MyProgrammingLab™

PROGRAMMING EXERCISES

three objectives

Pedagogical Note

The exercises in Chapters 9–13 help you achieve three objectives:

- Design classes and draw UML class diagrams.
- Implement classes from the UML.
- Use classes to develop applications.

Students can download solutions for the UML diagrams for the even-numbered exercises from the Companion Website, and instructors can download all solutions from the same site.

Sections 9.2–9.5

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.

9.2 (*The `Stock` class*) Following the example of the `Circle` class in Section 9.2, design a class named `Stock` that contains:

- A string data field named `symbol` for the stock's symbol.
- A string data field named `name` for the stock's name.
- A `double` data field named `previousClosingPrice` that stores the stock price for the previous day.
- A `double` data field named `currentPrice` that stores the stock price for the current time.
- A constructor that creates a stock with the specified symbol and name.
- A method named `getChangePercent()` that returns the percentage changed from `previousClosingPrice` to `currentPrice`.

Draw the UML diagram for the class and then implement the class. Write a test program that creates a `Stock` object with the stock symbol `ORCL`, the name `Oracle Corporation`, and the previous closing price of `34.5`. Set a new current price to `34.35` and display the price-change percentage.

Section 9.6

***9.3** (*Use the `Date` class*) Write a program that creates a `Date` object, sets its elapsed time to `10000`, `100000`, `1000000`, `10000000`, `100000000`, `1000000000`, `10000000000`, and `100000000000`, and displays the date and time using the `toString()` method, respectively.

***9.4** (*Use the `Random` class*) Write a program that creates a `Random` object with seed `1000` and displays the first 50 random integers between `0` and `100` using the `nextInt(100)` method.

***9.5** (*Use the `GregorianCalendar` class*) Java API has the `GregorianCalendar` class in the `java.util` package, which you can use to obtain the year, month, and day of a date. The no-arg constructor constructs an instance for the current date, and the methods `get(GregorianCalendar.YEAR)`, `get(GregorianCalendar.MONTH)`, and `get(GregorianCalendar.DAY_OF_MONTH)` return the year, month, and day. Write a program to perform two tasks:

- Display the current year, month, and day.
- The `GregorianCalendar` class has the `setTimeInMillis(long)`, which can be used to set a specified elapsed time since January 1, 1970. Set the value to `1234567898765L` and display the year, month, and day.

Sections 9.7–9.9

***9.6** (*Stopwatch*) Design a class named `StopWatch`. The class contains:

- Private data fields `startTime` and `endTime` with getter methods.
- A no-arg constructor that initializes `startTime` with the current time.
- A method named `start()` that resets the `startTime` to the current time.
- A method named `stop()` that sets the `endTime` to the current time.
- A method named `getElapsedTime()` that returns the elapsed time for the stopwatch in milliseconds.

Draw the UML diagram for the class and then implement the class. Write a test program that measures the execution time of sorting 100,000 numbers using selection sort.

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.

VideoNote
The Fan class

9.8 (*The* `Fan` *class*) Design a class named `Fan` to represent a fan. The class contains:

- Three constants named `SLOW`, `MEDIUM`, and `FAST` with the values `1`, `2`, and `3` to denote the fan speed.
- A private `int` data field named `speed` that specifies the speed of the fan (the default is `SLOW`).
- A private `boolean` data field named `on` that specifies whether the fan is on (the default is `false`).
- A private `double` data field named `radius` that specifies the radius of the fan (the default is `5`).
- A string data field named `color` that specifies the color of the fan (the default is `blue`).
- The accessor and mutator methods for all four data fields.
- A no-arg constructor that creates a default fan.
- A method named `toString()` that returns a string description for the fan. If the fan is on, the method returns the fan speed, color, and radius in one combined string. If the fan is not on, the method returns the fan color and radius along with the string "fan is off" in one combined string.

