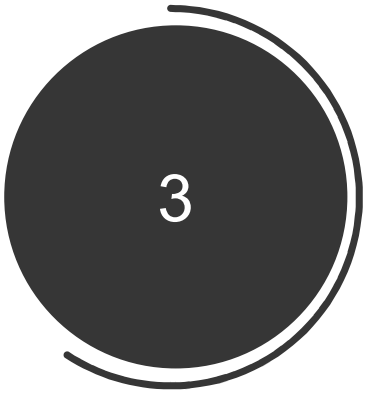
面向对象程序设计
工厂设计模式

factory pattern

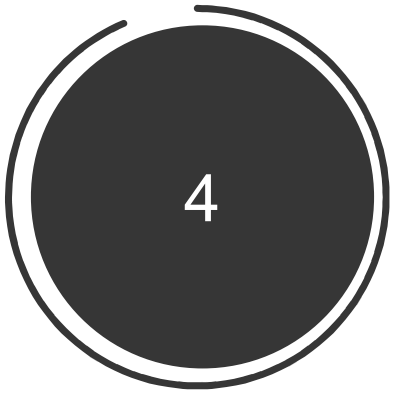
张宁鑫

目录

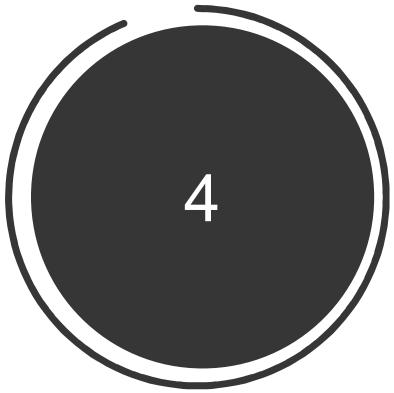
工厂设计模式
Factory Pattern



简单工厂
Naive Factory



工厂方法
Factory Method



抽象工厂
Abstract Factory

工厂设计模式

```
Coffee coffee;

if(americano) {
    return new Americano()
} else if (cappuccino) {
    return new Cappuccino(
} else if (latte) {
    return new Latte();
}
```

```
// 拿铁、美式咖啡、卡布奇诺等均为咖啡家族的一种产品
// 咖啡则作为一种抽象概念
public abstract class Coffee {
      // 获取coffee名称
      public abstract String getName();
}
// 美式咖啡
public class Americano extends Coffee {
      @Override
      public String getName() {
      return "美式咖啡";
      }
}
// 卡布奇诺
public class Cappuccino extends Coffee {
      @Override
      public String getName() {
      return "卡布奇诺";
      }
}
// 拿铁
public class Latte extends Coffee {
      @Override
      public String getName() {
      return "拿铁";
      }
}
```

就必须重新打开这段代码进

编程——工厂设计模式

工厂设计模式

工厂模式用于封装对象的创建
使得我们可以将程序从具体类解耦

简单工厂（静态方法）

```
public class NaiveFactory {
//通过类型获取Coffee实例对象
    public static Coffee createInstance(String type){
    if("americano".equals(type)){
        return new Americano();
    }else if("cappuccino".equals(type)){
        return new Cappuccino();
    }else if("latte".equals(type)){
        return new Latte();
    }
}
```

引入创建者的概念，将实例化的代码从应用代码中抽离，在创建者类的静态方法中只处理创建对象的细节，后续创建的实例如需改变，只需改造创建者类即可。

工厂方法

```
public abstract class CoffeeFactory {
    // 生产可制造的咖啡
    public abstract Coffee[] createCoffee();
}
// 中国咖啡工厂
public class ChinaCoffeeFactory extends CoffeeFactory {
    @Override
    public Coffee[] createCoffee() {
        return new Coffee[]{new Cappuccino(), new Latte()};
    }
}
// 美国咖啡工厂
public class AmericaCoffeeFactory extends CoffeeFactory {
    @Override
    public Coffee[] createCoffee() {
        return new Coffee[]{new Americano(), new Latte()};
    }
}
```

定义了一个创建对象的抽象类，再由子类决定具体要实例化的哪些类，工厂方法让类b把实例化推迟到了子类。

抽象工厂

```
public interface AbstractDrinksFactory {
    Coffee createCoffee();
    Tea createTea();
    Sodas createSodas();
}
public class ChinaDrinksFactory implements AbstractDrinksFactory {
    public Coffee createCoffee(){return new Latte();}
    public Tea createTea(){return new MilkTea();}
    public Sodas createSodas(){return null;}
}
public class AmericaDrinksFactory implements AbstractDrinksFactory {
    public Coffee createCoffee() {return new Latte();}
    public Tea createTea() {return null;}
    public Sodas createSodas() {return new CocaCola();}
}
```

提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类。

工厂方法 in guava

```
abstract static class ViewCachingAbstractMap<K, V> extends AbstractMap<K, V> {
    ...
}
private static class AsMapView<K, V> extends ViewCachingAbstractMap<K, V> {
    ...
}
private abstract static class AbstractFilteredMap<K, V> extends ViewCachingAbstractMap<K, V>
{
    ...
}
private class Column extends ViewCachingAbstractMap<R, V> {
    ...
}
class RowMap extends ViewCachingAbstractMap<R, Map<C, V>> {
    ...
}
```

抽象工厂 in guava

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}

class WrappedIterator implements Iterator<V> {
    final Iterator<V> delegateIterator;
    ...
}
private abstract class Itr<T> implements Iterator<T> {
    final Iterator<Entry<K, Collection<V>>> keyIterator;
    ...
}
class AsMapIterator implements Iterator<Entry<K, Collection<V>>> {
    final Iterator<Entry<K, Collection<V>>> delegateIterator = submap.entrySet().iterator();
    ...
}
```

面向对象程序设计
工厂设计模式

factory pattern

部分内容引用自：

1. https://www.cnblogs.com/carryjack/p/7709861.html
2. 《HeadFirst设计模式》
3. google guava

张宁鑫