华东师范大学数据科学与工程学院实验报告

课程名称：计算机网络与编程	年级：21级	上机实践成绩：
指导教师：张召	姓名：包亦晟	学号：10215501451
上机实践名称：	上机实践日期：2023.3.17	
上机实践编号：	组号：1-451	上机实践时间：2023.3.17

一、 实验目的

- 熟悉掌握IntelliJ IDEA的使用
- 学习并掌握Java面向对象部分基础知识，为使用Java进行网络编程打下基础

二、 实验任务

熟悉继承、多态、接口、抽象类、异常处理以及枚举等相关知识

三、 使用环境

- IntelliJ IDEA
- JDK 版本: Java 19

四、 实验过程

task1

设计一个名为StopWatch（秒表）的类，该类继承Watch（表）类

```
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Scanner;

class Watch {
    protected long startTime, endTime;
    public void start(long startTime) {
        this.startTime = startTime;
    }

    public void end(long endTime) {
        this.endTime = endTime;
    }
}

class StopWatch extends Watch {
    public long getElapsedTime() {
        return endTime - startTime;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        StopWatch t = new StopWatch();
        long st = 100000;
        long ed = 900000;
        t.start(st);
        t.end(ed);
        System.out.println(t.getElapsedTime());
    }
}
```

我发现Java和C++中都含有this。只不过Java中的this是对当前对象的引用，而C++中的this是对指向当前对象的指针。

bonus task1

回顾C++的继承方式，其有public继承、 protected继承和private继承，后两种常用作空基类优化等技巧，然而Java只有一种继承方式extends，这是为什么?

在C++中，继承方式包括public、protected和private，这些方式定义了派生类从基类中继承成员的可访问性。这种可访问性的控制使得C++的类继承关系更加灵活，可以更加细致地控制派生类对基类成员的访问。

而Java的继承方式只有extends，它是一种公有继承方式，子类可以访问父类的public和protected成员，但不能访问父类的private成员。这样的设计使得Java的类继承关系更加简洁和直观。Java的设计者认为，继承关系应该尽可能简单明了，减少语言的复杂度，以提高代码的可读性和可维护性。而且，即使没有private继承和protected继承，**它们的功能也可以通过封装实现**。而所带来这些便利，在java设计者心中，**是不能抵消它们所带来的复杂性的提高的**。因此，Java只提供了一种继承方式。

task2

对于提供的Fish类， implements Comparable接口。初始化10个Fish对象放入数组或容器，并使用按照size属性从小到大排序，排序后从前往后对每个对象调用print()进行打印

```
class Fish implements Animal, Comparable<Fish>{
    public void eat() {
        System.out.println("Fish eats");
    }

    public void travel() {
        System.out.println("Fish travels");
    }

    public void move() {
        System.out.println("Fish moves");
    }

    int size;
    public Fish(){
        Random r = new Random();
        this.size = r.nextInt(100);
    }
    void print(){
        System.out.print(this.size + " < ");
    }

    public int compareTo(Fish o) {
```

```
        return this.size - o.size;
    }
}

public class Main {
    public static void main(String[] args) {
        Fish[] a = new Fish[10];
        for (int i = 0; i < 10; i++) {
            a[i] = new Fish();
        }
        Arrays.sort(a);
        for (int i = 0; i < 10; i++) {
            a[i].print();
        }
    }
}
```

```
1 < 7 < 25 < 29 < 32 < 37 < 37 < 41 < 59 < 91 <
```

这里我一开始碰到的困难是，没有理解清楚Comparable这个接口到底是干什么的。Comparable是用来定义对象的比较逻辑的。而在Arrays类的sort方法中，是会强制使用这个接口的，所以不需要我们将其作为参数放入sort中。这点和C++不一样。在使用C++的时候，比较方法是要传入sort的，要么是自己写一个函数，要么就是用lambda。

task3

根据要求创建SalesEmployee、 HourlyEmployee、 SalariedEmployee三个类的对象各一个，并计算某个月这三个对象员工的工资

```
public abstract class Employee {
    protected String name;
    protected int birthMonth;
    protected double salary;

    public Employee() {
    }

    public Employee(String name, int birthMonth) {
        this.name = name;
        this.birthMonth = birthMonth;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public double getSalary() {
        return salary;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public int getBirthMonth() {
        return birthMonth;
    }

    public void setBirthMonth(int birthMonth) {
        this.birthMonth = birthMonth;
    }
    abstract public double computePay(int month);
}
```

