DESIGN PATTERN

기말과제: Borad Game

이름: 정현우

학번: 32204236

목차

Ⅰ. 서론.....2

1. 개요.....2

2. 디자인 패턴 적용 계획.....2

3. 클래스 다이어그램.....3

Ⅱ. 본론.....6

1. 추상 팩토리 패턴.....6

2. 스트라티지 패턴.....11

3. 템플릿 메소드 패턴.....15

4. 싱글턴 패턴.....18

5. 어댑터 패턴.....19

6. 파사드 패턴.....20

Ⅲ. 결과 및 전체 코드.....23

1. 결과.....23

2. 깃허브.....23

3. 전체 코드.....24

Ⅰ. 서론

1. 개요

최근 과거에 크게 유행했던 게임의 그래픽을 바꿔 리마스터 하거나, 아니면 그래픽 말고도 추가 기능, 컨텐츠를 추가하여 리메이크하는 경우들이 상당히 많이 존재한다. 이런 경우 말고도, 이미 만들었던 코드를 유지, 보수하는 일은 굉장히 중요하며, 운영체제가 32비트에서 64비트 지원으로 바뀔 때처럼 시대와 기술이 변함에 따라 기존 코드를 고쳐주어야 하는 경우가 있다. 이번 프로젝트에서는 이와 같이, 과거 프로젝트를 리메이크하는 것이 목표이다.

2020년 1학기, 1학년 1학기일 때 자바 프로젝트로 보드 게임을 구현한 적이 있다. 1학년 1학기에 했던 만큼, 상당히 미숙하게 작성한 부분이 많으며, 특히 상속과 객체 지향에 대한 이해가 거의 없던 상태로 진행하여 사실상 코드를 재사용하는 것은 불가능하다는 점에서 이번 프로젝트로 진행하기 적합하다고 생각했다. 보드 게임이라는 컨셉은 사용하되, 코드는 완전히 새롭게 작성될 것이고, 더 깔끔하고 디자인 패턴까지 적용시킬 수 있을 것이다.

보드 게임의 기본적인 규칙은, 가지고 있는 돈을 다 써버리면 지는 것이다. 주사위를 굴려 이동한 곳에 따라 돈을 사용하게 되고, 혹은 함정을 설치하여 상대방이 걸리면 돈을 얻을 수도 있다. 기존 1학년 때 만들었던 프로젝트에서는 서울, 인천 같이 실제 지명을 사용하고 통행료를 걷는 방식으로 돈을 내게 했지만, 이번엔 화산, 사막같은 곳으로 바꾸어 함정과 어울리도록 변경이 이루어진다. 또한, 각 지역에 따라 특수한 효과가 적용될 계획이다.

2. 디자인 패턴 적용 계획

적용될 디자인 패턴 중 하나는 추상 팩토리 패턴이다. 화산, 사막, 설산과 같은 지역들은 각 지역들의 특성이 있다. 그러한 각 지형들의 특성에 맞춰 지역을 위한 객체를 생성하기 위해서는 추상 팩토리 패턴을 적용하는 것이 좋을 것이라 판단하였다. 원래 계획에서는 지역에 따른 효과가 추상 팩토리에서 적용이 될 계획이었으나, 이후 기본 화산, 위험한 화산과 같은 방식으로 만드는 것으로 변경되어 통행료나, 위험한 장소에 걸맞는 효과를 주는 것에 추상 팩토리 패턴을 적용할 것이다.

두번째로는 스트라티지 패턴이다. 기본적으로 유저는 컴퓨터와 보드 게임을 진행하게

된다. 유저는 직접 주사위를 굴려야 하겠지만, 컴퓨터는 알아서 처리해야 한다. 이렇게 속성이 비슷하지만 액션을 달리해야 하는 경우엔 스트라티지 패턴을 사용하는 것이 적합할 것이다.

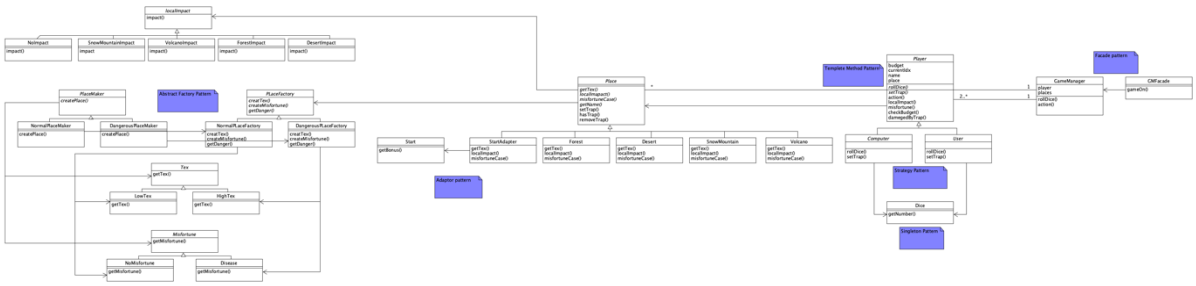
세번째로 사용될 디자인 패턴은 템플릿 메소드 패턴이다. 컴퓨터와 유저가 비슷한 행위를 하지만, 함정을 설치할 때는 서브 클래스에서 정의된 행동을 하게 된다. 하지만, 전체적으로 실행하는 절차는 같기 때문에 템플릿 메소드 패턴으로 알고리즘을 캡슐화하는 것이 적잘하다.

네번째로 적용될 디자인 패턴은 싱글턴 패턴이다. 컴퓨터와 유저가 다른 주사위를 사용하면 시드값 때문에 항상 굴릴 때마다 컴퓨터와 주사위가 같은 값을 가지게 될 수 있다. 컴퓨터와 유저가 같은 주사위를 사용하도록 싱글턴 패턴의 적용이 필요하다.

