**哈尔滨工业大学（威海）**

**Java语言实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | Java语言 | 课程编号 | SE33601 |
| 实验名称 | 类定义及继承的应用 | | |
| 实验类型 | 验证与编程 | 实验学时 | 2学时 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 171110516 | 姓名 | 刘宇奇 | 开始时间 | 2018.09.21 |
| 成绩 |  | | | 结束时间 | 2018.09.28 |
| 批语 |  | | | | |

目录

[1 实验目的、内容和要求 1](#_Toc524165538)

[1.1 实验目的 1](#_Toc524165539)

[1.2 实验内容 1](#_Toc524165540)

[1.3 实验要求 1](#_Toc524165541)

[2 实验环境与实验前准备 2](#_Toc524165542)

[2.1 硬件环境 2](#_Toc524165543)

[2.2 软件环境 2](#_Toc524165544)

[2.3 知识准备 2](#_Toc524165545)

[2.4 Eclipse使用自学 2](#_Toc524165546)

[2.5 StarUML使用自学 2](#_Toc524165547)

[3 实验内容 3](#_Toc524165548)

[3.1 需求描述 3](#_Toc524165549)

[3.2 具体类设计 4](#_Toc524165550)

[3.3 利用继承消除重复代码 6](#_Toc524165551)

[3.4 类似方法的统一处理 10](#_Toc524165552)

[3.5 防止流程篡改 15](#_Toc524165553)

[3.5 判断是否满足OCP 16](#_Toc524165554)

# 1 实验目的、内容和要求

## 1.1 实验目的

面向对象编程强调“封装”，“继承”和“多态”，这三者被称为面向对象的三大特性。数据和与数据相关的操作被包装成对象，每一种对象是相对完整和独立的。对象可以有派生的类型，派生的类型可以覆盖（或重载）原本已有的操作。从而达成更好的内聚性，即一种对象做好一件（或者一类相关的）事情，对象内部的细节外面世界不关心也看不到；以及降低耦合性，即不同种类的对象之间相互的依赖尽可能降低。而所有的这些，都有助于达成一个崇高的目标，就是可复用性。

1. 掌握类、类的属性和方法
2. 掌握继承的基本特性和超类与子类的定义方法
3. 掌握Eclipse的基本操作
4. 掌握StarUML绘制简单类图

## 1.2 实验内容

1. 下载、安装Eclipse
2. 下载安装StarUML
3. 利用StarUML进行具体类的设计，并使用Eclipse实现代码；
4. 利用StarUML对继承的类系进行设计，并使用Eclipse实现代码；
5. 利用StarUML对抽象类进行设计，并使用Eclipse实现代码；
6. 利用StarUML绘制扩展的类图，分析是否符合OCP，并使用Eclipse实现代码；

## 1.3 实验要求

1. 本次实验采用Eclipse快速开发工具；
2. 代码按照《Java面向对象程序设计》附录1中的Java编写规范书写；
3. 除了增删表格的行外，不允许改变本文档的页边距及其他格式；
4. 程序代码不得截图；
5. 程序代码采用consolas字体，五号；
6. 程序代码要求直接从Eclipse中拷贝代码，并粘贴到本报告相应的表格内
7. 实验结果可以截图
8. 中文宋体五号；
9. 由于每个同学写的程序长短不一，必然导致页码的变换，因此在正式提交本文档前，需要在目录部分，右键更新目录的页码，否则会出现“未定义标签”错误。

# 2 实验环境与实验前准备

## 2.1 硬件环境

本次实验测试环境中，对硬件环境要求不高，目前绝大多数的PC均能满足实验要求，包括软件学院实验室中的硬件配置，所用软件环境如表1所示。

表1 实验环境要求

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 |  | 名称 | 版本 |
| 1 | 硬件环境 | CPU | Intel core i5 或相当水平的其他CPU |
| 2 | RAM | 2G及以上 |
| 3 | Hard Disk | 150G及以上 |
| 4 | 软件环境 | 操作系统 | Win系列；Mac系列OS X ，Linux均可 |
| 5 | Java | jdk1.6或以上版本 |
| 6 | Eclipse | Eclipse3.4以上版本 |
| 7 | 设计工具 | StarUML | 5.0版本 |
| 8 | 文档编写环境 | 文档 | Word2013或以上版本；Wps2017 |

