Lesson 2

Big Numbers

Write a function which computes the integral power: n^k

```
    public static int ipow(final int n, final int k){
        if(k==0)
            return 1;
        int ans =n;
        for(int i=1; i<k ; i++){
            ans*=n;
        }
        return ans;
    }
```

Unfortunately this function stop working quite soon:

```
    public static void main(String[] args) {
        int n=2;
        for(int i=0;i<100 ; i++)
            System.out.println(n+"^"+i + "\t"+ ipow(n, i));
    }
```

```
2^0     1
2^1     2
2^2     4
2^3     8
2^4     16
2^5     32
2^6     64
2^7     128
2^8     256
2^9     512
2^10    1024
2^11    2048
2^12    4096
2^13    8192
2^14    16384
2^15    32768
2^16    65536
2^17    131072
2^18    262144
2^19    524288
2^20    1048576
2^21    2097152
2^22    4194304
2^23    8388608
2^24    16777216
2^25    33554432
2^26    67108864
2^27    134217728
```

```
2^28    268435456
2^29    536870912
2^30    1073741824
2^31    -2147483648
2^32    0
2^33    0
2^34    0
2^35    0
2^36    0
2^37    0
2^38    0
2^39    0
...
```

Yes you say the problem is that `int` cannot hold large number as 2^{31}.
Let's write a new version which does not (in our dreams) **overflow**:

```
public static long ipow(final long n, final long k){
        if(k==0)
            return 1;
        long ans =n;
        for(int i=1; i<k ; i++){
            ans*=n;
        }
        return ans;
    }
```

```
2^0     1
2^1     2
2^2     4
2^3     8
2^4     16
2^5     32
2^6     64
2^7     128
2^8     256
2^9     512
2^10    1024
2^11    2048
2^12    4096
2^13    8192
2^14    16384
2^15    32768
2^16    65536
2^17    131072
2^18    262144
2^19    524288
2^20    1048576
2^21    2097152
2^22    4194304
2^23    8388608
2^24    16777216
2^25    33554432
2^26    67108864
2^27    134217728
2^28    268435456
```

```
2^29    536870912
2^30    1073741824
2^31    2147483648
2^32    4294967296
2^33    8589934592
2^34    17179869184
2^35    34359738368
2^36    68719476736
2^37    137438953472
2^38    274877906944
2^39    549755813888
2^40    1099511627776
2^41    2199023255552
2^42    4398046511104
2^43    8796093022208
2^44    17592186044416
2^45    35184372088832
2^46    70368744177664
2^47    140737488355328
2^48    281474976710656
2^49    562949953421312
2^50    1125899906842624
2^51    2251799813685248
2^52    4503599627370496
2^53    9007199254740992
2^54    18014398509481984
2^55    36028797018963968
2^56    72057594037927936
2^57    144115188075855872
2^58    288230376151711744
2^59    576460752303423488
2^60    1152921504606846976
2^61    2305843009213693952
2^62    4611686018427387904
2^63    -9223372036854775808
2^64    0
2^65    0
2^66    0
2^67    0
2^68    0
```

We extended the range of our function up to 2^{62} but it's still failing.
We need an integral type with **infinite precision**. `BigInteger` provided by `Java.Utils` solves the problem.

```java
    public static BigInteger ipow(final long n, final long k){
        if(k==0)
            return BigInteger.ONE;
        BigInteger bin = BigInteger.valueOf(n);
        BigInteger bians = bin;
        for(int i=1; i<k ; i++){
            bians=bians.multiply(bin);
        }
        return bians;
    }
```

This version works! even for very large numbers like 2^{20000}.

```
2^20000 39802768403379665923543072061912024537047727804924259387134268656523863597493005704267600
...
```

Imagine now you want to compute number much larger than 2^{20000}. You soon realize that this takes quite a while using this approach.
Infact the following code runs quite slowly (~ `6s` on my Intel `i5`)

```
public static void main(String[] args) {
        int n=2;
        long s = System.currentTimeMillis();
        for(int i=0;i<500000 ; i++)
            if(i==400000)
                System.out.println(n+"^"+i + "\t"+ ipow(n, i));
        long e = System.currentTimeMillis();

        System.out.println((double)(e-s)/1000.0);
    }
```

That is due to the fact that we are not using the best algorithm available for computing integer power:

```
    public static BigInteger ipow(final int n, final int k){
        if(k==0)
            return BigInteger.ONE;
        if(k==1)
            return BigInteger.valueOf(n);
        if((k&1)==1)
            return ipow(n,k-1).multiply(BigInteger.valueOf(n));

        BigInteger res = ipow(n,k/2);
        return (res).multiply(res);
        //return ipow(n,k/2).multiply(ipow(n,k/2));

    }

ipow(2,22)
    ipow(2,11)                                  -> return ipow(2,10) *n
      ipow(2,10)                                -> return ipow(2,5) * ipow(2,5)
          ipow(2,5)                     -> return ipow(2,4) * n
              ipow(2,4)             -> return ipow(2,2)+ipow(2*2)
                  ipow(2,2)     -> return n*n
                    ipow(2,1)-> return n;


time ->-----------------------------------------------------------------
```

The same main now runs in `~0.18s` . A speedup of `~33x` !

