
廈門大學



信息学院软件工程系

《JAVA 程序设计》实验报告

实验 7

姓名：黄勛

学号：22920212204392

学院：信息学院

专业：软件工程

完成时间：2023.4.11

一、实验目的及要求

- 熟悉异常处理
- 熟悉泛型方法和泛型类

二、实验题目及实现过程

实验环境：Windows 10 21H2、jdk17、utf-8 编码

(基本题目) 题目一

(一) 实验题目

- ◆ 完善 LAB6，用异常处理语句处理输入类型不匹配、数据转换类型异常、创建对象时初始值不合法等问题。

(二) 实现过程

程序的主要修改逻辑如下：

1. 处理输入类型不匹配。

直接根据输入的数据进行抛出异常。

```
// 检查用户输入的车辆信息是否符合要求
if (infos.length == 6) {
    if (!infos[0].equals("小汽车")) {
        System.out.println("第一个参数必须为小汽车或者卡车");
        throw new Exception();
    }
    if (!infos[5].equals("2厢") && !infos[5].equals("3厢")) {
        throw new Exception();
    }
}
```

2. 处理数据转换类型异常等问题。

```

134      } // 处理类初始化错误的异常
135      catch (ExceptionInInitializerError e){
136          System.out.println("程序出现错误");
137          exit = true;
138      } // 处理数据类型不匹配的异常
139      catch (InputMismatchException e){
140          System.out.println("请输入1-4的整数");
141          scanner.nextLine();
142      } // 处理数组越界的异常
143      catch (ArrayIndexOutOfBoundsException e){
144          System.out.println("请输入1-4的整数");
145          scanner.nextLine();
146      } // 处理除数为0的异常
147      catch (ArithmeticException e){
148          System.out.println("请输入1-4的整数");
149          scanner.nextLine();
150      } // 处理空指针的异常
151      catch (NullPointerException e){
152          System.out.println("请输入1-4的整数");
153          scanner.nextLine();
154      } // 处理其他异常
155      catch (Exception e){
156          System.out.println("请输入1-4的整数");
157          scanner.nextLine();
158      }

```

3. 创建对象时初始值不合法。

利用 `IllegalArgumentException` 判断初始参数不合法

```

70      catch (IllegalArgumentException e) {
71          System.out.println("创建不成功");
72      } // 创建对象时输入的参数个数不合法
73      catch (InputMismatchException e) {
74          System.out.println("创建不成功");
75      } // 创建对象时输入的参数不合法
76      catch (Exception e) {
77          System.out.println("创建不成功");
78      }
79      } while (true);
80      }

```

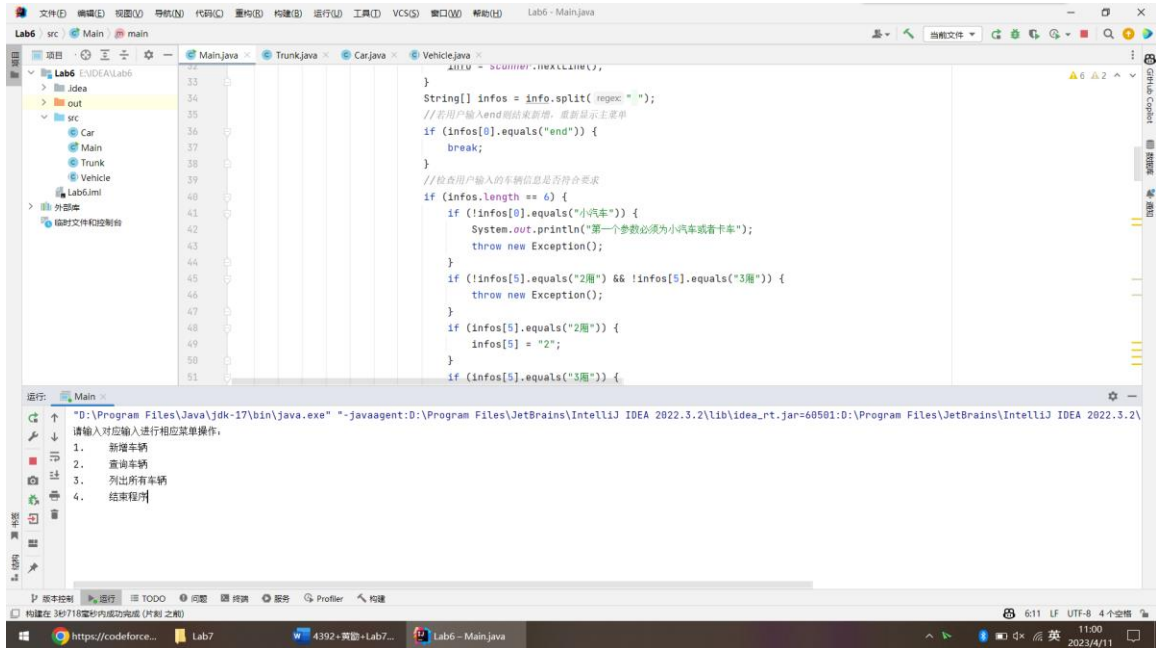
```

8      public Car(String brand, String color, int year, int passenger, int carNum) {
9          super(brand, color, year);
10         try{
11             this.passenger = passenger;
12             if (carNum != 2 && carNum != 3) {
13                 throw new Exception("车厢数只能为2或3");
14             }
15             // 检查车厢数 (2厢或3厢)
16             this.carNum = carNum;
17         } catch (Exception e){
18             System.out.println(e.getMessage());
19         }
20     }

```

(三) 过程截图

最终结果 (全屏截图)



题目二

(一) 实验题目

- ◆ (构造方法失败) 编写一个程序, 给出一个构造方法, 它将关于构造方法失败的信息传递给一个异常处理器。定义一个 SomeClass 类, 它在构造方法中抛出异常。程序应创建一个 SomeClass 型的对象, 并捕获由这个构造方法抛出的异常。

(二) 实现过程

思路: 先定义 SomeClass 类

```
1 //定义一个 SomeClass 类, 它在构造方法中抛出异常。
2 用法
2 public class SomeClass {
3     1 个用法
3     public SomeClass() throws Exception {
4         throw new Exception("构造方法失败");
5     }
6 }
```

