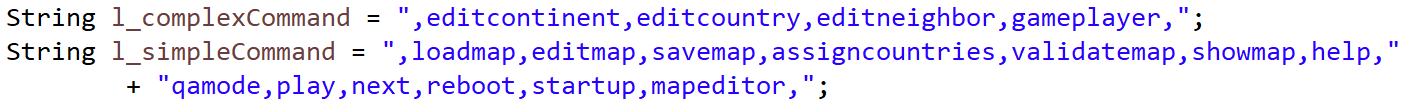
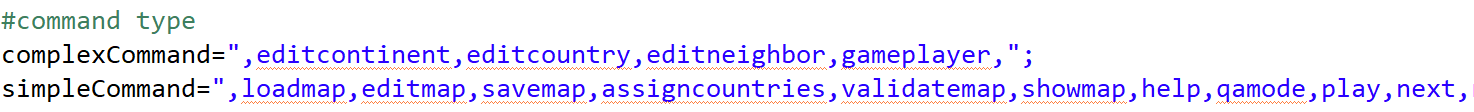
1. Potential refactoring targets
   1. hard coding:

Through discussion we find that the categorize commands into two types, simple commands and complex commands. The first version of our code is hard coding. We construct two strings containing all commands in our code, and we check the entered command whether is in the string to get the type of the command. The code is following:

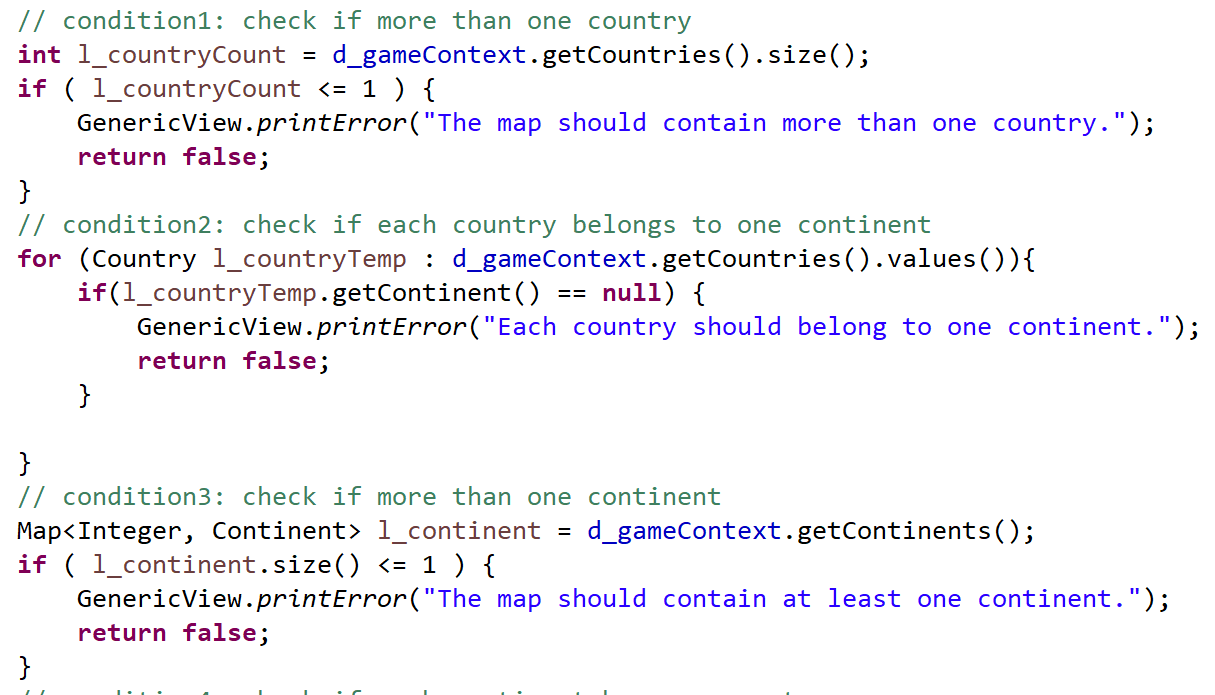


To avoid hard coding, we can put them into the configuration file. It is easier to modify the code in the future, and we do not need to change our codes when we add new commands.



* 1. simplifying the code:

After discussion, we know that there are some conditions check before the real operation on the object. We can put there condition check into another method and let it return the result of validation to make the current method only process the operation.