Garbage Collection and Immutable Strings

Java String does not provide any methods for modifying a string! That makes Strings `immutable` .

This is what happen you do

```
String s = "hello";
s = s.substring(0,3) + " blabla";
```

```
char* temp = malloc(9);
strncpy(temp, greeting, 3);
strncpy(temp + 3, "blabla", 6);
greeting = temp;
```

What happens if a String is assigned to another string like in the following?

```
String s = "hello";
...

s = "bye bye old string";
```

This is the perfect receipt for a memory leak. `s` was allocated on the heap and is now thrown away.
Java takes care of memory leaks like this using a **garbage collector**!

Java does not provide an deallocation mechanism because the garbage collector keeps track of all the object that cannot can be still referenced. Those who are not referenceable are deallocated by the JVM.

It has three main tasks:
1. Garbage detection i.e. understand when an object is now garbage.
2. Deallocate the heap referenced by the detected garbage
3. Minimize Heap Fragmentation (more in a operating system course)

GC marks all the objects that are reachable starting from the root object and at the end of this operationg it can free the memory of those object which are not marked.

Pink and blue boxes are colliding while green and blue are not for instance.

What you have to know at this point about fragmentation is that when a a large number of allocation and deallocation take place in the heap it is not hard to have a situation where an object cannot be allocated because the heap is lacking of subsequent space even if the overall free space in the heap is much larger than the amount required to allocate the new object.

At this point you should have covered everything about syntax and have seen concepts like
- `Exceptions`
- `Collection`

Hello Classes

A class is a descrition both the **state** and the **behaviour** of an Object.
Think about the state of a traffic light. At any point what are the possible state a traffic light can be?

Interfaces

In the Java programming language, an interface is not a class but a set of requirements for classes that want to conform to the interface.

Sorting and IComparable Interface

```
    /**
```

```
     *
     * @param lhs
     * @param rhs
     * @return True if lhs is strictly greater than rhs
     */
    boolean compare( final IComparable rhs);

}
```

We want to write a sorting function that is able to sort all objects that implements the interface `Icomparable`

```
public class BubbleSorter {


    public static void sort(IComparable[] elements){
        boolean go = true;
        while(go){
            go=false;
            for(int i=0 ; i< elements.length-1; i++){
                if((elements[i].compare(elements[i+1])))
                {
                    IComparable t=elements[i];
                    elements[i]=elements[i+1];
                    elements[i+1]=t;
                    go=true;
                }
            }
        }
    }

}
```

Let's implement a class Person which supports the compare funcitonality.

```
package Sorting;

public class Person implements IComparable{

    private String name;
    private Integer age;
    @Override
    public boolean compare(IComparable rhs) {
        if(name.length() > ((Person)rhs).name.length())
            return true;
        else
            if(name.length() < ((Person)rhs).name.length())
                return false;

        return (age > ((Person)rhs).age) ? true : false;
    }

    @Override
    public String toString() {
//      String s = name.length()+" ";
//      return s;
```

```
        StringBuilder sb = new StringBuilder();
        sb.append("[" + name);
        sb.append(" "+age+"]");
        return sb.toString();

    }
    /**
     * @return the name
     */
    public String getName() {
        return name;
    }
    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }
    /**
     * @return the age
     */
    public Integer getAge() {
        return age;
    }
    /**
     * @param age the age to set
     */
    public void setAge(Integer age) {
        this.age = age;
    }



}
```

Bubble sorter is a general, since it can sort objects of different kind as soon as they implements the interface.

```
    @Test
    public void testBubbleSort(){

        Random r = new Random(System.currentTimeMillis());
        IComparable[] persons = new Person[100];
        for (int i = 0; i < persons.length; i++) {
            int l = 1+r.nextInt(20);
            persons[i] = new Person();
            ((Person)persons[i]).setAge( 1+ r.nextInt(100));

            ((Person)persons[i]).setName(RandomString.randomAlphanumericString(l));
        }

        for (int i = 0; i < persons.length; i++) {
            System.out.print(persons[i].toString() + " ");

        }
        System.out.println();
```

```
        BubbleSorter.sort(persons);
        for (int i = 0; i < persons.length; i++) {
            System.out.print(persons[i].toString()+ " ");
        }
        System.out.println();

    }

    public static void main(String[] args) {
        BubbleSorterTest bst = new BubbleSorterTest();
        bst.testBubbleSort();

    }

}
```

Do computer programs have a state? If yes what are the *variables* of a computer program state?

Bookstore

Show the implementation in Eclipse.

What are getters and setters?
Show how to automatically generate them using *eclipse*

- *Source -> Generate Getters and Setters*
- *Source -> Generate toString* thanks to Paola Arcuri

```
import java.util.Date;

/**
* A object representing a simplified version of a book.
* A book has a Date of publishing, an author, a number of pages expressed as an integer and
* a String representing the International Standard Book Number.
*
* @author Davide Spataro
*/
public class Book {

    private String author;
    private Date year;
    private int npages;
    private String ISBN;



}
```

```
package BookStore;

/
public class BookStore {

    private Book[] library;
```

```
}
```

Make sure you get how to comment the code in the proper way. Check out the Chapter 4 of the `Java Core I` book.

How to implement the remove method?

What about the search method?