Exercises module 13 – Constructor

13.1 – Constructor

1. Given this class representing a User:

```
public class User
{
    public String Name;
    public String City;
    public String Country;
    public int Age;
}
```

We can always initialize it using the code below, but using a constructor is often better.

```
User u1 = new User();
u1.Name = "Kalle";
u1.City = "Stockholm";
u1.Country = "Sweden";
u1.Age = 25;
```

2. Add a **default constructor** that initializes all the fields to their default values and try it! With default value, we mean empty string for strings and zero for integers.

3. Add a second **constructor** that sets all the fields in the class via the parameters to the constructor.

4. Age and Country are optional values, create a **third constructor** that only takes **Name and City** as input parameters and set the other fields to their default values.

13.2 – Fix the bug

1. In the following code that compiles there is a bug, can you fix it because the current output is:

```
null
0
```

The code:

```java
Public class Main {

    public static void main(String[] args){
        // write your code here

        Car car = new Car("Volvo", 20);
        System.out.println(car.model);
        System.out.println(car.age);
    }
}

class Car
{
    public String model;
    public int age;

    public Car(String model, int age)
    {
        model = model;
        age = age;
    }
}
```

13.3 – Nested objects

1. We need to add the user **home address** to our **Order** class below:

```
public class Order
{
    public int id;
    public String Product;

}
```

2. Add the fields for the **home address** (**name, addressline1, addressline2, zipcode, city, country**) to the class above.

3. Now we got a new requirement to add the **delivery address** to the class. Add the same fields as above to the object and give all the fields some sensible names.

4. Some customers also require a separate **invoice address**, so add the invoice address fields (same as above) to the class. Make sure all the fields in the class have sensible names so that the end-user of the class can understand which to use where.

   As you notice, naming things is one of the most difficult thing in computer science.

   So clearly having an object with too many fields can sometimes be a bit ugly.

5. In our Order class, we now have three separate addresses, home, delivery and invoice.
   As developers, we usually want to **reuse things as much as possible**.

   In the code we can extract a new high-level concept/type called "Address" because it can make much more sense to have a dedicated Address type.

   It's also a matter of **separation of concerns,** by breaking out the address fields into an **Address** type, we can reduce the clutter. The Order class can now focus on the Order details and the Address type can focus on the Address issues including validation.

   If you have time, read more about "Separation of concerns" on the web.

6. Create a new class named Address and replace the address fields in the Order class with the following:

```
public class Order
{
    public int id;
    public String Product;

    public Address Home;
    public Address Delivery;
    public Address Invoice;
}
```

7. Create a Constructor to the Order class that initializes all the fields above.

8. Create three address object instances, one for Home, Delivery and Invoice.

9. Then create a new Order instance and pass in the three address fields to it, like:

```
Order order = new Order(13, "Pokemon", home, delivery, invoice);
```

10. To make sure it works, add code to print out to the console the **invoice address** by asking the order object for it.

Questions and concepts to study further on your own:

- Default constructor
- Having multiple constructors in a class
- When is the constructor called?
- What happens if we have a private constructor?
- The default values for the various data types, string, bool, integer, object…
- Data Transfer Objects (DTO)
- Separation of Concerns
- Single Responsibility Principle