# PRG – ETSINF – Second partial lab exam – Academic year 2015-2016
# May 31$^{st}$, 2016. Duration: one hour

(Notice: this exam is evaluated over 10 points, but its weight for the final grade is 1.2 points. So, the values of questions, which are 3, 3 and 4, have a real value in the final grade of 0.35, 0.35 and 0.5 respectively.)

| NAME: | LAB GROUP: |
|---|---|

1. 3 points Write a method `toArray()` in the class `QueueEntries` of the lab practice 5 for returning an array with all the entries in the queue of entries.

   **Notice:** recall that the attributes defined in the class `QueueEntries` are `size`, an integer, plus `first` and `last` that are references to objects of the class `NodeEntry`. And the attributes of the class `NodeEntry` are `value` of type `Entry` and `next` of the class `NodeEntry`.

   **Solution:**

```
public Entry [] toArray()
{
    Entry [] entries = new Entry[size];
    NodeEntry temp = first;
    for( int i = 0; i < size; i++ ) {
        entries[i] = temp.value;
        temp = temp.next;
    }
    return entries;
}
```

2. 3 points Write a method with the profile `public double averageEntries()`, in the class `Bank` of the lab practice 5, for computing the average amount of entries in the accounts of the bank.

   **Notice:** Recall that the attributes of the class `Bank` in the lab practice 5 are: `NodeAccount first` and `int numberOfAccounts`. And the attributes of the class `NodeAccount` are: `AccountEnt value` and `NodeAccount next`. In the class `AccountEnt` it is defined the method `getNumOfEntries()` that returns the number of entries in the account.

   **Solution:**

```
public public double averageEntries()
{
    double avg = 0;
    if ( numberOfAccounts != 0 ) {
        NodeAccount temp = first;
        while( temp != null ) {
            avg += temp.value.getNumOfEntries();
            temp = temp.next;
        }
        avg = avg / numberOfAccounts;
    }
    return avg;
}
```

3. 4 points It is needed to modify the class `Bank` for processing remittance receipts. Each remittance is provided in a text file where each line contains three white space-separated values: `accountId` (an integer

in the range $[10000, 90000]$ ), `amount` (a real value), and `receiptNumber` (a long integer in the range $[MIN\_RECEIPT\_NUMBER, MAX\_RECEIPT\_NUMBER]$ ).

**It is requested** to implemented a method with the profile `public String manageRemittance( Scanner sf )`, that uses the parameter `sf`, already initialised, for reading from the text file. You can assume that all the needed classes have been properly imported. The method should read the data of all the receipts, validating them and charging to the specified account the amount of the receipt. At the end, the method should return an `String` with all the receipt numbers, one per line. If any of the three values in a line is wrong, all the line containing the error or wrong values should be ignored. A message indicating the reason of each error should be shown on screen. You should also consider as an error when the amount to be withdrawn from the account is greater than the balance. The resulting `String` have to contain the valid receipt numbers only. If the file has no valid receipts the method should return an empty `String`.

**Notice:** Recall that the method `getAccount(int)` of the class `Bank` in the lab practice 4 returns the corresponding account if it exists, and `null` otherwise. The method `withdraw(double)` of the class `Account` can throw an exception of the class `IllegalArgumentException` when the amount to be withdrawn is greater than the balance of the account. And the methods for reading of the class `Scanner` can also throw exceptions of the class `InputMismatchException` if the next token mismatches with the type of value to be read.

**Solution:**

```
public String manageRemittance( Scanner sf ) {
    String res = "";
    int accountId = 0;
    double amount = 0;
    long receiptNumber = 0;
    while( sf.hasNext() ) {
        try {
            accountId = sf.nextInt();
            amount = sf.nextDouble();
            receiptNumber = sf.nextLong();
            Account c = this.getAccount( accountId );
            if ( c != null ) {
                if ( MIN_RECEIPT_NUMBER <= receiptNumber
                  && receiptNumber <= MAX_RECEIPT_NUMBER ) {
                    c.withdraw(amount);
                    res += receiptNumber + "\n";
                } else {
                    System.err.println( "Wrong receipt number!" );
                }
            } else {
                System.err.println( "Wrong account id!" );
            }
        }
        catch( InputMismatchException e ) {
            System.err.println( "Incorrect line!" );
        }
        catch( IllegalArgumentException e ) {
            System.err.println( "Incorrect amount!" );
        }
        finally { sf.nextLine(); }
    }
    return res;
}
```