SOEN 6441 - Advanced Programming Practices

Project - Warzone Risk Based game (Build #3)

Winter 2025

Refactoring Document

Submitted by **DABSV**

Arvind Nachiappan Lakshmanan - 40310757

Barath Sundararaj - 40324920

Devasenan Murugan - 40302170

Swathi Priya Pasumarthy- 40322468

Submitted to

Prof. M. Taleb

Potential Refactoring Targets

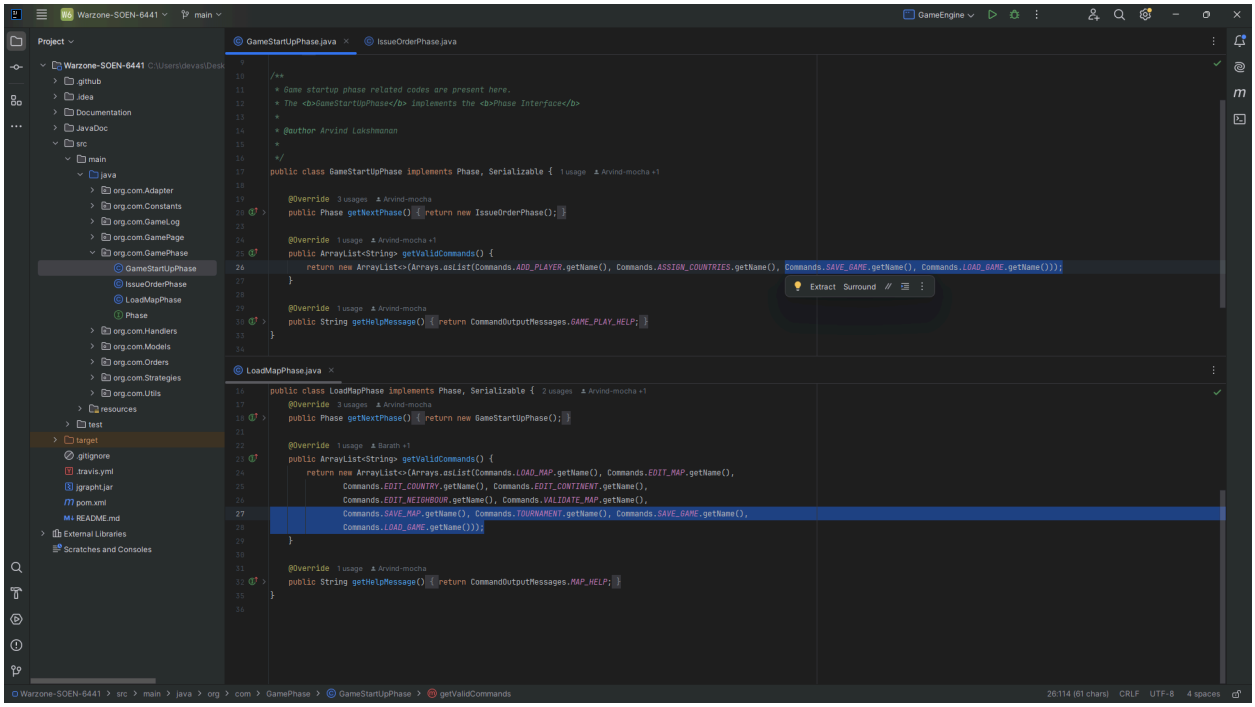
Given below is the list of potential refactoring targets:

1. **Updated all the Java classes to implement Serializable -** This was implemented to make the classes compatible for saving and loading game files.
2. **Encapsulated functions to handle different map types in the Adapter Module -** for the purpose of handling domination or conquest maps.
3. **Decomposed GameEngine into GameEngine and GameModeExecuter** - Either for a Single mode or Tournament mode the core logic and operations for executing the game is the same. So the GameModeExecuter has the logic to run a single game.
4. **Added new Model for the Tournament feature**
5. **Added new Module for implementing all the computer Strategies**
6. **Command validation for invalid option types and param length -** Validation of commands were properly handled to overcome the errors in case the player entered the wrong option type or parameter length.
7. **Command validation for invalid and redundant game phase commands**
8. **Updating the access modifiers of the methods -** This was discovered to ensure that methods have appropriate levels of visibility in order to maintain encapsulation and control access to specific functionalities.
9. **Updating Unintuitive variable names**
10. **Removing dead code and unused Import statements-** We identified this by seeing code and import statements that appear to have no effect on the program's behavior.
11. **Replacing if..else..if with switch..case -** This target was chosen to improve code readability and maintainability in situations where a switch statement would be preferable to a series of if-else statements
12. **Adding constants for the valid commands -** Constants were introduced to reduce the repetition of the same statements.

