i];

 indexOfMax = i;

 }

}

What is the consequence if (myList[i] > max) is replaced by (myList[i] >= max)?

#### 6.2.7. foreach Loops

Enables you to traverse the complete array sequentially without using an index variable.

for (double element: myList) {

 System.out.println(element);

}

* the variable, element, must be declared the same type as the elements in myList

**6.2.8. Example: Testing Arrays**

read six integers, finds the largest of them, and counts its occurrences

 **1 import javax.swing.JOptionPane;**

 **2**

 **3 public class TestArray {**

 **4 /\*\* Main method \*/**

 **5 public static void main(String[] args) {**

 **6 final int TOTAL NUMBERS = 6;**

 **7 int[] numbers = new int[TOTAL NUMBERS];**

**9 // Read all numbers**

**10 for (int i = 0; i < numbers.length; i++) {**

**11 String numString = JOptionPane.showInputDialog(**

**12 "Enter a number:");**

**14 // Convert string into integer**

**15 numbers[i] = Integer.parseInt(numString);**

**16 }**

**18 // Find the largest**

**19 int max = numbers[0];**

**20 for (int i = 1; i < numbers.length; i++) {**

**21 if (max < numbers[i])**

**22 max = numbers[i];**

**23 }**

**25 // Find the occurrence of the largest number**

**26 int count = 0;**

**27 for (int i = 0; i < numbers.length; i++) {**

**28 if (numbers[i] == max) count++;**

**29 }**

**31 // Prepare the result**

**32 String output = "The array is ";**

**33 for (int i = 0; i < numbers.length; i++) {**

**34 output += numbers[i] + " ";**

**35 }**

**36**

**37 output += "\nThe largest number is " + max;**

**38 output += "\nThe occurrence count of the largest number "**

**39 + "is " + count;**

**40**

**41 // Display the result**

**42 JOptionPane.showMessageDialog(null, output);**

**43 }**

**44 }**

**6.2.9. Example: Assigning Grades**

This example writes a program that reads student scores, gets the best score, and then assigns grades based on the following scheme:

Grade is A if score is > = best - 10;

Grade is B if score is > = best - 20;

Grade is C if score is > = best - 30;

Grade is D if score is > = best - 40;

Grade is F otherwise.

**Listing 6.2. AssignGrade.java**

|  |
| --- |
| 1 import javax.swing.JOptionPane; 2 3 public class AssignGrade { 4 /\*\* Main method \*/ 5 public static void main(String[] args) { 6 // Get number of students 7 **String numberOfStudentsString = JOptionPane.showInputDialog(** 8 "Please enter number of students:"); 10 // Convert string into integer11 **int numberOfStudents = Integer.parseInt(numberOfStudentsString);**1213 int[] scores = new int[numberOfStudents]; int best = 0; // The best score15 char grade; // The grade1617 // Read scores and find the best score18 for (int i = 0; i < scores.length; i++) {19 String scoreString = JOptionPane.showInputDialog(20 "Please enter a score:");22 // Convert string into integer23 scores[i] = Integer.parseInt(scoreString);24 if (scores[i] > best)25 best = scores[i];26 }2729 String output = "";31 // Assign and display grades32 for (int i = 0; i < scores.length; i++) {33 if (scores[i] >= best - 10)34 grade = 'A';35 else if (scores[i] >= best - 20)36 grade = 'B';37 else if (scores[i] >= best - 30)38 grade = 'C';39 else if (scores[i] >= best - 40)40 grade = 'D';41 else42 grade = 'F';4344 output += "Student " + i + " score is " +45 scores[i] + " and grade is " + grade + "\n";46 }4748 // Display the result49 JOptionPane.showMessageDialog(null, output);50 }51 } |

**6.3. Copying Arrays**

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;

This statement does not copy the contents of the array referenced by list1 to list2, but merely copies the reference value from list1 to list2.

There are three ways to copy arrays:

* Use a loop to copy individual elements one by one.
* Use the static arraycopy method in the System class.
* Use the clone method to copy arrays; this will be introduced in Chapter 9, "Inheritance and Polymorphism."

The following code, for instance, copies sourceArray to targetArray using a for loop:

int[] sourceArray = {2, 3, 1, 5, 10};

int[] targetArray = new int[sourceArray.length];

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

 targetArray[i] = sourceArray[i];

}

* Another approach is to use the arraycopy method in the java.lang.System class to copy arrays instead of using a loop. The syntax for arraycopy is shown below:

arraycopy(sourceArray, srcPos, targetArray, tarPos, length);

* The parameters srcPos and tarPos indicate the starting positions in sourceArray and targetArray, respectively. The number of elements copied from sourceArray to targetArray is indicated by length. For example, you can rewrite the loop using the following statement:

**System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);**

Part 2: Object-Oriented Programming

* You learned how to write simple Java applications using
	+ primitive data types,
	+ control statements,
	+ methods, and
	+ arrays, all of which are features commonly available in procedural programming languages.
* Java, however, is an object-oriented programming language that uses
	+ abstraction,
	+ encapsulation,
	+ inheritance, and
	+ polymorphism
		- to provide
			* great flexibility,
			* modularity, and
			* reusability for developing software.
* Now you will learn how to define, extend, and work with classes and their objects.

