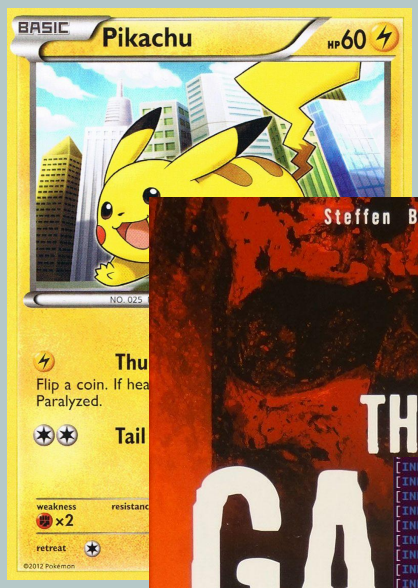
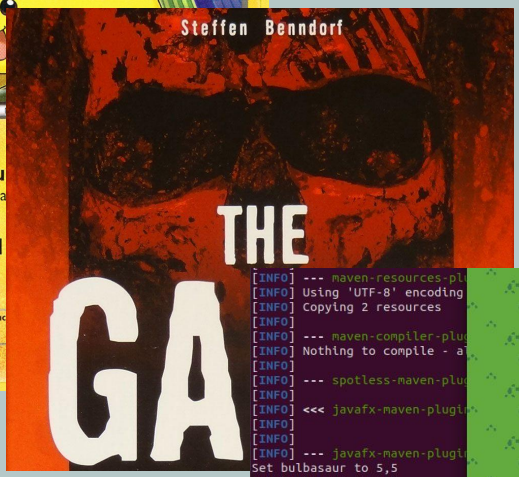
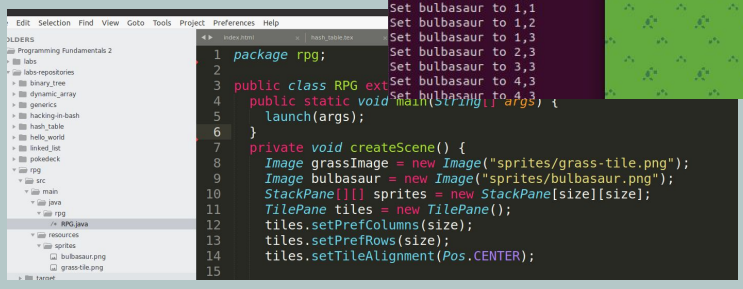
Programming Fundamental 2

LET'S CODE!

