Design Pattern Brochure

By: Sean Crim & Leo Benitez

Observer:

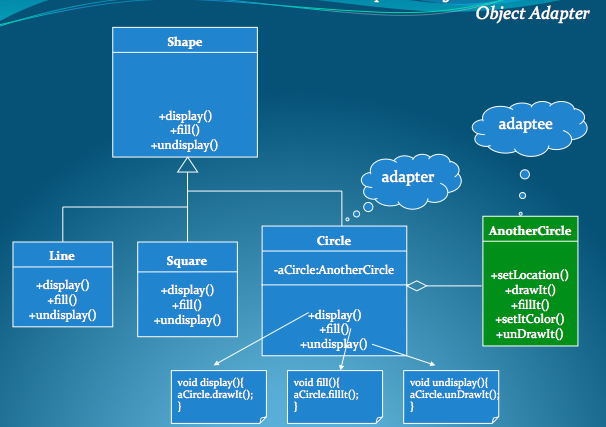
-UML Diagram:

-Intent:

-Sample Code:

Adapter:

-UML Diagram:



-Intent:

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

-Sample Code:

public abstract class Shape{

public abstract void display();

public abstract void fill();

public abstract void undisplay();

}

public class Line extends Shape{

public void display(){

// code for displaying line

}

public void fill(){

// code for filling in line

}

public void undisplay(){

// code for removing line

}

}

public class Square extends Shape{

public void display(){

// code for displaying square

}

public void fill(){

// code for filling in square

}

public void undisplay(){

// code for removing square

}

}

public class Circle extends Shape{

private AnotherCircle aCircle;

public Circle(){

aCircle = new AnotherCircle();

}

public void display(){

aCircle.drawIt(); //code for drawing circle

}

public void fill(){

aCircle.fillIt(); //code for filling in circle

}

public void undisplay(){

aCircle.unDrawIt(); //code for removing circle

}

}

public class AnotherCircle{

public void setLocation(){

//code for setting loation for circle

}

public void drawIt(){

// code for drawing out circle

}

public void fillIt(){

// code for filling in circle

}

public void setItColor(){

// code for setting circle color

}

public void unDrawIt(){

// code for removing circle

}

}

Strategy:

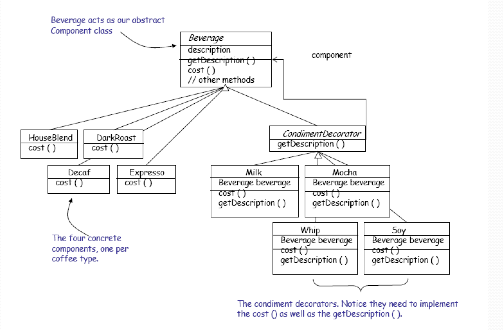
-UML Diagram:

-Intent:

-Sample Code:

Decorator:

-UML Diagram:



-Intent:

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

-Sample Code:

public abstract class Beverage{

public String description = "Unkown Description";

public String getDescription(){

return description;

}

public abstract double cost();

}

public class HouseBlend extends Beverage{

public HouseBlend(){

description = "House Blend";

}

public double cost(){

return 00.00;

}

}

public class DarkRoast extends Beverage{

public DarkRoast(){

description = "Dark Roast";

}

public double cost(){

return 00.00;

}

}

public class Decaf extends Beverage{

public Decaf(){

description = "Decaf";

}

public double cost(){

return 00.00;

}

}

public class Expresso extends Beverage{

public Expresso(){

description = "Expresso";

}

public double cost(){

return 00.00;

}

}

public abstract class CondimentDecorator extends Beverage{

public abstract String getDescription();

}

public class Mocha extends CondimentDecorator{

Beverage bev;

public Mocha(Beverage beverage){

bev = beverage;

}

public String getDescription(){

return bev.getDescription()+ ",Mocha";

}

public double cost(){

return 0.00+bev.cost();

}

}

public class Milk extends CondimentDecorator{

Beverage bev;

public Milk(Beverage beverage){

bev = beverage;

}

public String getDescription(){

return bev.getDescription()+ ",Milk";

}

public double cost(){

return 0.00+bev.cost();

}

}

public class Whip extends CondimentDecorator{

Beverage bev;

public Whip(Beverage beverage){

bev = beverage;

}

public String getDescription(){

return bev.getDescription()+ ",Whip";

}

public double cost(){

return 0.00+bev.cost();

}

}

public class Soy extends CondimentDecorator{

Beverage bev;

public Soy(Beverage beverage){

bev = beverage;

}

public String getDescription(){

return bev.getDescription()+ ",Soy";

}

public double cost(){

return 0.00+bev.cost();

}

}

Singleton:

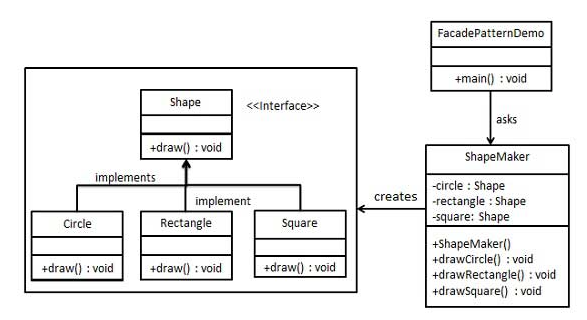
-UML Diagram:

-Intent:

-Sample Code:

Façade:

-UML Diagram:



-Intent:

Provide a unified interface to a set of interfaces in a

subsystem. Facade defines a higher-level interface

that makes the subsystem easier to use.

-Sample Code:

public interface Shape {

void draw();

}

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Rectangle::draw()");

}

}

public class Square implements Shape {

@Override

public void draw() {

System.out.println("Square::draw()");

}

}

public class Circle implements Shape {

@Override

public void draw() {

System.out.println("Circle::draw()");

}

}

public class ShapeMaker {

private Shape circle;

private Shape rectangle;

private Shape square;

public ShapeMaker() {

circle = new Circle();

rectangle = new Rectangle();

square = new Square();

}

public void drawCircle(){

circle.draw();

}

public void drawRectangle(){

rectangle.draw();

}

public void drawSquare(){

square.draw();

}

}

public class FacadePattern {

public static void main(String[] args) {

ShapeMaker shapeMaker = new ShapeMaker();

//create objects

shapeMaker.drawCircle();

shapeMaker.drawRectangle();

shapeMaker.drawSquare();

}

}