A Live Environment for Inspection and Refactoring of Software Systems - Empirical Experiment

We are working on a **Live Refactoring Environment**. With this approach, software developers will have access to several **Extract Method refactoring** suggestions that aim to improve the quality of their software systems. This refactoring environment was developed for **Java** code on the **IntelliJ IDE**, and it focuses on analyzing specific quality attributes to detect particular code smells. By identifying different smells, this environment selects and applies possible refactoring methods capable of mitigating the respective smells.

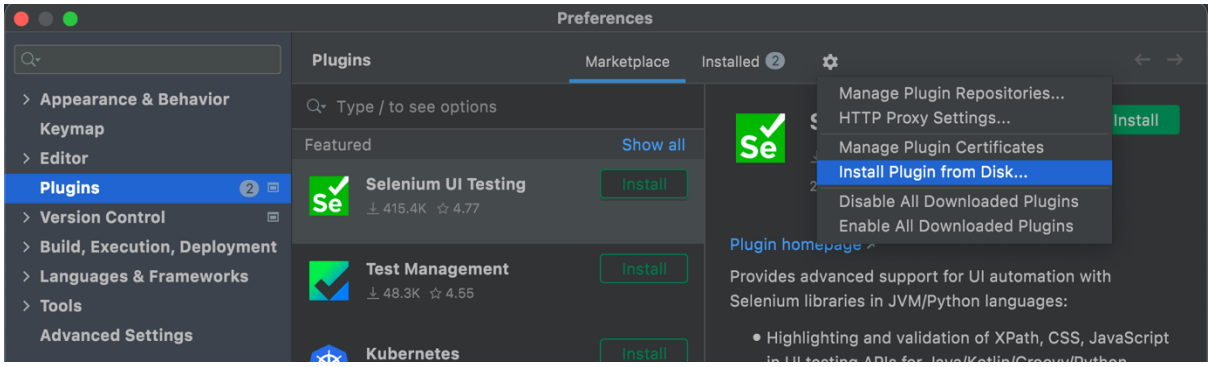
We are doing an empirical experiment that will help us evaluate and validate our project. To do it, we need your help!

This experiment is divided into three different tasks that will correspond to three different projects with different degrees of difficulty. Note that each task is described textually with the help of the respective UML diagrams (these diagrams are also attached to each project).

This experiment is estimated to take no more than **45 minutes**.

Before starting the programming tasks, read the guide related to the tool you will use and watch the sample video that is attached to it.

Please launch the IntelliJ IDE and install the provided tool to start the experiment. To install it, you just need to select your IDE's preferences. There you will find the “**Plugins**” menu, where you must select the option of “**Install Plugin from Disk**” (see the following image). Search the folder where you saved the attached files and select the .zip folder that is inside it. Then, IntelliJ will install the plugin. This description is demonstrated in the sample video.



Note that for each programming task, you can implement or refactor the code manually and undo a refactoring applied to the code when you believe it wasn't the best refactoring to be used on a specific situation.