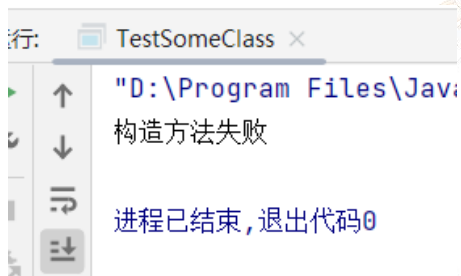
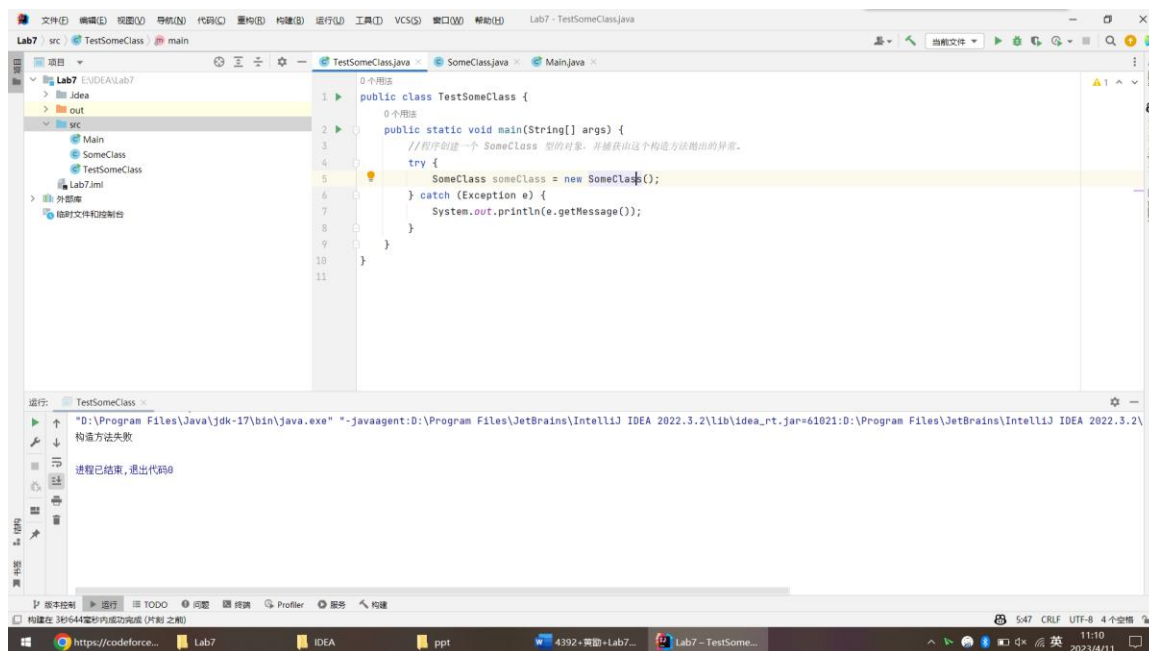
接着定义 TestSomeClass 类

```
1 0 个用法
1 public class TestSomeClass {
2     0 个用法
2     public static void main(String[] args) {
3         //程序创建一个 SomeClass 型的对象, 并捕获由这个构造方法抛出的异常。
4         try {
5             SomeClass someClass = new SomeClass();
6         } catch (Exception e) {
7             System.out.println(e.getMessage());
8         }
9     }
10 }
```

完成后便可以测试了。

(三) 过程截图

最终结果 (全屏截图)



题目三

(一) 实验题目

- ◆ (重抛异常) 编写一个演示重抛异常的程序。定义两个方法 someMethod 和 someMethod2, someMethod2 方法的功能就是抛出一个异常。someMethod 方法调用 someMethod2, 捕获一个异常并重抛它。用 main 方法调用 someMethod 方法, 并捕获被重抛的异常。输出这个异常的栈踪迹。

(二) 实现过程

思路: 设计两个方法, 具体实现参照题意:

```
1  ▶ public class RethrowException {  
    1 个用法  
2      public static void someMethod2() throws Exception {  
3          System.out.println("someMethod2().");  
4          throw new Exception("First Throw");  
5      }  
    1 个用法  
6      public static void someMethod() throws Exception {  
7          System.out.println("someMethod().");  
8          try  
9          {  
10             someMethod2();  
11         }  
12         catch(Exception e)  
13         {  
14             System.out.println("Rethrow the exception.");  
15             e.printStackTrace();  
16         }  
17     }  
}
```

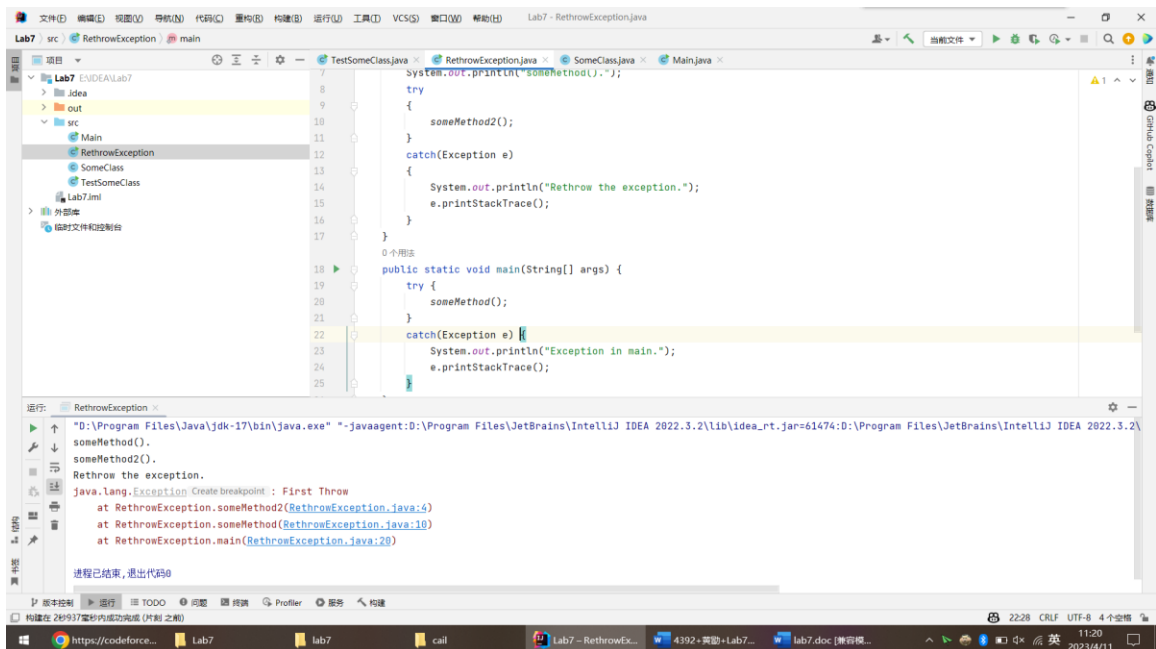
编写 main 方法，调用 someMethod 方法，并捕获被重抛的异常。输出这个异常的栈踪迹。

```
18  ▶ public static void main(String[] args) {  
19      try {  
20          someMethod();  
21      }  
22      catch(Exception e) {  
23          System.out.println("Exception in main.");  
24          e.printStackTrace();  
25      }  
26  }
```

程序主菜单使用 do-while 循环实现，循环条件为 exit 变量，如果用户选择了“4. 结束程序”，则将 exit 设置为 true，程序结束。

(三) 过程截图

最终结果（全屏截图）



题目四

(一) 实验题目

- 自定义异常的定义、抛出和捕获:
 - ◆ (1) 自定义两个异常类: 非法姓名异常 `IllegalNameException` 和非法地址异常 `IllegalAddressException`。
 - ◆ (2) 定义 `Student` 类包含 `name` 和 `address` 属性, 和 `setName`、`setAddress` 方法, 当姓名长度小于 1 或者大于 5 时抛出 `IllegalNameException`, 当地址中不含有“省”或者“市”关键字时抛出 `IllegalAddressException`。
 - ◆ (3) 编写程序抛出这两种异常, 在 `main` 方法中进行捕获并合理地处理。

