SOEN 6471 – Advanced Software Architecture

Professor Peter Rigby

Final Report- Risk Game

Qasim Naushad – 5658624

Prakash Gunasekaran - 6399185

Prabhjot Kaur Sekhon - 6473318

Rajwinder Kaur - 6282490

Harsahiljit Singh - 6405975

Yash Paliwal - 6562566

M1-Risk Game

Qasim Naushad 5658624
Harsahiljit Singh Mehmi 6405975
Prakash Gunasekaran 6399185
Rajwinder Kaur 6282490
Prabhjot Sekhon 6473318
Yash Paliwal 6562566

We have compiled and assessed this project a believe it to be of a reasonable size for a term project.

<Signatures of project members>

Qasim Naushad
Prakash Gunasekaran
Harsahiljit Singh Mehmi
Rajwinder Kaur
Prabhjot Sekhon
Yash Paliwal

Project Description

The project we picked is from Concordia's Last year master's students, it was submitted by a team of 5 people for the course advanced programming. The reason why it interest's most of our team members is because it was created by students themselves, we have a higher chance of finding code smells and architecture faults which we can improve on and learn from and find out the mistakes we make as new programmers.

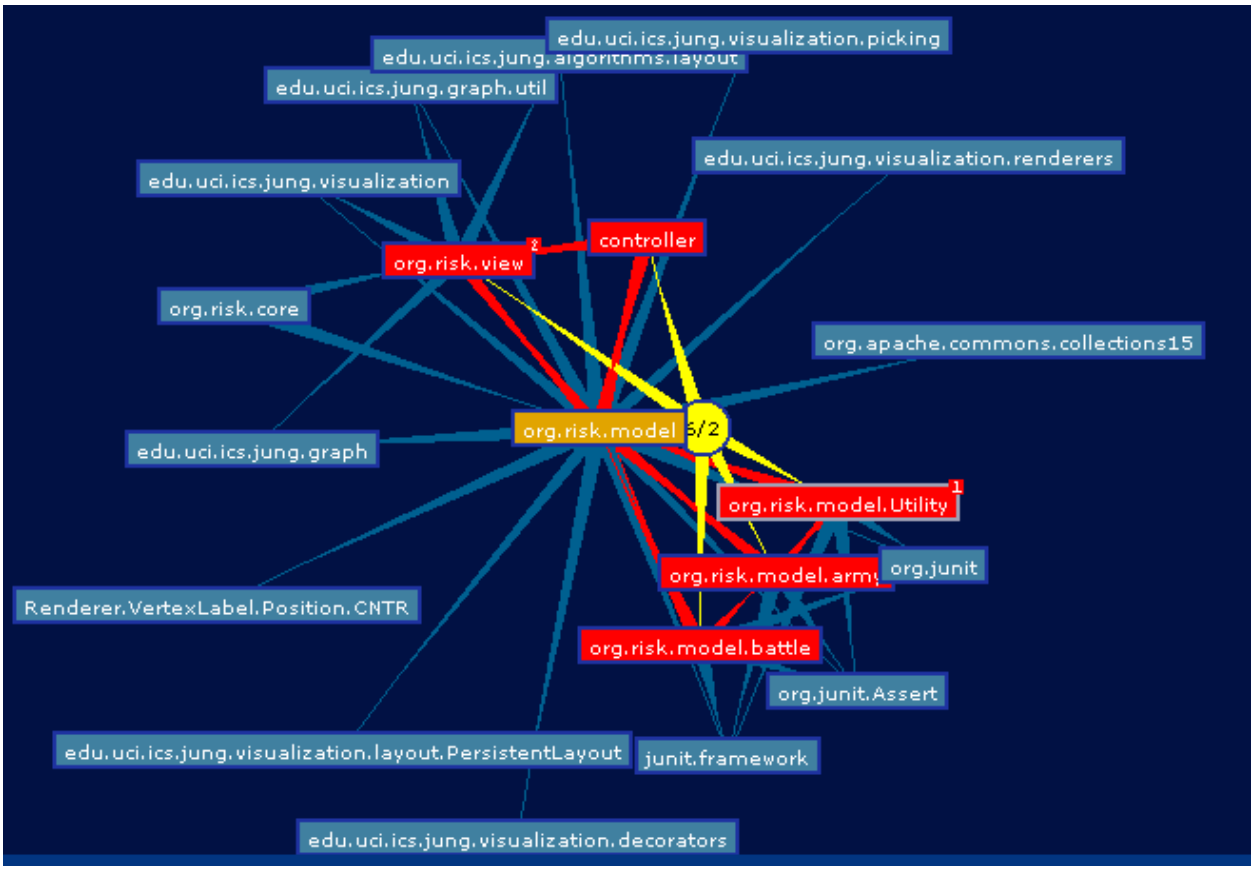
The project is about 4 months old, and I have taken permission from the group of people who submitted this last year, a total of 5 people worked on it. And this is the one and only version of the software available. It got highest marks in the term it was submitted in.

The project is about a board game quite famous named risk, the only change is that instead of a human playing with the computer its computer vs. computer just to show the game logic and mechanics of the game. There are a total of 3 computer players all playing against each other. Each player has resources in the shape of (energy, metal etc) which it uses to create weapons to win a fight against the other player, the other defining attribute a player has is technology (low, medium, high), we can combine these attributes in different ways to create different level of AI, and this all can be done dynamically while the program is running

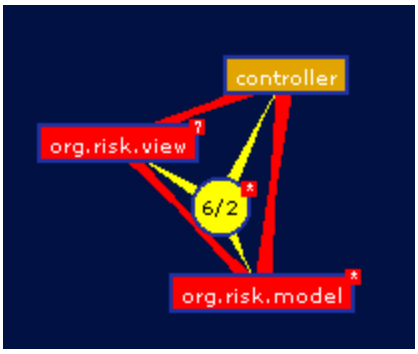
Project Size and Scope

The program has approximately 70 classes and 8000 lines of code and around 500 methods. We used two tools McCabe and eclipse plug-in (Metrics) to find out more about the program, e.g. Its complexity, total classes, total lines of code, cohesion between classes, inheritance etc

This information will help us in all the milestones to find out the relations between classes to make domain models and class models, and while refactoring we will know which class is coupled with which by just looking at the diagrams provided by the tools, rather than reading the whole code to understand the project we can take help from these tools. Some examples of the diagrams i was talking about are below



The graph above is the representation of the whole project and how the classes are connected together.



The graph above shows which other classes controller is connected to, and this can be done for every class in the project to understand the structure and architecture.

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per	1	0.014	0.116	1	/Risk/src/org/risk/model/Item.java	
Number of Attributes (avg/max per type)	259	3.548	6.441	37	/Risk/src/org/risk/view/NewGame.java	
Number of Children (avg/max per type)	12	0.164	1.182	10	/Risk/src/org/risk/model/army/Army.java	
Number of Classes (avg/max per packageFragr	73	6.636	4.598	16	/Risk/src/org/risk/model	
Method Lines of Code (avg/max per method)	5226	10.888	25.118	251	/Risk/src/org/risk/model/State.java	reduceArmyDetail
Number of Methods (avg/max per type)	462	6.329	9.195	41	/Risk/src/org/risk/model/Country.java	
Nested Block Depth (avg/max per method)		1.433	0.922	8	/Risk/src/controller/CControlPanel.java	init
Depth of Inheritance Tree (avg/max per type)		2.26	1.544	6	/Risk/src/org/risk/view/NewCountries.java	
Number of Packages	11					
Afferent Coupling (avg/max per packageFragn		6.818	7.614	26	/Risk/src/org/risk/model	
Number of Interfaces (avg/max per packageFra	0	0	0	0	/Risk/src/controller	
McCabe Cyclomatic Complexity (avg/max per		1.902	2.678	33	/Risk/src/org/risk/model/State.java	reduceArmyDetail
Total Lines of Code	7323					
Instability (avg/max per packageFragment)		0.528	0.267	1	/Risk/src/org/risk/core	
Number of Parameters (avg/max per method)		0.7	0.835	7	/Risk/src/org/risk/model/State.java	State
Lack of Cohesion of Methods (avg/max per typ		0.31	0.388	1	/Risk/src/controller/CMapVerifier.java	
Efferent Coupling (avg/max per packageFragm		4.545	3.144	10	/Risk/test/org/risk/model	
Number of Static Methods (avg/max per type)	18	0.247	0.658	3	/Risk/src/org/risk/model/Constants.java	
Normalized Distance (avg/max per packageFra		0.457	0.279	0.875	/Risk/src/org/risk/model/army	
Abstractness (avg/max per packageFragment)		0.015	0.048	0.167	/Risk/src/org/risk/model/battle	
Specialization Index (avg/max per type)		0.003	0.029	0.25	/Risk/src/org/risk/model/Item.java	
Weighted methods per Class (avg/max per typ	913	12.507	25.214	151	/Risk/src/org/risk/model/GameEngine.java	
Number of Static Attributes (avg/max per type	81	1.11	3.798	31	/Risk/src/org/risk/model/Constants.java	

This final snapshot shows all the details about our project and statistics.

Group Members

Qasim Naushad

Most of the work I have done is in C# and Vb.net and recently i started working on WPF and i believe that to be the most interesting work I have ever done. all my projects in masters were java related and were done in either eclipse or Net beans .my contribution to this project would be mainly writing the documents and helping with reverse engineering to make models, and definitely helping with refactoring.

