3. Java can automatically convert a primitive type value to its corresponding wrapper object in the context and vice versa.

4. The BigInteger class is useful for computing and processing integers of any size. The BigDecimal class can be used to compute and process floating-point numbers with any arbitrary precision.

5. A String object is immutable; its contents cannot be changed. To improve efficiency and save memory, the JVM stores two literal strings that have the same character sequence in a unique object. This unique object is called an *interned string object*.

6. A *regular expression* (abbreviated *regex*) is a string that describes a pattern for matching a set of strings. You can match, replace, or split a string by specifying a pattern.

7. The StringBuilder and StringBuffer classes can be used to replace the String class. The String object is immutable, but you can add, insert, or append new contents into StringBuilder and StringBuffer objects. Use String if the string contents do not require any change, and use StringBuilder or StringBuffer if they might change.

Quiz

Answer the quiz for this chapter online at www.cs.armstrong.edu/liang/intro10e/quiz.html.

Programming Exercises

MyProgrammingLab™

Sections 10.2–10.3

*10.1 (*The* Time *class*) Design a class named Time. The class contains:

- The data fields hour, minute, and second that represent a time.
- A no-arg constructor that creates a Time object for the current time. (The values of the data fields will represent the current time.)
- A constructor that constructs a Time object with a specified elapsed time since midnight, January 1, 1970, in milliseconds. (The values of the data fields will represent this time.)
- A constructor that constructs a Time object with the specified hour, minute, and second.
- Three getter methods for the data fields hour, minute, and second, respectively.
- A method named setTime(long elapseTime) that sets a new time for the object using the elapsed time. For example, if the elapsed time is 555550000 milliseconds, the hour is 10, the minute is 19, and the second is 10.

Draw the UML diagram for the class and then implement the class. Write a test program that creates two Time objects (using new Time() and new Time(555550000)) and displays their hour, minute, and second in the format hour:minute:second.

(*Hint*: The first two constructors will extract the hour, minute, and second from the elapsed time. For the no-arg constructor, the current time can be obtained using System.currentTimeMillis(), as shown in Listing 2.7, ShowCurrentTime.java.)

10.2 (*The* `BMI` *class*) Add the following new constructor in the `BMI` class:

```
/** Construct a BMI with the specified name, age, weight,
 * feet, and inches
 */
public BMI(String name, int age, double weight, double feet,
  double inches)
```

10.3 (*The* `MyInteger` *class*) Design a class named `MyInteger`. The class contains:

- An `int` data field named `value` that stores the `int` value represented by this object.
- A constructor that creates a `MyInteger` object for the specified `int` value.
- A getter method that returns the `int` value.
- The methods `isEven()`, `isOdd()`, and `isPrime()` that return `true` if the value in this object is even, odd, or prime, respectively.
- The static methods `isEven(int)`, `isOdd(int)`, and `isPrime(int)` that return `true` if the specified value is even, odd, or prime, respectively.
- The static methods `isEven(MyInteger)`, `isOdd(MyInteger)`, and `isPrime(MyInteger)` that return `true` if the specified value is even, odd, or prime, respectively.
- The methods `equals(int)` and `equals(MyInteger)` that return `true` if the value in this object is equal to the specified value.
- A static method `parseInt(char[])` that converts an array of numeric characters to an `int` value.
- A static method `parseInt(String)` that converts a string into an `int` value.

Draw the UML diagram for the class and then implement the class. Write a client program that tests all methods in the class.

10.4 (*The* `MyPoint` *class*) Design a class named `MyPoint` to represent a point with `x`- and `y`-coordinates. The class contains:

- The data fields `x` and `y` that represent the coordinates with getter methods.
- A no-arg constructor that creates a point (`0`, `0`).
- A constructor that constructs a point with specified coordinates.
- A method named `distance` that returns the distance from this point to a specified point of the `MyPoint` type.
- A method named `distance` that returns the distance from this point to another point with specified `x`- and `y`-coordinates.

