一、工厂模式

定义了一个创建对象的类，由这个类来封装实例化对象的行为。

public class SimplePizzaFactory {

public Pizza CreatePizza(String ordertype) {

Pizza pizza = null;

if (ordertype.equals("cheese")) {

pizza = new CheesePizza();

} else if (ordertype.equals("greek")) {

pizza = new GreekPizza();

} else if (ordertype.equals("pepper")) {

pizza = new PepperPizza();

}

return pizza;

}

}

特点是调用者只需要知道对象的名称即可创建对象。扩展性高，如果需要增加新产品，只需扩展一个工厂类即可。屏蔽了产品的具体实现，调用者只关心产品的接口。但是每次增加一个产品时，都需要增加一个具体类和对应的工厂，使系统中类的数量成倍增加，增加了系统的复杂度和具体类的依赖。

二、单例模式

确保一个类最多只有一个实例，并提供一个全局访问点

public class PreloadSingleton {

public static PreloadSingleton instance = new PreloadSingleton();

private PreloadSingleton() {

};

public static PreloadSingleton getInstance() {

return instance;

}

}

public class Singleton {

private static Singleton instance=null;

private Singleton(){

};

public static Singleton getInstance(){

if(instance==null) {

instance=new Singleton();

}

return instance;

}

}

特点是内存中只有一个实例，频繁创建和销毁实例时，减少内存开销。避免资源的多重占用但没有接口，不能继承。与单一职责原则冲突，一个类应该只关心内部逻辑，而不关心实例化方式。

三、生成器模式

封装一个复杂对象构造过程，并允许按步骤构造。

public abstract class ComputerBuilder {

protected Computer computer;

public Computer getComputer() {

return computer;

}

public void buildComputer() {

computer = new Computer();

System.out.println("生成了一台电脑！！！");

}

public abstract void buildMaster();

public abstract void buildScreen();

public abstract void buildKeyboard();

public abstract void buildMouse();

public abstract void buildAudio();

}

特点是将一个对象分解为各个组件，将对象组件的构造封装起来，可以控制整个对象的生成过程。

四、原型模式

通过复制现有实例来创建新的实例，无需知道相应类的信息。

public class Prototype implements Cloneable {

public Object clone() throws CloneNotSupportedException {

Prototype proto = (Prototype) super.clone();

return proto;

}

}

特点是避免构造函数的约束，提高了性能。

项目中，采用了工厂模式、原型模式、外观模式、组合模式