Prakash Gunasekaran

I am very good in Object oriented programming language. So, I can easily understand the concept of the programming, architecture, different techniques and implementation of design pattern. From my last experience, I am strong in coding part of the project. I believe my past experience will help to understand the architecture.

Rajwinder Kaur

My school and industry work experience has provided me the opportunity to work under object-oriented environment which includes languages such as C++, C#, JavaScripts. Moreover, my graduate school experience in Concordia provided me thorough knowledge of requirement engineering, use cases, sequence diagrams, domain modeling, acceptance criteria etc. Therefore, I would like to work on Use cases, domain modeling, sequence diagram and software refactoring part of the project.

Harsahiljit Singh Mehmi

I have good knowledge of object oriented programming languages. During my masters in Concordia University I have done courses like Software measurement, Software maintenance ,Software requirement specifications & User interface and design during these courses I have done various projects like interface design, UML modelling, how to deal with code smells etc. which will help me in this course .

Prabhjot Sekhon

I have past experience with PHP and web languages like HTML and CSS. Also I have basic knowledge of OO languages like C and JAVA. In my under graduation, I developed the project that uses JAVA as front end. For this particular project, I will use the basic knowledge of JAVA to build the architecture. I am also familiar with argoUML so will help in creating UML diagrams.

Yash Paliwal

My IT experience consisted of the academic projects that I made in my undergraduate degree where I got familiarized with the toughness of designing an architecture, optimization and testing . I have worked on Object Oriented Languages like C++ and Java. I can help in improving the architecture of this project by combining the techniques of data modelling and business modelling. I have also used argoUML in case studies in my graduate course and I have the basic knowledge to prepare UML diagrams and Classes.

M2 - Risk Game

Qasim Naushad–5658624

Prakash Gunasekaran - 6399185

Prabhjot Kaur Sekhon- 6473318

Rajwinder Kaur-6282490

Harsahiljit Singh- 6405975

Yash Paliwal- 6562566

Identifying the Actors and Stakeholders

A Helpful Persona

Name: Philip

Occupation: Student

Phillip is 23years old. He is a full time Engineering student at Concordia University. He is very busy in his studies. He is a huge fan of computer games. So whenever he gets any free time he loves to play games. Sports, war and board games are his favorites. Philip loves to play game with his friends so he is looking for game that enables multiple players to play to get her and against each other. Moreover, he wants a game that can be saved so that whenever it is started again, it starts form where it was left previously. As an engineering student Philip uses his computer for various other purposes like Coding etc. so he wants alight game that does not affect this computer much.

Stakeholders:

- **Developers:** They are responsible for the successful implementation of the project. Moreover they are also concerned that the project should be completed according to player specification within the schedule and resources provided.
- **Player:** Player is the one who plays the game. So he is interested in various aspects of game like save, modify, create new maps etc. Game is designed according to his needs and he is the one likes or dislike it.

Actors:

- **Admin/Player:** The Player is a human actor. Player creates, saves, modifies and displays the map for the players to play the game; the Player chooses which map is to be created and modifies it according to his instance. It displays the map using which resources are used by which countries which varies according to the strength of the countries. Player also manages the change in resources and has the control of pausing the game and makes the required changes or quit the game.

- **Computer System/Players:** The Player is a non-human actor. The game is played between the computers. Player updates log which keeps the track of the resources used by the Player and generates resources according to strength. Player also controls the Attacking and Defending action of the game on the availability of strength and resources. It loads the map on the player's choice.

Use Case Briefs:

Id: UC-1

Use Case: Create Map

Primary Actor: Player

The Player of the Risk Game wants to create a map.

Description:
This use case deals with creating of Map. Player creates the map by adding the continents, country and states. The player chooses the category and enters his choice in order to create the map.

Id: UC-2

Use Case: Save Map

Primary Actor: Player

The Player of the Risk Game wants to save a map.

Description
This use case deals with the saving the map created by the player.

Id: UC-3

Use Case: Modify Map

Primary Actor: Player

The Player of the Risk Game wants to modify a map.

Description

This use case deals with the modifying or editing the map created by the player. In this use case player can update, edit and delete the various choices(County, continent, states).

Id: UC-4

Use Case: Display Map

Primary Actor: Player

The Player of the Risk Game wants to display a map.

Description

This use case deals where the player can display the map of the running game with its resources being used and the current strength of both the players.

Id: UC-5

Use Case: Changing resources

Primary Actor: Player

The Player of the Risk Game wants to change the resources.

Description

These use case deals with changing the resources like army which player can use to attack his opponents.

Id: UC-6

Use Case: Pause Game

Primary Actor: Player

The Player of the Risk Game wants to pause the game.

Description

This use case deals where the player can pause game whenever he wants to quit or hold abruptly.

Id: UC-7

Use Case: Update Log

Primary Actor: System

The System of the Risk Game wants to keeps the track of the status.

Description

This use case deals with the game log where the system keeps the track of the status of the resources being used by the player.

Id: UC-8

Use Case: Attacking

Primary Actor: System

The System of the Risk Game wants to keeps the track of the status.

Description

This use case deals where player (system) can attack the territories that are adjacent or connected by map to his own territory. The attacking player attacks with one or two armies.

Id: UC-9

Use Case: Generating resources

Primary Actor: System

The System of the Risk Game wants to generate its resources.

Description:

This use case deals where player (system) can generate its resources that includes getting and placing new army and other game strengths.

Id: UC-10

Use Case: Generating resources

Primary Actor: System

The System of the Risk Game wants to generate its resources.

Description:

This use case deals where player (system) can generate its resources that includes getting and placing new army and other game strengths.

Id: UC-11

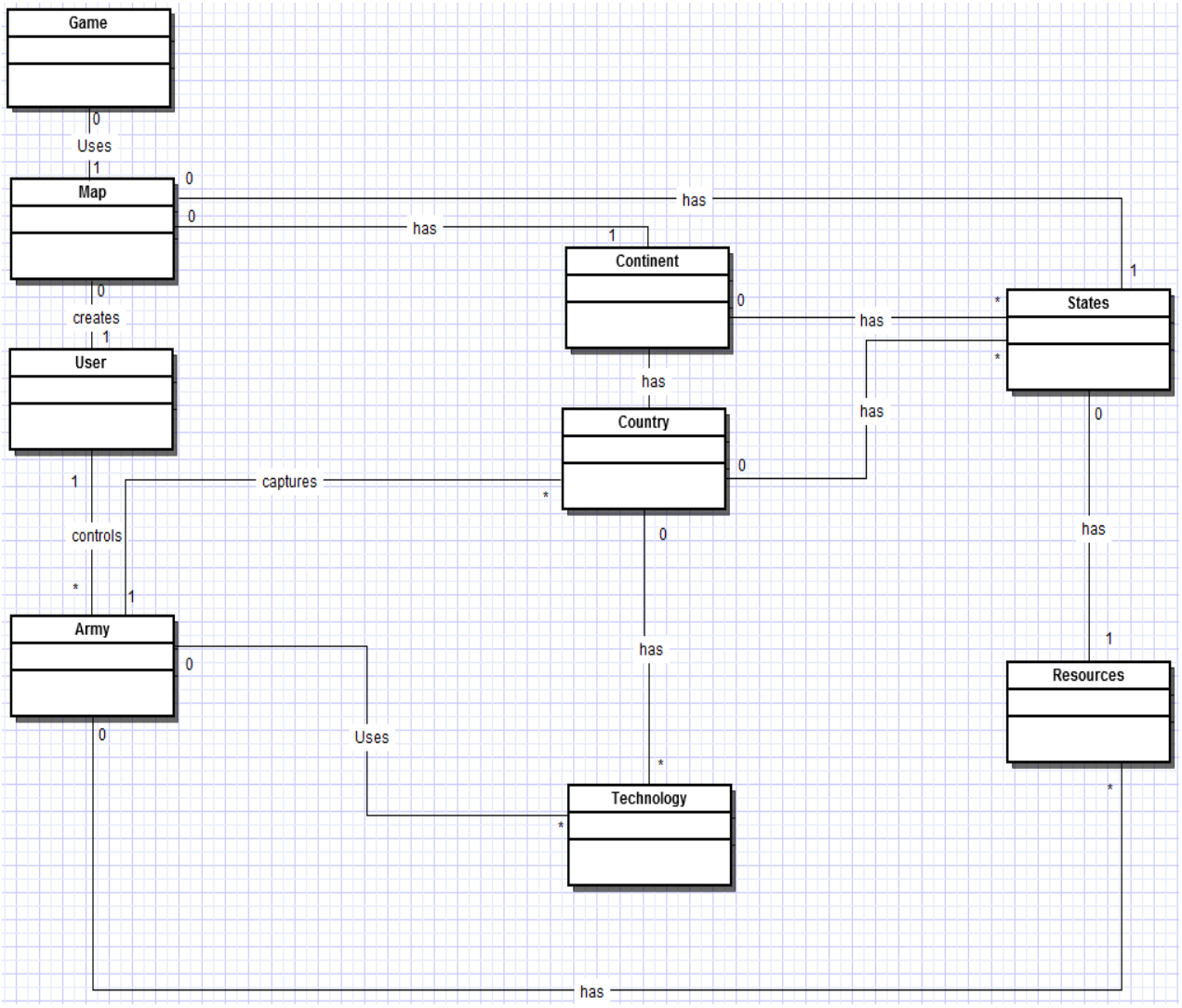
Use Case: Loading Map

Primary Actor: System

The System of the Risk Game wants to load the map.

