Object - Oriented Programming

LAB #13. Design Patterns : Decorator, Observer

Design Pattern

- 종류

Creational	Structural	Behavioral
Abstract Factory Builder Factory Method Prototype **Singleton**	Adapter Bridge Composite **Decorator** Facade Flyweight Proxy	Chain of Responsibility Command Interpreter Iterator Mediator Memento **Observer** State Strategy Template Method Visitor

Decorator Pattern

- 객체의 결합을 통해 기능을 **동적**으로 유연하게 확장할 수 있게 하는 패턴

- 기본 기능에 추가할 수 있는 기능의 종류가 많은 경우에
  각 **추가 기능**을 Decorator클래스로 정의한 후
  필요한 Decorator 객체를 조합하여 추가 기능의 조합을 설계하는 방식

- 추상 클래스를 활용하여 구현하는 방식

- 사용 예시
  - 카페
  - File I/O

Decorator Pattern

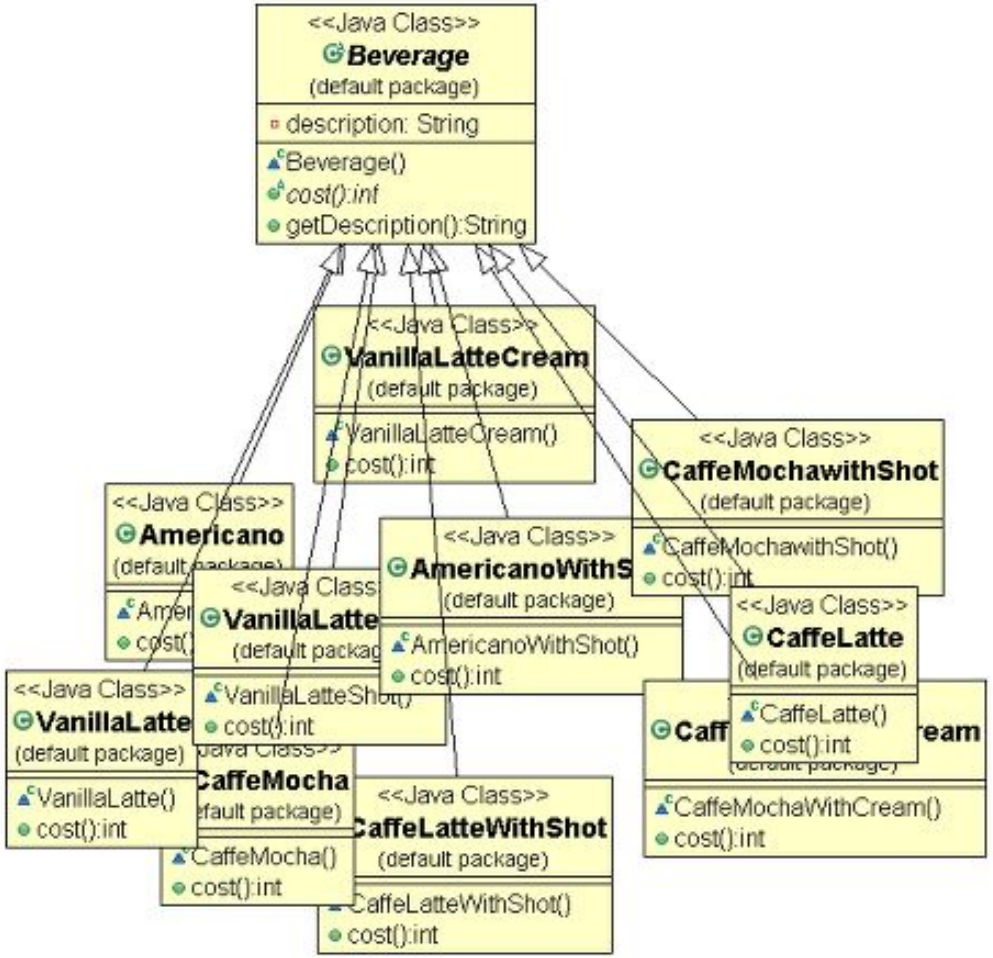
- 메뉴판에 손님이 주문할 수 있는 모든 주문을 적는다면?