## 2.2 软件环境

本次实验所有的软件和文档，均可在如下网址下载：

https://pan.baidu.com/s/1hQmEHBcbigflgceG-bfrNQ 密码：fec7，请根据需要自行下载，包括：

* Eclipse
* StarUML

## 2.3 知识准备

本实验对课堂内容进行了部分扩展，需自行学习第5、15、17章以及附录F2,F3,F4才能完成本实验。

## 2.4 Eclipse使用自学

参考《Java面向对象设计讲义》第5章，附录F3，F4内容即可。

## 2.5 StarUML使用自学

参考《Java面向对象设计讲义》附录F2内容即可。

# 3 实验内容

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库，它产生于距今五十年前，随着信息技术和市场的发展，特别是二十世纪九十年代以后，数据管理不再仅仅是存储和管理数据，而转变成用户所需要的各种数据管理的方式。数据库有很多种类型，从最简单的存储有各种数据的表格到能够进行海量数据存储的大型数据库系统都在各个方面得到了广泛的应用。

## 3.1 需求描述

为了满足不同用户数据量的要求，某软件产品拟采用多种数据库管理系统对数据进行管理。为便于实现用户定制，采用MySQL，Oracle，DB2三种数据库。

数据库的连接操作严格按照如下顺序进行：

* MySQL的数据库连接操作
  1. 注册MySQL驱动（Registration MySQL Driver）
  2. 建立MySQL连接（Establish MySQL Connection）
  3. 创建执行语句（Create Execute Statements）
  4. 执行语句（Execute Statement）
  5. 处理结果（Processing Results）
  6. 释放资源 （Free Resource）
* Oracle的数据库连接操作

1. 注册Oracle驱动（Registration Oracle Driver）
2. 建立Oracle连接（Establish Oracle Connection）
3. 创建执行语句（Create Execute Statements）
4. 执行语句（Execute Statement）
5. 处理结果（Processing Results）
6. 释放资源 （Free Resource）

* DB2的数据库连接操作

1. 注册DB2驱动（Registration DB2 Driver）
2. 建立DB2连接（Establish DB2 Connection）
3. 创建执行语句（Create Execute Statements）
4. 执行语句（Execute Statement）
5. 处理结果（Processing Results）
6. 释放资源 （Free Resource）

要求根据以上流程，为公司的数据库连接编写代码，实现三种数据库的连接。为设计和实现方便，假设以上数据库的连接操作步骤中没有输入参数，也没有返回值，方法体请参考《讲义》第17章的案例“咖啡与茶冲泡机案例”中的做法。

## 3.2 具体类设计

* **实验内容1：参考讲义中《咖啡与茶冲泡机案例》，对Mysql，Oracle，DB2三种数据库的连接进行设计**。首先使用starUML对具体类进行设计，然后再编写测试类进行测试。为了模拟真正的数据库连接操作，每个步骤需要参考“咖啡与茶冲泡机案例”，用打印语句输出操作信息。

**使用StarUML软件绘制具体类的类图。**

|  |
| --- |
|  |

根据3.1节中的需求描述及类图设计，完成各具体类的代码：

**MySQL的数据库连接操作类代码：**

|  |
| --- |
|  |

**Oracle的数据库连接操作类代码：**

|  |
| --- |
|  |

**DB2的数据库连接操作类代码：**

|  |
| --- |
|  |

**测试类的代码：**

|  |
| --- |
| **package** class\_hk;  **public** **class** test01 {  **public** **static** **void** main(String args[]) {  MySQL test1 = **new** MySQL();  test1.Registration\_MySQL\_Driver();  test1.Establish\_MySQL\_Connection();  test1.Create\_Execute\_Statements();  test1.Execute\_Statement();  test1.Processing\_Results();  test1.Free\_Resource();  }  } |