Description:

This usecase deals where player (system) loads the map according to the player choice or continuing with the existing map.



UML Domain Diagram

Description:

The UML Domain Diagram describes that, player has many relationships with other classes, for example Player can create a map, where relationship between classes is one to one. Player can control many armies; relationship is one-to-many, as one player can have many armies. Similarly, Player can dominate more than one country; therefore relationship is one-to-many.

Similarly Country has relationship with other classes as, Country has a map, relationship is one to one, and Country has technologies, relationship one-to-many. And Continent has countries, relationship one-to-many.

Likewise, Army has relationship with other classes. Army has resources, where relationship is one-to-many; Army uses technologies, so relationship one-to-many. As Army captures more than one country, therefore relationship is one-to-many.

M3 Actual Architecture - Risk Game

Qasim Naushad – 5658624
Prakash Gunasekaran - 6399185
Prabhjot Kaur Sekhon - 6473318
Rajwinder Kaur - 6282490
Harsahiljit Singh - 6405975
Yash Paliwal - 6562566

Class Diagram of Actual System

Our Ideal Architecture VS Risk's Actual Architecture

Figure 1. Depicts the domain model of our Risk Game system. It is actually the conceptual framework of our system describing the state of the problem domain and interactions between the various classes of the system along with its associated cardinalities showing the relationship between various classes. This figure shows the general structure of the system.

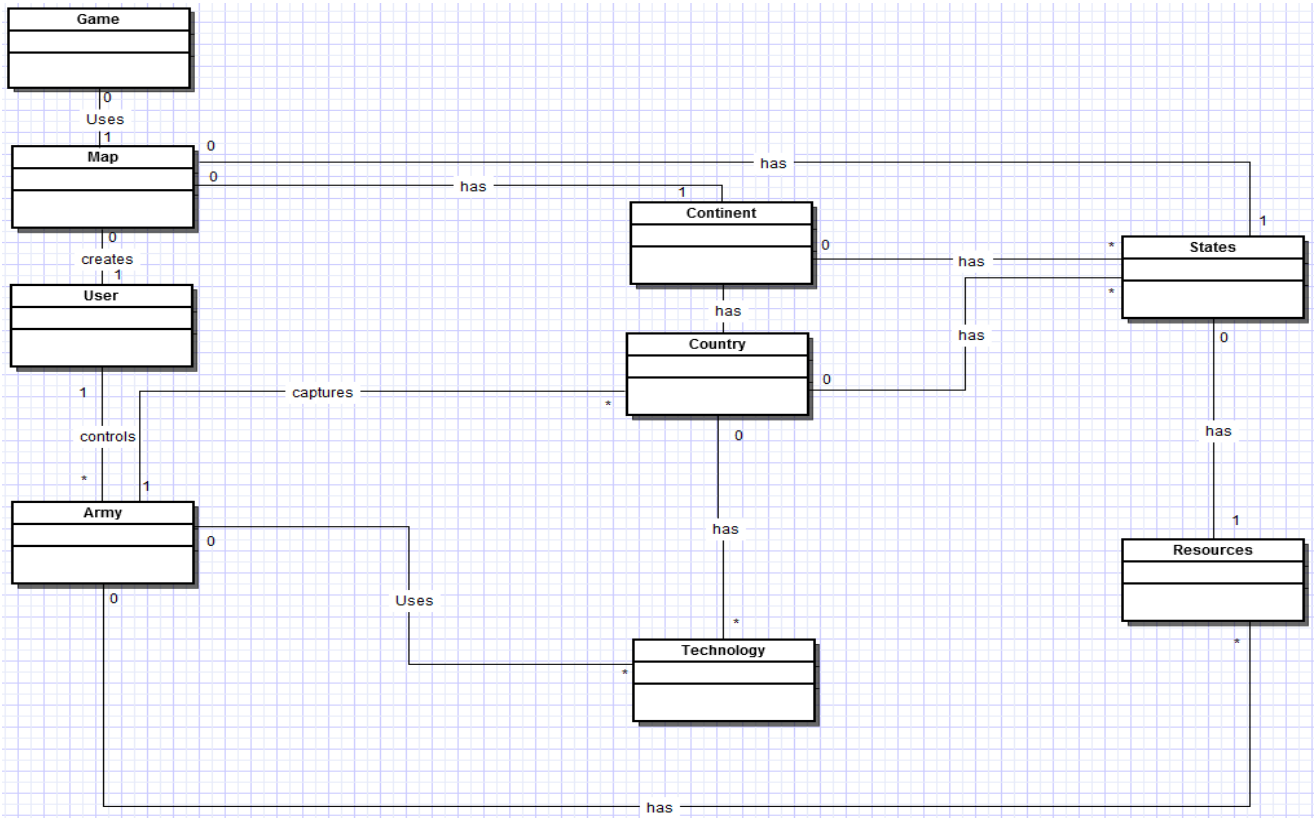


Figure 1. Our Domain Diagram from M2

Class Diagram:

After the detail analysis of the project we produced the actual class diagram shown in Figure2. using ObjectAid plug-in in Eclipse.

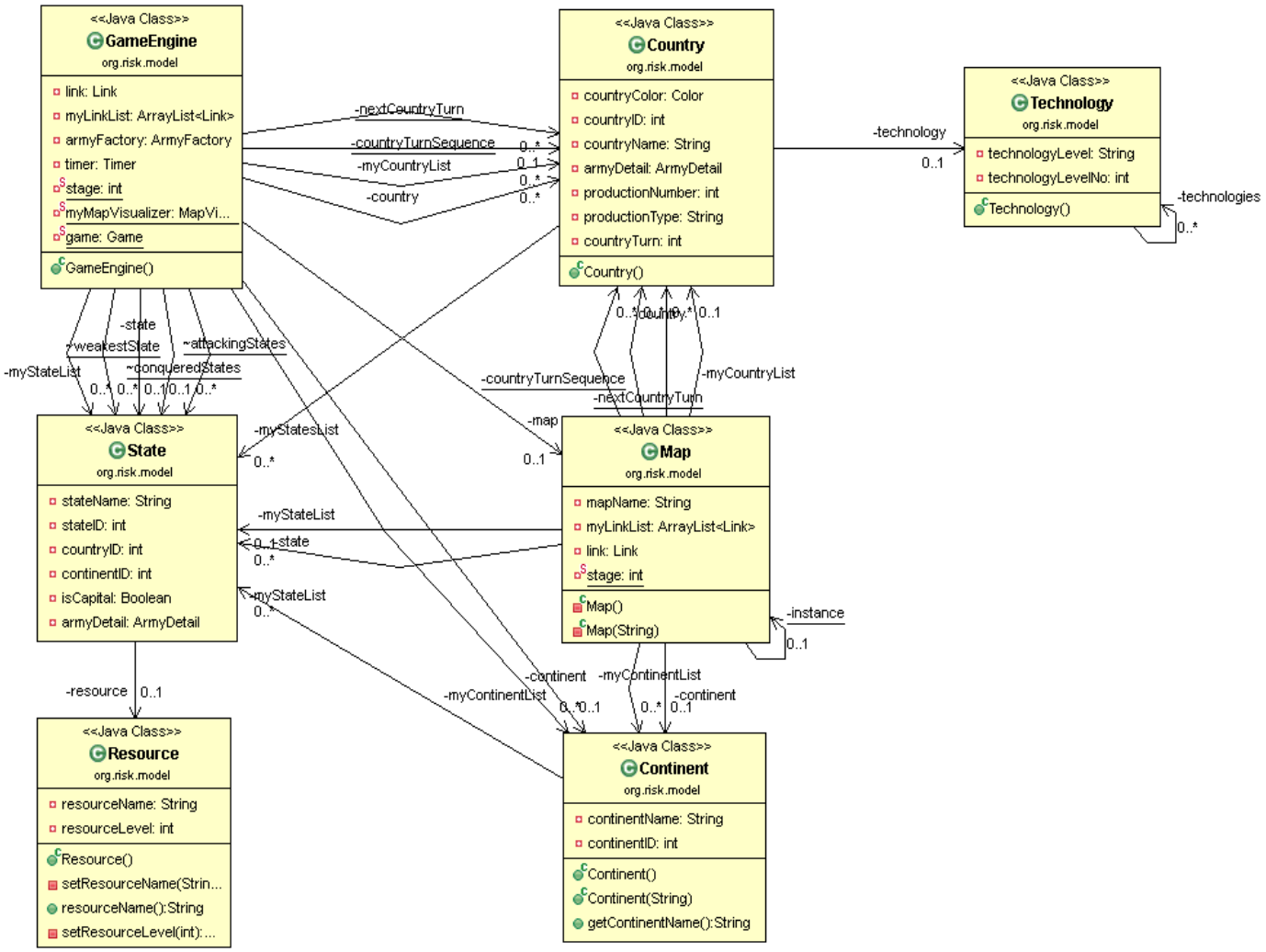


Figure 2. Risk's Actual Architecture

Most of the classes were mapped from domain model to class diagram for our project, some of the examples are, state class, continent class, Game Engine and map class etc, but some of the very important conceptual classes like Player (user) were not mapped, and all of the responsibilities of a player were meshed up with Country class which was making the

architecture very hard to comprehend. Like in our risk game, Country Class has two main methods country() and player() method the former handles the responsibilities like handling the information related to countries and later deals with the player responsibilities like taking the turns which ultimately increases the cohesion within the country class.

