Object Oriented Programming TDA547

Krasimir Angelov, tel. 031 772 10 19

2016-08-16

The total number of points is 40. 20 points certainly guarantee a pass. 27p correspond to grade 4 and 32p to grade 5.

No other help materials except an English Dictionary are allowed. Write clean and readable Java code. Trivial syntax errors will be tolerated without affecting the grades. You don't have to comment your code unless if you really want to.

1. Read the following program:

```
public class Question1 {
   public static int count(int[] a) {
      int i = 0;
      while (i < a.length) {
         if (a[i] == 0)
            break;
         i++;
      }
      return i;
   }

   public static void main(String[] args) {
      System.out.println(count(new int[] {1,2,0,4}));
      System.out.println(count(new int[] {1,2,4,0,2,1}));
      System.out.println(count(new int[] {1,2,2,1}));
   }
}
```

   What will the program print when it is executed? (4p)

2. In this task we do simple array processing:

   - Implement the method:

     ```
     public static double findNext(double x, double[] a)
     ```

     which receives a number `x` and an array `a` of doubles. The method should return the value of the smallest element in `a` which is bigger than `x`. For example if the method is called with `x=3.5` and `a={2,0,1,6,0,8,1,6}` then it should return 6. If all elements in the array are smaller or equal to `x`, than the method should return `Double.POSITIVE_INFINITY`. (4p)

   - Implement a class `Question2` which can be used to test the method `findNext`. The class should be possible to run like this:

     ```
     > java Question2 3.5 2 0 1 6 0 8 1 6
     6
     ```

     i.e. it takes `x` as the first command line argument, followed by a sequence of numbers which are the elemenets of `a`. (4p)

3. In this task you should implement a model class `Card`, which simulates the behaviour of a Västtrafik card. It should have the following components:

   - Two instance variables:
     - One of type `double` which is the current saldo in the card.
     - One of type `long` which keeps track of the time when the current stamp on the card expires.
   - A constructor:

     ```
     public Card(double saldo)
     ```

     which creates a new card with an initial saldo given by the argument of the constructor. It should also give an appropriate value for the time variable. The value should be such that, if the method `stamp` (bellow) is called immediately after the constructor, then the passenger should not get response `BYTE`.
   - a method:

     ```
     public void load(double amount)
     ```

     which loads the card, i.e. it adds the given `amount` to the saldo.
   - two methods:

```
public double getSaldo()
public long validTo()
```

that return the current saldo and the time when the current stamp expires.

- a method

  ```
  public Status stamp()
  ```

  which simulates the creation of a new stamp on the card. This is where most of the code will be. The method should return a value of type `Status` which is defined as:

  ```
  enum Status {OK, FAIL, BYTE}
  ```

  The implementation should consists of three rules:

  - If the current stamp on the card has not expired yet, then the method does nothing and returns `BYTE`.
  - If the current credit on the card is less than 10 SEK, then the metod should return `FAIL`.
  - Otherwise, the method should deduce the price of the ticket, i.e. 22 SEK, from the saldo. It should also set a new expiration time which is 90 minutes after the current time. Finally the method should return status `OK`.

Note: In order to implement some of the methods you will need to know the current time. The easiest way to do this is to use the static method `System.currentTimeMillis()`. This will give you the current time in milliseconds. If the time is measured in milliseconds then each new stamp should be valid for 5400000 milliseconds (= 90 minutes). (8p)

4. Implement the method:

   ```
   public static int pow(int x, int n)
   ```

   which computes the `n`-th degree of `x`, i.e. x^n. The computation should be done by a repeated multiplication in a loop. There are different ways to do this, but you should implement the optimal solution where the exponent becomes two times smaller on every iteration. Use the following equations:
   $x^0 = 1$
   $x^{2k} = (x * x)^k, \qquad x^{2k+1} = (x * x)^k * x$

The first equation is the basic case when the exponent is zero. The second equation covers the case where the exponent is even, i.e. `n % 2 == 0`. In that case just replace `x` with `x*x` and divide the exponent by two. Finally, if the exponent is odd then you should do the same but in addition you should multiply the final product with the current value of `x`. Apply this rules in a loop until you reach the basic case `n == 0`.

(10p)

5. The file `flights.txt` contains information about the available flights from one city to another. The following is an example for the file content:

```
Gothenburg London    PZ2122 200
Gothenburg Stockholm PZ3100 100
Gothenburg Stockholm XD3100 300
Paris      Madrid    LL1232 120
Paris      Madrid    XU2232 120
Paris      Madrid    ZA1135 250
```

Every line starts with the names of the initial and the final airport, followed by the code for the flight and its price. You can assume that both the airport names and the flight codes always consist of a single word, and the price is always an integer. This means that parsing the file is possible by using `next()` and `nextInt()` from `Scanner`.

Implement the method:

```
public static List<String> getCheapestFlight(String src,String dst)
  throws FileNotFoundException
```

which takes the names of the initial and the final airports and returns the cheapest flights between these destinations. For example the two cheapest flights between Paris and Madrid are `LL1232` and `XU2232`.

(10p)