VideoNote
The `MyPoint` class

Draw the UML diagram for the class and then implement the class. Write a test program that creates the two points (`0`, `0`) and (`10`, `30.5`) and displays the distance between them.

Sections 10.4–10.8

***10.5** (*Displaying the prime factors*) Write a program that prompts the user to enter a positive integer and displays all its smallest factors in decreasing order. For example, if the integer is `120`, the smallest factors are displayed as `5`, `3`, `2`, `2`, `2`. Use the `StackOfIntegers` class to store the factors (e.g., `2`, `2`, `2`, `3`, `5`) and retrieve and display them in reverse order.

***10.6** (*Displaying the prime numbers*) Write a program that displays all the prime numbers less than `120` in decreasing order. Use the `StackOfIntegers` class to store the prime numbers (e.g., `2`, `3`, `5`, `. . .`) and retrieve and display them in reverse order.

****10.7** (*Game: ATM machine*) Use the **Account** class created in Programming Exercise 9.7 to simulate an ATM machine. Create ten accounts in an array with id **0**, **1**, . . . , **9**, and initial balance $100. The system prompts the user to enter an id. If the id is entered incorrectly, ask the user to enter a correct id. Once an id is accepted, the main menu is displayed as shown in the sample run. You can enter a choice **1** for viewing the current balance, **2** for withdrawing money, **3** for depositing money, and **4** for exiting the main menu. Once you exit, the system will prompt for an id again. Thus, once the system starts, it will not stop.

```
Enter an id: 4 ↵Enter

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 ↵Enter
The balance is 100.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 2 ↵Enter
Enter an amount to withdraw: 3 ↵Enter

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 ↵Enter
The balance is 97.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 3 ↵Enter
Enter an amount to deposit: 10 ↵Enter

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1 ↵Enter
The balance is 107.0

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4 ↵Enter

Enter an id:
```

*****10.8** (*Financial: the* `Tax` *class*) Programming Exercise 8.12 writes a program for computing taxes using arrays. Design a class named `Tax` to contain the following instance data fields:

- `int filingStatus`: One of the four tax-filing statuses: `0`—single filer, `1`—married filing jointly or qualifying widow(er), `2`—married filing separately, and `3`—head of household. Use the public static constants `SINGLE_FILER` (`0`), `MARRIED_JOINTLY_OR_QUALIFYING_WIDOW(ER)` (`1`), `MARRIED_SEPARATELY` (`2`), `HEAD_OF_HOUSEHOLD` (`3`) to represent the statuses.
- `int[][] brackets`: Stores the tax brackets for each filing status.
- `double[] rates`: Stores the tax rates for each bracket.
- `double taxableIncome`: Stores the taxable income.

Provide the getter and setter methods for each data field and the `getTax()` method that returns the tax. Also provide a no-arg constructor and the constructor `Tax(filingStatus, brackets, rates, taxableIncome)`.

Draw the UML diagram for the class and then implement the class. Write a test program that uses the `Tax` class to print the 2001 and 2009 tax tables for taxable income from $50,000 to $60,000 with intervals of $1,000 for all four statuses. The tax rates for the year 2009 were given in Table 3.2. The tax rates for 2001 are shown in Table 10.1.

TABLE 10.1 2001 United States Federal Personal Tax Rates

Tax rate	*Single filers*	*Married filing jointly or qualifying widow(er)*	*Married filing separately*	*Head of household*
15%	Up to $27,050	Up to $45,200	Up to $22,600	Up to $36,250
27.5%	$27,051–$65,550	$45,201–$109,250	$22,601–$54,625	$36,251–$93,650
30.5%	$65,551–$136,750	$109,251–$166,500	$54,626–$83,250	$93,651–$151,650
35.5%	$136,751–$297,350	$166,501–$297,350	$83,251–$148,675	$151,651–$297,350
39.1%	$297,351 or more	$297,351 or more	$ 148,676 or more	$297,351 or more

****10.9** (*The* `Course` *class*) Revise the `Course` class as follows:

- The array size is fixed in Listing 10.6. Improve it to automatically increase the array size by creating a new larger array and copying the contents of the current array to it.
- Implement the `dropStudent` method.
- Add a new method named `clear()` that removes all students from the course.