**7.2. Defining Classes for Objects**

* *Object-oriented programming (OOP)* involves programming using objects.
* An object represents an entity in the real world that can be distinctly identified.
	+ For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
* An object has a unique identity, state, and behaviors.
	+ The state of an object is represented by data fields (also known as properties) with their current values.
	+ The behavior of an object is defined by a set of methods.
* A circle object, for example, has
	+ a data field, radius, which is the property that characterizes a circle.
	+ One behavior of a circle is that its area can be computed using the method getArea().
* Objects of the same type are defined using a common class.
	+ A class is a template or blueprint that defines what an object's data and methods will be.
	+ An object is an instance of a class.
	+ You can create many instances of a class.
	+ Creating an instance is referred to as *instantiation*.
	+ The terms object and *instance* are often interchangeable.
* A Java *class* uses
* variables to define data fields and
* methods to define behaviors.
* Additionally, a class provides methods of a special type, known as
	+ *constructors*,
* which are invoked when a new object is created.
* A constructor is a special kind of method.
* A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects

****

* The Circle class is different from all of the other classes you have seen thus far.
* It does not have a main method and therefore cannot be run;
* it is merely a definition used to declare and create Circle objects.
* For convenience, the class that contains the main method will be referred to as the main class
* The illustration of class templates and objects can be standardized using UML (Unified Modeling Language) notations.
* This notation, is called a UML class diagram, or simply a class diagram.
* For more information on UML, see [www.rational.com/uml/](http://www.rational.com/uml/).
* ****

**7.3. Constructors**

* The constructor has exactly the same name as the defining class.
* Like regular methods, constructors can be overloaded (i.e., multiple constructors with the same name but different signatures),
	+ making it easy to construct objects with different initial data values.
* To construct an object from a class, invoke a constructor of the class using the new operator, as follows:

new ClassName(arguments);

* For example,
	+ new Circle() creates an object of the Circle class using the first constructor defined in the Circle class, and
	+ new Circle(5) creates an object using the second constructor defined in the Circle class.
* A class normally provides a constructor without arguments (e.g., Circle()).
	+ Such a constructor is called a no-arg or no-argument constructor.
* A class may be declared without constructors.
	+ In this case, a no-arg constructor with an empty body is implicitly declared in the class.
	+ This constructor, called *a default constructor*, is provided automatically only if no constructors are explicitly declared in the class.
* Constructors are a **special kind of method**, with three differences:
	+ Constructors must have the same name as the class itself.
	+ Constructors do not have a return type—not even void.
	+ Constructors are invoked using the new operator when an object is created.
	+ Constructors play the role of initializing objects.

### 7.4. Accessing Objects via Reference Variables

* Newly created objects are allocated in the memory. How can they be accessed?

#### 7.4.1. Reference Variables and Reference Types

* Objects are accessed via object *reference variables*, which contain references to the objects. Such variables are declared using the following syntax:

 ClassName objectRefVar;

* A class defines a type, known as a *reference type*.
* Any variable of the class type can reference to an instance of the class.
* The following statement declares the variable myCircle to be of the Circle type:

 Circle myCircle;

* The variable myCircle can reference a Circle object. The next statement creates an object and assigns its reference to myCircle.

myCircle = new Circle();

You can write one statement that combines the declaration of an object reference variable, the creation of an object, and the assigning of an object reference to the variable.

ClassName objectRefVar = new ClassName();

example:

Circle myCircle = new Circle();

The variable myCircle holds a reference to a Circle object.

Note

* Arrays are treated as objects in Java.
* Arrays are created using the new operator.
* An array variable is actually a variable that contains a reference to an array

#### 7.4.2. Accessing an Object's Data and Methods

After an object is created,

* its data can be accessed and
* its methods invoked using
	+ the *dot operator (.*), also known as
	+ the object member access operator:

For example,

* myCircle.radius references the radius in myCircle, and
* myCircle.getArea() invokes the getArea method on myCircle.
* The data field radius is referred to as **an instance variable** because it is dependent on a specific instance.
* For the same reason, the method getArea is referred to as an ***instance method***, because you can only invoke it on a specific instance.
* Most of the time, you create an object and assign it to a variable. Later you can use the variable to reference the object.
* Occasionally, an object does not need to be referenced later. In this case, you can create an object without explicitly assigning it to a variable, as shown below:

new Circle();

or

System.out.println("Area is " + new Circle(5).getArea());

* The former statement creates a Circle object.
* The latter statement creates a Circle object and invokes its getArea method to return its area.
* An object created in this way is known as an **anonymous object.**