(二) 实现过程

思路：（1）自定义两个异常类：非法姓名异常 `IllegalNameException` 和非法地址异常 `IllegalAddressException`。

The image shows two screenshots of a Java IDE. The top screenshot displays the `IllegalNameException.java` file. It contains the following code:

```
1 package q4;
2
3 public class IllegalNameException extends Exception{
4     public IllegalNameException() {
5         super();
6     }
7
8     public IllegalNameException(String message) {
9         super(message);
10    }
11
12    public IllegalNameException(String message, Throwable cause) {
13        super(message, cause);
14    }
15
16    public IllegalNameException(Throwable cause) {
17        super(cause);
18    }
19 }
```

The bottom screenshot displays the `IllegalAddressException.java` file. It contains the following code:

```
1 package q4;
2
3 public class IllegalAddressException extends Exception{
4     public IllegalAddressException() {
5         super();
6     }
7
8     public IllegalAddressException(String message) {
9         super(message);
10    }
11
12    public IllegalAddressException(String message, Throwable cause) {
13        super(message, cause);
14    }
15
16    public IllegalAddressException(Throwable cause) {
17        super(cause);
18    }
19 }
```

(2) 定义 Student 类包含 name 和 address 属性，和 setName、setAddress 方法，

```

7      public class Student {
8          3个用法
9          private String name;
10         3个用法
11         private String address;
12
13         1个用法
14         public Student(String name, String address) {
15             this.name = name;
16             this.address = address;
17         }
18
19         1个用法
20         public String getName() {
21             return name;
22         }

```

当姓名长度小于 1 或者大于 5 时抛出 IllegalArgumentException，

```

20 @      public void setName(String name) throws IllegalArgumentException {
21         if (name.length() < 1 || name.length() > 5) {
22             throw new IllegalArgumentException("The length of name must be >=1 and <=5");
23         }
24         this.name = name;
25     }

```

当地址中不含有“省”或者“市”关键字时抛出 IllegalAddressException。

```

31 @      public void setAddress(String address) throws IllegalAddressException {
32         if (!(address.contains("省") || address.contains("市"))) {
33             throw new IllegalAddressException("Address must contain \"省\" or \"市\"");
34         }
35         this.address = address;
36     }

```

重写 toString 便于输出

```

38         @Override
39         public String toString() {
40             return String.format("Name: %s\nAddress: %s\n", getName(), getAddress());
41         }

```

(3) 编写程序抛出这两种异常，在 main 方法中进行捕获并合理地处理。

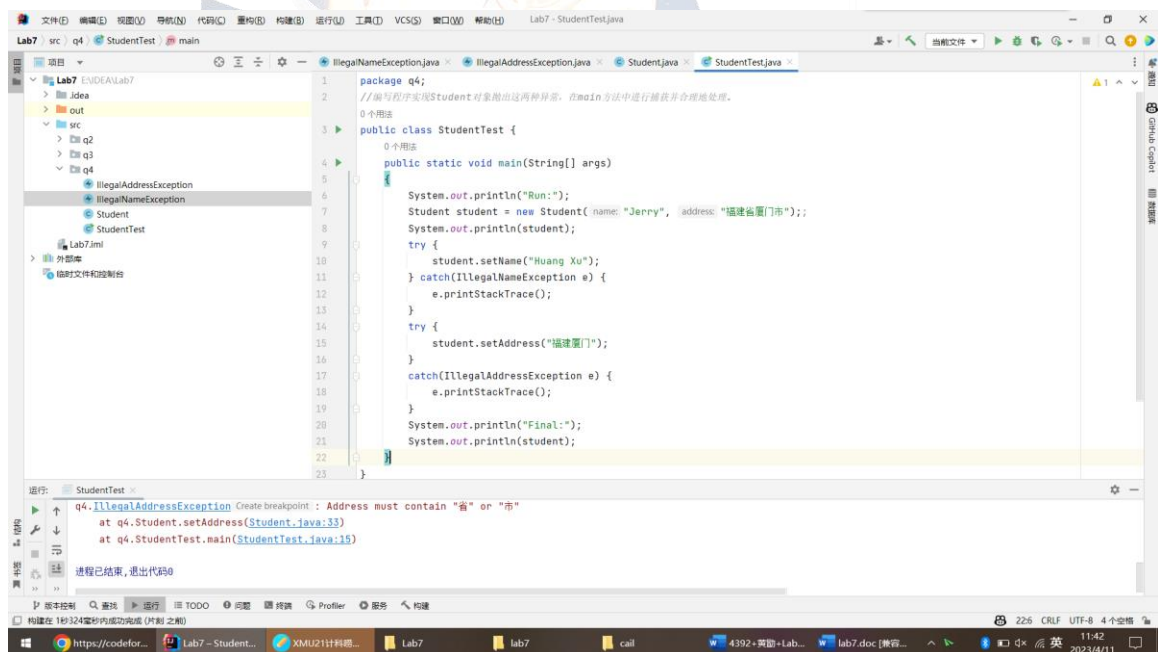
```

1 package q4;
2 //编写程序实现Student对象抛出这两种异常，在main方法中进行捕获并合理地处理。
3 0个用法
4 public class StudentTest {
5     0个用法
6     public static void main(String[] args)
7     {
8         System.out.println("Run:");
9         Student student = new Student( name: "Jerry", address: "福建省厦门市");
10        System.out.println(student);
11        try {
12            student.setName("Huang Xu");
13        } catch(IllegalArgumentException e) {
14            e.printStackTrace();
15        }
16        try {
17            student.setAddress("福建厦门");
18        }
19        catch(IllegalAddressException e) {
20            e.printStackTrace();
21        }
22        System.out.println("Final:");
23        System.out.println(student);
24    }
25 }

```

(三) 过程截图

最终结果（全屏截图）





```
运行: StudentTest x
"D:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ
Run:
Name: Jerry
Address: 福建省厦门市

Final:
Name: Jerry
Address: 福建省厦门市

q4.IllegalNameException Create breakpoint : The length of name must be >=1 and <=5
    at q4.Student.setName(Student.java:22)
    at q4.StudentTest.main(StudentTest.java:10)
q4.IllegalAddressException Create breakpoint : Address must contain "省" or "市"
    at q4.Student.setAddress(Student.java:33)
    at q4.StudentTest.main(StudentTest.java:15)

进程已结束,退出代码0
```

题目五

(一) 实验题目

- ◆ (泛型方法 isEqualTo) 编写 isEqualTo 方法的一个简单泛型版本。它用 equals 方法比较两个实参相等时返回 true, 否则返回 false。利用这个泛型方法, 在程序中调用 isEqualTo 处理各种内置的类型, 例如 Object 或 Integer。运行程序时, 传递给 isEqualTo 方法的对象会根据它们的内容或者所引用的对象进行比较吗?