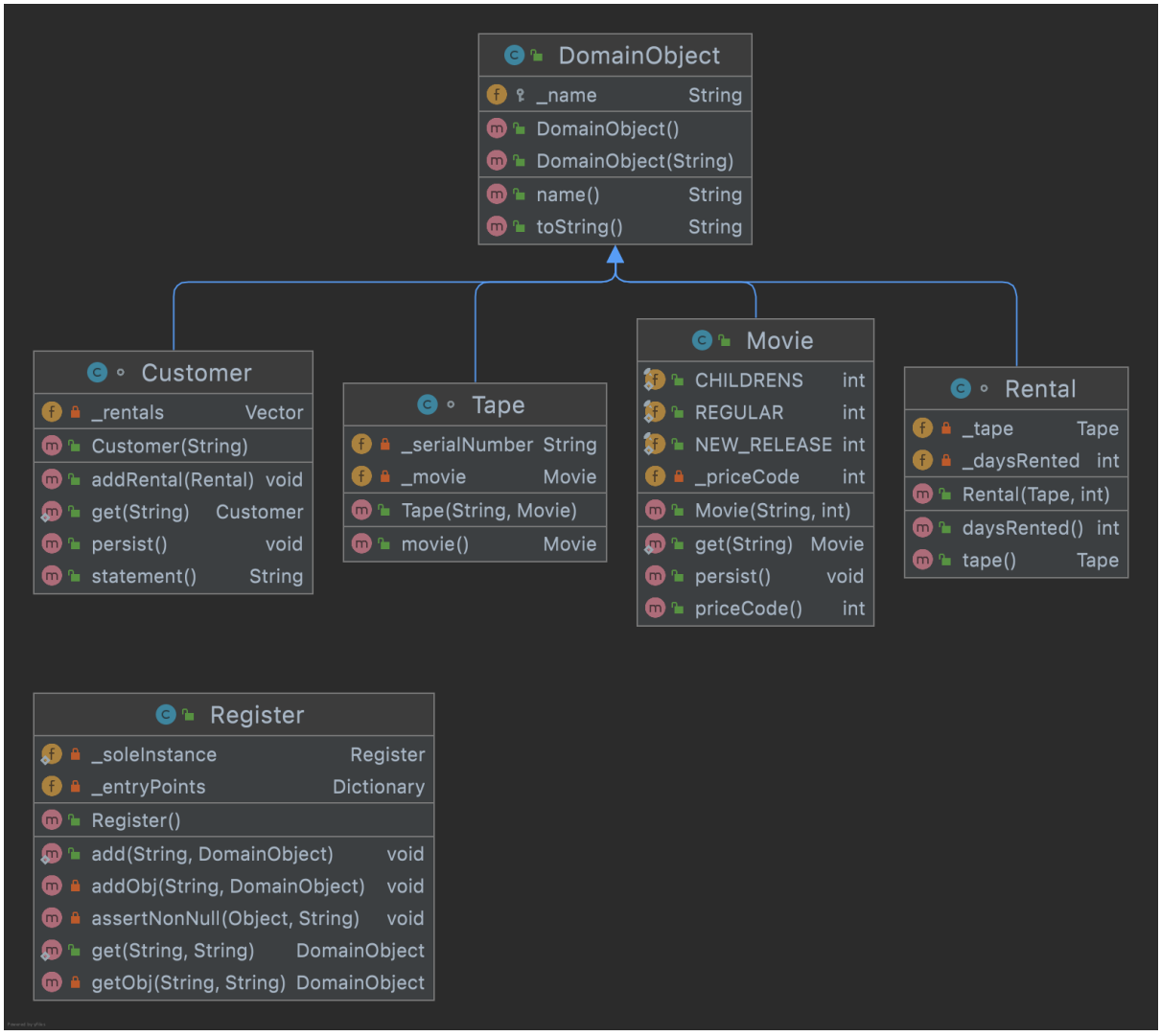
After doing all the programming tasks, you should return to the Goggle Form and answer the remaining questions.

Please read this guide and clarify any doubts you may have…

Group B

Programming Task 1

The first programming task is focused on a simple project that represents a movie rental system. The following image represents the UML diagram of that system.



