REFACTORING DOCUMENT

Potential refactoring targets (Actual refactoring targets in bold):

1. **Update data type of Borders data member of GameMap class and add Player name data member to Country class.**
2. **Remove method overloading in showMap() method of GameMap class.**
3. **Rename GamePhase class and add and remove existing methods to comply to state pattern architecture.**
4. **Phase related refactoring**
5. **GameInformation converted to singleton class.**
6. Storing country object in Continent objects' List of adjacent countries. – Instead of iterating through country object list to get the desired country object.
7. Storing continent object in Country object's continent it belongs to. - Instead of iterating through continent object list to get the desired continent object.
8. Store country objects (LinkedHashMap<Integer, List<Country>>). - Instead of iterating through country object list to get the desired country object.
9. Creating a separate class that has error message constants in constants package. – Easier to understand the type of message used while printing messages on console.
10. Remove for loops for searching for a country given country name in multiple places by defining a method in Country Class. – To increase the readability of code.

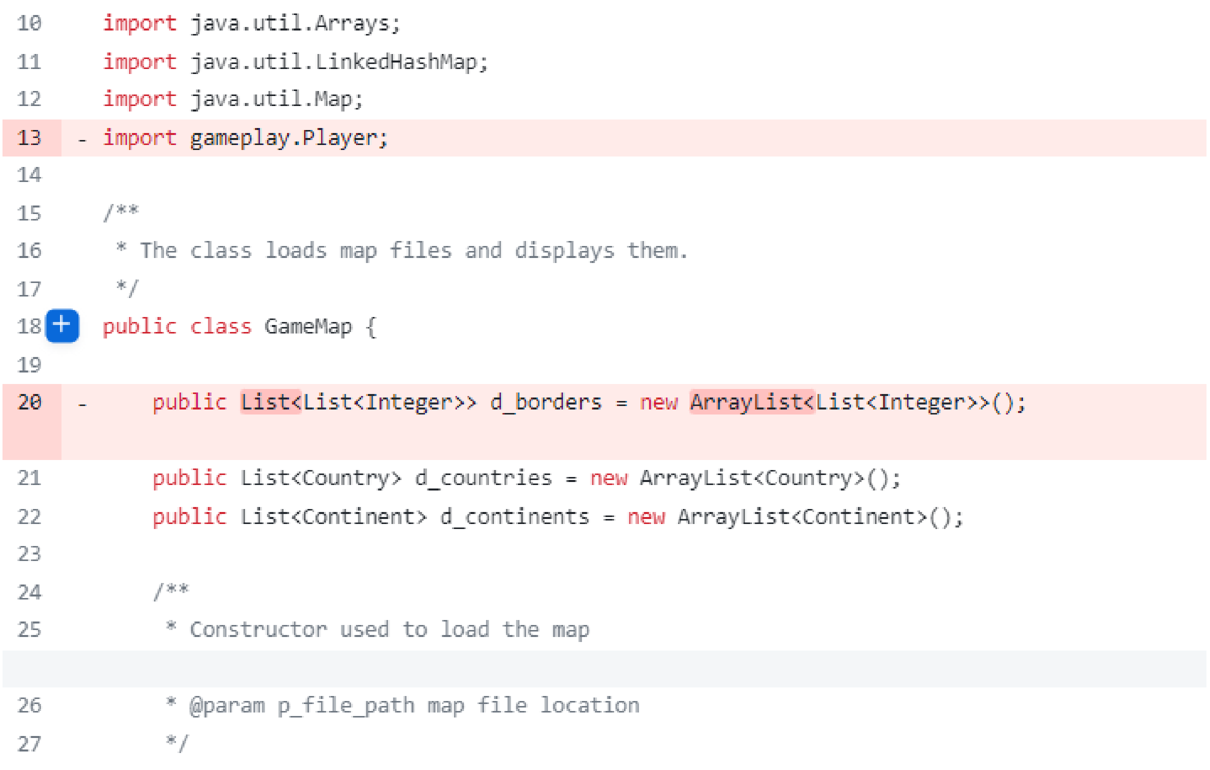
11. Define a separate class to perform map command validation instead of repeating the same code in multiple edit methods. – To increase the readability of code and to reduce code duplication.
12. Grouping multiple declarations of same type in different lines to one line. – To increase the readability of code and to reduce the length of code.
13. Remove d_completed_operations list used to store completed operations and track it differently in GameStartupPhase class. – to improve code readability.
14. Move option validation to validateAndExecuteCommands method instead of doing in individual methods. – to make the command parser more flexible for future builds.
15. Store players list in a list or array instead of HashMap. – to ease the access of player objects.
16. Move command parameter and other option validation to validateAndExecuteCommands – to reduce code duplication.
17. Use separate classes for Country and Continent objects instead of inner class. – As country and continent objects can be created without having to use GameMap object.
18. Remove lines that cannot throw Exceptions from try and catch blocks. – Only lines that may throw exceptions should be present in try block

Actual refactoring targets:

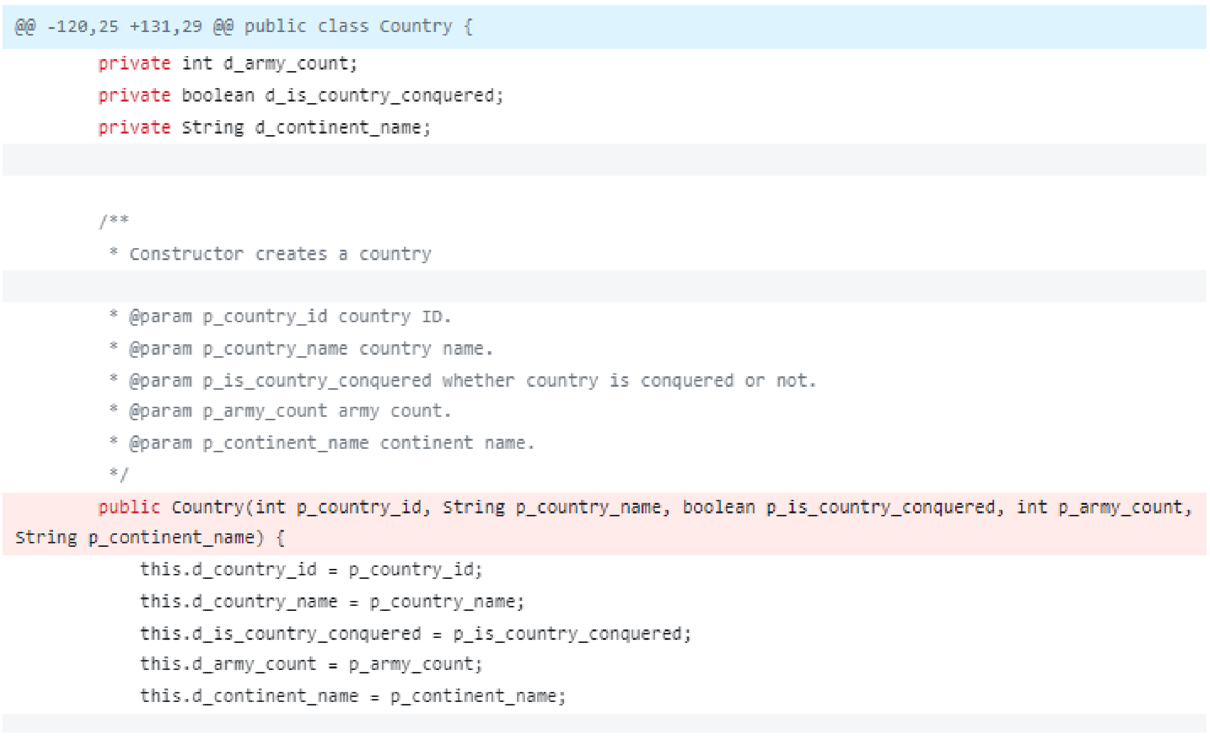
1. **Datatype changes: Updated data type of Borders data member of GameMap class and added Player name data member to Country class.**
   Test cases: testLoadValidMap, addContinentTest, removeContinentTest, addCountryTest, removeCountryTest, addBorderTest, removeBorderTest
   Additional test cases needed: None
   Why: **Borders data member:** Each time border list had to be read or altered, nested for loop was used to find the correct location to insert or delete a border relation for a country. By converting the data structure of the borders list from List<List<Integer>> to LinkedHashMap<Integer, List<Integer>> where the key is country ID and the value is the list of country IDs, nested for-loops were eliminated in multiple methods.