  - 카페 모카
  - 카페 모카에 휘핑 크림
  - 카페 모카에 휘핑 크림에 자바칩 추가
  - 카페 모카에 휘핑 크림에 자바칩 추가에 샷 추가
  - 카페 모카에 자바칩
  - 카페 모카에 자바칩에 샷 추가
  - …

- 커피 종류를 하나 추가하게 되면?

Decorator Pattern

- 메뉴가 늘어날수록 정의할 클래스의 수도 기하 급수적으로 증가함



Decorator Pattern

- 메뉴판을 다음과 같이 하면?

< Beverage >

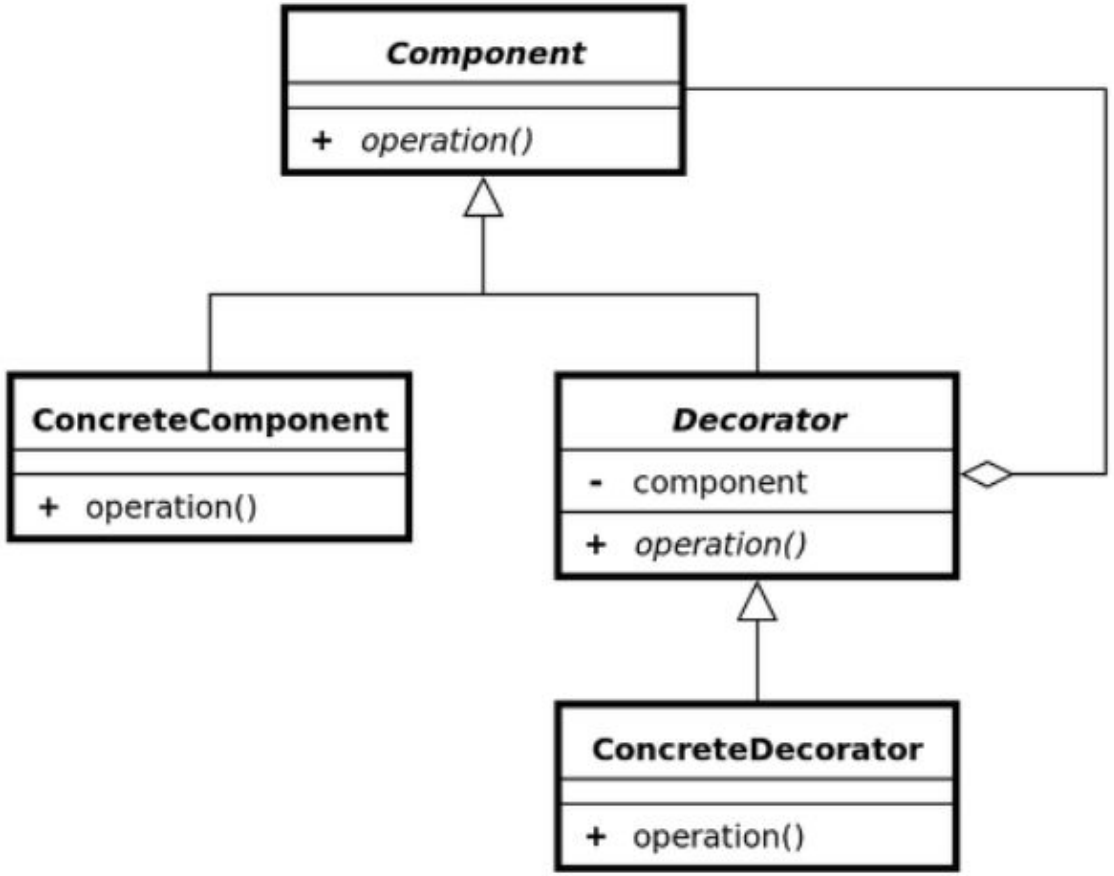
- 아메리카노
- 카페라떼
- 카페모카
- 바닐라라떼

< Condiment >

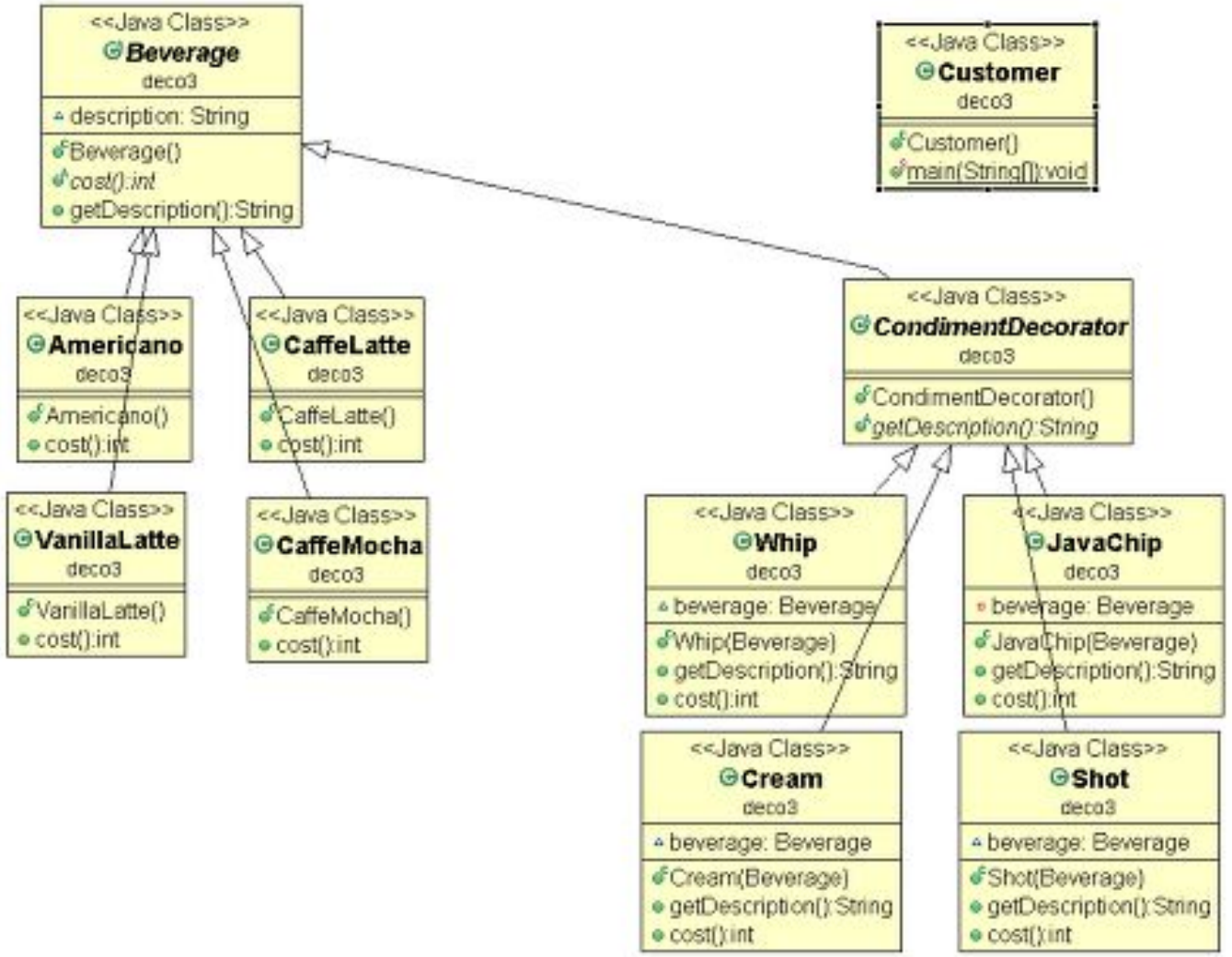
- 샷 추가
- 휘핑 크림
- 자바칩 추가

Decorator Pattern

- Decorator pattern의 기본적인 형태



Decorator Pattern Example



Decorator Pattern Example

- 가상 상위의 클래스
- 모든 음료는 이 클래스를 상속받음

```
public abstract class Beverage {

    String description = "no title";

    public abstract int cost();

    public String getDescription() {
        return description;
    }

}
```

Decorator Pattern Example

```
public class Americano extends Beverage {

    public Americano() {
        super();
