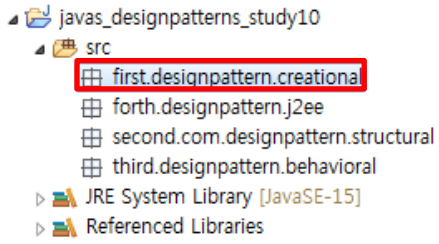
## 0. 디자인 패턴 타입: 생성(Creational), 구조(Structural), 동작(Behavioral), J2EE




## 1. 팩토리 패턴(Factory Pattern) – 생성

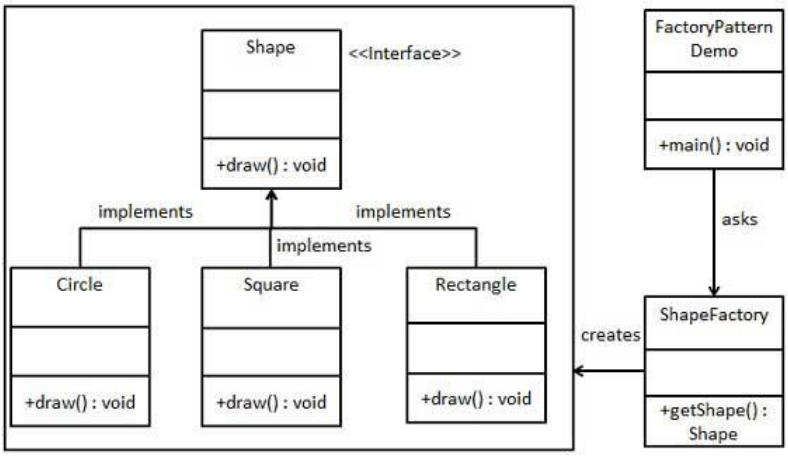
1) 개요

가. 자바에서 가장 많이 사용되는 디자인 패턴

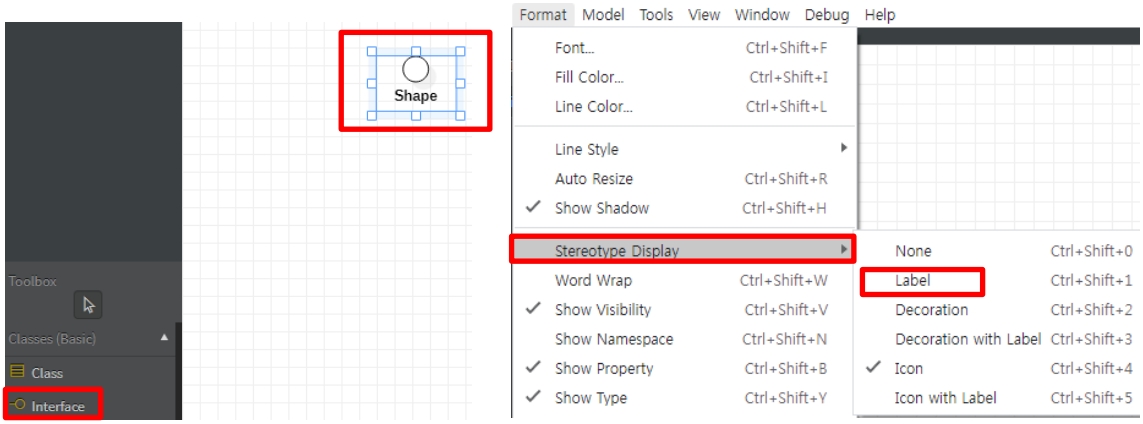
나. 객체를 생성하는 가장 좋은 방법 중 하나를 제공

다. 생성 로직을 클라이언트에 노출하지 않고 객체를 생성: 공통 인터페이스를 사용하여 새로 생성된 객체를 참조

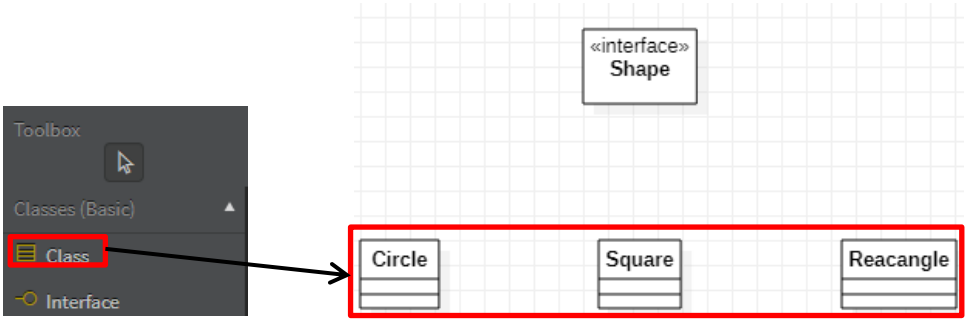
2) 설계

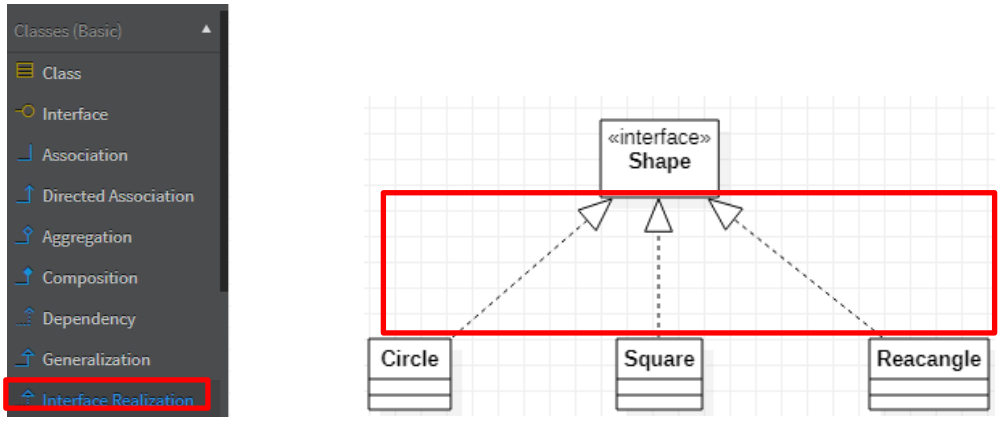


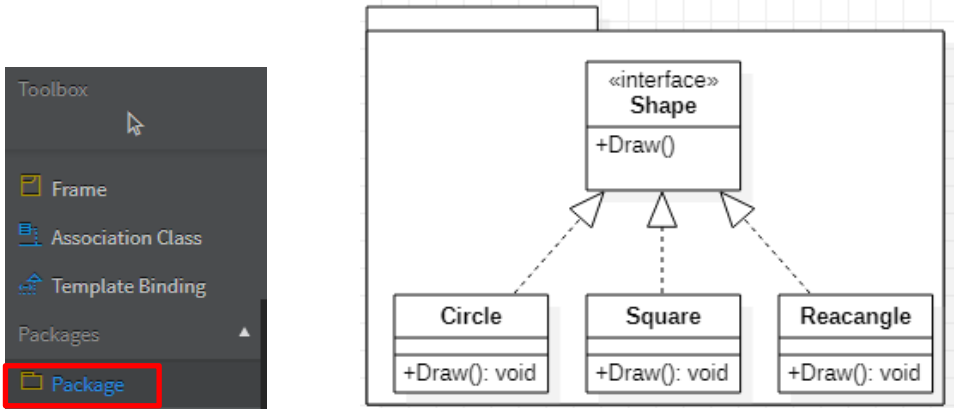

3) StarUML

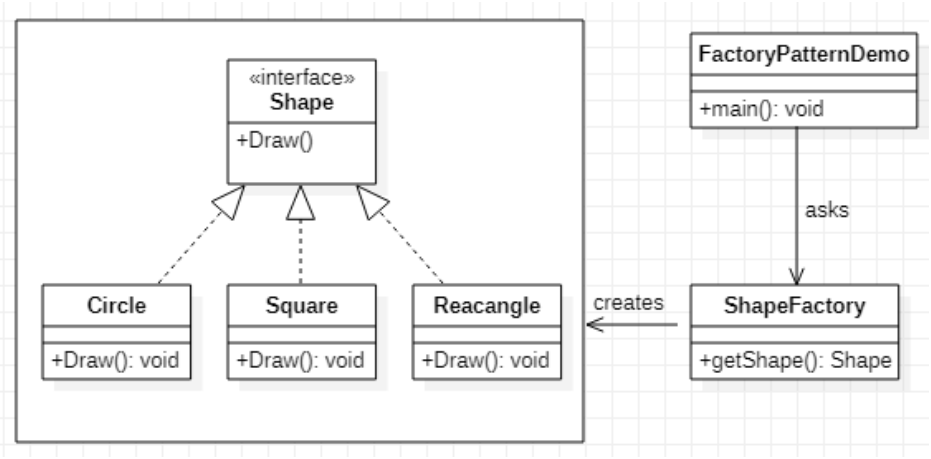



«interface»
Shape


Toolbox
Classes (Basic)
Class
Interface
«interface»
Shape
Circle
Square
Reacangle


Classes (Basic)
Class
Interface
Association
Directed Association
Aggregation
Composition
Dependency
Generalization
Interface Realization