   Player name data member: Even though player objects store the list of conquered countries (List<Country>), in many methods, especially while executing orders, checks had to be done to determine whether a country was owned by a player. Therefore, player name attribute was added to country object to avoid iterating through all player's conquered list to find the particular country.

Before:

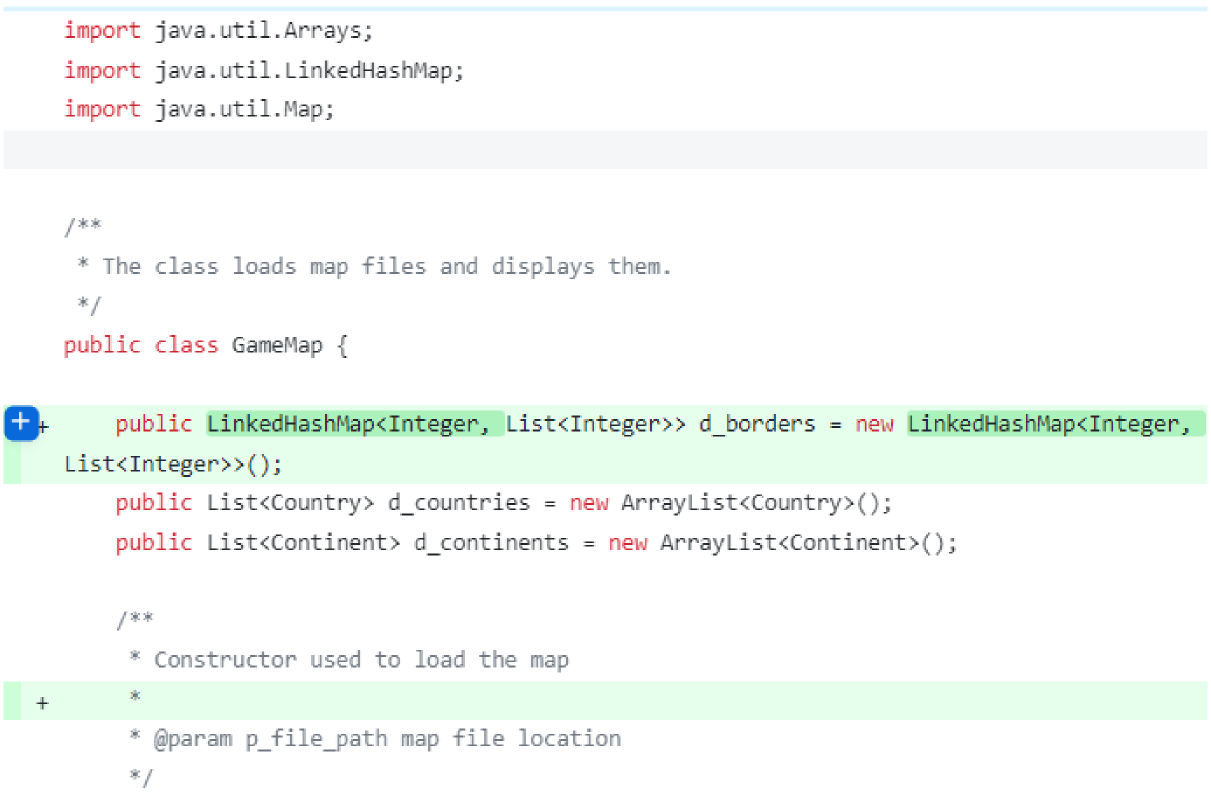


```
    import java.util.Arrays;
    import java.util.LinkedHashMap;
    import java.util.Map;
-   import gameplay.Player;

    /**
     * The class loads map files and displays them.
     */
    public class GameMap {

-       public List<List<Integer>> d_borders = new ArrayList<List<Integer>>();

        public List<Country> d_countries = new ArrayList<Country>();
        public List<Continent> d_continents = new ArrayList<Continent>();

        /**
         * Constructor used to load the map

         * @param p_file_path map file location
         */
```

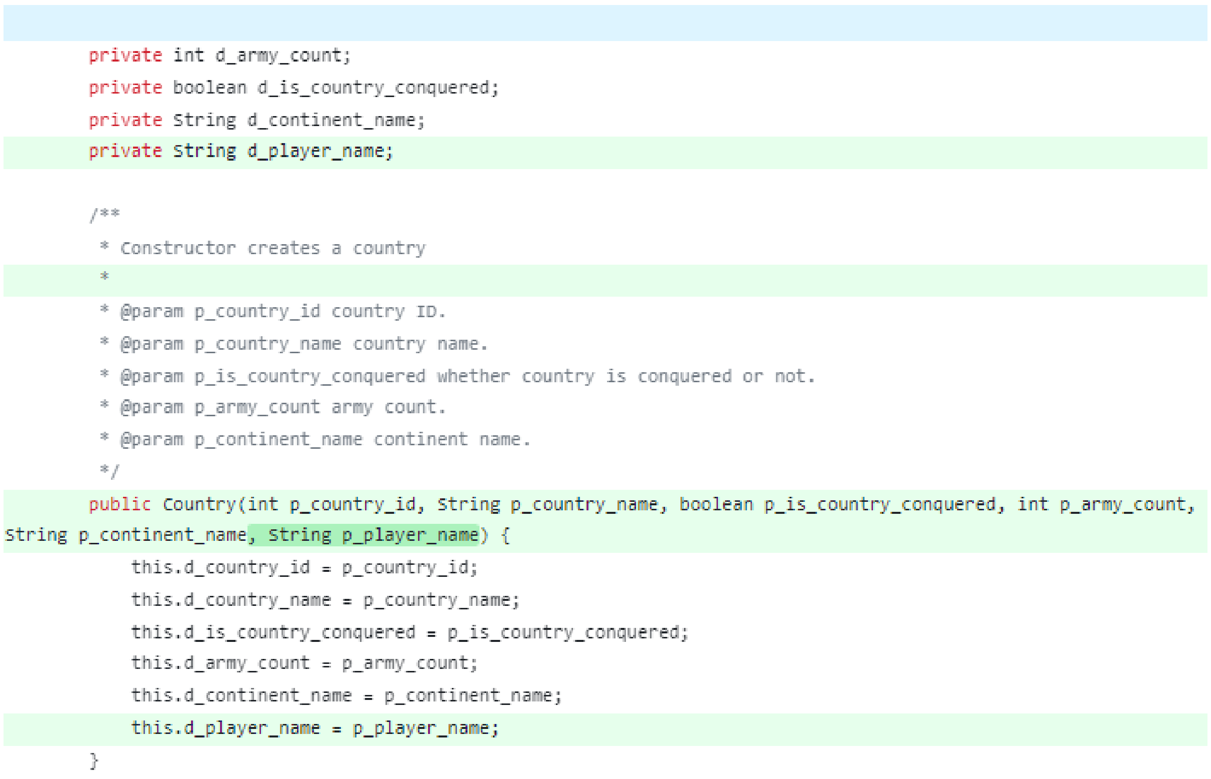


```
@@ -120,25 +131,29 @@ public class Country {
        private int d_army_count;
        private boolean d_is_country_conquered;
        private String d_continent_name;


        /**
         * Constructor creates a country

         * @param p_country_id country ID.
         * @param p_country_name country name.
         * @param p_is_country_conquered whether country is conquered or not.
         * @param p_army_count army count.
         * @param p_continent_name continent name.
         */
        public Country(int p_country_id, String p_country_name, boolean p_is_country_conquered, int p_army_count,
String p_continent_name) {
            this.d_country_id = p_country_id;
            this.d_country_name = p_country_name;
            this.d_is_country_conquered = p_is_country_conquered;
            this.d_army_count = p_army_count;
            this.d_continent_name = p_continent_name;
```

