SYS5110 – Foundations on Modelling and Simulation
Java Exercises, Liang Chapters 9 and 10
Fall 2015

9.1 (*The* `Rectangle` *class*) Following the example of the `Circle` class in Section 9.2, design a class named `Rectangle` to represent a rectangle. The class contains:

- Two `double` data fields named `width` and `height` that specify the width and height of the rectangle. The default values are `1` for both `width` and `height`.
- A no-arg constructor that creates a default rectangle.
- A constructor that creates a rectangle with the specified `width` and `height`.
- A method named `getArea()` that returns the area of this rectangle.
- A method named `getPerimeter()` that returns the perimeter.

Draw the UML diagram for the class and then implement the class. Write a test program that creates two `Rectangle` objects—one with width `4` and height `40` and the other with width `3.5` and height `35.9`. Display the width, height, area, and perimeter of each rectangle in this order.

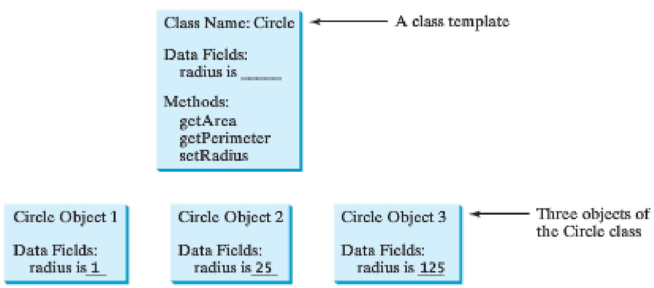

FIGURE 9.2 A class is a template for creating objects.

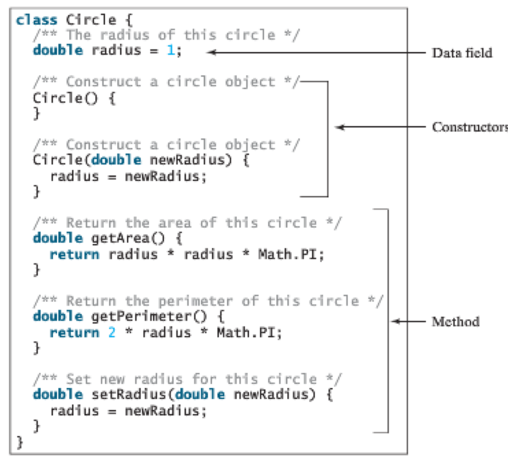

FIGURE 9.3 A class is a construct that defines objects of the same type.

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 for circle objects. The class that contains the `main` method will be referred to in this book, for convenience, as the *main class*.

main class

The illustration of class templates and objects in Figure 9.2 can be standardized using *Unified Modeling Language (UML)* notation. This notation, as shown in Figure 9.4, is called a *UML class diagram*, or simply a *class diagram*. In the class diagram, the data field is denoted as

Unified Modeling Language (UML)

class diagram

```
dataFieldName: dataFieldType
```

The constructor is denoted as

```
ClassName(parameterName: parameterType)
```

The method is denoted as

```
methodName(parameterName: parameterType): returnType
```

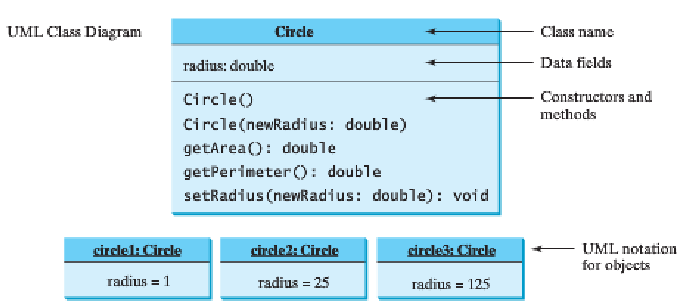

FIGURE 9.4 Classes and objects can be represented using UML notation.