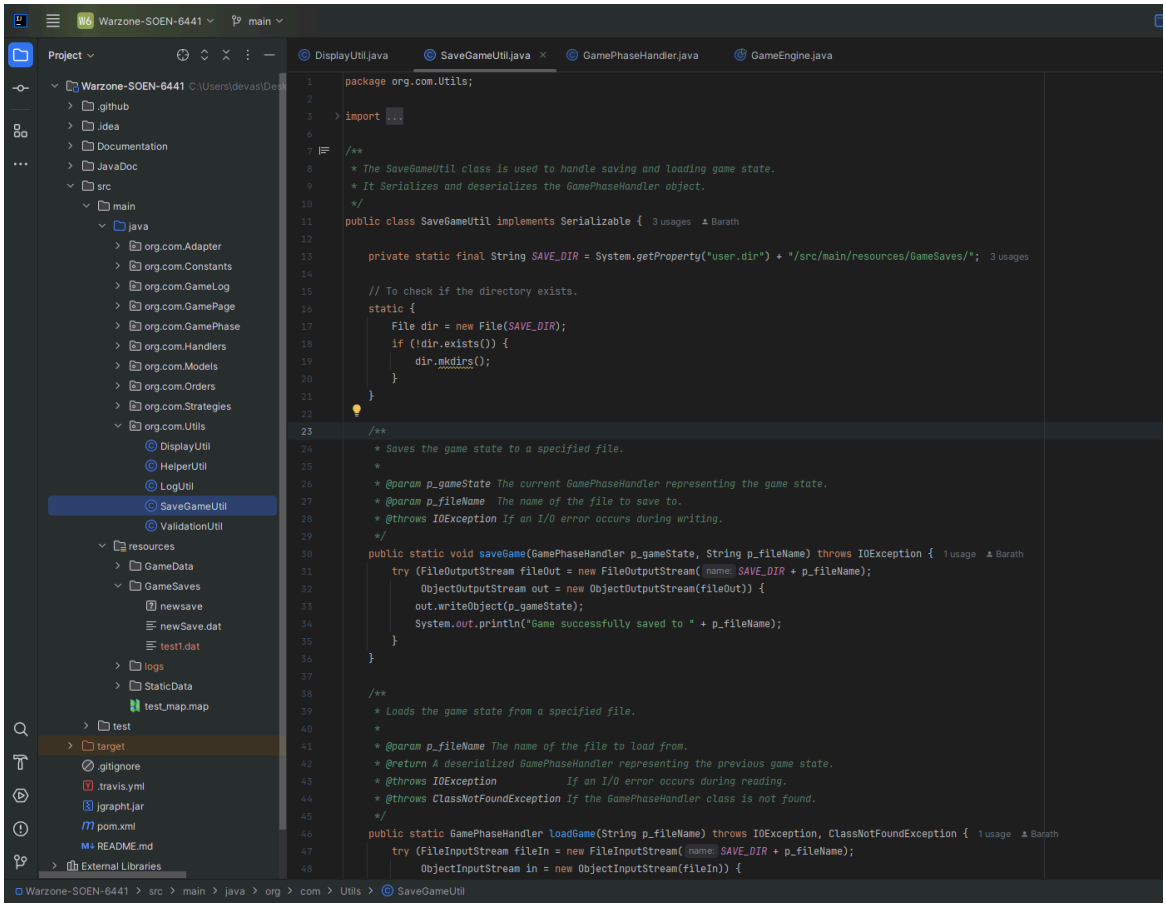
```
Hunk 2 : Lines 29-39
    public static final String COUNTRY_SET_TO_UNKNOWN_CONTINENT = "Country set with unknown Continent";
    public static final String PLAYER_NOT_EXISTS_REMOVAL = "Player %s does not exist to be removed";
    public static final String NO_PLAYER_EXISTS = "No player exists to be removed";
+   public static final String IMPROPER_OWNER_COUNTRY = "Country %s does not belong to the player %s";
+   public static final String ARMY_COUNT_EXCEEDS = "Deployable army count is less than the army count which you are trying to deploy";
+   public static final String NEUTRAL_COUNTRY_DEPLOYMENT = "Cannot deploy armies to a neutral country";
+   public static final String ARMY_COUNT_ZERO = "Army count of zero cannot be used for deployment";
+   public static final String INVALID_COUNTRY = "Country mentioned does not exists";
+   public static final String NO_ADVANCE_COMMAND = "Player %s as no advance commands to be executed";
+
}
```

