EPITECH. TECHNOLOGY

CASH MANAGER

BOOTSTRAP

CASH MANAGER

OOP with Java

We assume that you are already at ease with **Object Oriented Programming** in general, and have some experience of Java in particular.

However, let us start with some java practising in order to get in shape, and check your level.

We intend you to do the following steps:

- ✓ create a class `User` with some attributes and two methods: `login` and `viewSettings`.
- ✓ the first one has two arguments `name` and `password`, and switch an attribute `isLogged` to True.
- ✓ the second one reads and displays the content of a file *Settings.txt*.
- ✓ create a class `Admin` that inherits `User` and has a third method: `changeSettings`.
- ✓ this method has two arguments `field` and `value` and write `field:value` at the end *Settings.txt*.
- ✓ create a third class with a `main`, which simulates and assess the behaviour of the above mentioned methods.

Once the programs are written, build them, export the jar and verify it runs correctly.

If you don't feel comfortable with these questions, get updated before starting this project.

Building project with maven

The aim of that part is to get familiar with the maven tool, while introducing a useful library, **Jackson**, which maps json text to java objects.

We intend you to do the following steps:

- ✓ create a maven Project with two components model and controller ;
- ✓ in the POM file, set the java version to the most recent and add Jackson as a dependency ;
- ✓ in the model component build a class `Account` ;
- ✓ in the controller component write a class that can read a json file and instantiate the corresponding Account Object ;
- ✓ write a `main` method that takes a json as an argument, build the account object and displays information about it ;
- ✓ build the jar ;
- ✓ in the resources section write a json file myaccount.json whose syntax is compliant with the class above ;
- ✓ test your jar on that file.

```
Terminal
T-DEV-700> cat comptes.json
{“name”:“Jason”,“cash”:“34.50”}
```

```
Terminal
T-DEV-700> mvn package
...
[INFO] -- maven-jar-plugin:2.4:jar (default-jar) @ maven --
[INFO] Building jar:
eclipse-workspace/jaspack/target/jaspack-0.0.1-SNAPSHOT.jar
...
[INFO] BUILD SUCCESS
...
[INFO] Finished at: 2019-04-13T18:38:53+02:00
...
```

```
Terminal
T-DEV-700> java -jar jaspack-0.0.1-SNAPSHOT.jar comptes.json
Jason owns $34.50
```

If you don’t feel comfortable with these questions, get updated befor starting this project.

Organising your components

Our project is well organized but kind of lack of substance.
Let us populate it with components and classes.

We want you to build an app that:

- ✓ asks the user (trough the console terminal) a name and a list of number of bank accounts ;
- ✓ creates the appropriate `User` and `Account` objects ;
- ✓ displays these objects in the terminal.

Do not write everything in a single class or even package, but introduce some modularity:

- ✓ have some research on design patterns ;
- ✓ choose a pattern that suits you and organise your code in several packages, one for each component.

```
Terminal
T-DEV-700> java -jar jaspack-0.0.1-SNAPSHOT.jar
What is your name?Jason
What are your account numbers?230071,873330,664552
Account number:230071 Owner:Jason Money:$0.0
Account number:873330 Owner:Jason Money:$0.0
Account number:664552 Owner:Jason Money:$0.0
```

Of course, these packages are quite scarce at this stage, but they form the base structure you can later populate with more features.

If you don't feel comfortable with these questions, get updated befor starting this project.

Code coverage with JUnit

Now that we have a decent project, let us produce automated tests on it. The paradigmatic library for that purpose in Java is JUnit; you can use it as a standalone, but here we will integrate it in our Maven build.

As you can see, Maven duplicates your source packages in two top-level repository named `main` and `test`. Tests (*methods with @Test annotation*) you write in the second one should be matched with corresponding classes and method in the first. JUnit uses a wide range of assert methods, depending of the kind of output you expect.

In that part we intend you to:

- ✓ think of all possible elementary tests you can make on that program (attribute types and values, local variables, exception raised, etc) ;
- ✓ implement these tests with the appropriate expectation ;
- ✓ build the test coverage.

Terminal

```
T-DEV-700> mvn test
...
-------------------------------
T E S T S
-------------------------------
Running Controler.ConverterTest
Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.042 sec
Results :
Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
```

In the future, you might want to build tests before the actual program. This is a way to enforce one aspect of OOP: considering the interfaces and the use cases **before** implementing real classes.

If you don't feel comfortable with these questions, get updated befor starting this project.

Creating & building an Android project

You'll call Github REST API and especially the endpoint GET repository *(ex: repos/googlesamples/android-Camera2Basic)* then print the received data. You'll need to use Retrofit for your API calls.

First, you have to prepare your HTTP client:

- ✓ install Android Studio with the latest version of the Kotlin plugin installed.
- ✓ create a new project with an *Empty Activity* and the language Kotlin.
- ✓ create a data class `Repository` with some properties (`id`, `name`, `full_name` and `html_url` are required).
- ✓ create an interface for your endpoints.
- ✓ create a class who builds the *Retrofit* client and implement your interface.

Look *Retrofit* documentation to know how to create the interface. You have to use built-in annotations.

Now let's get the data form Github and show it:

- ✓ in the layout `activity_main` add `TextViews` and other components if necessary to show your data.
- ✓ in the `MainActivity` class you can now get data and set them in the view.

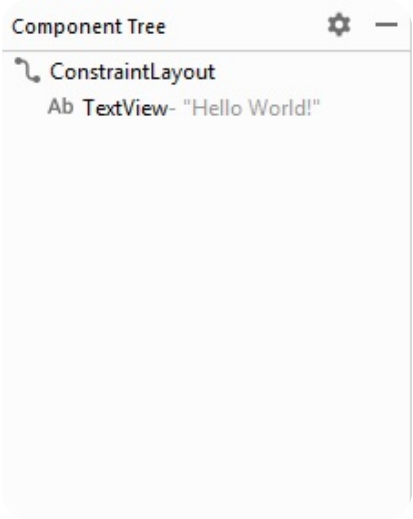
Now you're able to show a repository, you can add features to train yourself, like a list of repositories and the detail when you click on one element.

In this exercise, you only used `Activities` but `fragments` exist too.

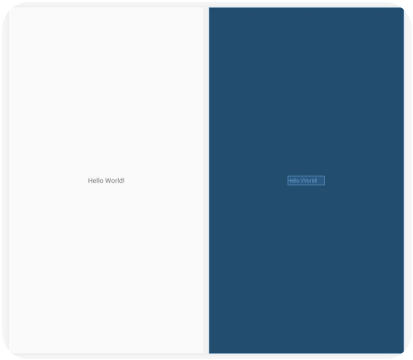
Your UI is very basic. To improve it, you have to understand how works user interface in Android and the XML syntax.

The user interface uses a hierarchy of Layouts and Widgets. You can see Layouts as containers positioning Widgets on the screen. Widgets are components like `TextView`, `Button` or `ImageView`.

Previously you used the file `activity_main.xml`. You can see a component tree in this file.



In this tree, the layout `ContraintLayout` is the container and the widget `TextView` is the component (showing “Hello World!”).
You can change the position of elements with the properties. By default the layout takes the size of his parent (the screen in this example) so the widget is in the middle of the screen/layout.



Properties are usefull to move elements, but also to custom background color or visibility for instance. Some elements have specific properties like `TextView` with `textColor`, `textSize`,...

If you need help to create your interface, you can use Material Design.

Now, create a nice and handy interface.

v 2.0