

# Design Patterns - TP 1

TD1 initial code. This is a template for the students' assignments.

## Assignment info

**LAST NAME**

BRUEL

**First Name**

Jean-Michel

**Group #**

☒ Teachers

☐ 1

☐ 2

☐ 3

☐ 4

☐ Innopolis

## Requirements

You'll need:

☒ A [GitHub](#) account

☐ A [Git Bash](#) terminal (if you use Window\$)



Try the following command in your terminal to check your **git** environment:

```
git config --global -l
```

## Initial tasks

- ☒ Click on the Github Classroom link provided by your teacher (in fact, this should be done if you read this).
- ☐ Clone on your machine the Github project generated by Github Classroom.
- ☐ Modify the README file to add your last name, first name and group number.
- ☐ Commit and push using the following message:

fix 0 Initial task done



In the following, every time you'll see à **fix ...** text, make sure all your files are committed, and then push your modifications in the distant repo, making sure you used the corresponding message (**fix ...**) in one of the **commit** messages.



- If you want to check that you're really ready for **fix 0**, you can run the command in your shell: **make check**.
- If you want to list the Todos of the day, run **make todos**.

# 1. The Adventure game

## 1.1. Basics

1. Using *Strategy*, and using the package organization provided at [the end of this doc.](#), write the classes and behaviors of the adventure game of TD1 (section 1.7.2):

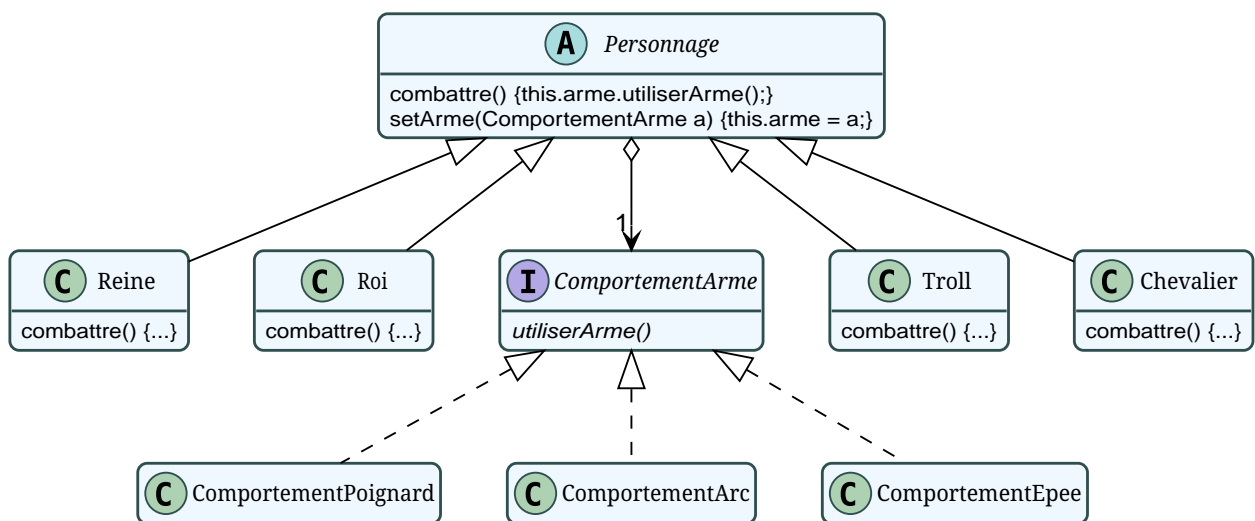


Figure 1. Minimal adventure game

As this is your own code, please write your Classes in English:

Table 1. Translations French/English/Russian

FR	US	RU
Personnage	Character	Персонаж
Reine	Queen	Королева
Roi	King	Король
Troll	Troll	Троль
Chevalier	Knight	Рыцарь
ComportementArme	BehaviorWeapon	ВладениеОружием
ComportementPoignard	BehaviorKnife	ВладениеНожом
ComportementArc	BehaviorBow	ВладениеЛуком
ComportementEpee	BehaviorSword	ВладениеМечом
tirer()	fight()	сражаться()
setArme()	setWeapon()	установитьОружие()
utiliserArme()	useWeapon()	использоватьОружие()

 commit/push

fix 1.1: Initial Adventure Game

## 1.2. Improvements

We want to force any designer of a new **Character** to systematically decide the weapon behavior at the instantiation time.