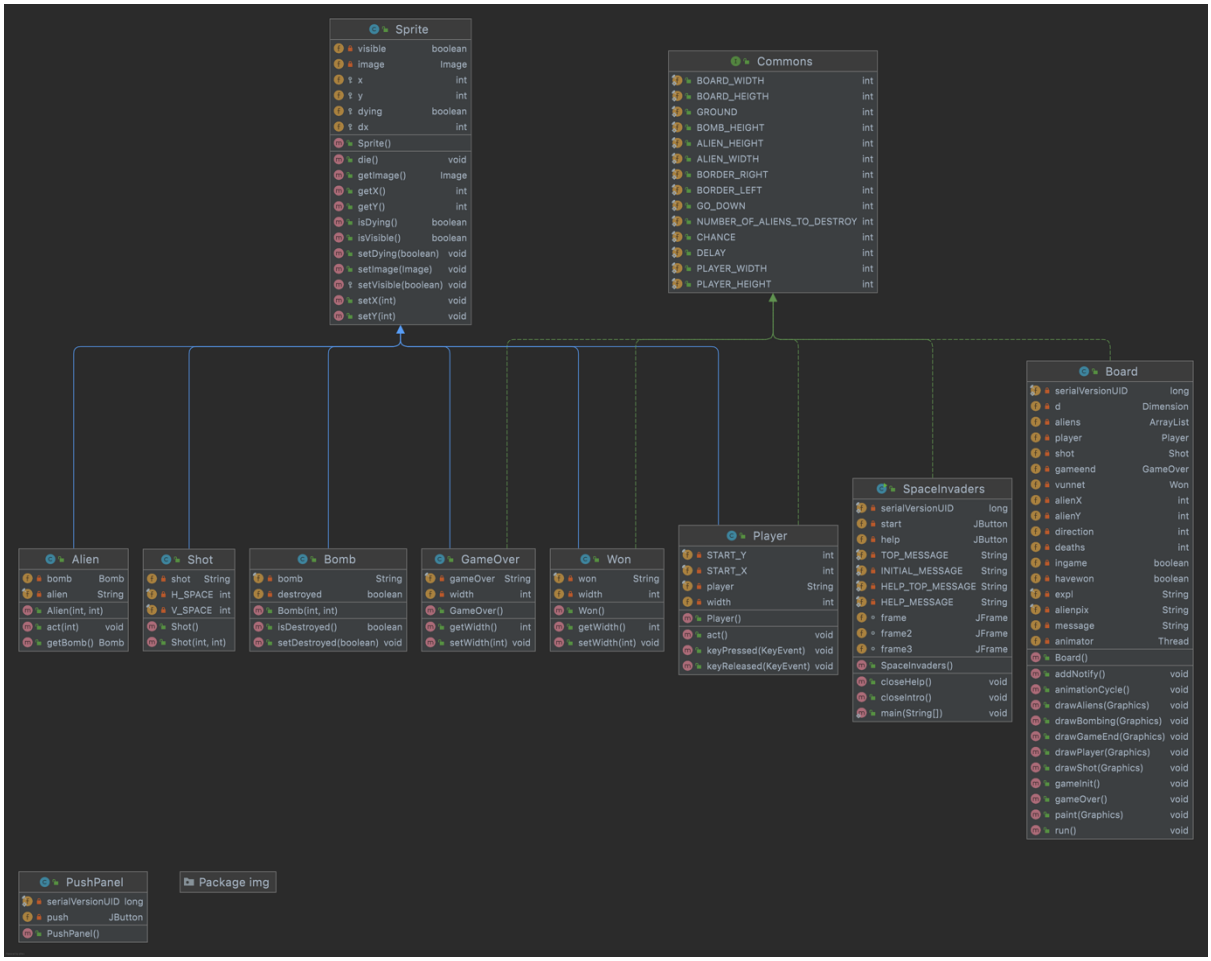
Programming Task: This programming task aims to manually identify, using the available tool, a possible code smell that can be corrected using the Extract Method refactoring.

So, you should follow the next steps:

17. Open the "**MovieRental**" project in your IntelliJ;
18. Analyze the project and try to answer the questions posed on the Google Form;
19. After you have responded to the questions, come back to this guide;
20. Now, start the complementary tool by clicking on the "**Start Experiment**" option and open the class where you think there is a method where an Extract Method refactoring can be applied;
21. Try to manually refactor that method using the Extract Method refactoring.
22. After applying a refactoring, you should click on the option "**Save Metrics**". Then, you can apply as many refactorings as you want. <u>But don't forget to save the metrics after each refactoring.</u>
23. The task is complete when you believe that the code as good quality.

Programming Task 2

The second programming task is focused on a medium project that implements the game “**Space Invaders”**. The following image represents the UML diagram of the game.



<u>Programming Task:</u> This programming task aims to identify, using the available tool, while programming, possible candidate code blocks to be refactored using the Extract Method refactoring.