(二) 实现过程

- ◆ 思路: 编写 isEqualTo 方法

用 equals 方法比较两个实参相等时返回 true, 否则返回 false

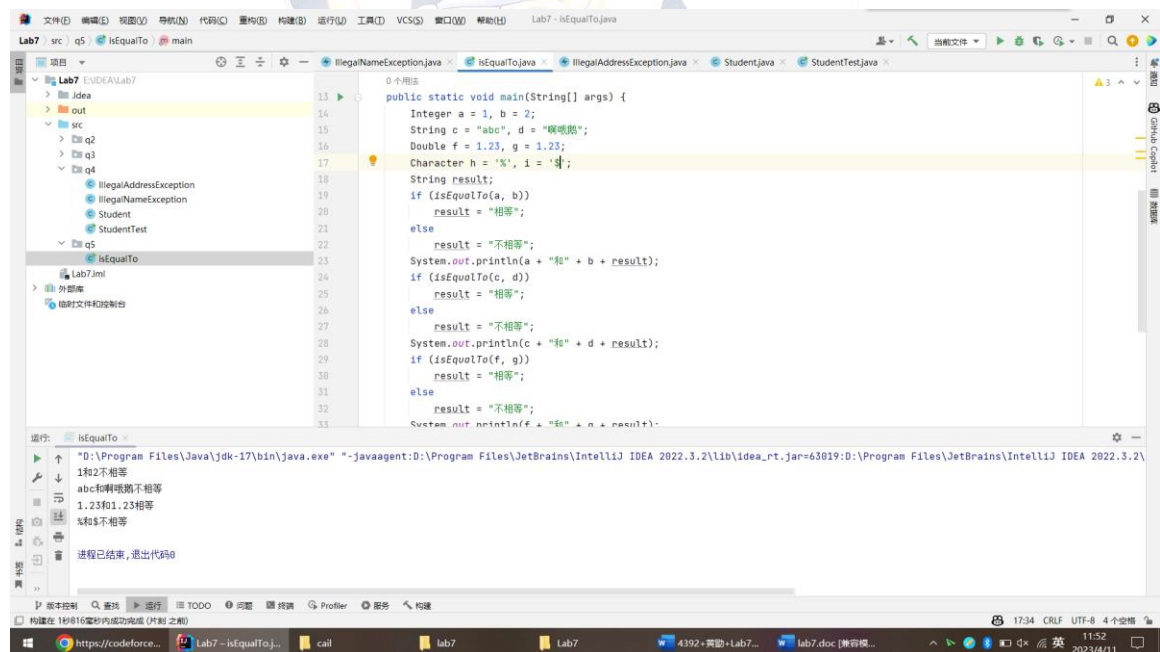
```
7 @  public static <T> boolean isEqualTo(T a, T b) {
8     if (a.equals(b))
9         return true;
10    else
11        return false;
12    }
```

在 main 方法中调用 isEqualTo 处理各种内置的类型, 例如 Object 或 Integer。

```
13 public static void main(String[] args) {
14     Integer a = 1, b = 2;
15     String c = "abc", d = "啊哦鹅";
16     Double f = 1.23, g = 1.23;
17     Character h = '%', i = '$';
18     String result;
19     if (isEqualTo(a, b))
20         result = "相等";
21     else
22         result = "不相等";
23     System.out.println(a + "和" + b + result);
24     if (isEqualTo(c, d))
25         result = "相等";
26     else
27         result = "不相等";
28     System.out.println(c + "和" + d + result);
29     if (isEqualTo(f, g))
30         result = "相等";
31     else
32         result = "不相等";
33     System.out.println(f + "和" + g + result);
34     if (isEqualTo(h, i))
35         result = "相等";
36     else
37         result = "不相等";
38     System.out.println(h + "和" + i + result);
39 }
```

(三) 过程截图

最终结果 (全屏截图)



运行程序时，传递给 `isEqualTo` 方法的对象会根据它们的内容或者所引用的对象进行比较吗？

经检验，会。

题目六

（一）实验题目

- ◆ （泛型类 `Pair`）编写一个泛型类 `Pair`，它有两个类型参数 `F` 和 `S`，分别代表一对值中第一个元素第二个元素的类型。为第一个元素和第二个元素添加 `get` 方法和 `set` 方法。（提示：类首部应当是 `public class Pair <F, S>.`）

（二）实现过程

思路：编写泛型类 `Pair`，添加 `get` 方法和 `set` 方法。

```
3  public class Pair<F, S> {  
4      private F a;  
5      private S b;  
6  
7      public F geta() {  
8          return a;  
9      }  
10  
11     public S getb() {  
12         return b;  
13     }  
14  
15     public void seta(F a) {  
16         this.a = a;  
17     }  
18  
19     public void setb(S b) {  
20         this.b = b;  
21     }  
}
```

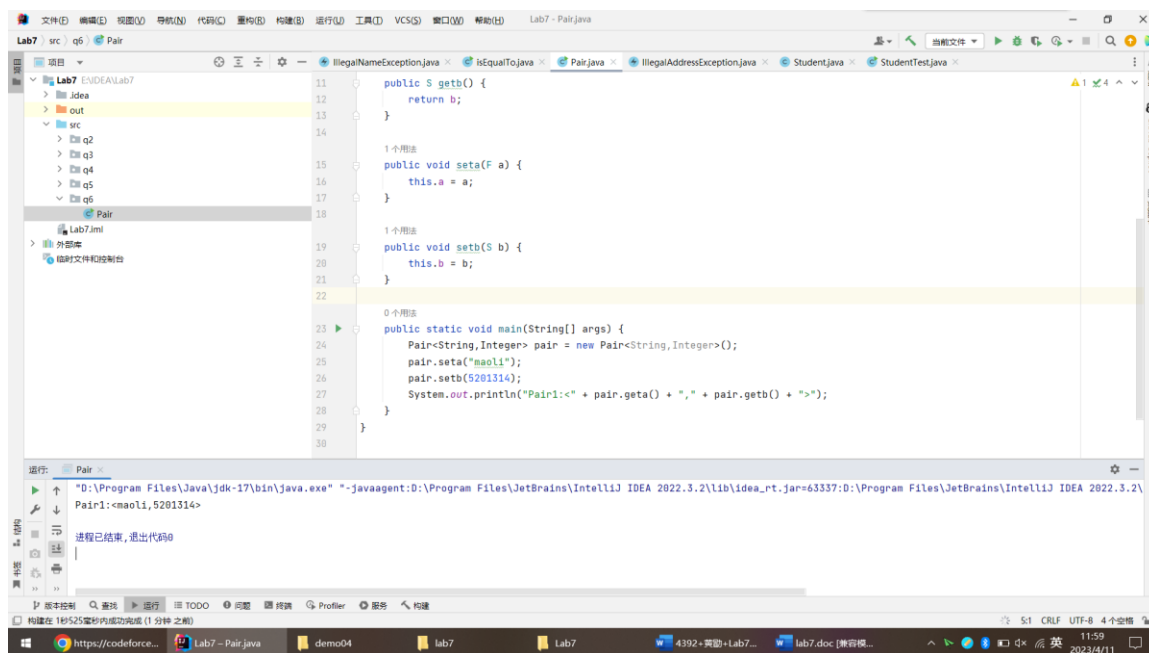


测试

```
23  public static void main(String[] args) {  
24      Pair<String,Integer> pair = new Pair<String,Integer>();  
25      pair.seta("maoli");  
26      pair.setb(5201314);  
27      System.out.println("Pair1:<" + pair.geta() + "," + pair.getb() + ">");  
28  }
```