«interface»
Shape
Circle
Square
Reacangle


Toolbox
Frame
Association Class
Template Binding
Packages
Package
«interface»
Shape
+Draw()
Circle
+Draw(): void
Square
+Draw(): void
Reacangle
+Draw(): void




4) 구현

가. 1단계

```
public interface Shape {
   void draw();
}
```

나. 2단계 - 1

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

다. 2단계 - 2

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

라. 3단계

```
public class ShapeFactory {

   //use getShape method to get object of type shape
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
```

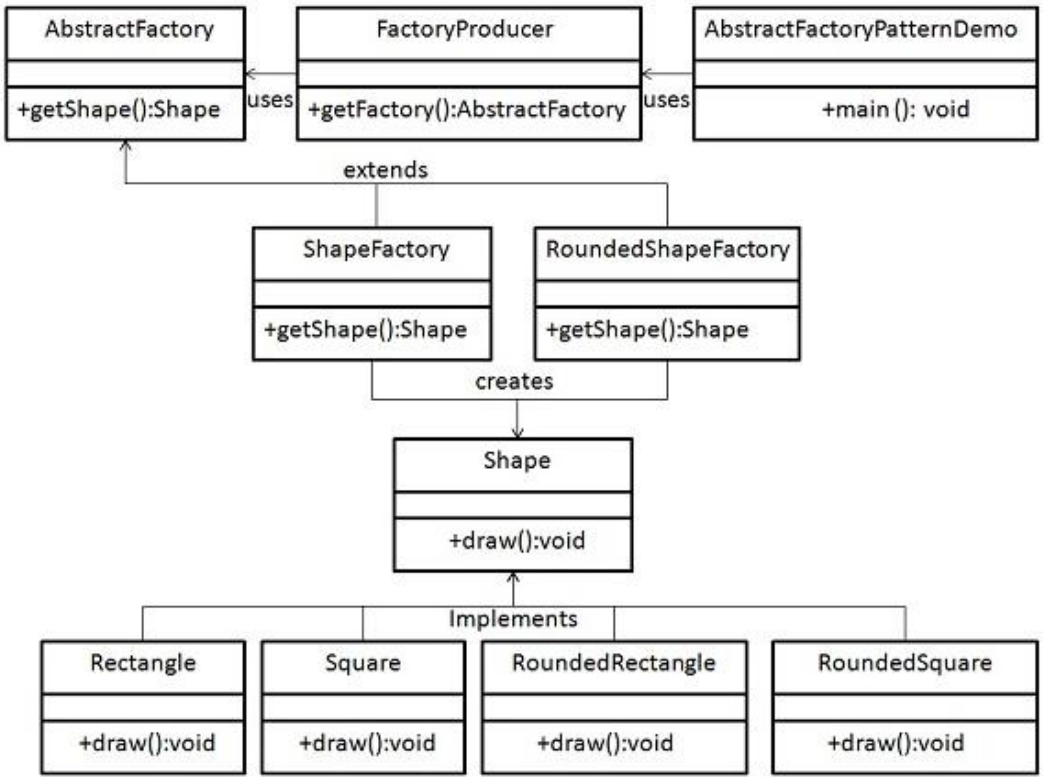
```
            return new Rectangle();

        } else if(shapeType.equalsIgnoreCase("SQUARE")){
            return new Square();
        }

        return null;
    }
}
```