After:



```
import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.Map;


/**
 * The class loads map files and displays them.
 */
public class GameMap {

+       public LinkedHashMap<Integer, List<Integer>> d_borders = new LinkedHashMap<Integer,
List<Integer>>();
        public List<Country> d_countries = new ArrayList<Country>();
        public List<Continent> d_continents = new ArrayList<Continent>();

        /**
         * Constructor used to load the map
+        *
         * @param p_file_path map file location
         */
```



```
        private int d_army_count;
        private boolean d_is_country_conquered;
        private String d_continent_name;
        private String d_player_name;

        /**
         * Constructor creates a country
         *
         * @param p_country_id country ID.
         * @param p_country_name country name.
         * @param p_is_country_conquered whether country is conquered or not.
         * @param p_army_count army count.
         * @param p_continent_name continent name.
         */
        public Country(int p_country_id, String p_country_name, boolean p_is_country_conquered, int p_army_count,
String p_continent_name, String p_player_name) {
            this.d_country_id = p_country_id;
            this.d_country_name = p_country_name;
            this.d_is_country_conquered = p_is_country_conquered;
            this.d_army_count = p_army_count;
            this.d_continent_name = p_continent_name;
            this.d_player_name = p_player_name;
        }
```

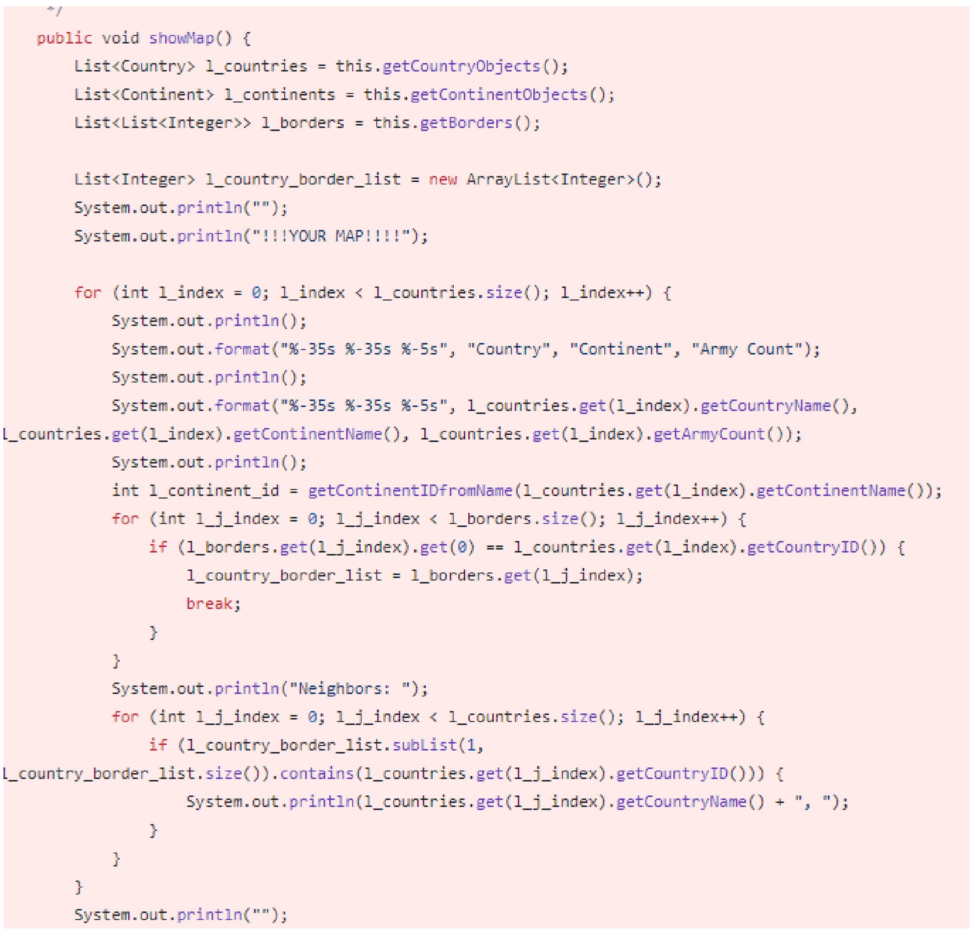
2. **Removed method overloading in showMap() method of GameMap class**