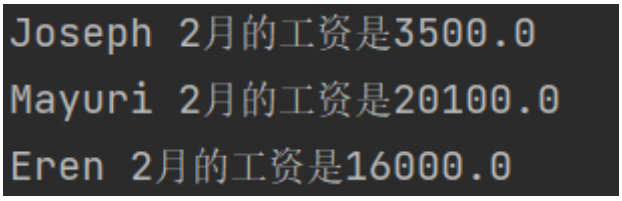
```
public class SalesEmployee extends Employee{
    private double monthSales;
    private double commission;
    private double basicSalary;

    public SalesEmployee() {
    }

    public SalesEmployee(String name, int birthMonth, double monthSales, double
commission, double basicSalary) {
        super(name, birthMonth);
        this.monthSales = monthSales;
        this.commission = commission;
        this.basicSalary = basicSalary;
    }

    public SalesEmployee(double monthlySales, double commission, double
basicSalary) {
        this.monthSales = monthSales;
        this.commission = commission;
        this.basicSalary = basicSalary;
    }

    public double getMonthlySales() {
        return monthSales;
    }

    public void setMonthlySales(double monthlySales) {
        this.monthSales = monthlySales;
    }

    public double getCommission() {
        return commission;
    }

    public void setCommission(double commission) {
        this.commission = commission;
    }

    public double getBasicSalary() {
        return basicSalary;
    }

    public void setBasicSalary(double monthlySales) {
        this.basicSalary = basicSalary;
    }
```

```
    public double computePay(int month) {
        return this.salary = getMonthlySales() * getCommission() + basicSalary;
    }
}
```

```
public class SalariedEmployee extends Employee{
    public SalariedEmployee() {
    }

    public SalariedEmployee(String name, int birthMonth, double salary) {
        super(name, birthMonth);
        this.salary = salary;
    }

    @Override
    public double getSalary() {
        return salary;
    }

    @Override
    public void setSalary(double salary) {
        this.salary = salary;
    }

    public double computePay(int month) {
        if (month == birthMonth) this.salary = 100 + salary;
        return salary;
    }
}
```

```
public class HourlyEmployee extends Employee{
    private int workHour;
    private int hourSalary;

    public HourlyEmployee() {
    }

    public HourlyEmployee(String name, int birthMonth, int workHour, int
hourSalary) {
        super(name, birthMonth);
        this.workHour = workHour;
        this.hourSalary = hourSalary;
    }

    public HourlyEmployee(int workHour, int hourSalary) {
        this.workHour = workHour;
        this.hourSalary = hourSalary;
    }

    public HourlyEmployee(String name, int birthMonth) {
        super(name, birthMonth);
    }

    public int getWorkHour() {
        return workHour;
    }
```

```
    public void setWorkHour(int workHour) {
        this.workHour = workHour;
    }

    public int getHourMoney() {
        return hourSalary;
    }

    public void setHourMoney(int hourMoney) {
        hourSalary = hourMoney;
    }

    public double computePay(int month) {
        return salary = getHourMoney() * getWorkHour();
    }
}
```



