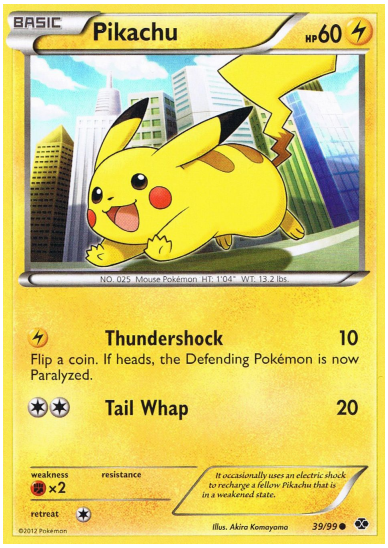
UNIVERSITÉ DU LUXEMBOURG

Lab 2: Object-oriented modelling

Programming Fundamentals 2

2nd March 2021



Goals

- ✯ Coding and understanding the *dynamic array* data structure.
- ✯ Object-oriented design and coding of a management-like application.
- ✯ From specification to implementation.

- This laboratory can be done by group of at most 2 students (alone is also fine).
- **Special**: group of a "mentor" and a "student".
- **Deadline:** 15th March 08:00AM

Deliverables

1. The code on your Github repository generated by clicking here: `https://classroom.github.com/g/IN1vRr5V`
2. A presentation video of 6' minutes on FLIPGRID: `https://flipgrid.com/c652f2a9`
   - One video per group: mention your respective contributions in the video.
   - Use your @student.uni.lu email to connect.
   - Share your screen and comment the Exercise 3 (Pokedeck).

Exercise 1 – Soon-a-coding-legend-but-not-today exercises

These exercises must be done by the "student", with the help of the "mentor" if one was attributed to you. If you don't have a mentor, don't hesitate to ask for help to *Ezhilmathi KRISHNASAMY* (`ezhilmathi.krishnasamy@uni.lu`) by email or on Discord. These exercises are mandatory for all groups containing at least one student who failed the first lab. If you succeeded the first lab, you can skip these exercises.

1. Create a file `Soon.java` with the following code:

   ```
   package lab2;
   public class Soon {
     public static void main(String[] args) {
     }
   }
   ```

   Compile the file with `javac -d target src/lab2/Soon.java` and execute it with `java -cp target lab2.Soon`. The following questions must be written in this `main` method, unless specified otherwise.

2. Ask the user two integers and print them.

3. Ask the user two integers and print the maximum and minimum of these two integers.

4. Extract the minimum and maximum of two integers inside the functions `public static int max(int a, int b)` and `public static int min(int a, int b)`.

5. Ask the user how many numbers s.he wants to type. Then, read those numbers in a loop, and keep track of the minimum and maximum elements.

6. Same question as before, but this time, use an array to store the elements first. Then, iterate on the array to find the minimum and maximum elements.

7. Create a file `ComputeArray.java` with the following code:

   ```
   package lab2;
   public class ComputeArray {
     public int[] data;
     public ComputeArray() {
       data = new int[10];
     }
   }
   ```

   In the class `Soon`, create an object `ComputeArray array = new ComputeArray()`.

8. Add a method to add an element to this array:

   ```
   package lab2;
   public class ComputeArray {
     //..
     public void set(int idx, int x) { ... }
   }
   ```

   Use this method in `Soon` to populate the array with 5 numbers.

9. Add two methods in `ComputeArray` to find the minimum and maximum elements:

   ```
   package lab2;
   public class ComputeArray {
     //....
     public int maximum() { ... }
     public int minimum() { ... }
   }
   ```

   Use these methods in `Soon` and make sure they are working properly.

10. Create three classes (each in a different file) `Rectangle`, `Triangle` and `Circle`, each with a method `draw()`. Put a main function in a file `AdvancedGeometry.java`. The goal is to develop a small drawing program, where the user can choose which shape to draw as well as the dimension of this shape. Do not forget to start very small, and add new functionalities step by step!

Exercise 2 – From scratch: `ArrayList`

`ArrayList` is a very useful class in Java for creating *dynamic array*. For now, we used *fixed-size array* such as:

```
int[] fixed_array = new int[10];
```

which creates an array of integers of size 10. If, at one point in the program, we want to store more than 10 elements, then we must create a new array, that is larger, and copy the smaller array into the larger one. This is tedious. So, we will create a class `DynamicArray` that makes this process transparent for the user.

1. The cloned repository comes with a file `DynamicArray.java` that is filled with holes! Read the comments in this file and complete the unimplemented methods, one by one! More exactly, do one commit for each new method you add. The file `DynamicArrayTest.java` test your class, do not modify it (unless you go plus ultra). There are 4 tests in the auto-grader for this exercise.

2. **Plus Ultra.** (This question is only relevant to those who completely terminated the Pokedeck.) Add any additional methods to this class such as for sorting the array, or for returning a slice of the array (*e.g.*, `slice([1, 4, 7, 8, 9], 2, 4) = [7, 8, 9]`). Extend this class in any direction.