        description = "Americano";
    }

    @Override
    public int cost() {
        // TODO Auto-generated method stub
        return 4000;
    }
}
```

Decorator Pattern Example

- 첨가물들은 CondimentDecorator 클래스를 상속받음
  - 이 클래스는 Beverage 클래스는 확장한 것
  - 첨가물들에게 getDescription 메소드를 재정의하도록 함.

```
public abstract class CondimentDecorator extends Beverage {

    public abstract String getDescription();

}
```

Decorator Pattern Example

```
public class Shot extends CondimentDecorator {
    Beverage beverage;

    public Shot(Beverage beverage) {
        super();
        this.beverage = beverage;
    }

    @Override
    public String getDescription() {
        // TODO Auto-generated method stub
        return beverage.getDescription() + ", Shot";
    }

    @Override
    public int cost() {
        // TODO Auto-generated method stub
        return beverage.cost() + 400;
    }
}
```

Decorator Pattern Example

- 주문하는 Customer 클래스

```
public class Customer {

    public static void main(String[] args) {

        Beverage beverage = new Americano();
        beverage = new Shot(beverage);
        beverage = new Shot(beverage);

        System.out.println("Menu: " + beverage.getDescription());
        System.out.println("Price: " + beverage.cost());
    }
}
```

- 결과

```
Menu: Americano, Shot, Shot
Price: 4800
```

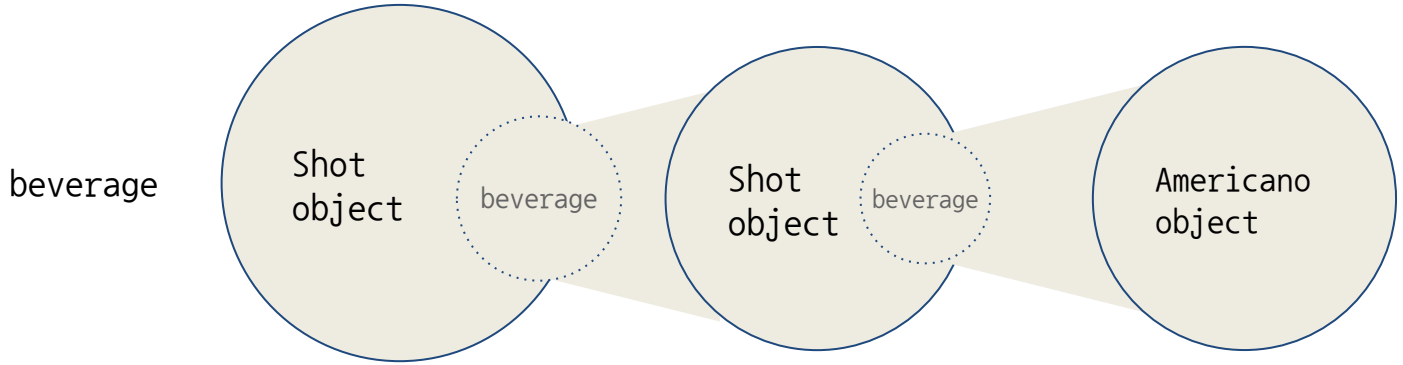
Decorator Pattern Example

```
public class Customer {

    public static void main(String[] args) {

        Beverage beverage = new Americano();
        beverage = new Shot(beverage);
        beverage = new Shot(beverage);

        System.out.println("Menu: " + beverage.getDescription());
        System.out.println("Price: " + beverage.cost());
    }
}
```



Observer Pattern

- 한 객체의 상태가 바뀔 경우, 다른 객체들에게 알려주는 패턴

- Swing API의 Listener Method로 쓰임

- JAVA에 내장되어 있음
  - java.util.Observer
  - java.util.Observable

Observer Pattern

- Subject
  - 상태가 변경되면 observer들에게 알림메시지를 보낸다.
  - 메시지와 함께 변경된 상태에 대한 정보를 제공

- Observers
  - Subject의 상태 변경에 대해 통지를 받는 객체들

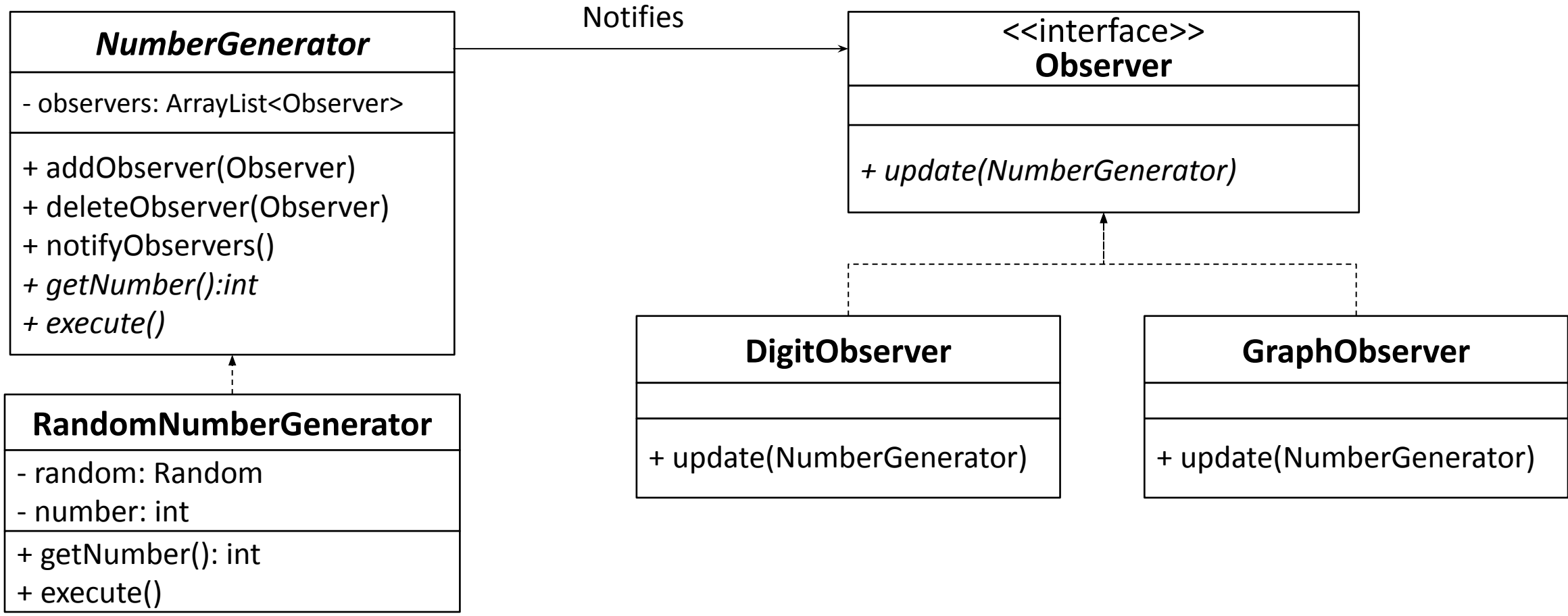
Observer Pattern 실습

- NumberGenerator에서 생성된 값을 Digital과 Graph로 표시하기
  - NumberGenerator에서 값을 생성하면
    Digital과 Graph로 화면에 출력하는 어플리케이션