* 1. hard coding:

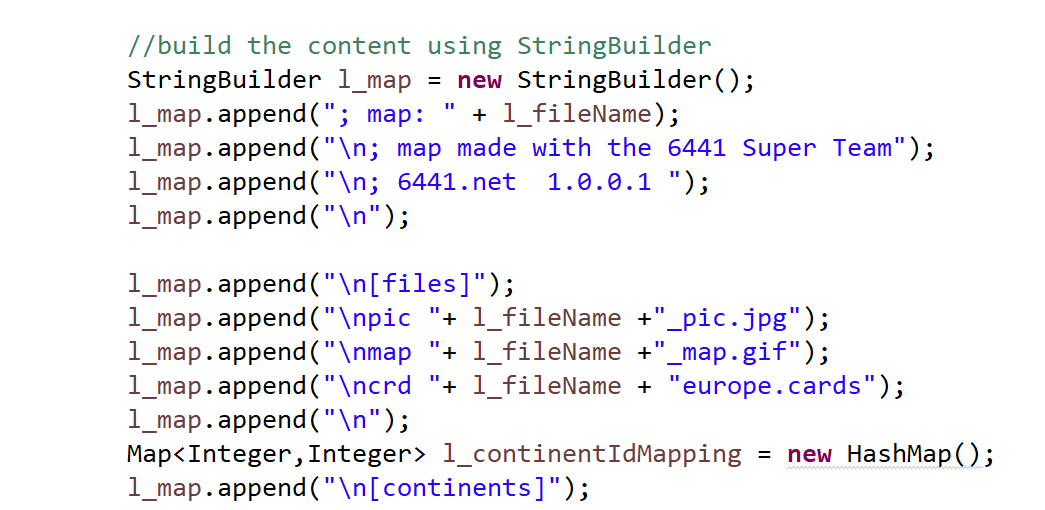
Through discussion, we find that the Action only has two types, ‘-add’ and ‘-remove’. We can use enumeration to represent it.

Graphical user interface, text, application, email

Description automatically generated

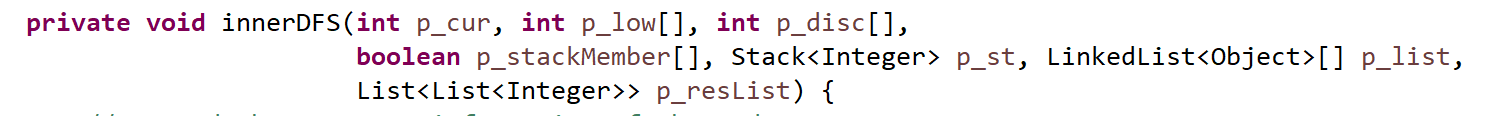
* 1. simplifying the method:

By reviewing the code, we find that we can move this operation to another method. We should avoid a method with too many lines and logic. If so, we can separate the responsibility to other methods.



* 1. change vague method names:

After reviewing the code, we find that the name of this method is not straightaway. The DFS is the name of an algorithm. We can use its full name(depth first search).



* 1. Unclear naming of methods in NeighborService.java

By reviewing NeighborService.java, we find that the function – add() and remove() – is named unclearly in this file is unclear. We suggest changing it into addNeighbor() and removeNeighbor().

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

* 1. Never used local members in Country.java

Some private members are not necessary, such as xPosition and yPosition in map. Though it is recorded in some of the map file, it is not used at all in our implementation. Therefore, they could be removed as a refactoring.

Timeline

Description automatically generated with low confidence

* 1. Never used local members in StartupService.java

We review the StartupService.java and find that: in function assigncountries(), there is a local value l\_ctr never acturally used. It should be removed.

Graphical user interface, text, application, email

Description automatically generated

* 1. Unsuitable construction of LinkedList in MapService.java

We read from the warning and find that the construct of the l\_continntAdjList in validateSubGraph() is not a good practice of java language.



It should be:

LinkedList<Object> b = new LinkedList<>();

LinkedList<LinkedList<Object>> a = new LinkedList<>();

a.add(b);

* 1. Unconventional naming of method in MapService.java

One of the experienced developer in our group indicate that the ifConnected() method is not following the naming convention as usual. It should be isConnected().

