先上Reference(*https://github.com/DespairYoke/java-advance/tree/master/design-pattern*):

（简单）工厂模式(**Factory**)：在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。

创建实例

**Interface Restaurant**

Rice

Noodle

Pizza

**cook()**

Order

implements

**我是接口里的方法**

**我是接口**

**我是工厂类**

**我是实现类**

例：

public interface Restaurant {

void cook();

}

public class Duck implements Restaurant{

public void cook() {

System.out.println("来一份烤鸭");

}

}

public class Fish implements Retsaurant{

public void cook() {

System.out.println("来一份红烧鱼");

}

}

public class Meet implements Resaurant {

public void cook() {

System.out.println("来一份回锅肉");

}

}

public class Oreder {

public static final int MEAN\_MEET = 1;

public static final int MEAN\_FISH = 2;

public static final int MEAN\_DUCK = 3;

public static Resaurant getMean(int meantype){

switch (meantype){

case MEAN\_MEET :

return new Meet();

case MEAN\_FISH :

return new Fish();

case MEAN\_DUCK :

return new Duck();

default:

return null;

}

}

}

代理模式(**Proxy**)：一个对象为另一个对象提供一个代理或者占位符以控制对这个对象的访问。

在Station里New一个Station,即StationProxy的构造方法里需要添加一个Station

**StationProxy**

implements

**Station**

**ReturnTicket()**

**Consulte()**

**sellTicket()**

**TicketService**

添加Station引用到StationProxy

public interface TicketService {

//售票

void sellTicket();

//咨询

void Consultation();

//退票

void ReturnTicket();

}

public class Station implements TicketService {

@Override

public void sellTicket() {

System.out.println("售票");

}

@Override

public void Consultation() {

System.out.println("咨询");

}

@Override

public void ReturnTicket() {

System.out.println("退票");

}

}

public class StationProxy implements TicketService {

private Station station;

public StationProxy(Station station){

this.station = station;

}

@Override

public void sellTicket() {

System.out.println("欢迎使用车票代售点进行购票，每张票将会收取5元手续费！");

station.sellTicket();

System.out.println("欢迎下次光临");

}

@Override

public void Consultation() {

System.out.println("欢迎咨询，咨询不收取费用");

station.Consultation();

System.out.println("欢迎下次光临");

}

@Override

public void ReturnTicket() {

System.out.println("欢迎使用车票代售点进行退票，每张票将会收取5元手续费！");

station.ReturnTicket();

System.out.println("欢迎下次光临");

}

}

适配器模式：

Client

**Duck**

添加Turkey的引用到Duck

**Turkey**

**Adapter**

示例（让火鸡伪装成鸭子）：

public interface Duck {

public void quack();

public void fly();

}

public interface Turkey {

public void gobble();

public void fly();

}

public class WildTurkey implements Turkey{

@Override

public void gobble() {

System.out.println("Gobble");

}

@Override

public void fly() {

System.out.println("I'm flying a short distance");

}

} //野火鸡的一般实现

public class MallardDuck implements Duck{

@Override

public void quack() {

System.out.println("Quack");

} //鸭子叫

@Override

public void fly() {

System.out.println("I am flying");

} //鸭子飞翔

} //野鸭子的一般实现

public class Adapter implements Duck{

Turkey turkey;

public Adapter(Turkey turkey){

this.turkey = turkey;

}

@Override

public void quack() {

turkey.gobble();

}

@Override

public void fly() {

turkey.fly();

}

} //将火鸡适配成鸭子

public class AdapterTest {

public static void main(String[] args) {

Turkey turkey = new WildTurkey();

Duck turkeyAdapter = new Adapter(turkey);

turkey.gobble();

turkey.fly();

turkeyAdapter.quack();

turkeyAdapter.fly();

}

} //测试

装饰器模式：

将咖啡的引用添加到配料中

**Peanut**

**Mocha**

**CondimentDecorator**

**HouseBlend**

**Espresso**

**Beverage**

Abstract

public abstract class Beverage {

String description = "Unknown Beverage";

public String getDescription() {

return description;

}

public abstract double cost();

}

public abstract class CondimentDecorator extends Beverage{

@Override

public String getDescription() {

return super.getDescription();