For example in our country class, the country() has got many methods within itself like: getCountry(), setCountry(), getCountryName(), setCountryName(), getMyStatesLists(), setMyStatesLists() etc. and the player() methods has getCountry(), getTchnology(), getTurn() etc. All the methods are invoked in the country class.

Furthermore, conceptual class army and army types were implemented in the project by using factory pattern design to take advantage of polymorphism because there were about ten different types of armies.

Classes and Relationships between them

Game engine class handles the work flow of whole project. Map class carries whole information about map like continents, countries, states, links etc. The game engine class uses the map class to get map information and distribute it to other classes. Now as the class game engine holds all the data of map other classes instead of going to map classes uses game engine class to get map information which reduces redundancy.

Source Code

```
import java.util.Set;

import javax.sound.midi.Receiver;

……
public class GameEngine {
	private Map map;
	private ArrayList<Continent> myContinentList;
	private ArrayList<Country> myCountryList;
	private ArrayList<State> myStateList;
	private ArrayList<Link> myLinkList;
	……
	private void populateMap() {
		map = Map.getInstance();
```

```
            myContinentList = map.getMyContinentList();
            myCountryList = map.getMyCountryList();
            myStateList = map.getMyStateList();
            myLinkList = map.getMyLinkList();
            nextCountryTurn = map.getNextCountryTurn();
            countryTurnSequence = map.getCountryTurnSequence();
            stage = map.getStage();
    }
    ……
    public class Map extends Observable {
    private String mapName;
    private ArrayList<Continent> myContinentList;
    private ArrayList<Country> myCountryList;
    private ArrayList<State> myStateList;
    private ArrayList<Link> myLinkList;

    public ArrayList<Continent> getMyContinentList() {
            return myContinentList;
    }

    public ArrayList<Country> getMyCountryList() {
            return myCountryList;

    }public ArrayList<State> getMyStateList() {

            return myStateList;
    }

……
```

Code Smell and Possible Refactoring

To find code smells in the project we used a plug-in named Jdeodrant, it found two types of code smells which were in abundance in our project, Feature envy, and Long methods. Besides those code smells there was another major problem with the architecture of the system which was the country class, it was playing two roles, first to handle the information related to countries in the map, and the 2nd was to handle the player responsibilities like taking turns.

In order to minimize the responsibilities of country class, we divided the whole class into two classes i.e. country and the player Class. Both these classes now individually handles their responsibilities and eventually lower the cohesion.

The last code smell which we found out was that the class named GameEngine which is responsible to control the whole flow of the simulation and is very poorly structured. For example, GameEngine class has Feature Envy and Long Method types code smells and also if else statements alone make the code hard to read and understand. So, for this we extracted methods from the If-Else statements to make the code more readable and comprehendible.

We noticed that the project has no sense of responsibility in its architecture, there are few classes with all the code in them and the rest of them are fairly empty. After analyzing the project manually and by using tools like objectAid, Jdeodrant and McCabe we came up with a very basic solution, to evenly distribute responsibilities among classes.

Tool Used for generating UML Diagram as Objectaid.

See: http://www.objectaid.com/download

Tool Used to find refactoring as Jdeodorant

See: http://marketplace.eclipse.org/content/jdeodorant#.UVCwPhyk968

Tool Used to find Metrics as Metrics,

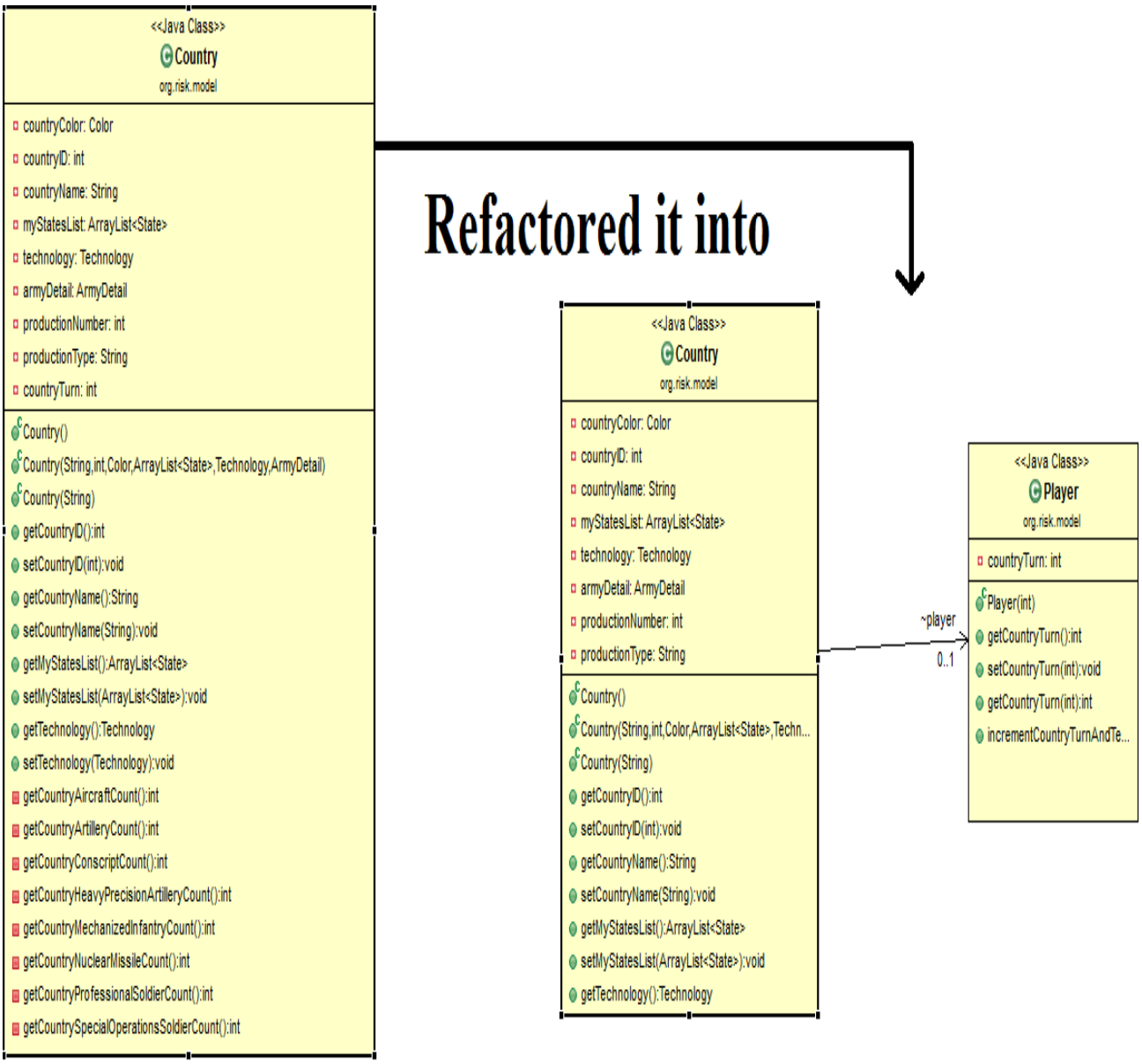
See: http://marketplace.eclipse.org/content/eclipse-metrics-plugin-continued#.UVCx2Ryk968

Our Proposed Refactoring:

1. Extracting another class named player from country class to divide responsibilities like taking turns and keeping country information
2. Extracting another class named Attack from GameEngine which will take over the responsibility of attack methods because the class GameEngine is overwhelmed.
3. Moving methods like (remove army strength(),addArmyDetail() from class State to class Army to eliminate feature envy
4. Moving methods from class Country to Army to eliminate feature envy
5. Lastly we extracted methods from the If-Else statements from the GameEngine, continent and state classes to make the code more readable

See: http://sourcemaking.com/refactoring

Before Refactoring:



Country class had both the responsibilities as we mentioned above which means it had low cohesion, and by doing the refactoring by using jdeodrant plugin we reduce feature envy, and by doing so we increase cohesion between classes. Both the classes have individual responsibilities now, and code comprehension is much easier.

