1、什么是设计模式？

* 模式就是解决一类问题的固定步骤。

2、软件行业中23种设计模式：

* 单例设计模式
* 模板设计模式
* 装饰者设计模式
* 观察者设计模式
* 工厂设计模式

# 1. 单例设计模式

## 1、单例设计模式：

* 保证一个类在内存中只有一个对象。

## 2、单例设计模式的步骤：

### （1）饿汉单例设计模式

1. 私有化构造函数。

2. 声明本类的引用类型变量，并且使用该变量指向本类对象。

3. 提供一个公共静态的方法获取本类的对象。

|  |
| --- |
| **// 确保Single在内存中只有一个对象（饿汉单例模式） public class** Single {  // 私有化构造函数，确保外部 不能创建Single对象  **private** Single() {  }   //初始化对象；static静态的，保证类加载后就创建该对象；private私有的，确保外部不能访问该对象  **private static** Single single = **new** Single();   // 对外提供获取该对象实例的方法  **public static** Single getInstance() {  **return** single;  } }  **// 单例模式测试类**  **class** SingleTest {  **public static void** main(String[] args) {  Single single1 = Single.getInstance();  Single single2 = Single.getInstance();  System.**out**.println(**"Single1 和 Single2是否是同一个对象？"** + (single1 == single2));  } } |

### （2）懒汉单例设计模式：

1. 私有化构造函数。

2. 声明本类的引用类型变量，但是不要创建对象，

3. 提供公共静态 的方法获取本类 的对象，获取之前先判断是否已经创建了本类 对象

，如果已经创建了，那么直接返回对象即可，如果还没有创建，那么先创建本类的对象，

然后再返回。

**推荐使用： 饿汉单例设计模式。 因为懒汉单例设计模式会存在线程安全问题，目前还不能保证一类在内存中只有一个对象。**

|  |
| --- |
| **// 确保Single在内存中只有一个对象（懒汉单例模式）** **public class** Single {  // 声明对象，但不创建对象  **private static** Single single;   // 私有化构造函数  **private** Single() {   }   // 提供获取Single对象的方法  **public static** Single getInstance() {  // 判断Single是否已被创建，如果没有被创建，那么创建single对象；如果已经存在，直接返回single对象  **if** (single == **null**) {  single = **new** Single();  }  **return** single;  } }  **// 单例模式测试类**  **class** SingleTest {  **public static void** main(String[] args) {  Single single1 = Single.getInstance();  Single single2 = Single.getInstance();  System.**out**.println(**"Single1 和 Single2是否是同一个对象？"** + (single1 == single2));  } } |

# 2. 装饰者设计模式

## 1.装饰者设计模式的作用：

* 增强一个类的功能
* 能够让这些类彼此互相装饰

## 2.装饰者模式的使用过步骤

* 1. 在装饰类（增强类，子类）内部维护维护一个被装饰类（被增强类，父类）的引用。
* 2. 让装饰类有一个共同的父类或父接口（所有装饰类都继承同一个父类）

具体步骤：

1. 在装饰类内部维护一个被装饰类的引用
2. 通过构造方法初始化这个引用
3. 使用类内部的引用去读取数据，不再使用super调用readLine()方法
4. 每个装饰类都按照以上三点去构建装饰类，这样多个装饰类之间就可以互相装饰了。

例如：增强一个BufferedReader类

|  |
| --- |
| **class** BufferedReaderSemicolon2 **extends** BufferedReader {  // **1. 在装饰类内部维护一个被装饰类的引用**  **BufferedReader bufferedReader;**//内部维护了被装饰类的引用(BufferedReader 被装饰类)  // **2. 通过构造方法初始化这个引用**  **public** BufferedReaderSemicolon2(BufferedReader bufferedReader) {  **super**(bufferedReader);// 注意，这句话没有任何作用，只是为了不让代码报错  **this**.**bufferedReader** = bufferedReader;  }   @Override  **public** String readLine() **throws** IOException {  //**3. 使用类内部维护这个引用去读取数据，不再使用super调用readLine()方法。**  String line = **bufferedReader**.readLine();  **if** (line == **null**) {  **return null**;  }  line = line + **";"**;  **return** line;  } }  // **4. 多个类都实现了同一个类（如 BufferedReader）,这时就可以使用过修饰者模式互相修饰了。** |

## 3. 继承实现的增强类和修饰模式实现的增强类有何区别？

* 继承实现的增强类：

优点：代码结构清晰，而且实现简单.