마. 4단계

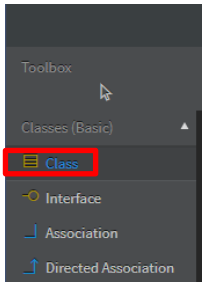
```
public class FactoryPatternDemo {

    public static void main(String[] args) {
        ShapeFactory shapeFactory = new ShapeFactory();

        //get an object of Circle and call its draw method.
        Shape shape1 = shapeFactory.getShape("CIRCLE");

        //call draw method of Circle
        shape1.draw();

        //get an object of Rectangle and call its draw method.
        Shape shape2 = shapeFactory.getShape("RECTANGLE");

        //call draw method of Rectangle
        shape2.draw();

        //get an object of Square and call its draw method.
        Shape shape3 = shapeFactory.getShape("SQUARE");

        //call draw method of square
        shape3.draw();
    }
}
```

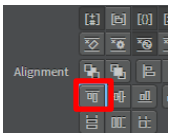
## 2. 추상 팩토리(Abstract Factory Pattern) – 생성

1) 설계

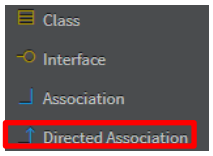



2) StarUML

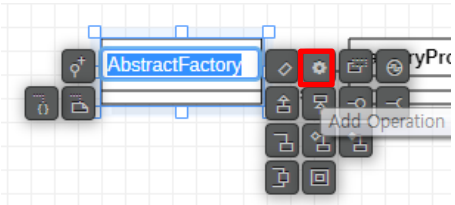


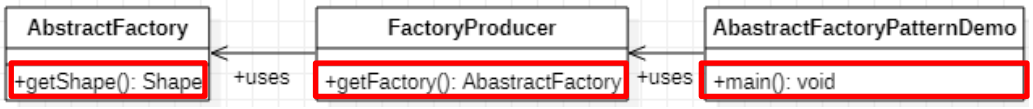



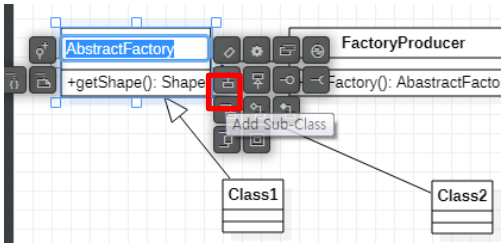


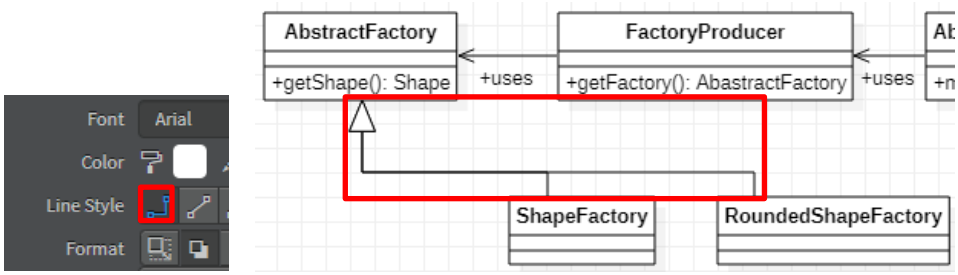




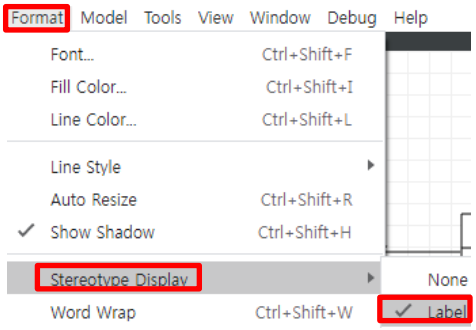


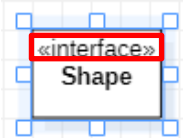

AbstractFactory
+getShape(): Shape
FactoryProducer
Add Sub-Class
Class1
Class2

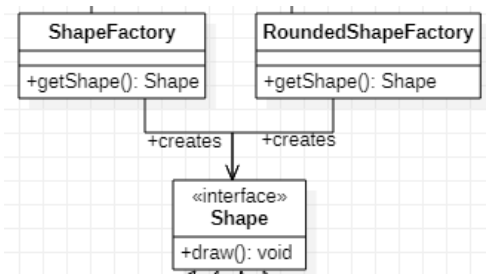

AbstractFactory
+getShape(): Shape
+uses
FactoryProducer
+getFactory(): AbastractFactory
+uses
ShapeFactory
RoundedShapeFactory

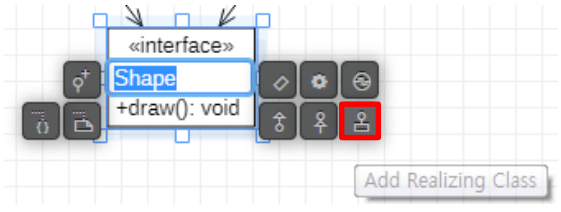

Font
Arial
Color
Line Style
Format

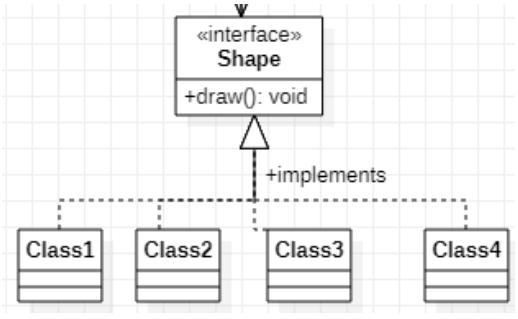

Format
Model
Tools
View
Window
Debug
Help
Font... Ctrl+Shift+F
Fill Color... Ctrl+Shift+I
Line Color... Ctrl+Shift+L
Line Style
Auto Resize Ctrl+Shift+R
Show Shadow Ctrl+Shift+H
Stereotype Display
None
Word Wrap Ctrl+Shift+W
Label