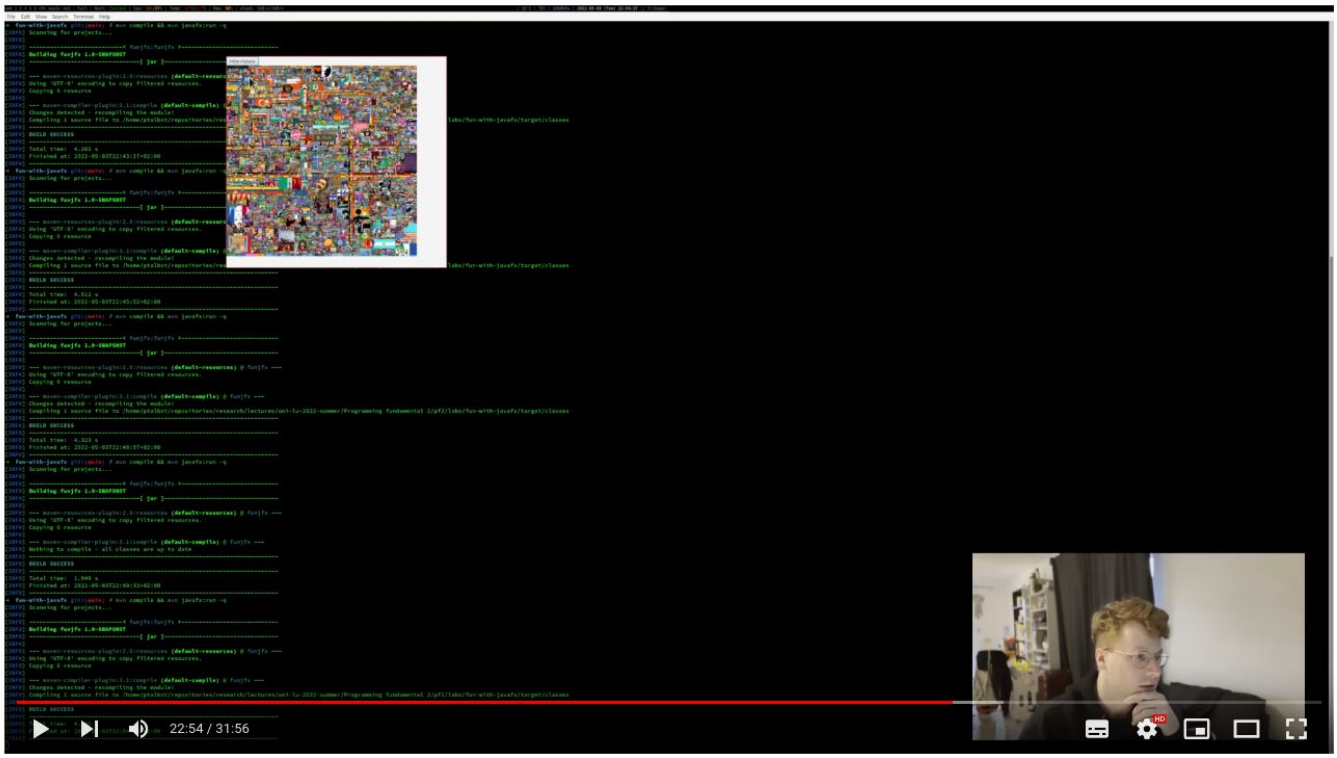






Start your journey

PF2



Simple graphical interface with JavaFX

Non répertoriée

PF2 is designed to be a journey where you learn at your pace the object-oriented paradigm with the Java language. More than coding, you will learn the best software engineering practices and important tools surrounding programming. It includes build automation with Maven, testing with JUnit, documenting with Markdown and JavaDoc, source versioning with Git and Github, and editing with Sublime Text.

PF2 will be hard and time consuming but we have designed this class so you can make the most out of it. **The lectures are recorded online** in short YouTube videos. **The laboratories have no deadline** so you can work at your pace. If you do not finish all the laboratories on time, you resume this class next year where you left off. The goal is to let you study in detail every concept, and to give you a chance to progress even if you failed PF1. In contrast, if you are motivated, you might be able to finish most of this class in just 1 month!

Reviewing

WE CARE ABOUT THE CODE YOU WRITE.

WE CAREFULLY REVIEW YOUR CODE AND YOU UNLOCK THE NEXT LABORATORY ONLY ONCE IT IS GOOD ENOUGH.

A << B << C

For each laboratory, you get a letter:

- A: Success
- B: Minor review
- C: Major review

If you get B or C, you will need to revise your code and resubmit it for review. The reviews are done by the teaching team every Monday morning. If you submit your laboratory Monday afternoon, you will have to wait one week to get your review, so you should be well organized.

To avoid being stuck waiting for reviews, you will have several laboratories unlocked in parallel that you can complete in the order you wish.

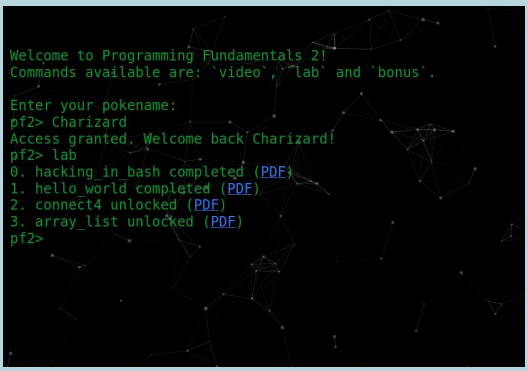
But beware, **time is counted**! You must not wait too long to submit your laboratories otherwise you might have not enough weeks left to complete the course. Note that you will need to complete a bunch of core laboratories to pass the class.

Because of this rather unusual reviewing system, and because you do not necessarily unlock laboratories in the same order than your peers, we have made a website where you can track your progresses:

www.hyc.io/pf2/

There you can find all the laboratory PDF sheets and the recorded videos. It also gives you the reviews letters for all your laboratories.

To access this website, you will need a Pokename, which is a unique identifier that is given to you on the first lecture.



4 Pillars

Programming is not just the pure activity of typing lines of code. This class will teach you the foundation of 4 essential pillars to become a better programmer! The laboratories will strengthen your foundation for each pillar.

Infrastructure.
There is a lot to learn when it comes to managing a programming project: how to automate builds, share your code, document and test your code, etc. The infrastructure pillar is transversal to all laboratories, although the very first one will set the stage.

Labs: `hacking_in_bash`

Object-Oriented Modelling.
This is the main purpose of this class: to teach you how to program in the object-oriented paradigm. We will start by learning the syntax of Java and move forward with object-oriented concepts such as encapsulation and the various kinds of polymorphisms.