   Test cases: showMapTestWithoutPlayers

   Additional test cases needed: showMapTestWithPlayers

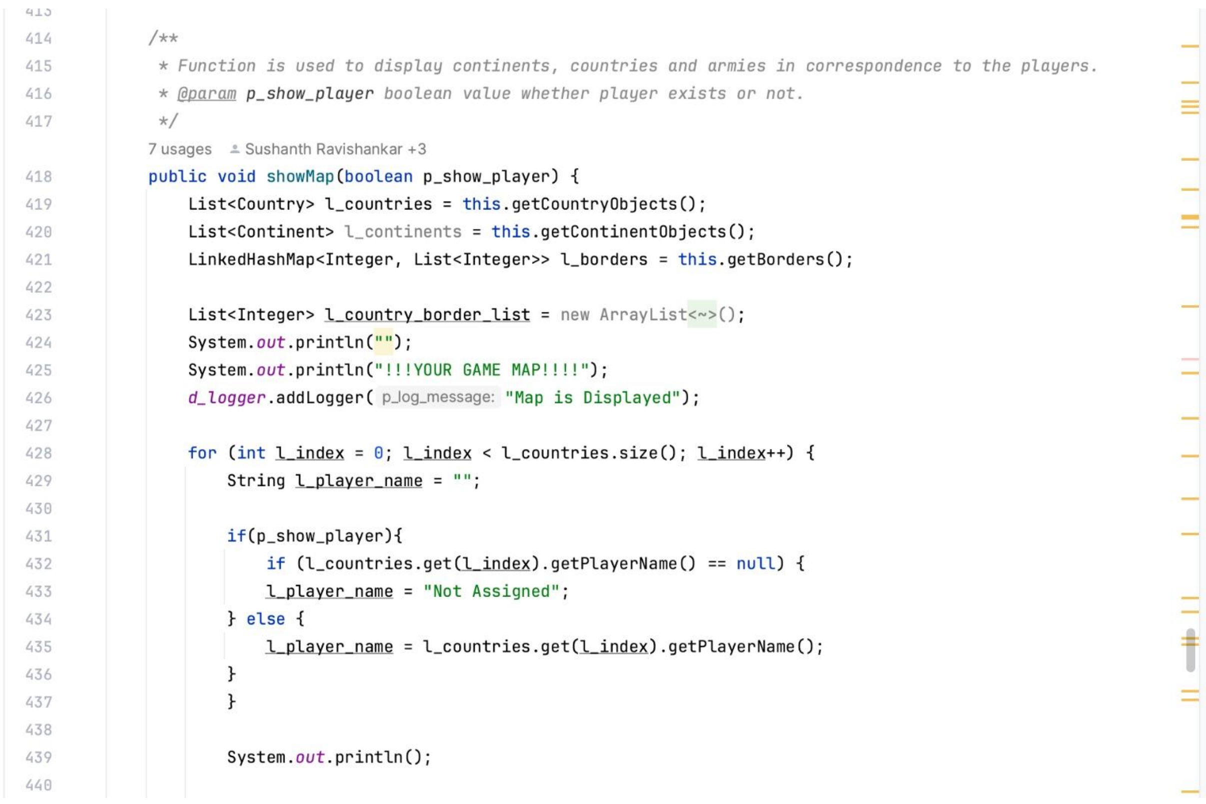
   Why: showMap() was overloaded in build1, one with no parameters and another with player object as a parameter as showMap() can be triggered during game play and map editing phase. The overloading was removed and the refactored showMap() method takes a Boolean parameter that is set to “true” if player information should be displayed and “false” if player information should not be displayed. This refactoring was possible as there is player name attribute in country objects.

   Before:



```
     */
    public void showMap() {
        List<Country> l_countries = this.getCountryObjects();
        List<Continent> l_continents = this.getContinentObjects();
        List<List<Integer>> l_borders = this.getBorders();

        List<Integer> l_country_border_list = new ArrayList<Integer>();
        System.out.println("");
        System.out.println("!!!YOUR MAP!!!!");

        for (int l_index = 0; l_index < l_countries.size(); l_index++) {
            System.out.println();
            System.out.format("%-35s %-35s %-5s", "Country", "Continent", "Army Count");
            System.out.println();
            System.out.format("%-35s %-35s %-5s", l_countries.get(l_index).getCountryName(),
l_countries.get(l_index).getContinentName(), l_countries.get(l_index).getArmyCount());
            System.out.println();
            int l_continent_id = getContinentIDfromName(l_countries.get(l_index).getContinentName());
            for (int l_j_index = 0; l_j_index < l_borders.size(); l_j_index++) {
                if (l_borders.get(l_j_index).get(0) == l_countries.get(l_index).getCountryID()) {
                    l_country_border_list = l_borders.get(l_j_index);
                    break;
                }
            }
            System.out.println("Neighbors: ");
            for (int l_j_index = 0; l_j_index < l_countries.size(); l_j_index++) {
                if (l_country_border_list.subList(1,
l_country_border_list.size()).contains(l_countries.get(l_j_index).getCountryID())) {
                    System.out.println(l_countries.get(l_j_index).getCountryName() + ", ");
                }
            }
        }
        System.out.println("");
```

After:



```
    /**
     * Function is used to display continents, countries and armies in correspondence to the players.
     * @param p_show_player boolean value whether player exists or not.
     */
    7 usages   Sushanth Ravishankar +3
    public void showMap(boolean p_show_player) {
        List<Country> l_countries = this.getCountryObjects();
        List<Continent> l_continents = this.getContinentObjects();
        LinkedHashMap<Integer, List<Integer>> l_borders = this.getBorders();

        List<Integer> l_country_border_list = new ArrayList<~>();
        System.out.println("");
        System.out.println("!!!YOUR GAME MAP!!!!");
        d_logger.addLogger( p_log_message: "Map is Displayed");

        for (int l_index = 0; l_index < l_countries.size(); l_index++) {
            String l_player_name = "";

            if(p_show_player){
                if (l_countries.get(l_index).getPlayerName() == null) {
                l_player_name = "Not Assigned";
            } else {
                l_player_name = l_countries.get(l_index).getPlayerName();
            }
            }

            System.out.println();

```

3. **Renaming GamePhase class and added and removed methods**

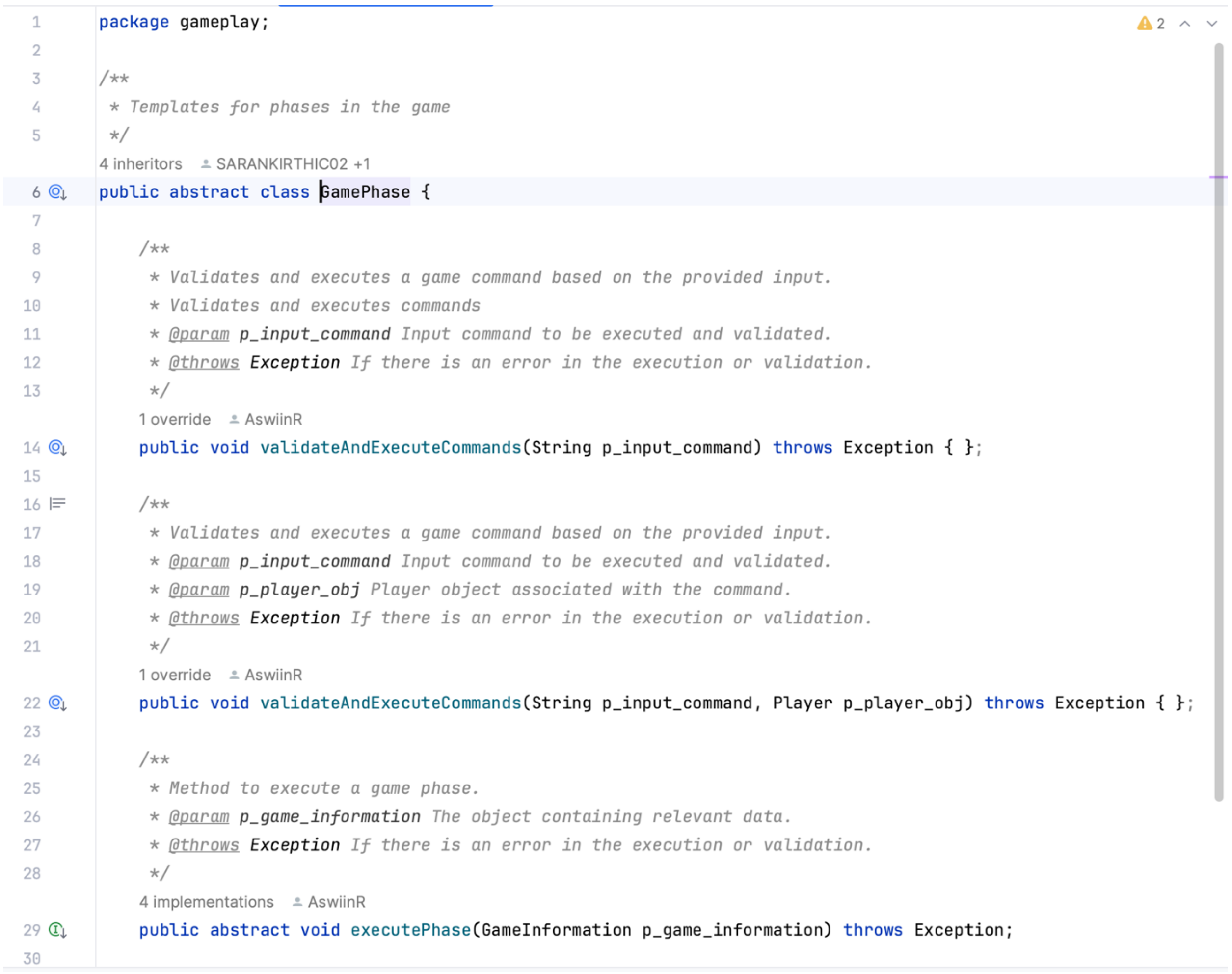
   Test cases: testAddorRemovePlayer, testAddorRemovePlayerInvalid