M4- Risk Game

M4_Risk Game: Individual Part

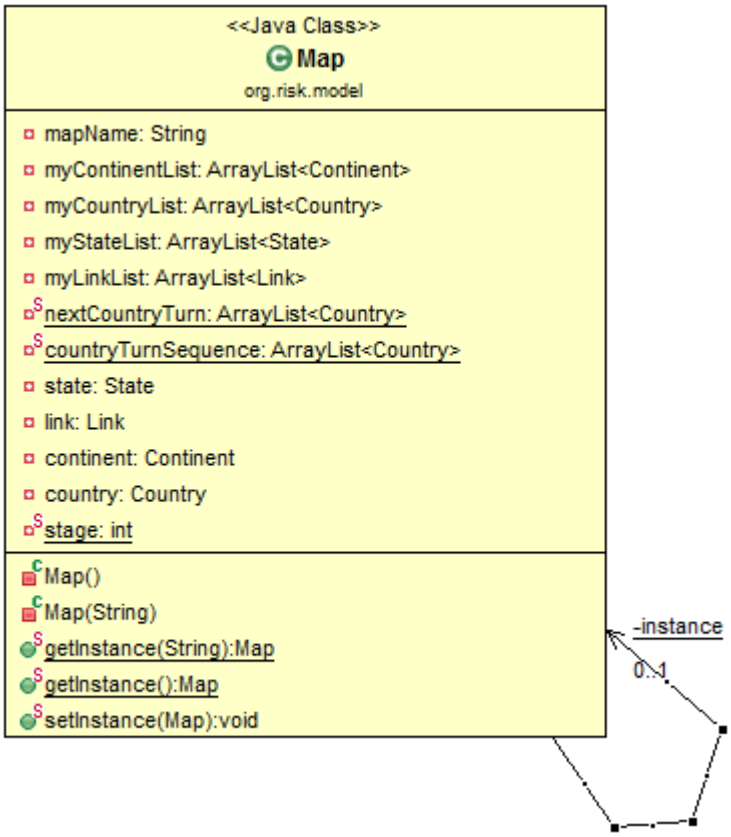
Qasim Naushad 5658624

The project we have is a game named risk, it's using model view controller as the architectural design which helps separate the Logic layer from the UI layer, it also promotes code re-usability and most of all it accommodates the developer when making changes. I used plug-in like Jeodrant and ObjectAid to better understand the structure of the system, after analysis I came up with 4 basic patterns which my project is using

1. Factory pattern, used by the org.risk.model.army package
2. Singleton pattern, used by the Map.Java class
3. Observer pattern, used by the org.risk.view
4. And the whole System is based on model view controller (MVC)

Singleton

The game is played on a single map, the way the map class is used is that it creates a single instance of the map and keeps using and modifying the same map object throughout the flow of system Below is the class diagram of map class, showing singleton in action



The purpose of map class is to read all the inputs in the provided map, like number of states, continents , links between them etc, the map class loads all that data into the system to use and distribute, rather than loading the input each time its needed the map class does it once. The pattern singleton provides the required functionality by making it impossible to make more than one instance of the map class, in this way the same map instance is used all around the system and keeping track of map becomes easier, rather than making more than one instances and keeping track of all of them.

The working code of the map class is shown below and you can see all the necessary methods types like static map instance, private constructor which are all a part of singleton

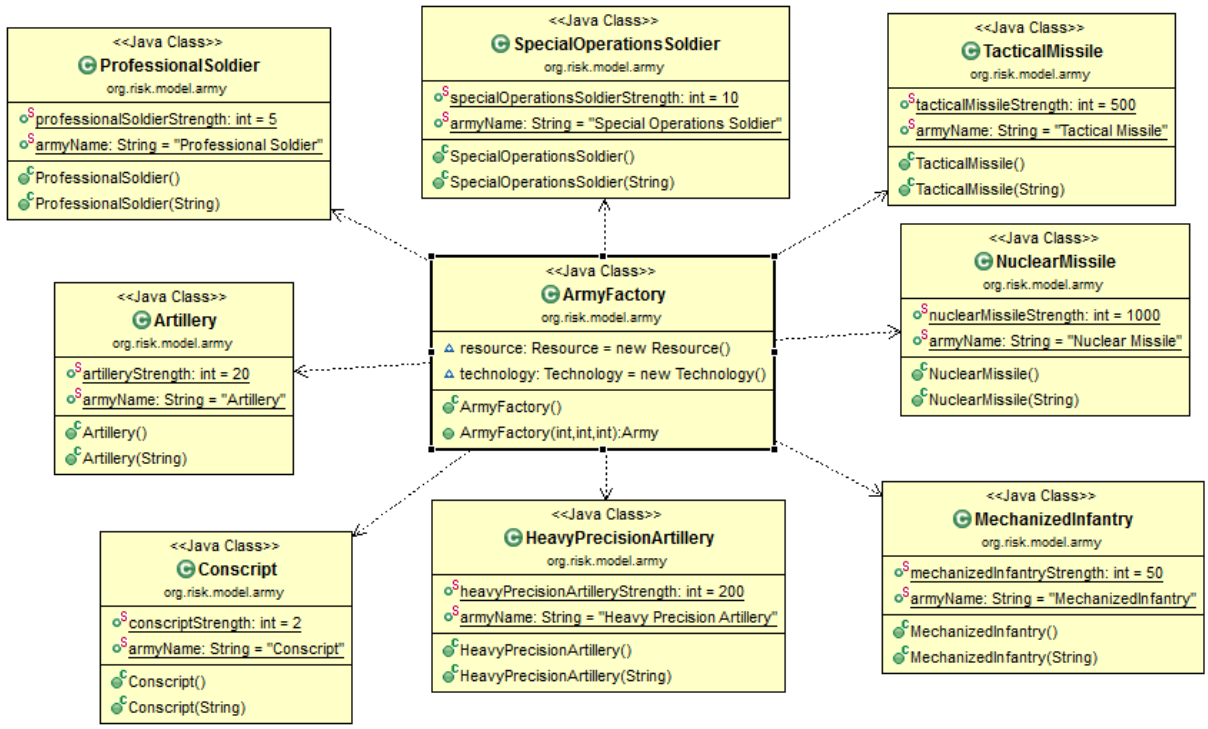
```
public class Map extends Observable {
	private static Map instance = null;
	....
	private Map() {
	.....
	}

	public static Map getInstance() {
		if (instance == null) {
			instance = new Map();
		}
		return instance;
	}
```

```
        public static void setInstance(Map map) {
                instance = map;
                }
.....
        }
```

Link to singleton pattern: http://sourcemaking.com/design_patterns/singleton

Factory pattern

in our project each country can command different types of armies, and each army has its own properties, but the basic structure of the classes is the same, each type of army at most adds on extra method to itself for that reason all types of army classes extend (inherit) from Army class taking use of polymorphism (OOP), in this scenario using factory pattern was a good choice, using polymorphism and class hierarchy to extend functionality rather than hard coding every class with all methods and dynamically deciding which class to initialize. The dynamic initialization is where factory pattern comes in. The diagram below shows how the package under observation is interacting within and using factory pattern.



The code below shows how the factory method depends on two attributes, technology and resources and based on that it chooses which army type to initialize

```
public class ArmyFactory {

        ......

        public Army ArmyFactory(int resourceLevel, int technologyLevel,int
numberOfStates) {
```

```
                if (resourceLevel ==resource.noResource().resourceLevel()) {
                return new Volunteer();
            }

            else if (resourceLevel == resource.metalResource().resourceLevel()
                        && technologyLevel == technology.technologyLevelBasic()
                                    .technologyLevelNo()) {
                return new Conscript();
            }


            else if (resourceLevel == resource.energyResource().resourceLevel()
                        && technologyLevel == technology.technologyLevelBasic()
                                    .technologyLevelNo()) {
                return new ProfessionalSoldier();
            }

        .......
            else {
                return new Army();
            }
        .......
}
```

Link to factory pattern: http://sourcemaking.com/design_patterns/factory_method

M4- Risk Game (Individual Part) Submitted By Prakash Gunasekaran (Student ID- 6399185)