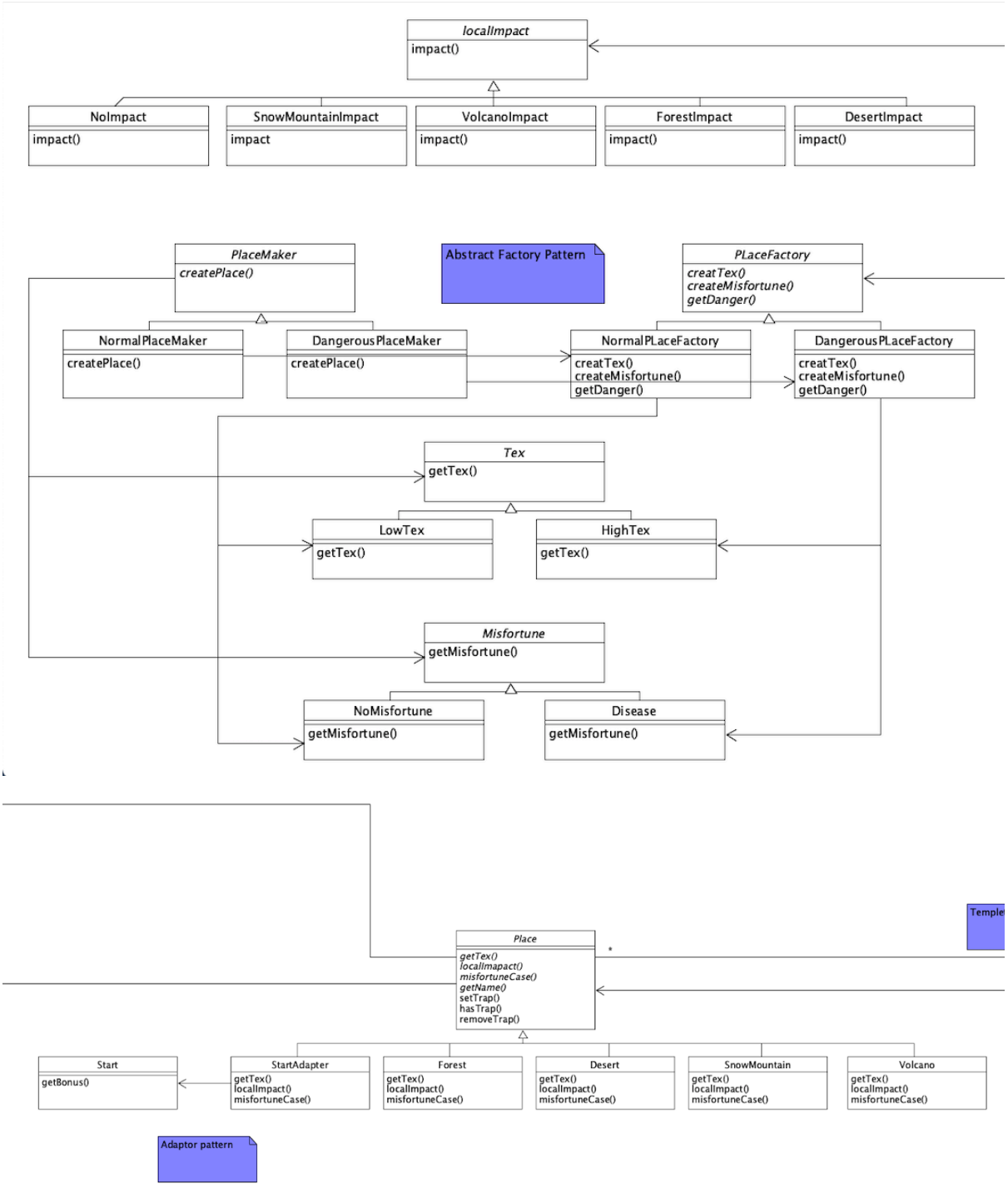
다섯 번째로 적용될 디자인 패턴은 어댑터 패턴이다. 적용될 지역은 화산, 숲, 설산, 사막 네 가지인데, 시작 지점을 네 지역 중 하나로 하는 것 보다는 새로 만드는 것이 낫다고 판단했다. 하지만, 기존 지역들과는 그 성질이 많이 다르기 때문에 일단 시작 지점을 위한 클래스를 만들고, 어댑터로 연결시키는 것이 적절하다 판단된다.

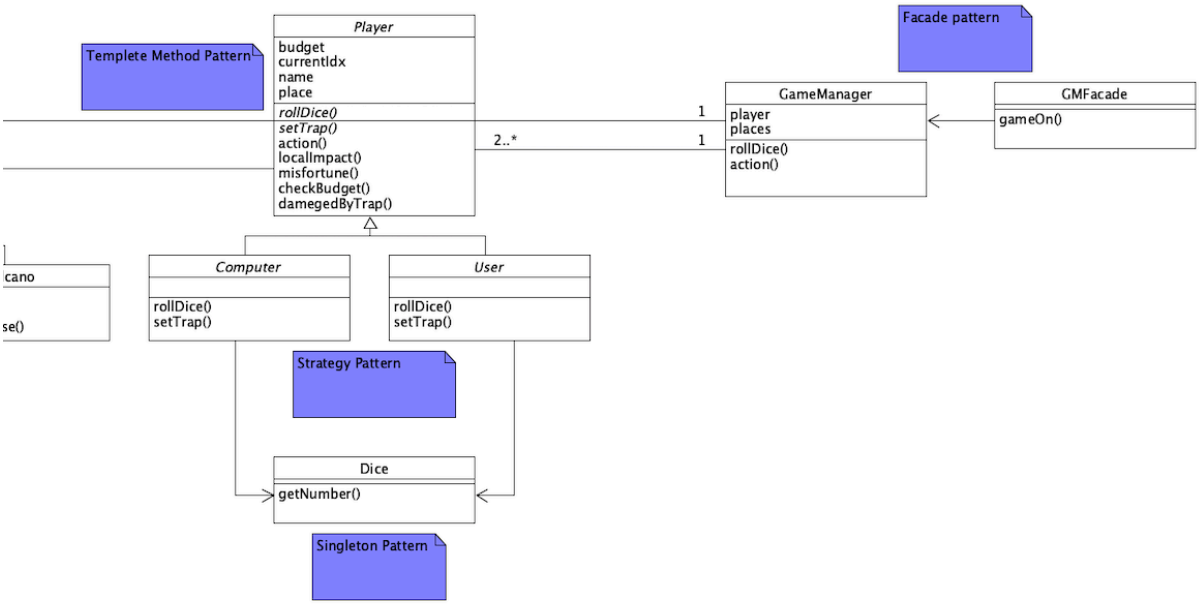
3. 클래스 다이어그램

전체



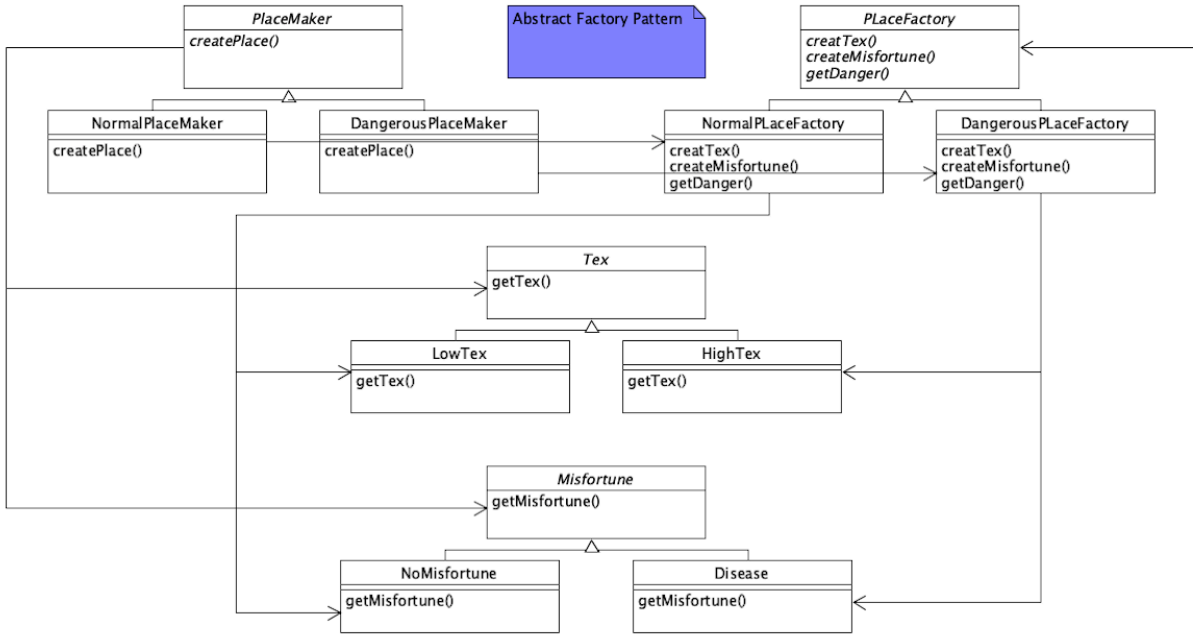
3분할


localImpact
impact()
NoImpact
impact()
SnowMountainImpact
impact
VolcanoImpact
impact()
ForestImpact
impact()
DesertImpact
impact()
PlaceMaker
createPlace()
Abstract Factory Pattern
PLaceFactory
creatTex()
createMisfortune()
getDanger()
NormalPlaceMaker
createPlace()
DangerousPlaceMaker
createPlace()
NormalPLaceFactory
creatTex()
createMisfortune()
getDanger()
DangerousPLaceFactory
creatTex()
createMisfortune()
getDanger()
Tex
getTex()
LowTex
getTex()
HighTex
getTex()
Misfortune
getMisfortune()
NoMisfortune
getMisfortune()
Disease
getMisfortune()
Temple
Place
getTex()
localImapact()
misfortuneCase()
getName()
setTrap()
hasTrap()
removeTrap()
Start
getBonus()
StartAdapter
getTex()
localImpact()
misfortuneCase()
Forest
getTex()
localImpact()
misfortuneCase()
Desert
getTex()
localImpact()
misfortuneCase()
SnowMountain
getTex()
localImpact()
misfortuneCase()
Volcano
getTex()
localImpact()
misfortuneCase()
Adaptor pattern