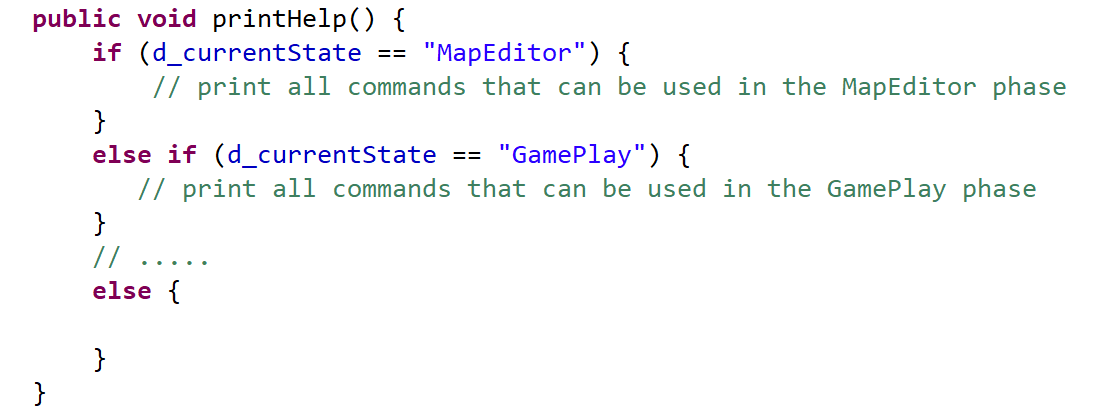
Text

Description automatically generated

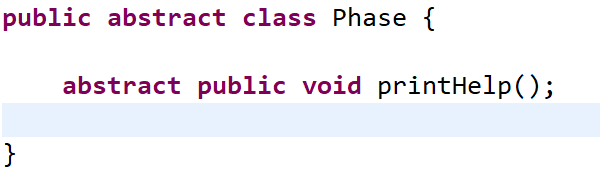
For other 5 refactoring target, we will discuss in section 2.

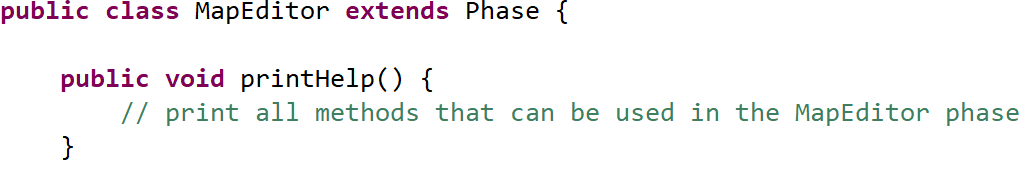
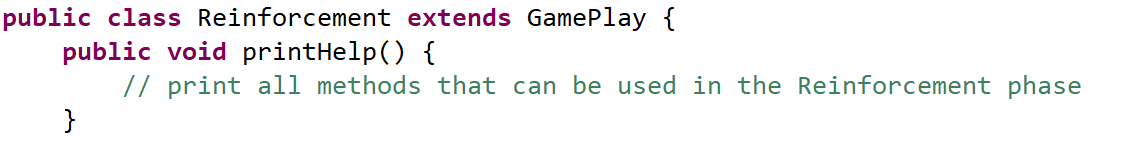
1. Actual refactoring targets
   1. Refactoring architecture using state pattern:

* Reason: The game can be divided into different phases, we can use different states to represent them so that the same command can perform different actions in different phases. For example, if we want to get some help, we can input ‘help’. We can overwrite the ‘help()’ function in each subclass of state class to give hints according to the current phase to the player, because each phase has its own order that cannot be used in other phases.
* Compare:

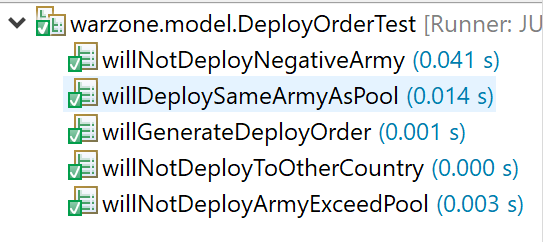
We use MVC pattern in the first version to process the actions of players. For instance, if a player wants to get some help at the beginning of the game, he enters ‘help’ into the console. There is a field to indicate current phase in the game context. The controller will invoke the ‘printHelp()’ method in the service. The trouble is this method can be very complicated and hard to maintain in the future, because we need to write all different content into the method. The code can be following: 

The situation will be better when we refactor the architecture by state pattern. We can create different classes for every phase in the game, and overwrite the ‘printHelp()’ methods. The responsibility is divided into the corresponding class, so the code is easier to understand and maintain after refactoring. The code can be following:



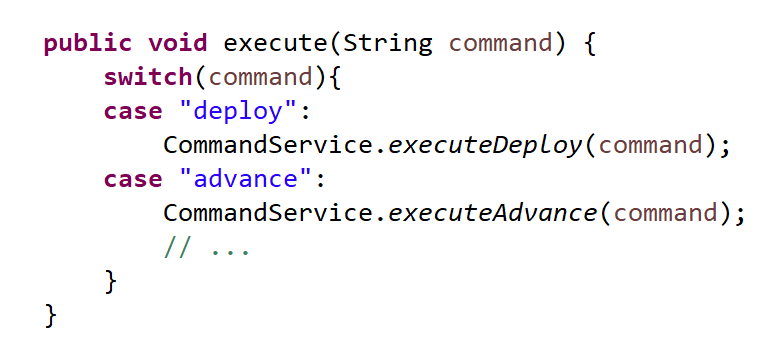
 

* Test Cases: there is just ‘deploy’ order in the first scrum.