(三) 过程截图

最终结果（全屏截图）



(选做题) 题目一

(一) 实验题目

- （混用组合和继承）将 `CommissionEmployee`—`BaseCommissionEmployee` 继承层次重新建模成一个 `Employee` 层次，使得每一种员工都具有不同 `CompensationModel` 对象。本练习中，要求重新实现 `CompensationModel` 类成为一个接口，提供的公共抽象方法 `earnings` 不包含参数，且返回一个 `double` 值。然后，实现了 `CompensationModel` 接口的如下几个类：
 - ◆ a) `SalariedCompensationModel` 类——对于周薪固定的员工，这个类需包含一个 `weeklySalary` 实例变量，且需要实现 `earnings` 方法，返回 `weeklySalary` 值。
 - ◆ b) `HourlyCompensationModel` 类——对于按时薪计酬（包括每周工作超过 40 小时的加班工资）的员工，这个类需包含 `wage` 和 `hours` 实例变量，且需根据工作的小时数实现 `earnings` 方法。
 - ◆ c) `CommissionCompensationModel` 类——对于按佣金付酬的员工，这个类需包含 `grossSales` 和 `commissionRate` 实例变量，还需要实现 `earnings` 方法，它返回 `grossSales x commissionRate` 的结果。
 - ◆ d) `BasePlusCommissionCompensationMode` 类——对按加金付酬的员工，这个类需包含 `grossSales` `commissionRate` 和 `baseSalary` 实例变量，

还需要实现 `earnings` 方法，它返回 `baseSalary + grossSales x commissionRate`。

- 在测试程序中，需为上面描述的每一种 `CompensationModel` 创建一个 `Employee` 对象，然后显示每一类员工的收入。接着，动态地改变员工的 `CompensationModel`，重新显示他的收入。

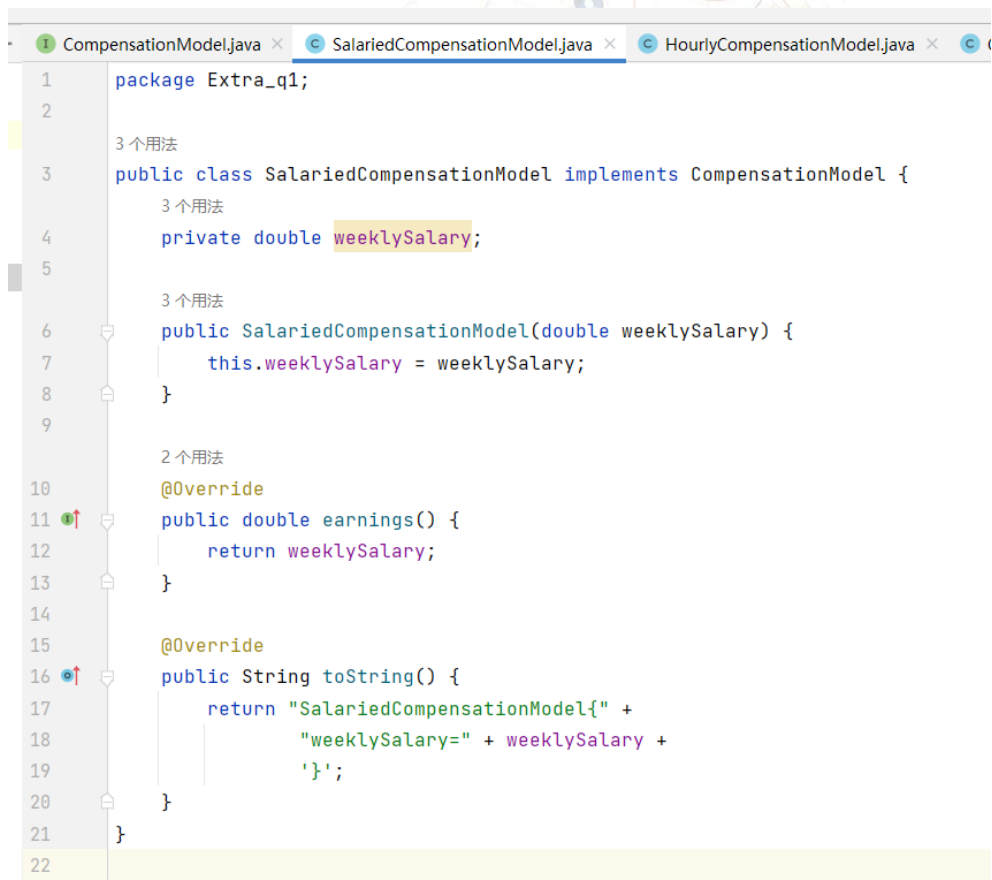
(二) 实现过程

思路：先实现 `CompensationModel` 接口，提供的公共抽象方法 `earnings` 不包含参数，且返回一个 `double` 值。



```
1 package Extra_q1;
2
3 public interface CompensationModel {
4     double earnings();
5 }
6
```

`SalariedCompensationModel` 类——对于周薪固定的员工，这个类需包含一个 `weeklySalary` 实例变量，且需要实现 `earnings` 方法，返回 `weeklySalary` 值。



```
1 package Extra_q1;
2
3 public class SalariedCompensationModel implements CompensationModel {
4     private double weeklySalary;
5
6     public SalariedCompensationModel(double weeklySalary) {
7         this.weeklySalary = weeklySalary;
8     }
9
10    @Override
11    public double earnings() {
12        return weeklySalary;
13    }
14
15    @Override
16    public String toString() {
17        return "SalariedCompensationModel{" +
18            "weeklySalary=" + weeklySalary +
19            '}';
20    }
21 }
22
```


HourlyCompensationModel 类——对于按时薪计酬（包括每周工作超过 40 小时的加班工资）的员工，这个类需包含 wage 和 hours 实例变量，且需根据工作的小时数实现 earnings 方法。

```
1 CompensationModel.java x SalariedCompensationModel.java x HourlyCompensationModel.java x Comm
3 public class HourlyCompensationModel implements CompensationModel {
4     5 个用法
5     private double wage;
6     5 个用法
7     private double hours;
8
9     0 个用法
10    public HourlyCompensationModel(double wage, double hours) {
11        this.wage = wage;
12        this.hours = hours;
13    }
14
15    2 个用法
16    @Override
17    public double earnings() {
18        if (hours <= 40) {
19            return wage * hours;
20        } else {
21            return 40 * wage + (hours - 40) * wage * 1.5;
22        }
23    }
24
25    @Override
26    public String toString() {
27        return "HourlyCompensationModel{" +
28            "wage=" + wage +
29            ", hours=" + hours +
```

CommissionCompensationModel 类——对于按佣金付酬的员工，这个类需包含 grossSales 和 commissionRate 实例变量，还需要实现 earnings 方法，它返回 grossSales x commissionRate 的结果。

```
1 CompensationModel.java x SalariedCompensationModel.java x HourlyCompensationModel.java x CommissionCompensationModel.java x
3 public class CommissionCompensationModel implements CompensationModel {
4     3 个用法
5     private double grossSales;
6     3 个用法
7     private double commissionRate;
8
9     1 个用法
10    public CommissionCompensationModel(double grossSales, double commissionRate) {
11        this.grossSales = grossSales;
12        this.commissionRate = commissionRate;
13    }
14
15    0 个用法
16    @Override
17    public double earnings() {
18        return grossSales * commissionRate;
19    }
20
21    @Override
22    public String toString() {
23        return "CommissionCompensationModel{" +
24            "grossSales=" + grossSales +
25            ", commissionRate=" + commissionRate +
26            '}';
27    }
28 }
```

