ID1020 – Lab 1
Basic Programming and Data Abstraction

1 Organisation

This lab consists of

a) a theoretical part that you will answer questions about in Canvas (marked with a T) and

b) a programming part that is to be *presented orally* in the lab session and to be submitted in Canvas (marked with a C).

All the questions must be answered in Canvas. For each programming task, you need to upload your java files into its designated answer box in Canvas.

1.1 Goals

This lab has the following goals:

- Introduce the *Java* programming language and its environment (JVM, Netbeans/Eclipse, Maven).
- Introduce Object Oriented Programming (OOP) with respect to *interfaces* and their *implementations*.
- Explore recursive programming techniques and compare with iterative counterparts.

1.2 Requirements

For this lab you will need the following:

- Java SDK, version 6 or newer. Oracle Java is recommended but OpenJDK should work as well.
- Maven
- An IDE like Netbeans or Eclipse is recommended, but any simple text-editor would be enough.

2 Tasks

2.1 Object Oriented Programming (T) – 20P

Consider the following Java code snippets, Answer the questions about the following code in Canvas.

```java
public interface Stack<T> {
 public void push(T value);
 public T pop();
}
```

```java
public class LinkedStack<T> implements Stack<T> {
 private Node<T> first = null;
 public void push(T value) {
        Node<T> n = new Node<T>(value);
        n.next = first;
        first = n;
 }
 public T pop() {
        if (first == null) {
         throw new IllegalAccessException();
        }
        T value = first.value;
        first = first.next;
        return value;
 }
}
```

```java
public class FixedArrayStack<T> implements Stack<T> {
 private T[] items;
 private int top = 0;
 public FixedArrayStack(int size) {
  items = (T[])new Object[size];
 }
 public void push(T value) {
  items[top] = value;
  top++;
 }
 public T pop() {
  top--;
  T value = items[top];
  items[top] = null;
  return value;
 }
}
```

```java
public static main(String[] args) {
 Stack<String> stack;
 if (args[0] == "ARRAY") {
  stack = new FixedArrayStack<String>(args.length);
 } else {
    stack = new LinkedStack<String>();
  }
  for (int i = 0; i < args.length; i++) {
    stack.push(args[i]);
  }
  for (int i = 0; i < args.length; i++) {
    System.out.println(stack.pop(args[i]));
  }
}
```

2.2 Recursion (C) – 40P

Write a *recursive* program that prints out Pascal's Triangle[1] to the command line. That is, implement a recursive method with the signature `public void printPascal(int n)` that prints the first n levels of Pascal's Triangle to standard out (`System.out.print()`) and another recursive method `public int binom(int n, int k)` which calculates $\binom{n}{k}$ and is used by the printing method. In your code be sure to use the identity $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ and **not** $\binom{n}{k} = \frac{n!}{k!\cdot(n-k)!}$.
Make both methods part of a class `RecursivePascal` and use a `boolean` variable to switch between printing the triangle upside down or right-side up (i.e. incrementing or decrementing levels). Do not use any other class variables for your implementation!

Once you have your initial version working, try running it with some bigger n (e.g. $20, 30, 40, 50, \ldots$ stop when it gets very slow.) Think about why it gets slow. Most likely you will notice that you are doing a large number of duplicate calculations. Come up with some kind of memory structure that allows you to avoid some *(max. 17P)* or most *(max. 20P)* of your duplicate operations. This could be one or more class variables. Example for such memory structures could be a (multi-dimensional) array, a map or a simple tree structure. There are no extra points for implementing your own structures at this point, so try to use what Java gives you. Take a look at http://docs.oracle.com/javase/7/docs/technotes/guides/collections/reference.html if you need ideas.

Note: Don't waste (too much) time on making the printout look nice. It is sufficient if it is vaguely identifiable as a triangle. In your optimization, to avoid most of the duplicates you need to take advantage of the triangle properties

2.3 Designing APIs (C)– 20P

Create a Java interface `Pascal` that contains the methods `printPascal` and `binom` from task 2.2. Let your class `RecursivePascal` implement the new interface. Furthermore, create a new class `IterativePascal` that implements the same interface but has an iterative implementation for both methods. Now create an abstract class `ErrorPascal`, have it implement the `Pascal` interface and let your two implementation classes extend the abstract class.

[1] http://en.wikipedia.org/wiki/Pascal%27s_triangle

Now add some basic sanity checking to the parameters of printPascal and binom, e.g. $n > 0, 0 < k \leq n$. Try to avoid code duplication!
Be prepared to discuss the differences (advantages, disadvantages) of interfaces and abstract classes in Java.

Write a Driver class with a method public void main(String[] args) to test both your implementations. Check the correctness of some sample values for binom and simply print some triangles for printPascal. Again try to avoid code duplication!

2.4 Binary Search (T)– 20P

Preparation Download the following two files:

- http://introcs.cs.princeton.edu/java/42sort/BinarySearch.java
- http://algs4.cs.princeton.edu/12oop/largeW.txt

Hint: *You probably cannot copy/paste content of the largeW.txt file because of the size limit in your computer's clipboard. Alternatively you can use wget command on linux/Mac to download it.*

Create a *maven* project and add a BinarySearch Java class, copying the code from the .java file you downloaded (you may need to change the package name in the java file you downloaded). You will now need to add the "stdlib-package" and "algs4-package" dependency and also "sics-release" repository from https://www.kth.se/social/course/ID1020/page/bibliotek-fram-algorithms-bok-stdin-st/ to your pom.xml file. Compile and run the BinarySearch.java program with a single argument, the largeW.txt file. You can download largeW.txt to the root folder of your project, and add its filename as an argument when executing the program from Netbeans, for example. Alternatively, you can run it from the command line using something like this (assuming your classpath is set correctly):

```
>java BinarySearch largeW.txt
489910
115771
123223
13412
121241
110112
```

Questions: Once you have the code working, answer the questions related to this part in Canvas.