Write a test program that creates a course, adds three students, removes one, and displays the students in the course.

***10.10** (*The* `Queue` *class*) Section 10.6 gives a class for `Stack`. Design a class named `Queue` for storing integers. Like a stack, a queue holds elements. In a stack, the elements are retrieved in a last-in first-out fashion. In a queue, the elements are retrieved in a first-in first-out fashion. The class contains:

- An `int[]` data field named `elements` that stores the `int` values in the queue.
- A data field named `size` that stores the number of elements in the queue.
- A constructor that creates a `Queue` object with default capacity `8`.
- The method `enqueue(int v)` that adds `v` into the queue.

- The method `dequeue()` that removes and returns the element from the queue.
- The method `empty()` that returns true if the queue is empty.
- The method `getSize()` that returns the size of the queue.

Draw an UML diagram for the class. Implement the class with the initial array size set to 8. The array size will be doubled once the number of the elements exceeds the size. After an element is removed from the beginning of the array, you need to shift all elements in the array one position the left. Write a test program that adds 20 numbers from 1 to 20 into the queue and removes these numbers and displays them.

***10.11** (*Geometry: the* `Circle2D` *class*) Define the `Circle2D` class that contains:

- Two `double` data fields named `x` and `y` that specify the center of the circle with getter methods.
- A data field `radius` with a getter method.
- A no-arg constructor that creates a default circle with (`0`, `0`) for (`x`, `y`) and `1` for `radius`.
- A constructor that creates a circle with the specified `x`, `y`, and `radius`.
- A method `getArea()` that returns the area of the circle.
- A method `getPerimeter()` that returns the perimeter of the circle.
- A method `contains(double x, double y)` that returns `true` if the specified point (`x`, `y`) is inside this circle (see Figure 10.21a).
- A method `contains(Circle2D circle)` that returns `true` if the specified circle is inside this circle (see Figure 10.21b).
- A method `overlaps(Circle2D circle)` that returns `true` if the specified circle overlaps with this circle (see Figure 10.21c).

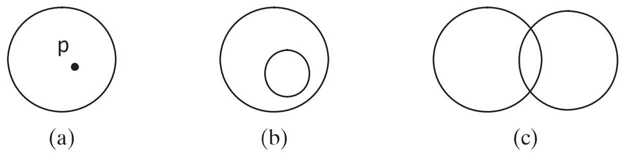


FIGURE 10.21 (a) A point is inside the circle. (b) A circle is inside another circle. (c) A circle overlaps another circle.

Draw the UML diagram for the class and then implement the class. Write a test program that creates a `Circle2D` object `c1` (`new Circle2D(2, 2, 5.5)`), displays its area and perimeter, and displays the result of `c1.contains(3, 3)`, `c1.contains(new Circle2D(4, 5, 10.5))`, and `c1.overlaps(new Circle2D(3, 5, 2.3))`.

*****10.12** (*Geometry: the* `Triangle2D` *class*) Define the `Triangle2D` class that contains:

- Three points named `p1`, `p2`, and `p3` of the type `MyPoint` with getter and setter methods. `MyPoint` is defined in Programming Exercise 10.4.
- A no-arg constructor that creates a default triangle with the points (`0`, `0`), (`1`, `1`), and (`2`, `5`).
- A constructor that creates a triangle with the specified points.
- A method `getArea()` that returns the area of the triangle.
- A method `getPerimeter()` that returns the perimeter of the triangle.
- A method `contains(MyPoint p)` that returns `true` if the specified point `p` is inside this triangle (see Figure 10.22a).

