CS203 Java Programming and Applications
Fall 2019

Assignment 2 (7 Questions, 100 marks)

Assigned Date: October 8, 2019
Due Date: November 3, 2019 @ 23:59

QUESTION 1 (15 Marks) Design a Simple 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 annual interest rate (default `0.012`, e.g., 1.2%). Assume all accounts have the same interest rate. (*Hint*: This assumption implies that `annualInterestRate` should be static.)
- A private `Date` data field named `dateCreated` and 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 getter and setter methods for `id`, `balance`, and `annualInterestRate`.
- The getter method for `dateCreated`.
- A method named `getMonthlyInterestRate()` that returns the monthly interest rate. (*Hint*: `monthlyInterestRate` is `annualInterestRate/12`.)
- A method named `getMonthlyInterest()` that returns the monthly interest. (*Hint*: This method returns monthly interest, **NOT** the interest rate. Monthly interest is `balance * monthlyInterestRate`.)
- A method named `withdraw` that withdraws a specified amount from the account. (*Hint*: The amount may be greater than the balance. You need to handle this in your method.)
- A method name `deposit` that deposits a specified amount to the account.

Draw the UML diagram for the class and implement the class. Then write a test program (name the class as `TestAccount`) that creates an `Account` object with an account ID of 1122, a balance of $2,500, use the `deposit` method to deposit $3,000, and print the balance, the annual interest rate, the monthly interest rate, the monthly interest, and the date when this account was created. All floating-point numbers should be printed out with 2 decimal places. The annual and monthly interest rates should be printed out in percentage. A sample output is like:

```
The balance is $5500.00
The annual interest rate is 1.20%
The monthly interest rate is 0.10%
The monthly interest is $5.50
The account was created on Wed Dec 19 13:06:50 CST 2018
```

QUESTION 2 (15 Marks) Array of Objects

Use the `Account` class created in QUESTION 1 to simulate an ATM machine (name the class as `ATMWithAccount`). Create ten accounts in an array with id `1, 2, ..., 10`, and initial balance $100. The array index for each account is the same as the account id. (*Hint*: The array size is 11.) Set the first array element as null. 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. Once you enter id 0, the program prints out a summary of the ten accounts, including the id and the balance. Then the program is terminated. All floating-point numbers should be printed out with 2 decimal places. A sample run is as follows:

```
Enter an account id (1-10): 11
Invalid id
Enter an account id (1-10): 4

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1
The balance is 100.00

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

Enter an account id (1-10): 5

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 2
Enter an amount to withdraw: 200
Your balance is not enough.

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 2
Enter an amount to withdraw: 30

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4
You exit this account
```

```
Enter an account id (1-10): 8

Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 3
Enter an amount to deposit: 500

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

Enter an account id (1-10): 0

Here is a summary of your accounts:
Account 1: balance is 100.00
Account 2: balance is 100.00
Account 3: balance is 100.00
Account 4: balance is 100.00
Account 5: balance is 70.00
Account 6: balance is 100.00
Account 7: balance is 100.00
Account 8: balance is 600.00
Account 9: balance is 100.00
Account 10: balance is 100.00

You exit the ATM
```

QUESTION 3 (10 Marks) The String, StringBuilder, and StringBuffer Classes

1. Let `s1` be "`Welcome`" and `s2` be "`welcome`". Write the code for the following statements:

   a. Replace all occurrences of the character `e` with `E` in `s1` and assign the new string to `s2`.

   b. Split `Welcome to Java and HTML` into an array `tokens` delimited by a space and assign the first two tokens into `s1` and `s2`.

2. Show the output of the following code and briefly explain your answer.

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Hi, ABC, good".matches("ABC "));
    System.out.println("Hi, ABC, good".matches(".*ABC.*"));
    System.out.println("A,B;C".replaceAll(",;", "#"));
    System.out.println("A,B;C".replaceAll("[,;]", "#"));

    String[] tokens = "A,B;C".split("[,;]");
    for (int i = 0; i < tokens.length; i++)
      System.out.print(tokens[i] +  " ");
  }
}
```

3. Show the output of the following program and briefly explain your answer.