Exercise 3 – Pokedeck

For this project, methods or functions longer than 10 lines are forbidden! Decompose, decompose, decompose!

The goal of this project is to implement a *Pokedeck*, a very useful app allowing us to manage (**and not to play**) our Pokemon cards collection. Your main resource for this project will be the official card game book: `https://assets.pokemon.com/assets/cms2/pdf/trading-card-game/rulebook/xy1-rulebook-en.pdf`. Beware, however, that there are *too many information* in this PDF. You need to sort out the information relevant to this project. Unlike the first laboratory, we give only the main directions of this project and implementation details are left to your wisdom.

For this exercise, you will compile with *Maven*. Install Maven with `sudo apt install maven` on Windows or Ubuntu in the terminal, and with `brew install maven` on OSX. A file `main/java/lab2/pcg/Pokedeck.java` is already present, you will need to add new files in this folder too.

- To compile, write `mvn compile` at the root of your project (as usual).
- To run the program, write `mvn exec:java -Dexec.mainClass="lab2.pcg.Pokedeck"`.

The main advantage of Maven, is that you can easily add dependencies (see plus ultra question below).

1. **Analysis and modelling.**
   Important note: This question must be done in parallel to the other questions. Follow a cycle of designing - implementing - designing - ...

   The first step is to analyse the domain of the problem (here, Pokemon cards), and find out a computer representation suited to solve the problem. Concretely, we draw a class diagram with the various classes, attributes and methods, that we need. Following the same practice than in laboratory 1, we must decompose this project in very small steps. Otherwise, our diagram might quickly escalate to out of reach features, such as a playable Pokemon card game. For instance, you should start with a simple menu (in the terminal, of course!), such that, besides being printed,

selecting an item does nothing. You can then implement the last one which exits the program. Further, you should not immediately propose a class with all the attributes of a card (e.g. attacks, HP, kind, ...)! *Start small* (e.g., just the name of the card), implement it, test it, and then add another *small* feature.

Create a directory `doc`, in which you will place a PDF of the class diagram. You can either draw it by hand (as cleanly as possible!) and scan it, or use a dedicated software such as MS-VISIO, BOUML or DIA (the last two are free), or the web-based tool I used in the lectures: `https://yuml.me/diagram/scruffy/class/draw`. You will also add a small *design rational* document (PDF as well), which explains your architectural choices, and comment the UML class diagram.

Remember the key is to develop **step by step** following a compile-run-test-fix-commit-push workflow.

2. Our Pokedeck must have the two following basic functionalities:

   - Add the description of a card.
   - Delete a card from our collection.

   You can use `ArrayList<Card>` to represent a collection of cards. `ArrayList` follows the same principles than our class `DynamicArray`, but contains much more methods, see for yourselves in the doc: `https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/ArrayList.html`

3. It is quite hard to know if your Pokedeck is actually working in all situations... In `.github/classroom/autograding.json` you will find some tests for the autograder system. Add new tests for your Pokedeck that test some corner cases, and basic scenario. These kind of "black-box tests" are very useful for testing terminal-based user interfaces. For the next questions, create new tests for each new functionality. Before jumping to the next functionality, make sure what you have created is actually working!

4. The previous release of our Pokedeck impressed many people, but we now have many feature requests. In particular, our users want to:

   - Update a card.
   - Print the cards collection.
   - Search for a card according to various criteria. You must implement at least two search on different criteria (for instance, search the card by collection number, search the card by name).

   Did the design you proposed in the previous question could be easily extended to these new features? What were your mistakes? You can discuss that in the design rational document. Don't forget to add new tests!

5. **Plus Ultra.** Currently, we cannot save our cards collection. Indeed, when we close our app, all changes are lost... That's pretty inconvenient. Therefore, we want to add a "save" and "load" features that allow users to save their cards collection in a file. A very readable and interesting format is JSON (see `https://www.w3schools.com/js/js_json_intro.asp`). You can find many Pokemon cards described as JSON format here: `https://github.com/PokemonTCG/pokemon-tcg-data`. We will need to add a dedicated JSON library as a dependency to your Maven project (see this tutorial `https://mkyong.com/java/how-do-convert-java-object-t`

6. **Plus Ultra.** Extend the game in any direction you want, be sure to mention the extensions in the FlipGrid video.

Exercise 4 – Competitive track

Those interested in the competitive track must register here (you can join anytime): `https://docs.google.com/spreadsheets/d/1KMZx58SoE08g-l4usphtaLFnPhKzBDhTpa9PgixOok8/edit?usp=sharing`.

See Laboratory 1 for the submission instructions. This time, the exercises are:

- *10038 - Jolly Jumpers* (easy!): `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=979`
- *11475 - Extend to Palindrome*: `https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=2470`
- *11462 - Age Sort* (smart!): `https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=2457`
- *10978 - Let's Play Magic!*: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1919`
- *11581 - Grid Successors* (for math lover!): `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=2628`

Good luck!