การทดลองที่ 6

Chapter 10 Object-Oriented Thinking

Program # 1 (Exercise 10.2 The BMI class) Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters.

```
// Construct a BMI with the specified name, age, weight, feet, and inches
```

`public BMI(String name, int age, double weight, double feet, double inches)`

BMI	Interpretation
BMI < 18.5	Underweight
18.5 ≤ BMI < 25.0	Normal
25.0 ≤ BMI < 30.0	Overweight
30.0 ≤ BMI	Obese

Write a program that prompts the user to enter name, age, weight in pounds, and height in inches and displays the BMI together with the interpretation.

Note that one pound is 0.45359237 kilograms, and one inch is 0.0254 meters.

Program # 2 (Exercise 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.

Program # 3 (Exercise 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.

Program # 4 (Exercise 8.12 Financial application: compute tax) Rewrite Listing 3.5, ComputeTax.java, using arrays. For each filing status, there are six tax rates. Each rate is applied to a certain amount of taxable income.

For example, from the taxable income of $400,000 for a single filer,
$8,350 is taxed at 10%,
(33,950 - 8,350) at 15%,
(82,250 - 33,950) at 25%,
(171,550 - 82,550) at 28%,
(372,550 - 82,250) at 33%,
and
(400,000 - 372,950) at 36%.

The six rates are the same for all filing statuses, which can be represented in the following array:

```
double[] rates = {0.10, 0.15, 0.25, 0.28, 0.33, 0.36};
```

The brackets for each rate for all the filing statuses can be represented in a two-dimensional array as follows:

```
int[][] brackets = {
      {8350, 33950, 82250, 171550, 372950},     // Single filer
      {16700, 67900, 137050, 20885, 372950},    // Married jointly
                                                // -or qualifying widow(er)
      {8350, 33950, 68525, 104425, 186475},     // Married separately
      {11950, 45500, 117450, 190200, 372950}    // Head of household
};
```

Suppose the taxable income is $400,000 for single filers. The tax can be computed as follows:

```
tax = brackets[0][0] * rates[0] +
      (brackets[0][1] - brackets[0][0]) * rates[1] +
      (brackets[0][2] - brackets[0][1]) * rates[2] +
      (brackets[0][3] - brackets[0][2]) * rates[3] +
      (brackets[0][4] - brackets[0][3]) * rates[4] +
      (400000 - brackets[0][4]) * rates[5];
```

```
Filing Status
   [0]-Single filer
   [1]-Married jointly or qualifying widow(er)
   [2]-Married separately
   [3]-Head of household
Enter the filing status: 0 <Enter>
Enter the taxable income: 400000 <Enter>
Tax is 117683.50
```

Program # 5 (Exercise 10.8 Financial: the Tax class) Programming #4 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 tax for taxable income $400,000.

Program # 6 (Exercise 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 100.00

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 97.00
```

```
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 107.00

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4 <Enter>
End of Program.
```