}

}

public class Espresso extends Beverage{

public Espresso(){

description = "Espresso";

}

@Override

public double cost() {

return 1.99;

}

}

public class HouseBlend extends Beverage{

public HouseBlend(){

description = "House Blend Coffee";

}

@Override

public double cost() {

return 0.89;

}

}

public class Mocha extends CondimentDecorator{

Beverage beverage;

public Mocha(Beverage beverage){

this.beverage = beverage;

}

@Override

public String getDescription() {

return super.getDescription();

}

@Override

public double cost() {

return 0.2 + beverage.cost();

}

}

public class Peanut extends CondimentDecorator{

Beverage beverage;

public Peanut(Beverage beverage){

this.beverage = beverage;

}

@Override

public String getDescription() {

return super.getDescription();

}

@Override

public double cost() {

return 0.8 + beverage.cost();

}

}

public class StarBucksCoffee {

public static void main(String[] args) {

Beverage beverage = new Espresso();

System.out.println(beverage.cost());

Beverage espressowithmocha = new Mocha(beverage);

System.out.println(espressowithmocha.cost());

Beverage espressowithmochaandpeanut = new Peanut(espressowithmocha);

System.out.println(espressowithmochaandpeanut.cost());

}

}

观察者模式：

**DisplayElement**

**Observer**

**Subject**

**WeatherData**

**CurrentConditionsDisplay**

**ForecastDisplay**

**StatisticsDisplay**

public interface Subject {

public void registerObserver(Observer o);

public void removeObserver(Observer o);

public void notifyObservers();

}

public interface Observer {

public void update(float temp, float humidity, float pressure);

}

public interface DisplayElement {

public void display();

}

public class WeatherData implements Subject {

private ArrayList<Observer> observers;

private float temperature;

private float humidity;

private float pressure;

public WeatherData() {

observers = new ArrayList<Observer>();

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

int i = observers.indexOf(o);

if( i >=0 ){

observers.remove(i);

}

}

@Override

public void notifyObservers() {

for (int i = 0; i < observers.size(); i++) {

Observer observer = (Observer) observers.get(i);

observer.update(temperature, humidity, pressure);

}

}

public void measurementsChanged() {

notifyObservers();

}

public void setMeasurements(float temperature, float humidity, float pressure) {

this.temperature = temperature;

this.humidity = humidity;

this.pressure = pressure;

measurementsChanged();

}

}

public class StatisticsDisplay implements Observer,DisplayElement{

private float temperature;

private float humidity;

private float pressure;

private Subject weatherData;

public StatisticsDisplay(Subject weatherData){

this.weatherData=weatherData;

weatherData.registerObserver(this);

}

public void update(float temperature,float humidity,float pressure){

this.temperature=temperature;

this.humidity=humidity;

this.pressure=pressure;

display();

}

public void display(){

System.out.println("Avg/Max/Min temperature= "+temperature

+"/"+temperature+"/"+temperature);

}

}

public class CurrentConditionsDisplay implements Observer, DisplayElement{

private float temperature;

private float humidity;

private Subject weatherData;

public CurrentConditionsDisplay(Subject weatherData) {

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

@Override

public void display() {

System.out.println("Current conditions:" +

temperature + "F degrees and " +

humidity + "% humidity");

}

@Override

public void update(float temperature, float humidity, float pressure) {

this.temperature = temperature;

this.humidity = humidity;

display();

}

}

public class ForecastDisplay implements Observer,DisplayElement{

private float temperature;

private float humidity;

private float pressure;

private Subject weatherData;

public ForecastDisplay(Subject weatherData){

this.weatherData=weatherData;

weatherData.registerObserver(this);

}

public void update(float temperature,float humidity,float pressure){

this.temperature=temperature;

this.humidity=humidity;

this.pressure=pressure;

display();

}

public void display(){

System.out.println("Forecast: More of the same");

}

}

public class WeatherStation {

public static void main(String[] args) {

WeatherData weatherData = new WeatherData();

CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(weatherData);

StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);

ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

weatherData.setMeasurements(80, 65, 30.4f);

weatherData.setMeasurements(82, 70, 29.2f);

weatherData.setMeasurements(78, 90, 29.2f);

}

}