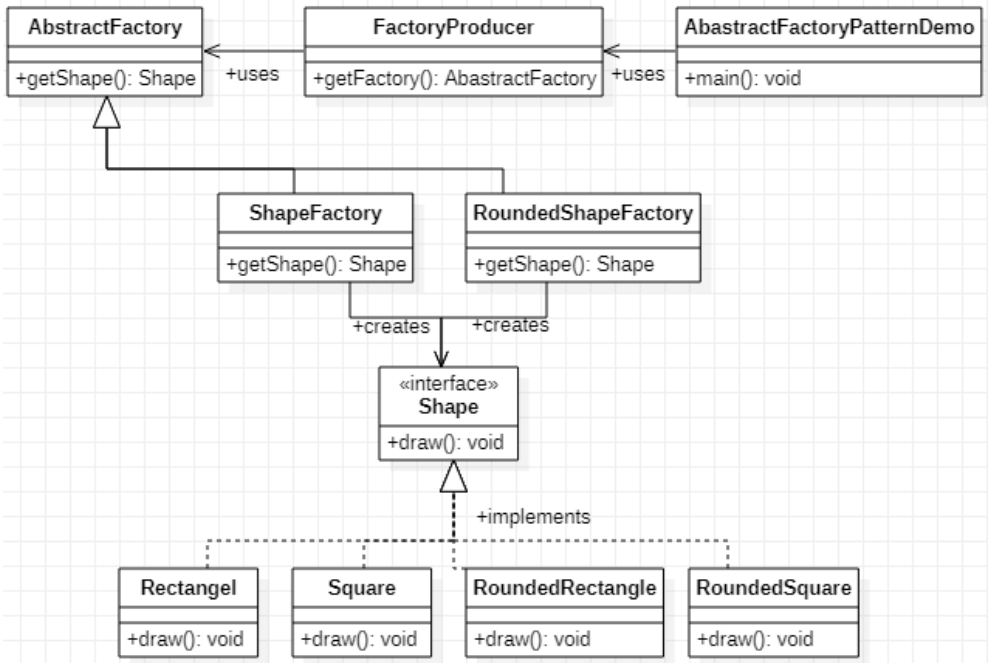

Class
Interface
Shape


«interface»
Shape


ShapeFactory
+getShape(): Shape
RoundedShapeFactory
+getShape(): Shape
+creates
+creates
«interface»
Shape
+draw(): void


«interface»
Shape
+draw(): void
Add Realizing Class


«interface»
Shape
+draw(): void
+implements
Class1
Class2
Class3
Class4




3) 구현

가. 1단계

```
public interface Shape {
    void draw();
}
```

나. 2단계

```
public class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside Rectangle::draw() method.");
    }
}
```

```
public class Square implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside Square::draw() method.");
    }
}
```

```
public class RoundedRectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside RoundedRectangle::draw() method.");
    }
}
```

```
public class RoundedSquare implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside RoundedSquare::draw() method.");
    }
}
```

라. 3단계

```
public abstract class AbstractFactory {
    abstract Shape getShape(String shapeType) ;
}
```

다. 4단계

```
public class ShapeFactory extends AbstractFactory {
    @Override
    public Shape getShape(String shapeType){
        if(shapeType.equalsIgnoreCase("RECTANGLE")){
            return new Rectangle();
        }else if(shapeType.equalsIgnoreCase("SQUARE")){
            return new Square();
        }
        return null;
    }
}

public class RoundedShapeFactory extends AbstractFactory {
    @Override
    public Shape getShape(String shapeType){
        if(shapeType.equalsIgnoreCase("RECTANGLE")){
            return new RoundedRectangle();
        }else if(shapeType.equalsIgnoreCase("SQUARE")){
            return new RoundedSquare();
        }
        return null;
    }
}
```

라. 5단계

```
public class FactoryProducer {
    public static AbstractFactory getFactory(boolean rounded){
        if(rounded){
            return new RoundedShapeFactory();
        }else{
            return new ShapeFactory();
        }
    }
}
```

마. 6단계

```
public class AbstractFactoryPatternDemo {
    public static void main(String[] args) {
        //get shape factory
        AbstractFactory shapeFactory = FactoryProducer.getFactory(false);
        //get an object of Shape Rectangle
        Shape shape1 = shapeFactory.getShape("RECTANGLE");
        //call draw method of Shape Rectangle
        shape1.draw();
        //get an object of Shape Square
        Shape shape2 = shapeFactory.getShape("SQUARE");
        //call draw method of Shape Square
        shape2.draw();
        //get shape factory
        AbstractFactory shapeFactory1 = FactoryProducer.getFactory(true);
        //get an object of Shape Rectangle
        Shape shape3 = shapeFactory1.getShape("RECTANGLE");
        //call draw method of Shape Rectangle
        shape3.draw();
        //get an object of Shape Square
        Shape shape4 = shapeFactory1.getShape("SQUARE");
        //call draw method of Shape Square
        shape4.draw();

    }
}
```

### 3. 싱글톤(Single) – 생성

1) 개념

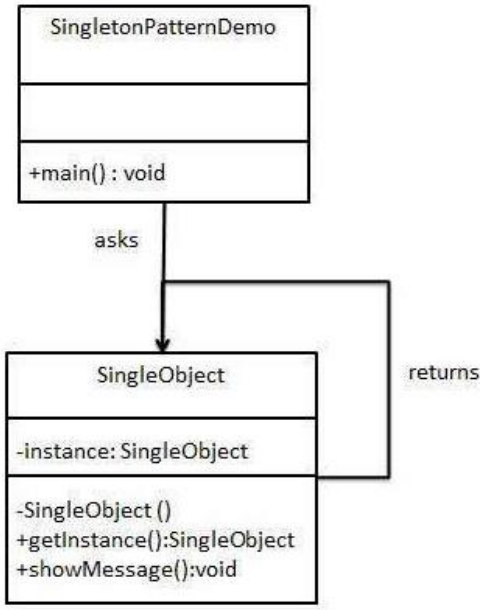
가. 가장 간단한 디자인 패턴

나. 객체를 생성하는 생성하는 가장 좋은 방법 중 하나를 제공

다. 단일 객체만 생성되도록 함

라. 클래스의 개체를 인스턴스 할 필요없이 직접 엑세스할 수 있도록 함

2) 설계




3) 구현

가. 1단계

```
public class SingleObject {

   //create an object of SingleObject
   private static SingleObject instance = new SingleObject();

   //make the constructor private so that this class cannot be
   //instantiated
   private SingleObject(){}

   //Get the only object available
   public static SingleObject getInstance(){
      return instance;
   }

   public void showMessage(){
      System.out.println("Hello World!");
   }
}
```