BasePlusCommissionCompensationMode 类——对按加金付酬的员工，这个类需包含 grossSales commissionRate 和 baseSalary 实例变量，还需要实现 earnings 方法，它返回 baseSalary + grossSales x commissionRate。

```

1  java < SalariedCompensationModel.java < HourlyCompensationModel.java < CommissionCompensationModel.java < BasePlusCommissionCompensationModel.java
2  2 个用法
3  public class BasePlusCommissionCompensationModel implements CompensationModel {
4      3 个用法
5      private double grossSales;
6      3 个用法
7      private double commissionRate;
8      3 个用法
9      private double baseSalary;
10
11      2 个用法
12      public BasePlusCommissionCompensationModel(double grossSales, double commissionRate, double baseSalary) {
13          this.grossSales = grossSales;
14          this.commissionRate = commissionRate;
15          this.baseSalary = baseSalary;
16      }
17
18      2 个用法
19      @Override
20      public double earnings() {
21          return grossSales * commissionRate + baseSalary;
22      }
23
24      @Override
25      public String toString() {
26          return "BasePlusCommissionCompensationModel{" +
27              "grossSales=" + grossSales +
28              ", commissionRate=" + commissionRate +
29              "baseSalary=" + baseSalary +
30              "}";
31      }
32  }

```

Employee 类，包含基本信息以及每一种员工都具有不同 CompensationModel 对象。

```

1  Employee.java < EmployeeTest.java <
2  package Extra_q1;
3
4      8 个用法
5      public class Employee {
6          4 个用法
7          private String firstName;
8          4 个用法
9          private String lastName;
10         4 个用法
11         private String socialSecurityNumber;
12         6 个用法
13         public CompensationModel compensationModel;
14
15         4 个用法
16         public Employee(String firstName, String lastName, String socialSecurityNumber, CompensationModel compensationModel) {
17             this.firstName = firstName;
18             this.lastName = lastName;
19             this.socialSecurityNumber = socialSecurityNumber;
20             this.compensationModel = compensationModel;
21         }
22
23         0 个用法
24         public String getFirstName() {
25             return firstName;
26         }
27     }

```

编写 Test，创建四个 Employee 对象，然后显示每一类员工的收入。接着，动态地改变员工的 CompensationModel，重新显示他的收入。

```
Employee.java EmployeeTest.java
// 创建四个 Employee 对象
System.out.println("初始化员工信息:");
Employee[] employees = new Employee[4];
employees[0] = new Employee( firstName: "John", lastName: "Smith", socialSecurityNumber: "111-11-1111", new SalariedCompensationModel( weeklySalary: 800.0));
employees[1] = new Employee( firstName: "Lisa", lastName: "Barnes", socialSecurityNumber: "888-88-8888", new CommissionCompensationModel( grossSales: 10000.0, commissionRate: 0.06));
employees[2] = new Employee( firstName: "Mark", lastName: "Jones", socialSecurityNumber: "777-77-7777", new BasePlusCommissionCompensationModel( grossSales: 5000.0, baseSalary: 300.0, commissionRate: 0.04));
employees[3] = new Employee( firstName: "John", lastName: "Doe", socialSecurityNumber: "666-66-6666", new SalariedCompensationModel( weeklySalary: 1000.0));

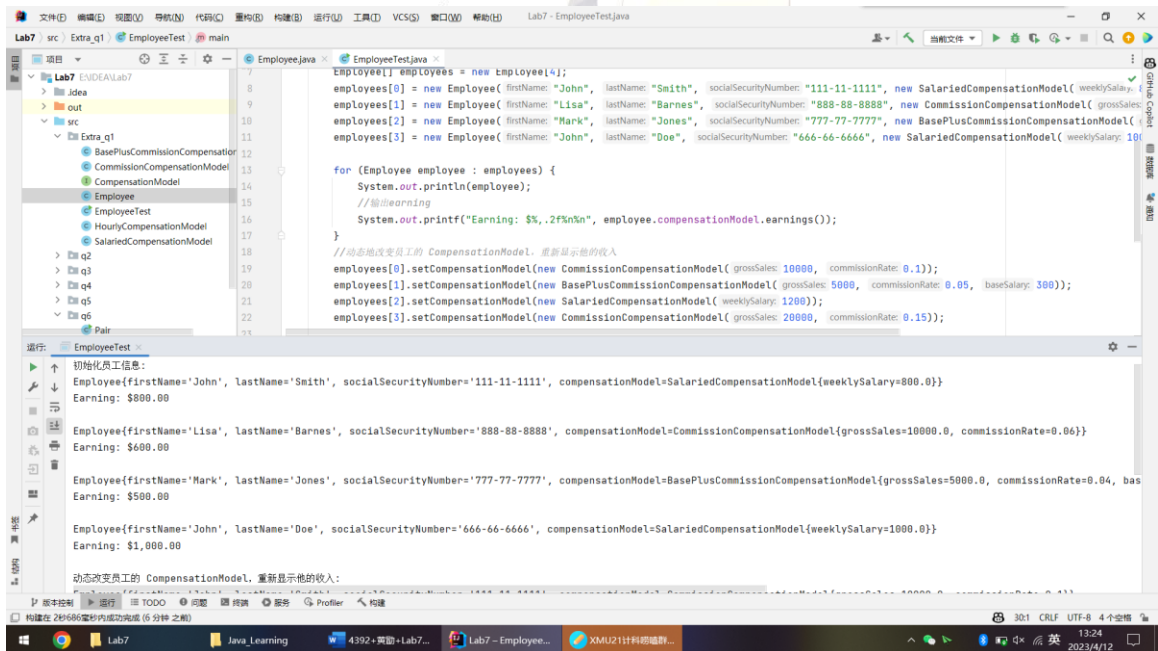
for (Employee employee : employees) {
    System.out.println(employee);
    // 输出 earning
    System.out.printf("Earning: $%,.2f\n", employee.compensationModel.earnings());
}

// 动态地改变员工的 CompensationModel, 重新显示他的收入
employees[0].setCompensationModel(new CommissionCompensationModel( grossSales: 10000, commissionRate: 0.1));
employees[1].setCompensationModel(new BasePlusCommissionCompensationModel( grossSales: 5000, commissionRate: 0.05, baseSalary: 300));
employees[2].setCompensationModel(new SalariedCompensationModel( weeklySalary: 1200));
employees[3].setCompensationModel(new CommissionCompensationModel( grossSales: 20000, commissionRate: 0.15));

System.out.println("动态改变员工的 CompensationModel, 重新显示他的收入:");
for (Employee employee : employees) {
    System.out.println(employee);
    // 输出 earning
    System.out.printf("Earning: $%,.2f\n", employee.compensationModel.earnings());
}
}
```