7.4.3. Example: Declaring Classes and Creating Objects

**Listing 7.1. TestCircle1.java**

**1 public class TestCircle1**

**2 {**

**3 public static void main(String[] args)**

 **{**

 **4 // Create a circle with radius 5.0**

 **5 Circle1 myCircle = new Circle1(5.0);**

 **6 System.out.println("The area of the circle of radius "**

 **7 + myCircle.radius + " is " + myCircle.getArea());**

**9 // Create a circle with radius 1**

**10 Circle1 yourCircle = new Circle1();**

**11 System.out.println("The area of the circle of radius "**

**12 + yourCircle.radius + " is " + yourCircle.getArea());**

**15 yourCircle.radius = 100;**

**16 System.out.println("The area of the circle of radius "**

**17 + yourCircle.radius + " is " + yourCircle.getArea());**

**18 }**

**19 }**

**21 // Define the circle class with two constructors**

**21 class Circle1**

**22 {**

**23 double radius;**

**25 /\*\* Construct a circle with radius 1 \*/**

**26 Circle1()**

**26 {**

**27 radius = 1.0;**

**28 }**

**30 /\*\* Construct a circle with a specified radius \*/**

**31 Circle1(double newRadius)**

 **{**

**32 radius = newRadius;**

**33 }**

**34**

**35 /\*\* Return the area of this circle \*/**

**36 double getArea()**

 **{**

**37 return radius \* radius \* Math.PI;**

**38 }**

**39 }**

* The program contains two classes. The first class, TestCircle1, is the main class. Its sole purpose is to test the second class, Circle1. Every time you run the program, the JVM invokes the main method in the main class.
* You can put the two classes into one file, **but only one class in the file can be a public class**.
* Furthermore, the public class must have the **same name as the file name** and the **main method must be in a public class**. Therefore, the file name is TestCircle1.java if the TestCircle1 and Circle1 classes are both in the same file.
* The main class contains the main method (line 3) that creates two objects.
	+ The constructor Circle1(5.0) was used to create myCircle with a radius of 5.0 (line 5), and
	+ The constructor Circle1() was used to create yourCircle with a radius of 1.0 (line 10).
* These two objects (referenced by myCircle and yourCircle) have different data but share the same methods. Therefore, you can compute their respective areas by using the getArea() method.
* There are many ways to write Java programs. For instance, you can combine the two classes in the example into one
* **Listing 7.2. Circle1.java**

1 public class Circle1

 {

3 public static void main(String[] args)

 {

5 Circle1 myCircle = new Circle1(5.0);

 6 System.out.println("The area of the circle of radius "

 7 + myCircle.radius + " is " + myCircle.getArea());

10 Circle1 yourCircle = new Circle1();

11 System.out.println("The area of the circle of radius "

12 + yourCircle.radius + " is " + yourCircle.getArea());

14 // Modify circle radius

15 yourCircle.radius = 100;

16 System.out.println("The area of the circle of radius "

17 + yourCircle.radius + " is " + yourCircle.getArea());

18 }

20 double radius;

23 Circle1()

 {

24 radius = 1.0;

25 }

28 Circle1(double newRadius)

 {

29 radius = newRadius;

30 }

33 double getArea()

 {

34 return radius \* radius \* Math.PI;

35 }

36 }

This demonstrates that you can test a class by simply adding a main method in the same class.

* Recall that you use
	+ Math.methodName(arguments) (e.g., Math.pow(3, 2.5)) to invoke a method in the Math class.
	+ Can you invoke
		- getArea() using Circle1.getArea()?

The answer is no.

All the methods in the **Math class are static methods**, which are defined using the static keyword. However, **getArea()** **is an instance method**, and thus **non-static**. It **must be invoked from an object** using objectRefVar.methodName(arguments)

(e.g., myCircle.getArea()).

#### 7.4.4. Reference Data Fields and the null Value

* The data fields can be of reference types.
* For example, the following Student class contains a data field name of the String type. String is a predefined Java class.

class Student {

 String name; // name has default value null

 int age; // age has default value 0

boolean isScienceMajor; // isScienceMajor has default value false

 char gender; // c has default value '\u0000'

}

If a data field of a reference type does not reference any object, the data field holds a special Java value, **null**.

The default value of a data field is

* null for a reference type,
* 0 for a numeric type,
* false for a boolean type, and
* '\u0000' for a char type.

However, Java assigns no default value to a local variable inside a method.

The following code displays the default values of data fields name, age, isScienceMajor, and gender for a Student object:

class Test {

 public static void main(String[] args) {

 Student student = new Student();

 System.out.println("name? " + student.name);

 System.out.println("age? " + student.age);

 System.out.println("isScienceMajor? " +

 student.isScienceMajor);

 System.out.println("gender? " + student.gender);

 }

}

The following code has a compilation error because local variables x and y are not initialized:

class Test

{

 public static void main(String[] args) {

 int x; // x has no default value

 String y; // y has no default value

 System.out.println("x is " + x);

 System.out.println("y is " + y);

 }

}