**运行结果截图：**

|  |
| --- |
|  |

## 3.3 利用继承消除重复代码

3.3.1 继承的定义

面向对象开发的目的就是为了消除掉重复的代码。因此，按照3.2节中的设计，类和类之间没有任何关系，都需要单独定义所有的方法和属性的编写套路，已经不能满足于这种现实问题，就需要采用继承的形式。

**继承的语法如下：**

|  |
| --- |
| [可见性] class 子类 extends 父类 {  } |

有时，父类也称为超类（super class），子类也称为派生类。

3.3.2 继承的特征

继承关系是传递的。若类C继承类B，类B继承类A，则类C既有从类B那里继承下来的属性与方法，也有从类A那里继承下来的属性与方法，还可以有自己新定义的属性和方法。

继承来的属性和方法尽管是隐式的，但仍是类C的属性和方法。继承是在一些比较一般的类的基础上构造、建立和扩充新类的最有效的手段。

继承简化了人们对事物的认识和描述，能清晰体现相关类间的层次结构关系。

继承提供了软件复用功能。若类B继承类A，那么建立类B时只需要再描述与基类(类A)不同的少量特征（数据成员和成员方法）即可。这种做法能减小代码和数据的冗余度，大大增加程序的重用性。

Java出于安全性和可靠性的考虑，仅支持单重继承，而通过使用接口机制来实现多重继承。

子类可以继承父类的全部操作（属性、方法），但对于所有的公共操作是可以直接继承，而所有的私有操作和属性是无法直接继承的，而是通过其他的方式间接访问。

3.3.3 利用继承优化设计

* **实验内容2：为达到代码复用的目的，需要对本报告第3.2节中的设计进行优化，对具体类进行抽象，从而形成继承结构，利用继承的特性消除重复代码。**

根据需求，优化3.2节中的设计，消除重复代码，类图绘制如下：

|  |
| --- |
|  |

超类的代码实现

|  |
| --- |
| **package** class\_hk02;  **abstract** **public** **class** DataBase {  **public** **void** Create\_Execute\_Statements() {  System.***out***.println("Create Execute Statements");  }  **public** **void** Execute\_Statement() {  System.***out***.println("Execute Statement");  }  **public** **void** Processing\_Results() {  System.***out***.println("Processing Results");  }  **public** **void** Free\_Resource() {  System.***out***.println("Free Resource");  }  } |

MySQL的数据库连接操作类代码：

|  |
| --- |
| **package** class\_hk02;  **public** **class** MySQL **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration MySQL Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish MySQL Connection");  }  } |

Oracle的数据库连接操作类代码：

|  |
| --- |
| **package** class\_hk02;  **public** **class** Oracle **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration Oracle Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish Oracle Connection");  }  } |

DB2的数据库连接操作类代码：

|  |
| --- |
| **package** class\_hk02;  **public** **class** DB2 **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration DB2 Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish DB2 Connection");  }  } |

测试类的代码：

|  |
| --- |
| **package** class\_hk02;  **public** **class** test {  **public** **static** **void** main(String args[]) {  MySQL test01 = **new** MySQL();  test01.Create\_Execute\_Statements();  test01.Establish\_Connection();  test01.Registration\_Driver();  }  } |

运行结果截图：

|  |
| --- |
|  |

## 3.4 类似方法的统一处理

分析3.1节中的需求，发现三种数据库连接的区别在于第1步和第2步上：

* MySQL的数据库连接操作

1) 注册MySQL驱动（Registration MySQL Driver）

2) 建立MySQL连接（Establish MySQL Connection）

-------

* Oracle的数据库连接操作

1. 注册Oracle驱动（Registration Oracle Driver）
2. 建立Oracle连接（Establish Oracle Connection）