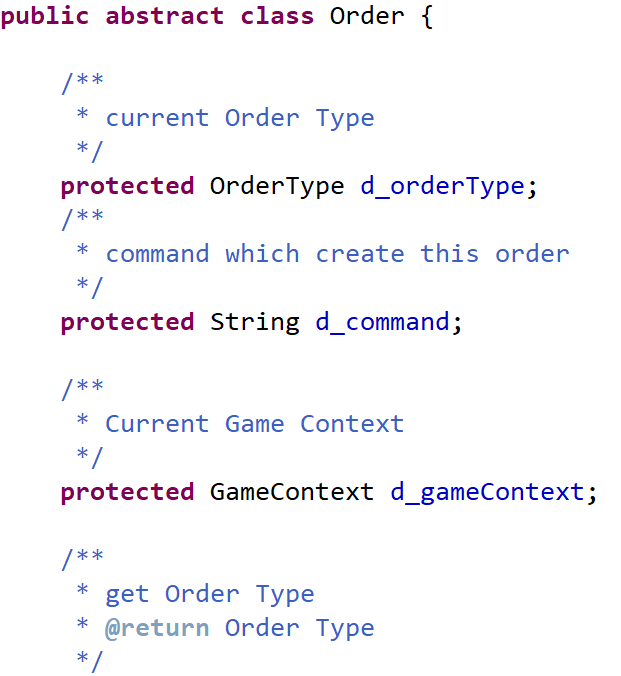


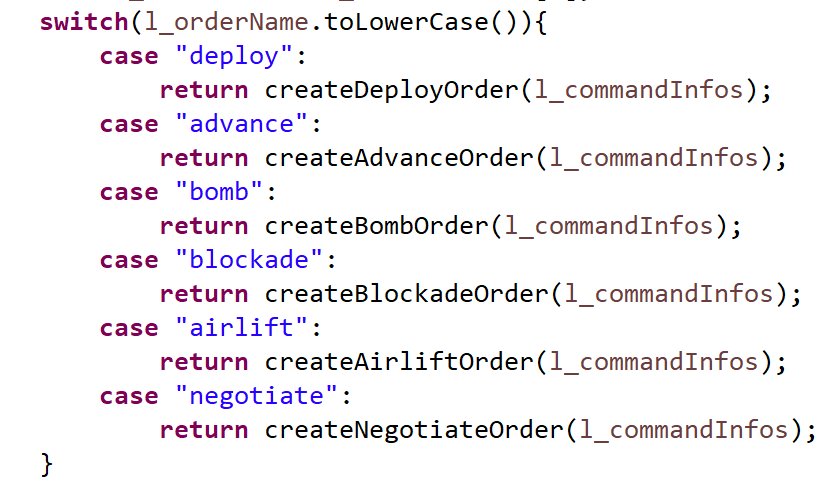
* 1. Refactoring architecture using command pattern:
* Reason: the game engine executes the orders from players in round-robin fashion, which are stored in the list of player. The execution of the commands is at a different time compared to where they are created. Besides, we do not want to know what the actual command we are executing now. If we use command pattern, we can use a clear and unified interface to execute the command.
* Compare:

If we do not use command pattern, we have to execute a certain command immediately after it is created. The extension of the code will be very awful. We will find it is very difficult to add new commands, delete and modify the existing orders. The code will be following:

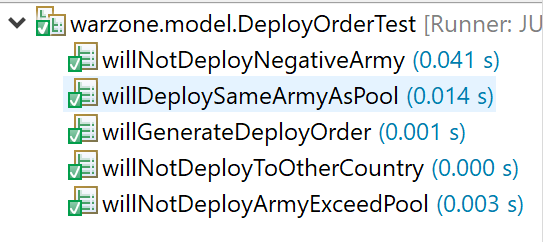


It is a different situation when we use command pattern. We can encapsulate the concreate command into different instances and initiate them with parameters. We can just return the concrete instance of the command after they are created, so we can execute the command at a different time and site. We can also do more operations on it, such as storing or reverting it in the future. The code can be following:



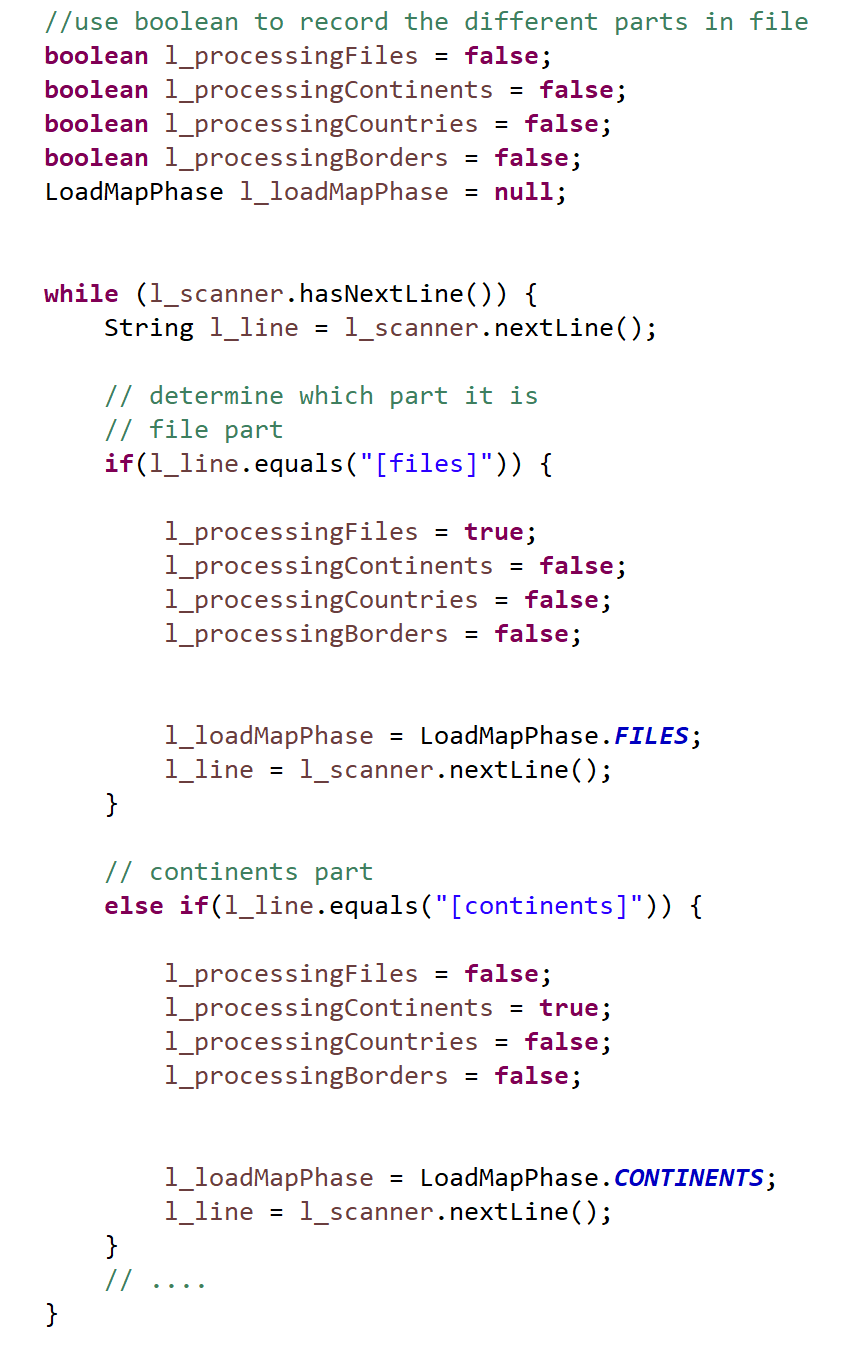


* Test Cases: there is just ‘deploy’ order in the first scrum.