13. Update allowed commands in the startUp phase to include savegame loadgame commands

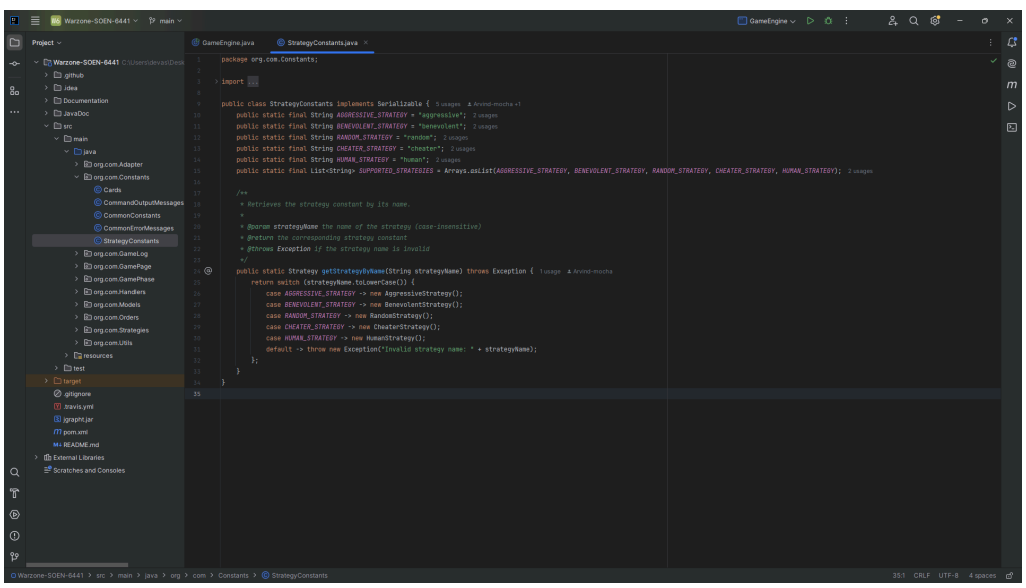
```
Hunk 1 : Lines 22-28                                                    Reverse hunk
...
    @Override
    public ArrayList<String> getValidCommands() {
-       return new ArrayList<>(Arrays.asList(Commands.ADD_PLAYER.getName(), Commands.ASSIGN_COUNTRIES.getName(), Commands.DEPLOY_ARMY_COMMAND.getName(), Commands.ADVANCE_ARMY_COMMAND.getName()));
+       return new ArrayList<>(Arrays.asList(Commands.ADD_PLAYER.getName(), Commands.ASSIGN_COUNTRIES.getName()));
    }
...
    @Override
```



14. Added savegame & loadgame features



15. Adding Common constants for Game Strategies

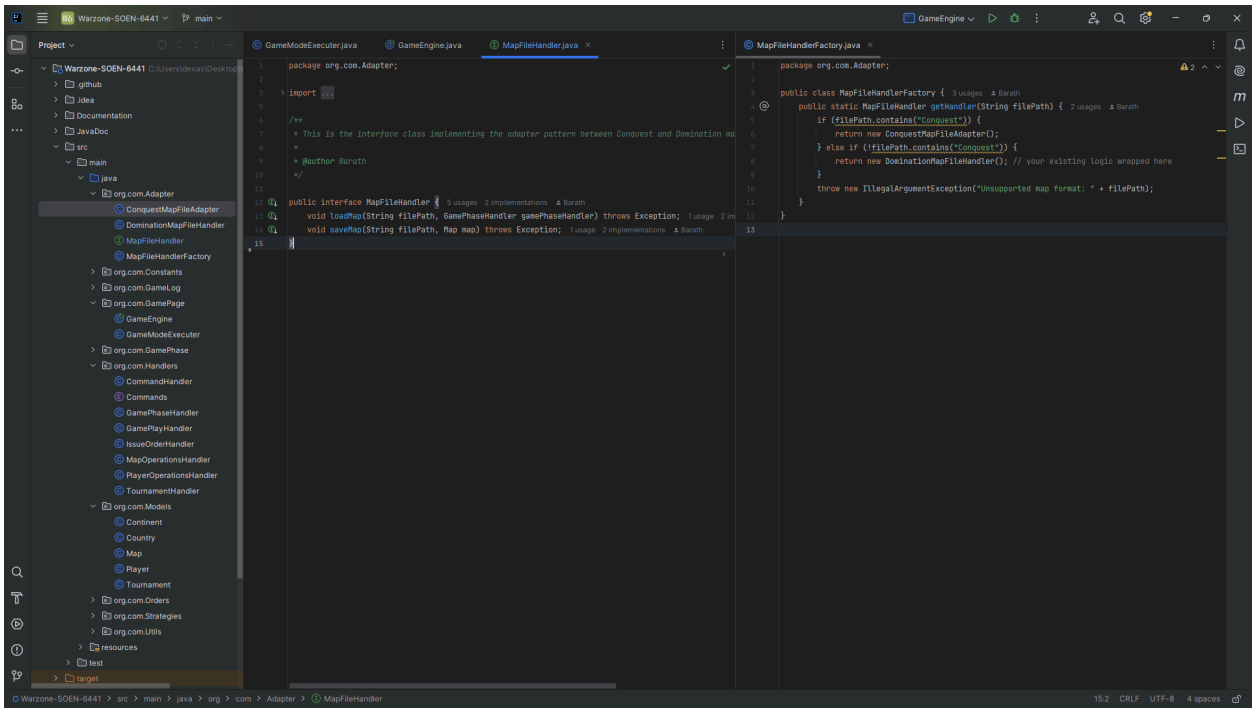


Actual Refactoring Targets

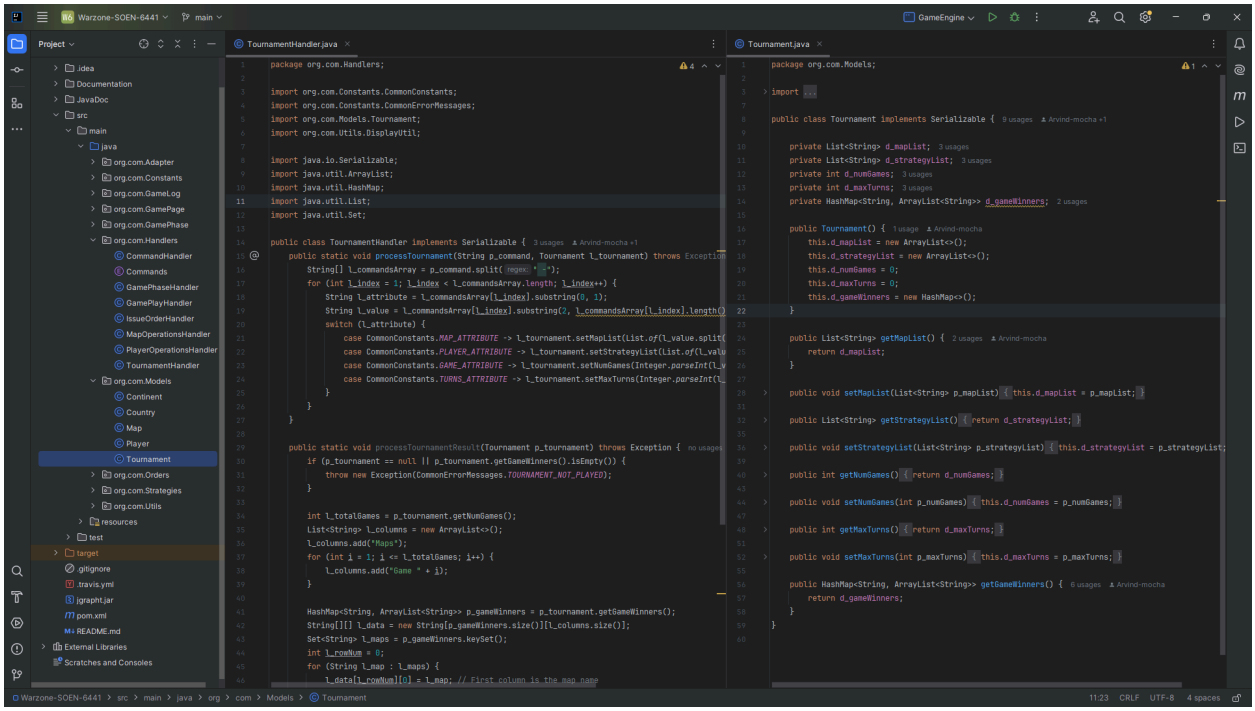
Below are the 5 refactoring targets chosen from the above mentioned list of all potential targets, based on the requirements established in Build 3.

1. **Updated all the Java classes to implement Serializable -** This was implemented to make the classes compatible for saving and loading game files.

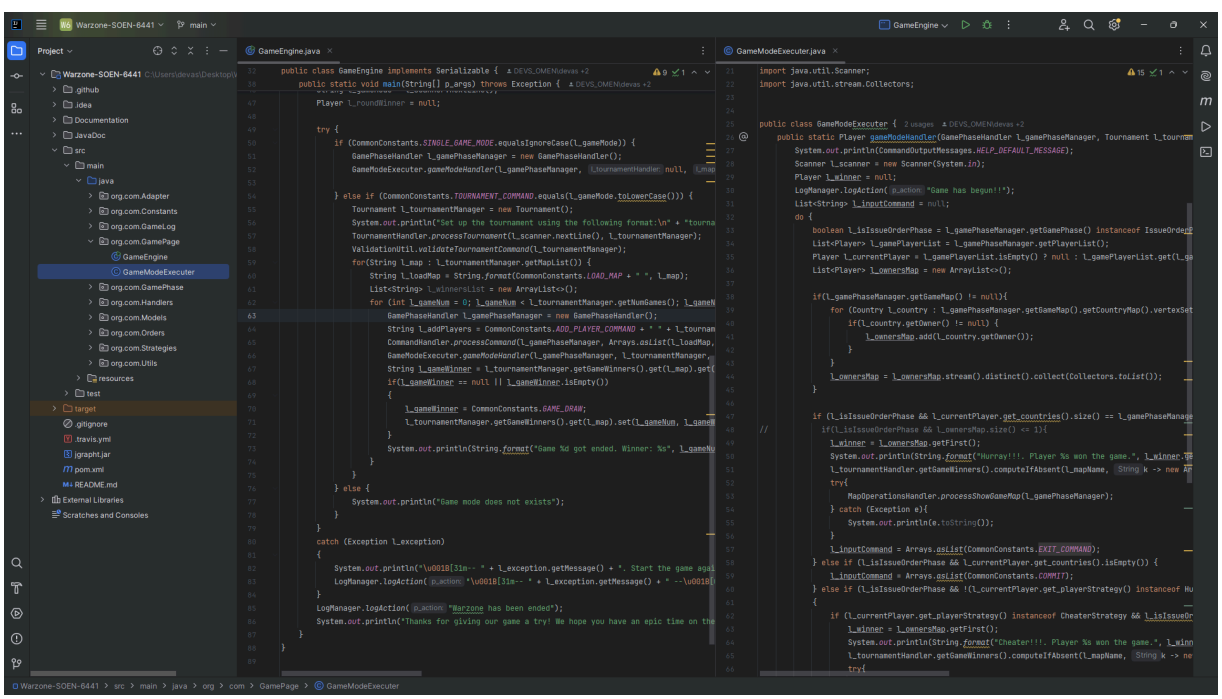
2. **Encapsulated functions to handle different map types in the Adapter Module**



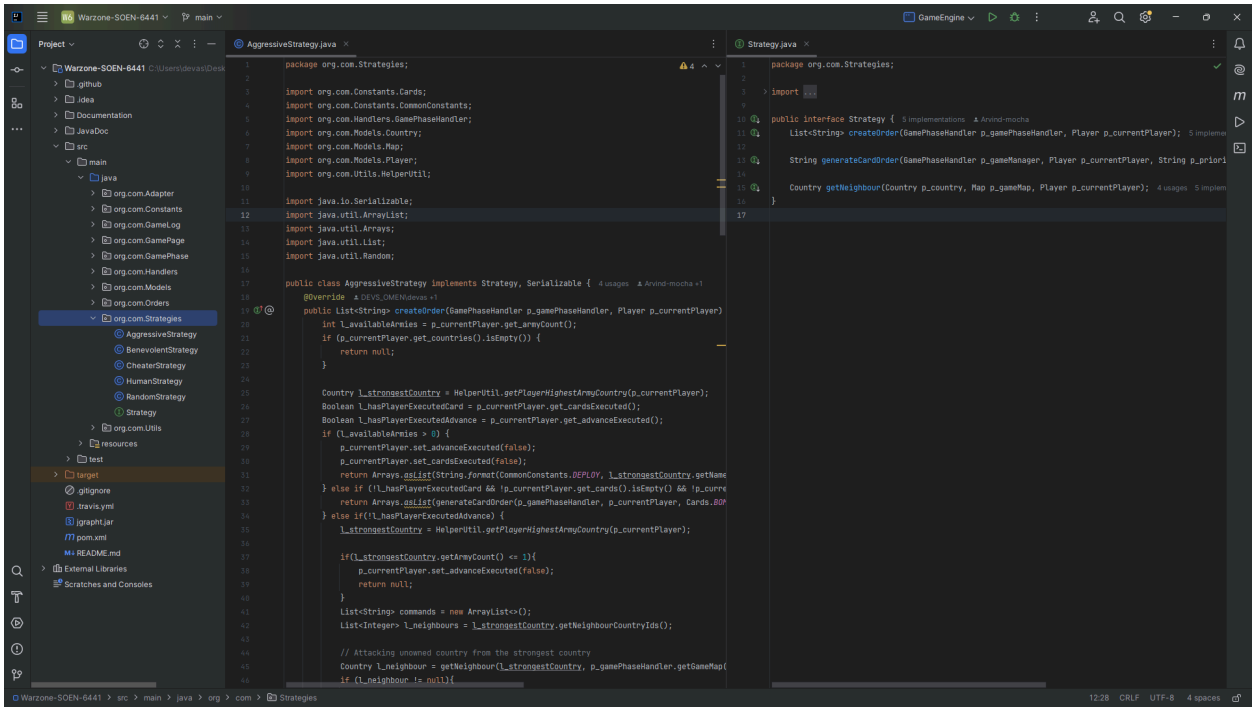
3. **Added new Model for the Tournament feature -**



4. **Decomposed GameEngine into GameEngine and GameModeExecuter -** Either for a Single mode or Tournament mode the core logic and operations for executing the game is the same. So the GameModeExecuter has the logic to run a single game.



5. **Added new Module for implementing all the computer Strategies**



Reasons for choosing the Actual Refactoring Targets

The following elements were chosen based on the aspects of code quality, maintainability and better functionality of the game in whole. This has helped in improving the internal structure of the program without breaking the external behaviour of the game observed in the initial build.

1. **Updated all the Java classes to implement Serializable -** In the second build, the classes defined were not serializable. To save or load game phase, the phase has to be stored as objects in .ser or .dat formats. Hence, the classes were implemented as serializable in this phase.

2. **Encapsulated functions to handle different map types in the Adapter Module** Initially the game was developed to handle only domination based maps. Based on

the requirements, we introduced an adapter pattern to handle conquest and domination maps. This helps us in reducing re-writing lines of code and introduces the choice of using different maps.

3. **Added a new Model for the Tournament feature -** This was done to introduce the tournament feature. Tournament feature uses different player types and automates the entire game except for “human” type players, where the commands have to be given manually.

4. **Decomposed GameEngine into GameEngine and GameModeExecuter -** Decomposed the GameEngine into 2 files to handle the type of game played. Based on the mode single or tournament, the GameModeExecuter is responsible for handling the mode of the games. This is done for the game to be run smoothly, efficiently and without any conflicts.

5. **Added new Module for implementing all the computer Strategies**
   A new module was introduced to implement all computer strategies because if these were added to PlayerOperationsHandler, it would have made the codebase complicated and unreadable. This would have also made the codebase extremely difficult to understand and hard to work on for future enhancements. Hence we came up with the idea to add a new module because in case if there are any new strategies to be added, this could be done easily and efficiently to the current module by just introducing a new java file.