ECS414U - Object Oriented Programming
Week 2 lab session (5 marks)

Bruno Ordozgoiti

1 February 2022

Rules & instructions: This assignment is individual work. You must complete it by yourself. You can discuss it with your demonstrator and other students in your lab session, and you are in fact encouraged to do so. However, you must not share your solution with other students, so that they can submit it as their own, or request that others share their solution with you. This would result in disciplinary action.

It is mandatory to attend the lab session in order to get the marks. You must report to your demonstrator.

You are given a program consisting of three files: `BankingApp.java, Account.java` and `User.java`. You must modify the source code to satisfy the list of requirements below. When you compile and run the program, a series of tests will be run. When you think that your code is ready, you must submit the files `Account.java` and `User.java` to your demonstrator for them to run the tests. You will get one mark for each passed test (a test is passed if and only if it outputs `Test OK`). Your demonstrator and the module staff may ask you questions about your code, if something is unclear. In that case, only once the answers to these questions are clear, you will get 1 mark for the corresponding test. If any issues are encountered, you can keep working on the code and try again for the duration of the lab session.

You may look at `BankingApp.java` to make sure you understand the requirements of the code in order to pass the tests.

Important: You must submit your solution (`Account.java` and `User.java`) to QM+ before 23:59 Sunday, 6 Feb. This applies to everyone, even those of you who completed the assignment and showed it to your demonstrator in the lab. The activity to submit your code is *Week 2 lab submission*. If you didn't have time to get your code evaluated by your demonstrator during the lab, it will be assessed after the submission deadline.

Advice: Try to follow good coding practices and the principles of OOP mentioned in the lectures. Even though a passed test will get you the marks, there are many ways to write code, and some are arguably better than others. Try to discuss your approach with your demonstrator and fellow students, and try to understand whether there is something you could have done better. When you get the solutions, compare them to your code. The more effort you put into writing good code, and

the better you understand the concepts involved in these exercises, the better your chances will be of doing well in the assessed tests and mini-project.

List of requirements:

R1 Each `Account` object must have a unique identifier of type `int`. This value must be readable via a method called `getUniqueId` defined in class `Account`.

R2 Users can have up to three different accounts. The class `User` must implement a method `addAccount`, taking an `Account` object as argument, to assign a new account to a user. If we try to assign a fourth, the program should print an error message and do nothing else. The total funds available to a user —i.e. the sum of the balances of this user's accounts— should be readable by a method called `getFunds` in class `User`.

R3 Users should have methods to deposit and withdraw funds, called `deposit` and `withdraw`. Both must take exactly two arguments, in the following order: and `int` specifying the account to deposit to/withdraw from (0,1 or 2) and an `int` specifying the amount to deposit/withdraw. If a non-existing account number is given (e.g., the user only has one account but number 2 is given), the program should print an error message and do nothing else.

R4 The class `User` must have a method to generate a report. The method must be called `generateReport` and return a `String` in the following format: `Name: <name>. Funds: <funds>`, with the values of the corresponding fields.

R5 Each bank account pays an amount of yearly interest. There must be a constructor that takes an `int` parameter for the initial balance and a `double` parameter for the interest rate, in that order. The default amount is 1%. The `User` class must implement a method, `calculateEarnings`, that returns the expected earnings given their currently available funds.

Example: Suppose a user has three accounts, with respective balances $100, 50, 20$. The accounts give returns of 4%, 2% and 1% respectively. Then the expected returns can be calculated as

$$0.04 \times 100 + 0.02 \times 50 + 0.01 \times 20.$$