(三) 过程截图

最终结果 (全屏截图)



```
Lab7 - EmployeeTest.java
// 创建四个 Employee 对象
employees[0] = new Employee( firstName: "John", lastName: "Smith", socialSecurityNumber: "111-11-1111", new SalariedCompensationModel( weeklySalary: 800.0));
employees[1] = new Employee( firstName: "Lisa", lastName: "Barnes", socialSecurityNumber: "888-88-8888", new CommissionCompensationModel( grossSales: 10000.0, commissionRate: 0.06));
employees[2] = new Employee( firstName: "Mark", lastName: "Jones", socialSecurityNumber: "777-77-7777", new BasePlusCommissionCompensationModel( grossSales: 5000.0, baseSalary: 300.0, commissionRate: 0.04));
employees[3] = new Employee( firstName: "John", lastName: "Doe", socialSecurityNumber: "666-66-6666", new SalariedCompensationModel( weeklySalary: 1000.0));

for (Employee employee : employees) {
    System.out.println(employee);
    // 输出 earning
    System.out.printf("Earning: $%,.2f\n", employee.compensationModel.earnings());
}

// 动态地改变员工的 CompensationModel, 重新显示他的收入
employees[0].setCompensationModel(new CommissionCompensationModel( grossSales: 10000, commissionRate: 0.1));
employees[1].setCompensationModel(new BasePlusCommissionCompensationModel( grossSales: 5000, commissionRate: 0.05, baseSalary: 300));
employees[2].setCompensationModel(new SalariedCompensationModel( weeklySalary: 1200));
employees[3].setCompensationModel(new CommissionCompensationModel( grossSales: 20000, commissionRate: 0.15));

System.out.println("动态改变员工的 CompensationModel, 重新显示他的收入:");
for (Employee employee : employees) {
    System.out.println(employee);
    // 输出 earning
    System.out.printf("Earning: $%,.2f\n", employee.compensationModel.earnings());
}
}
```

运行: EmployeeTest

初始化员工信息:

Employee{firstName='John', lastName='Smith', socialSecurityNumber='111-11-1111', compensationModel=SalariedCompensationModel{weeklySalary=800.0}}

Earning: \$800.00

Employee{firstName='Lisa', lastName='Barnes', socialSecurityNumber='888-88-8888', compensationModel=CommissionCompensationModel{grossSales=10000.0, commissionRate=0.06}}

Earning: \$600.00

Employee{firstName='Mark', lastName='Jones', socialSecurityNumber='777-77-7777', compensationModel=BasePlusCommissionCompensationModel{grossSales=5000.0, baseSalary=300.0, commissionRate=0.04}}

Earning: \$500.00

Employee{firstName='John', lastName='Doe', socialSecurityNumber='666-66-6666', compensationModel=SalariedCompensationModel{weeklySalary=1000.0}}

Earning: \$1,000.00

动态改变员工的 CompensationModel, 重新显示他的收入:

```

运行: EmployeeTest
初始化员工信息:
Employee{firstName='John', lastName='Smith', socialSecurityNumber='111-11-1111', compensationModel=SalariedCompensationModel{weeklySalary=800.0}}
Earning: $800.00

Employee{firstName='Lisa', lastName='Barnes', socialSecurityNumber='888-88-8888', compensationModel=CommissionCompensationModel{grossSales=10000.0, c
Earning: $600.00

Employee{firstName='Mark', lastName='Jones', socialSecurityNumber='777-77-7777', compensationModel=BasePlusCommissionCompensationModel{grossSales=500
Earning: $500.00

Employee{firstName='John', lastName='Doe', socialSecurityNumber='666-66-6666', compensationModel=SalariedCompensationModel{weeklySalary=1000.0}}
Earning: $1,000.00

动态改变员工的 CompensationModel. 重新显示他的收入:
Employee{firstName='John', lastName='Smith', socialSecurityNumber='111-11-1111', compensationModel=CommissionCompensationModel{grossSales=10000.0, cc
Earning: $1,000.00

Employee{firstName='Lisa', lastName='Barnes', socialSecurityNumber='888-88-8888', compensationModel=BasePlusCommissionCompensationModel{grossSales=500
Earning: $550.00

Employee{firstName='Mark', lastName='Jones', socialSecurityNumber='777-77-7777', compensationModel=SalariedCompensationModel{weeklySalary=1200.0}}
Earning: $1,200.00

Employee{firstName='John', lastName='Doe', socialSecurityNumber='666-66-6666', compensationModel=CommissionCompensationModel{grossSales=20000.0, comm
Earning: $3,000.00

```

(选做题) 题目二

(一) 实验题目

◆ (CarbonFootprint Interface: Polymorphism) Using interfaces, as you learned in this chapter, you can specify similar behaviors for possibly disparate classes. Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon called global warming. Create three small classes unrelated by inheritance classes **Building**, **Car** and **Bicycle**. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes. Write an **interface CarbonFootprint** with a **getCarbonFootprint** method. Have each of your classes implement that interface, so that its **getCarbonFootprint** method calculates an appropriate carbon footprint for that class (check out a few websites that explain how to calculate carbon footprints). Write an application that creates objects of each of the three classes, places references to those objects in **ArrayList<CarbonFootprint>**, then iterates through the **ArrayList**, polymorphically invoking each object's **getCarbonFootprint** method. For each object, print some identifying information and the object's carbon footprint.

(二) 实现过程

思路：

使用 `getCarbonFootprint` 方法编写一个接口 `CarbonFootprint`。让您的每个类实现该接口，以便其 `getCarbonFootprint` 方法为该类计算适当的碳足迹（查看一些解释如何计算碳足迹的网站）。



```
1 package Extra_q2;
2
3 5 个用法 3 个实现
4 public interface CarbonFootprint {
5     1 个用法 3 个实现
6     double getCarbonFootprint();
7 }
8
```