缺点：对于每一个的需要增强的类都要创建具体的子类来帮助其增强，这样会导致继承体系过于庞大。

* 修饰模式实现的增强类：

优点：内部可以通过多态技术对多个需要增强的类进行增强， 可以是这些装饰类达到互相装饰的效果。使用比较灵活。

缺点：需要内部通过多态技术维护需要被增强的类的实例。进而使得代码稍微复杂。

## 4. 案例代码

### 4.1 继承实现增强类的功能

* 继承实现的增强类：

优点：代码结构清晰，而且实现简单.

缺点：对于每一个的需要增强的类都要创建具体的子类来帮助其增强，这样会导致继承体系过于庞大。

|  |
| --- |
| **package** designmode;  **import** java.io.\*;  **public class** Dome1 {  **public static void** main(String[] args) {  // 找到目标文件  File file = **new** File(**"E:\\aa\\bb\\Demo1.java"**);  FileReader fileReader = **null**;  BufferedReaderLineNum bufferedReaderLineNum = **null**;  BufferedReaderSemicolon bufferedReaderSemicolon = **null**;  BufferedReaderQuotes bufferedReaderQuotes = **null**;   **try** {  // 搭建数据通道  fileReader = **new** FileReader(file);  bufferedReaderLineNum = **new** BufferedReaderLineNum(fileReader);// 带行号的缓冲输入字符流  bufferedReaderSemicolon = **new** BufferedReaderSemicolon(fileReader);// 带分号的缓冲输入字符流  bufferedReaderQuotes = **new** BufferedReaderQuotes(fileReader);    //传输数据  String content = **null**;//存储读取到的数据  **while** ((content = bufferedReaderQuotes.readLine()) != **null**) {  System.***out***.println(content);  }  } **catch** (FileNotFoundException e) {  **throw new** RuntimeException(e);  } **catch** (IOException e) {  **throw new** RuntimeException(e);  } **finally** {  **try** {  bufferedReaderLineNum.close();  } **catch** (IOException e) {  **throw new** RuntimeException(e);  }  }  } }  // 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有行号 **class** BufferedReaderLineNum **extends** BufferedReader {   **int count** = 1;   **public** BufferedReaderLineNum(Reader in) {  **super**(in);  }   @Override  **public** String readLine() **throws** IOException {  String line = **super**.readLine(); //读取到的数据  // 读不到数据时，返回null  **if** (line == **null**) {  **return null**;// 用来结束最后一次读取数据  }  line = **count** + **" "** + line; //将每行数据加上行号  **count**++;// 每读取完一行，行号+1  **return** line;  } }  //编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有分号。 **class** BufferedReaderSemicolon **extends** BufferedReader {   **public** BufferedReaderSemicolon(Reader in) {  **super**(in);  }   @Override  **public** String readLine() **throws** IOException {  String line = **super**.readLine();  **if** (line == **null**) {  **return null**;// 用来结束最后一次读取数据  }  line = line + **";"**;//将每行数据加上分号  **return** line;  } }  // 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有双引号。 **class** BufferedReaderQuotes **extends** BufferedReader {  **public** BufferedReaderQuotes(Reader in) {  **super**(in);  }  @Override  **public** String readLine() **throws** IOException {  String line = **super**.readLine();  **if** (line == **null**) {  **return null**;//用来结束最后一次读取数据  }  line = **"\""** + line + **"\""**;// 将每行数据加上双引号  **return** line;  } } |