-------

* DB2的数据库连接操作

1. 注册DB2驱动（Registration DB2 Driver）
2. 建立DB2连接（Establish DB2 Connection）

-------

是否可以将三种数据库连接的第1）2）两个步骤的命名分别统一处理？例如：分别将两个步骤统一命名为：

1) 注册驱动（Registration Driver）

2) 建立连接（Establish Connection）

但数据库连接操作中的第1）步和第2）步的代码是不同的，它们需要加载不同的驱动程序，连接的URL和端口也不同。

Question1：如果将MySQL，Oracle，DB2中的第1），第2）命名统一为注册驱动（Registration Driver）和建立连接（Establish Connection），那么这两个步骤的实现方法是否可以放到超类？

请在下面的表格中阐述你的理由。

|  |
| --- |
| 不可以,因为前两个步骤注册的驱动和建立的连接都不同,数据库类型不同,须指明不同类型的数据库,而统一放在超类中实现在继承的时候就没有各自的特征,因此应该设计为abstract方法,在其子类中再实现. |

如果将MySQL，Oracle，DB2中的第1），第2）步命名统一为注册驱动（Registration Driver）和建立连接（Establish Connection），如案例“咖啡与茶冲泡机”中控制冲泡流程的方法prepareRecipe()一样，本实验中的三个具体类的控制调用顺序的方法就会完全一样，因此可以放到超类中实现，从而子类可以通过继承实现代码的复用。

但如前所述，数据库连接操作中的步骤1)和2)的代码是不同的，只能放到子类实现。这样就形成了一个矛盾：超类中的流程控制方法需要调用步骤1)和2)，但这两个步骤并没有在超类中，因此流程控制方法必然出现错误。

* **实验内容3：统一步骤1)和2)的命名以后，如何设计才能够将流程控制方法放到超类进行继承？复习C++中纯虚函数的概念，并参考讲义第15章“抽象类接口与多态”中有关抽象类概念，进行设计并实现代码。**

实验内容3的类图设计

|  |
| --- |
|  |

**对以上设计的类图进行实现，各类代码如下：**

超类的代码实现

|  |
| --- |
| **abstract** **public** **class** DataBase {  **public** **abstract** **void** Registration\_Driver();  **public** **abstract** **void** Establish\_Connection();  **public** **void** Create\_Execute\_Statements() {  System.***out***.println("Create Execute Statements");  }  **public** **void** Execute\_Statement() {  System.***out***.println("Execute Statement");  }  **public** **void** Processing\_Results() {  System.***out***.println("Processing Results");  }  **public** **void** Free\_Resource() {  System.***out***.println("Free Resource");  }  } |

MySQL的数据库连接操作类代码：

|  |
| --- |
| **public** **class** MySQL **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration MySQL Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish MySQL Connection");  }  } |

Oracle的数据库连接操作类代码：

|  |
| --- |
| **public** **class** Oracle **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration Oracle Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish Oracle Connection");  }  } |

DB2的数据库连接操作类代码：

|  |
| --- |
| **public** **class** DB2 **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration DB2 Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish DB2 Connection");  }  } |

测试类的代码：

|  |
| --- |
| **public** **class** test {  **public** **static** **void** main(String args[]) {  MySQL test01 = **new** MySQL();  Oracle test02 = **new** Oracle();  DB2 test03 = **new** DB2();  test01.Registration\_Driver();  test01.Establish\_Connection();    test02.Registration\_Driver();  test02.Establish\_Connection();    test03.Registration\_Driver();  test03.Establish\_Connection();    test01.Create\_Execute\_Statements();  test01.Execute\_Statement();  test01.Processing\_Results();  test01.Free\_Resource();  test02.Create\_Execute\_Statements();  test02.Execute\_Statement();  test02.Processing\_Results();  test02.Free\_Resource();  test03.Create\_Execute\_Statements();  test03.Execute\_Statement();  test03.Processing\_Results();  test03.Free\_Resource();  }  } |