在设计类的时候我运用了封装，将每个类的状态信息隐藏在类的内部，外部无法直接访问。只能通过我所提供的方法，来对隐藏的状态信息进行访问。

在抽象类这一点上，感觉java和C++是很像的。只不过在C++中，拥有纯虚函数的类被称作抽象类，而在java中包含抽象方法的类就是抽象类。感觉只不过是名字上不太一样。

task4

请在实验报告中列举出Error和Exception的区别

- 相同点：两者全都继承于Throwable类
- 不同点：
  - Error：程序发生了严重的问题，程序应该直接崩溃而不是尝试处理错误
  - Exception：程序执行过程中可以预料的意外情况，应该尝试捕捉处理错误
    - checked：会被编译器强制检查
    - unchecked：程序运行时出现的异常，需要程序员自行处理

task5

请设计可能会发生的5个RuntimeException案例并将其捕获，将捕获成功的运行时截图和代码附在实验报告中

1.NullPointerException

空指针异常：调用了未经初始化的对象或者是不存在的对象。

```
public class Main {
    public static void main(String[] args) {
        try {
            Object obj = null;
            obj.toString();
        } catch (NullPointerException e) {
            System.out.println("Caught NullPointerException: " +
e.getMessage());
        }

    }
}
```

```
Caught NullPointerException: Cannot invoke "Object.toString()" because "obj" is null
```

2.ArrayIndexOutOfBoundsException

数组角标越界异常，常见于操作数组对象时发生

```
public class Main {
    public static void main(String[] args) {
        try {
            int[] arr = new int[2];
            int num = arr[5];
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught ArrayIndexOutOfBoundsException: " +
e.getMessage());
        }

    }
}
```

```
Caught ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 2
```

3.ArithmeticException

数学运算异常

```
public class Main {
    public static void main(String[] args) {
        try {
            int num = 4396 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        }
    }
}
```

```
Caught ArithmeticException: / by zero
```

4.ClassCastException

数据类型转换异常

```
public class Main {
    public static void main(String[] args) {
        try {
            Object obj = new Object[10];
            String str = (String) obj;
        } catch (ClassCastException e) {
            System.out.println("Caught ClassCastException: " + e.getMessage());
        }
    }
}
```

```
Caught ClassCastException: class [Ljava.lang.Object; cannot be cast to class java.lang.String ([Ljava.lang.Object; and java.lang.String are in mod
```

5.IllegalArgumentException

数据类型转换异常

在这个例子中，我尝试使用了一下throw

```
public static void main(String[] args) {
            try {
                int weight = -65;
                if (weight < 0) {
                    throw new IllegalArgumentException("You are way too slim");
                }
            } catch (IllegalArgumentException e) {
                System.out.println("Caught IllegalArgumentException: " +
e.getMessage());
            }
        }
```

```
Caught IllegalArgumentException: You are way too slim
```

task6

对于list中的每个Color枚举类，输出其type（不用换行），请使用swtich语句实现，请将代码和运行截图附在实验报告中

```
enum Color{
    RED(1),
    GREEN(2),
    BLUE(3);
    int type;
    Color(int _type){
        this.type = _type;
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Color> list = new ArrayList<>();
        for(int i=1;i<=3;i++){
            Collections.addAll(list, Color.values());
        }
        Random r = new Random(1234567);
        Collections.shuffle(list, r);
        for(int i=0;i<list.size();i++){
```

```
            Color c = list.get(i);
            switch (c) {
                case RED : System.out.print(Color.RED.type + " ");break;
                case GREEN : System.out.println(Color.GREEN.type + " ");break;
                case BLUE : System.out.println(Color.BLUE.type + " ");break;
            }
        }
    }
}
```

```
2 2 1 2 3 3 1 3 1
```

C++和Java中都有enum关键词，两者虽然非常相像，但仍然存在差距。在C++中，枚举就是命名过后的int型常量，可以隐式地把枚举值转化成对应的值。而对于java，其实枚举更类似于类的实例，而且必须显式地转换枚举值与对应的值。就如果我上面的代码一样，switch语句中case后面跟的必须是枚举值，而不是对应的值。

五、 总结

在本次实验课中我学习了Java面向对象的部分基础知识。说来神奇，上一次实验中我感受到的是Java与C++之间的差异。而在本次实验中我感受到的更多的却是Java和C++之间相似的地方。

希望自己在将来学习java的过程中，能够时刻认识到C++与Java的异同并最终能够融会贯通。