CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

SOEN-6441

Risk game - build 2

Application Design document

Team# 31

Divyaprabha Rajendran	40089282
Chitra Gunasekaran	40107070
Ishpreet Singh	40093666
Yash Pragneshkumar Doshi	40105936
Fareed Tayar	40102053

I. Assignment Objective

To design and build a simple Risk game, A Risk game consists of a connected graph map representing a world map, where each node is a country and each edge represents adjacency between countries. Three to six players can play by placing armies on countries they own, from which they can attack adjacent countries to conquer them. The objective of the game is to conquer all countries on the map.

To deliver an operational version demonstrating a subset of the capacity of your system. This is about demonstrating that the code build is effectively aimed at solving specific project problems or completely implementing specific system features.

II. High level system architecture*

The application consists of below main components and functionalities

A. Game main model components
   i. Graph node: consists of graph data (the country) and links to neighboring graph nodes
   ii. Graph: an abstraction of a graph, used in player, continent and game map types. It provides verification methods for graph connectivity and finding path between two graph nodes. As well as view method for graph nodes.
   iii. Country: encapsulates data related to the country along with needed methods to modify its data. Every country has an observer (using Observer pattern) which is triggered whenever country ownership and/or number of armies exists changed.
   iv. Continent: holder of continent data like name, value of bonus armies, continent graph. As well as view continent method and a utility method to verify if a single user owns all countries belongs to this continent.
   v. Game map: holder for played game map data (continents, reference to map xml file…etc.)
   vi. Player: the player main type, holds players related data like its name, hand (hosting cards), player's graph along with needed methods to modify players related data and calculate number of reinforcement armies for this player
   vii. Players: holder to all players playing the game.

B. Configuration constants: constants representing default number of armies each player should have in startup phase based on number of players playing the game.
C. Exceptions: customized exceptions representing invalid runtime faults related to graph, continent, country, naming conventions, player … etc.
D. Map creation and validation: to read xml map file, build continent and game map objects and validate its data.
E. Game main driver: provides the game main follow, according to risk game rules
   i. start with start-up phase (creation, edit and loading of map xml files), then
   ii. creation of game model objects (continents, game map, players… etc.)
   iii. initial distribution of counties (randomly)
   iv. initial distribution of players armies

v. Play game flow (provides in players turn)
    1. Reinforcement phase in accordance to risk game rules, where for every turn all reinforcement armies are placed on the player owned countries
    2. Attack phase (optional based on user input) according to game rule with all-out option for attack, till end of game.
    3. Fortification phase (optional based on user input), according to risk game rules with valid move of armies (valid path between the two countries in the player graph)

vi. Views (based on observer pattern)
    1. Players world domination view
    2. Phase view (including startup phase)
    3. Card exchange view
    4. Country view to reflect changed in ownership and number of armies.

(*) Refer to first appendix for complete UML representation

III. Used Technology and references

- Standard Java library (JDK 1.8) including java doc tool
- Junit 4
- Brown university coding standard - java style
- Course material in reference to design patterns (Observer Model)
- Bitbucket GIT repository
- Eclipse IDE