- A method `contains(Triangle2D t)` that returns `true` if the specified triangle is inside this triangle (see Figure 10.22b).
- A method `overlaps(Triangle2D t)` that returns `true` if the specified triangle overlaps with this triangle (see Figure 10.22c).

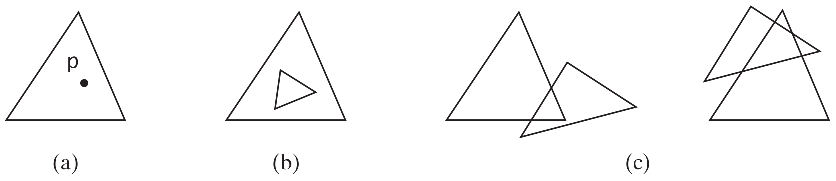


Figure 10.22 (a) A point is inside the triangle. (b) A triangle is inside another triangle. (c) A triangle overlaps another triangle.

Draw the UML diagram for the class and then implement the class. Write a test program that creates a `Triangle2D` objects `t1` using the constructor `new Triangle2D(new MyPoint(2.5, 2), new MyPoint(4.2, 3), new MyPoint(5, 3.5))`, displays its area and perimeter, and displays the result of `t1.contains(3, 3)`, `r1.contains(new Triangle2D(new MyPoint(2.9, 2), new MyPoint(4, 1), MyPoint(1, 3.4)))`, and `t1.overlaps(new Triangle2D(new MyPoint(2, 5.5), new MyPoint(4, -3), MyPoint(2, 6.5)))`.

(*Hint*: For the formula to compute the area of a triangle, see Programming Exercise 2.19. To detect whether a point is inside a triangle, draw three dashed lines, as shown in Figure 10.23. If the point is inside a triangle, each dashed line should intersect a side only once. If a dashed line intersects a side twice, then the point must be outside the triangle. For the algorithm of finding the intersecting point of two lines, see Programming Exercise 3.25.)

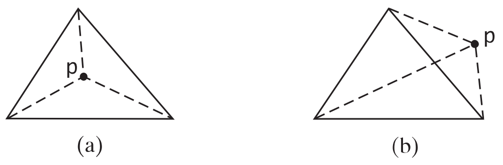


Figure 10.23 (a) A point is inside the triangle. (b) A point is outside the triangle.

***10.13** (*Geometry: the* `MyRectangle2D` *class*) Define the `MyRectangle2D` class that contains:

- Two `double` data fields named `x` and `y` that specify the center of the rectangle with getter and setter methods. (Assume that the rectangle sides are parallel to `x-` or `y-` axes.)
- The data fields `width` and `height` with getter and setter methods.
- A no-arg constructor that creates a default rectangle with (`0`, `0`) for (`x`, `y`) and `1` for both `width` and `height`.
- A constructor that creates a rectangle with the specified `x`, `y`, `width`, and `height`.

- A method `getArea()` that returns the area of the rectangle.
- A method `getPerimeter()` that returns the perimeter of the rectangle.
- A method `contains(double x, double y)` that returns `true` if the specified point (`x`, `y`) is inside this rectangle (see Figure 10.24a).
- A method `contains(MyRectangle2D r)` that returns `true` if the specified rectangle is inside this rectangle (see Figure 10.24b).
- A method `overlaps(MyRectangle2D r)` that returns `true` if the specified rectangle overlaps with this rectangle (see Figure 10.24c).

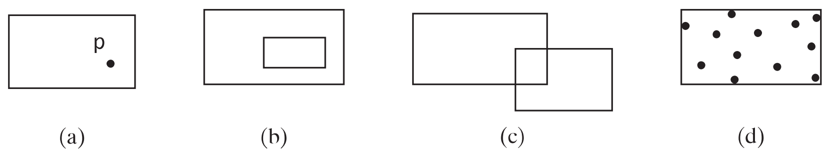


FIGURE 10.24 A point is inside the rectangle. (b) A rectangle is inside another rectangle. (c) A rectangle overlaps another rectangle. (d) Points are enclosed inside a rectangle.