Labs: `hello_world`, `the_card_game`, `space_exploration`, `pokedeck`

Data Structures.
Data structures are at the heart of computer science. They give us various ways to organize our data with different tradeoffs. This would deserve an entire class, so we just cover four basics data structures here!

Labs: `dynamic_array`, `linked_list`, `generics`, `hash_table`, `binary_tree`

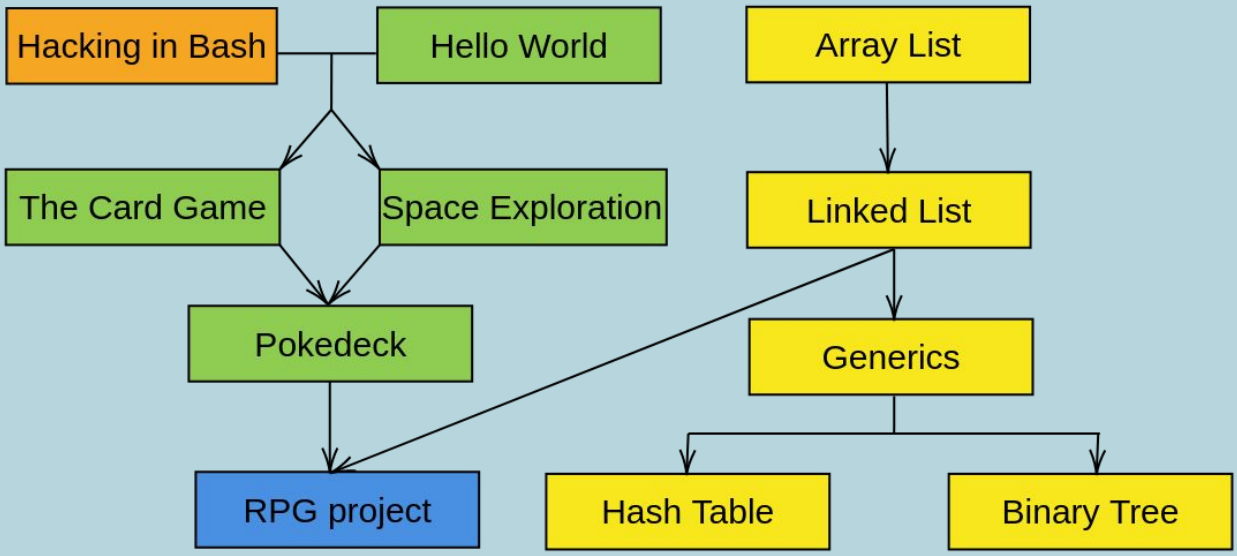
Team Work.
Working in a team is essential to build large software application. But it is also harder and need more management and planning to distribute the tasks correctly among the team members. The final laboratory of this class is a team project!

Labs: `rpg`

To unlock the final project and have the right to make a team, you will have to complete the laboratories `hacking_in_bash`, `hello_world`, `the_card_game`, `space_exploration`, `pokedeck`, `dynamic_array` and `linked_list`.

Labs Tree

EACH COMPLETED LABORATORY UNLOCKS ONE OR MORE NEW LABORATORIES. HERE IS THE DEPENDENCIES TREE.



No deadline is a double-edged sword. With flexibility comes greater responsibilities... Therefore, you might prefer to follow this planning with around one laboratory per week.

Suggested planning

- ❑ 27/02 hacking_in_bash
- ❑ 06/03 hello_world
- ❑ 13/03 array_list
- ❑ 20/03 the_card_game
- ❑ 27/03 space_exploration
- ❑ 03/04 linked_list
- ❑ 17/04 pokedeck
- ❑ 24/04 generics
- ❑ 01/05 hash_table
- ❑ 07/05 binary_tree

RPG Project.

You can compose your team of three people and start the RPG project once all your team members got an A or a B on all the required laboratories (see previous page).

The oral presentations for this project will be **during the last class on 30/05.**

Grading

> **The main goal of this class is to make you a better programmer. You should be only concerned by the knowledge you will gain in this class :)**

There is 70% of continuous evaluation spread among the laboratories:

- `5% hacking_in_bash`
- `5% hello_world`
- `5% array_list`
- `5% the_card_game`
- `5% space_exploration`
- `5% linked_list`
- `10% pokedeck`
- `5% generics`
- `5% hash_table`
- `10% binary_tree`

The remaining 30% will be the project. And yes, I know, it does not add up to 100%. There is 10% missing... Keep reading.