IV. Folders and packages structure

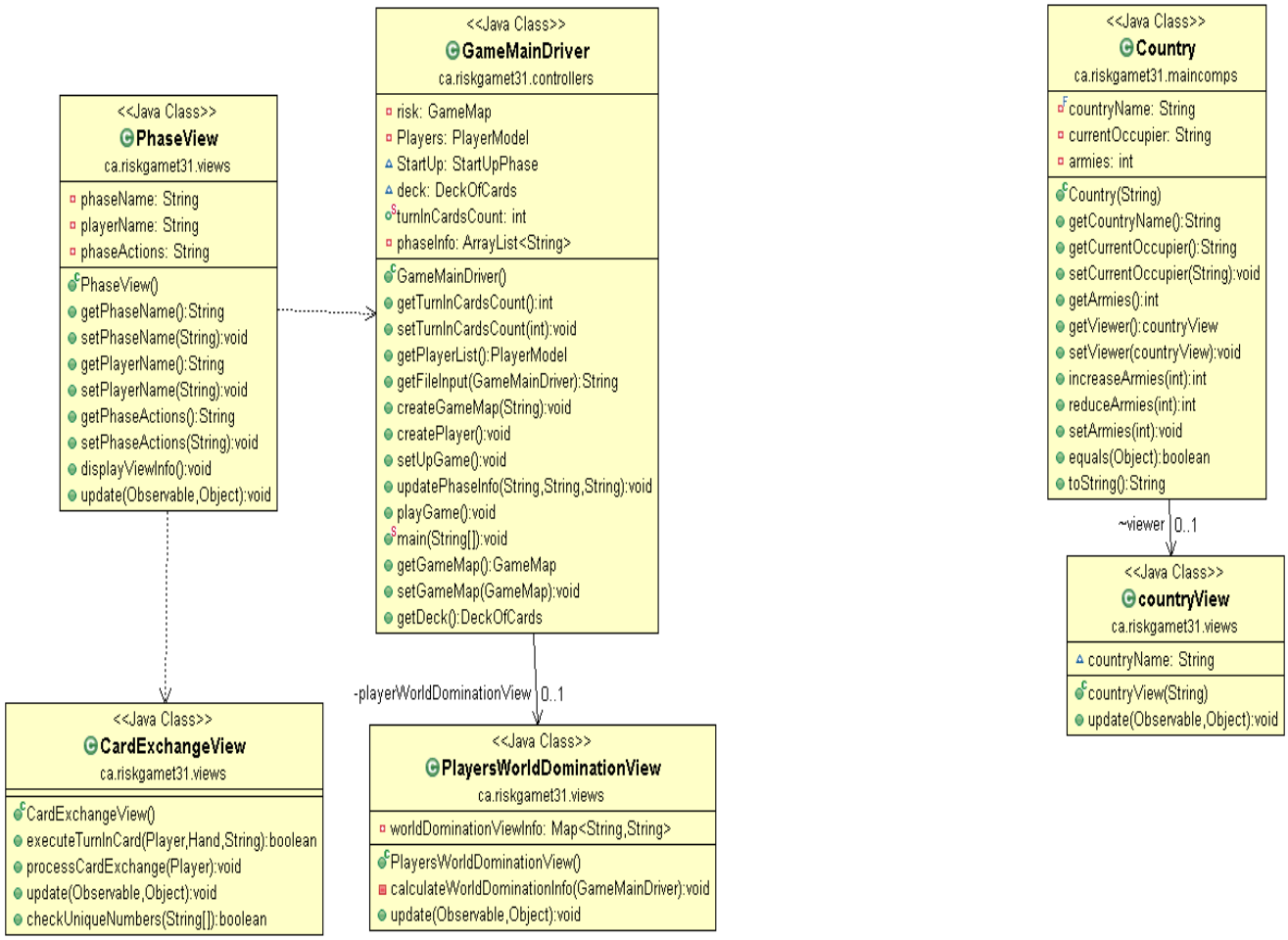
A. Folders structure
    i. src: contains application source code files
        1. /ca
            a. /riskgamet31: contains application source code packages and files
            b. /riskgamet31test: contains Junit test package, suite and classes
    ii. ProjDocs: contains project reference files (i.e. coding style reference, assignment 1 documentation and presentation.
    iii. Risk_JDocs: contains the automatically generated java documentation using jdoc tool.
    iv. Risk_MapData: contains several valid and invalid maps source xml files.