运行结果截图：

|  |
| --- |
|  |

## 3.5 防止流程篡改

* **实验内容4：根据课堂讲授，如何防止流程控制方法被子类篡改？请修改超类代码，并测试，证明可以实现超类中的流程控制方法不能被子类覆盖。**

超类的代码实现：

|  |
| --- |
| **abstract** **public** **class** DataBase {  **public** **abstract** **void** Registration\_Driver();  **public** **abstract** **void** Establish\_Connection();  **public** **void** Create\_Execute\_Statements() {  System.***out***.println("Create Execute Statements");  }  **public** **void** Execute\_Statement() {  System.***out***.println("Execute Statement");  }  **public** **void** Processing\_Results() {  System.***out***.println("Processing Results");  }  **public** **void** Free\_Resource() {  System.***out***.println("Free Resource");  }  **final** **public** **void** Process() {  Registration\_Driver();  Establish\_Connection();  Create\_Execute\_Statements();  Execute\_Statement();  Processing\_Results();  Free\_Resource();  }  } |

MySQL的数据库连接操作类代码：

|  |
| --- |
| **public** **class** Access **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration Access Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish Access Connection");  }  **public** **void** Process() {  Registration\_Driver();  Establish\_Connection();  Create\_Execute\_Statements();  Execute\_Statement();  Free\_Resource();  Processing\_Results();  }  } |

测试类的代码：

|  |
| --- |
| **public** **class** test {  **public** **static** **void** main(String args[]) {  MySQL test01 = **new** MySQL();  test01.Process();  }  } |

运行结果截图：

|  |
| --- |
|  |

## 3.5 判断是否满足OCP

 “开-闭”原则（Open-Closed Principle – OCP）：一个软件实体应当对扩展开放，对修改关闭( Software entities should be open for extension, but closed for modification)。即在设计一个模块的时候，应当使这个模块可以在不被修改的前提下被扩展。该原则成为软件设计是否优秀的一个标准。

* **实验内容5：为测试你的软件是否设计优秀，请添加一个新的数据库Access的连接类，绘制类图，编写Access类，在测试类中构建Access对象，并测试该对象是否能够满足数据库连接的6个步骤。**

绘制新加入Access数据库后全体类的类图：

|  |
| --- |
|  |

Access的数据库连接操作类代码：

|  |
| --- |
| **public** **class** Access **extends** DataBase {  **public** **void** Registration\_Driver() {  System.***out***.println("Registration Access Driver");  }  **public** **void** Establish\_Connection() {  System.***out***.println("Establish Access Connection");  }  } |

测试类的代码：

|  |
| --- |
| **public** **class** test {  **public** **static** **void** main(String args[]) {  Access test01 = **new** Access();  test01.Registration\_Driver();  test01.Establish\_Connection();  test01.Create\_Execute\_Statements();  test01.Execute\_Statement();  test01.Processing\_Results();  test01.Free\_Resource();  }  } |

运行结果截图：

|  |
| --- |
|  |

设计结论：

|  |
| --- |
| 满足开闭原则，因为模块和函数对扩展是开放的，添加一个Access子类的扩展实现了变化，但并没有对原来的代码进行修改，所以未修改原有代码，但却成功扩展，而且代码可复用，满足开闭原则。 |

# 4 实验总结

在下面的空表中撰写本次实验的总结

|  |
| --- |
| 本次实验中，首先学会了利用uml图来对具体类进行设计，对一个程序进行了反复的修改，深入理解并巩固了类的继承设计，理解了开闭原则的重要性。在设计的过程当中，要注重代码的复用，以及扩展性，ocp原则的遵循等等。 |