Identified Pattern in Army package (org.risk.model.army) as Factory Pattern

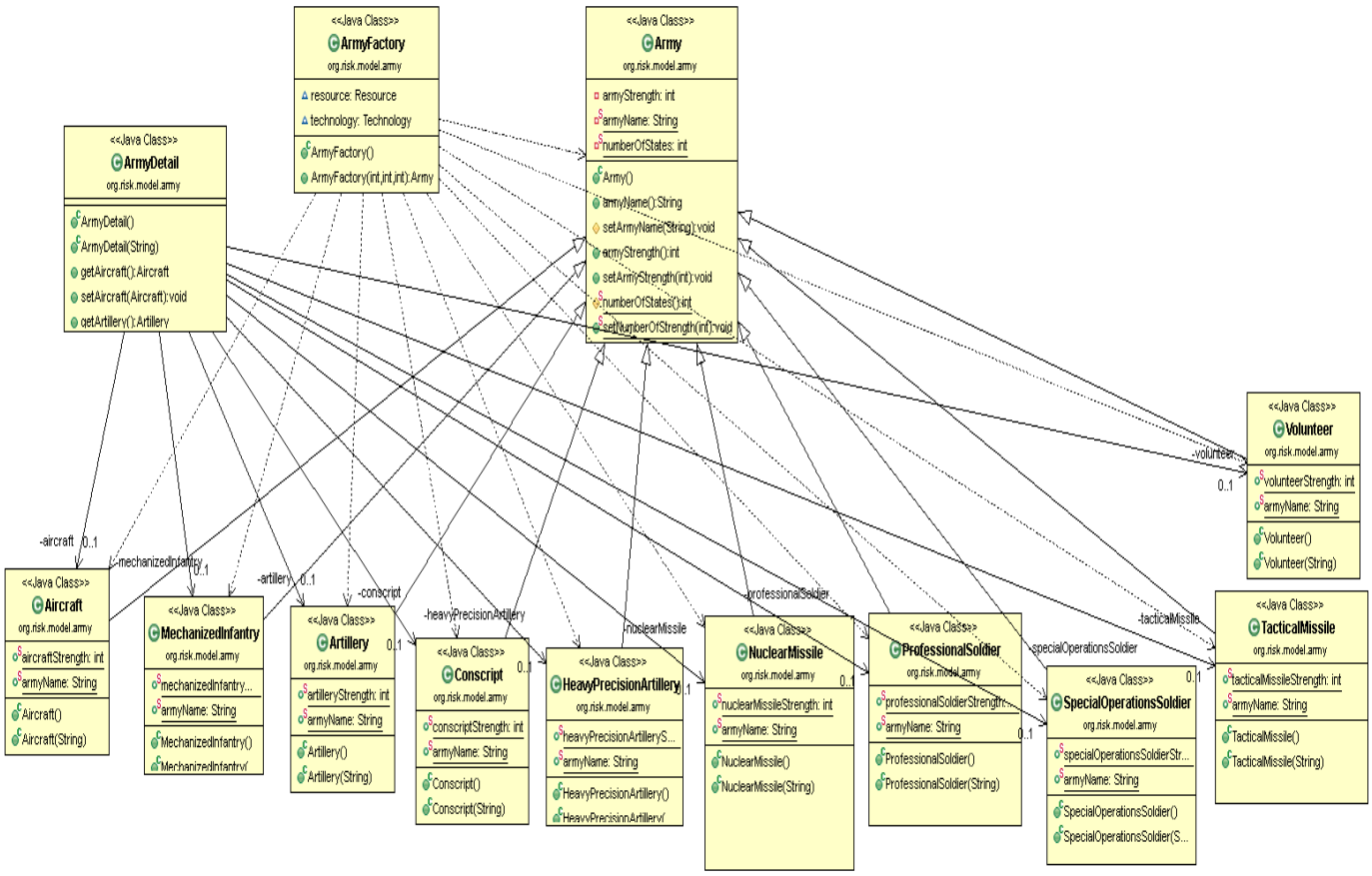


Fig 1 - UML Diagram for Factory pattern

Identified Pattern in Map.java (org.risk.model) as Singleton Pattern

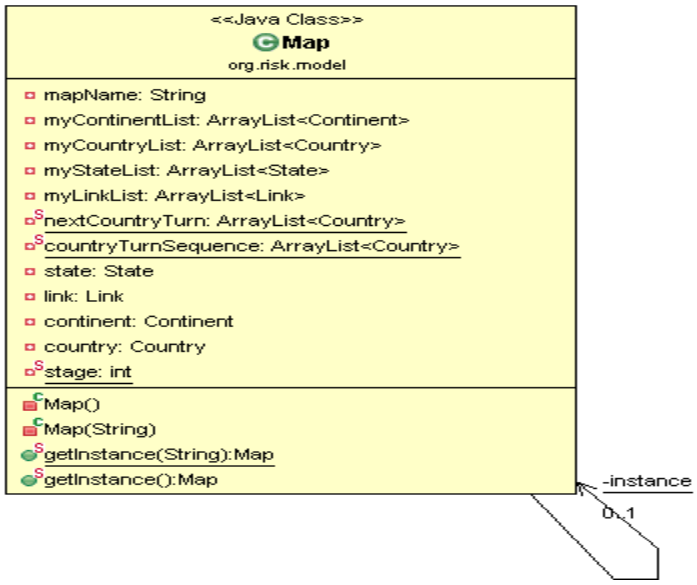
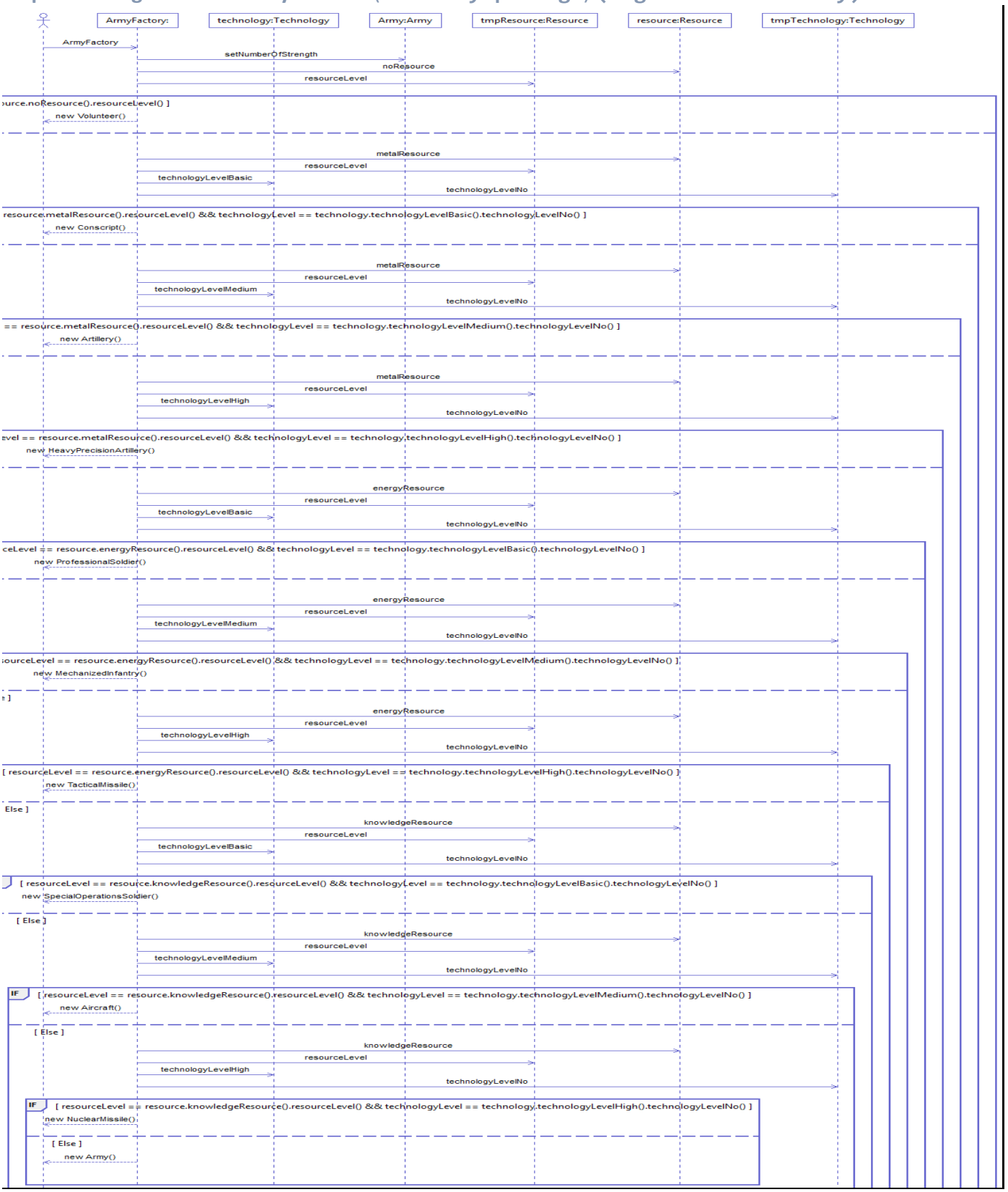


Fig 2 - UML Diagram for Singleton pattern

Tool Used to genetate UML Diagrams, See http://www.objectaid.com/download

Sequence Diagram for Factory Pattern (For Army package) (org.risk.model.army)



Tool Used to genetate Sequence Diagram, See http://www.objectaid.com/download

From Army package (ArmyFactory.java)

```
package org.risk.model.army;
import org.risk.model.Resource;
import org.risk.model.Technology;
public class ArmyFactory {
    // The class is responsible for which type of the army to call.
       Resource resource = new Resource();
       Technology technology = new Technology();
public Army ArmyFactory(int resourceLevel, int technologyLevel,
                  int numberOfStates) {
             Army.setNumberOfStrength(numberOfStates);
             // If a country has no Resource it will produce Volunteer as Army
             if (resourceLevel == resource.noResource().resourceLevel()) {
                   return new Volunteer();
             }
             // if a country has metal as a resource and Basic technology, it will
             // produce Conscript as Army
             else if (resourceLevel == resource.metalResource().resourceLevel()
                         && technologyLevel == technology.technologyLevelBasic()
                                     .technologyLevelNo()) {
                   return new Conscript();}
.................
}
```

From Army package (Army.java)

```
package org.risk.model.army;
import javax.xml.bind.annotation.XmlAccessType;
public class Army {
       private int armyStrength;
       private static String armyName;
       private static int numberOfStates;
public String armyName(){
             return armyName;
       } ..............
}
```

From Army package (ArmyDetail.java)

```
package org.risk.model.army;
import java.util.Enumeration;
public class ArmyDetail {
       private Aircraft aircraft;
       private Artillery artillery;
        ……..
public ArmyDetail(String name) {
             aircraft = new Aircraft(name);
             artillery = new Artillery(name); ….}
public Aircraft getAircraft() {
             return this.aircraft;
       }
private int getNuclearMissileCount(ArmyDetail armyDetail) {
             return (armyDetail.getNuclearMissile().armyStrength() /
NuclearMissile.nuclearMissileStrength);
       }…….
.…}
```

Identified Patterns:
Factory pattern is used in the org.risk.model.army package:

In an Army Package, I found Factory Pattern. From Figure1 and Figure 3, we can easily identify this pattern. Figure1 explains the UML diagram of an Army Package. Figure 3 explains sequence diagram of an Army Package. In an Army Package, ArmyFactory class focus the Army class into other classes like Aircraft.java, MechanizedInfantry.java, Artillery.java, Conscript.java, HeavyPrecisionArtillery.java, NuclearMissile.java, ProfessionalSolider.java, SpecialOperationsSoldier, TacticalMissile.java and Volunteer.java classes. All these classes have been inherited from Army.java class. Theses subclasses decide which class to instantiate. That is polymorphism extending its functionalities. While using Factory Pattern, dynamical initialization performs it. So, by creating one or more objects, it all shares the same interface and behaviours.