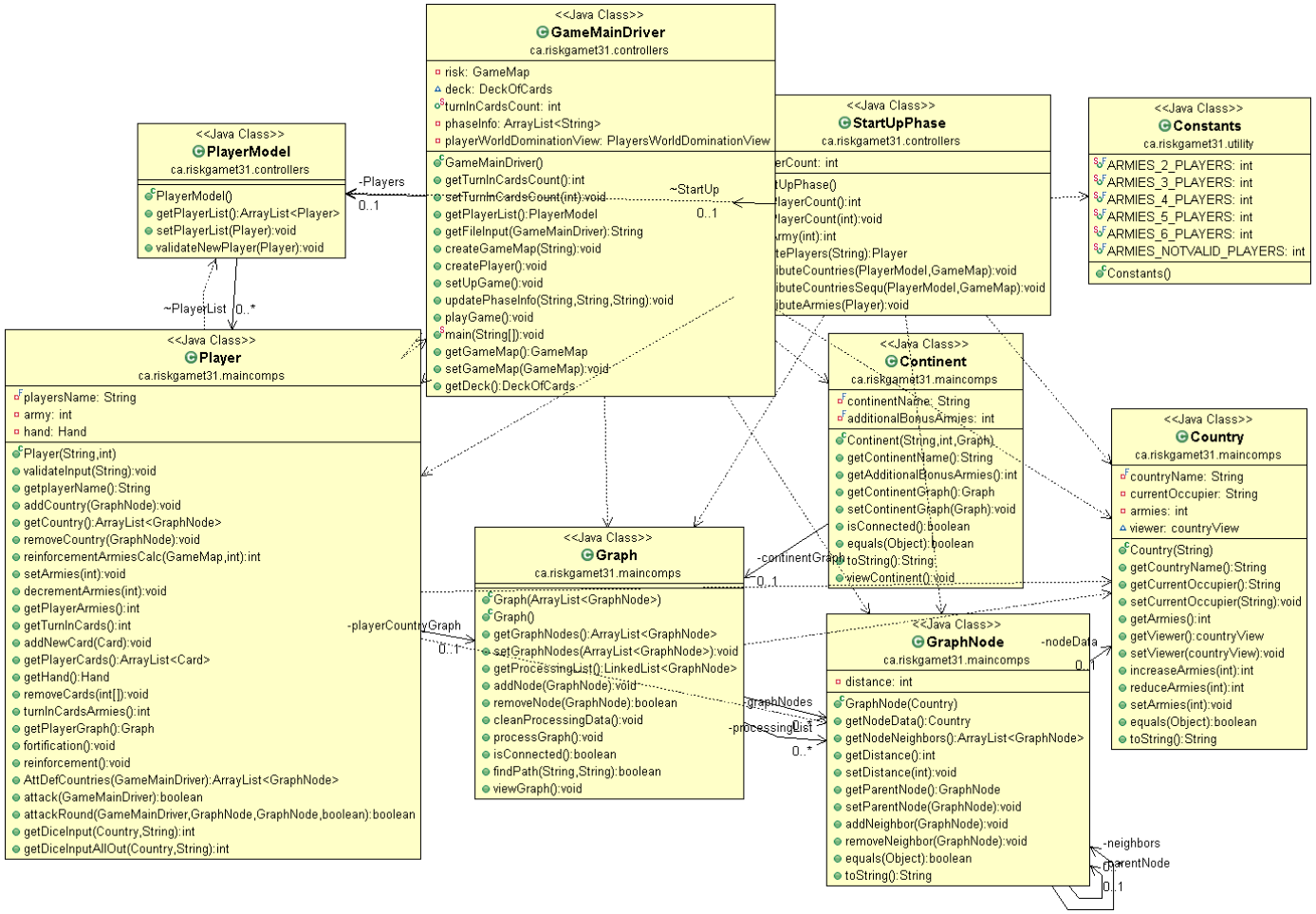
B. Packages structure
    i. ca.riskgamet31.utilitiy: utility classes(constants …etc)
    ii. ca.riskgamet31.controllers: controllers classes
    iii. ca.riskgamet31.exceptions: customized exception classes
    iv. ca.riskgamet31.maincomps: game main model classes
    v. ca.riskgamet31.mapdata: map related classes
    vi. ca.riskgamet31.views: views classes