```
public class Test {
  public static void main(String[] args) {
    String s = "Java";
    StringBuilder builder = new StringBuilder(s);
    change(s, builder);

    System.out.println(s);
    System.out.println(builder);
  }

  private static void change(String s, StringBuilder builder) {
    s = s + " and HTML";
    builder.append(" and HTML");
  }
}
```

QUESTION 4 (15 Marks) New String split Method

The split method in the String class returns an array of strings consisting of the substrings split by the delimiters. However, the delimiters are not returned. Implement the following new method that returns an array of strings consisting of the substrings split by the matching delimiters, including the matching delimiters.

```
public static String[] split(String s, String regex)
```

For example, split("ab#12#453", "#") returns ab, #, 12, #, 453 in an array of String, and split("a?b?gf#e##", "[?#]") returns a, ?, b, ?, gf, #, e, and ## in an array of String. Name your class as NewSplit. (*Hint*: Check whether there are delimiters at the end of the string s. See the second example.)

QUESTION 5 (15 Marks) Inheritance and Polymorphism

Design a Ship class that has the following members:

- A private String data field named name for the name of the ship.
- A private String data field named yearBuilt for the year that the ship was built.
- A constructor that creates a ship with the specified name and the specified year that the ship was built.
- Appropriate getter and setter methods.
- A toString method that overrides the toString method in the Object class and returns a string containing the class name Ship, the ship's name and the year it was built. Use @Override to ensure you override the method.

Design a CruiseShip class that extends the Ship class. The CruiseShip class should have the following members:

- A private int data field named passengerCapacity for the maximum number of passengers.
- A constructor that creates a cruise ship with the specified name, the specified year that the ship was built, and the specified passenger capacity.
- Appropriate getter and setter methods.
- A toString method that overrides the toString method in the base class and returns a string containing the class name CruiseShip, the ship's name and passenger capacity. Use @Override to ensure you override the method.

Design a CargoShip class that extends the Ship class. The CargoShip class should have the following members:

- A private int data field named tonnage for the cargo capacity.

- A constructor that creates a cargo ship with the specified name, the specified year that the ship was built, and the specified tonnage.
- Appropriate getter and setter methods.
- A `toString` method that overrides the `toString` method in the base class and returns a string containing the class name `CargoShip`, the ship's name and cargo capacity. Use `@Override` to ensure you override the method.

Write a test program (name the class as `TestShips`) to verify your Ship, CruiseShip, and CargoShip classes. The test program has a `Ship` array of size 3, named `ships`. Assign various `Ship`, `CruiseShip`, and `CargoShip` objects to the array elements. Make sure your array elements contains objects from all the three classes. The program then steps through the array and print each object. A sample run is like:

```
Ship
Name: Lolipop
Year Built: 1960


CruiseShip
Name: Disney Magic
Passenger Capacity: 2400 persons


CargoShip
Name: Black Pearl
Cargo Capacity: 50000 tons
```

QUESTION 6 (15 Marks) The ArrayList Class

In this question, you will design two classes `Flight` and `Itinerary` and write a program to test them.

Let's first look at the `java.util.GregorianCalendar` class which you will use in designing your `Flight` and `Itinerary` classes. The `java.util.GregorianCalendar` class provides the standard calendar system used by most of the world. It has a constructor `GregorianCalender(int year, int month, int dayOfMonth, int hour, int minute, int second)` which creates a `GregorianCalendar` object with the given date and time set for the default time zone with the default locale. The `month` parameter is `0` based, that is, `0` for January. It has a method `getTimeInMillis()` (inherited from its parent class `java.util.Calendar`) that returns this Calendar's time in milliseconds (elapsed since midnight, January 1, 1970 GMT) which is in `long` type.

Now let's design the two classes `Flight` and `Itinerary`. The `Flight` class stores the information about a flight with the following members:

- A private `String` data field named `flightNo` for the flight number.

- A private `GregorianCalendar` data field named `departureTime`.
- A private `GregorianCalendar` data field named `arrivalTime`.
- A constructor that creates a flight with the specified flight number, departure time, and arrival time.
- Appropriate getter and setter methods. (*Hint*: Do we need setter for `flightNo`?)
- A method named `getFlightTime()` that returns the flight time in minutes. Assume the departureTime and arrivalTime are in the same time zone. (*Hint*: Use the method `getTimeInMillis()` to get `arrivalTime` and `departureTime` in milliseconds, compute their subtraction, and transfer the result from milliseconds to minutes.)

The `Itinerary` class stores the information about itinerary with the following members:

- A private `ArrayList<Flight>` data field named `flights` that contains the flights for the itinerary in increasing order of departureTime. (*Hint*: You do not need to do the sorting.)
- A constructor that creates an itinerary with the specified flights in `ArrayList<Flight>` type.
- A method named `getTotalFlightTime()` that returns the total flight time of the itinerary in minutes. (*Hint*: Invoke the `getFlightTime()` method for each `Flight` object.)
- A method named `getTotalTravelTime()` that returns the total travel time in minutes from the departure time of the first flight to the arrival time of the last flight in the itinerary. Assume all the times are in the same time zone.

Now write a test program (name the class as `TestFlightItinerary`) that has an `ArrayList` of `Flight` named `flights`. Add the following three flights one-by-one to `flights`:

- Flight number: "US230"
  Departure time: 05:05:00 May 5, 2014
  (*Hint*: `new GregorianCalendar(2014, 4, 5, 5, 5, 0)`)
  Arrival time: 06:15:00 May 5, 2014

- Flight number: "US235"
  Departure time: 06:55:00 May 5, 2014
  Arrival time: 07:45:00 May 5, 2014

- Flight number: "US237"
  Departure time: 09:35:00 May 5, 2014
  Arrival time: 12:55:00 May 5, 2014

Use `flights` to create an `Itinerary` object named `itinerary`. Print out the total flight time and the total travel time of `itinerary`.

QUESTION 7 (15 Marks) Exception Handling

PART I: (7 Marks)

Create an exception hierarchy of `ExceptionA`, `ExceptionB` and `ExceptionC` such that `ExceptionB` inherits from `ExceptionA` and `ExceptionC` inherits from `ExceptionB`.

Write a test program to show that the catch block for supertype exception (`ExceptionA`) can catch all the subtype exceptions (`ExceptionB` and `ExceptionC`). Print out a message such as "ExceptionB caught" to indicate which type of exception is caught. Name your class file as `ExceptionHierarchy.java`.

Hints:

1. To build the exception hierarchy, `ExceptionA`, `ExceptionB` and `ExceptionC` may have empty body.
2. You may use `System.err.println("ExceptionB caught");` instead of `System.out.println("ExceptionB caught");` so that the message will be output in red color and in the format of exception messages. This is a more common way to output error messages.

<u>PART II:</u> (8 Marks)

Use the exception hierarchy from PART I to demonstrate that the order of catch block is important if you want to specifically catch every exception in the same hierarchy. In other words, now you should be able to catch subtype exceptions (`ExceptionB` and `ExceptionC`) separately rather than using one catch block of supertype exception (`ExceptionA`) as you did in PART I.

You should define a method called `someMethod()` that can throw all three exceptions ***randomly***. To do this, generate a random integer of 1, 2 or 3 representing `ExceptionA`, `ExceptionB` or `ExceptionC` respectively, and then throw the corresponding exception. Output the random number in `someMethod()` to show which type of exception is randomly generated. Invoke `someMethod()` in your test program and catch all three exceptions separately in three catch blocks. In each catch block, print out a message such as "ExceptionA caught" to indicate which type of exception is caught. Run your program a few times to make sure that `someMethod()` may throw different exception for different run and you are able to catch all three types of exceptions. Name your class file as `CatchDifferentExceptions.java`.