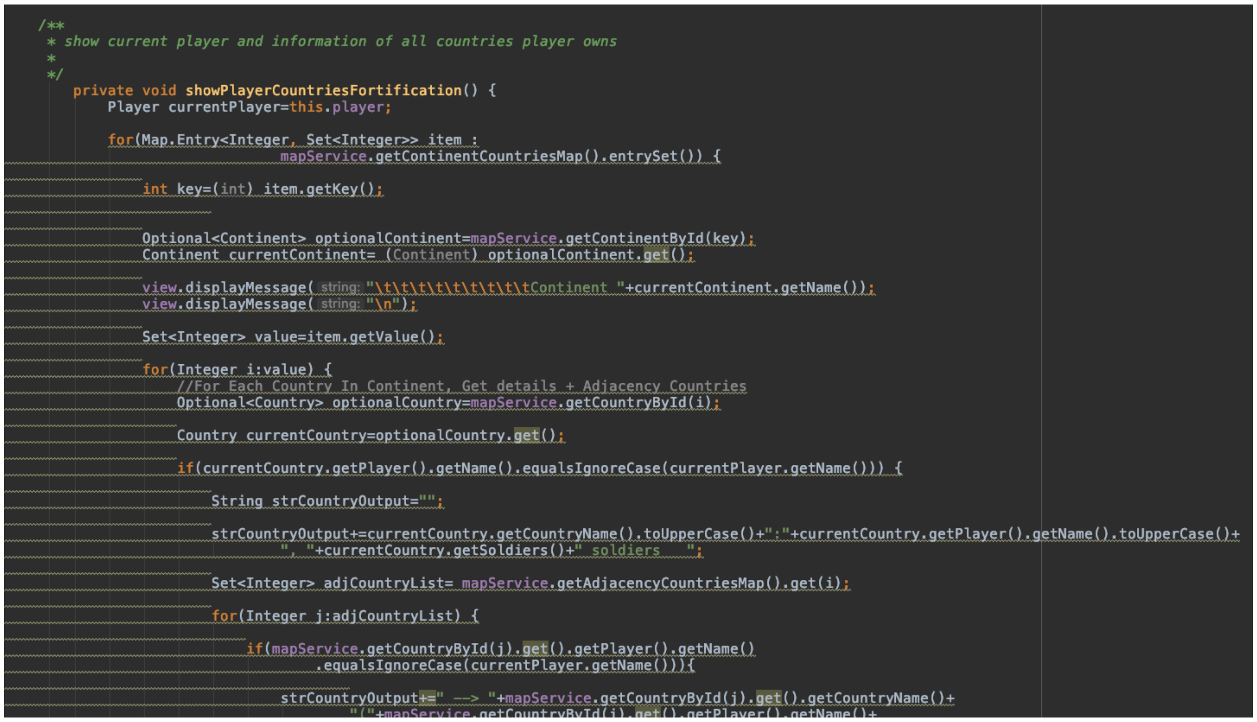
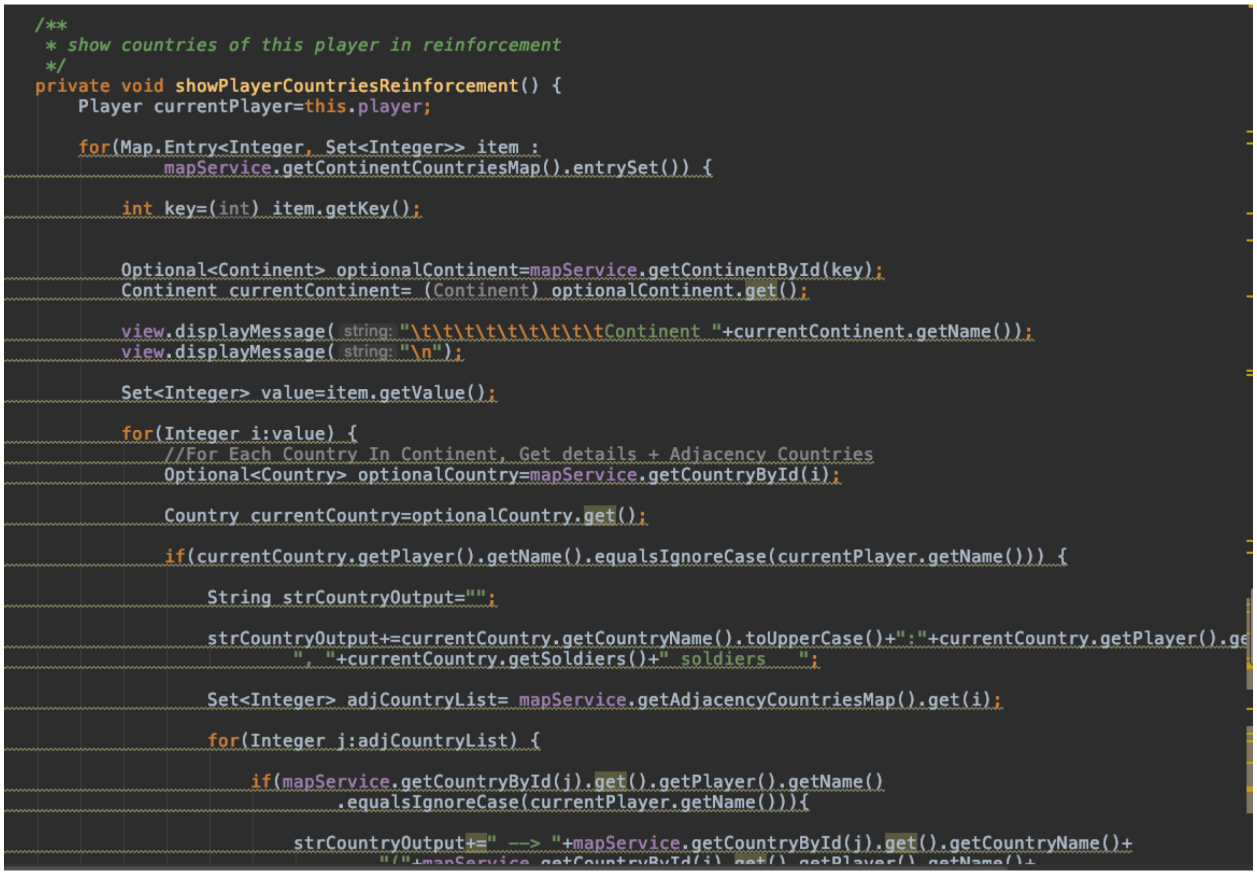
Refactoring List-Group 7

(We write tests for reinforce, startup and fortify phase before refactoring)

1. remove duplicated showCommands
   At build 1, showCommands such as showPlayer(), showAllPlayer(), showMap() exist in controllers including gameController, startUpGameController, reinforceGameController and fortifyGameController.

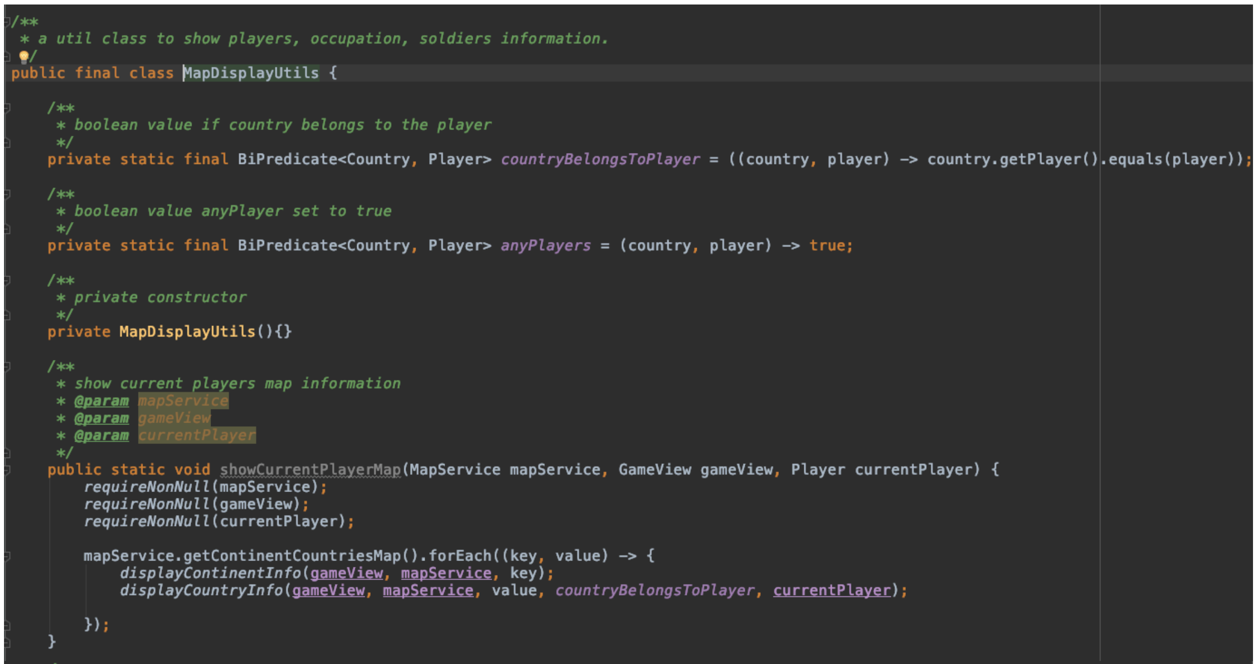


```java
/**
 * show current player and information of all countries player owns
 *
 */
    private void showPlayerCountriesFortification() {
        Player currentPlayer=this.player;

        for(Map.Entry<Integer, Set<Integer>> item :
                            mapService.getContinentCountriesMap().entrySet()) {

            int key=(int) item.getKey();


            Optional<Continent> optionalContinent=mapService.getContinentById(key);
            Continent currentContinent= (Continent) optionalContinent.get();

            view.displayMessage( string: "\t\t\t\t\t\t\t\t\tContinent "+currentContinent.getName());
            view.displayMessage( string: "\n");

            Set<Integer> value=item.getValue();

            for(Integer i:value) {
                //For Each Country In Continent, Get details + Adjacency Countries
                Optional<Country> optionalCountry=mapService.getCountryById(i);

                Country currentCountry=optionalCountry.get();

                if(currentCountry.getPlayer().getName().equalsIgnoreCase(currentPlayer.getName())) {

                    String strCountryOutput="";

                    strCountryOutput+=currentCountry.getCountryName().toUpperCase()+":"+currentCountry.getPlayer().getName().toUpperCase()+
                            ", "+currentCountry.getSoldiers()+" soldiers   ";

                    Set<Integer> adjCountryList= mapService.getAdjacencyCountriesMap().get(i);

                    for(Integer j:adjCountryList) {

                        if(mapService.getCountryById(j).get().getPlayer().getName()
                                .equalsIgnoreCase(currentPlayer.getName())){

                            strCountryOutput+=" --> "+mapService.getCountryById(j).get().getCountryName()+
                                    "("+mapService.getCountryById(j).get().getPlayer().getName()+
```

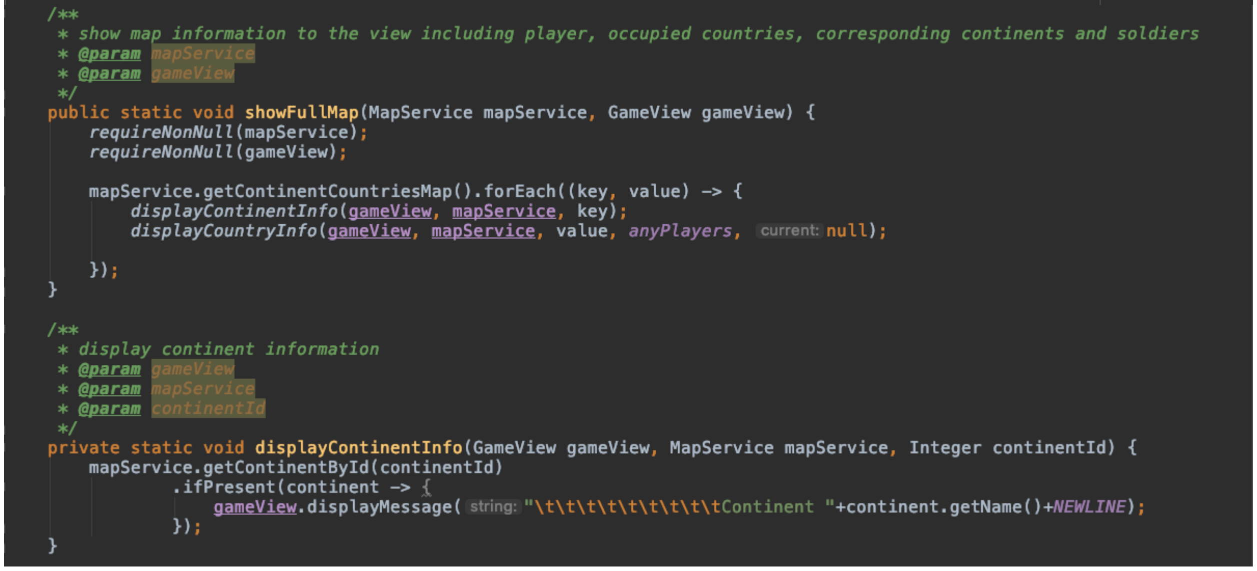


```java
/**
 * show countries of this player in reinforcement
 */
private void showPlayerCountriesReinforcement() {
    Player currentPlayer=this.player;

    for(Map.Entry<Integer, Set<Integer>> item :
            mapService.getContinentCountriesMap().entrySet()) {

        int key=(int) item.getKey();


        Optional<Continent> optionalContinent=mapService.getContinentById(key);
        Continent currentContinent= (Continent) optionalContinent.get();

        view.displayMessage( string: "\t\t\t\t\t\t\t\t\tContinent "+currentContinent.getName());
        view.displayMessage( string: "\n");

        Set<Integer> value=item.getValue();

        for(Integer i:value) {
            //For Each Country In Continent, Get details + Adjacency Countries
            Optional<Country> optionalCountry=mapService.getCountryById(i);

            Country currentCountry=optionalCountry.get();

            if(currentCountry.getPlayer().getName().equalsIgnoreCase(currentPlayer.getName())) {

                String strCountryOutput="";

                strCountryOutput+=currentCountry.getCountryName().toUpperCase()+":"+currentCountry.getPlayer().ge
                        ", "+currentCountry.getSoldiers()+" soldiers    ";

                Set<Integer> adjCountryList= mapService.getAdjacencyCountriesMap().get(i);

                for(Integer j:adjCountryList) {

                    if(mapService.getCountryById(j).get().getPlayer().getName()
                            .equalsIgnoreCase(currentPlayer.getName())){

                        strCountryOutput+=" --> "+mapService.getCountryById(j).get().getCountryName()+
                                "("+mapService.getCountryById(j).get().getPlayer().getName()+
```

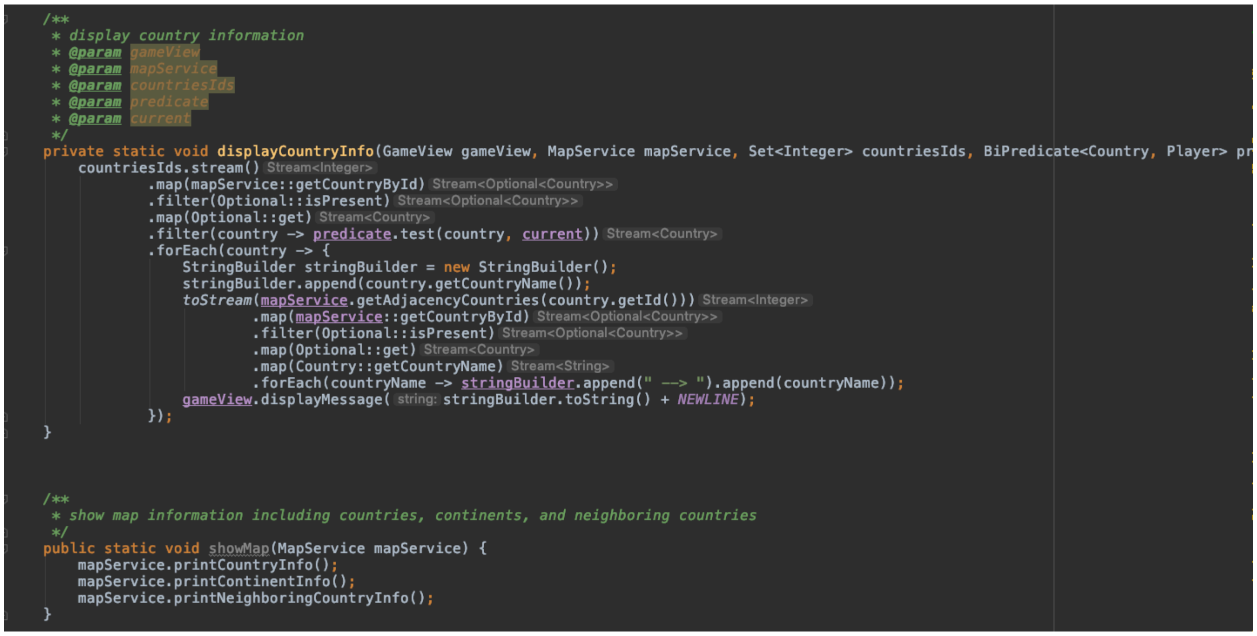
At build 2, we extract all showCommand methods and make them a static method in the MapDisplayUtils.class to avoid duplication.



```java
/**
 * a util class to show players, occupation, soldiers information.
 */
public final class MapDisplayUtils {

    /**
     * boolean value if country belongs to the player
     */
    private static final BiPredicate<Country, Player> countryBelongsToPlayer = ((country, player) -> country.getPlayer().equals(player));

    /**
     * boolean value anyPlayer set to true
     */
    private static final BiPredicate<Country, Player> anyPlayers = (country, player) -> true;

    /**
     * private constructor
     */
    private MapDisplayUtils(){}

    /**
     * show current players map information
     * @param mapService
     * @param gameView
     * @param currentPlayer
     */
    public static void showCurrentPlayerMap(MapService mapService, GameView gameView, Player currentPlayer) {
        requireNonNull(mapService);
        requireNonNull(gameView);
        requireNonNull(currentPlayer);

        mapService.getContinentCountriesMap().forEach((key, value) -> {
            displayContinentInfo(gameView, mapService, key);
            displayCountryInfo(gameView, mapService, value, countryBelongsToPlayer, currentPlayer);

        });
    }
```



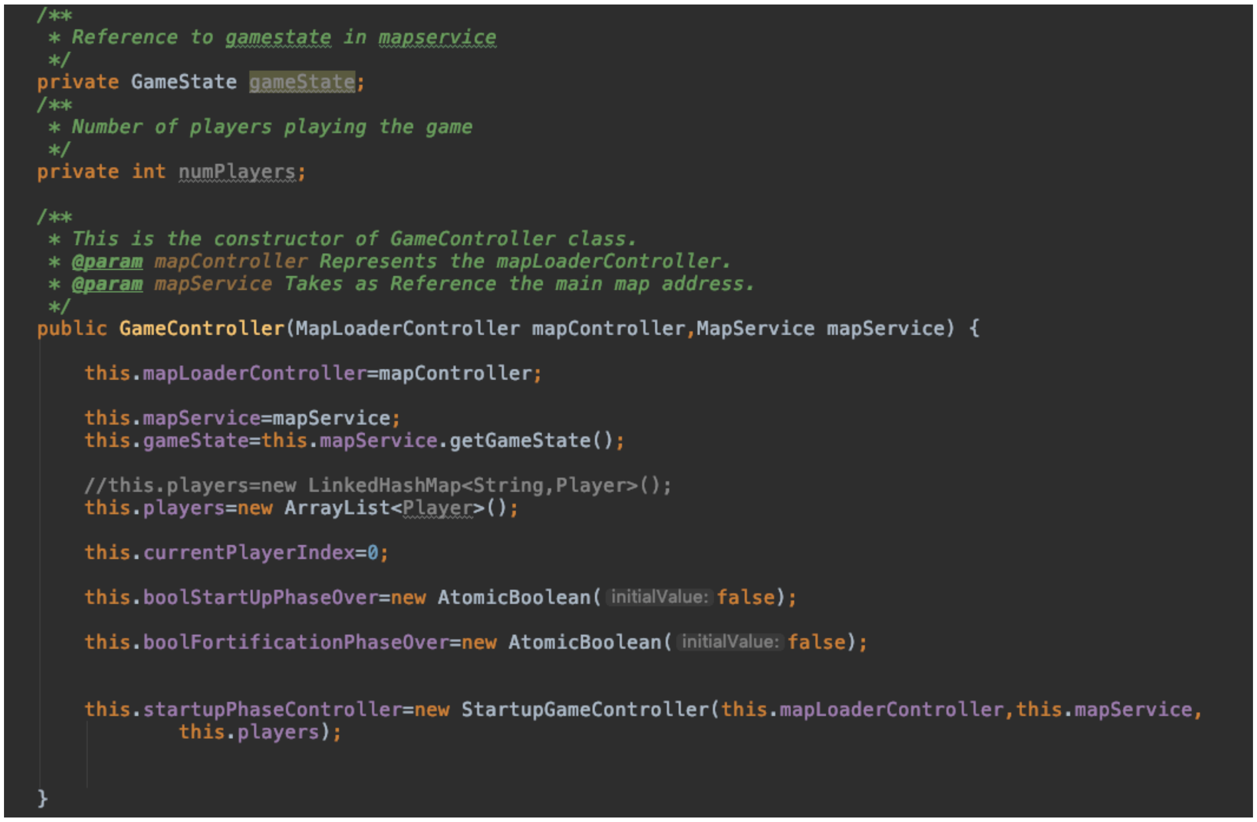
```java
    /**
     * show map information to the view including player, occupied countries, corresponding continents and soldiers
     * @param mapService
     * @param gameView
     */
    public static void showFullMap(MapService mapService, GameView gameView) {
        requireNonNull(mapService);
        requireNonNull(gameView);

        mapService.getContinentCountriesMap().forEach((key, value) -> {
            displayContinentInfo(gameView, mapService, key);
            displayCountryInfo(gameView, mapService, value, anyPlayers,  current: null);

        });
    }

    /**
     * display continent information
     * @param gameView
     * @param mapService
     * @param continentId
     */
    private static void displayContinentInfo(GameView gameView, MapService mapService, Integer continentId) {
        mapService.getContinentById(continentId)
                .ifPresent(continent -> {
                    gameView.displayMessage( string: "\t\t\t\t\t\t\t\t\tContinent "+continent.getName()+NEWLINE);
                });
    }
```



```java
/**
 * display country information
 * @param gameView
 * @param mapService
 * @param countriesIds
 * @param predicate
 * @param current
 */
private static void displayCountryInfo(GameView gameView, MapService mapService, Set<Integer> countriesIds, BiPredicate<Country, Player> pr
    countriesIds.stream() Stream<Integer>
            .map(mapService::getCountryById) Stream<Optional<Country>>
            .filter(Optional::isPresent) Stream<Optional<Country>>
            .map(Optional::get) Stream<Country>
            .filter(country -> predicate.test(country, current)) Stream<Country>
            .forEach(country -> {
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.append(country.getCountryName());
                toStream(mapService.getAdjacencyCountries(country.getId())) Stream<Integer>
                        .map(mapService::getCountryById) Stream<Optional<Country>>
                        .filter(Optional::isPresent) Stream<Optional<Country>>
                        .map(Optional::get) Stream<Country>
                        .map(Country::getCountryName) Stream<String>
                        .forEach(countryName -> stringBuilder.append(" --> ").append(countryName));
                gameView.displayMessage( string: stringBuilder.toString() + NEWLINE);
            });
}



/**
 * show map information including countries, continents, and neighboring countries
 */
public static void showMap(MapService mapService) {
    mapService.printCountryInfo();
    mapService.printContinentInfo();
    mapService.printNeighboringCountryInfo();
}
```

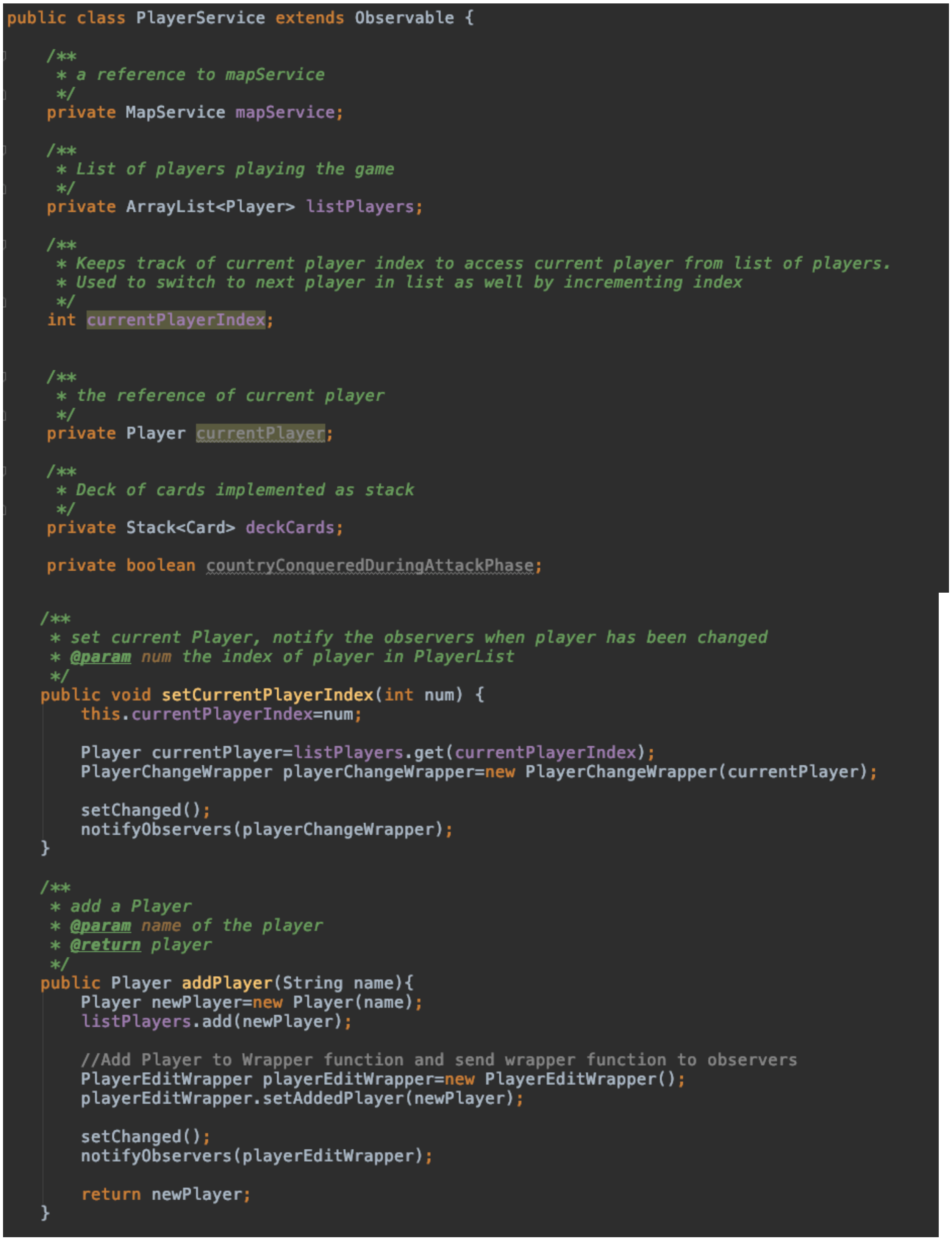
2. move player state and corresponding logics (set and get current player, add and remove player, switch players) from the gameController to model named PlayerService.class.

   At build 1, player state and corresponding logic in gameController

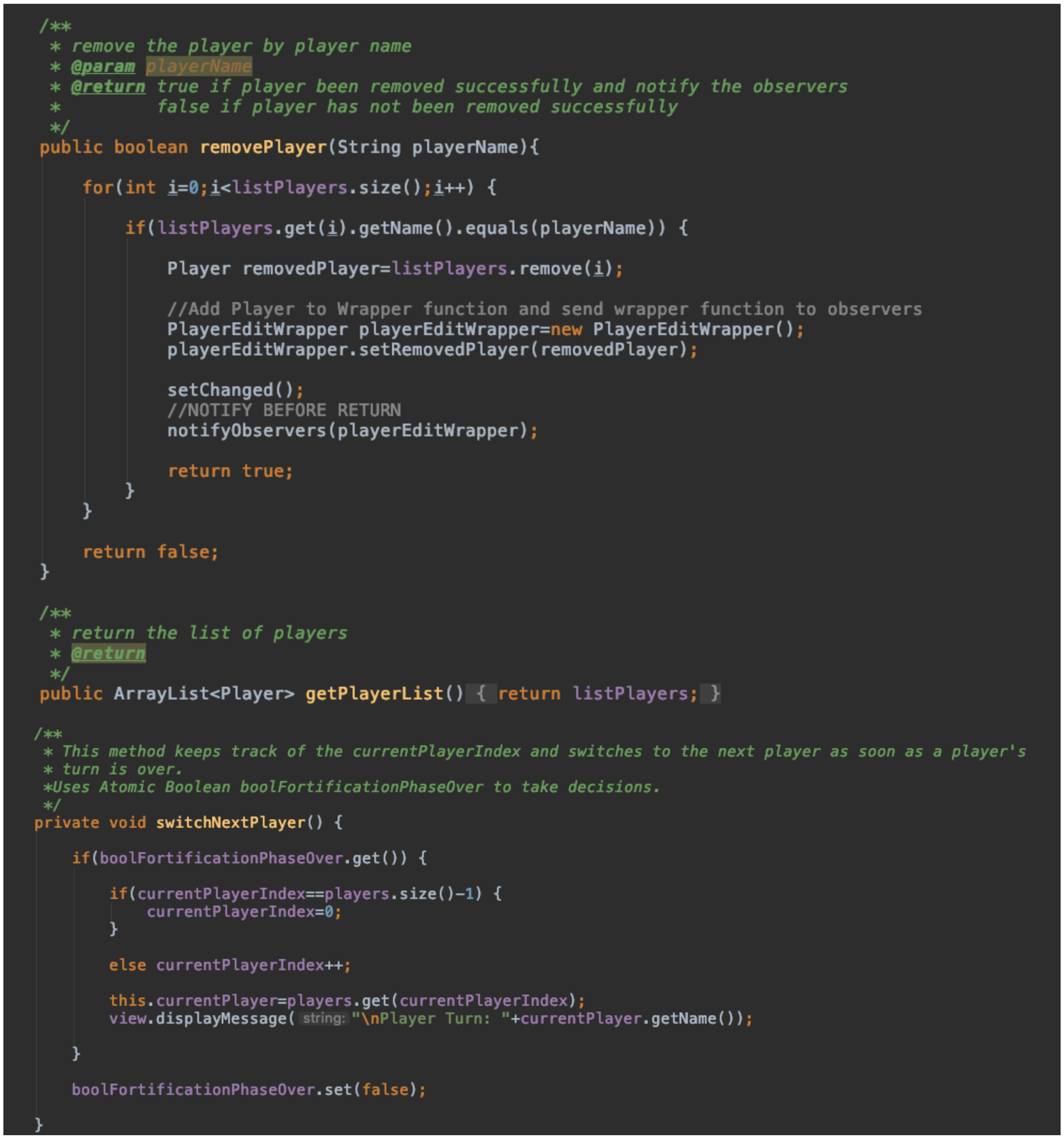


```java
/**
 * Reference to gamestate in mapservice
 */
private GameState gameState;
/**
 * Number of players playing the game
 */
private int numPlayers;

/**
 * This is the constructor of GameController class.
 * @param mapController Represents the mapLoaderController.
 * @param mapService Takes as Reference the main map address.
 */
public GameController(MapLoaderController mapController,MapService mapService) {

    this.mapLoaderController=mapController;

    this.mapService=mapService;
    this.gameState=this.mapService.getGameState();

    //this.players=new LinkedHashMap<String,Player>();
    this.players=new ArrayList<Player>();

    this.currentPlayerIndex=0;

    this.boolStartUpPhaseOver=new AtomicBoolean( initialValue: false);

    this.boolFortificationPhaseOver=new AtomicBoolean( initialValue: false);


    this.startupPhaseController=new StartupGameController(this.mapLoaderController,this.mapService,
            this.players);


}
```

At build 2, we move player state and corresponding logic in model named PlayerService



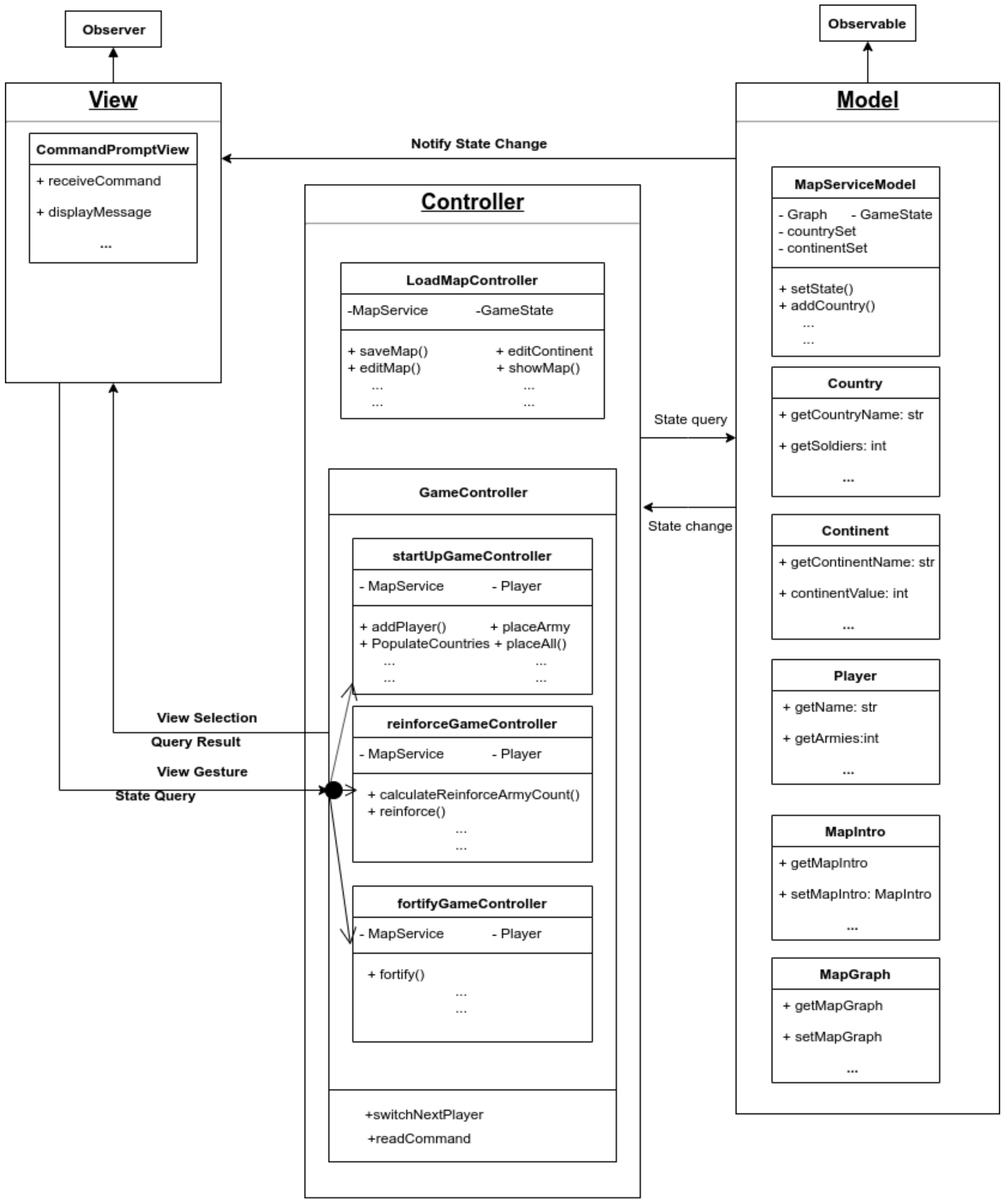
```java
public class PlayerService extends Observable {

    /**
     * a reference to mapService
     */
    private MapService mapService;

    /**
     * List of players playing the game
     */
    private ArrayList<Player> listPlayers;

    /**
     * Keeps track of current player index to access current player from list of players.
     * Used to switch to next player in list as well by incrementing index
     */
    int currentPlayerIndex;


    /**
     * the reference of current player
     */
    private Player currentPlayer;

    /**
     * Deck of cards implemented as stack
     */
    private Stack<Card> deckCards;

    private boolean countryConqueredDuringAttackPhase;


   /**
    * set current Player, notify the observers when player has been changed
    * @param num the index of player in PlayerList
    */
   public void setCurrentPlayerIndex(int num) {
       this.currentPlayerIndex=num;

       Player currentPlayer=listPlayers.get(currentPlayerIndex);
       PlayerChangeWrapper playerChangeWrapper=new PlayerChangeWrapper(currentPlayer);

       setChanged();
       notifyObservers(playerChangeWrapper);
   }

   /**
    * add a Player
    * @param name of the player
    * @return player
    */
   public Player addPlayer(String name){
       Player newPlayer=new Player(name);
       listPlayers.add(newPlayer);

       //Add Player to Wrapper function and send wrapper function to observers
       PlayerEditWrapper playerEditWrapper=new PlayerEditWrapper();
       playerEditWrapper.setAddedPlayer(newPlayer);

       setChanged();
       notifyObservers(playerEditWrapper);

       return newPlayer;
   }
```



```
/**
 * remove the player by player name
 * @param playerName
 * @return true if player been removed successfully and notify the observers
 *         false if player has not been removed successfully
 */
public boolean removePlayer(String playerName){

    for(int i=0;i<listPlayers.size();i++) {

        if(listPlayers.get(i).getName().equals(playerName)) {

            Player removedPlayer=listPlayers.remove(i);

            //Add Player to Wrapper function and send wrapper function to observers
            PlayerEditWrapper playerEditWrapper=new PlayerEditWrapper();
            playerEditWrapper.setRemovedPlayer(removedPlayer);

            setChanged();
            //NOTIFY BEFORE RETURN
            notifyObservers(playerEditWrapper);

            return true;
        }
    }

    return false;
}

/**
 * return the list of players
 * @return
 */
public ArrayList<Player> getPlayerList() { return listPlayers; }

/**
 * This method keeps track of the currentPlayerIndex and switches to the next player as soon as a player's
 * turn is over.
 *Uses Atomic Boolean boolFortificationPhaseOver to take decisions.
 */
private void switchNextPlayer() {

    if(boolFortificationPhaseOver.get()) {

        if(currentPlayerIndex==players.size()-1) {
            currentPlayerIndex=0;
        }

        else currentPlayerIndex++;

        this.currentPlayer=players.get(currentPlayerIndex);
        view.displayMessage( string: "\nPlayer Turn: "+currentPlayer.getName());

    }

    boolFortificationPhaseOver.set(false);

}
```

3. remove game controller, unnecessary boolean values and unnecessary dependency of other controllers on gameController. This improves the readability and extensibility of the code.

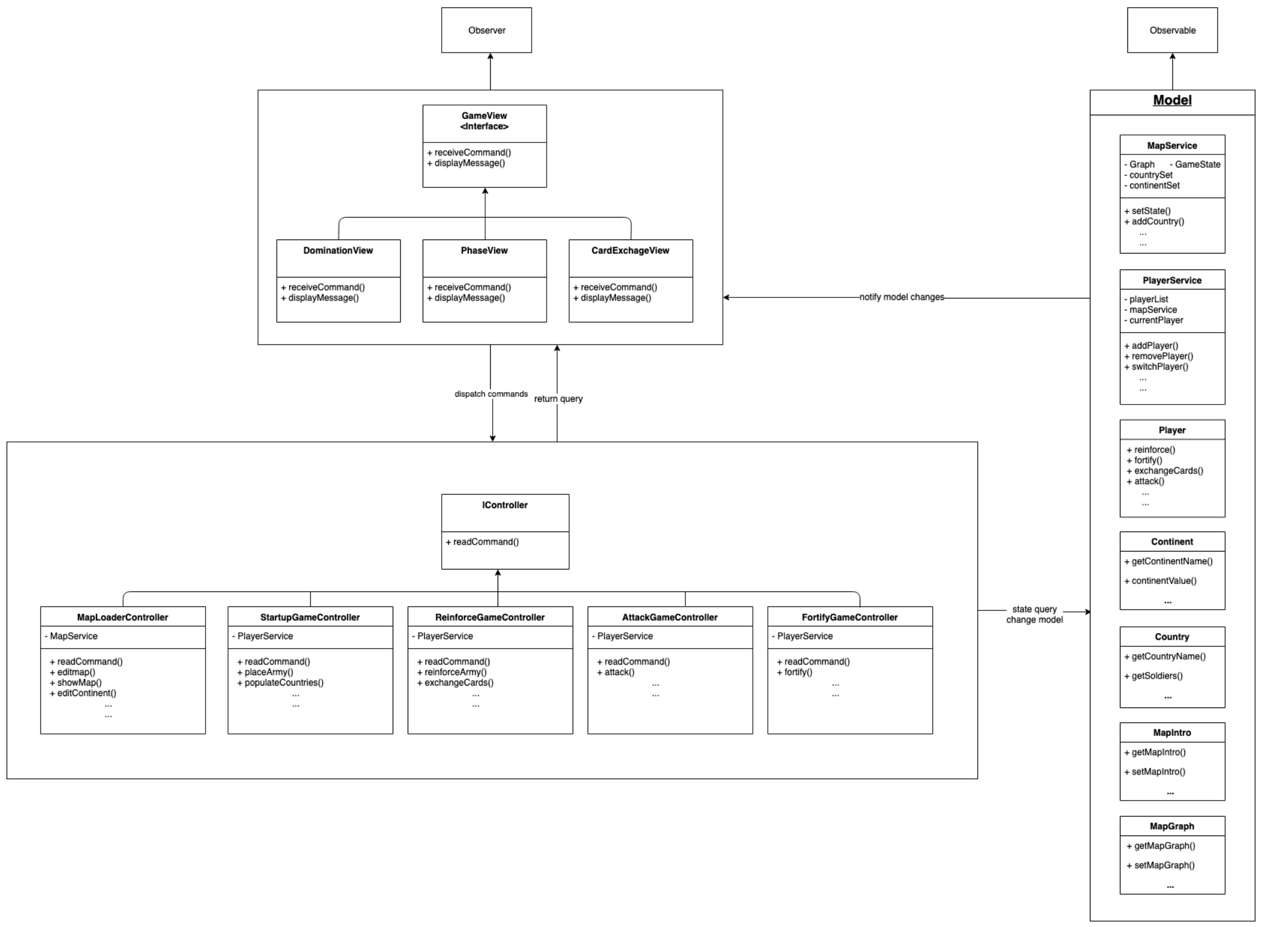
   At build 1, game controller has references of other controllers. It also has a lot of boolean values to determine which controller to call based on these boolean values.



Achitecture Design of Risk Project

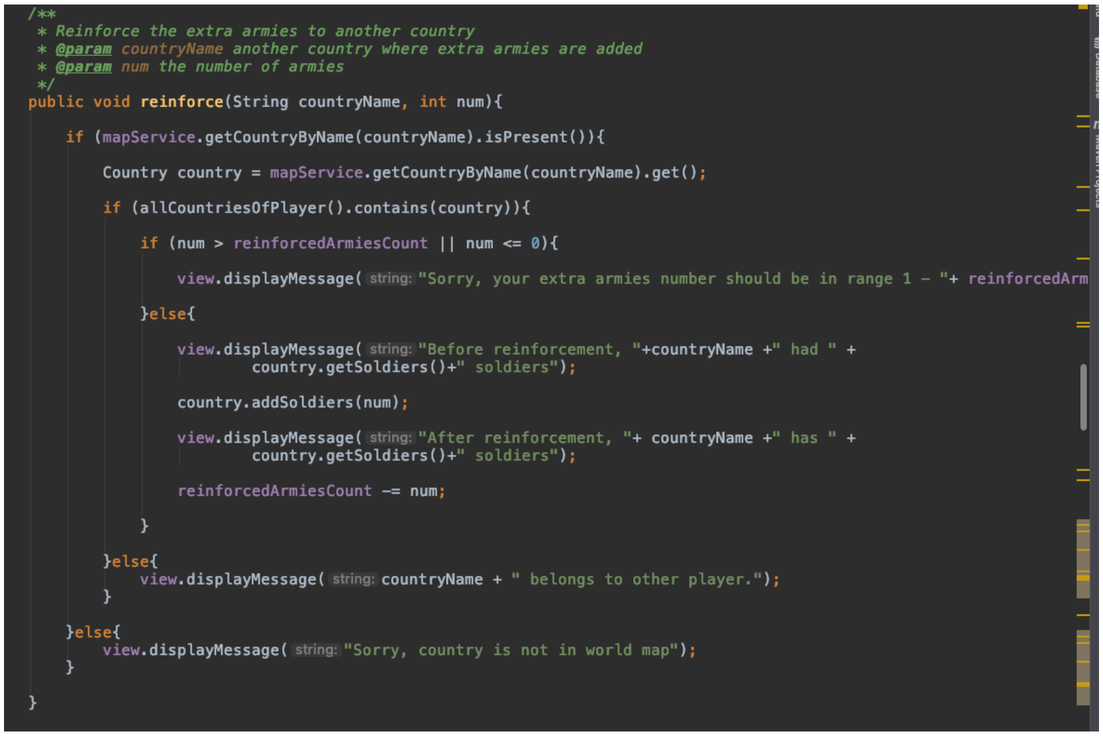
At build 2, we remove game controller, its dependency on other controllers and remove unnecessary Boolean values.

Achitecture Design of Risk Project



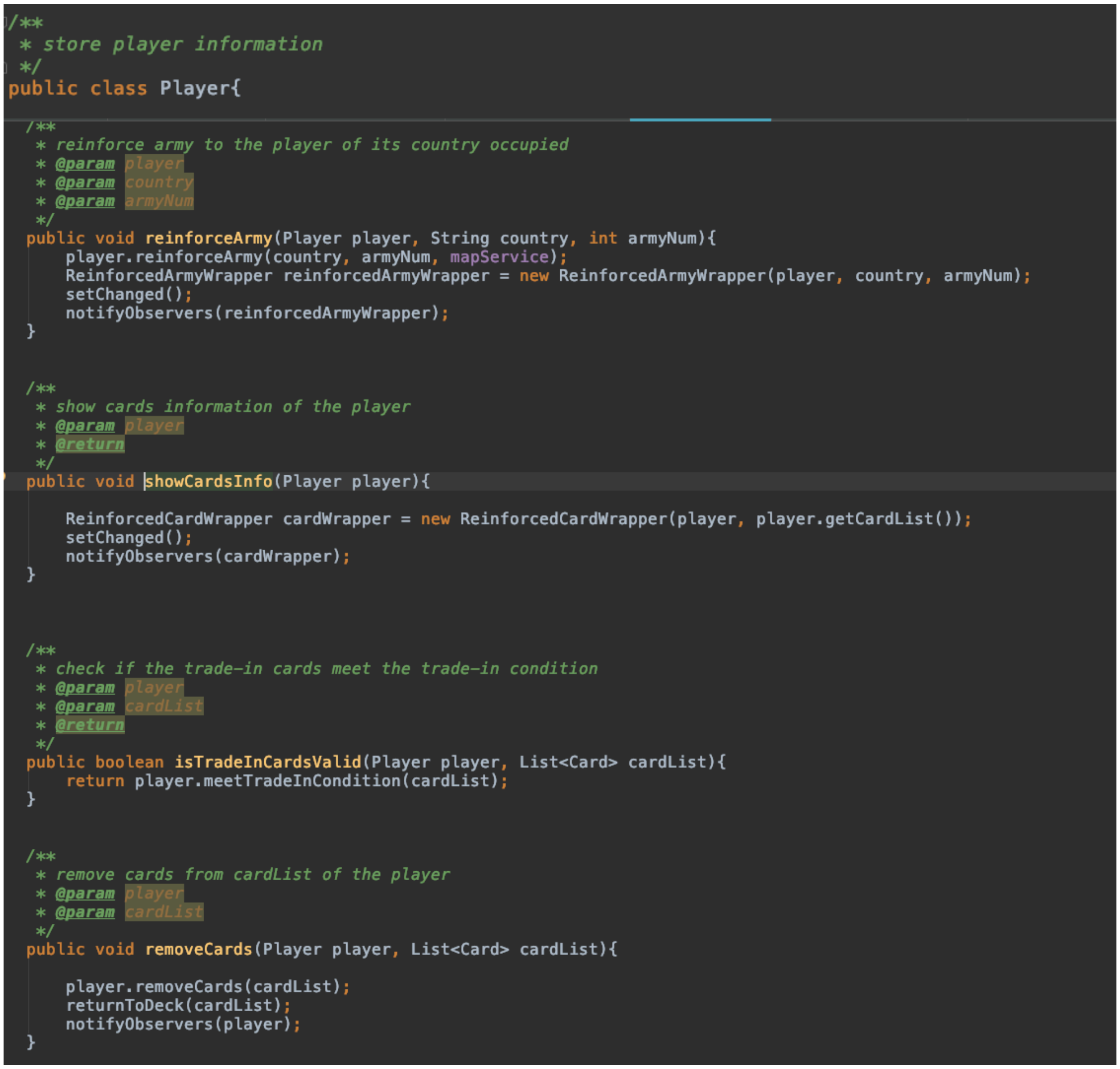
4. remove logic relating to reinforce(), exchangeCards() from reinforceController to model Player.class. The reinforceController is used to check the validity of the command

At build 1, logic of reinforce and exchangeCards and command validation are all in ReinforceGameController



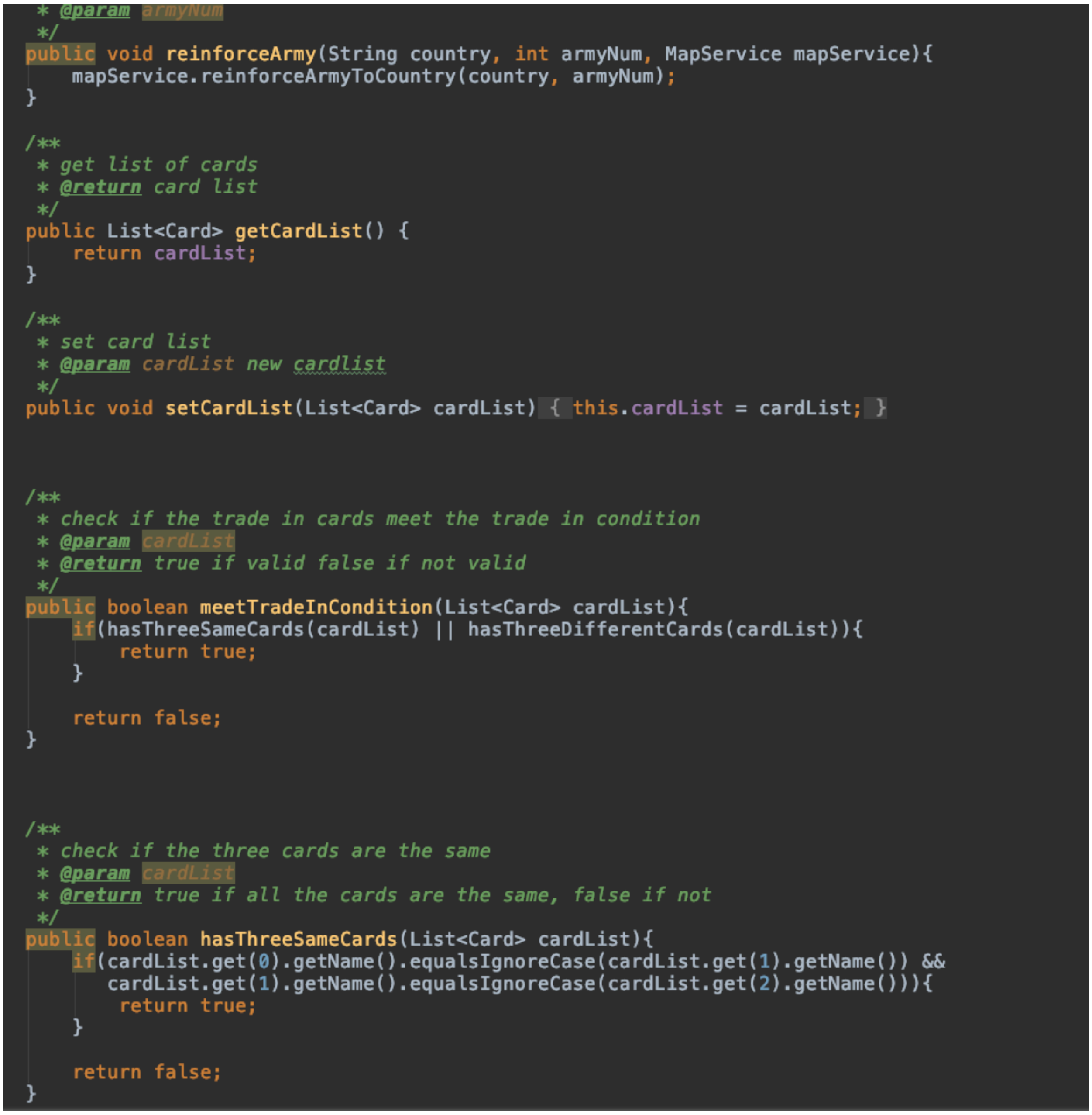
```
/**
 * Reinforce the extra armies to another country
 * @param countryName another country where extra armies are added
 * @param num the number of armies
 */
public void reinforce(String countryName, int num){

    if (mapService.getCountryByName(countryName).isPresent()){

        Country country = mapService.getCountryByName(countryName).get();

        if (allCountriesOfPlayer().contains(country)){

            if (num > reinforcedArmiesCount || num <= 0){

                view.displayMessage( string: "Sorry, your extra armies number should be in range 1 - "+ reinforcedArm

            }else{

                view.displayMessage( string: "Before reinforcement, "+countryName +" had " +
                        country.getSoldiers()+" soldiers");

                country.addSoldiers(num);

                view.displayMessage( string: "After reinforcement, "+ countryName +" has " +
                        country.getSoldiers()+" soldiers");

                reinforcedArmiesCount -= num;

            }

        }else{
            view.displayMessage( string: countryName + " belongs to other player.");
        }

    }else{
        view.displayMessage( string: "Sorry, country is not in world map");
    }

}
```

At build 2, we move logic of reinforce and exchangeCards to model named Player while keeping command validation in ReinforceGameController

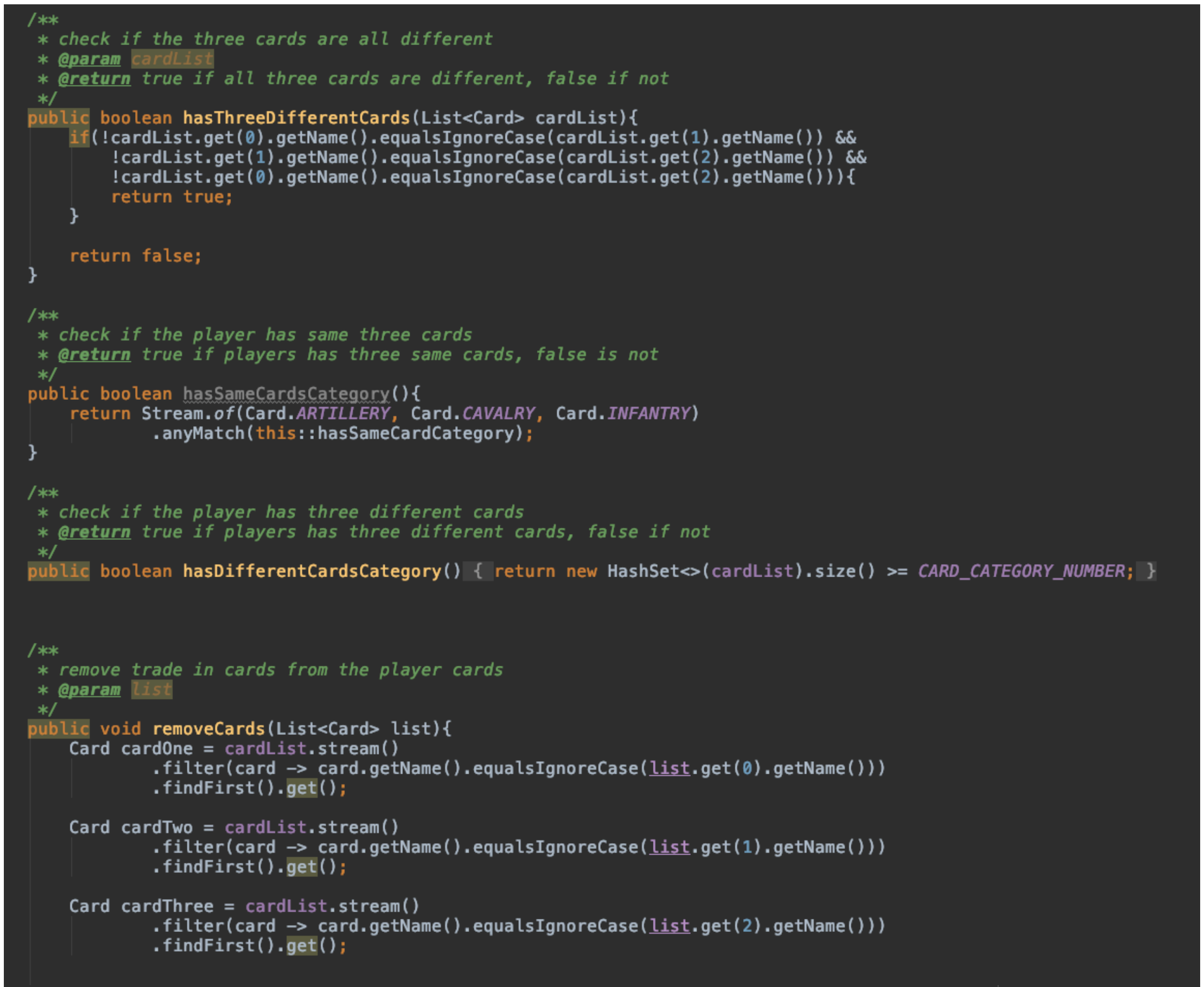


```
/**
 * store player information
 */
public class Player{
```

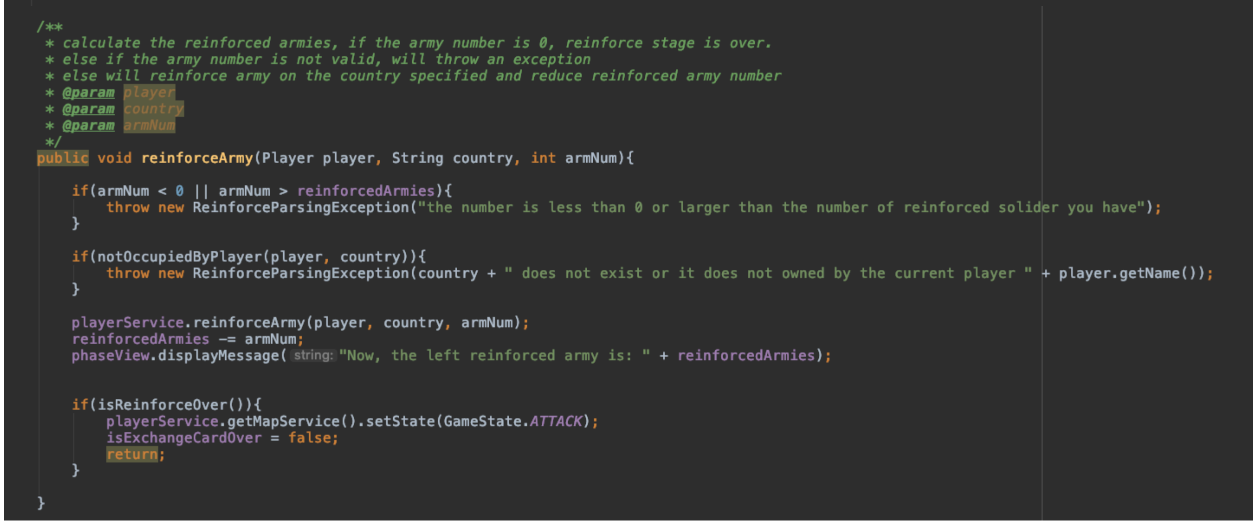
```
    /**
     * reinforce army to the player of its country occupied
     * @param player
     * @param country
     * @param armyNum
     */
    public void reinforceArmy(Player player, String country, int armyNum){
        player.reinforceArmy(country, armyNum, mapService);
        ReinforcedArmyWrapper reinforcedArmyWrapper = new ReinforcedArmyWrapper(player, country, armyNum);
        setChanged();
        notifyObservers(reinforcedArmyWrapper);
    }


    /**
     * show cards information of the player
     * @param player
     * @return
     */
    public void showCardsInfo(Player player){

        ReinforcedCardWrapper cardWrapper = new ReinforcedCardWrapper(player, player.getCardList());
        setChanged();
        notifyObservers(cardWrapper);
    }



    /**
     * check if the trade-in cards meet the trade-in condition
     * @param player
     * @param cardList
     * @return
     */
    public boolean isTradeInCardsValid(Player player, List<Card> cardList){
        return player.meetTradeInCondition(cardList);
    }


    /**
     * remove cards from cardList of the player
     * @param player
     * @param cardList
     */
    public void removeCards(Player player, List<Card> cardList){

        player.removeCards(cardList);
        returnToDeck(cardList);
        notifyObservers(player);
    }
```



```
     * @param armyNum
     */
    public void reinforceArmy(String country, int armyNum, MapService mapService){
        mapService.reinforceArmyToCountry(country, armyNum);
    }

    /**
     * get list of cards
     * @return card list
     */
    public List<Card> getCardList() {
        return cardList;
    }

    /**
     * set card list
     * @param cardList new cardlist
     */
    public void setCardList(List<Card> cardList) { this.cardList = cardList; }



    /**
     * check if the trade in cards meet the trade in condition
     * @param cardList
     * @return true if valid false if not valid
     */
    public boolean meetTradeInCondition(List<Card> cardList){
        if(hasThreeSameCards(cardList) || hasThreeDifferentCards(cardList)){
            return true;
        }

        return false;
    }



    /**
     * check if the three cards are the same
     * @param cardList
     * @return true if all the cards are the same, false if not
     */
    public boolean hasThreeSameCards(List<Card> cardList){
        if(cardList.get(0).getName().equalsIgnoreCase(cardList.get(1).getName()) &&
           cardList.get(1).getName().equalsIgnoreCase(cardList.get(2).getName())){
            return true;
        }

        return false;
    }
```



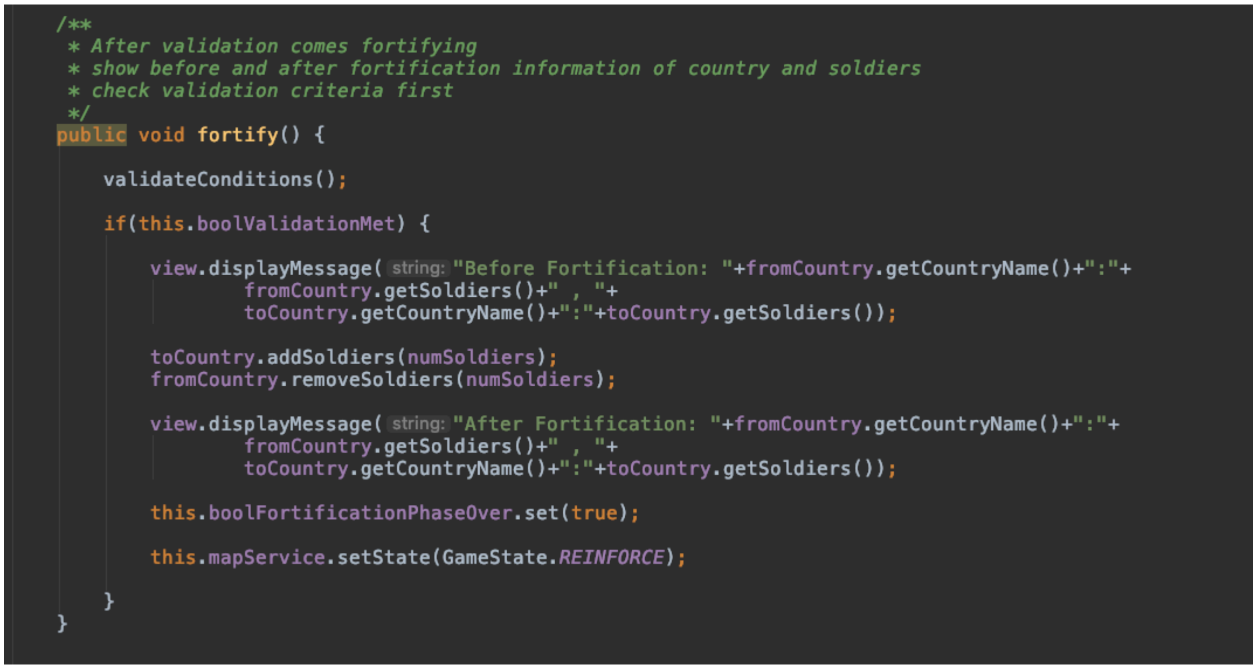
```java
/**
 * check if the three cards are all different
 * @param cardList
 * @return true if all three cards are different, false if not
 */
public boolean hasThreeDifferentCards(List<Card> cardList){
    if(!cardList.get(0).getName().equalsIgnoreCase(cardList.get(1).getName()) &&
        !cardList.get(1).getName().equalsIgnoreCase(cardList.get(2).getName()) &&
        !cardList.get(0).getName().equalsIgnoreCase(cardList.get(2).getName())){
        return true;
    }

    return false;
}

/**
 * check if the player has same three cards
 * @return true if players has three same cards, false is not
 */
public boolean hasSameCardsCategory(){
    return Stream.of(Card.ARTILLERY, Card.CAVALRY, Card.INFANTRY)
            .anyMatch(this::hasSameCardCategory);
}

/**
 * check if the player has three different cards
 * @return true if players has three different cards, false if not
 */
public boolean hasDifferentCardsCategory() { return new HashSet<>(cardList).size() >= CARD_CATEGORY_NUMBER; }



/**
 * remove trade in cards from the player cards
 * @param list
 */
public void removeCards(List<Card> list){
    Card cardOne = cardList.stream()
            .filter(card -> card.getName().equalsIgnoreCase(list.get(0).getName()))
            .findFirst().get();

    Card cardTwo = cardList.stream()
            .filter(card -> card.getName().equalsIgnoreCase(list.get(1).getName()))
            .findFirst().get();

    Card cardThree = cardList.stream()
            .filter(card -> card.getName().equalsIgnoreCase(list.get(2).getName()))
            .findFirst().get();
```



```java
/**
 * calculate the reinforced armies, if the army number is 0, reinforce stage is over.
 * else if the army number is not valid, will throw an exception
 * else will reinforce army on the country specified and reduce reinforced army number
 * @param player
 * @param country
 * @param armNum
 */
public void reinforceArmy(Player player, String country, int armNum){

    if(armNum < 0 || armNum > reinforcedArmies){
        throw new ReinforceParsingException("the number is less than 0 or larger than the number of reinforced solider you have");
    }

    if(notOccupiedByPlayer(player, country)){
        throw new ReinforceParsingException(country + " does not exist or it does not owned by the current player " + player.getName());
    }

    playerService.reinforceArmy(player, country, armNum);
    reinforcedArmies -= armNum;
    phaseView.displayMessage( string: "Now, the left reinforced army is: " + reinforcedArmies);


    if(isReinforceOver()){
        playerService.getMapService().setState(GameState.ATTACK);
        isExchangeCardOver = false;
        return;
    }

}
```

5. remove logic relating fortify() from fortifyController to model Player.class. The fortifyGameController is used to check the validity of the command

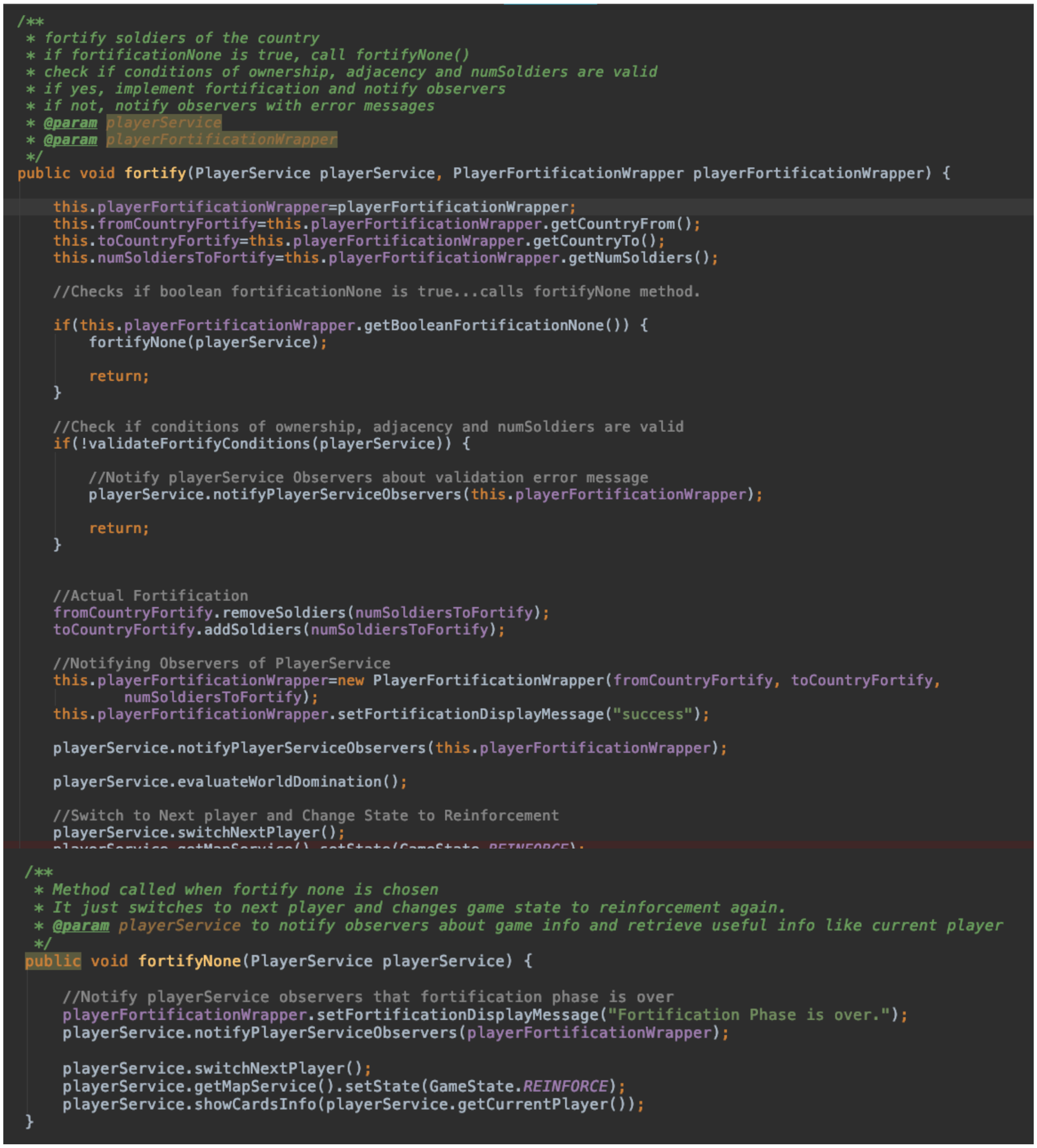
At build 1, logic of fortify and command validation are all in FortifyGameController



```
/**
 * After validation comes fortifying
 * show before and after fortification information of country and soldiers
 * check validation criteria first
 */
public void fortify() {

    validateConditions();

    if(this.boolValidationMet) {

        view.displayMessage( string: "Before Fortification: "+fromCountry.getCountryName()+":"+
                fromCountry.getSoldiers()+" , "+
                toCountry.getCountryName()+":"+toCountry.getSoldiers());

        toCountry.addSoldiers(numSoldiers);
        fromCountry.removeSoldiers(numSoldiers);

        view.displayMessage( string: "After Fortification: "+fromCountry.getCountryName()+":"+
                fromCountry.getSoldiers()+" , "+
                toCountry.getCountryName()+":"+toCountry.getSoldiers());

        this.boolFortificationPhaseOver.set(true);

        this.mapService.setState(GameState.REINFORCE);

    }
}
```

At build 2, we move logic of fortify to model named Player while keeping command validation in FortifyGameController

```
/**
 * store player information
 */
public class Player{
```

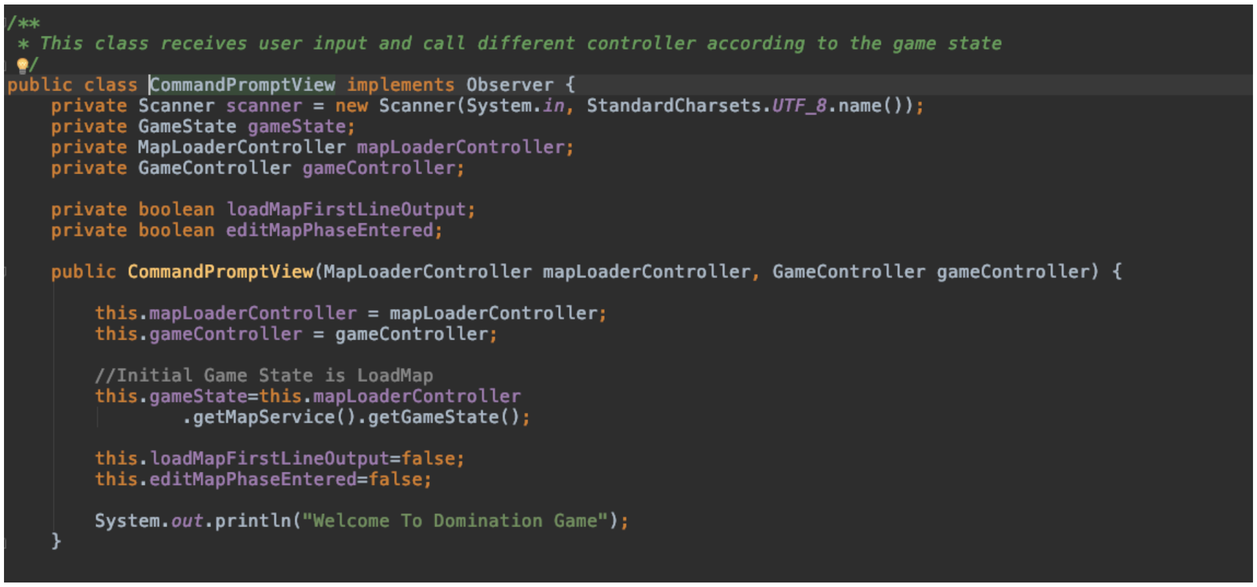


```java
/**
 * fortify soldiers of the country
 * if fortificationNone is true, call fortifyNone()
 * check if conditions of ownership, adjacency and numSoldiers are valid
 * if yes, implement fortification and notify observers
 * if not, notify observers with error messages
 * @param playerService
 * @param playerFortificationWrapper
 */
public void fortify(PlayerService playerService, PlayerFortificationWrapper playerFortificationWrapper) {

    this.playerFortificationWrapper=playerFortificationWrapper;
    this.fromCountryFortify=this.playerFortificationWrapper.getCountryFrom();
    this.toCountryFortify=this.playerFortificationWrapper.getCountryTo();
    this.numSoldiersToFortify=this.playerFortificationWrapper.getNumSoldiers();

    //Checks if boolean fortificationNone is true...calls fortifyNone method.

    if(this.playerFortificationWrapper.getBooleanFortificationNone()) {
        fortifyNone(playerService);

        return;
    }

    //Check if conditions of ownership, adjacency and numSoldiers are valid
    if(!validateFortifyConditions(playerService)) {

        //Notify playerService Observers about validation error message
        playerService.notifyPlayerServiceObservers(this.playerFortificationWrapper);

        return;
    }


    //Actual Fortification
    fromCountryFortify.removeSoldiers(numSoldiersToFortify);
    toCountryFortify.addSoldiers(numSoldiersToFortify);

    //Notifying Observers of PlayerService
    this.playerFortificationWrapper=new PlayerFortificationWrapper(fromCountryFortify, toCountryFortify,
            numSoldiersToFortify);
    this.playerFortificationWrapper.setFortificationDisplayMessage("success");

    playerService.notifyPlayerServiceObservers(this.playerFortificationWrapper);

    playerService.evaluateWorldDomination();

    //Switch to Next player and Change State to Reinforcement
    playerService.switchNextPlayer();
    playerService.getMapService().setState(GameState.REINFORCE);
```

```java
 /**
  * Method called when fortify none is chosen
  * It just switches to next player and changes game state to reinforcement again.
  * @param playerService to notify observers about game info and retrieve useful info like current player
  */
 public void fortifyNone(PlayerService playerService) {

     //Notify playerService observers that fortification phase is over
     playerFortificationWrapper.setFortificationDisplayMessage("Fortification Phase is over.");
     playerService.notifyPlayerServiceObservers(playerFortificationWrapper);

     playerService.switchNextPlayer();
     playerService.getMapService().setState(GameState.REINFORCE);
     playerService.showCardsInfo(playerService.getCurrentPlayer());
 }
```

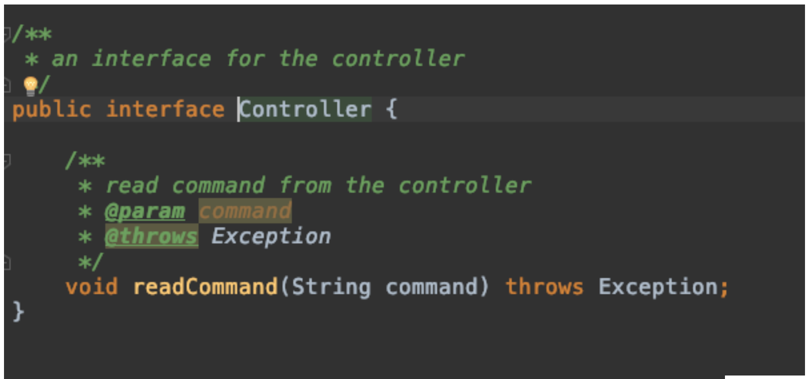
6. add GameView and Controller as the interface.

At build 1, we have only CommandPromptView to receive users' commands, and send it to different controllers according to the game state.

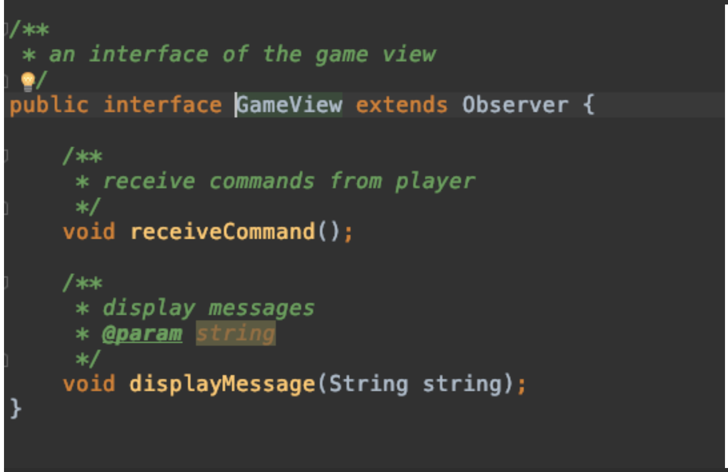


```java
/**
 * This class receives user input and call different controller according to the game state
 */
public class CommandPromptView implements Observer {
    private Scanner scanner = new Scanner(System.in, StandardCharsets.UTF_8.name());
    private GameState gameState;
    private MapLoaderController mapLoaderController;
    private GameController gameController;

    private boolean loadMapFirstLineOutput;
    private boolean editMapPhaseEntered;

    public CommandPromptView(MapLoaderController mapLoaderController, GameController gameController) {

        this.mapLoaderController = mapLoaderController;
        this.gameController = gameController;

        //Initial Game State is LoadMap
        this.gameState=this.mapLoaderController
                .getMapService().getGameState();

        this.loadMapFirstLineOutput=false;
        this.editMapPhaseEntered=false;

        System.out.println("Welcome To Domination Game");
    }
```

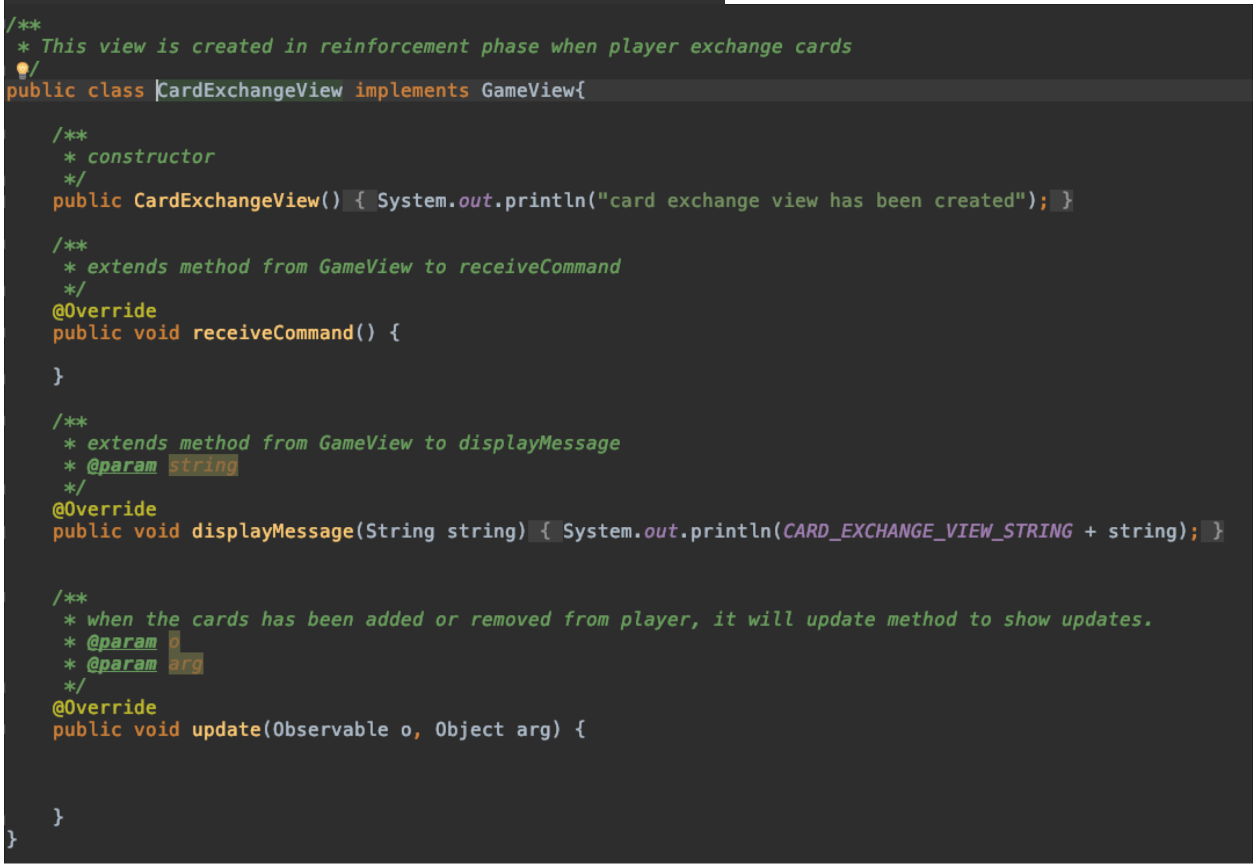
At build 2, we have three different views, phaseView, dominationView and exchangeCardView. We make these three views implement GameViewInterface. This would make the system more flexible as we could replace these views in the future. The same case for the controllers, we would be able to replace these controllers easily in the future based on changing requirements.



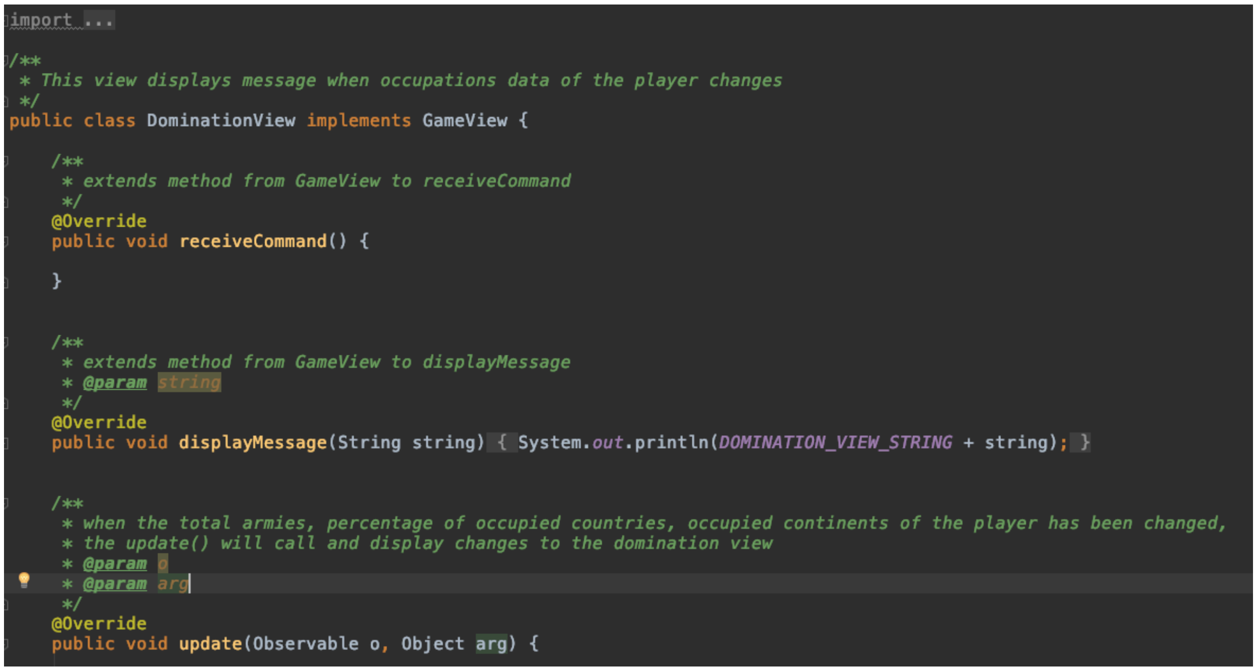
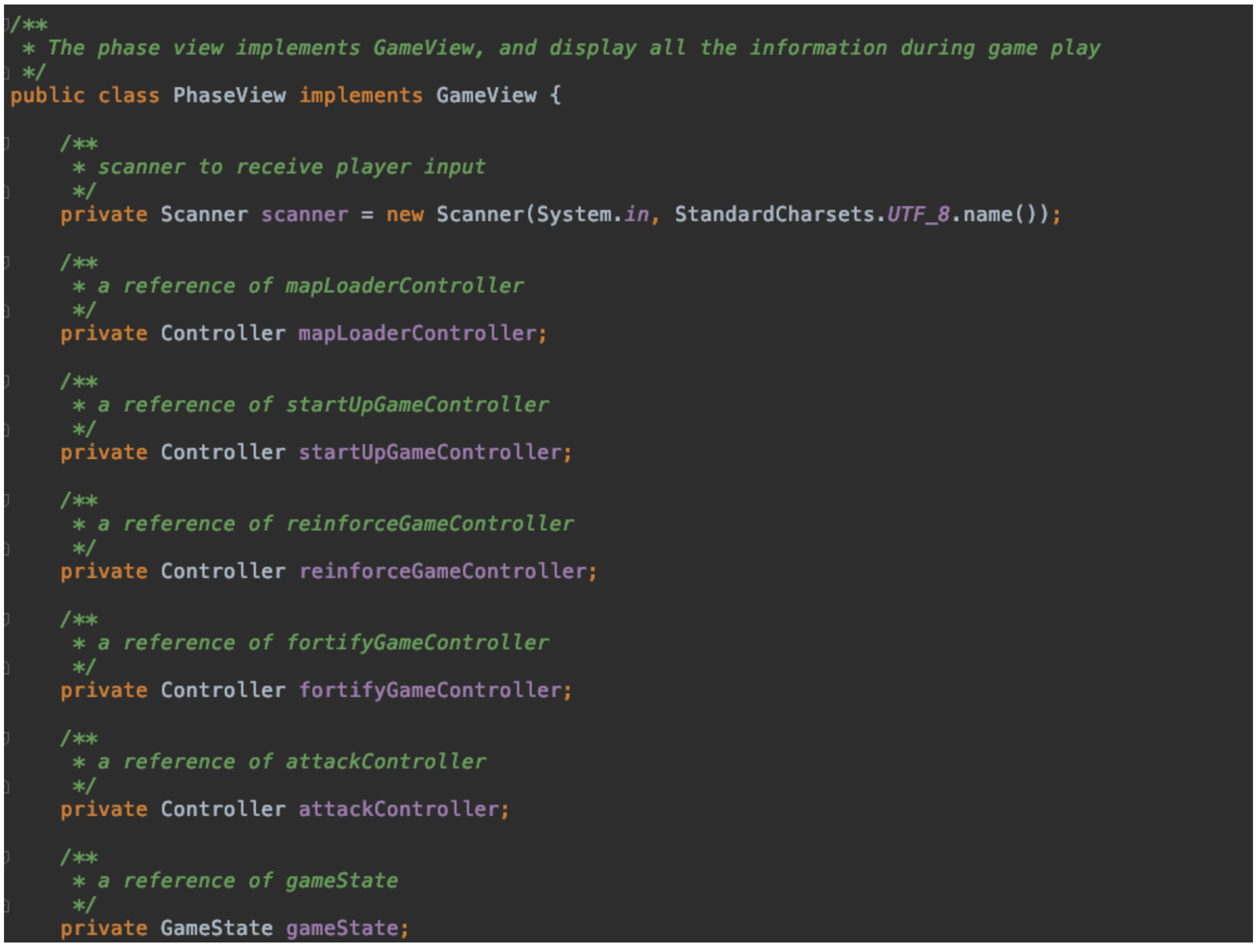
```java
/**
 * an interface for the controller
 */
public interface Controller {

    /**
     * read command from the controller
     * @param command
     * @throws Exception
     */
    void readCommand(String command) throws Exception;
}
```



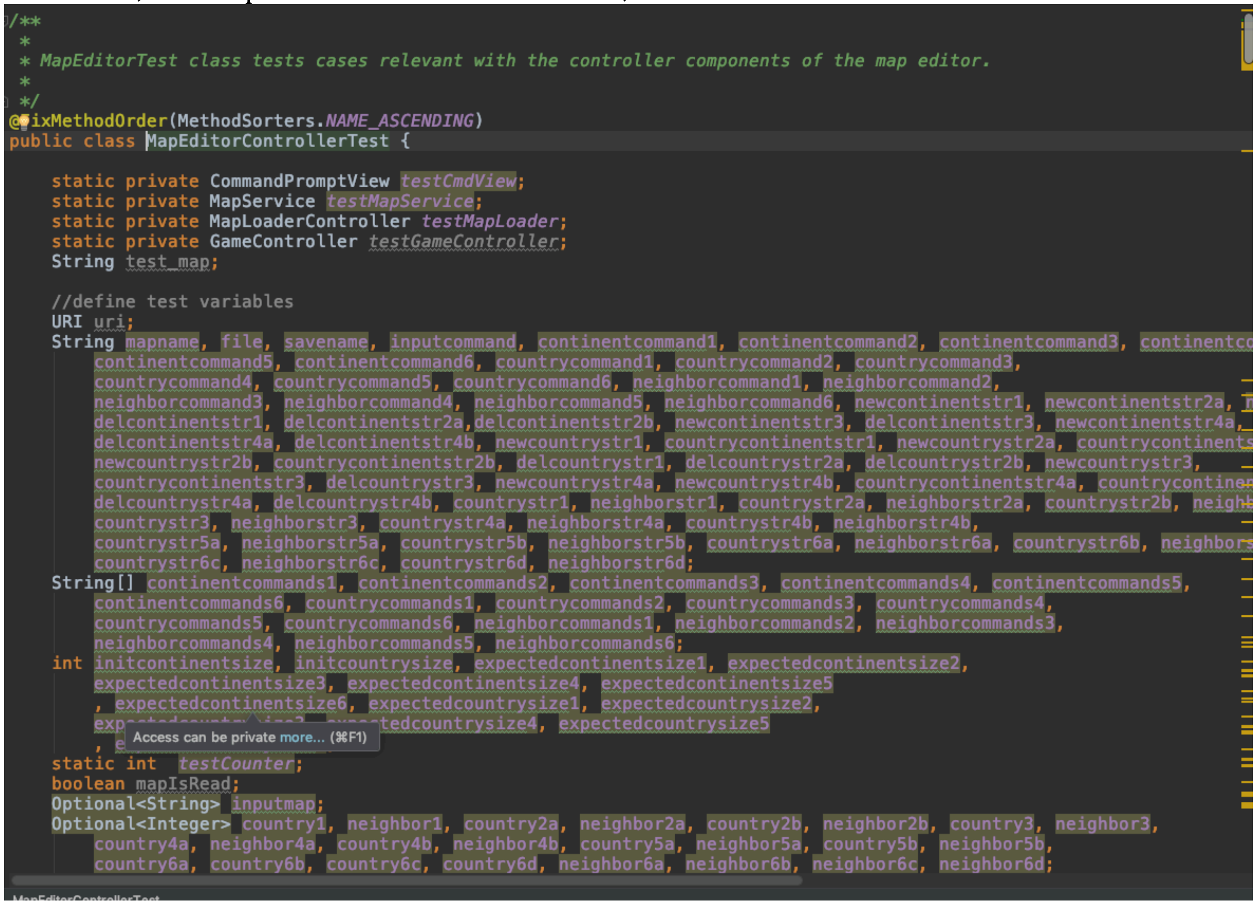
```java
/**
 * an interface of the game view
 */
public interface GameView extends Observer {

    /**
     * receive commands from player
     */
    void receiveCommand();

    /**
     * display messages
     * @param string
     */
    void displayMessage(String string);
}
```



```java
/**
 * This view is created in reinforcement phase when player exchange cards
 */
public class CardExchangeView implements GameView{

    /**
     * constructor
     */
    public CardExchangeView() { System.out.println("card exchange view has been created"); }

    /**
     * extends method from GameView to receiveCommand
     */
    @Override
    public void receiveCommand() {

    }

    /**
     * extends method from GameView to displayMessage
     * @param string
     */
    @Override
    public void displayMessage(String string) { System.out.println(CARD_EXCHANGE_VIEW_STRING + string); }


    /**
     * when the cards has been added or removed from player, it will update method to show updates.
     * @param o
     * @param arg
     */
    @Override
    public void update(Observable o, Object arg) {



    }
}
```

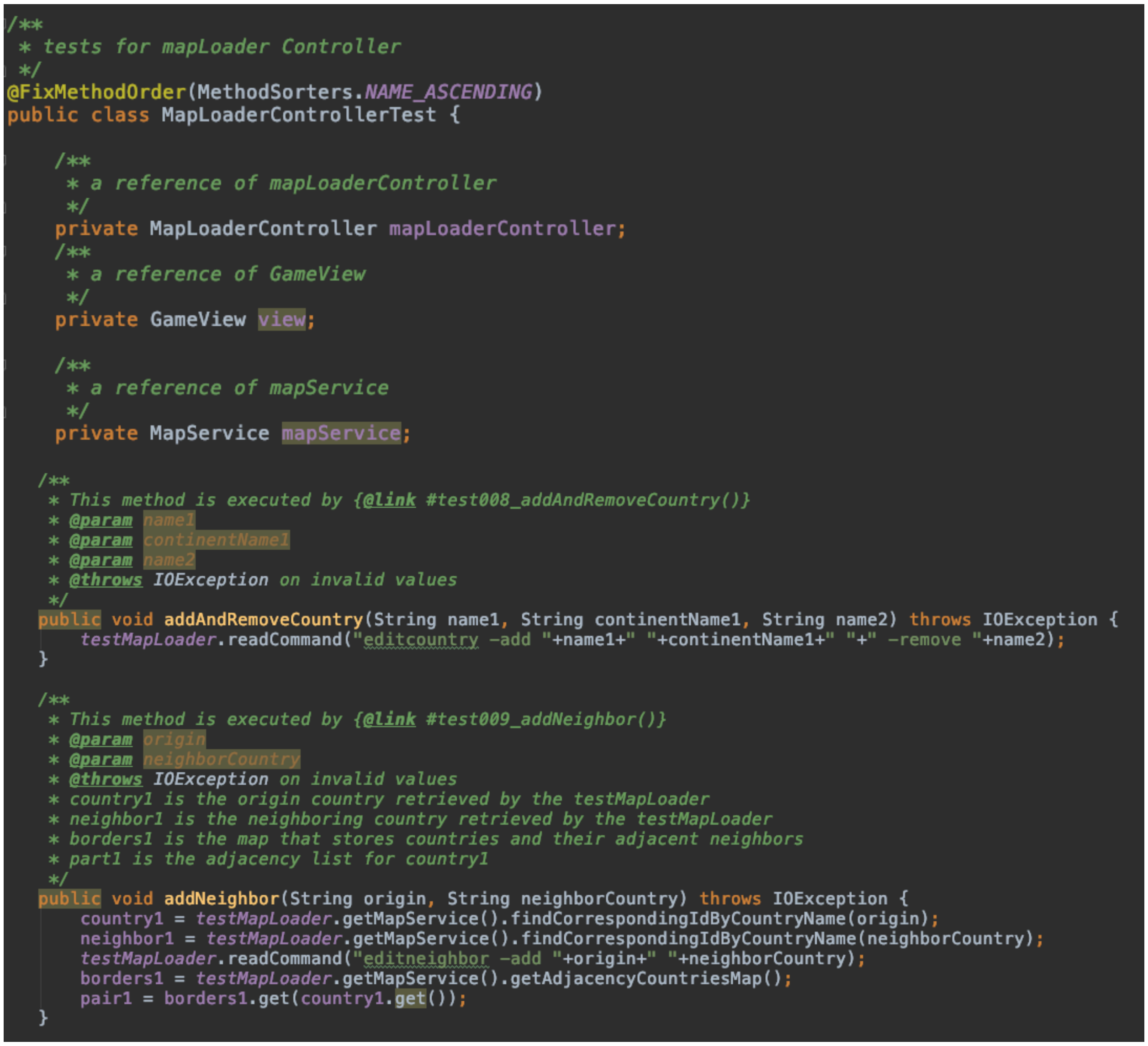

```java
import ...

/**
 * This view displays message when occupations data of the player changes
 */
public class DominationView implements GameView {

    /**
     * extends method from GameView to receiveCommand
     */
    @Override
    public void receiveCommand() {

    }


    /**
     * extends method from GameView to displayMessage
     * @param string
     */
    @Override
    public void displayMessage(String string) { System.out.println(DOMINATION_VIEW_STRING + string); }


    /**
     * when the total armies, percentage of occupied countries, occupied continents of the player has been changed,
     * the update() will call and display changes to the domination view
     * @param o
     * @param arg
     */
    @Override
    public void update(Observable o, Object arg) {
```

```java
/**
 * The phase view implements GameView, and display all the information during game play
 */
public class PhaseView implements GameView {

    /**
     * scanner to receive player input
     */
    private Scanner scanner = new Scanner(System.in, StandardCharsets.UTF_8.name());

    /**
     * a reference of mapLoaderController
     */
    private Controller mapLoaderController;

    /**
     * a reference of startUpGameController
     */
    private Controller startUpGameController;

    /**
     * a reference of reinforceGameController
     */
    private Controller reinforceGameController;

    /**
     * a reference of fortifyGameController
     */
    private Controller fortifyGameController;

    /**
     * a reference of attackController
     */
    private Controller attackController;

    /**
     * a reference of gameState
     */
    private GameState gameState;
```
7. refactor mapLoaderTests including extracting logics and parameters in different methods to make tests more readable and understandable.

At build 1, the mapLoaderTest is hard to read, understand and maintain.

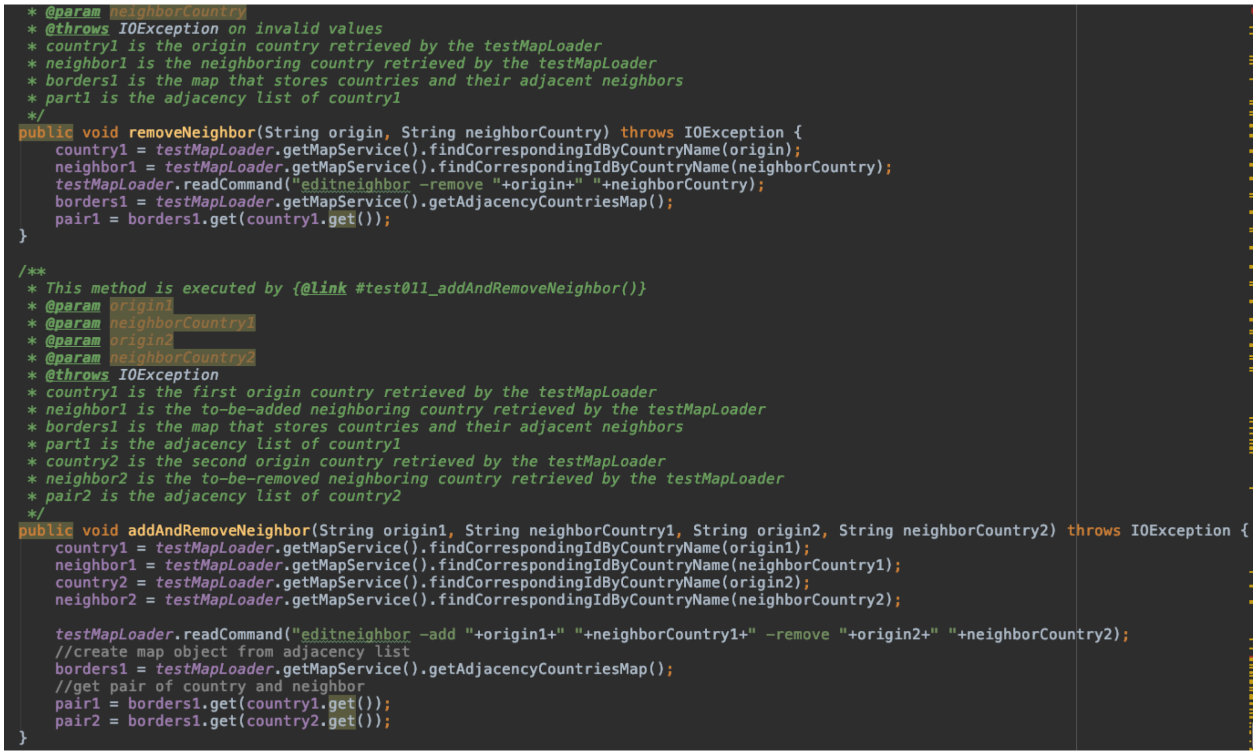


```
/**
 *
 * MapEditorTest class tests cases relevant with the controller components of the map editor.
 *
 */
@ixMethodOrder(MethodSorters.NAME_ASCENDING)
public class MapEditorControllerTest {

    static private CommandPromptView testCmdView;
    static private MapService testMapService;
    static private MapLoaderController testMapLoader;
    static private GameController testGameController;
    String test_map;

    //define test variables
    URI uri;
    String mapname, file, savename, inputcommand, continentcommand1, continentcommand2, continentcommand3, continentco
        continentcommand5, continentcommand6, countrycommand1, countrycommand2, countrycommand3,
        countrycommand4, countrycommand5, countrycommand6, neighborcommand1, neighborcommand2,
        neighborcommand3, neighborcommand4, neighborcommand5, neighborcommand6, newcontinentstr1, newcontinentstr2a, n
        delcontinentstr1, delcontinentstr2a,delcontinentstr2b, newcontinentstr3, delcontinentstr3, newcontinentstr4a,
        delcontinentstr4a, delcontinentstr4b, newcountrystr1, countrycontinentstr1, newcountrystr2a, countrycontinents
        newcountrystr2b, countrycontinentstr2b, delcountrystr1, delcountrystr2a, delcountrystr2b, newcountrystr3,
        countrycontinentstr3, delcountrystr3, newcountrystr4a, newcountrystr4b, countrycontinentstr4a, countrycontine
        delcountrystr4a, delcountrystr4b, countrystr1, neighborstr1, countrystr2a, neighborstr2a, countrystr2b, neigh
        countrystr3, neighborstr3, countrystr4a, neighborstr4a, countrystr4b, neighborstr4b,
        countrystr5a, neighborstr5a, countrystr5b, neighborstr5b, countrystr6a, neighborstr6a, countrystr6b, neighbor
        countrystr6c, neighborstr6c, countrystr6d, neighborstr6d;
    String[] continentcommands1, continentcommands2, continentcommands3, continentcommands4, continentcommands5,
        continentcommands6, countrycommands1, countrycommands2, countrycommands3, countrycommands4,
        countrycommands5, countrycommands6, neighborcommands1, neighborcommands2, neighborcommands3,
        neighborcommands4, neighborcommands5, neighborcommands6;
    int initcontinentsize, initcountrysize, expectedcontinentsize1, expectedcontinentsize2,
        expectedcontinentsize3, expectedcontinentsize4, expectedcontinentsize5
        , expectedcontinentsize6, expectedcountrysize1, expectedcountrysize2,
        exp[illegible]tedcountrysize4, expectedcountrysize5
        , e[illegible]
    static int  testCounter;
    boolean mapIsRead;
    Optional<String> inputmap;
    Optional<Integer> country1, neighbor1, country2a, neighbor2a, country2b, neighbor2b, country3, neighbor3,
        country4a, neighbor4a, country4b, neighbor4b, country5a, neighbor5a, country5b, neighbor5b,
        country6a, country6b, country6c, country6d, neighbor6a, neighbor6b, neighbor6c, neighbor6d;
```

8. At build 2, we extract logics and parameters in different methods to make tests more readable and understandable.



```java
/**
 * tests for mapLoader Controller
 */
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class MapLoaderControllerTest {

    /**
     * a reference of mapLoaderController
     */
    private MapLoaderController mapLoaderController;
    /**
     * a reference of GameView
     */
    private GameView view;

    /**
     * a reference of mapService
     */
    private MapService mapService;

   /**
    * This method is executed by {@link #test008_addAndRemoveCountry()}
    * @param name1
    * @param continentName1
    * @param name2
    * @throws IOException on invalid values
    */
   public void addAndRemoveCountry(String name1, String continentName1, String name2) throws IOException {
       testMapLoader.readCommand("editcountry -add "+name1+" "+continentName1+" "+" -remove "+name2);
   }

   /**
    * This method is executed by {@link #test009_addNeighbor()}
    * @param origin
    * @param neighborCountry
    * @throws IOException on invalid values
    * country1 is the origin country retrieved by the testMapLoader
    * neighbor1 is the neighboring country retrieved by the testMapLoader
    * borders1 is the map that stores countries and their adjacent neighbors
    * part1 is the adjacency list for country1
    */
   public void addNeighbor(String origin, String neighborCountry) throws IOException {
       country1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin);
       neighbor1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry);
       testMapLoader.readCommand("editneighbor -add "+origin+" "+neighborCountry);
       borders1 = testMapLoader.getMapService().getAdjacencyCountriesMap();
       pair1 = borders1.get(country1.get());
   }
```



```
 * @param neighborCountry
 * @throws IOException on invalid values
 * country1 is the origin country retrieved by the testMapLoader
 * neighbor1 is the neighboring country retrieved by the testMapLoader
 * borders1 is the map that stores countries and their adjacent neighbors
 * part1 is the adjacency list of country1
 */
public void removeNeighbor(String origin, String neighborCountry) throws IOException {
    country1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin);
    neighbor1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry);
    testMapLoader.readCommand("editneighbor -remove "+origin+" "+neighborCountry);
    borders1 = testMapLoader.getMapService().getAdjacencyCountriesMap();
    pair1 = borders1.get(country1.get());
}

/**
 * This method is executed by {@link #test011_addAndRemoveNeighbor()}
 * @param origin1
 * @param neighborCountry1
 * @param origin2
 * @param neighborCountry2
 * @throws IOException
 * country1 is the first origin country retrieved by the testMapLoader
 * neighbor1 is the to-be-added neighboring country retrieved by the testMapLoader
 * borders1 is the map that stores countries and their adjacent neighbors
 * part1 is the adjacency list of country1
 * country2 is the second origin country retrieved by the testMapLoader
 * neighbor2 is the to-be-removed neighboring country retrieved by the testMapLoader
 * pair2 is the adjacency list of country2
 */
public void addAndRemoveNeighbor(String origin1, String neighborCountry1, String origin2, String neighborCountry2) throws IOException {
    country1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin1);
    neighbor1 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry1);
    country2 = testMapLoader.getMapService().findCorrespondingIdByCountryName(origin2);
    neighbor2 = testMapLoader.getMapService().findCorrespondingIdByCountryName(neighborCountry2);

    testMapLoader.readCommand("editneighbor -add "+origin1+" "+neighborCountry1+" -remove "+origin2+" "+neighborCountry2);
    //create map object from adjacency list
    borders1 = testMapLoader.getMapService().getAdjacencyCountriesMap();
    //get pair of country and neighbor
    pair1 = borders1.get(country1.get());
    pair2 = borders1.get(country2.get());
}
```