Draw the UML diagram for the class and then implement the class. Write a test program that creates two `Fan` objects. Assign maximum speed, radius `10`, color `yellow`, and turn it on to the first object. Assign medium speed, radius `5`, color `blue`, and turn it off to the second object. Display the objects by invoking their `toString` method.

****9.9** (*Geometry: n-sided regular polygon*) In an n-sided regular polygon, all sides have the same length and all angles have the same degree (i.e., the polygon is both equilateral and equiangular). Design a class named `RegularPolygon` that contains:

- A private `int` data field named `n` that defines the number of sides in the polygon with default value `3`.
- A private `double` data field named `side` that stores the length of the side with default value `1`.
- A private `double` data field named `x` that defines the x-coordinate of the polygon's center with default value `0`.
- A private `double` data field named `y` that defines the y-coordinate of the polygon's center with default value `0`.
- A no-arg constructor that creates a regular polygon with default values.
- A constructor that creates a regular polygon with the specified number of sides and length of side, centered at (`0`, `0`).
- A constructor that creates a regular polygon with the specified number of sides, length of side, and x- and y-coordinates.
- The accessor and mutator methods for all data fields.
- The method `getPerimeter()` that returns the perimeter of the polygon.
- The method `getArea()` that returns the area of the polygon. The formula for computing the area of a regular polygon is $Area = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$.

Draw the UML diagram for the class and then implement the class. Write a test program that creates three `RegularPolygon` objects, created using the no-arg constructor, using `RegularPolygon(6, 4)`, and using `RegularPolygon(10, 4, 5.6, 7.8)`. For each object, display its perimeter and area.

*9.10 (*Algebra: quadratic equations*) Design a class named `QuadraticEquation` for a quadratic equation $ax^2 + bx + x = 0$. The class contains:

- Private data fields `a`, `b`, and `c` that represent three coefficients.
- A constructor for the arguments for `a`, `b`, and `c`.
- Three getter methods for `a`, `b`, and `c`.
- A method named `getDiscriminant()` that returns the discriminant, which is $b^2 - 4ac$.
- The methods named `getRoot1()` and `getRoot2()` for returning two roots of the equation

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

These methods are useful only if the discriminant is nonnegative. Let these methods return `0` if the discriminant is negative.

Draw the UML diagram for the class and then implement the class. Write a test program that prompts the user to enter values for a, b, and c and displays the result based on the discriminant. If the discriminant is positive, display the two roots. If the discriminant is 0, display the one root. Otherwise, display "The equation has no roots." See Programming Exercise 3.1 for sample runs.

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

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

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.

**9.12 (*Geometry: intersecting point*) Suppose two line segments intersect. The two endpoints for the first line segment are (`x1`, `y1`) and (`x2`, `y2`) and for the second line segment are (`x3`, `y3`) and (`x4`, `y4`). Write a program that prompts the user to enter these four endpoints and displays the intersecting point. As discussed in Programming Exercise 3.25, the intersecting point can be found by solving a linear equation. Use the `LinearEquation` class in Programming Exercise 9.11 to solve this equation. See Programming Exercise 3.25 for sample runs.

**9.13 (*The `Location` class*) Design a class named `Location` for locating a maximal value and its location in a two-dimensional array. The class contains public data fields `row`, `column`, and `maxValue` that store the maximal value and its indices in a two-dimensional array with `row` and `column` as `int` types and `maxValue` as a `double` type.

Write the following method that returns the location of the largest element in a two-dimensional array:

```
public static Location locateLargest(double[][] a)
```

The return value is an instance of `Location`. Write a test program that prompts the user to enter a two-dimensional array and displays the location of the largest element in the array. Here is a sample run:

```
Enter the number of rows and columns in the array:  3 4 Enter
Enter the array:
23.5 35 2 10 Enter
4.5 3 45 3.5 Enter
35 44 5.5 9.6 Enter
The location of the largest element is 45 at (1, 2)
```