The source code above shows how the factory method depends on two attributes, technology and resources and based on that it chooses which army type to initialize

Definition is taken, See: http://sourcemaking.com/design_patterns

Identified Pattern in Map.java(org.risk.model) as Singleton pattern

From Figure 2, it creates a single instance of the map. In Map.java class, from the whole operation, it modifies the same map object.

Definition is taken, See: http://sourcemaking.com/design_patterns

Some other identified patterns are:

1. Factory pattern is used in the org.risk.model.army package
2. Singleton pattern is used in the Map.Java class (org.risk.model package)
3. Observer pattern is used in the org.risk.view
4. And the whole System is based on model view controller (MVC)

See: http://sourcemaking.com/design_patterns

M4_Risk Game: Individual Part

Rajwinder Kaur: 6282490

Our Risk Game project is designed by following Model View Controller (MVC) architectural design. Plugins like Jeodrant and ObjectAid has been used in order to perform reverse engineering of the code for better code comprehensibility. The project has implementation of many patterns like Singleton Pattern, Factory Pattern & Observer Pattern etc.

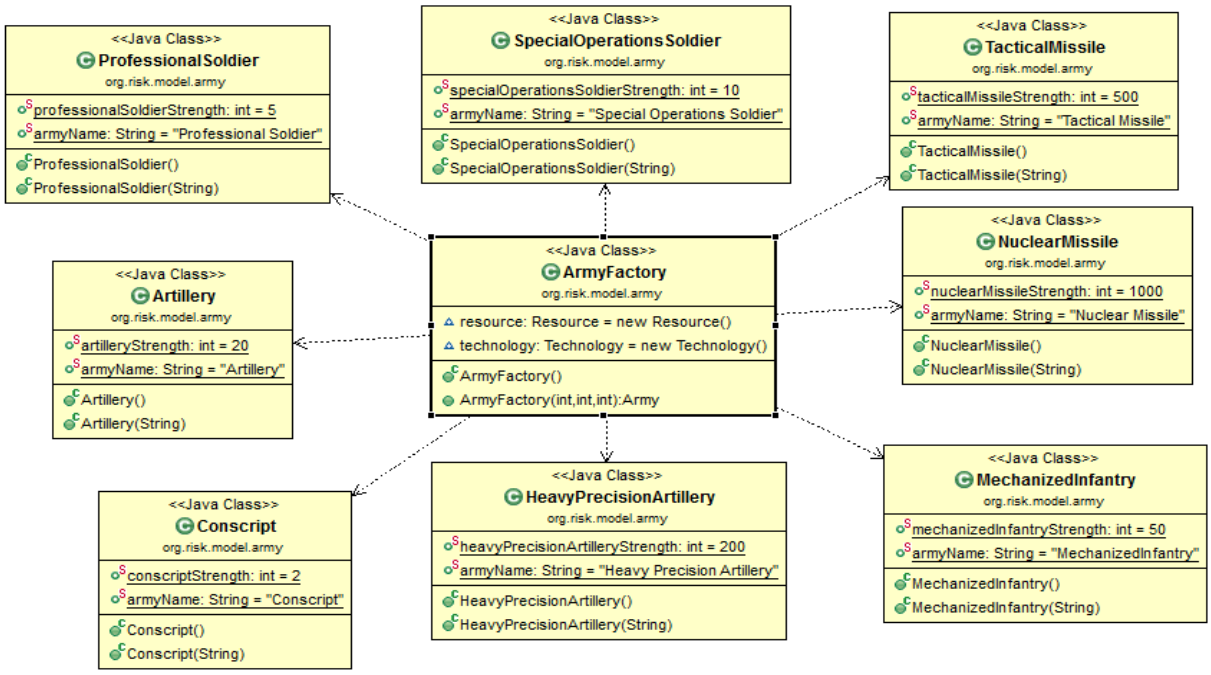
Brief description of one of the pattern named as Factory Pattern is as follows:

Factory Pattern

According to definition, Factory pattern defines an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses. In general, it allows us to create one or more context specific objects that all share the same Interface and basic functionalities.

Similarly, in our Risk Game project, package org.risk.model.army has the implementation of Factory pattern. The ArmyFactory class is created and specializes the army into different types of other classes like Conscript.java, ProfessionalSoldiers.java etc., each with their own behavior & properties. All these army classes are inherited from the main base class army i.e we have an inheritance hierarchy that exercises polymorphism extending its functionalities. That's why a static factory method in the base class was added hence implementing the Factory Pattern.

Class diagram is shown below that implements factory pattern:



These classes are inherited from the ArmyFactory.java, so whenever any army class is called, as per the conditions defined, instead of directly accessing the desired class, methods of the base class will be called. This refactors the code and makes it easier to differentiate between the called classes and provides the output accordingly.

The code below shows how the factory method depends on two attributes, technology and resources and based on that it chooses which army type to initialize

```
public class ArmyFactory {


      ......


      public Army ArmyFactory(int resourceLevel, int technologyLevel,int
numberOfStates) {
```

```
                if (resourceLevel ==resource.noResource().resourceLevel()) {
                return new Volunteer();
            }

            else if (resourceLevel == resource.metalResource().resourceLevel()
                        && technologyLevel == technology.technologyLevelBasic()
                                    .technologyLevelNo()) {
                return new Conscript();
            }



            else if (resourceLevel == resource.energyResource().resourceLevel()
                        && technologyLevel == technology.technologyLevelBasic()
                                    .technologyLevelNo()) {
                return new ProfessionalSoldier();
            }

        .......
                else {
                return new Army();
                }
        .......
}
```

Link to factory pattern:

http://sourcemaking.com/design_patterns/factory_method

http://patterns.instantinterfaces.nl/current/Refactoring-and-Design-Patterns-INTRDP-GMDV.html

M4_Risk Game: Individual Part

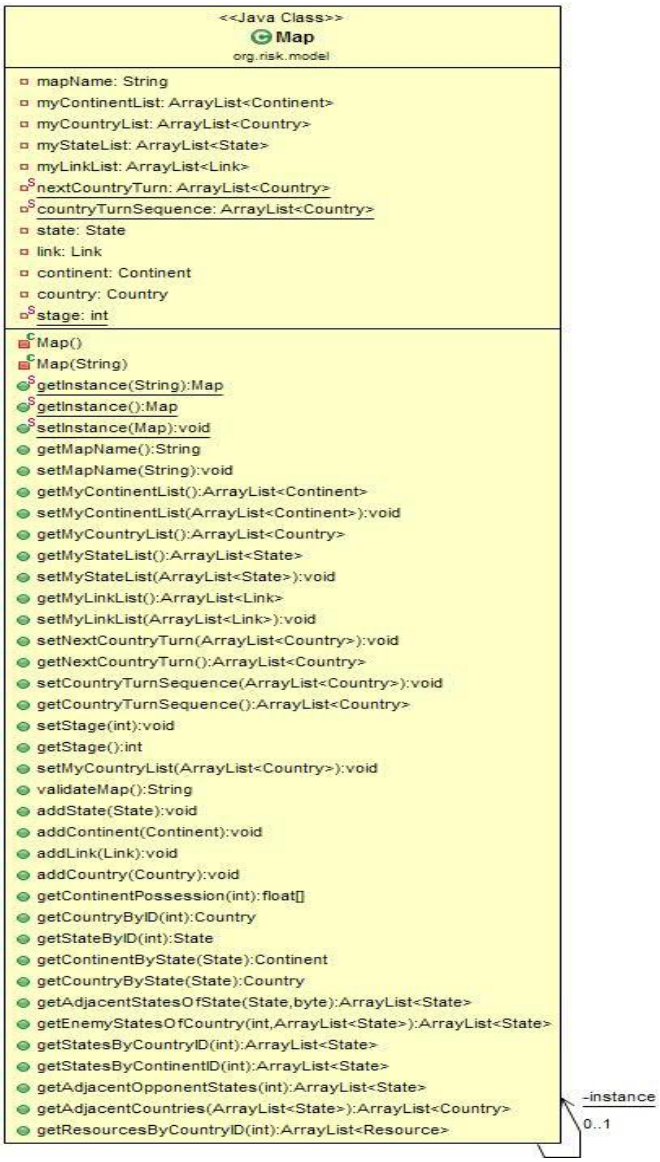
Harsahiljit Singh - 6405975

Singleton Pattern

This pattern ensures that class only has one instance and provide global point of access to it. In our project this pattern is implemented in class Map.java This class contains whole information about the map like Continents, States, and Number of links and provides this information to other classes like the class Game engine uses this class to get information about whole map and further provides this information to other classes.

This pattern is very helpful in our project as the Map class only has one instance so all classes which want to interact with Map class uses only one instance. Singleton pattern makes it very easy for other classes to interact with Map class.