创建三个类 `Building`、`Car` 和 `Bicycle`。为每个类提供一些独特的适当属性和行为，这些属性和行为与其他类没有共同之处。



```
32 public String getName() {
33     return name;
34 }
35
36 0 个用法
37 public void setName(String name) {
38     this.name = name;
39 }
40
41 @Override
42 public String toString()
43 {
44     return String.format("Building's name is %s.\nHeight is %.2f.\nVolume is %.2f.",
45         getName(), getHeight(), getVolume());
46 }
47
48 1 个用法
49 @Override
50 public double getCarbonFootprint()
51 {
52     return getVolume();
53 }
```

```
Employee.java x EmployeeTest.java x Building.java x Car.java x Bicycle.java x Test.java x CarbonFootprint.java x

1 个用法
32 public String getName() {
33     return name;
34 }
35
0 个用法
36 public void setName(String name) {
37     this.name = name;
38 }
39
@Override
40 public String toString()
41 {
42     return String.format("Car's name is %s.%nFuel consumption is %.2f.%nDistance is %.2f.",
43         getName(), getFuelConsumption(), getDistance());
44 }
45
1 个用法
46
@Override
47 public double getCarbonFootprint()
48 {
49     return getFuelConsumption() * getDistance();
50 }
51

Employee.java x EmployeeTest.java x Building.java x Car.java x Bicycle.java x Test.java x CarbonFootprint.java x

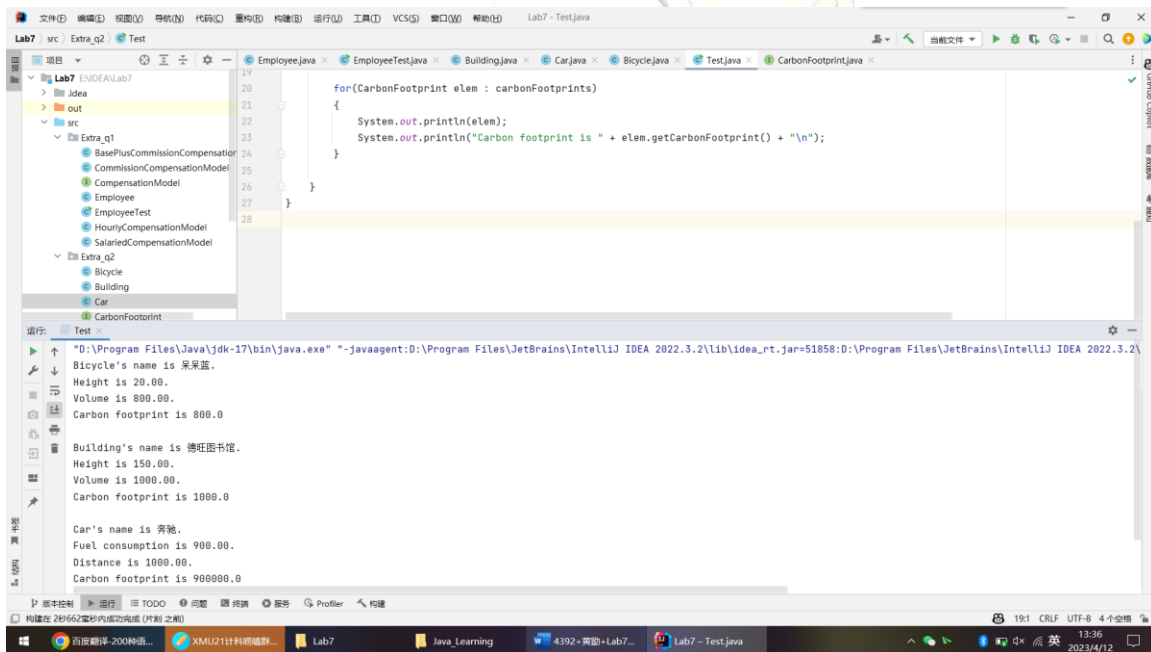
32 public String getName() {
33     return name;
34 }
35
0 个用法
36 public void setName(String name) {
37     this.name = name;
38 }
39
@Override
40 public String toString()
41 {
42     return String.format("Bicycle's name is %s.%nHeight is %.2f.%nVolume is %.2f.",
43         getName(), getHeight(), getVolume());
44 }
45
1 个用法
46
@Override
47 public double getCarbonFootprint()
48 {
49     return getVolume();
50 }
51
52 }
```

编写一个应用程序，创建三个类中每一个类的对象，在 `ArrayList<CarbonFootprint>` 中放置对这些对象的引用，然后遍历 `ArrayList`，以多态方式调用每个对象的 `getCarbonFootprint` 方法。对于每个对象，打印一些识别信息和对象的碳足迹。

```
Employee.java x EmployeeTest.java x Building.java x Car.java x Bicycle.java x Test.java x CarbonFootprint.java x
6 1
7 0 个用法
8 public static void main(String[] args)
9 {
10 // 创建三个类中每一个类的对象，在ArrayList<CarbonFootprint>中放置对这些对象的引用，然后遍历ArrayList，以多态方式调用每个对象的
11 Bicycle bicycle = new Bicycle( name: "呆呆蓝", volume: 800.0, height: 20.0);
12 Building building = new Building( name: "德旺图书馆", volume: 1000.0, height: 150.0);
13 Car car = new Car( name: "奔驰", fuelConsumption: 900.0, distance: 1000.0);
14
15 ArrayList<CarbonFootprint> carbonFootprints = new ArrayList<>();
16
17 carbonFootprints.add(bicycle);
18 carbonFootprints.add(building);
19 carbonFootprints.add(car);
20
21 for(CarbonFootprint elem : carbonFootprints)
22 {
23     System.out.println(elem);
24     System.out.println("Carbon footprint is " + elem.getCarbonFootprint() + "\n");
25 }
26 }
27 }
```

(三) 过程截图

最终结果（全屏截图）



三、实验总结与心得记录

Java 中的继承和多态是面向对象编程的重要概念，对于 Java 程序员来说是必须掌握的基础知识。继承是一种创建新类的方式，让已有类的属性和方法在新的类中得以重复使

用。通过继承，可以实现代码的复用性和可维护性。Java 中的继承是单继承的，即每个类只能直接继承自一个父类。

在继承的基础上，Java 还提供了多态机制。多态是指同一种行为（方法）在不同的对象上有不同的表现形式。在 Java 中，多态可以通过方法重载和方法重写来实现。方法重载是指在同一个类中定义多个方法，方法名相同但参数不同，编译器会根据不同的参数类型和个数选择合适的方法。方法重写是指子类重写父类中的方法，使得子类可以使用自己的实现方式。

继承和多态是 Java 面向对象编程的内核概念，理解和掌握这些概念对于 Java 程序员来说非常重要。在实际编程中，应该尽可能地使用继承和多态，以提高代码的复用性和可维护性。总之，这个实验对我的 Java 编程技能和面向对象编程能力的提升非常有帮助。通过这个实验，我深入了解了 Java 的更多知识。这些技能和知识将在我的未来的 Java 开发中起到重要的作用。

Java 中的异常处理机制可以帮助我们有效地诊断和调试程序，以及提高程序的健壮性。在 Java 中，异常处理机制主要由以下几个方面组成：

1. 异常类：Java 中提供了多个异常类，包括 `RuntimeException`、`IOException`、`SQLException` 等。程序员也可以自己定义异常类。
2. 异常处理语句：Java 中提供了 `try-catch` 和 `try-catch-finally` 语句用于捕获和处理异常。`try` 块中的代码可能会抛出异常，`catch` 块中的代码用于处理异常。
3. 抛出异常：在 Java 中，程序员可以使用 `throw` 关键字抛出异常。如果一个方法可能抛出异常，就需要在方法签名中声明 `throws` 子句。
4. 异常链：Java 中的异常可以形成链式结构。一个异常对象可能包含另一个异常对象，这样就可以更好地追踪异常的来源和原因。

在实际编程中，我们应该注意以下几点：

1. 对于可预见的异常情况，我们应该尽可能地通过 try-catch 语句来处理异常，以保证程序的正常运行。
2. 在处理异常时，应该注意将异常信息打印出来，以便诊断问题。
3. 对于不可预见的异常情况，应该在程序中使用 assert 语句或编写单元测试，以保证程序的健壮性。
4. 在自定义异常类时，应该遵循 Java 的命名规范，并且要确保异常类具有清晰的语义。

综上所述，Java 中的异常处理机制是 Java 程序设计中非常重要的一部分。我们应该熟练掌握 Java 中的异常类和异常处理语句，并且在实际编程中注重异常处理，以保证程序的正常运行和健壮性。

