Lecture 3, Reviewing Java
Students' questions

Q. When calculating space&time complexity, what do you look for?
A. A good training on complexity is the first assignment. I ask you to design a basic algorithm and to evaluate its space and time complexity. Implicitly, I told you that the question itself may not make sense: there is not always one complexity but several, since different cases (best, worst, average, …) lead to different results. So, I would suggest that you go the following way:

1. Space complexity.
   Are all cases the same? If so, find how much memory will be consumed and simplify use O operator.
   Are there different cases? Unless you're asked explicitly for one case, you should discuss the best and the worst case. For each, you find the situation that leads to it and how much memory is consumed in that situation. Then simplify.
2. Time complexity.
   Same but instead of memory you count the number of operations.

Q. static stuff?
A. Let say that I have a class People. There is a function getName(). This only makes sense on an object: I can ask "what is the name of this person" but I cannot ask "what is the name of a person"? When there is a question that you can ask for any instance (*i.e.* any object), this question can be *static*. Here's the sequence for a question that you ask to a specific instance:

```
People p = new People("Jean", 21);    //create an instance with name Jean, 21 years old
System.out.println(p.getName());      //show the name of that instance
```

Here's a sequence for a question you can ask to the class:

```
System.out.println(People.getMaximumHeight());
```

This question asks: what's the maximum height that any person can have? We do not have to create an instance to ask this question, so getMaximumHeight() is static. In the example we had in class, we did the same thing with a Contract: I ask "give me a new contract", so this question does not depend on an object and can be static.

Q. Why is the main function static?
A. Good observation. Indeed the signature of the main function is:

```
public static void main(String[] args)
```

Why? Your program is started by a mythical creature called the Java Virtual Machine (JVM). That creature wants to enter in the program directly: just one line *Main.main(arguments…)*. If it wasn't static, the creature should have two lines: one to create an instance

```
Main m = new Main();
```

and one to call the method:

```
m.main(arguments);
```

Other than being lazy with one line instead of two, there is a problem if we have to use the constructors instead of calling the static method. There may be several constructors: how should the JVM figure out which one it feels like using?

Q. throw exception?
A. Let's consider an example: you have a function that returns the result of the division of *a* by *b*. Here's one way to write it:

```
public double divide(double a, double b){
        return a/b;
}
```

However you cannot divide by 0. It is illegal, punished by death (at least). So what can you do if the user provides you with *b* equal to 0? This function has to return something, but you can't return a special value that would mean "this didn't work". Indeed, all values may be used for normal behaviour of the function. Let say you want to return -1 to say that the function had a problem: -1 may be a normal value for a = -3 and b = 3. So the question becomes: what can you do when there is a problem in the function and there is no way you can return a special value to indicate the problem?
We start looking at multiple channels of communication between function: the regular channel that returns a double is too busy, so you need another channel. This is possible by throwing exceptions. In the regular channel, you gently *return* something: in the exception channel, you brutally *throw* stuff. The exception channel is there for emergency such as abnormal behaviours. How does it work? It's exactly the same that the regular channel: you have to say what will be returned (thrown), and you throw it at some point in the function.

```
public double divide(double a, double b) throws Exception{
        if(b==0.0)          // this is very bad by the way but we'll ignore it for now
                throw new Exception("Division by 0!");
        return a/b;
}
```

If b is not 0.0 then we will return a/b, using the regular channel. Otherwise, we will throw an Exception using the emergency channel. An Exception is just an object and you can embed a message in it. The person who has called the function normally has a way of catching what is being thrown, so it will catch the Exception and read the message in it. That's just a rough way to communicate.

Q. this.ID = ID?
A. That's a problem with variables name. Let's look at an example:

```
public class Frenchman{
        private String typeOfWine;                  // variable of the class
        public Frenchman(String typeOfWine){        // variable of the function
                this.typeOfWine=typeOfWine;
        }
}
```

In this case, when we are in the constructor, we have a little issue: there are two variables called with the same name, typeOfWine. What we want to do is to store the variable given to the function into the variable of the class. How can we distinguish between them those two variables if they have exactly the same name? We use *this.* to refer to the variable of the class. So when we write "this.X = X;" it means "store the content of the variable X into the class' variable called X". Personally, I don't see the point of giving exactly the same name to a function argument and to a class variable: it's not like we are that limited in the number of names we can choose, so creating a confusion and solving it with this seems somewhat unnecessary. How would I write that constructor? Simply by using another name:

```
Public Frenchman(String wineType){
        typeOfWine = wineType;
}
```

Q. What is Mafiosi?
A. Literally eh? The *mafia* is the name of a society. Its members are referred to as *mafiosi*. In Italian, you have genders and plural: generally, "o" is the singular for a male-typed word and "i" the plural for a male-typed word. *Mafioso* is for one member of the mafia, *mafiosi* is the plural. So:

Io sono un Mafioso. → I'm a mafioso.
Noi siamo mafiosi. → We're mafiosi.

I don't remember why there is the article "un". That's just how I'd say it. ☺
Ok, maybe you actually were serious about that… So there is a variable in class Company:

```
private Executant[] mafiosi;
```

This is an array in which each cell contains an object of type Executant (which is another class that we defined). The array is named mafiosi (remember, several, plural in *–i*…). If you want another example:

```
private int[] blurp;// declares an array named blurp, in which elements are of type int
```

Q. if(mafiosi[i] == null) continue;
A. In the constructor, there is more space in this array than what we are really using. So in some of the cells, there is an object, and in some cells there isn't. When there isn't, what you have is *null*. If you try to access an element that is null, it will give you an error: NullPointerException. You can't ask a question to something that does not exist. In order for our program not to crash, we just want to skip the null elements in our array. To do so, we test if the current element is null: if so, we *skip it*. To skip something when you're in a loop, you write continue: this goes directly to the next iteration. For example, this will only print "1…3…5…7…9…", and skip the numbers "0…2…4" :

```
for(int i=0; i<100; i++){
        if(i%2==0) continue;
        System.out.println(i);
}
```