SOAR – LAB 2

Introduction to Code Versioning, Debugging, loggers, Refactoring and Profiling

Objective: This lab will provide an introduction to several key concepts of effective and collaborative code development

Step 1: Login in to your GitLab Account: https://gitlab.unil.ch

You should be able to login using your UNIL credentials
(username (not email), UNIL password)

Gitlab is an Open source software to collaborate on code

Step 2: Download and Install Sourcetree: https://www.sourcetreeapp.com/

Sourcetree is a free git client to visualize and manage your repositories

Step 3: Versioning

Clone the repository containing the source code of todays lab on your local machines to perform todays Lab exercises.

Enter the following into sourcetree [New -> clone from URL]
Source URL: https://gitlab.unil.ch/adiallo4/soar_lab2.git

In case, you are on Linux machine, you can directly clone the repository using the following command in the Terminal

git clone https://gitlab.unil.ch/adiallo4/soar_lab2.git

After cloning, you have to create a new project using your Gitlab account
(https://gitlab.unil.ch/projects/new)

Navigate to your project using the terminal, follow the instructions under **"Existing folder or Git repository"**

Step 4: Coding

a. Open the downloaded project in NetBeans

b. Execute the code [Right click on the project -> Run]

c. Make sure that the code Builds and Executes
 (You should be able to see the game screen) (don't play too much)

Step 5: Versioning

Create a branch in Netbeans [Team -> Branch -> Create Branch]
Use you last name as the name of the branch

Step 6: Coding

Switch to your branch [Team -> branch -> switch to branch -> choose your branch]

Add the following line in the main class
system.out.println("It works!");

Commit this change on your branch [Team-> Commit -> Enter commit message -> commit]

Now push [Team -> Remote -> Push]

Step 7: Using Loggers

Navigate to the Main Class file and declare the logger

private static final Logger LOGGER = Logger.getLogger("Hello");

Then in the main method, show a message using Logger

LOGGER.info("It works");

Now, repeat it for Class Enemy to check the collision.

Step 8: Debugging

Add a breakpoint in the `Enemy.checkCollision()` method and run in debug mode (click the icon next to the run button)

In order to add a breakpoint, you simply need to click on the corresponding code line.

Now the game will stop once there is a collision.

Step 9: examine local variables, objects and method calls in the stack trace

Step 10: rename the `Enemy.isdead` variable into `Enemy.isgone` with preview before actually doing the refactoring
[Rick click on the variable -> refactor -> rename -> preview]

Make sure to preview before refactoring to check all the changes performed due to renaming.

Now click on Do Refactoring.

Step 11: Similar to Step 10, rename the `Enemy.follow()` method into `Enemy.chase()` with preview before actually doing the refactoring

Step 12: Now rename the Enemy class into `EnemyImpl` with preview before actually doing the refactoring

Step 13: extract an interface from the `EnemyImpl` class (will all public methods) and name it Enemy, explain the rational by saying you want to add new types of enemies in the game (not just crows)

[Right click on the class -> Refactor - > Extract interface -> Select all methods -> Rename new interface to Enemy]

Step 14: run the project in profiling mode in telemetry mode (graphs)

[Profile project -> configure session -> Telemetery]

Now click the Profile button in the tool bar to collect profiling data

Check the memory usage, CPU, Garbage collection and other stats.

Step 15: run the project in profiling mode in method mode (time spent in each method / hotspots)

[Profile -> Methods]

Here you see execution time and stats related to each method.