Quick concept tests.

We plan to have five in-class tests, to study various aspects of Java, that are more difficult to study in the laboratories. The topic and date of the test will be announced one week before the actual test. Among the topics we will study:

- Memory representation of objects.
- Unified Modelling Language (UML).
- The Object class.
- Polymorphic cocktail.
- Time complexity of data structure operations.

Your best 4 out of 5 tries will be worth 10% (4 * 2.5%).

Asking questions

Asking a question is actually something we need to learn! There is a number of things you should always do before asking a question, in particular you should carefully read the instruction of the laboratory, and research online an answer to your question. Do not hesitate to read it from the beginning. There are sometimes small details that we easily overlook on the first read, but that are very important.

How to ask a question:

- ❑ Avoid private messaging the teaching team, ask in the Discord channel #ask-anything. Unless it is for private issues (health or others), in which case you should use email.
- ❑ Be brief. Write your sentence first, and then summarize it. It takes time and editing to ask for something. But since you ask for help, it is your job to make the task of the person who will help you easier.
- ❑ Don't ask to ask, just ask.
- ❑ If there is a compilation error, or a runtime error, include the error when you ask the question. Or take a screenshot. We cannot guess what is your problem.
- ❑ Include a minimal version of the code posing problem, it should be small and hard to reuse by someone else working on the same project. Basically, don't spoil the fun of figuring out a solution to your peers.
- ❑ Additional infos: https://stackoverflow.com/help/how-to-ask

The Small Programming Handbook

Pierre Talbot
University of Luxembourg, Luxembourg
pierre.talbot@uni.lu

Contents

1 Shell Cheat Sheet

Lexicon:

- *Terminal*: Computer program in which we can type commands (aka. the black box in which you type stuff).
- *Shell*: The language of the terminal.
- *Filesystem*: The organization of all your files and directories in your computer. For instance, it remembers which files is in which directory, and which directory in which directory.
- *Path*: Sequence of directory's name separated by slash, which optionally terminates by a file's name. Example: `/home/ptalbot/Downloads/Shingeki\ no\ Kyojin.mkv`, `../Documents/`. (spaces in directory or file's names must be escaped by a backslash).
- *Root directory*: This is the topmost directory which is contained in no other directory. It is named `/`.
- *Home directory*: Each user of a computer has a home directory, this is where you put all your directories and files (*e.g.*, the directory which contains `Documents`, `Downloads`, `Pictures`, ...). When you open a terminal, you are placed in your home directory.
- *Absolute path*: A path starting by the root directory, *e.g.*, `/home/ptalbot`.
- *Current directory*: It has the special name `.` (just a dot). It is the directory in which you are when you are typing in the terminal.
- *Change directory*: The command `cd /home/ptalbot` allows you to navigate in your filesystem from the terminal.
- *Relative path*: A path relative to where you are in the current terminal, *e.g.*, `cd Documents/` brings you to the directory `Documents` if the current directory has a subdirectory named `Documents`.
- *Parent directory*: It has the special name `..` (yes, just two dots), by typing `cd ..` you move to the parent directory of the current directory.

Commands:

- `pwd`: Show where you are in the filesystem.

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- `cd /home/ptalbot`: Change the current directory to the path `/home/ptalbot`. Note that all commands accepts absolute and relative paths.
- `ls`: List the content of the current directory.
- `mkdir ProjectZ`: Create a new directory inside the current directory, named `ProjectZ`.
- `touch ProjectZ/secret.txt`: Create a new empty file named `secret.txt`, by using a relative path, the file will be created inside the directory `ProjectZ`, which is inside the current directory.
- `javac -d target src/lab1/HelloWorld.java`: Compile the Java program named `HelloWorld.java` into a class file `HelloWorld.class` in the directory `target`.
- `java -cp target lab1.HelloWorld`: Execute the `main` function of the class `HelloWorld` which was compiled in the directory `target`.