  - 목표
    - NumberGenerator에서 랜덤 값을 생성하면,
      화면에 Digital과 Graph로 표시되도록 한다.
    - (Ex. Digital : 10, Graph : **********)

Observer Pattern 실습

- 예제 프로그램의 클래스 다이어그램



Observer Pattern 실습

- NumberGenerator 클래스
  - 등록되어 있는 Observer에게 최신 값을 update 해야 한다. (update() 호출)
- RandomNumberGenerator 클래스
  - number의 값이 바뀔 때 마다 notify 해야 한다. (notifyObservers() 호출)
- DigitObserver 클래스
  - 생성자 (인자 : NumberGenerator)
  - 생성자 내에서 addObserver()를 호출하여 Observer 객체 등록
- GraphObserver 클래스
  - 생성자 (인자 : NumberGenerator)
  - 생성자 내에서 addObserver()를 호출하여 Observer 객체 등록

Observer Pattern 실습

- NumberGenerator Class

```
import java.util.ArrayList;

public abstract class NumberGenerator {
     private ArrayList<Observer> observers = new ArrayList<Observer>();

     public abstract int getNumber();
     public abstract void execute();

     public void addObserver(Observer observer) {
          observers.add(observer);
     }

     public void deleteObserver(Observer observer) {
          observers.remove(observer);
     }

     public void notifyObservers() {
          for(int i = 0; i < observers.size(); i++) {
               observers.get(i).update(this);
          }
     }
}
```

Observer Pattern 실습

- RandomNumberGenerator Class

```
import java.util.Random;

public class RandomNumberGenerator extends NumberGenerator {
    private Random random = new Random();
    private int number;

    public int getNumber() {return this.number;}

    public void execute() {
        for(int i = 0;  i < 10; i++) {
            this.number = random.nextInt(50);
            notifyObservers();
        }
    }
}
```

Observer Pattern 실습

- Observer Interface

```
public interface Observer {
    public abstract void update(NumberGenerator generator);
}
```

Observer Pattern 실습

- DigitObserver Class

```
public class DigitObserver implements Observer {
	private NumberGenerator num;

	public DigitObserver(NumberGenerator num) {
		this.num = num;
		num.addObserver(this);
	}

	public void update(NumberGenerator generator) {
		System.out.println("DigitObserver: " + generator.getNumber());

		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}
```

Observer Pattern 실습

- GraphObserver Class

```
public class GraphObserver implements Observer {
	private NumberGenerator num;

	public GraphObserver(NumberGenerator num) {
		this.num = num;
		num.addObserver(this);
	}

	public void update(NumberGenerator generator) {
		System.out.print("GraphObserver: ");

		for(int i = 0; i < generator.getNumber(); i++) {
			System.out.print("*");
		}
		System.out.println("");

		try {
			Thread.sleep(100);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}
```

Observer Pattern 실습

- PrintRandomNumber Class

```
public class PrintRandomNumber {
    public static void main(String[] args) {
        NumberGenerator a = new RandomNumberGenerator();
        DigitObserver b = new DigitObserver(a);
        GraphObserver c = new GraphObserver(a);

        a.execute();
    }
}
```

Observer Pattern 실습

- Result

```
DigitObserver: 45
GraphObserver: *********************************************
DigitObserver: 44
GraphObserver: ********************************************
DigitObserver: 1
GraphObserver: *
DigitObserver: 46
GraphObserver: **********************************************
DigitObserver: 5
GraphObserver: *****
DigitObserver: 10
GraphObserver: **********
DigitObserver: 13
GraphObserver: *************
DigitObserver: 34
GraphObserver: **********************************
DigitObserver: 10
GraphObserver: **********
DigitObserver: 19
GraphObserver: *******************
```

실습 과제

- Observer pattern 실습 작성, 제출