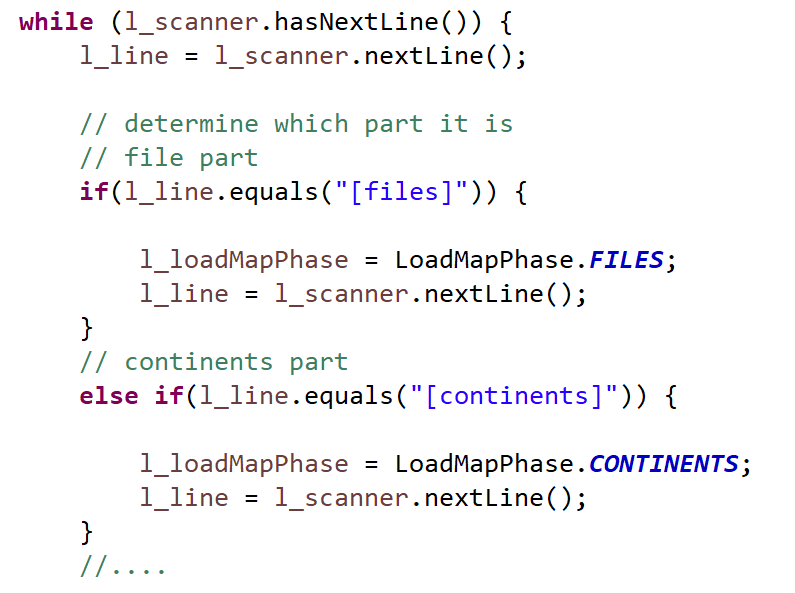


* 1. Simplifying MapEditor
* Reason: when we edit map, we have different operations for adding, removing and modifying continent, country and borders respectively. The code in the first version is difficult to understand, which uses four Boolean field to indicate the current stage. We can use four enumerations to do that thing.
* Compare:

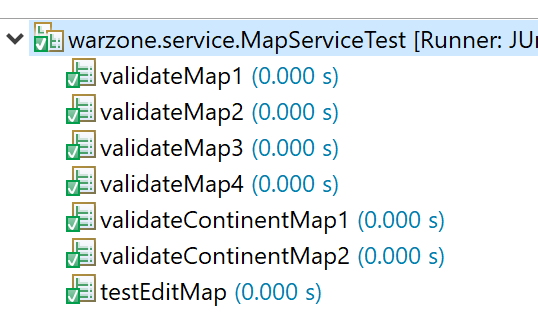
If we do not refactor out codes, we need a lot of if-else to process continent, country and borders respectively, and we also need extra fields to indicate the current operated object to use different methods for it. The code will be difficult to maintain and extent in the future. The code is following:



Things will change after refactoring it. We use a single field to indicate the current state. When we want to add something into the map, what we need to do is to add a new enumeration into the class. The code can be following:

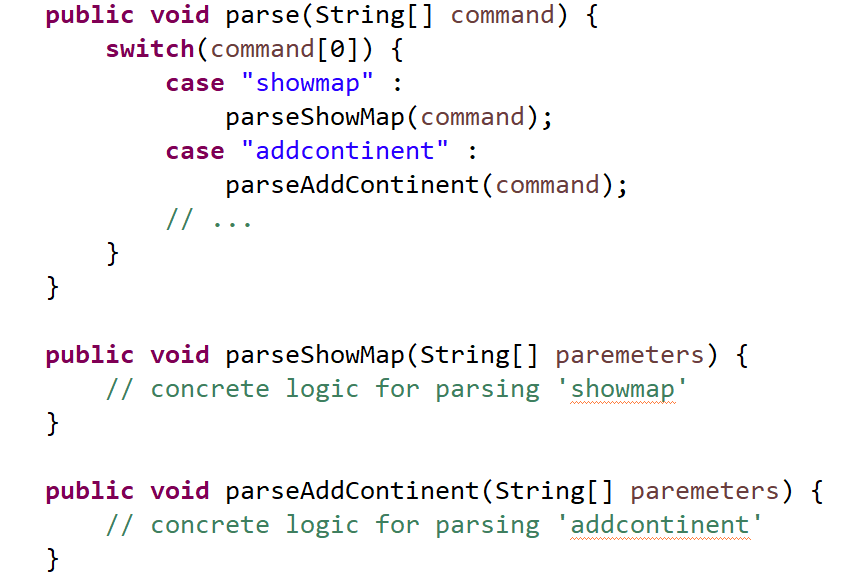


* Test cases:

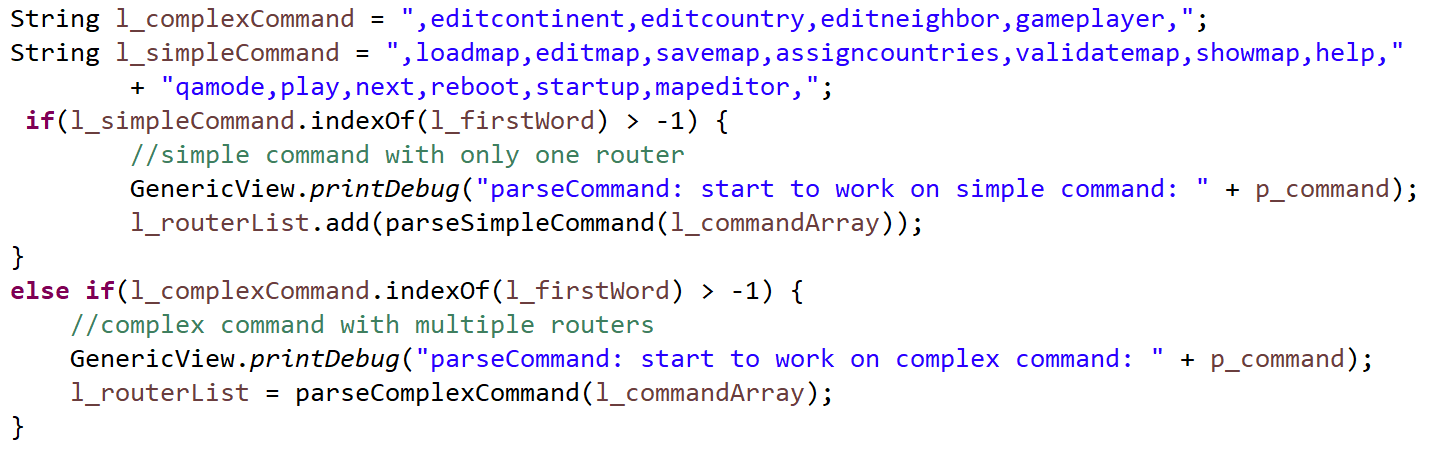


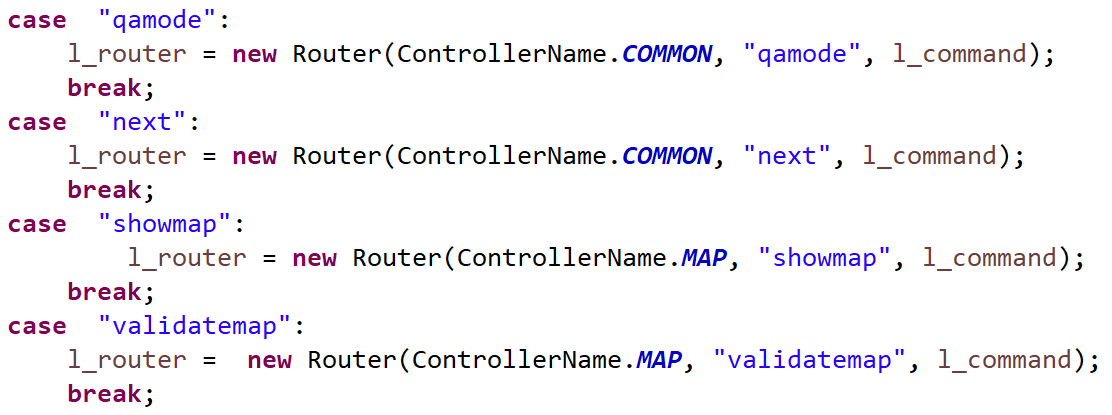
* 1. Refactoring Router by decoupling:
* Reason: the first thing we need to do is parsing the command after the player enter it. If we do not find a good way to process the command, the code will be vary hard to understand and maintain in the future. After analyzing the format of commands, we will know that commands can be categorized into simple, only containing a single word, and complicated commands which have some parameters. We can process them in different ways.
* Compare:

We need many methods to deal with every command in the game. The number of methods will be very large, and there are many unnecessary codes. The code can be following:

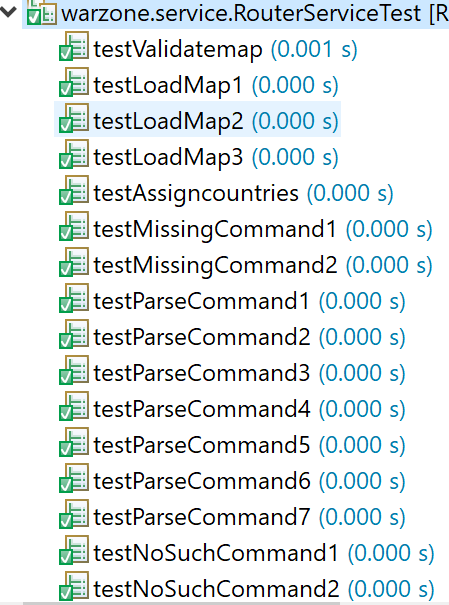


After we categorize the command type into simple and complex commands, we can use different logic to deal with them. For complex commands, such as “editcontinent -add 1 2 -remove 2”, we can also divide it into two parts which are ‘editcontinent’ and ‘-add -remove’. The first part is the concrete command, and the second part is the action of the command. The code is easier to understand and extent in the form. It can be following:

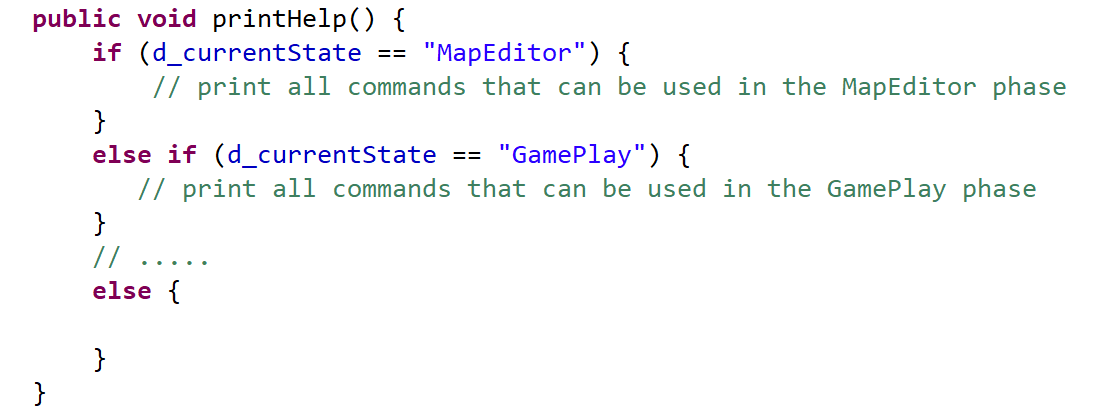




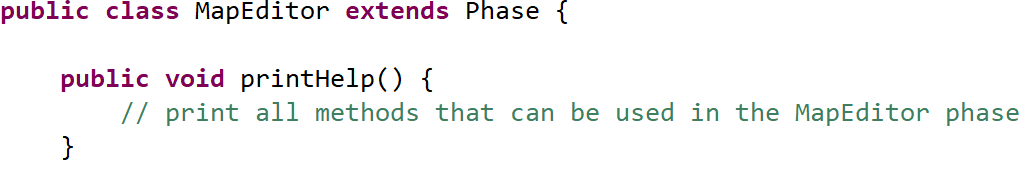
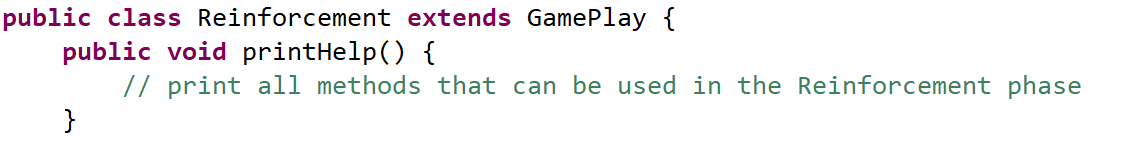
* Test cases:



* 1. Refactoring architecture using state pattern:
* Reason: The game can be divided into different phases, we can use different states to represent them so that the same command can perform different actions in different phases. For example, if we want to get some help, we can input ‘help’. We can overwrite the ‘help()’ function in each subclass of state class to give hints according to the current phase to the player, because each phase has its own order that cannot be used in other phases.
* Compare:

We use MVC pattern in the first version to process the actions of players. For instance, if a player wants to get some help at the beginning of the game, he enters ‘help’ into the console. There is a field to indicate current phase in the game context. The controller will invoke the ‘printHelp()’ method in the service. The trouble is this method can be very complicated and hard to maintain in the future, because we need to write all different content into the method. The code can be following: 

The situation will be better when we refactor the architecture by state pattern. We can create different classes for every phase in the game, and overwrite the ‘printHelp()’ methods. The responsibility is divided into the corresponding class, so the code is easier to understand and maintain after refactoring. The code can be following:

* 1. Refactoring architecture using command pattern:
* Reason: the game engine executes the orders from players in round-robin fashion, which are stored in the list of player. The execution of the commands is at a different time compared to where they are created. Besides, we do not want to know what the actual command we are executing now. If we use command pattern, we can use a clear and unified interface to execute the command.
* Compare:

If we do not use command pattern, we have to execute a certain command immediately after it is created. The extension of the code will be very awful. We will find it is very difficult to add new commands, delete and modify the existing orders. The code can be following: