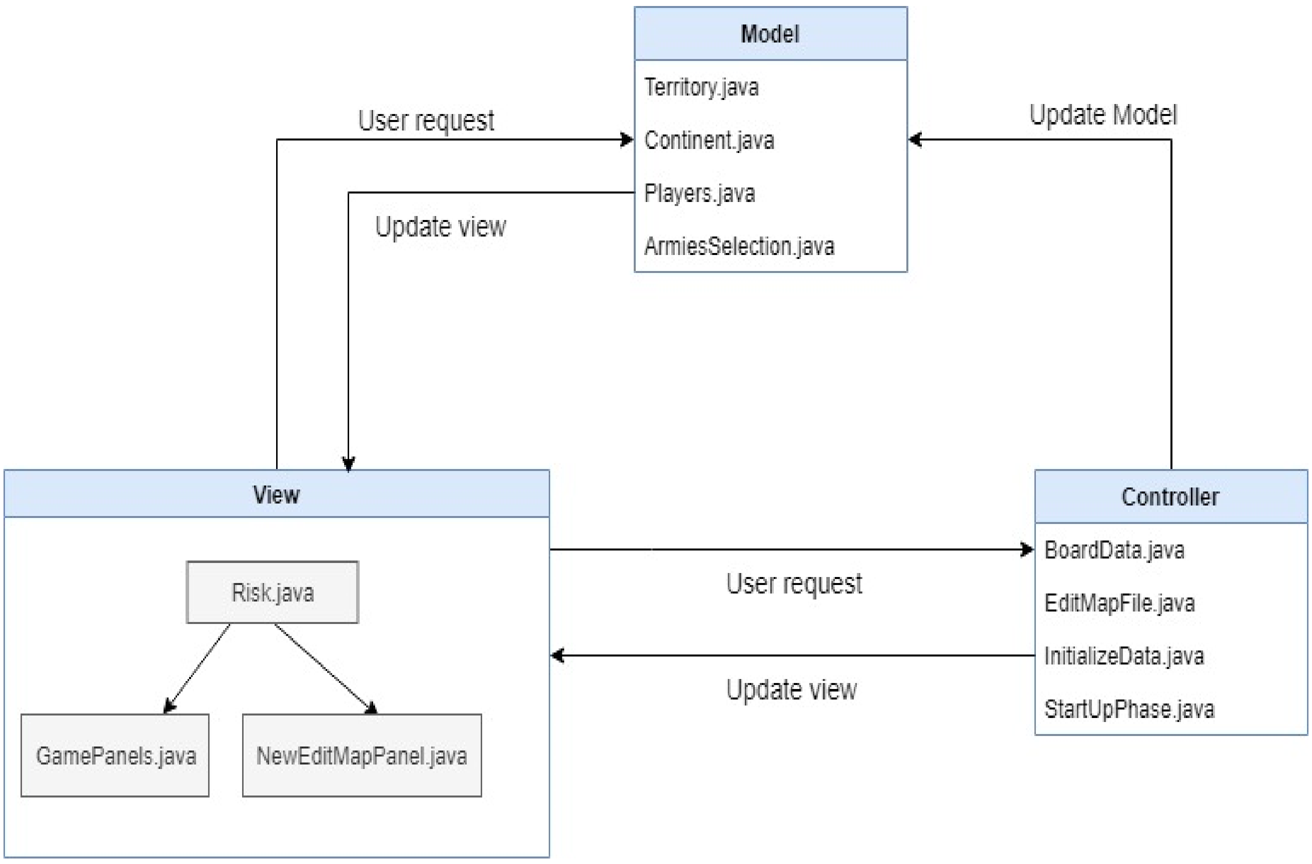
Architectural Design Document

Introduction

Design and develop a RISK Strategy game using suitable software design architecture with iterative software development approach to make a modular design and deliver several working coherent modules in small increments or builds. We implemented a Model View Controller (MVC) architectural design model. It was an effort to use extreme programming key features such as Pair programming, Collective ownership, Coding Standards and many more with the focus of System Metaphor.

Overall Design Architecture



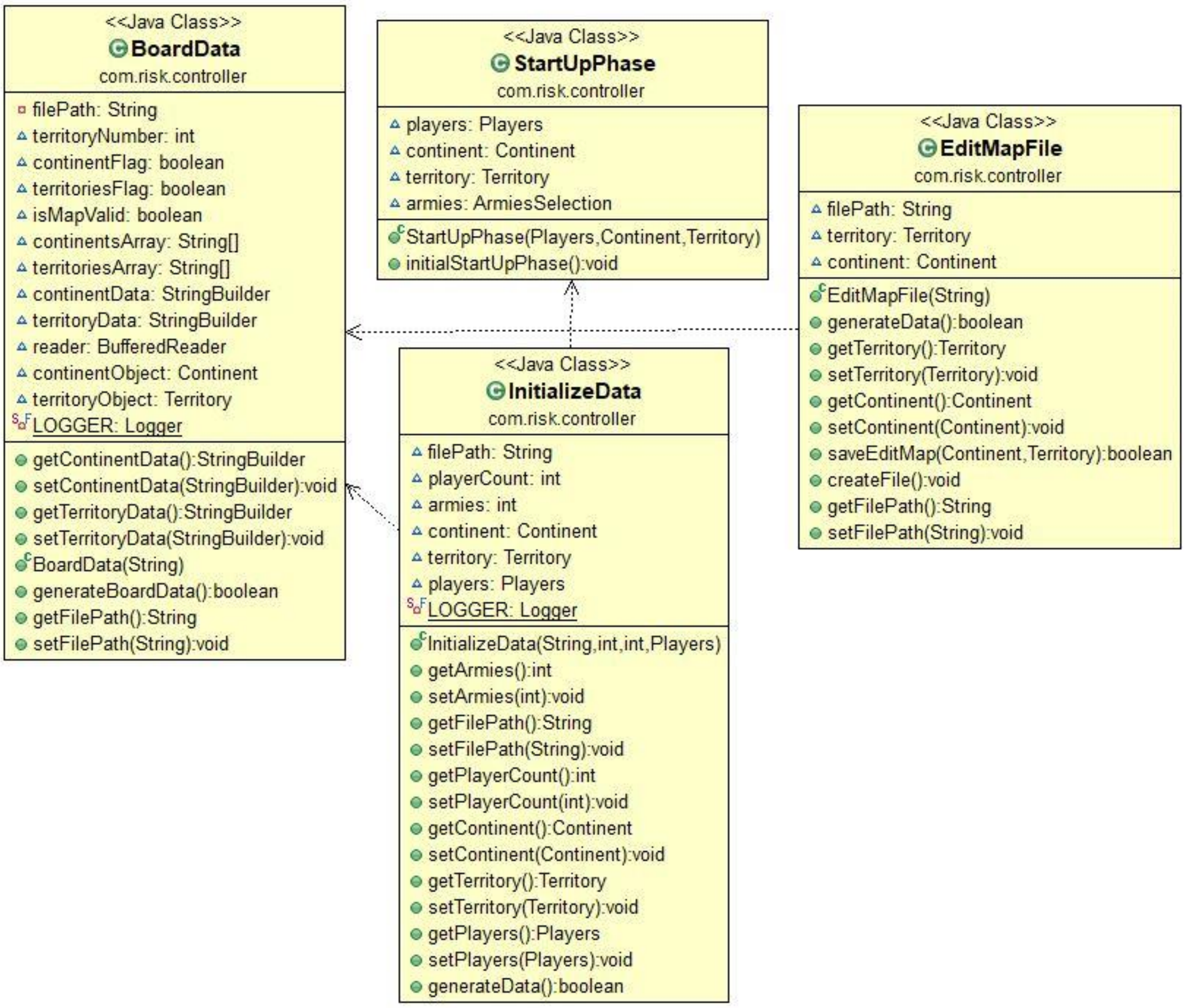
Module descriptions:

Model	**Description**
Terittory.java	This model class contains the Territory data generated from map file.
Players.java	This model class contains the data of Players.
Continent.java	This model class contains the Continent data generated from map file.
ArmiesSelection.java	This model class set the Number of initial armies depending on number of players playing.
View	
Risk.java	A User-Interface to Launch the Game Play, Create Map or Edit Map and quit the Game.
NewEditMapPanel.java	A User-Interface to edit existing map file.
GamePanels.java	A complete User-Interface to play game, select players.
Controller	
BoardData.java	This Controller used to parse map file, validate map file and generate map data from it.
EditMapFile.java	This Controller is used to edit existing map file and save the new edited map file.
Fortification.java	
InitializeData.java	This controller used to initialize data for game play and commence the start-up phase of RISK game.
StartUpPhase.java	Process the start-up phase, assign the territories to players in round-robin fashion.
Exception	
InvalidMapException.java	This class is used to print InvalidMapException message if map is not valid.
Validate	
MapValidator.java	This class is used to validate the map file.
ConnectedGraph.java	Class to check is graph is connected or not?
Observer	
Obsever.java	Observer class of Observer design pattern.
Subject.java	Subject class of Observer design pattern.
Strategy	
Context.java	Context class of Strategy pattern.
Strategy.java	Strategy class of Strategy pattern.
Test	
ControllerTestSuite.java	Test suite class of controller folder.
ReinforcementTest.java	Test all methods of Reinforcement.java class.
StartUpPhaseTest.java	
ArmiesSelectionTest.java	This test class test all methods of ArmiesSelection.java Model class.
ContinentTest.java	This test class test all methods of Continent.java Model class.
ModelsTestSuite.java	Test suite class of models folder.
PlayersTest.java	This test class test all methods of Players.java Model class.
TerritoryTest.java	This test class test all methods of Territory.java Model class.
MapValidatorTest.java	Test all methods of MapValidator.java class.

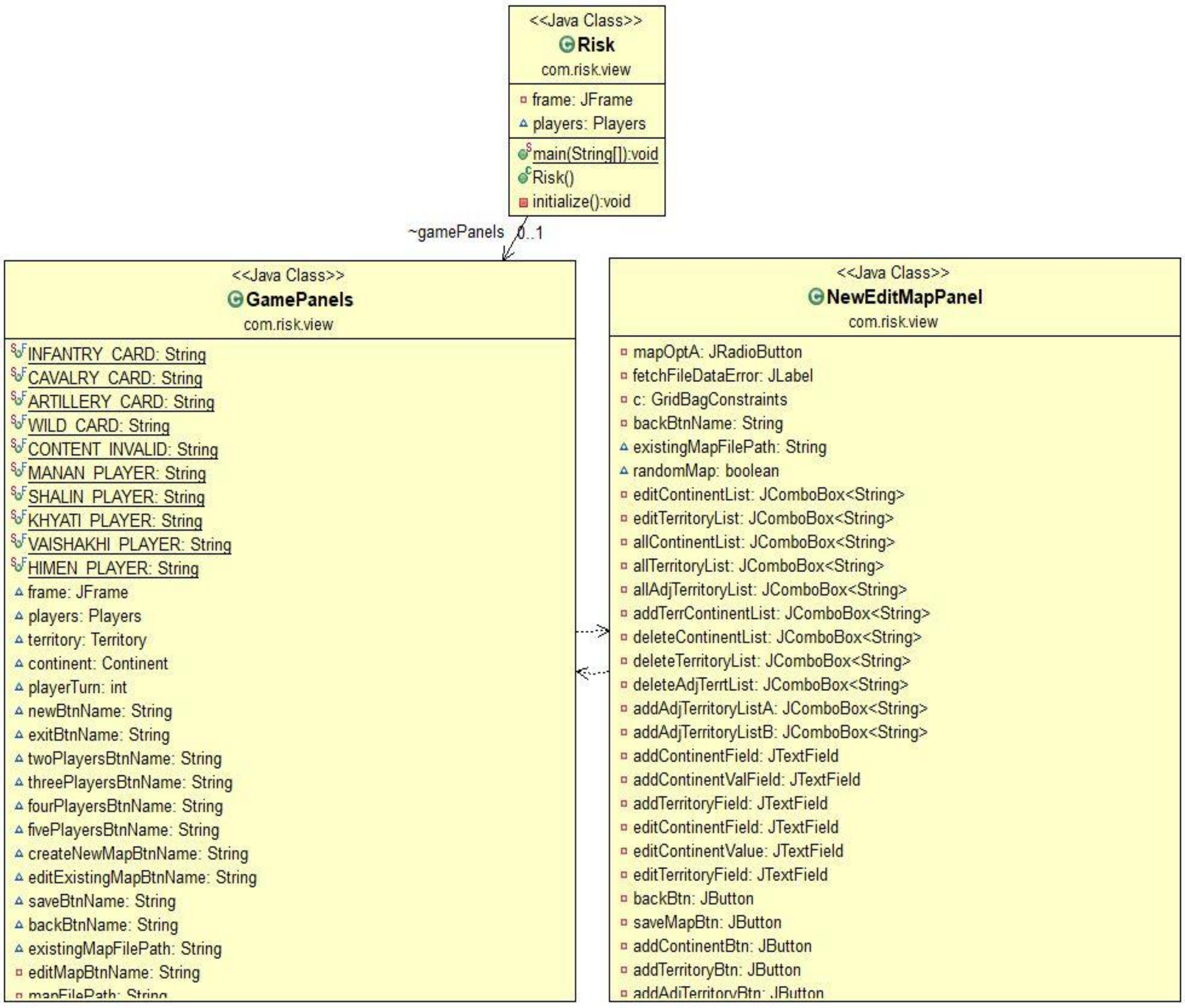
MapValidatorTestIncorrectMap.java	Reads Incorrect maps and test it.
ValidateTestSuite.java	Test suite class of validate folder.
RiskTestSuite.java	Run all test cases together.(Test suite class)

Class Diagrams

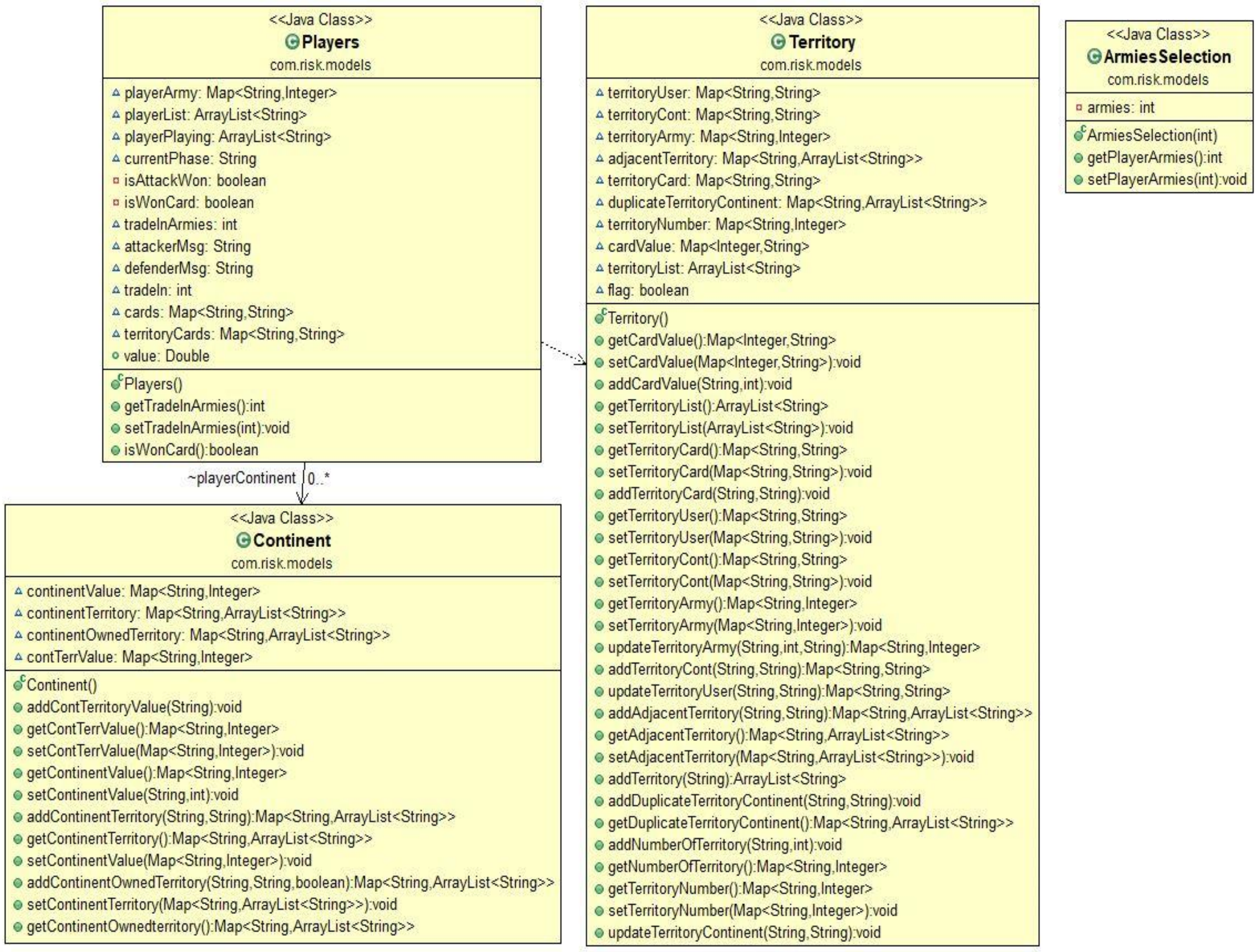
Controller Implementation Class Diagram



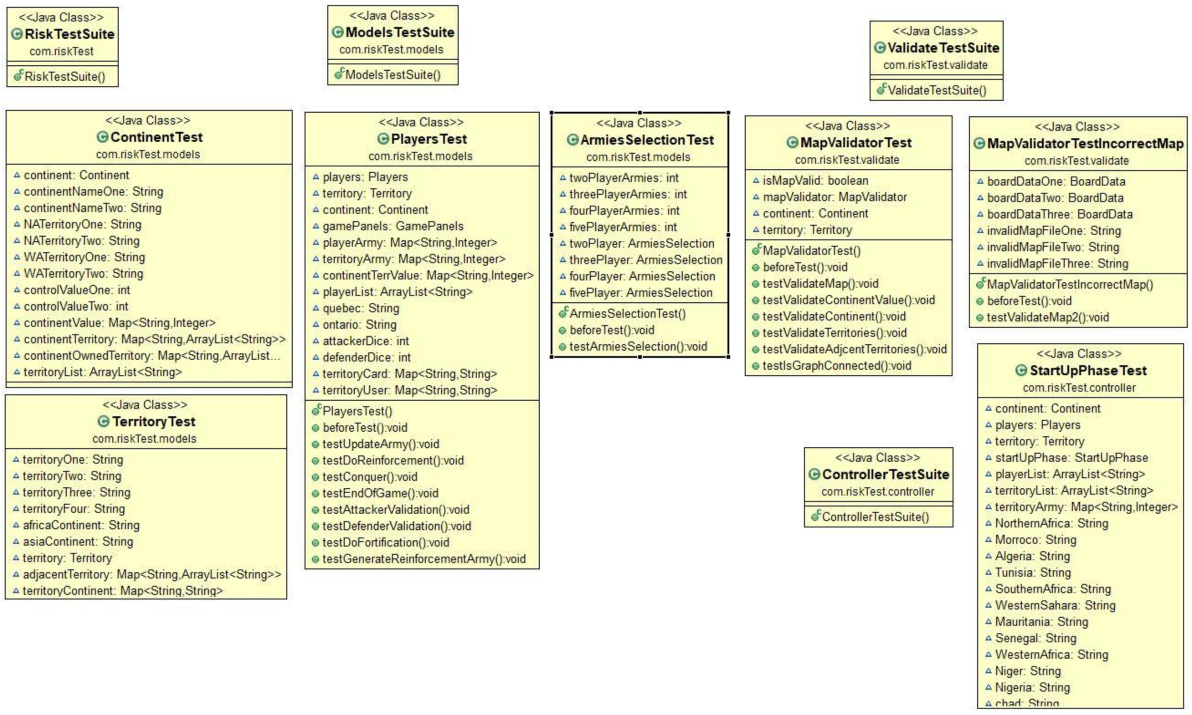
View Implementation Class Diagram


<<Java Class>>
Risk
com.risk.view
frame: JFrame
players: Players
main(String[]):void
Risk()
initialize():void
~gamePanels
0..1
<<Java Class>>
GamePanels
com.risk.view
INFANTRY_CARD: String
CAVALRY_CARD: String
ARTILLERY_CARD: String
WILD_CARD: String
CONTENT_INVALID: String
MANAN_PLAYER: String
SHALIN_PLAYER: String
KHYATI_PLAYER: String
VAISHAKHI_PLAYER: String
HIMEN_PLAYER: String
frame: JFrame
players: Players
territory: Territory
continent: Continent
playerTurn: int
newBtnName: String
exitBtnName: String
twoPlayersBtnName: String
threePlayersBtnName: String
fourPlayersBtnName: String
fivePlayersBtnName: String
createNewMapBtnName: String
editExistingMapBtnName: String
saveBtnName: String
backBtnName: String
existingMapFilePath: String
editMapBtnName: String
mapFilePath: String
<<Java Class>>
NewEditMapPanel
com.risk.view
mapOptA: JRadioButton
fetchFileDataError: JLabel
c: GridBagConstraints
backBtnName: String
existingMapFilePath: String
randomMap: boolean
editContinentList: JComboBox<String>
editTerritoryList: JComboBox<String>
allContinentList: JComboBox<String>
allTerritoryList: JComboBox<String>
allAdjTerritoryList: JComboBox<String>
addTerrContinentList: JComboBox<String>
deleteContinentList: JComboBox<String>
deleteTerritoryList: JComboBox<String>
deleteAdjTerrtList: JComboBox<String>
addAdjTerritoryListA: JComboBox<String>
addAdjTerritoryListB: JComboBox<String>
addContinentField: JTextField
addContinentValField: JTextField
addTerritoryField: JTextField
editContinentField: JTextField
editContinentValue: JTextField
editTerritoryField: JTextField
backBtn: JButton
saveMapBtn: JButton
addContinentBtn: JButton
addTerritoryBtn: JButton
addAdjTerritoryBtn: JButton

Model Implementation Class Diagram


<<Java Class>>
Players
com.risk.models
playerArmy: Map<String,Integer>
playerList: ArrayList<String>
playerPlaying: ArrayList<String>
currentPhase: String
isAttackWon: boolean
isWonCard: boolean
tradeInArmies: int
attackerMsg: String
defenderMsg: String
tradeIn: int
cards: Map<String,String>
territoryCards: Map<String,String>
value: Double
Players()
getTradeInArmies():int
setTradeInArmies(int):void
isWonCard():boolean
~playerContinent
0..*
<<Java Class>>
Continent
com.risk.models
continentValue: Map<String,Integer>
continentTerritory: Map<String,ArrayList<String>>
continentOwnedTerritory: Map<String,ArrayList<String>>
contTerrValue: Map<String,Integer>
Continent()
addContTerritoryValue(String):void
getContTerrValue():Map<String,Integer>
setContTerrValue(Map<String,Integer>):void
getContinentValue():Map<String,Integer>
setContinentValue(String,int):void
addContinentTerritory(String,String):Map<String,ArrayList<String>>
getContinentTerritory():Map<String,ArrayList<String>>
setContinentValue(Map<String,Integer>):void
addContinentOwnedTerritory(String,String,boolean):Map<String,ArrayList<String>>
setContinentTerritory(Map<String,ArrayList<String>>):void
getContinentOwnedterritory():Map<String,ArrayList<String>>
<<Java Class>>
Territory
com.risk.models
territoryUser: Map<String,String>
territoryCont: Map<String,String>
territoryArmy: Map<String,Integer>
adjacentTerritory: Map<String,ArrayList<String>>
territoryCard: Map<String,String>
duplicateTerritoryContinent: Map<String,ArrayList<String>>
territoryNumber: Map<String,Integer>
cardValue: Map<Integer,String>
territoryList: ArrayList<String>
flag: boolean
Territory()
getCardValue():Map<Integer,String>
setCardValue(Map<Integer,String>):void
addCardValue(String,int):void
getTerritoryList():ArrayList<String>
setTerritoryList(ArrayList<String>):void
getTerritoryCard():Map<String,String>
setTerritoryCard(Map<String,String>):void
addTerritoryCard(String,String):void
getTerritoryUser():Map<String,String>
setTerritoryUser(Map<String,String>):void
getTerritoryCont():Map<String,String>
setTerritoryCont(Map<String,String>):void
getTerritoryArmy():Map<String,Integer>
setTerritoryArmy(Map<String,Integer>):void
updateTerritoryArmy(String,int,String):Map<String,Integer>
addTerritoryCont(String,String):Map<String,String>
updateTerritoryUser(String,String):Map<String,String>
addAdjacentTerritory(String,String):Map<String,ArrayList<String>>
getAdjacentTerritory():Map<String,ArrayList<String>>
setAdjacentTerritory(Map<String,ArrayList<String>>):void
addTerritory(String):ArrayList<String>
addDuplicateTerritoryContinent(String,String):void
getDuplicateTerritoryContinent():Map<String,ArrayList<String>>
addNumberOfTerritory(String,int):void
getNumberOfTerritory():Map<String,Integer>
getTerritoryNumber():Map<String,Integer>
setTerritoryNumber(Map<String,Integer>):void
updateTerritoryContinent(String,String):void
<<Java Class>>
ArmiesSelection
com.risk.models
armies: int
ArmiesSelection(int)
getPlayerArmies():int
setPlayerArmies(int):void

Test Implementation Class Diagram



Rules Followed:

Rules followed of Ultra Board Games : http://www.ultraboardgames.com/risk/game-rules.php