05 观察者模式、抽象工厂模式

C-3 创建: 张林伟, 最后修改: 张林伟 2018-10-19 18:07

观察者模式（发布-订阅模式）

定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态发生变化时，会通知所有的观察者对象，使它们能够自动更新自己。

Demo

主题

代码块 Java

```java
public class Subject {

    private List<Observer> observers = new ArrayList<>();

    @Setter
    @Getter
    private String name;

    @Getter
    private boolean state;

    public Subject(String name, boolean state) {
        this.name = name;
        this.state = state;
    }

    public Subject(Observer... observers) {
        for (int i = 0; i < observers.length; i++) {
            this.observers.add(observers[i]);
        }
    }

    public void setState(boolean state) {
        if (state) {
            System.out.println("今天国庆放假");
        } else {
            System.out.println("今天未到国庆放假");
        }
        this.state = state;
        notifyAllObservers();
    }

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

    public void notifyAllObservers() {
        observers.forEach(observer -> observer.listen());
    }
}
```

观察者

代码块 Java

```java
// Observer.java
public abstract class Observer {

    protected Subject subject;

    public abstract void listen();
}

// StudentObserver.java
```

```java
public class StudentObserver extends Observer {

    public StudentObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }

    @Override public void listen() {
        System.out.println("学生：放假咯。");
    }
}

// WorkerObserver.java
public class WorkerObserver extends Observer {

    public WorkerObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }

    @Override public void listen() {
        System.out.println("工人：终于可以休息了。");
    }
}
```

客户端

代码块 Java

```java
public class Client {

    public static void main(String[] args) {
        Subject subject = new Subject("国庆节", false);

        new StudentObserver(subject);
        new WorkerObserver(subject);

        subject.setState(true);
    }
}
```

抽象工厂模式

提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

系统的产品有多于一个的产品族，而系统只消费其中某一族的产品。

Demo

产品父类

代码块 Java

```java
// AbstractProductA.java
public abstract class AbstractProductA {
    public abstract void use();
}

// AbstractProductB.java
public abstract class AbstractProductB {
    public abstract void eat();
}
```

产品具体类

```java
// ProductA1.java
public class ProductA1 extends AbstractProductA {
    @Override public void use() {
        System.out.println("use productA1");
    }
}

// ProductA2.java
public class ProductA2 extends AbstractProductA {
    @Override public void use() {
        System.out.println("use productA2");
    }
}

// ProductB1.java
public class ProductB1 extends AbstractProductB {
    @Override public void eat() {
        System.out.println("eat productB1");
    }
}

// ProductB2.java
public class ProductB2 extends AbstractProductB {
    @Override public void eat() {
        System.out.println("eat productB2");
    }
}
```

工厂父类

代码块 Java

```java
public abstract class AbstractFactory {
    public abstract AbstractProductA createProductA();
    public abstract AbstractProductB createProductB();
}
```

工厂具体类

代码块 Java

```java
// ConcreteFactory1.java
public class ConcreteFactory1 extends AbstractFactory {

    @Override public AbstractProductA createProductA() {
        return new ProductA1();
    }

    @Override public AbstractProductB createProductB() {
        return new ProductB1();
    }
}

// ConcreteFactory2.java
public class ConcreteFactory2 extends AbstractFactory {

    @Override public AbstractProductA createProductA() {
        return new ProductA2();
    }

    @Override public AbstractProductB createProductB() {
        return new ProductB2();
    }
}
```

客户端

```java
public class Client {

    public static void main(String[] args) {
        AbstractFactory factory1 = new ConcreteFactory1();

        AbstractProductA productA = factory1.createProductA();
        AbstractProductB productB = factory1.createProductB();

        productA.use();
        productB.eat();
    }
}
```