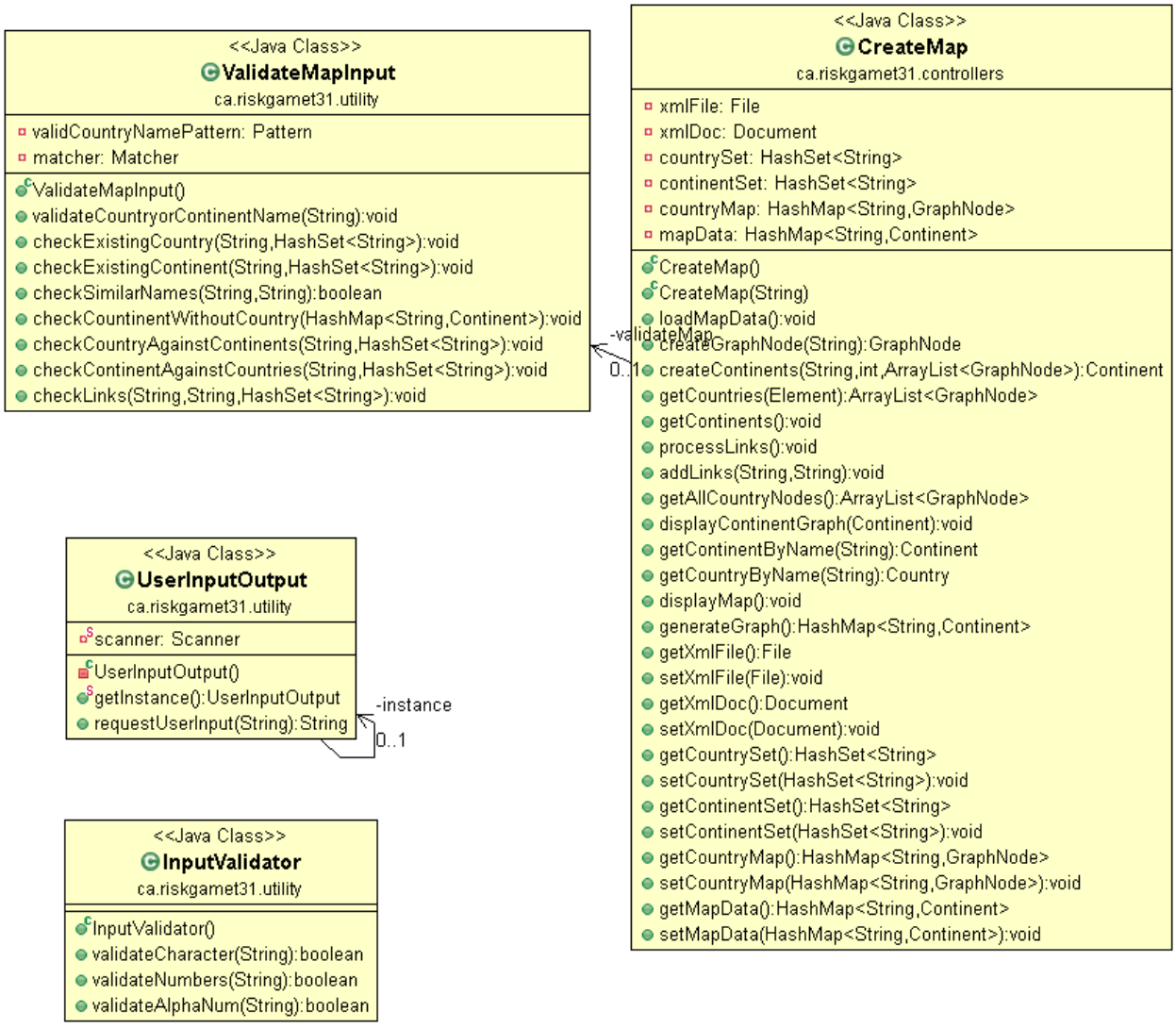
    below contains joint test classes and suites for various packages
    vii. ca.riskgamet31test.maincomps