   Additional test cases needed: gameStartUpPhaseTest

   Why: GamePhase abstract class was renamed to Phase and a new abstract method nextPhase() was added.

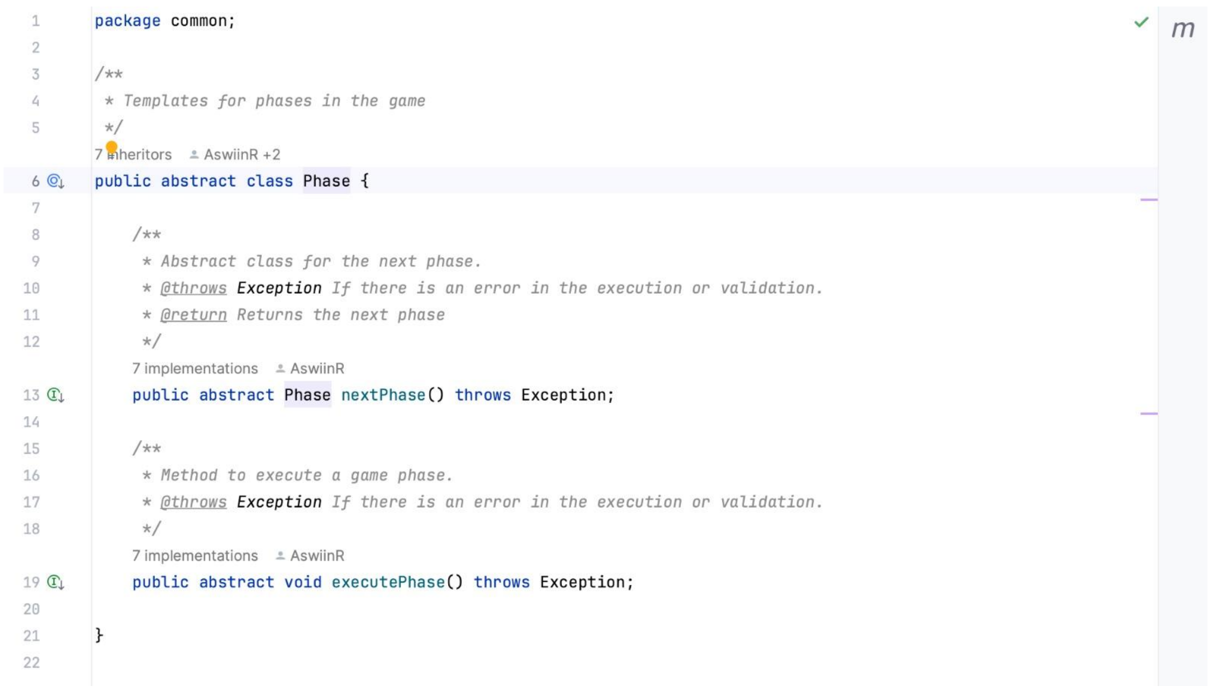
   validateAndExecuteCommand() overloaded methods were moved to the IssueorderPhase and gameStartupPhase classes that extend Phase class. This was done to comply with the State pattern architecture.

Before:



```
package gameplay;

/**
 * Templates for phases in the game
 */
4 inheritors   SARANKIRTHIC02 +1
public abstract class GamePhase {

    /**
     * Validates and executes a game command based on the provided input.
     * Validates and executes commands
     * @param p_input_command Input command to be executed and validated.
     * @throws Exception If there is an error in the execution or validation.
     */
    1 override   AswiinR
    public void validateAndExecuteCommands(String p_input_command) throws Exception { };

    /**
     * Validates and executes a game command based on the provided input.
     * @param p_input_command Input command to be executed and validated.
     * @param p_player_obj Player object associated with the command.
     * @throws Exception If there is an error in the execution or validation.
     */
    1 override   AswiinR
    public void validateAndExecuteCommands(String p_input_command, Player p_player_obj) throws Exception { };

    /**
     * Method to execute a game phase.
     * @param p_game_information The object containing relevant data.
     * @throws Exception If there is an error in the execution or validation.
     */
    4 implementations   AswiinR
    public abstract void executePhase(GameInformation p_game_information) throws Exception;

```

After:



```
package common;

/**
 * Templates for phases in the game
 */
7 inheritors   AswiinR +2
public abstract class Phase {

    /**
     * Abstract class for the next phase.
     * @throws Exception If there is an error in the execution or validation.
     * @return Returns the next phase
     */
    7 implementations   AswiinR
    public abstract Phase nextPhase() throws Exception;

    /**
     * Method to execute a game phase.
     * @throws Exception If there is an error in the execution or validation.
     */
    7 implementations   AswiinR
    public abstract void executePhase() throws Exception;

}