9.7 (*The* **Account** *class*) Design a class named `Account` that contains:

- A private `int` data field named `id` for the account (default `0`).
- A private `double` data field named `balance` for the account (default `0`).
- A private `double` data field named `annualInterestRate` that stores the current interest rate (default `0`). Assume all accounts have the same interest rate.
- A private `Date` data field named `dateCreated` that stores the date when the account was created.
- A no-arg constructor that creates a default account.
- A constructor that creates an account with the specified id and initial balance.
- The accessor and mutator methods for `id`, `balance`, and `annualInterestRate`.
- The accessor method for `dateCreated`.
- A method named `getMonthlyInterestRate()` that returns the monthly interest rate.
- A method named `getMonthlyInterest()` that returns the monthly interest.
- A method named `withdraw` that withdraws a specified amount from the account.
- A method named `deposit` that deposits a specified amount to the account.

Draw the UML diagram for the class and then implement the class. (*Hint*: The method `getMonthlyInterest()` is to return monthly interest, not the interest rate. Monthly interest is `balance * monthlyInterestRate`. `monthlyInterestRate` is `annualInterestRate / 12`. Note that `annualInterestRate` is a percentage, e.g., like 4.5%. You need to divide it by 100.)

Write a test program that creates an `Account` object with an account ID of 1122, a balance of $20,000, and an annual interest rate of 4.5%. Use the `withdraw` method to withdraw $2,500, use the `deposit` method to deposit $3,000, and print the balance, the monthly interest, and the date when this account was created.

***9.11** (*Algebra: 2 × 2 linear equations*) Design a class named `LinearEquation` for a 2 × 2 system of linear equations:

$$\begin{array}{l} ax + by = e \\ cx + dy = f \end{array} \qquad x = \frac{ed - bf}{ad - bc} \qquad y = \frac{af - ec}{ad - bc}$$

The class contains:

- Private data fields `a`, `b`, `c`, `d`, `e`, and `f`.
- A constructor with the arguments for `a`, `b`, `c`, `d`, `e`, and `f`.
- Six getter methods for `a`, `b`, `c`, `d`, `e`, and `f`.
- A method named `isSolvable()` that returns true if $ad - bc$ is not 0.
- Methods `getX()` and `getY()` that return the solution for the equation.

Draw the UML diagram for the class and then implement the class. Write a test program that prompts the user to enter `a`, `b`, `c`, `d`, `e`, and `f` and displays the result. If $ad - bc$ is 0, report that "The equation has no solution." See Programming Exercise 3.3 for sample runs.

```
Enter a, b, c, d, e, f: 9.0 4.0 3.0 -5.0 -6.0 -21.0 ↵Enter
x is -2.0 and y is 3.0
```



```
Enter a, b, c, d, e, f: 1.0 2.0 2.0 4.0 4.0 5.0 ↵Enter
The equation has no solution
```

Use an array to pass all values to the *LinearEquation* constructor and an array to store the parameter values. Define within the *LinearEquation* class a set of integer constants (e.g. *public final int A_IX = 0*) for all indexes and the length of the parameter array to simplify (and clearly identify) the elements of the array (e.g. *p[A_IX]* would be the expression to access the value of *a*).

***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.)

Use the *toString* method to display the Time object time value. A *main* method has been provided to test the *Time* class. Note that *System.currentTimeMillis* returns the GMT time.

LISTING 2.7 ShowCurrentTime.java

```
public class ShowCurrentTime {
  public static void main(String[] args) {
    // Obtain the total milliseconds since midnight, Jan 1, 1970
    long totalMilliseconds = System.currentTimeMillis();

    // Obtain the total seconds since midnight, Jan 1, 1970
    long totalSeconds = totalMilliseconds / 1000;

    // Compute the current second in the minute in the hour
    long currentSecond = totalSeconds % 60;

    // Obtain the total minutes
    long totalMinutes = totalSeconds / 60;

    // Compute the current minute in the hour
    long currentMinute = totalMinutes % 60;

    // Obtain the total hours
    long totalHours = totalMinutes / 60;

    // Compute the current hour
    long currentHour = totalHours % 24;

    // Display results
    System.out.println("Current time is " + currentHour + ":"
      + currentMinute + ":" + currentSecond + " GMT");
  }
}
```

totalMilliseconds

totalSeconds

currentSecond

totalMinutes

currentMinute

totalHours

currentHour

preparing output

```
Current time is 17:31:8 GMT
```

**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

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

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

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

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

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

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

Enter an id:
```