Class diagram of Map.java class using java plugin “Objectaid”



I used java plugin WOP to find patterns http://www-ist.massey.ac.nz/wop/

```
public class Map extends Observable {
	private String mapName;

	private ArrayList<Continent> myContinentList;

	private ArrayList<Country> myCountryList;

	private ArrayList<State> myStateList;

	private ArrayList<Link> myLinkList;

	private static ArrayList<Country> nextCountryTurn;

	private static ArrayList<Country> countryTurnSequence;
```

```
        private State state;
        private Link link;
        private Continent continent;
        private Country country;
        private static int stage;


private static Map instance = null;

        private Map() {
                this.mapName = "";
                ........
                ........

        }

private Map(String name) {
                this.mapName = name;
                .......
                ......
        }



        public static Map getInstance(String name) {
                if (instance == null) {
                        instance = new Map(name);
                }
                return instance;
        }



      public static void setInstance(Map map) {
                instance = map;
        }
```

http://sourcemaking.com/design_patterns/singleton

M4_Risk Game Indiviual Part

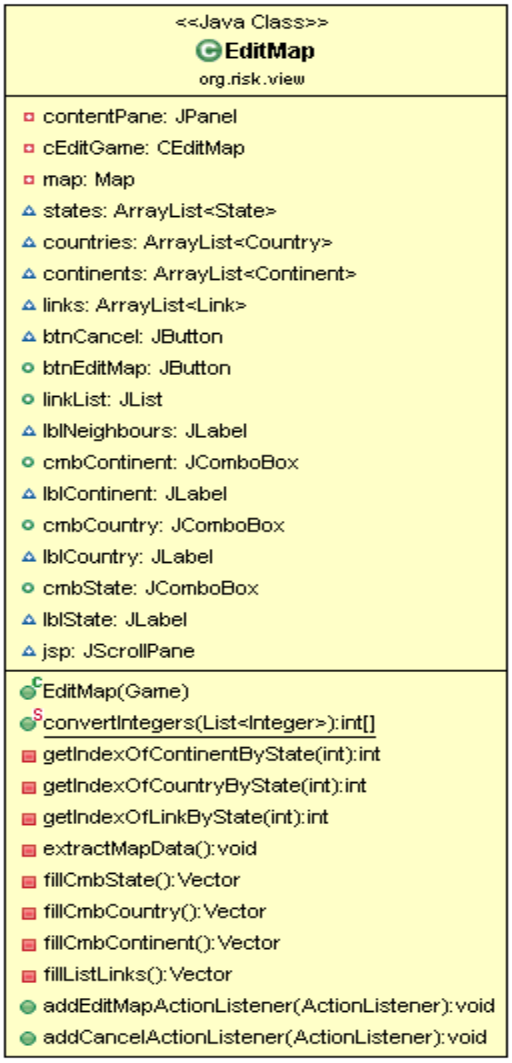
Prabhjot Kaur Sekhon 6473318

OBSERVER PATTERN:

Our project is based on MVC architecture. The View Model object is a sort of mediator between what happens on screen, between what the user does and the Model that simulates the World that forms the Game.
In the project, to understand the structure of the system, Jeodrant and Object Aid plug-ins are used. Various patterns have been used in this project, one of which is “observer pattern” used by org.risk.view.

The Observer pattern is the basis of events and event driven systems. It defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
Following class, edit map, of org.risk.view uses “observer pattern”. Whenever any action is performed, all other class get notified. Classes including state, country, and continent are affected when changes in class Edit Map are done. When subject changes all the observer classes are notified to perform certain actions according to the conditions defined.

Below is the class diagram of EditMap class, showing observer pattern in action :



Below, is the working code, defining required actions according to the given condition :

```
cmbState.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
         JComboBox cbs = (JComboBox) e.getSource();

int stateID = ((Item)cbs.getItemAt(cbs.getSelectedIndex()))
                              .getId();
if (stateID >= 1) {
                    // Enable elements of the window
                    cmbContinent.setEnabled(true);
                    cmbCountry.setEnabled(true);
                    linkList.setEnabled(true);
                    btnEditMap.setEnabled(true);
        }…….
        ……
        ……
 else {
                    int[] emptyList = { -1 };
                    // Disable elements of the window
                    cmbContinent.setSelectedIndex(-1);
                    cmbCountry.setSelectedIndex(-1);
                    linkList.setSelectedIndices(emptyList);
                    cmbContinent.setEnabled(false);
                    cmbCountry.setEnabled(false);
                    linkList.setEnabled(false);
                    btnEditMap.setEnabled(false);
     }
   }
 });
```

Link to observer pattern:
http://sourcemaking.com/design_patterns/observer
http://patterns.instantinterfaces.nl/current/Refactoring-and-Design-Patterns-INTRDP-GMDV.html

Risk M4 Individual Part

YASH PALIWAL 6562566

Our project is a game developed in Java. It's name is Risk Game in which computer is the player and the game is about the battles fought between countries. The whole system is based on MVC - Model View Controller. I used plug-ins like **Jeodrant** to understand the architecture of the system and **Object-aid** to create class diagrams and **WOP** to find patterns in Eclipse.

The patterns which we discovered in our projects are :

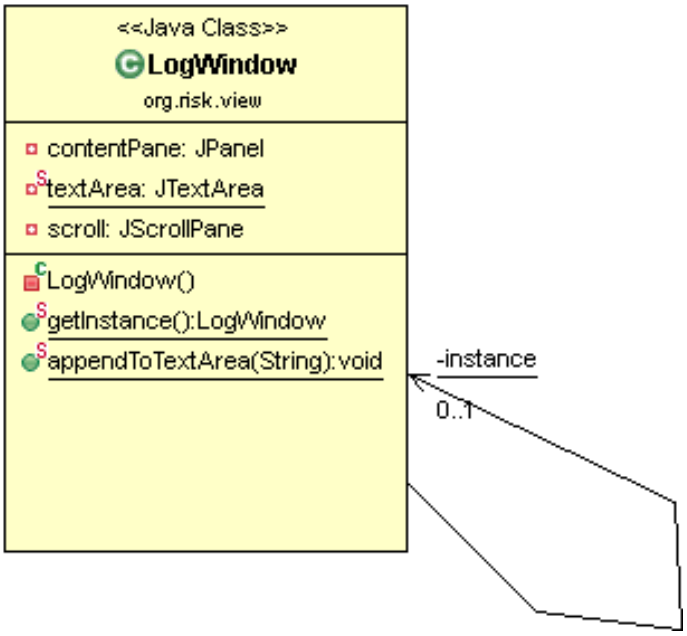
1. Factory pattern, used in the org.risk.model.army package
2. Singleton pattern, used by the Map.Java class
3. Singleton pattern, used by the LogWindow.java class
3. Observer pattern, used in the org.risk.view package

The pattern I observed is known as Singleton Pattern. I found this pattern in the LogWindow.java class in org.risk.view package. The **Singleton Pattern** is a design pattern that restricts the instantiation of a class to one object. This pattern is useful when exactly one object is needed to coordinate actions across the system. The term comes from the mathematical concept of singleton.

LogWindow.java class has the GUI (Graphic User Interface) of the log window i.e. this class is responsible for the display of the log window which is one of the important part of our game. Singleton pattern makes it easier for other classes to interact with this class by making a single instance rather than more.

Link for singleton pattern: http://sourcemaking.com/design_patterns/singleton

Class Diagram of Logwidow.java class by plug-in **Objectaid**



Link For Java plug-in ObjectAid: http://www.objectaid.com/

The source code of the loginwindow.java class is shown below and we can see that all the necessary methods like static logWindow instance, private constructor are used which are all parts of this pattern.

```
public class LogWindow extends JFrame {

     private JPanel contentPane;
     private static JTextArea textArea;
     private JScrollPane scroll;
     private static LogWindow instance = null;

     ....

     ....

/**
      *
      *
      * @return : Instance of LogWindow Class
      */
     public static LogWindow getInstance() {
          if (instance == null) {
               instance = new LogWindow();
          }
          return instance;
     }

     /**
      * Append content to Log Window
      *
      * @param msg
      *             : Log message
      */
```

Link for Java plug-in **WOP:** http://www-ist.massey.ac.nz/wop/

M4 - Implement a Refactoring - Risk Game

Qasim Naushad – 5658624
Prakash Gunasekaran - 6399185
Prabhjot Kaur Sekhon - 6473318
Rajwinder Kaur - 6282490
Harsahiljit Singh - 6405975
Yash Paliwal - 6562566

Patch set 1

- **Patchset 1 (0/3)**
  In this patch set we will pull up a method named addArmyDetails from class country and state and move it to class army, according to the reverse engineering tool (jeodrant) we found out that both classes have the same method doing the same job ,the code smells we will be eliminating are feature envy and duplicate code.
- **Patchset 1 (1/3)**
  We remove the method addArmyDetails from the country class to army class, and ran our test cases, the project was compiling properly and all the test cases passed.
- **Patchset 1 (2/3)**
  We remove the method addArmyDetails from the state class to army class, and ran the tests again and they all worked, the behaviour did not change
- **Patchset 1 (3/3)**
  As the final step we moved (method pull up) the method addArmyDetails from state and country class both and placed it in army class shown here, and ran the test cases again, they all passed showing successful method refactoring.

Patch set 2

- **Patch set 2 (0/3)**
  In this patch set we are going to extract a class named Player from class country, and after extracting the class we will move methods related to player turns into player class to divide responsibilities, country class will keep data about country and army details and player will handle player turns now
- **Patch set 2 (1/3)**
  We extracted a class named player and added turn methods to it, the diff file shows the changes done
- **Patch set 2 (2/2)**
  We moved the method player turn from gameEngine to player class