```

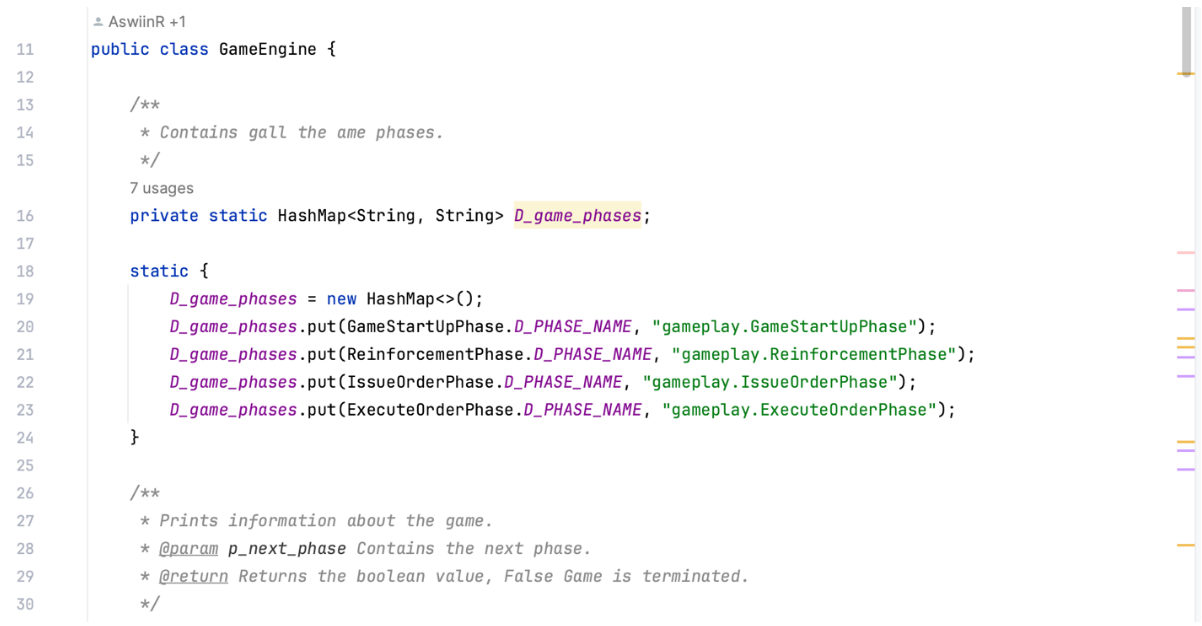
4. **Phase related refactoring**

Test cases: EditMapPhaseTest, GameMapTest

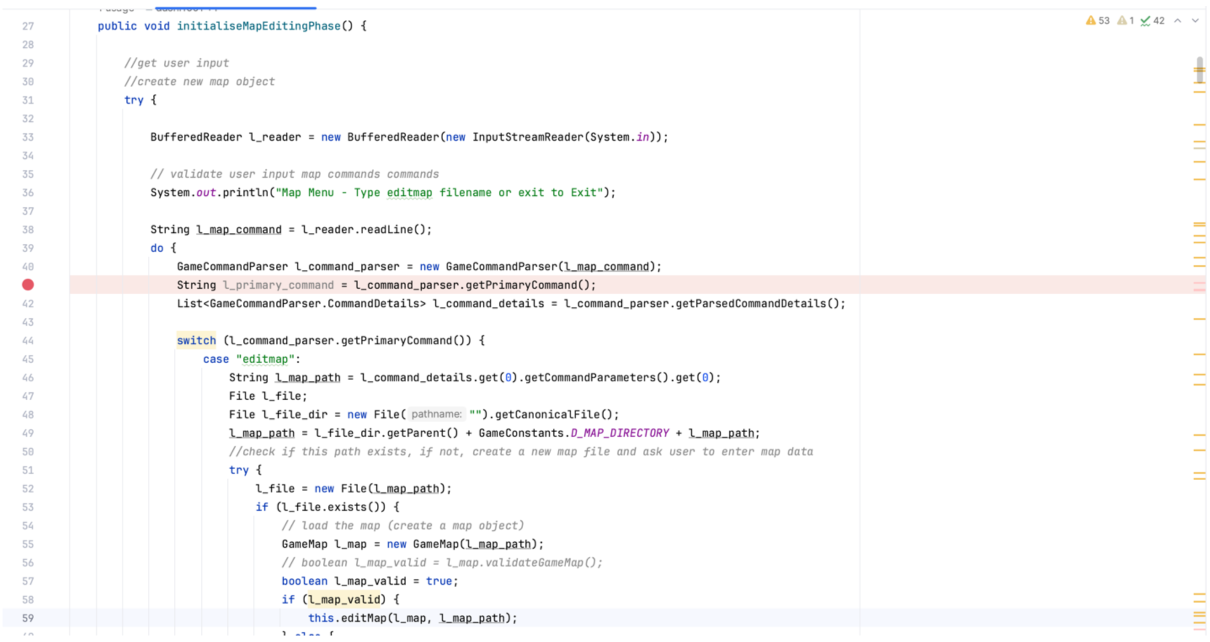
Additional test cases needed: gameStartUpPhaseTest

Why: To comply with State pattern architecture, the two classes were added to mapparser package, namely, LoadMapPhase and EditMapPhase, both the classes extend Phase class. One more class namely, EndGamePhase that extends Phase class was added to the gameplay package. All classes extending Phase class overrides nextPhase() and executePhase() methods.

Before:

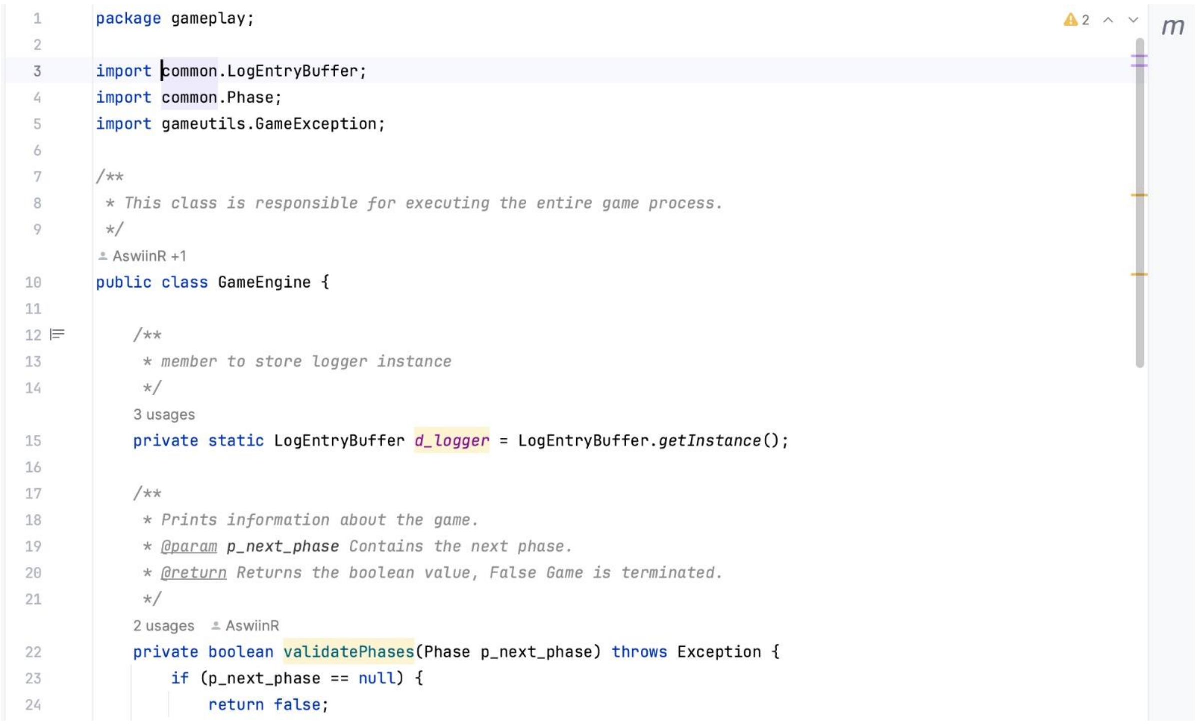


```
AswiinR +1
public class GameEngine {

    /**
     * Contains gall the ame phases.
     */
    7 usages
    private static HashMap<String, String> D_game_phases;

    static {
        D_game_phases = new HashMap<>();
        D_game_phases.put(GameStartUpPhase.D_PHASE_NAME, "gameplay.GameStartUpPhase");
        D_game_phases.put(ReinforcementPhase.D_PHASE_NAME, "gameplay.ReinforcementPhase");
        D_game_phases.put(IssueOrderPhase.D_PHASE_NAME, "gameplay.IssueOrderPhase");
        D_game_phases.put(ExecuteOrderPhase.D_PHASE_NAME, "gameplay.ExecuteOrderPhase");
    }

    /**
     * Prints information about the game.
     * @param p_next_phase Contains the next phase.
     * @return Returns the boolean value, False Game is terminated.
     */
```

