Exercises module 12 – Basic OO

12.1 – Searching array of users

1. Given this class representing a User

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

Create a class named **SearchUsers** that contains a method named **getMaxAgedUser** that takes an **array** of users as input and returns the age of the user with the highest age.

Call the method with the following code: (Make sure you understand it!)

```
User[] users = new User[3];

User u1 = new User();
u1.Name = "Kalle";
u1.City = "Stockholm";
u1.Country = "Sweden";
u1.Age = 25;
users[0] = u1;

User u2 = new User();
u2.Name = "Joel";
u2.City = "Malmö";
u2.Country = "Sweden";
u2.Age = 55;
users[1] = u2;

User u3 = new User();
u3.Name = "Adam";
u3.City = "Helsingborg";
u3.Country = "Sweden";
u3.Age = 45;
users[2] = u3;

SearchUsers susers = new SearchUsers();

int maxage = susers.getMaxAgedUser(users);

System.out.println("max age= " + maxage);
```

12.2 – Searching array of users

1. Add a new method named **getUserByName** to the **SearchUsers** class that will search for the first user with a given name and return the found user back to the caller.

   If no user is found then return null;

   Call the method with code like:

```
User result = susers.getUserByName(users, name);
if(result != null){
    System.out.println("Found " + result.Name + " from " + result.City);
}
else{
    System.out.println("User not found!");
}
```

12.3 – Recursion (Advanced)

1. What happens if a method calls itself?

   Try running this code:

```
class DoStuff{
    public void doSomething(){
        doSomething();
    }
}
```

   Eventually you will get a **StackOverflowError** exception

   Read and research online what a **java.lang.StackOverflowError** means. For example, read: https://stackoverflow.com/questions/3197708

   The problem is that you will run out of memory because the stack is full. The stack is a special memory area that the computer uses to keep track of all the method calls. By calling the same method and never returning back from it, we eventually run out of stack memory.

2. When a method calls itself, we call this **recursion**. Read online about what recursion is before you continue.

3. Can we use recursion to something useful?

Yes we can, for example we can calculate the sum of all the numbers between **1..n** with code like:

```
class DoStuff {
    public int doSomething(int input) {

        System.out.println("Input=" + input);

        if (input > 0) {
            return input + doSomething(input - 1);
        }

        return input;
    }
}
```

Try calling the code with and try to step through the code and try to understand how it works.

```
System.out.println("The sum is: " + stuff.doSomething(5));
```

The result should be:

```
Input=5
Input=4
Input=3
Input=2
Input=1
Input=0
The sum is: 15
```

4. Can we use this recursion to do something really useful? Yes we can! For example, we can use recursion to get a list of all the sub-directories in a given folder.

   Add a new method to the class used in the previous step and call it with the following code:

```
String path = "c:\\Program Files";
stuff.getDirectories(Paths.get(path));
```

5. Create the **getDirectories** method and in it add the following code to list all the directories found in the provided path:

```
public void getDirectories(Path path) throws IOException {

    try (DirectoryStream<Path> stream = Files.newDirectoryStream(path)) {
        for (Path entry : stream) {

            System.out.println(entry);
        }
    }
    catch(Exception ex)
    {
        System.out.println("Can't access :" + path);
    }
}
```

6. Run the code and you should see a list of all the directories found in that directory.

7. But how can we also list all the directories found inside each directory found? How would you do that?

   Feel free to try do to that by yourself without using recursion, but soon you will find that to be quite hard to get right.

   A better solution is to use recursion by calling the method above each time we find a directory.

   Add the following code inside the for loop above:

```
for (Path entry : stream) {

    if (Files.isDirectory(entry)) {
        System.out.println(entry);

        getDirectories(entry);
    }
}
```

   Run the application again and you should now see a list of all the directories and their sub-directories listed. Step through the code and make sure you understand how it works.

   As you might notice you might not have access to some of the found directories.

Questions and concepts to study further on your own:

Get familiar with the concepts of:

- Encapsulation
- Data abstraction
- Reusability
- Behaviour
- State
- Recursion
- What does Stack overflow mean?