2 Clean Coding Checklist

You should follow the Google Java Style Guidelines. This will be checked automatically in your projects (after Hello World) with the Maven plugin `spotless`. Here is a summary of what you should check before submitting your project:

- ☐ My project is developed step-by-step, one commit and functionality at a time.
- ☐ My project compiles without errors.
- ☐ Indentation is correct and coherent.
- ☐ Naming:
  - ☐ Variables, functions and packages start with lowercase.
  - ☐ Classes start with uppercase.
  - ☐ Functions and variables name use the "camelCase" notation (e.g., `computeSum`, `averageOfGrade`) and not `compute_sum`.
  - ☐ Constants (static final attributes) are in uppercase (`DAMAGE_PER_SECOND`).
  - ☐ Names are explicit and precise.
- ☐ No code is copy pasted, all similar treatments are encapsulated inside methods.
- ☐ Methods are short (10-15 lines of code at most).
- ☐ Classes:
  - ☐ One class per file.
  - ☐ Classes have a *single responsibility*.
  - ☐ I don't have a *god object*, *i.e.*, a class that does everything.
  - ☐ Get and set are almost absent of my program. When I use a get/set I'm asking myself: "Why do I need this method?", and instead try to put that treatment inside the class itself.
  - ☐ I don't use static attributes unless for the use cases seen in class.
- ☐ I don't declare variables before I need them, *e.g.*, `Student s;` then later `s = ...`. Instead write `Student s = ...;`.
- ☐ Input/output is uncoupled from the business logic, *e.g.*, most methods should be reusable if we switch to a graphical user interface.
- ☐ The documentation is useful (*e.g.*, not `//returns a string.`).
- ☐ Because everything is well decomposed and methods are small, the documentation is primarily the code.
- ☐ My project can be compiled and ran using Maven (`mvn compile`).
- ☐ Use `for(Card card : deck)` instead of `for(int i = 0; i < deck.size(); ++i)` when possible and when iterating over collections.

3 Git Cheat Sheet

Lexicon:

- *Repository*: A place where you store your software code.
- *Git repository*: A repository managed by the tool *git* to have the whole history of the project.
- *Local repository*: The git repository on *your computer*.
- *Remote repository*: The git repository on *a server* such as Github.

Commands (to type in a shell):

- `git clone https://github.com/ptal/amazing_game.git`: Create a local git repository by downloading the remote git repository at `https://github.com/ptal/amazing_game.git`.
- `git add Hello.java`: Add the file `Hello.java` to the local git repository.
- `git commit -a -m "Fix a bug where users could enter their age."`: Commit the local change to the local repository.
- `git push`: Uni-directional synchronization from your local repository to the remote repository.
- `git pull`: Uni-directional synchronization from the remote repository to your local repository (especially useful if several people work on the same project).
- `git status`: Show the current state of your local repository, it tells you if files are not tracked (were not added), or changes not committed or not pushed.

4 Quick and dirty workflow

Clone, compile and run

```
git clone https://github.com/ptal/amazing_game.git
cd amazing_game/
javac -d target src/lab/Game.java
java -cp target lab.Game
```

Create file, add, edit, compile, run, commit, push

```
touch src/lab/Game.java
git add src/lab/Game.java
subl -n .
javac -d target src/lab/Game.java
java -cp target lab.Game
git commit -a -m "Add new feature: users can now enter their age."
git push
```

5 Sublime Text Cheat Sheet

See `Preferences > Key Bindings` for all shortcuts.

Classic shortcuts:

- `ctrl + n`: Open a new tab in the current editor window.
- `ctrl + a`: Select everything in the current file.
- `ctrl + f`: Find a pattern (words, function's name, ...) in the current file.
- `ctrl + shift + f`: Same as ctrl + f, but in the whole project.

- `ctrl + r`: Replace a pattern (words, function's name, ...) with something else in the current file.
- `ctrl + shift + r`: Same as crtl + r, but in the whole project.

Cool shortcuts:

- `ctrl + p`: Go to any file in the directory opened in Sublime Text.

6 Java Pitfalls

1. In command `java`: Be careful to use `lab1.HelloWorld`, the directory and the class are separated by a dot and not a slash!
2. When running the commands `javac` and `java`, you must be at the root of the repository just cloned! If you type `ls`, you should see the `src` directory.
3. Don't forget to write `package lab1;` at the top of the file. It must match the folder path in which the file is stored (below `src/`). For instance, if you have a file `src/myapp/gui/App.java`, then the file `App.java` must start with `package myapp.gui;`.
4. `package lab1;` (or similar) must be the first statement of the file.
5. Don't forget to import the class `Scanner` with `import java.util.Scanner`.
6. In order to obtain meaningful feedback from the automated correction tool, you should stick to the output we show here (that also holds for the next exercises).
7. Shadowing of the attributes:

```
public class Concert {
  private int startTime;
  private int endTime;
  public Concert(int startTime, int endTime) {
    startTime = startTime;
    endTime = endTime;
  }
}
```

   You need to write `this.startTime = startTime`, otherwise you assign a local variable (the one passed as parameter) to itself! Which has no effect...