나. 2단계

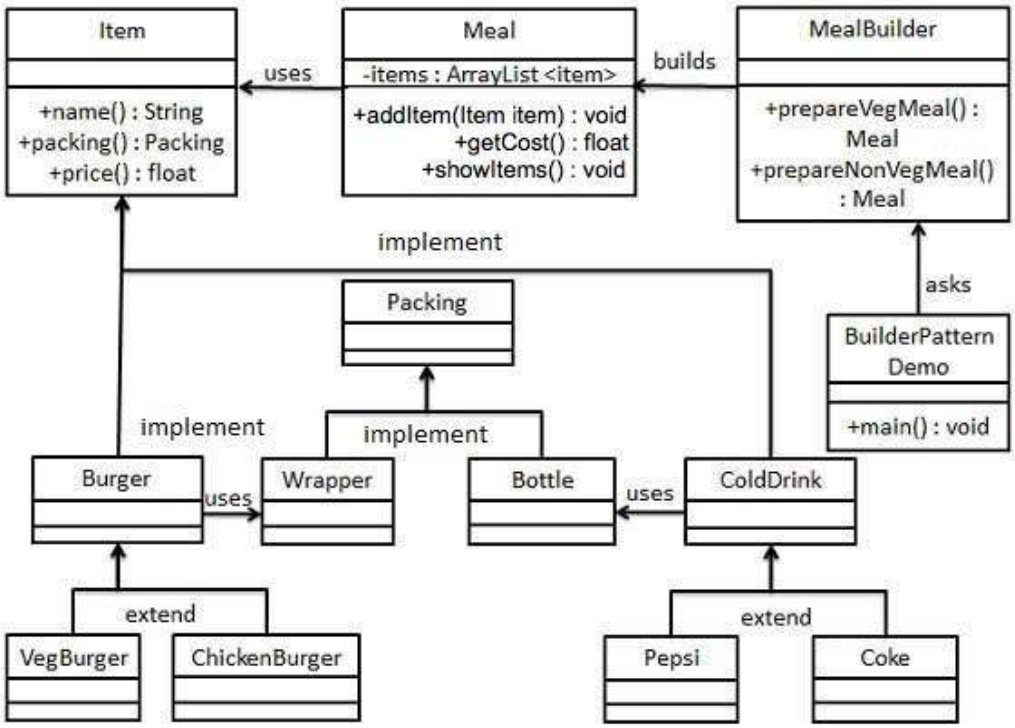
```
public class SingletonPatternDemo {
    public static void main(String[] args) {

        //illegal construct
        //Compile Time Error: The constructor SingleObject() is not visible
        //SingleObject object = new SingleObject();

        //Get the only object available
        SingleObject object = SingleObject.getInstance();

        //show the message
        object.showMessage();
    }
}
```

## 4. 빌더(Builder) – 생성

1) 개념

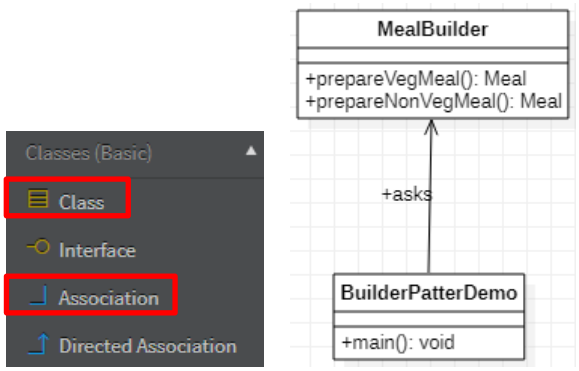
가. 단계별 접근 방식을 사용하여 복잡한 객체를 빌드

2) 설계

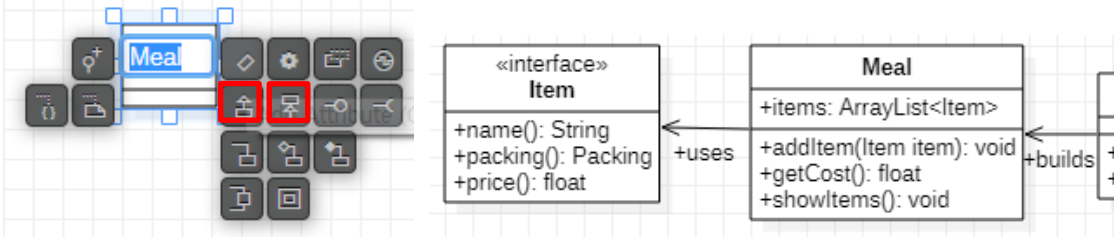


3) StarUML

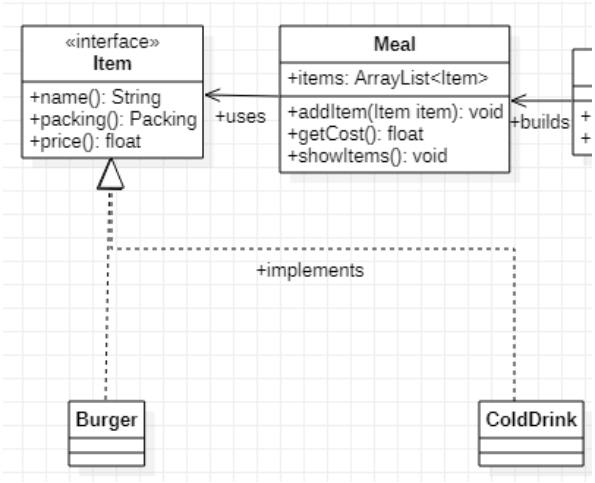
가. 메인 클래스와 빌더 클래스 생성



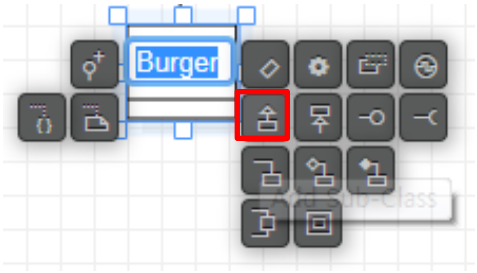

나. Meal 클래스와 Item 클래스 생성

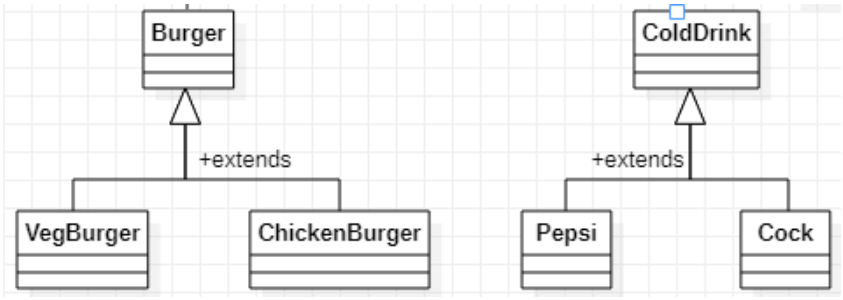



다. Item 인터페이스 구현체 클래스 생성

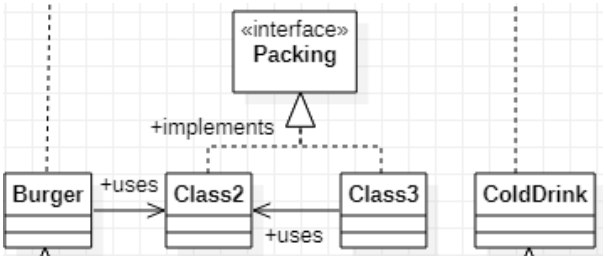


라. Burger/ColdDrink 클래스 확장

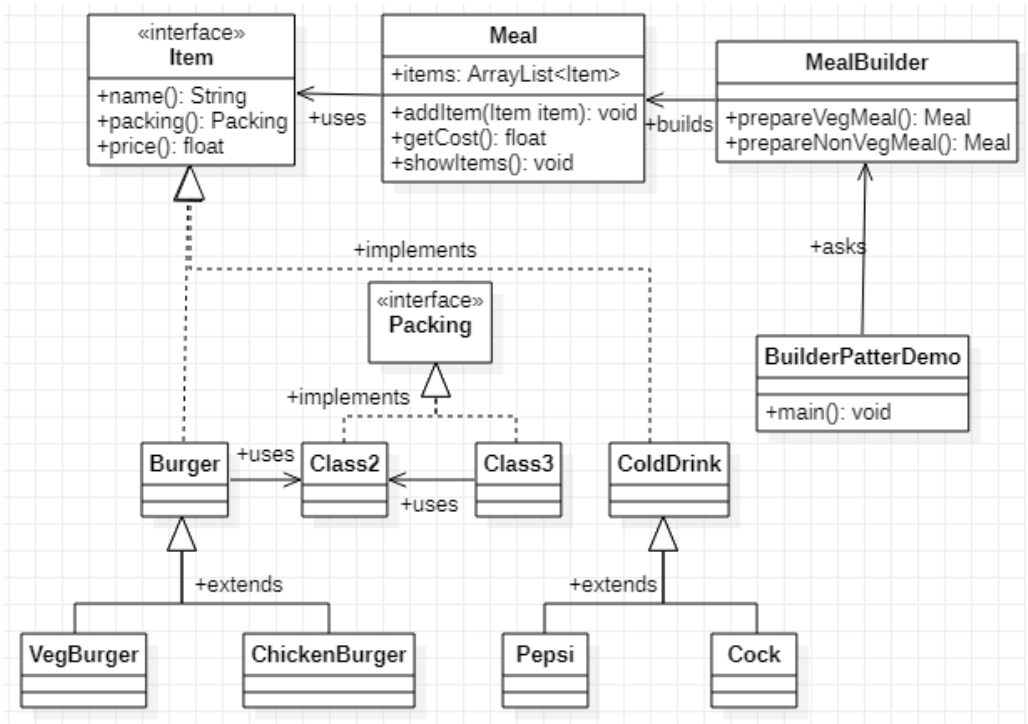



마. 패킹 인터페이스 및 구현체 클래스 생성




바. 전체 모습



4) 구현

가. 1단계

```
public interface Item {
    public String name();
    public Packing packing();
    public float price();
}

public interface Packing {
    public String pack();
}
```

나. 2단계

```
public class Wrapper implements Packing {

    @Override
    public String pack() {
        return "Wrapper";
    }
}

public class Bottle implements Packing {

    @Override
    public String pack() {
        return "Bottle";
    }
}
```

다. 3단계

```
public abstract class Burger implements Item {

    @Override
    public Packing packing() {
        return new Wrapper();
    }

    @Override
    public abstract float price();
}

public abstract class ColdDrink implements Item {

    @Override
    public Packing packing() {
        return new Bottle();
    }

    @Override
    public abstract float price();
}
```

라. 4단계

```
public class VegBurger extends Burger {

    @Override
    public float price() {
        return 25.0f;
    }

    @Override
    public String name() {
        return "Veg Burger";
    }
}

public class ChickenBurger extends Burger {

    @Override
    public float price() {
        return 50.5f;
    }

    @Override
    public String name() {
        return "Chicken Burger";
    }
}
```

```
public class Coke extends ColdDrink {

    @Override
    public float price() {
        return 30.0f;
    }

    @Override
    public String name() {
        return "Coke";
    }
}

public class Pepsi extends ColdDrink {

    @Override
    public float price() {
        return 35.0f;
    }

    @Override
    public String name() {
        return "Pepsi";
    }
}
```

마. 5단계

```
import java.util.ArrayList;
import java.util.List;

public class Meal {
    private List<Item> items = new ArrayList<Item>();

    public void addItem(Item item){
        items.add(item);
    }

    public float getCost(){
        float cost = 0.0f;

        for (Item item : items) {
            cost += item.price();
        }
        return cost;
    }

    public void showItems(){

        for (Item item : items) {
            System.out.print("Item : " + item.name());
```

```
            System.out.print(", Packing : " + item.packing().pack());
            System.out.println(", Price : " + item.price());
        }
    }
}
```

바. 6단계

```
public class MealBuilder {

    public Meal prepareVegMeal (){
        Meal meal = new Meal();
        meal.addItem(new VegBurger());
        meal.addItem(new Coke());
        return meal;
    }

    public Meal prepareNonVegMeal (){
        Meal meal = new Meal();
        meal.addItem(new ChickenBurger());
        meal.addItem(new Pepsi());
        return meal;
    }
}
```

사. 7단계

```
public class BuilderPatternDemo {
    public static void main(String[] args) {

        MealBuilder mealBuilder = new MealBuilder();

        Meal vegMeal = mealBuilder.prepareVegMeal();
        System.out.println("Veg Meal");
        vegMeal.showItems();
        System.out.println("Total Cost: " + vegMeal.getCost());

        Meal nonVegMeal = mealBuilder.prepareNonVegMeal();
        System.out.println("\n\nNon-Veg Meal");
        nonVegMeal.showItems();
        System.out.println("Total Cost: " + nonVegMeal.getCost());
    }
}
```

## 5. MVC 패턴(MVC)

1) 개념

가. Model-View-Controller의 약자

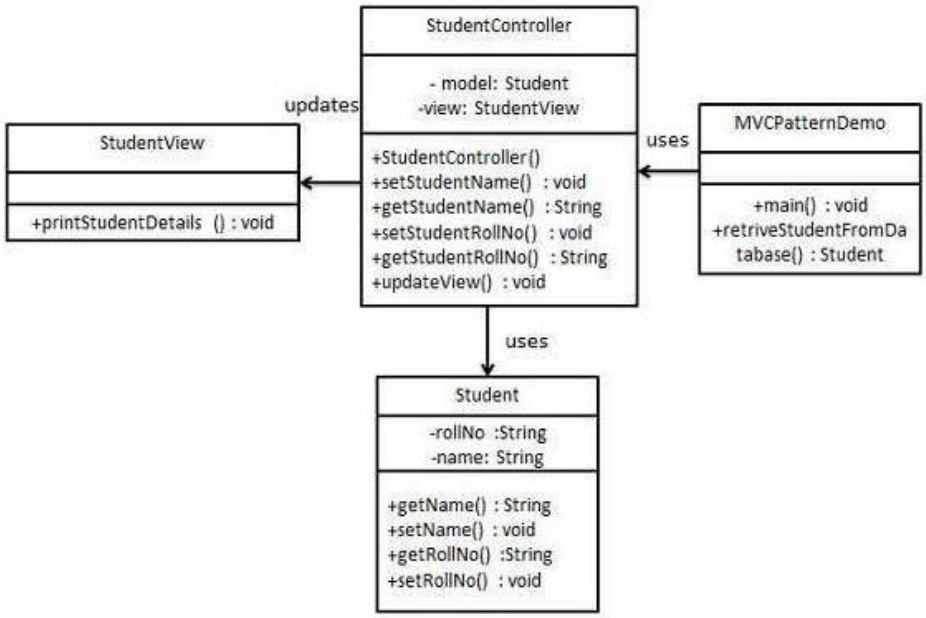
나. 애플리케이션 문제를 분리하는데 사용

다. 모델: 데이터를 운반하는 객체

라. 뷰: 모델에 포함된 데이터의 시각화

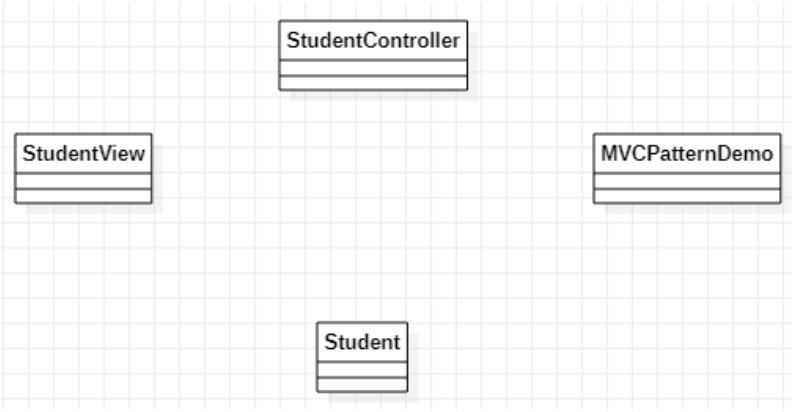
마. 컨트롤러: 모델 개체로의 데이터 흐름을 제어하고 데이터가 변경될 때마다 뷰를 업데이트

2) 설계

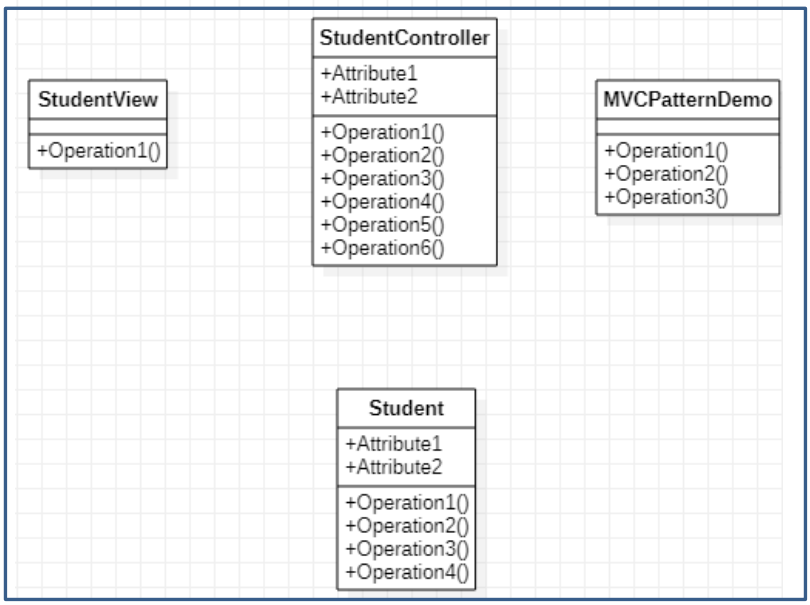


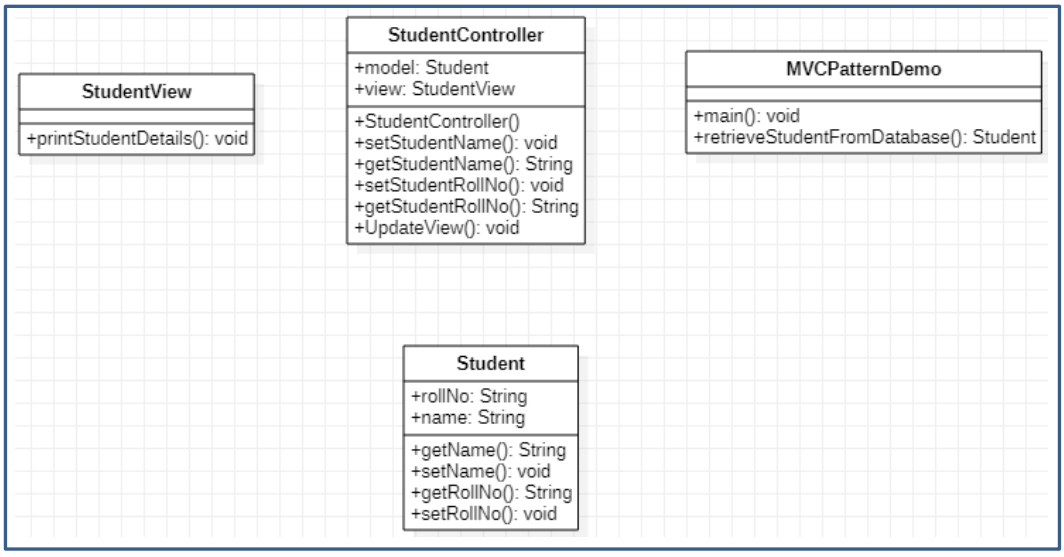

3) StarUML

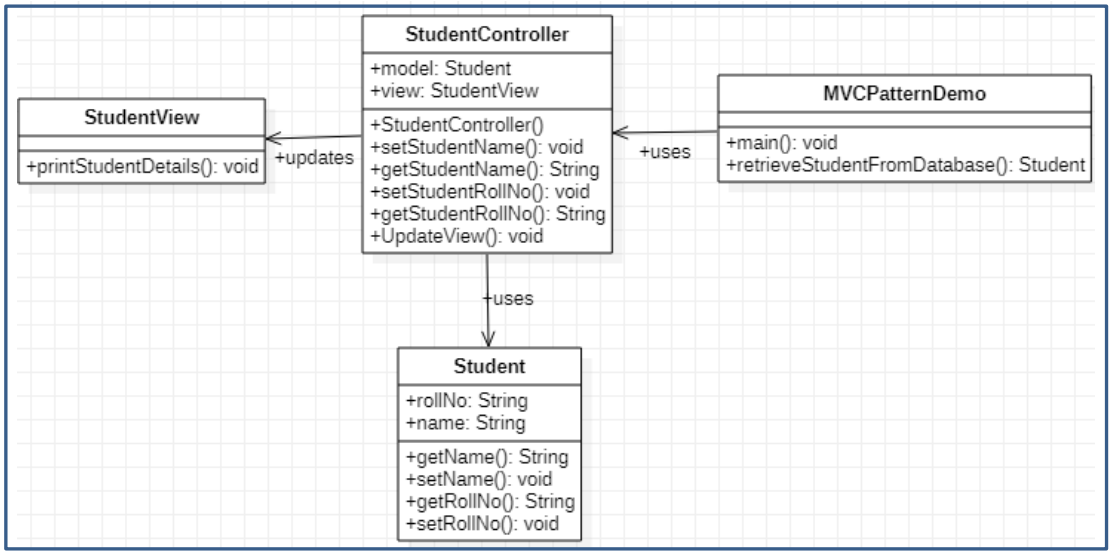
가. 전체 클래스 생성



나. 각 클래스의 속성과 오퍼레이션 추가



4) 구현

가. 1단계

```
public class Student {
    private String rollNo;
    private String name;

    public String getRollNo() {
        return rollNo;
    }

    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

나. 2단계

```
public class StudentView {
    public void printStudentDetails(String studentName, String studentRollNo){
        System.out.println("Student: ");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}
```

다. 3단계

```
public class StudentController {
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view){
        this.model = model;
        this.view = view;
    }

    public void setStudentName(String name){
        model.setName(name);
    }

    public String getStudentName(){
        return model.getName();
    }

    public void setStudentRollNo(String rollNo){
        model.setRollNo(rollNo);
    }

    public String getStudentRollNo(){
        return model.getRollNo();
    }

    public void updateView(){
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```

라. 4단계

```
public class MVCPatternDemo {
    public static void main(String[] args) {

        //fetch student record based on his roll no from the database
        Student model    = retriveStudentFromDatabase();

        //Create a view : to write student details on console
        StudentView view = new StudentView();

        StudentController controller = new StudentController(model, view);

        controller.updateView();

        //update model data
        controller.setStudentName("John");

        controller.updateView();
    }

    private static Student retriveStudentFromDatabase(){
        Student student = new Student();
        student.setName("Robert");
        student.setRollNo("10");
        return student;
    }
}
```

마. 5단계

Student:
Name: Robert
Roll No: 10
Student:
Name: John
Roll No: 10