1. Propose a solution to this problem.



it is perfectly authorized to create a constructor for an abstract class...

2. Try this solution by modifying your application

 commit/push

fix 1.2.2: Initial Adventure Game

3. Test to add a new **Character** concrete class who does not apply with this new regulation and see that it does not compile.

fix 1.2.3: Initial Adventure Game

## 1.3. Dynamic Behavior

Let's experiment how it is possible now to change at runtime.

1. Write an app where, from a menu, you can create a **Character** by selecting a type, and you can then select his/her weapon.
2. Check that, when you change the weapon, the associated behavior change too.

fix 1.3: Dynamic Behavior

## 1.4. Reverse engineering of the code

You have to provide the final class diagram of your application. You can use any valuable technique to get this diagram:

- Here is an **ant** example:

*Example of an **ant** file*

```
<project default="main">
<target name="main">
<javadoc doclet="de.mallox.doclet.PlantUMLDoclet"
  docletpath="plantUmlDoclet.jar"
  access="private"
  additionalparam=
    "-encoding utf-8 -J-DdestinationFile=uml.txt -J-DcreatePackages=false -J
-DshowPublicMethods=true -J-DshowPublicConstructors=false -J
-DshowPublicFields=true"
  >
  <packageset dir="../src">
    <include name="**"/>
  </packageset>
</javadoc>

<java jar="plantuml.jar" fork="true" maxmemory="128m">
  <arg value="uml.txt"/>
</java>
</target>
</project>
```

- Here is a Windows script example:

### Example of a .BAT file

```
set UML=TD1.uml
set TYPE='PNG'
set DOCLETPATH=
echo "Creating %UML%..."

javadoc -J-DdestinationFile=%UML% -J-DcreatePackages=false
-J-DshowPublicMethods=true -J-DshowPublicConstructors=false
-J-DshowPublicFields=true -doclet de.mallox.doclet.PlantUMLDoclet
-docletpath plantUmlDoclet.jar src/appli/*.java
src/behaviors/weapon/*.java src/behaviors/weapon/impl/*.java
src/Characters/*.java
echo "Done."

set TYPE='png'
echo "Converting %UML% to $TYPE..."
java -jar plantuml.jar -config "config.cfg" -t %TYPE% %UML%
echo "Done."
```

- There is a new doclet available: <https://github.com/gboersma/uml-java-doclet>
- Directly get the code from an eclipse plugin **Window › Show View › Other... › PlantUML › PlantUML Source**.



#### TODO:

- ☐ Place the resulting plantUML version of your diagram in a **doc** folder at the root of your repo, with the exact name: **TP1.plantuml**
- ☐ When everything is OK, push your code:

 **commit/push**

fix 1.4: Final TP1 commit

- ☐ Check your autograding on your repo

The autograding will use the `tests/checkModel.rb` ruby file. For example, when applied to the following model:

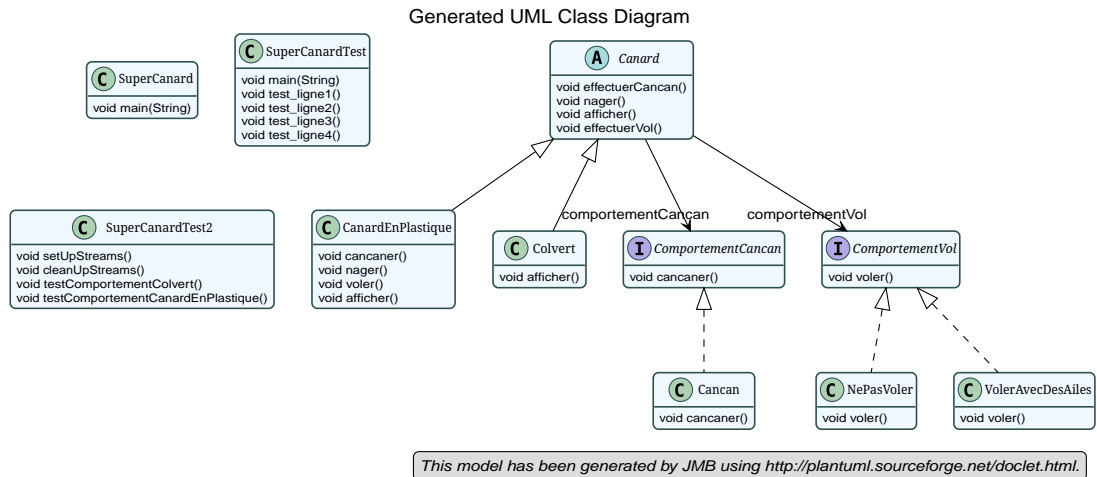


Figure 2. `TP1.plantuml` graphical version (source: `tests/[TP1.plantuml]`)



the test (the one dedicated to `SuperCanard`, in `tests/checkSuperCanardModel.rb`) will provide the following result:

```

tests — jmb@MBPdeJeanMichel — ../master/tests — -zsh — 80x18
[→ tests git:(master) × ruby checkSuperCanardModel.rb
MiniTest::Unit.after_tests is now Minitest.after_run. From checkSuperCanardModel
.rb:25:in `<main>'
MiniTest::Unit::TestCase is now Minitest::Test. From checkSuperCanardModel.rb:27
:in `<main>'
Run options: --seed 33202

# Running:

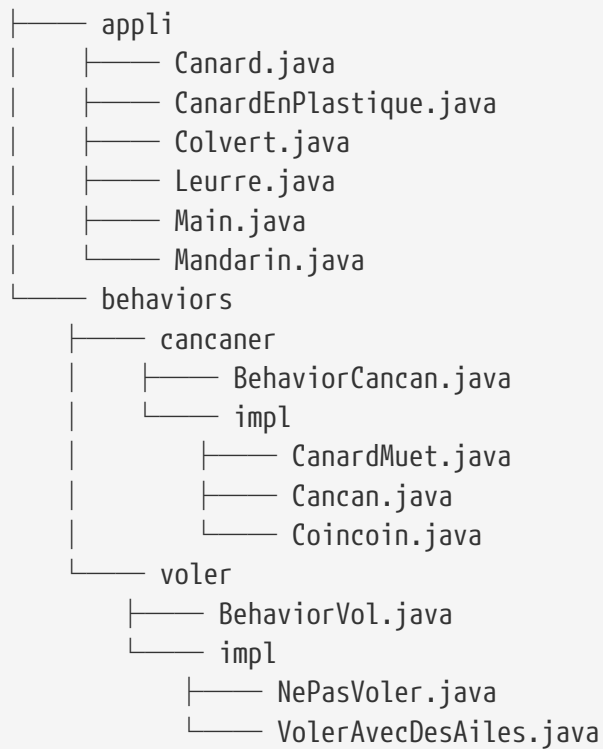
.....

Finished in 0.001910s, 4712.0419 runs/s, 6282.7225 assertions/s.

9 runs, 12 assertions, 0 failures, 0 errors, 0 skips
nil
→ tests git:(master) ×
  
```

Figure 3. Check result for `TP1.plantuml`

## Appendix A: Example of a package organization (taken from `SuperCanard`):



## Appendix B: Still hungry?

1. Test the limits of your implementation (for example what happens if someone does `new Knight(null,null)` ?).
2. Propose some tests in the same way `testfix1.1` was testing the commits in `TD1`.


 `commit/push`

fix 1.5: Give me a bonus I did: [please explain]

## Contributors

- [Jean-Michel Bruel](#)

## About...

Document réalisé via {asciidoctorlink} (version `2.0.11`) de 'Dan Allen', lui même basé sur {asciidoc}. Libre d'utilisation et géré par la 'Licence Creative Commons'.  [licence Creative Commons Paternité - Partage à l'Identique 3.0 non transposé](#).