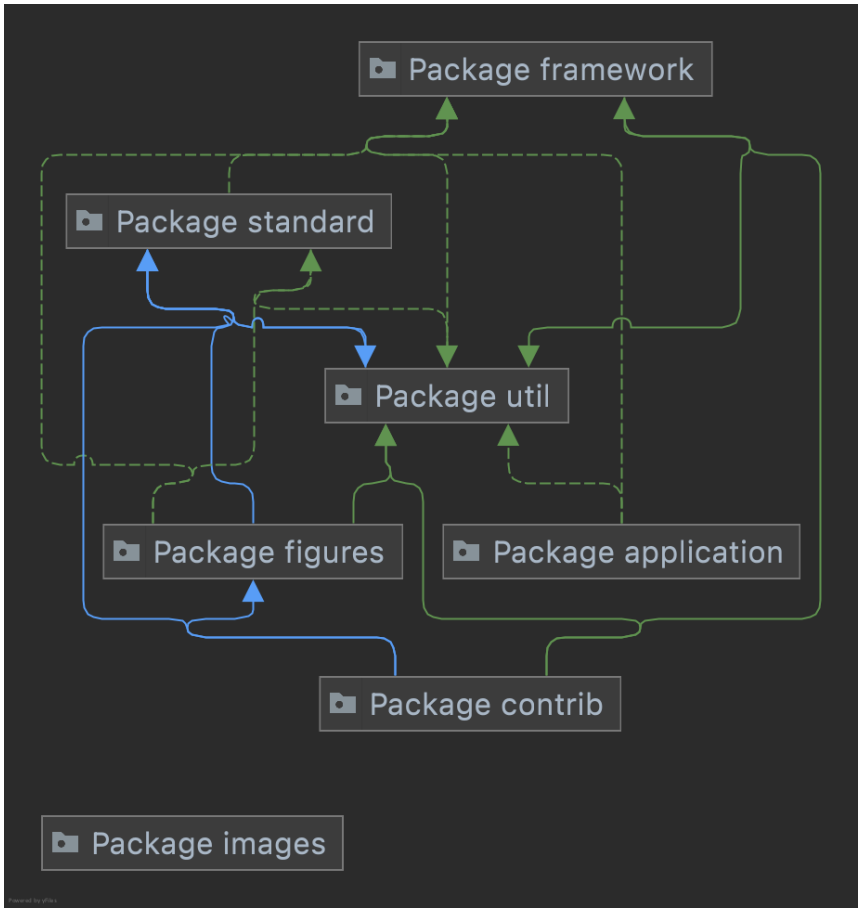
So, you should follow the next steps:

17. Open the “**SpaceInvaders**” project in your IntelliJ;
18. Analyze the project using the provided diagram;

19. Now, open the Board.java file and start the refactoring tool by clicking on the "**Start Analysis**" option;
20. Focus on the "**animationCycle**" method (class **Board**), which aims to implement the game's animations on the respective board. It will be the method you will have to implement;
21. Within the method itself, you find comments about the code you must implement;
22. Follow the steps described there;
24. After implementing the code, try to refactor it (if you believe the code needs to be refactored). Don't forget to save the metrics after each refactoring applied to the code.
23. The task is completed when you believe that the code has good quality.

Programming Task 3

The third and final programming task is focused on a large-scale project that implements the **JHotDraw** system, which is a two-dimensional graphics framework for structured drawing editors. Since the project has several classes and methods, its UML is not easy to understand when inserted into a file like this. In this way, we just create the simplified UML diagram of the system. Attached to this file are more complete UML diagrams representing the packages where the programming task should be applied.



Programming Task: This programming task aims to identify, using the available tool, all code blocks that must be refactored using the Extract Method refactoring.

So, you should follow the next steps:

13. Open the "**JHotDraw**" project in your IntelliJ;
14. Analyze the project using the diagram provided to you;
15. Now, open the file **DrawApplication.java** (package **Application**) and start the complementary tool by clicking on the "**Start Experiment**" option;
16. There, try to refactor the code implemented on the file (if you believe the code needs to be refactored). Don't forget to save the metrics after each refactoring applied to the code.
17. The task in that file is completed when you believe that the code has good quality;
18. Then, repeat these steps on **StandardDrawingView.java** (package **standard**).