Facade pattern
Templete Method Pattern
Player
budget
currentIdx
name
place
rollDice()
setTrap()
action()
localImpact()
misfortune()
checkBudget()
damegedByTrap()
1
2..*
1
GameManager
player
places
rollDice()
action()
GMFacade
gameOn()
cano
se()
Computer
rollDice()
setTrap()
User
rollDice()
setTrap()
Strategy Pattern
Dice
getNumber()
Singleton Pattern

Ⅱ. 본론

1. 추상 팩토리 패턴




```java
import misfortune.*;
import tex.*;

// 추상 팩토리 패턴
public interface PlaceFactory {
	public Tex creatTex();
	public Misfortune creatMisfortune();
	public String getDanger();
}
```

```java
import misfortune.*;
import tex.*;

//추상 팩토리 패턴
public class DangerousPlaceFactory implements PlaceFactory {

	@Override
	public Tex creatTex() {
		return new HighTex();
	}

	@Override
	public Misfortune creatMisfortune() {
		return new Disease();
	}

	@Override
	public String getDanger() {
		return "Dangerous";
	}

}
```

```java
import misfortune.*;
import tex.*;

//추상 팩토리 패턴
public class NormalPlaceFactory implements PlaceFactory {

	@Override
	public Tex creatTex() {
		return new LowTex();
	}

	@Override
	public Misfortune creatMisfortune() {
		return new NoMisfortune();
	}

	@Override
	public String getDanger() {
		return "Normal";
	}
}

// 추상 팩토리 패턴
public interface PlaceMaker {
	public Place createPlace(String name);
}

// 추상 팩토리 패턴
public class DangerousPlaceMaker implements PlaceMaker {

	@Override
	public Place createPlace(String name) {
		Place place = null;
		PlaceFactory placeFactory = new DangerousPlaceFactory();

		if(name.equals("Volcano")) {
			place = new Volcano(placeFactory);
		}
		else if(name.equals("Forest")) {
			place = new Forest(placeFactory);
		}
		else if(name.equals("Desert")) {
			place = new Desert(placeFactory);
		}
		else if(name.equals("SnowMountain")) {
			place = new SnowMountain(placeFactory);
		}

		return place;
	}

}

// 추상 팩토리 패턴
public class NormalPlaceMaker implements PlaceMaker {

	@Override
	public Place createPlace(String name) {
		Place place = null;
		PlaceFactory placeFactory = new NormalPlaceFactory();

		if(name.equals("Volcano")) {
			place = new Volcano(placeFactory);
		}
		else if(name.equals("Forest")) {
```

```
                        place = new Forest(placeFactory);
                }
                else if(name.equals("Desert")) {
                        place = new Desert(placeFactory);
                }
                else if(name.equals("SnowMountain")) {
                        place = new SnowMountain(placeFactory);
                }

                return place;
        }

}
```

```
import localImpact.LocalImpact;
import misfortune.Misfortune;
import tex.Tex;

public abstract class Place {
        protected boolean trap;
        protected Misfortune mf;
        protected Tex tex;
        protected LocalImpact lm;
        protected PlaceFactory pf;

        public Place() {
                trap = false;
        }

        public abstract int getTex();
        public abstract String localImpact();
        public abstract int misfortuneCase();
        public abstract String getName();

        public void setTrap() {
                this.trap = true;
        }
        public boolean hasTrap() {
                return trap;
        }
        public void removeTrap() {
                this.trap = false;
        }
}
```

```
import localImpact.*;

//추상 팩토리 패턴
public class Volcano extends Place {

        public Volcano(PlaceFactory pf) {
                this.pf = pf;

                this.tex = this.pf.creatTex();
                this.mf = this.pf.creatMisfortune();
                this.lm = new VolcanoImpact();
        }

        @Override
        public int getTex() {
                return this.tex.getTex() + 20;
        }

        @Override
        public String localImpact() {
                return lm.impact();
```

```
	}

	@Override
	public int misfortuneCase() {
		return mf.getMisfortune();
	}

	@Override
	public String getName() {
		return pf.getDanger() + " Volcano";
	}

}

import localImpact.*;

//추상 팩토리 패턴
public class SnowMountain extends Place{

	public SnowMountain(PlaceFactory pf) {
		this.pf = pf;

		this.tex = this.pf.creatTex();
		this.mf = this.pf.creatMisfortune();
		this.lm = new SnowMountainImpact();
	}

	@Override
	public int getTex() {
		return this.tex.getTex() + 30;
	}

	@Override
	public String localImpact() {
		return lm.impact();
	}

	@Override
	public int misfortuneCase() {
		return mf.getMisfortune();
	}

	@Override
	public String getName() {
		return pf.getDanger() + " Snow Mountain";
	}

}

import localImpact.*;

//추상 팩토리 패턴
public class Forest extends Place {

	public Forest(PlaceFactory pf) {
		this.pf = pf;

		this.tex = this.pf.creatTex();
		this.mf = this.pf.creatMisfortune();
		lm = new ForestImpact();
	}

	@Override
	public int getTex() {
		return this.tex.getTex() - 10;
	}
```

```
	@Override
	public String localImpact() {
		return lm.impact();
	}

	@Override
	public int misfortuneCase() {
		return mf.getMisfortune();
	}

	@Override
	public String getName() {
		return pf.getDanger() + " Forest";
	}

}
```

```
import localImpact.*;

//추상 팩토리 패턴
public class Desert extends Place{

	public Desert(PlaceFactory pf) {
		this.pf = pf;

		this.tex = this.pf.creatTex();
		this.mf = this.pf.creatMisfortune();
		this.lm = new DesertImpact();
	}

	@Override
	public int getTex() {
		return this.tex.getTex();
	}

	@Override
	public String localImpact() {
		return lm.impact();
	}

	@Override
	public int misfortuneCase() {
		return mf.getMisfortune();
	}

	@Override
	public String getName() {
		return pf.getDanger() + " Desert";
	}

}
```

```
package misfortune;

//추상 팩토리 패턴
public class Disease implements Misfortune {
	@Override
	public int getMisfortune() {
		return 80;
	}

}
```

```
package misfortune;
```

```java
// 추상 팩토리 패턴
public interface Misfortune {
	public int getMisfortune();
}
```

```java
package misfortune;

//추상 팩토리 패턴
public class NoMisfortune implements Misfortune {
	@Override
	public int getMisfortune() {
		return 0;
	}

}
```

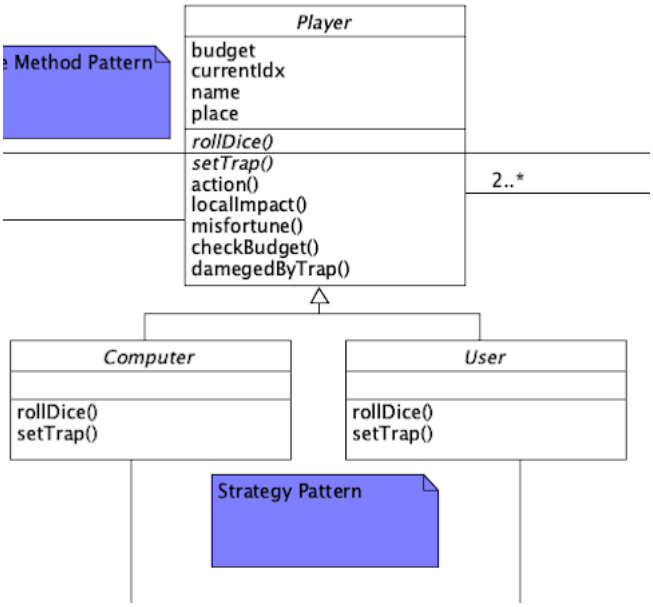
```java
package tex;

//추상 팩토리 패턴
public class HighTex implements Tex {
	@Override
	public int getTex() {
		return 40;
	}

}
```

```java
package tex;

//추상 팩토리 패턴
public class LowTex implements Tex {
	@Override
	public int getTex() {
		return 20;
	}

}
```

```java
package tex;

//추상 팩토리 패턴
public interface Tex {
	public int getTex();
}
```

기본적으로 만들 수 있는 지역은 Volcano, SnowMountain, Desert, Forest 네 곳이다. 이 네 곳에서 위험한 지역, 평범한 지역이 다시 나뉜다. 위험한 지역은 통행료가 높고(HighTex), 운이 나빠 병에 걸린다(Disease). 평범한 지역은 통행료가 낮고(LowTex) 병에 걸리지 않는다(NoMisfortune). 이를 DangerousPlaceFactory와 NormalPlaceFactory에서 각 지역에 할당해주며, 네 지역 중에서 어떤 지역을 만들고 싶은지는 DangerousPlaceMaker와 NormalPlaceMaker에서 정하고, 클래스에 맞는 팩토리와 연결해준다.

2. 스트라티지 패턴




```
// 템플릿 메소드, 스트라티지 패턴 적용

public abstract class Player {
	private int budget;
	private int currentIdx;
	protected String name;
	protected Place place;

	public Player() {
		this.budget = 500;                                    // 초기 소지금 500
		this.currentIdx = 0;
	}

	public abstract int rollDice();      // 스트라티지 패턴
	public abstract void setTrap();

	// 템플릿 메소드 패턴
	public boolean action() {

		damegedByTrap();
		setTrap();
		localImpact();
		misfortune();
		// 훅
		if( checkBudget() ) {
			System.out.println(this.getName() + "가 패배했습니다.");
			return false;
		}
		System.out.println();

		return true;
	}

	public void localImpact() {
		String impact = this.place.localImpact();

		// 다음 턴에 효과가 적용됨
		if(impact.equals("Forset")) {
```

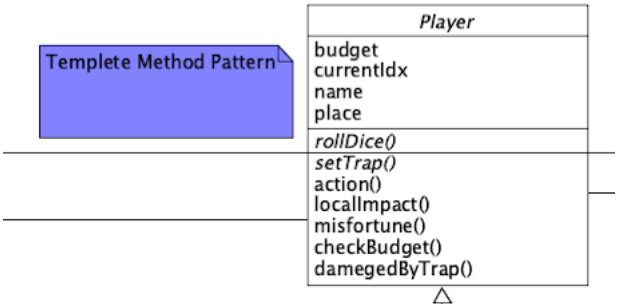
```
                        System.out.println("숲에서 열매를 주워 팔았습니다. (+ 20)");
                        this.setBudget(this.getBudget() + 20);
                }
                else if(impact.equals("Volcano")) {
                        System.out.println("화산에서 물을 사먹었습니다. (- 20)");
                }
                else if(impact.equals("Desert")) {
                        System.out.println("낙타를 타고 갑니다. (다음 턴 이동: 주사위
+1)");
                        this.setCurrentIdx(this.getCurrentIdx() + 1);
                }
                else if(impact.equals("SnowMountain")) {
                        System.out.println("추워서 움직임이 느려집니다. (다음 턴 이동:
주사위 -1)");
                        this.setCurrentIdx(this.getCurrentIdx() + 1);
                }
                else if(impact.equals("NoImpact")) {
                        System.out.println("시작 지점입니다.");
                }
        }

        public void misfortune() {
                int mf = this.place.misfortuneCase();
                if(mf > 0) {
                        System.out.println("병에 걸려 치료비를 소모했습니다.");
                        this.setBudget(this.getBudget() - mf);
                }
        }

        public boolean checkBudget() {
                this.setBudget(this.getBudget() - this.place.getTex());
                System.out.println("현재 자금: " + this.getBudget());
                if( this.getBudget() <= 0 ) {
                        return true;
                }
                else {
                        return false;
                }
        }

        public void damegedByTrap() {
                if( place.hasTrap() ) {
                        System.out.println(this.name + "가 함정에 타격");
                        this.budget = this.budget - 10;
                        place.removeTrap();
                }
        }

        public String getName() {
                return this.name;
        }


        public int getBudget() {
                return budget;
        }
        public void setBudget(int budget) {
                this.budget = budget;
        }

        public int getCurrentIdx() {
                return this.currentIdx;
```

```
	}
	public void setCurrentIdx(int idx) {
		this.currentIdx = idx;
	}

	public void setPlace(Place place) {
		this.place = place;
	}
	public Place getPlace() {
		return this.place;
	}

}
import java.util.Scanner;

//템플릿 메소드, 스트라티지 패턴 적용
public class User extends Player{
	private Scanner scan;

	public User() {
		this.name = "User";
		scan = new Scanner(System.in);
	}

	// 템플릿 메소드에서 내용이 다른 절차
	@Override
	public void setTrap() {
		if( !place.hasTrap() ) {
			System.out.print("트랩을 설치하겠습니까? (y): ");
			if( scan.nextLine() == "y" )  {
				place.setTrap();
			}
		}
	}

	// 스트라티지 패턴
	@Override
	public int rollDice() {
		System.out.print("주사위를 굴리려면 아무거나 입력");
		scan.nextLine();
		return Dice.getNumber();
	}

}

// 템플릿 메소드, 스트라티지 패턴 적용
public class Computer extends Player {

	public Computer() {
		this.name = "Computer";
	}

	// 템플릿 메소드에서 내용이 다른 절차
	@Override
	public void setTrap() {
		if( !place.hasTrap() ) {
			place.setTrap();
		}
	}

	// 스트라티지 패턴
	@Override
```

```
        public int rollDice() {
                return Dice.getNumber();
        }
}
```

Player 클래스에서는 rollDice()의 작동 방식을 서브 클래스마다 다르게 해주어야 한다. 사람은 주사위를 굴리는 행위를 해야 하지만, 컴퓨터는 자동으로 굴리도록 설정을 해야 하기 때문이다. setTrap()의 경우 컴퓨터와 유저가 다른 방식으로 작동하지만, 상위 클래스에 존재하는 action() 메소드 내부에 절차로 존재하여, 템플릿 메소드 패턴에 포함시켰다.

3. 템플릿 메소드 패턴




```
// 템플릿 메소드, 스트라티지 패턴 적용

public abstract class Player {
        private int budget;
        private int currentIdx;
        protected String name;
        protected Place place;

        public Player() {
                this.budget = 500;                                      // 초기 소지금 500
                this.currentIdx = 0;
        }

        public abstract int rollDice();      // 스트라티지 패턴
        public abstract void setTrap();

        // 템플릿 메소드 패턴
        public boolean action() {

                damegedByTrap();
                setTrap();
                localImpact();
                misfortune();
                // 훅
                if( checkBudget() ) {
                        System.out.println(this.getName() + "가 패배했습니다.");
                        return false;
                }
                System.out.println();

                return true;
        }

        public void localImpact() {
                String impact = this.place.localImpact();
```

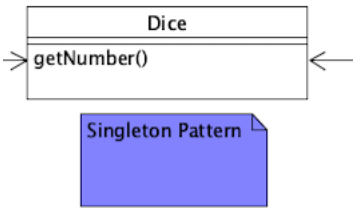
```
                // 다음 턴에 효과가 적용됨
                if(impact.equals("Forset")) {
                        System.out.println("숲에서 열매를 주워 팔았습니다. (+ 20)");
                        this.setBudget(this.getBudget() + 20);
                }
                else if(impact.equals("Volcano")) {
                        System.out.println("화산에서 물을 사먹었습니다. (- 20)");
                }
                else if(impact.equals("Desert")) {
                        System.out.println("낙타를 타고 갑니다. (다음 턴 이동: 주사위
+1)");
                        this.setCurrentIdx(this.getCurrentIdx() + 1);
                }
                else if(impact.equals("SnowMountain")) {
                        System.out.println("추워서 움직임이 느려집니다. (다음 턴 이동:
주사위 -1)");
                        this.setCurrentIdx(this.getCurrentIdx() + 1);
                }
                else if(impact.equals("NoImpact")) {
                        System.out.println("시작 지점입니다.");
                }
        }

        public void misfortune() {
                int mf = this.place.misfortuneCase();
                if(mf > 0) {
                        System.out.println("병에 걸려 치료비를 소모했습니다.");
                        this.setBudget(this.getBudget() - mf);
                }
        }

        public boolean checkBudget() {
                this.setBudget(this.getBudget() - this.place.getTex());
                System.out.println("현재 자금: " + this.getBudget());
                if( this.getBudget() <= 0 ) {
                        return true;
                }
                else {
                        return false;
                }
        }

        public void damegedByTrap() {
                if( place.hasTrap() ) {
                        System.out.println(this.name + "가 함정에 타격");
                        this.budget = this.budget - 10;
                        place.removeTrap();
                }
        }

        public String getName() {
                return this.name;
        }


        public int getBudget() {
                return budget;
        }
        public void setBudget(int budget) {
                this.budget = budget;
```

```
	}

	public int getCurrentIdx() {
		return this.currentIdx;
	}
	public void setCurrentIdx(int idx) {
		this.currentIdx = idx;
	}

	public void setPlace(Place place) {
		this.place = place;
	}
	public Place getPlace() {
		return this.place;
	}

}

import java.util.Scanner;

//템플릿 메소드, 스트라티지 패턴 적용
public class User extends Player{
	private Scanner scan;

	public User() {
		this.name = "User";
		scan = new Scanner(System.in);
	}

	// 템플릿 메소드에서 내용이 다른 절차
	@Override
	public void setTrap() {
		if( !place.hasTrap() ) {
			System.out.print("트랩을 설치하겠습니까? (y): ");
			if( scan.nextLine() == "y" )  {
				place.setTrap();
			}
		}
	}

	// 스트라티지 패턴
	@Override
	public int rollDice() {
		System.out.print("주사위를 굴리려면 아무거나 입력");
		scan.nextLine();
		return Dice.getNumber();
	}

}

// 템플릿 메소드, 스트라티지 패턴 적용
public class Computer extends Player {

	public Computer() {
		this.name = "Computer";
	}

	// 템플릿 메소드에서 내용이 다른 절차
	@Override
	public void setTrap() {
		if( !place.hasTrap() ) {
			place.setTrap();
```

```
                }
        }

        // 스트라티지 패턴
        @Override
        public int rollDice() {
                return Dice.getNumber();
        }
}
```

플레이어가 주사위를 굴리고 난 후의 행동들을 action() 메소드에 정의하였다. 트랩이 존재하면 데미지를 받고 트랩을 해제하는 damegedByTrap(), 트랩이 없을 경우 유저는 트랩 설치 여부를 묻고 트랩을 설치하고, 컴퓨터는 자동으로 설치하는 setTrap(), 지역 영향을 받는 localImpact(), 운이 나쁜 경우 돈을 그만큼 잃는 misfortune(), 그리고 혹으로 현재 자금이 0 이하가 되면 패배 처리를 하도록 하는 checkBudget()이 캡슐화되어 존재한다. rollDice()가 같이 캡슐화되지 않은 이유는, 전체 게임을 관리해줄 다른 클래스, GamaManager가 존재하며 해당 클래스에서 게임 진행 상황을 관리하고, 출력해줘야 하는데, rollDice()가 분리되어야 더 효율적으로 작동 가능하다. 만약 gameManager에서 그러한 작업을 해주지 않는다면, 플레이어 객체가 존재하는 모든 지역의 정보를 알아야 한다. 이 경우, 플레이어에게 주어진 책임이 너무 많아져 분리되었다.

4. 싱글턴 패턴

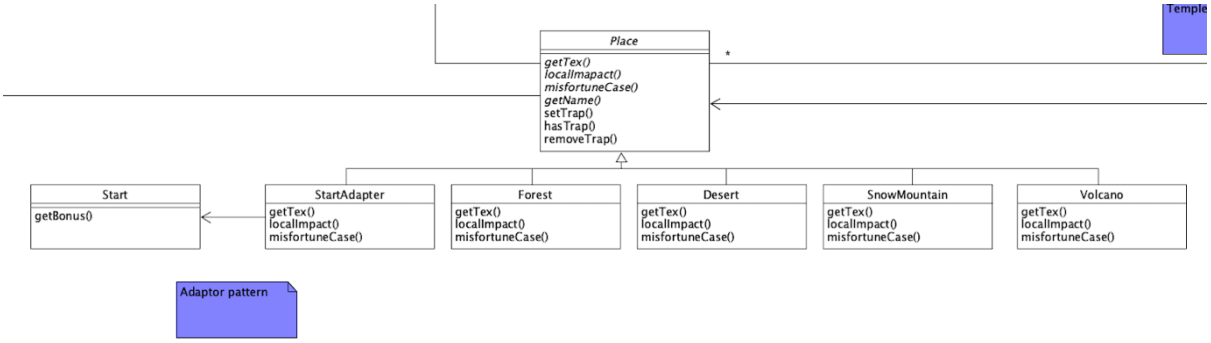



```
import java.util.Random;

// 싱글턴 패턴
public class Dice {
        private volatile static Random random = null;

        public static int getNumber() {
                if(random == null) {
                        synchronized(Dice.class) {
                                if(random == null) {
                                        random = new Random();
                                }
                        }
                }

                return random.nextInt(6) + 1;
        }
}
```

```
<terminated> Main (7) [Java Application] /L
User의 주사위: 5, 도착 장소: Normal
현재 자금: 490
트랩을 설치하겠습니까? (y):

Computer의 주사위: 5, 도착 장소: Nor
현재 자금: 490
Computer가 함정에 타격

User의 주사위: 1, 도착 장소: Normal
현재 자금: 480
트랩을 설치하겠습니까? (y):

Computer의 주사위: 1, 도착 장소: Nor
현재 자금: 470
Computer가 함정에 타격
```

초기에는 싱글턴 패턴이 적용되지 않고, Dice() 객체를 만들어 해당 객체의 getNumer() 메소드로 주사위 값을 얻었다. 하지만, 유저와 컴퓨터가 같은 턴에 굴리면 위 이미지처럼 같은 주사위 값을 가지는 문제가 발생하였다. 다른 시드를 준다면 해결될 문제지만, 기본적으로 보드 게임에서는 같은 주사위를 플레이어들이 같이 사용한다. 그러한 점을 고려해서, 코드 구현 상에서도 같은 주사위를 사용하도록 했다. 랜덤 객체를 한 번만 생성되도록 하고, 해당 객체에서 난수를 생성하여 리턴하도록 하였다.

5. 어댑터 패턴



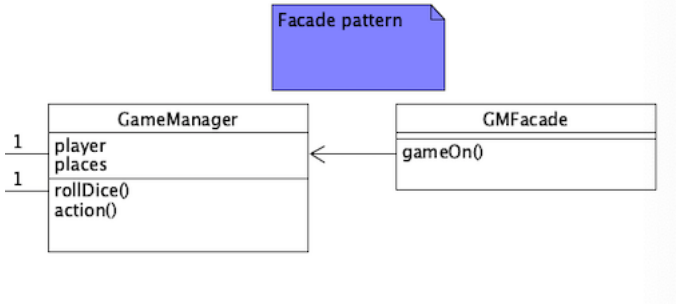

```java
// 어댑터 패턴
public class Start {
	public int getBonus() {
		return 150;
	}
}
```

```java
import localImpact.*;
import misfortune.*;

// 어댑터 패턴
public class StartAdapter extends Place{
	private Start start;
```

```
	private LocalImpact lm = new NoImpact();
	private Misfortune mf = new NoMisfortune();

	public StartAdapter(Start start) {
		this.start = start;
	}

	@Override
	public int getTex() {
		return -start.getBonus();
	}

	@Override
	public String localImpact() {
		return lm.impact();
	}

	@Override
	public int misfortuneCase() {
		return mf.getMisfortune();
	}

	@Override
	public String getName() {
		return "Start Point";
	}

}
```

Start 지점은, 기존 Volcano, Desert, Forest, Snow Mountain 네 지역이 완성된 이후 추가된 지역이다. 추가된 이유는 한 바퀴 돌고 시작 지점에 정확히 안착하면 보너스를 주는 기능을 추가하고 싶었기 때문이다. 통행료를 걷지 않고 오히려 보너스를 받는 특수한 지형으로, getTex() 메소드 대신 getBonnus() 메소드를 가지도록 하는 것이 목적이다. 그래서 Start 클래스를 따로 만들고, 어댑터에서 getBonus()와 getTex()를 연결하는 방식으로 구현하였다.

6. 파사드 패턴




```
import java.util.ArrayList;

public class GameManager {
	private ArrayList<Player> players;
	private ArrayList<Place> places;
	private Player player;
	private Place place;

	public GameManager() {
		this.players = new ArrayList<Player>();
		this.places = new ArrayList<Place>();
```

```
		players.add(new User());
		players.add(new Computer());

		StartAdapter sa = new StartAdapter(new Start());
		places.add(sa);

		PlaceMaker pm = new NormalPlaceMaker();
		for(int i = 0; i < 9; i++) {
			places.add(pm.createPlace("Forest"));
		}
		pm = new DangerousPlaceMaker();
		places.add(pm.createPlace("Forest"));

		pm = new NormalPlaceMaker();
		for(int i = 0; i < 9; i++) {
			places.add(pm.createPlace("Volcano"));
		}
		pm = new DangerousPlaceMaker();
		places.add(pm.createPlace("Volcano"));

		pm = new NormalPlaceMaker();
		for(int i = 0; i < 9; i++) {
			places.add(pm.createPlace("Desert"));
		}
		pm = new DangerousPlaceMaker();
		places.add(pm.createPlace("Desert"));

		pm = new NormalPlaceMaker();
		for(int i = 0; i < 9; i++) {
			places.add(pm.createPlace("SnowMountain"));
		}
		pm = new DangerousPlaceMaker();
		places.add(pm.createPlace("SnowMountain"));
	}

	public boolean action() {
		int size = players.size();

		for( int i = 0; i < size; i++ ) {
			player = players.get(i);

			// 주사위 굴리기
			this.rollDice();

			if(!player.action()) return false;

			try {
				Thread.sleep(800);
			} catch(InterruptedException e) {
				System.out.println(e.getMessage());
			}

		}

		return true;

	}

	public void rollDice() {
		int num = player.rollDice();
		player.setCurrentIdx((player.getCurrentIdx() + num) %
places.size());
		place = places.get(player.getCurrentIdx());
		player.setPlace(place);
		System.out.print(player.getName() + "의 주사위: " + num);
```

```
                int money = place.getTex();
                if(money < 0) {
                        System.out.println(", 도착 장소: " + place.getName() + ",
보너스" + (-place.getTex()) +", 넘버: " + player.getCurrentIdx());
                }
                else {
                        System.out.println(", 도착 장소: " + place.getName() + ",
통행료" + place.getTex() +", 넘버: " + player.getCurrentIdx());
                }
        }


}

// 파사드 패턴
public class GMFacade {
        GameManager gm;

        public GMFacade() {
                gm = new GameManager();
        }

        public void gameOn() {
                while(gm.action());
        }
}

public class Main {

        public static void main(String[] args) {
                GMFacade facade = new GMFacade();
                facade.gameOn();
        }

}
```

보드 게임을 하기 위해서 GameManager 클래스에서 플레이어들, 장소들을 만들고 그에 관한 작업들을 관리한다. 그리고 그 관리하는 행동들이 담긴 action()을 무한 반복하면 게임이 진행된다. 그러한 행위를 GMFacade를 만들어 넣어주고, gameOn()이라는 메소드라 한다. 결국 Main 에서는 GMfacade의 gameOn() 만 접근하여 플레이어, 장소, 주사위 등 다른 시스템들을 몰라도, action()의 무한 반복을 몰라도 간편하게 게임을 실행할 수 있게 된다.

Ⅲ. 결과 및 전체 코드

1. 결과

```
현재 자금: 120

Computer의 주사위: 1, 도착 장소: Dangerous Desert, 통행료40, 넘버: 30
낙타를 타고 갑니다. (다음 턴 이동: 주사위 +1)
병에 걸려 치료비를 소모했습니다.
현재 자금: 80

주사위를 굴리려면 아무거나 입력
User의 주사위: 6, 도착 장소: Dangerous Forest, 통행료30, 넘버: 10
User가 함정에 타격
트랩을 설치하겠습니까? (y): y
병에 걸려 치료비를 소모했습니다.
현재 자금: 0
User가 패배했습니다.
```

실행하면 주사위 넘버, 지역 특수 효과, 현재 자금, 자산 체크와 같은 작업들이 제대로 행해지고 있는 것을 볼 수 있다. 스타트 지점과 화산, 숲, 사막, 설산을 지나며 돈을 먼저 다 쓴 사람이 패배하며, 위 실행 결과에서는 컴퓨터가 승리하고 유저가 패배하고 게임이 끝났다. 추상 팩토리 패턴, 스트라티지 패턴, 템플릿 메소드 패턴, 싱글턴 패턴, 어댑터 패턴, 파사드 패턴 총 6가지 디자인 패턴이 적용되었다.

초기 계획은 추상 팩토리 패턴, 스트라티지 패턴, 템플릿 메소드 패턴 세 가지를 사용하는 것이었다. 하지만 실제로 구현을 하다 보니 디자인 패턴이 적용되면 문제가 깔끔하게 해결되는 경우가 많아, 싱글턴 패턴, 어댑터 패턴, 파사드 패턴까지 총 6가지 디자인 패턴을 사용하게 되었다. 특히, Dice 클래스에 사용된 싱글턴 패턴을 활용하여 문제를 해결한 것이 그러한 경우이다.

이번 과제를 하며 느낀 점 중 하나는, 클래스 다이어그램 같은 계획의 중요성이다. 물론 구현하면서 처음에 구상한 것과는 클래스 다이어그램이 조금 달라지기는 했지만, 어떤 클래스들이, 어떤 메소드들이 필요한지 대략적인 가이드를 만들고 시작하니 에러가 발생하는 것도 거의 없었으며, 굉장히 수월하게 코드를 작성할 수 있었다.

2. 깃허브

https://github.com/Lagooneng/designpattern_boardgame

3. 전체 코드

```
// 템플릿 메소드, 스트라티지 패턴 적용
public class Computer extends Player {

	public Computer() {
		this.name = "Computer";
	}

	// 템플릿 메소드에서 내용이 다른 절차
	@Override
	public void setTrap() {
		if( !place.hasTrap() ) {
			place.setTrap();
		}
	}

	// 스트라티지 패턴
	@Override
	public int rollDice() {
		return Dice.getNumber();
	}
}
```

```
import misfortune.*;
import tex.*;

//추상 팩토리 패턴
public class DangerousPlaceFactory implements PlaceFactory {

	@Override
	public Tex creatTex() {
		return new HighTex();
	}

	@Override
	public Misfortune creatMisfortune() {
		return new Disease();
	}

	@Override
	public String getDanger() {
		return "Dangerous";
	}

}
```

```
// 추상 팩토리 패턴
public class DangerousPlaceMaker implements PlaceMaker {

	@Override
	public Place createPlace(String name) {
		Place place = null;
		PlaceFactory placeFactory = new DangerousPlaceFactory();

		if(name.equals("Volcano")) {
			place = new Volcano(placeFactory);
		}
		else if(name.equals("Forest")) {
			place = new Forest(placeFactory);
		}
		else if(name.equals("Desert")) {
			place = new Desert(placeFactory);
		}
```

```
                else if(name.equals("SnowMountain")) {
                        place = new SnowMountain(placeFactory);
                }

                return place;
        }

}
```

```
import localImpact.*;

//추상 팩토리 패턴
public class Desert extends Place{

        public Desert(PlaceFactory pf) {
                this.pf = pf;

                this.tex = this.pf.creatTex();
                this.mf = this.pf.creatMisfortune();
                this.lm = new DesertImpact();
        }

        @Override
        public int getTex() {
                return this.tex.getTex();
        }

        @Override
        public String localImpact() {
                return lm.impact();
        }

        @Override
        public int misfortuneCase() {
                return mf.getMisfortune();
        }

        @Override
        public String getName() {
                return pf.getDanger() + " Desert";
        }

}
```

```
import java.util.Random;

// 싱글턴 패턴
public class Dice {
        private volatile static Random random = null;

        public static int getNumber() {
                if(random == null) {
                        synchronized(Dice.class) {
                                if(random == null) {
                                        random = new Random();
                                }
                        }
                }

                return random.nextInt(6) + 1;
        }
}
```

```
import localImpact.*;
```

```java
//추상 팩토리 패턴
public class Forest extends Place {

	public Forest(PlaceFactory pf) {
		this.pf = pf;

		this.tex = this.pf.creatTex();
		this.mf = this.pf.creatMisfortune();
		lm = new ForestImpact();
	}

	@Override
	public int getTex() {
		return this.tex.getTex() - 10;
	}

	@Override
	public String localImpact() {
		return lm.impact();
	}

	@Override
	public int misfortuneCase() {
		return mf.getMisfortune();
	}

	@Override
	public String getName() {
		return pf.getDanger() + " Forest";
	}

}

import java.util.ArrayList;

public class GameManager {
	private ArrayList<Player> players;
	private ArrayList<Place> places;
	private Player player;
	private Place place;

	public GameManager() {
		this.players = new ArrayList<Player>();
		this.places = new ArrayList<Place>();

		players.add(new User());
		players.add(new Computer());

		StartAdapter sa = new StartAdapter(new Start());
		places.add(sa);

		PlaceMaker pm = new NormalPlaceMaker();
		for(int i = 0; i < 9; i++) {
			places.add(pm.createPlace("Forest"));
		}
		pm = new DangerousPlaceMaker();
		places.add(pm.createPlace("Forest"));

		pm = new NormalPlaceMaker();
		for(int i = 0; i < 9; i++) {
			places.add(pm.createPlace("Volcano"));
		}
		pm = new DangerousPlaceMaker();
		places.add(pm.createPlace("Volcano"));

		pm = new NormalPlaceMaker();
```

```
                for(int i = 0; i < 9; i++) {
                        places.add(pm.createPlace("Desert"));
                }
                pm = new DangerousPlaceMaker();
                places.add(pm.createPlace("Desert"));

                pm = new NormalPlaceMaker();
                for(int i = 0; i < 9; i++) {
                        places.add(pm.createPlace("SnowMountain"));
                }
                pm = new DangerousPlaceMaker();
                places.add(pm.createPlace("SnowMountain"));
        }

        public boolean action() {
                int size = players.size();

                for( int i = 0; i < size; i++ ) {
                        player = players.get(i);

                        // 주사위 굴리기
                        this.rollDice();

                        if(!player.action()) return false;

                        try {
                                Thread.sleep(800);
                        } catch(InterruptedException e) {
                                System.out.println(e.getMessage());
                        }

                }

                return true;

        }

        public void rollDice() {
                int num = player.rollDice();
                player.setCurrentIdx((player.getCurrentIdx() + num) %
places.size());
                place = places.get(player.getCurrentIdx());
                player.setPlace(place);
                System.out.print(player.getName() + "의 주사위: " + num);

                int money = place.getTex();
                if(money < 0) {
                        System.out.println(", 도착 장소: " + place.getName() + ",
보너스" + (-place.getTex()) +", 넘버: " + player.getCurrentIdx());
                }
                else {
                        System.out.println(", 도착 장소: " + place.getName() + ",
통행료" + place.getTex() +", 넘버: " + player.getCurrentIdx());
                }
        }


}

// 파사드 패턴
public class GMFacade {
        GameManager gm;
```

```
	public GMFacade() {
		gm = new GameManager();
	}

	public void gameOn() {
		while(gm.action());
	}
}

public class Main {

	public static void main(String[] args) {
		GMFacade facade = new GMFacade();
		facade.gameOn();
	}

}
```

```
import misfortune.*;
import tex.*;

//추상 팩토리 패턴
public class NormalPlaceFactory implements PlaceFactory {

	@Override
	public Tex creatTex() {
		return new LowTex();
	}

	@Override
	public Misfortune creatMisfortune() {
		return new NoMisfortune();
	}

	@Override
	public String getDanger() {
		return "Normal";
	}
}
```

```
// 추상 팩토리 패턴
public class NormalPlaceMaker implements PlaceMaker {

	@Override
	public Place createPlace(String name) {
		Place place = null;
		PlaceFactory placeFactory = new NormalPlaceFactory();

		if(name.equals("Volcano")) {
			place = new Volcano(placeFactory);
		}
		else if(name.equals("Forest")) {
			place = new Forest(placeFactory);
		}
		else if(name.equals("Desert")) {
			place = new Desert(placeFactory);
		}
		else if(name.equals("SnowMountain")) {
			place = new SnowMountain(placeFactory);
		}

		return place;
	}

}
```

```java
import localImpact.LocalImpact;
import misfortune.Misfortune;
import tex.Tex;

public abstract class Place {
	protected boolean trap;
	protected Misfortune mf;
	protected Tex tex;
	protected LocalImpact lm;
	protected PlaceFactory pf;

	public Place() {
		trap = false;
	}

	public abstract int getTex();
	public abstract String localImpact();
	public abstract int misfortuneCase();
	public abstract String getName();

	public void setTrap() {
		this.trap = true;
	}
	public boolean hasTrap() {
		return trap;
	}
	public void removeTrap() {
		this.trap = false;
	}
}

import misfortune.*;
import tex.*;

// 추상 팩토리 패턴
public interface PlaceFactory {
	public Tex creatTex();
	public Misfortune creatMisfortune();
	public String getDanger();
}

// 추상 팩토리 패턴
public interface PlaceMaker {
	public Place createPlace(String name);
}

// 템플릿 메소드, 스트라티지 패턴 적용

public abstract class Player {
	private int budget;
	private int currentIdx;
	protected String name;
	protected Place place;

	public Player() {
		this.budget = 500;                                    // 초기 소지금 500
		this.currentIdx = 0;
	}

	public abstract int rollDice();      // 스트라티지 패턴
	public abstract void setTrap();

	// 템플릿 메소드 패턴
	public boolean action() {
```

```
                damegedByTrap();
                setTrap();
                localImpact();
                misfortune();
                // 혹
                if( checkBudget() ) {
                        System.out.println(this.getName() + "가 패배했습니다.");
                        return false;
                }
                System.out.println();

                return true;
        }

        public void localImpact() {
                String impact = this.place.localImpact();

                // 다음 턴에 효과가 적용됨
                if(impact.equals("Forset")) {
                        System.out.println("숲에서 열매를 주워 팔았습니다. (+ 20)");
                        this.setBudget(this.getBudget() + 20);
                }
                else if(impact.equals("Volcano")) {
                        System.out.println("화산에서 물을 사먹었습니다. (- 20)");
                }
                else if(impact.equals("Desert")) {
                        System.out.println("낙타를 타고 갑니다. (다음 턴 이동: 주사위
+1)");
                        this.setCurrentIdx(this.getCurrentIdx() + 1);
                }
                else if(impact.equals("SnowMountain")) {
                        System.out.println("추워서 움직임이 느려집니다. (다음 턴 이동:
주사위 -1)");
                        this.setCurrentIdx(this.getCurrentIdx() + 1);
                }
                else if(impact.equals("NoImpact")) {
                        System.out.println("시작 지점입니다.");
                }
        }

        public void misfortune() {
                int mf = this.place.misfortuneCase();
                if(mf > 0) {
                        System.out.println("병에 걸려 치료비를 소모했습니다.");
                        this.setBudget(this.getBudget() - mf);
                }
        }

        public boolean checkBudget() {
                this.setBudget(this.getBudget() - this.place.getTex());
                System.out.println("현재 자금: " + this.getBudget());
                if( this.getBudget() <= 0 ) {
                        return true;
                }
                else {
                        return false;
                }
        }

        public void damegedByTrap() {
```

```
                if( place.hasTrap() ) {
                        System.out.println(this.name + "가 함정에 타격");
                        this.budget = this.budget - 10;
                        place.removeTrap();
                }
        }

        public String getName() {
                return this.name;
        }


        public int getBudget() {
                return budget;
        }
        public void setBudget(int budget) {
                this.budget = budget;
        }

        public int getCurrentIdx() {
                return this.currentIdx;
        }
        public void setCurrentIdx(int idx) {
                this.currentIdx = idx;
        }

        public void setPlace(Place place) {
                this.place = place;
        }
        public Place getPlace() {
                return this.place;
        }

}

import localImpact.*;

//추상 팩토리 패턴
public class SnowMountain extends Place{

        public SnowMountain(PlaceFactory pf) {
                this.pf = pf;

                this.tex = this.pf.creatTex();
                this.mf = this.pf.creatMisfortune();
                this.lm = new SnowMountainImpact();
        }

        @Override
        public int getTex() {
                return this.tex.getTex() + 30;
        }

        @Override
        public String localImpact() {
                return lm.impact();
        }

        @Override
        public int misfortuneCase() {
                return mf.getMisfortune();
        }

        @Override
        public String getName() {
```

```
                return pf.getDanger() + " Snow Mountain";
        }

}

// 어댑터 패턴
public class Start {
        public int getBonus() {
                return 150;
        }
}

import localImpact.*;
import misfortune.*;

// 어댑터 패턴
public class StartAdapter extends Place{
        private Start start;
        private LocalImpact lm = new NoImpact();
        private Misfortune mf = new NoMisfortune();

        public StartAdapter(Start start) {
                this.start = start;
        }

        @Override
        public int getTex() {
                return -start.getBonus();
        }

        @Override
        public String localImpact() {
                return lm.impact();
        }

        @Override
        public int misfortuneCase() {
                return mf.getMisfortune();
        }

        @Override
        public String getName() {
                return "Start Point";
        }

}

import java.util.Scanner;

//템플릿 메소드, 스트라티지 패턴 적용
public class User extends Player{
        private Scanner scan;

        public User() {
                this.name = "User";
                scan = new Scanner(System.in);
        }

        // 템플릿 메소드에서 내용이 다른 절차
        @Override
        public void setTrap() {
                if( !place.hasTrap() ) {
                        System.out.print("트랩을 설치하겠습니까? (y): ");
                        if( scan.nextLine() == "y" )  {
```

```
                                place.setTrap();
                        }
                }
        }

        // 스트라티지 패턴
        @Override
        public int rollDice() {
                System.out.print("주사위를 굴리려면 아무거나 입력");
                scan.nextLine();
                return Dice.getNumber();
        }

}
```

```
import localImpact.*;

//추상 팩토리 패턴
public class Volcano extends Place {

        public Volcano(PlaceFactory pf) {
                this.pf = pf;

                this.tex = this.pf.creatTex();
                this.mf = this.pf.creatMisfortune();
                this.lm = new VolcanoImpact();
        }

        @Override
        public int getTex() {
                return this.tex.getTex() + 20;
        }

        @Override
        public String localImpact() {
                return lm.impact();
        }

        @Override
        public int misfortuneCase() {
                return mf.getMisfortune();
        }

        @Override
        public String getName() {
                return pf.getDanger() + " Volcano";
        }

}
```

```
package localImpact;

public class DesertImpact implements LocalImpact {
        @Override
        public String impact() {
                return "Desert";
        }

}
```

```
package localImpact;

public class ForestImpact implements LocalImpact {
        @Override
        public String impact() {
```

```
                return "Forest";
        }

}
```

```
package localImpact;

public interface LocalImpact {
        public String impact();
}
```

```
package localImpact;

public class NoImpact implements LocalImpact {

        @Override
        public String impact() {
                return "NoImpact";
        }

}
```

```
package localImpact;

public class SnowMountainImpact implements LocalImpact {

        @Override
        public String impact() {
                return "SnowMountain";
        }

}
```

```
package localImpact;

public class VolcanoImpact implements LocalImpact {
        @Override
        public String impact() {
                return "Volcano";
        }

}
```

```
package misfortune;

//추상 팩토리 패턴
public class Disease implements Misfortune {
        @Override
        public int getMisfortune() {
                return 80;
        }

}
```

```
package misfortune;

// 추상 팩토리 패턴
public interface Misfortune {
        public int getMisfortune();
}
```

```
package misfortune;

//추상 팩토리 패턴
public class NoMisfortune implements Misfortune {
        @Override
```

```
        public int getMisfortune() {
                return 0;
        }

}
```

```
package tex;

//추상 팩토리 패턴
public class HighTex implements Tex {
        @Override
        public int getTex() {
                return 40;
        }

}
```

```
package tex;

//추상 팩토리 패턴
public class LowTex implements Tex {
        @Override
        public int getTex() {
                return 20;
        }

}
```

```
package tex;

//추상 팩토리 패턴
public interface Tex {
        public int getTex();
}
```