Draw the UML diagram for the class and then implement the class. Write a test program that creates a `MyRectangle2D` object `r1` (`new MyRectangle2D(2, 2, 5.5, 4.9)`), displays its area and perimeter, and displays the result of `r1.contains(3, 3)`, `r1.contains(new MyRectangle2D(4, 5, 10.5, 3.2))`, and `r1.overlaps(new MyRectangle2D(3, 5, 2.3, 5.4))`.

***10.14** (*The* `MyDate` *class*) Design a class named `MyDate`. The class contains:

- The data fields `year`, `month`, and `day` that represent a date. `month` is 0-based, i.e., `0` is for January.
- A no-arg constructor that creates a `MyDate` object for the current date.
- A constructor that constructs a `MyDate` object with a specified elapsed time since midnight, January 1, 1970, in milliseconds.
- A constructor that constructs a `MyDate` object with the specified year, month, and day.
- Three getter methods for the data fields `year`, `month`, and `day`, respectively.
- A method named `setDate(long elapsedTime)` that sets a new date for the object using the elapsed time.

Draw the UML diagram for the class and then implement the class. Write a test program that creates two `MyDate` objects (using `new MyDate()` and `new MyDate(34355555133101L)`) and displays their year, month, and day.

(*Hint*: The first two constructors will extract the year, month, and day from the elapsed time. For example, if the elapsed time is `561555550000` milliseconds, the year is `1987`, the month is `9`, and the day is `18`. You may use the `GregorianCalendar` class discussed in Programming Exercise 9.5 to simplify coding.)

***10.15** (*Geometry: the bounding rectangle*) A bounding rectangle is the minimum rectangle that encloses a set of points in a two-dimensional plane, as shown in Figure 10.24d. Write a method that returns a bounding rectangle for a set of points in a two-dimensional plane, as follows:

```
public static MyRectangle2D getRectangle(double[][] points)
```

The **Rectangle2D** class is defined in Programming Exercise 10.13. Write a test program that prompts the user to enter five points and displays the bounding rectangle's center, width, and height. Here is a sample run:

```
Enter five points: 1.0 2.5 3 4 5 6 7 8 9 10 ↵Enter
The bounding rectangle's center (5.0, 6.25), width 8.0, height 7.5
```

Section 10.9

***10.16** (*Divisible by* **2** *or* **3**) Find the first ten numbers with **50** decimal digits that are divisible by **2** or **3**.

***10.17** (*Square numbers*) Find the first ten square numbers that are greater than **Long.MAX_VALUE**. A square number is a number in the form of n^2. For example, 4, 9, and 16 are square numbers. Find an efficient approach to run your program fast.

***10.18** (*Large prime numbers*) Write a program that finds five prime numbers larger than **Long.MAX_VALUE**.

***10.19** (*Mersenne prime*) A prime number is called a *Mersenne prime* if it can be written in the form $2^p - 1$ for some positive integer p. Write a program that finds all Mersenne primes with $p \leq 100$ and displays the output as shown below. (*Hint*: You have to use **BigInteger** to store the number, because it is too big to be stored in **long**. Your program may take several hours to run.)

```
    p       2^p - 1

    2         3
    3         7
    5        31
...
```

***10.20** (*Approximate e*) Programming Exercise 5.26 approximates e using the following series:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{i!}$$

In order to get better precision, use **BigDecimal** with **25** digits of precision in the computation. Write a program that displays the **e** value for **i** = **100**, **200**, . . ., and **1000**.

10.21 (*Divisible by* **5** *or* **6**) Find the first ten numbers greater than **Long.MAX_VALUE** that are divisible by **5** or **6**.

