Programming Exercises

MyProgrammingLab™

Sections 11.2–11.4

11.1 (*The `Triangle` class*) Design a class named `Triangle` that extends `GeometricObject`. The class contains:

- Three `double` data fields named `side1`, `side2`, and `side3` with default values `1.0` to denote three sides of the triangle.
- A no-arg constructor that creates a default triangle.
- A constructor that creates a triangle with the specified `side1`, `side2`, and `side3`.
- The accessor methods for all three data fields.
- A method named `getArea()` that returns the area of this triangle.
- A method named `getPerimeter()` that returns the perimeter of this triangle.
- A method named `toString()` that returns a string description for the triangle.

For the formula to compute the area of a triangle, see Programming Exercise 2.19. The `toString()` method is implemented as follows:

```
return "Triangle: side1 = " + side1 + " side2 = " + side2 +
  " side3 = " + side3;
```

Draw the UML diagrams for the classes `Triangle` and `GeometricObject` and implement the classes. Write a test program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a `Triangle` object with these sides and set the `color` and `filled` properties using the input. The program should display the area, perimeter, color, and true or false to indicate whether it is filled or not.

Sections 11.5–11.14

11.2 (*The `Person`, `Student`, `Employee`, `Faculty`, and `Staff` classes*) Design a class named `Person` and its two subclasses named `Student` and `Employee`. Make `Faculty` and `Staff` subclasses of `Employee`. A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date hired. Use the `MyDate` class defined in Programming Exercise 10.14 to create an object for date hired. A faculty member has office hours and a rank. A staff member has a title. Override the `toString` method in each class to display the class name and the person's name.

Draw the UML diagram for the classes and implement them. Write a test program that creates a `Person`, `Student`, `Employee`, `Faculty`, and `Staff`, and invokes their `toString()` methods.

11.3 (*Subclasses of `Account`*) In Programming Exercise 9.7, the `Account` class was defined to model a bank account. An account has the properties account number, balance, annual interest rate, and date created, and methods to deposit and withdraw funds. Create two subclasses for checking and saving accounts. A checking account has an overdraft limit, but a savings account cannot be overdrawn.

Draw the UML diagram for the classes and then implement them. Write a test program that creates objects of `Account`, `SavingsAccount`, and `CheckingAccount` and invokes their `toString()` methods.

11.4 (*Maximum element in `ArrayList`*) Write the following method that returns the maximum value in an `ArrayList` of integers. The method returns `null` if the list is `null` or the list size is `0`.

```
public static Integer max(ArrayList<Integer> list)
```

Write a test program that prompts the user to enter a sequence of numbers ending with `0`, and invokes this method to return the largest number in the input.

11.5 (*The `Course` class*) Rewrite the `Course` class in Listing 10.6. Use an `ArrayList` to replace an array to store students. Draw the new UML diagram for the class. You should not change the original contract of the `Course` class (i.e., the definition of the constructors and methods should not be changed, but the private members may be changed.)

11.6 (*Use `ArrayList`*) Write a program that creates an `ArrayList` and adds a `Loan` object, a `Date` object, a string, and a `Circle` object to the list, and use a loop to display all the elements in the list by invoking the object's `toString()` method.

11.7 (*Shuffle `ArrayList`*) Write the following method that shuffles the elements in an `ArrayList` of integers.

```java
public static void shuffle(ArrayList<Integer> list)
```

VideoNote
New Account class

****11.8** (*New `Account` class*) An `Account` class was specified in Programming Exercise 9.7. Design a new `Account` class as follows:

- Add a new data field `name` of the `String` type to store the name of the customer.
- Add a new constructor that constructs an account with the specified name, id, and balance.
- Add a new data field named `transactions` whose type is `ArrayList` that stores the transaction for the accounts. Each transaction is an instance of the `Transaction` class. The `Transaction` class is defined as shown in Figure 11.6.

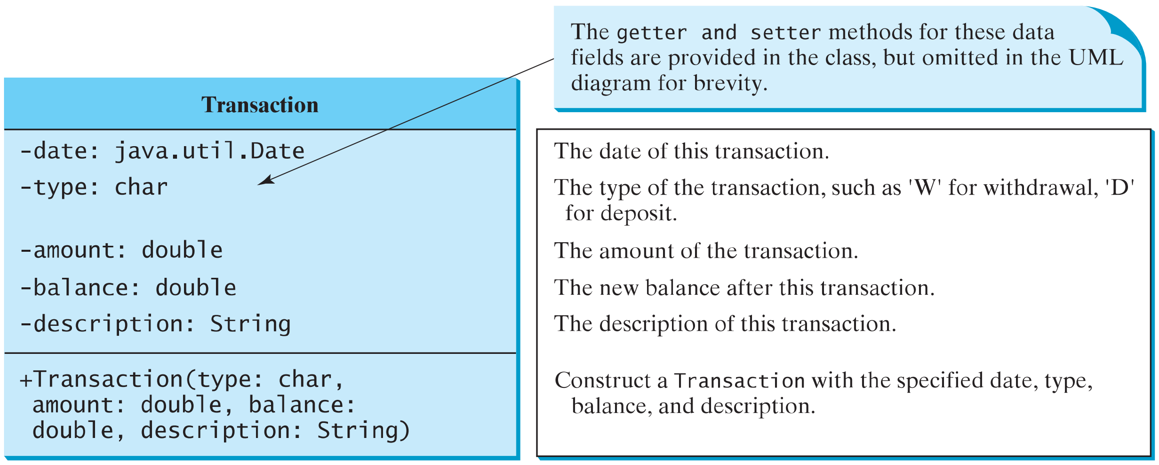
The getter and setter methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
Transaction
-date: java.util.Date
-type: char
-amount: double
-balance: double
-description: String
+Transaction(type: char, amount: double, balance: double, description: String)
The date of this transaction.
The type of the transaction, such as 'W' for withdrawal, 'D' for deposit.
The amount of the transaction.
The new balance after this transaction.
The description of this transaction.
Construct a Transaction with the specified date, type, balance, and description.