### 4.2 装修者模式实现类功能增强

* 修饰模式实现的增强类：

优点：内部可以通过多态技术对多个需要增强的类进行增强， 可以是这些装饰类达到互相装饰的效果。使用比较灵活。

缺点：需要内部通过多态技术维护需要被增强的类的实例。进而使得代码稍微复杂。

|  |
| --- |
| **package** designmode;  **import** java.io.\*;  **public class** Dome2 {  // 找到目标文件  **static** BufferedReader *bufferedReader*;  **static** BufferedReaderLineNum2 *bufferedReaderLineNum2* = **null**;  **static** BufferedReaderSemicolon2 *bufferedReaderSemicolon2* = **null**;  **static** BufferedReaderQuotes2 *bufferedReaderQuotes2* = **null**;   **static** {  **try** {  *bufferedReader* = **new** BufferedReader(**new** FileReader(**new** File(**"E:\\aa\\bb\\Demo1.java"**)));  } **catch** (FileNotFoundException e) {  **throw new** RuntimeException(e);  }  }    // 程序入口  **public static void** main(String[] args) {   //testBufferedReaderLineSemicolom2();// 需求4：带分号和行号   //testBufferedReaderSemicolonQuotes2();// 需求5：带分号和双引号   *testBufferedReaderQuotesLineNum2*();// 需求6：带行号和双引号  }    // 需求4： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回的字符串带有行号+ 分号。  **public static void** testBufferedReaderLineSemicolom2() {  // 构建数据通道  **try** {  *bufferedReaderLineNum2* = **new** BufferedReaderLineNum2(*bufferedReader*);// 行号修饰类  *bufferedReaderSemicolon2* = **new** BufferedReaderSemicolon2(*bufferedReaderLineNum2*);//用分号装饰行号  String line = **null**;  //读取数据  **while** ((line = *bufferedReaderSemicolon2*.readLine()) != **null**) {  System.***out***.println(line);  }  } **catch** (IOException e) {  **throw new** RuntimeException(e);  } **finally** {  **try** {  *bufferedReaderSemicolon2*.close();  } **catch** (IOException e) {  **throw new** RuntimeException(e);  }  }  }    // 需求5： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有分号+ 双引号。  **public static void** testBufferedReaderSemicolonQuotes2() {  **try** {  *bufferedReaderSemicolon2* = **new** BufferedReaderSemicolon2(*bufferedReader*);  *bufferedReaderQuotes2* = **new** BufferedReaderQuotes2(*bufferedReaderSemicolon2*);   String line = **null**;  **while** ((line = *bufferedReaderQuotes2*.readLine()) != **null**) {  System.***out***.println(line);  }  } **catch** (IOException e) {  **throw new** RuntimeException(e);  }   }    // 需求6： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有双引号+ 行号。  **public static void** testBufferedReaderQuotesLineNum2() {  **try** {  *bufferedReaderLineNum2* = **new** BufferedReaderLineNum2(*bufferedReader*);  *bufferedReaderQuotes2* = **new** BufferedReaderQuotes2(*bufferedReaderLineNum2*);   String line = **null**;  **while** ((line = *bufferedReaderQuotes2*.readLine()) != **null**) {  System.***out***.println(line);  }  } **catch** (IOException e) {  **throw new** RuntimeException(e);  }  } }   // 装饰类（带分号的缓冲输入字符流）需求2：编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有分号。 **class** BufferedReaderSemicolon2 **extends** BufferedReader {   BufferedReader **bufferedReader**;//内部维护了被装饰类的引用   **public** BufferedReaderSemicolon2(BufferedReader bufferedReader) {  **super**(bufferedReader);// 注意，这句话没有任何作用，只是为了不让代码报错  **this**.**bufferedReader** = bufferedReader;  }   @Override  **public** String readLine() **throws** IOException {  String line = **bufferedReader**.readLine();// 使用类内部维护的被装饰类的引用去读取数据  **if** (line == **null**) {  **return null**;  }  line = line + **";"**;  **return** line;  } }   // 装饰类（带双引号的缓冲输入字符流）需求3： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有双引号。 **class** BufferedReaderQuotes2 **extends** BufferedReader {  BufferedReader **bufferedReader**;   **public** BufferedReaderQuotes2(BufferedReader bufferedReader) {  **super**(bufferedReader);// 注意：这句话没有任何作用，只是防止代码报错  **this**.**bufferedReader** = bufferedReader;  }   @Override  **public** String readLine() **throws** IOException {  String line = **bufferedReader**.readLine();// 使用类内部维护的被装饰类的引用去读取数据  **if** (line == **null**) {  **return null**;  }  line = **"\""** + line + **"\""**;  **return** line;  } }   // 装饰类（带行号的缓存输入字符流）需求1： 编写一个类拓展BufferedReader的功能， 增强readLine方法返回 的字符串带有行号。 **class** BufferedReaderLineNum2 **extends** BufferedReader {   BufferedReader **bufferedReader**;  **int count** = 1;   **public** BufferedReaderLineNum2(BufferedReader bufferedReader) {  **super**(bufferedReader);  **this**.**bufferedReader** = bufferedReader;  }   @Override  **public** String readLine() **throws** IOException {  String line = **bufferedReader**.readLine();// 使用类内部维护的被装饰类的引用去读取数据  **if** (line == **null**) {  **return** line;  }  line = **count** + **" "** + line;  **count**++;  **return** line;  } } |

## 5.练习

* 需求：

一家三口每个人都会工作，

儿子的工作就是画画，

母亲的工作就是在儿子的基础上做一个增强，不单止可以画画，还可以上涂料。

爸爸的工作就是在妈妈基础上做了增强，就是上画框。

* 分析

1. 一家三口都会工作（Work 基类）
2. 儿子会画画 （Son draw）
3. 母亲会画画和涂料（Mother draw dope）
4. 父亲 会做画框（Father frame）