viii. ca.riskgamet31test.controllers
ix. ca.riskgamet31test.mapdata

v. Appendix I – UML




<<Java Class>>
GameMainDriver
ca.riskgamet31.controllers
risk: GameMap
deck: DeckOfCards
turnInCardsCount: int
phaseInfo: ArrayList<String>
playerWorldDominationView: PlayersWorldDominationView
GameMainDriver()
getTurnInCardsCount():int
setTurnInCardsCount(int):void
getPlayerList():PlayerModel
getFileInput(GameMainDriver):String
createGameMap(String):void
createPlayer():void
setUpGame():void
updatePhaseInfo(String,String,String):void
playGame():void
main(String[]):void
getGameMap():GameMap
setGameMap(GameMap):void
getDeck():DeckOfCards
<<Java Class>>
PlayerModel
ca.riskgamet31.controllers
PlayerModel()
getPlayerList():ArrayList<Player>
setPlayerList(Player):void
validateNewPlayer(Player):void
-Players
0..1
~StartUp
0..1
~PlayerList
0..*
<<Java Class>>
StartUpPhase
ca.riskgamet31.controllers
erCount: int
tUpPhase()
layerCount():int
layerCount(int):void
rmy(int):int
tePlayers(String):Player
ibuteCountries(PlayerModel,GameMap):void
ibuteCountriesSequ(PlayerModel,GameMap):void
ibuteArmies(Player):void
<<Java Class>>
Constants
ca.riskgamet31.utility
ARMIES_2_PLAYERS: int
ARMIES_3_PLAYERS: int
ARMIES_4_PLAYERS: int
ARMIES_5_PLAYERS: int
ARMIES_6_PLAYERS: int
ARMIES_NOTVALID_PLAYERS: int
Constants()
<<Java Class>>
Player
ca.riskgamet31.maincomps
playersName: String
army: int
hand: Hand
Player(String,int)
validateInput(String):void
getplayerName():String
addCountry(GraphNode):void
getCountry():ArrayList<GraphNode>
removeCountry(GraphNode):void
reinforcementArmiesCalc(GameMap,int):int
setArmies(int):void
decrementArmies(int):void
getPlayerArmies():int
getTurnInCards():int
addNewCard(Card):void
getPlayerCards():ArrayList<Card>
getHand():Hand
removeCards(int[]):void
turnInCardsArmies():int
getPlayerGraph():Graph
fortification():void
reinforcement():void
AttDefCountries(GameMainDriver):ArrayList<GraphNode>
attack(GameMainDriver):boolean
attackRound(GameMainDriver,GraphNode,GraphNode,boolean):boolean
getDiceInput(Country,String):int
getDiceInputAllOut(Country,String):int
<<Java Class>>
Continent
ca.riskgamet31.maincomps
continentName: String
additionalBonusArmies: int
Continent(String,int,Graph)
getContinentName():String
getAdditionalBonusArmies():int
getContinentGraph():Graph
setContinentGraph(Graph):void
isConnected():boolean
equals(Object):boolean
toString():String
viewContinent():void
<<Java Class>>
Country
ca.riskgamet31.maincomps
countryName: String
currentOccupier: String
armies: int
viewer: countryView
Country(String)
getCountryName():String
getCurrentOccupier():String
setCurrentOccupier(String):void
getArmies():int
getViewer():countryView
setViewer(countryView):void
increaseArmies(int):int
reduceArmies(int):int
setArmies(int):void
equals(Object):boolean
toString():String
<<Java Class>>
Graph
ca.riskgamet31.maincomps
Graph(ArrayList<GraphNode>)
Graph()
getGraphNodes():ArrayList<GraphNode>
setGraphNodes(ArrayList<GraphNode>):void
getProcessingList():LinkedList<GraphNode>
addNode(GraphNode):void
removeNode(GraphNode):boolean
cleanProcessingData():void
processGraph():void
isConnected():boolean
findPath(String,String):boolean
viewGraph():void
-continentGraph
0..1
-playerCountryGraph
0..1
-graphNodes
-processingList
0..*
<<Java Class>>
GraphNode
ca.riskgamet31.maincomps
distance: int
GraphNode(Country)
getNodeData():Country
getNodeNeighbors():ArrayList<GraphNode>
getDistance():int
setDistance(int):void
getParentNode():GraphNode
setParentNode(GraphNode):void
addNeighbor(GraphNode):void
removeNeighbor(GraphNode):void
equals(Object):boolean
toString():String
-nodeData
0..1
-neighbors
-parentNode
0..1