FIGURE 11.6 The `Transaction` class describes a transaction for a bank account.

- Modify the `withdraw` and `deposit` methods to add a transaction to the `transactions` array list.
- All other properties and methods are the same as in Programming Exercise 9.7.

Write a test program that creates an `Account` with annual interest rate `1.5%`, balance `1000`, id `1122`, and name `George`. Deposit $30, $40, and $50 to the account and withdraw $5, $4, and $2 from the account. Print an account summary that shows account holder name, interest rate, balance, and all transactions.

***11.9** (*Largest rows and columns*) Write a program that randomly fills in `0`s and `1`s into an n-by-n matrix, prints the matrix, and finds the rows and columns with the most `1`s. (*Hint*: Use two `ArrayList`s to store the row and column indices with the most `1`s.) Here is a sample run of the program:

```
Enter the array size n: 4 ↵
The random array is
0011
0011
1101
1010
The largest row index: 2
The largest column index: 2, 3
```

11.10 (*Implement `MyStack` using inheritance*) In Listing 11.10, `MyStack` is implemented using composition. Define a new stack class that extends `ArrayList`.

Draw the UML diagram for the classes and then implement `MyStack`. Write a test program that prompts the user to enter five strings and displays them in reverse order.

11.11 (*Sort `ArrayList`*) Write the following method that sorts an `ArrayList` of numbers:

```
public static void sort(ArrayList<Integer> list)
```

Write a test program that prompts the user to enter 5 numbers, stores them in an array list, and displays them in increasing order.

11.12 (*Sum `ArrayList`*) Write the following method that returns the sum of all numbers in an `ArrayList`:

```
public static double sum(ArrayList<Double> list)
```

Write a test program that prompts the user to enter 5 numbers, stores them in an array list, and displays their sum.

***11.13** (*Remove duplicates*) Write a method that removes the duplicate elements from an array list of integers using the following header:

```
public static void removeDuplicate(ArrayList<Integer> list)
```

Write a test program that prompts the user to enter 10 integers to a list and displays the distinct integers separated by exactly one space. Here is a sample run:

```
Enter ten integers: 34 5 3 5 6 4 33 2 2 4 ↵
The distinct integers are 34 5 3 6 4 33 2
```

11.14 (*Combine two lists*) Write a method that returns the union of two array lists of integers using the following header:

```
public static ArrayList<Integer> union(
  ArrayList<Integer> list1, ArrayList<Integer> list2)
```

For example, the union of two array lists {2, 3, 1, 5} and {3, 4, 6} is {2, 3, 1, 5, 3, 4, 6}. Write a test program that prompts the user to enter two lists, each with five integers, and displays their union. The numbers are separated by exactly one space in the output. Here is a sample run:

```
Enter five integers for list1: 3 5 45 4 3 ↵
Enter five integers for list2: 33 51 5 4 13 ↵
The combined list is 3 5 45 4 3 33 51 5 4 13
```

***11.15** (*Area of a convex polygon*) A polygon is convex if it contains any line segments that connects two points of the polygon. Write a program that prompts the user to enter the number of points in a convex polygon, then enter the points clockwise, and display the area of the polygon. Here is a sample run of the program:

```
Enter the number of the points: 7 ↵
Enter the coordinates of the points:
  -12 0 -8.5 10 0 11.4 5.5 7.8 6 -5.5 0 -7 -3.5 -3.5 ↵
The total area is 250.075
```

****11.16** (*Addition quiz*) Rewrite Listing 5.1 RepeatAdditionQuiz.java to alert the user if an answer is entered again. *Hint: use an array list to store answers.* Here is a sample run:

```
What is 5 + 9? 12 ↵
Wrong answer. Try again. What is 5 + 9? 34 ↵
Wrong answer. Try again. What is 5 + 9? 12 ↵
You already entered 12
Wrong answer. Try again. What is 5 + 9? 14 ↵
You got it!
```

****11.17** (*Algebra: perfect square*) Write a program that prompts the user to enter an integer `m` and find the smallest integer `n` such that `m * n` is a perfect square. (*Hint:* Store all smallest factors of `m` into an array list. `n` is the product of the factors that appear an odd number of times in the array list. For example, consider `m` = 90, store the factors 2, 3, 3, 5 in an array list. 2 and 5 appear an odd number of times in the array list. So, `n` is 10.) Here are sample runs:

```
Enter an integer m: 1500 ↵
The smallest number n for m * n to be a perfect square is 15
m * n is 22500
```

```
Enter an integer m: 63 ↵
The smalle
st number n for m * n to be a perfect square is 7
m * n is 441
```