Second Partial Lab Exam – PRG – ETSInf – Academic year 2013/2014
June 10th, 2014 – Duration: 1 hour and 15 minutes

1. [2.5 points] The method `readInt()` reads from an object of the class `Scanner` and returns an integer value belonging to the range `[from, to]`. This method is used from `main()` for reading account numbers of five digits, i.e. numbers $\in [10000, 99999]$.

```
public static int readInt( Scanner sf, String prompt, int from, int top ) {
    System.out.print( prompt );
    int value = sf.nextInt();
    return value;
}
public static void main( String[] args ) {
    Scanner sf = new Scanner( System.in );
    int accountNumber = readInt( sf, "Enter an account number (five digits): ", 10000, 99999 );
    System.out.println( "The account number is: " + accountNumber );
}
```

It is requested:

a) [1.5 points]: Modify the method `readInt()` for checking if the read value belongs to the range `[from,to]`. If the value is not in the range then the method should

- **throw** an exception of the class `IllegalArgumentException` with a message indicating that the entered value doesn't belong to the range
- and **propagates** the exception.

Solution:

```
public static int readInt( Scanner sf, String message, int from, int to )
        throws IllegalArgumentException
{
    System.out.print( message );
    int value = sf.nextInt();

    if ( value < from || value > top )
        throw new IllegalArgumentException( value + " doesn't belong to [" + from + ".." + to + "]" );

    return value;
}
```

b) [1 point]: Modify the method `main()` for catching the exception thrown by the method `readInt()` and showing the message by using the method `getMessage()`. This method is available in all classes derived from the class `Exception`.

Solution:

```
public static void main( String[] args )
{
    Scanner sf = new Scanner( System.in );
    try {
        int accountNumber = readInt( sf, "Enter an account number (five digits): ", 10000, 99999 );
        System.out.println( "The account number is: " + accountNumber );
    }
    catch( IllegalArgumentException e )
    {
        System.out.println( e.getMessage() );
    }
}
```

2. [2.5 points] **It is requested** to write a method that given the name of a text file with the information of the accounts in a bank returns the sum of all the balances of all the accounts. Each line in the file contains two items, the first one is the account number (an integer) and the second one is the balance of the account (a real number). The profile of the method is provided to you and **you don't need to catch any exception**. Please, notice that the method propagates any possible exception that could be thrown in it.

```
public static double sumOfBalances( String filename ) throws Exception
```

Solution:

```
public static double sumOfBalances( String filename ) throws Exception
{
    double sum = 0;
    Scanner sf = new Scanner( new File( filename ) ).useLocale(Locale.US);
    while(sf.hasNext()) {
        int accountNumber = sf.nextInt();
        double balance = sf.nextDouble();
        sum += balance;
    }
    sf.close();
    return sum;
}
```

3. [3 points] Given the classes `Concordance` and `NodeCnc` we have studied in lab practises. The attributes of both classes are the following ones:

```
Concordance                          NodeCnc
-----------                          -------
private NodeCnc first;               String         word;
private int     size;                QueueIntLinked lineNumbers;
private boolean sorted;              NodeCnc        next;
private String  delimiters;
```

With these attributes the concordance is a simple linked list. If your preference is to use double linked lists then two additional attributes are needed: "`NodeCnc previous`" for class `NodeCnc` and "`NodeCnc last`" for class `Concordance`.

It is requested to write a method in the class `Concordance` with the following profile:

```
// PRECONDITION: n >= 1
public boolean moreAppearancesThanN( int n )
```

that returns true if the text used for building the concordance contains one or more words that appear **at least** `n` times.

Solution:

```
// PRECONDITION n >= 1
public boolean moreAppearancesThanN( int n )
{
    NodeCnc temp = first;
    while( temp !=null && temp.lineNumbers.size() < n ) temp = temp.next;
    return temp != null;
}
```

4. [1 point] The public methods of the class `QueueIntLinked` are the constructor, `enqueue(int)` and `dequeue()` that change the state of the queue, `first()`, `isEmpty()`, `size()` and `toString()` for consulting or getting some values.

It is requested to enumerate which of these methods are used in:

a) the class `NodeCnc`

Solution: The constructor and `enqueue(int)`.

b) the class `Concordance`

Solution: `enqueue(int)`, `size()` and `toString()`.

5. [1 point] Explain briefly what is the asymptotic behaviour (constant, linear, quadratic, logarithmic, ...) of the temporal cost function for the method `isSorted()` of the class `Concordance`. This method returns whether the concordance is sorted.

```
public boolean isSorted()
```

Solution: The method `isSorted()` has a constant running time because it is a getter method that returns the value of the attribute `sorted`.