Sections 10.10–10.11

****10.22** (*Implement the* **String** *class*) The **String** class is provided in the Java library. Provide your own implementation for the following methods (name the new class **MyString1**):

```java
public MyString1(char[] chars);
public char charAt(int index);
public int length();
public MyString1 substring(int begin, int end);
public MyString1 toLowerCase();
public boolean equals(MyString1 s);
public static MyString1 valueOf(int i);
```

****10.23** (*Implement the `String` class*) The `String` class is provided in the Java library. Provide your own implementation for the following methods (name the new class `MyString2`):

```
public MyString2(String s);
public int compare(String s);
public MyString2 substring(int begin);
public MyString2 toUpperCase();
public char[] toChars();
public static MyString2 valueOf(boolean b);
```

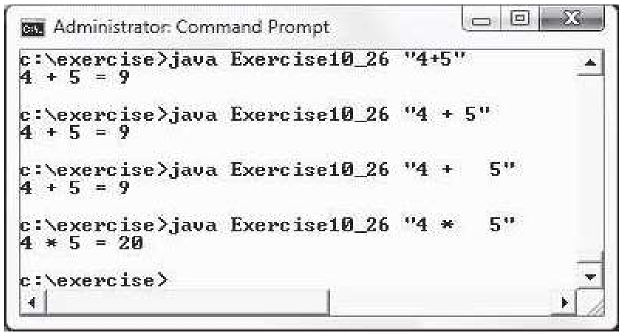
10.24 (*Implement the `Character` class*) The `Character` class is provided in the Java library. Provide your own implementation for this class. Name the new class `MyCharacter`.

****10.25** (*New string `split` method*) The `split` method in the `String` class returns an array of strings consisting of the substrings split by the delimiters. However, the delimiters are not returned. Implement the following new method that returns an array of strings consisting of the substrings split by the matching delimiters, including the matching delimiters.

```
public static String[] split(String s, String regex)
```

For example, `split("ab#12#453", "#")` returns `ab`, `#`, `12`, `#`, `453` in an array of `String`, and `split("a?b?gf#e", "[?#]")` returns `a`, `b`, `?`, `b`, `gf`, `#`, and `e` in an array of `String`.

***10.26** (*Calculator*) Revise Listing 7.9, Calculator.java, to accept an expression as a string in which the operands and operator are separated by zero or more spaces. For example, `3+4` and `3 + 4` are acceptable expressions. Here is a sample run:


Administrator: Command Prompt

```
c:\exercise>java Exercise10_26 "4+5"
4 + 5 = 9

c:\exercise>java Exercise10_26 "4 + 5"
4 + 5 = 9

c:\exercise>java Exercise10_26 "4 +   5"
4 + 5 = 9

c:\exercise>java Exercise10_26 "4 *   5"
4 * 5 = 20

c:\exercise>
```

****10.27** (*Implement the `StringBuilder` class*) The `StringBuilder` class is provided in the Java library. Provide your own implementation for the following methods (name the new class `MyStringBuilder1`):

```
public MyStringBuilder1(String s);
public MyStringBuilder1 append(MyStringBuilder1 s);
public MyStringBuilder1 append(int i);
public int length();
public char charAt(int index);
public MyStringBuilder1 toLowerCase();
public MyStringBuilder1 substring(int begin, int end);
public String toString();
```

****10.28** (*Implement the* **StringBuilder** *class*) The **StringBuilder** class is provided in the Java library. Provide your own implementation for the following methods (name the new class **MyStringBuilder2**):

```
public MyStringBuilder2();
public MyStringBuilder2(char[] chars);
public MyStringBuilder2(String s);
public MyStringBuilder2 insert(int offset, MyStringBuilder2 s);
public MyStringBuilder2 reverse();
public MyStringBuilder2 substring(int begin);
public MyStringBuilder2 toUpperCase();
```