Programming Exercise 10.7 p.401 (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:
```

Programming Exercise 10.8 p.402 (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

---

3.9 Case Study: Computing Taxes

You can use nested `if` *statements to write a program for computing taxes.*

VideoNote
Use multi-way `if-else` statements

The United States federal personal income tax is calculated based on filing status and taxable income. There are four filing statuses: single filers, married filing jointly or qualified widow(er), married filing separately, and head of household. The tax rates vary every year. Table 3.2 shows the rates for 2009. If you are, say, single with a taxable income of $10,000, the first $8,350 is taxed at 10% and the other $1,650 is taxed at 15%, so, your total tax is $1,082.50.

TABLE 3.2 2009 U.S. Federal Personal Tax Rates

Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualifying Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351 – $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,526 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 – $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

```
For each filing status there are six tax rates.
Each rate is applied to a certain amount of taxable income.
For example, of a taxable income of $400,000 for single filers,
      $8,350 is taxed at 10%,
      $(33,950 - 8,350) at 15%,
      $(82,250 - 33,950) at 25%,
      $(171,550 - 82,250) at 28%,
      $(372,950 - 171,550) at 33%,
      and
      $(400,000 - 372,950) at 35%.
```

```
(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 117683.5
```

Listing 3.5 ComputeTax.java

```
import java.util.Scanner;

public class ComputeTax {
  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter filing status
    System.out.print("(0-single filer, 1-married jointly or " +
      "qualifying widow(er), 2-married separately, 3-head of " +
      "household) Enter the filing status: ");
    int status = input.nextInt();

    // Prompt the user to enter taxable income
    System.out.print("Enter the taxable income: ");
    double income = input.nextDouble();

    // Compute tax
    double tax = 0;

    if (status == 0) { // Compute tax for single filers
      if (income <= 8350)
        tax = income * 0.10;
      else if (income <= 33950)
        tax = 8350 * 0.10 + (income - 8350) * 0.15;
      else if (income <= 82250)
        tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
          (income - 33950) * 0.25;
      else if (income <= 171550)
        tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
          (82250 - 33950) * 0.25 + (income - 82250) * 0.28;
      else if (income <= 372950)
        tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
          (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
          (income - 171550) * 0.33;
      else
        tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
          (82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
          (372950 - 171550) * 0.33 + (income - 372950) * 0.35;
    }
    else if (status == 1) { // Compute tax for married file jointly
      // Left as exercise
    }
    else if (status == 2) { // Compute tax for married separately
      // Left as exercise
    }
    else if (status == 3) { // Compute tax for head of household
      // Left as exercise
    }
    else {
      System.out.println("Error: invalid status");
      System.exit(1);
    }

    // Display the result
    System.out.println("Tax is " + (int)(tax * 100) / 100.0);
  }
}
```

Programming Exercise 10.9 p.402 (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.

LISTING 10.6 Course.java

```
1  public class Course {
2    private String courseName;
3    private String[] students = new String[100];          create students
4    private int numberOfStudents;
5
6    public Course(String courseName) {                    add a course
7      this.courseName = courseName;
8    }
9
10   public void addStudent(String student) {
11     students[numberOfStudents] = student;
12     numberOfStudents++;
13   }
14
15   public String[] getStudents() {                       return students
16     return students;
17   }
18
19   public int getNumberOfStudents() {                    number of students
20     return numberOfStudents;
21   }
22
23   public String getCourseName() {
24     return courseName;
25   }
26
27   public void dropStudent(String student) {
28     // Left as an exercise in Programming Exercise 10.9
29   }
30 }
```

■ 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.

```
public class Student {
  private Course[]
    courseList;

  public void addCourse(
    Course s) { ... }
}
```

```
public class Course {
  private Student[]
    classList;
  private Faculty faculty;

  public void addStudent(
    Student s) { ... }

  public void setFaculty(
    Faculty faculty) { ... }
}
```

```
public class Faculty {
  private Course[]
    courseList;

  public void addCourse(
    Course c) { ... }
}
```

FIGURE 10.5 The association relations are implemented using data fields and methods in classes.