<<Java Class>>
ValidateMapInput
ca.riskgamet31.utility
validCountryNamePattern: Pattern
matcher: Matcher
ValidateMapInput()
validateCountryorContinentName(String):void
checkExistingCountry(String,HashSet<String>):void
checkExistingContinent(String,HashSet<String>):void
checkSimilarNames(String,String):boolean
checkCountinentWithoutCountry(HashMap<String,Continent>):void
checkCountryAgainstContinents(String,HashSet<String>):void
checkContinentAgainstCountries(String,HashSet<String>):void
checkLinks(String,String,HashSet<String>):void
<<Java Class>>
CreateMap
ca.riskgamet31.controllers
xmlFile: File
xmlDoc: Document
countrySet: HashSet<String>
continentSet: HashSet<String>
countryMap: HashMap<String,GraphNode>
mapData: HashMap<String,Continent>
CreateMap()
CreateMap(String)
loadMapData():void
createGraphNode(String):GraphNode
createContinents(String,int,ArrayList<GraphNode>):Continent
getCountries(Element):ArrayList<GraphNode>
getContinents():void
processLinks():void
addLinks(String,String):void
getAllCountryNodes():ArrayList<GraphNode>
displayContinentGraph(Continent):void
getContinentByName(String):Continent
getCountryByName(String):Country
displayMap():void
generateGraph():HashMap<String,Continent>
getXmlFile():File
setXmlFile(File):void
getXmlDoc():Document
setXmlDoc(Document):void
getCountrySet():HashSet<String>
setCountrySet(HashSet<String>):void
getContinentSet():HashSet<String>
setContinentSet(HashSet<String>):void
getCountryMap():HashMap<String,GraphNode>
setCountryMap(HashMap<String,GraphNode>):void
getMapData():HashMap<String,Continent>
setMapData(HashMap<String,Continent>):void
-validateMap
0..1
<<Java Class>>
UserInputOutput
ca.riskgamet31.utility
scanner: Scanner
UserInputOutput()
getInstance():UserInputOutput
requestUserInput(String):String
-instance
0..1
<<Java Class>>
InputValidator
ca.riskgamet31.utility
InputValidator()
validateCharacter(String):boolean
validateNumbers(String):boolean
validateAlphaNum(String):boolean

VI. Appendix II – Refactoring list

Class	Method/code part	Refactor description	Priority/status	TM
UserInputOutput	New singleton for user input, phased approach towards MVC	Retrieve and validate user input	1	Fareed
Player	Reinforcement / fortification (drop)	Take out user input	Dropped	Yash
Country	Constructor	Remove auto view link	2	Fareed
Country	Constructor	Remove validate map input (should be prior constructing the country)	3	Fareed
Graph	Viewgraph	Double check graph could be displayed in better way	Dropped	Fareed
Continent	Data member/ constructor	Remove validate data, should be checked prior creating object.	4	Fareed
ValidateUserInput	New class provides validation (text only, number only, alphanumeric) no nulls are accepted	New class provides validation	5	Fareed
Player	Remove validate input method	To be replaced with the new class	8	Yash
Player	Fortification, distribute armies, (not needed)	To remove getting user input.	9	Yash
Hand	Review and test with card and deck of cards, add to string method overriding Object,		6	Yash
Consolidate to utility package	To restructure packages	Package restructuring	7	Fareed