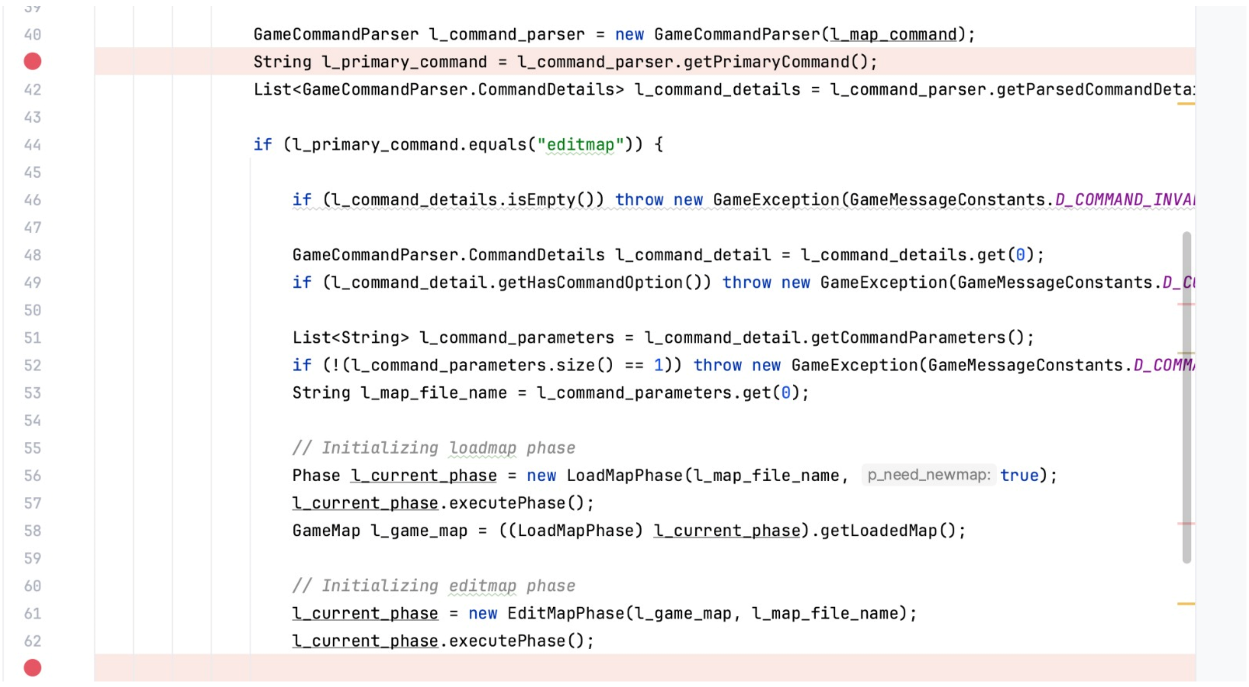


```java
public void initialiseMapEditingPhase() {

    //get user input
    //create new map object
    try {

        BufferedReader l_reader = new BufferedReader(new InputStreamReader(System.in));

        // validate user input map commands commands
        System.out.println("Map Menu - Type editmap filename or exit to Exit");

        String l_map_command = l_reader.readLine();
        do {
            GameCommandParser l_command_parser = new GameCommandParser(l_map_command);
            String l_primary_command = l_command_parser.getPrimaryCommand();
            List<GameCommandParser.CommandDetails> l_command_details = l_command_parser.getParsedCommandDetails();

            switch (l_command_parser.getPrimaryCommand()) {
                case "editmap":
                    String l_map_path = l_command_details.get(0).getCommandParameters().get(0);
                    File l_file;
                    File l_file_dir = new File( pathname: "").getCanonicalFile();
                    l_map_path = l_file_dir.getParent() + GameConstants.D_MAP_DIRECTORY + l_map_path;
                    //check if this path exists, if not, create a new map file and ask user to enter map data
                    try {
                        l_file = new File(l_map_path);
                        if (l_file.exists()) {
                            // load the map (create a map object)
                            GameMap l_map = new GameMap(l_map_path);
                            // boolean l_map_valid = l_map.validateGameMap();
                            boolean l_map_valid = true;
                            if (l_map_valid) {
                                this.editMap(l_map, l_map_path);
```

After:



```java
package gameplay;

import common.LogEntryBuffer;
import common.Phase;
import gameutils.GameException;

/**
 * This class is responsible for executing the entire game process.
 */
AswiinR +1
public class GameEngine {

    /**
     * member to store logger instance
     */
    3 usages
    private static LogEntryBuffer d_logger = LogEntryBuffer.getInstance();

    /**
     * Prints information about the game.
     * @param p_next_phase Contains the next phase.
     * @return Returns the boolean value, False Game is terminated.
     */
    2 usages   AswiinR
    private boolean validatePhases(Phase p_next_phase) throws Exception {
        if (p_next_phase == null) {
            return false;
```



```java
        GameCommandParser l_command_parser = new GameCommandParser(l_map_command);
        String l_primary_command = l_command_parser.getPrimaryCommand();
        List<GameCommandParser.CommandDetails> l_command_details = l_command_parser.getParsedCommandDeta

        if (l_primary_command.equals("editmap")) {

            if (l_command_details.isEmpty()) throw new GameException(GameMessageConstants.D_COMMAND_INVA

            GameCommandParser.CommandDetails l_command_detail = l_command_details.get(0);
            if (l_command_detail.getHasCommandOption()) throw new GameException(GameMessageConstants.D_C

            List<String> l_command_parameters = l_command_detail.getCommandParameters();
            if (!(l_command_parameters.size() == 1)) throw new GameException(GameMessageConstants.D_COMM
            String l_map_file_name = l_command_parameters.get(0);

            // Initializing loadmap phase
            Phase l_current_phase = new LoadMapPhase(l_map_file_name,  p_need_newmap: true);
            l_current_phase.executePhase();
            GameMap l_game_map = ((LoadMapPhase) l_current_phase).getLoadedMap();

            // Initializing editmap phase
            l_current_phase = new EditMapPhase(l_game_map, l_map_file_name);
            l_current_phase.executePhase();
```

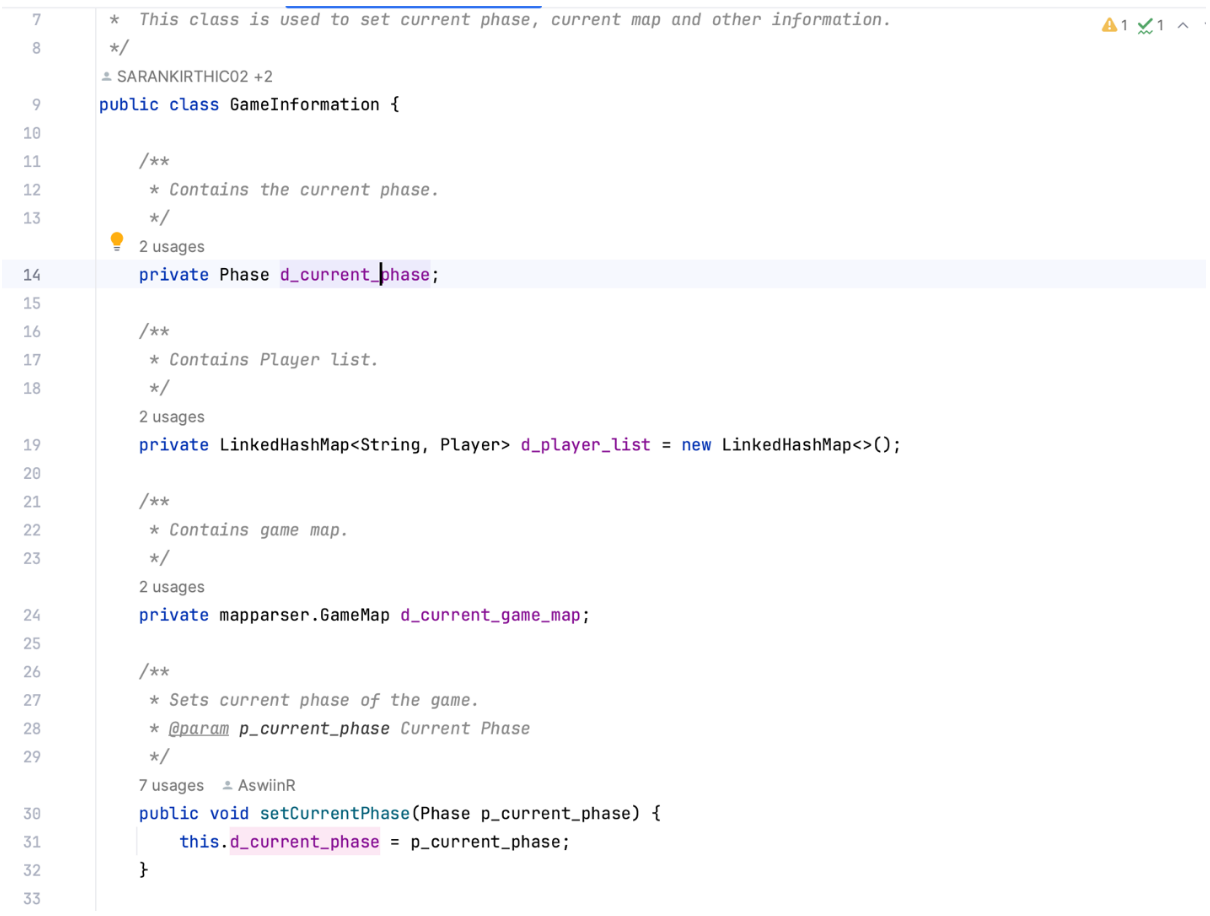
5. GameInformation converted to singleton class

Test cases: All Test classes in gameplay package

Additional test cases needed: None

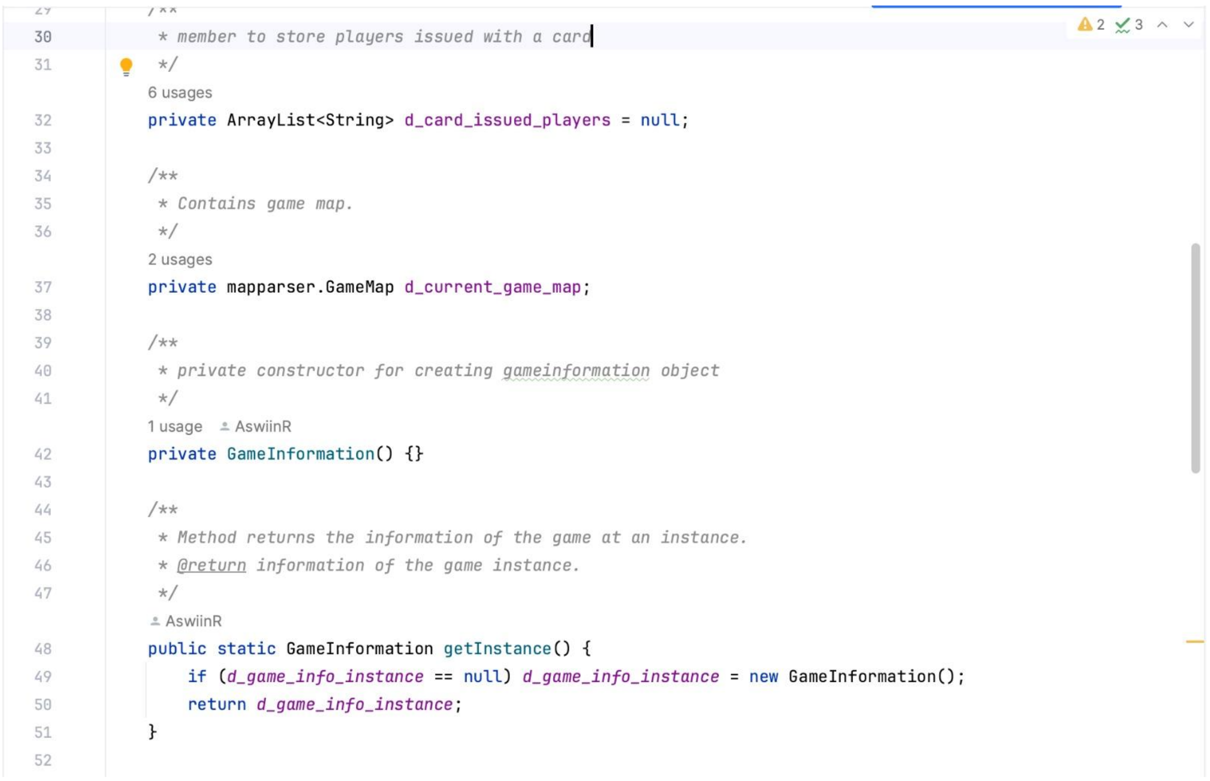
Why: GameInformation object that stores the map object, current game phase and list of player objects, is initialized when gameplay starts and is used by all other classes related to Game play. Therefore, until the game ends, only one instance of game information is required and all classes that need game information can access the object of GameInformation by calling getInstance() method.

Before:



```
 *  This class is used to set current phase, current map and other information.
 */
SARANKIRTHIC02 +2
public class GameInformation {

    /**
     * Contains the current phase.
     */
    2 usages
    private Phase d_current_phase;

    /**
     * Contains Player list.
     */
    2 usages
    private LinkedHashMap<String, Player> d_player_list = new LinkedHashMap<>();

    /**
     * Contains game map.
     */
    2 usages
    private mapparser.GameMap d_current_game_map;

    /**
     * Sets current phase of the game.
     * @param p_current_phase Current Phase
     */
    7 usages   AswiinR
    public void setCurrentPhase(Phase p_current_phase) {
        this.d_current_phase = p_current_phase;
    }

```

After:



```
    /**
     * member to store players issued with a card
     */
    6 usages
    private ArrayList<String> d_card_issued_players = null;

    /**
     * Contains game map.
     */
    2 usages
    private mapparser.GameMap d_current_game_map;

    /**
     * private constructor for creating gameinformation object
     */
    1 usage   AswiinR
    private GameInformation() {}

    /**
     * Method returns the information of the game at an instance.
     * @return information of the game instance.
     */
    AswiinR
    public static GameInformation getInstance() {
        if (d_game_info_instance == null) d_game_info_instance = new GameInformation();
        return d_game_info_instance;
    }

```