

Lab3: Java编程基本语法和基础2

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

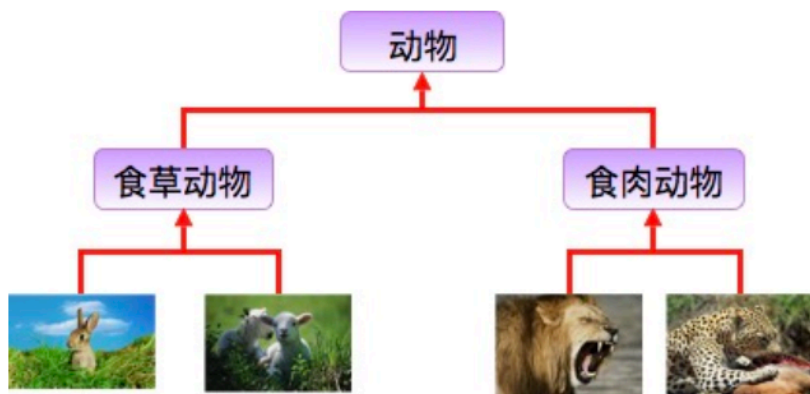
- 熟悉继承、多态、接口、抽象类、异常处理以及枚举等相关知识

三、使用环境

- IntelliJ IDEA
- JDK 版本: Java 19

四、实验过程

1. 继承



继承是指子类继承父类的特征和方法，使得子类对象（实例）具有父类的实例域和方法。

在Java中可以通过extends关键字申明类的继承关系，一般形式如下：

```
class ParentClass {  
  
}  
  
class ChildClass extends ParentClass {  
  
}
```

通过extends关键字继承实现两种动物类，其中动物分别为企鹅和老鼠，要求如下：

- 企鹅：属性（姓名、id），方法（吃、自我介绍）

- 老鼠：属性（姓名、id），方法（吃、自我介绍）

```
public class Animal {
    public String name;
    public int id;

    // 构造函数
    public Animal(String myName, int myId) {
        this.name = myName;
        this.id = myId;
    }

    public void eat() {
        System.out.println(name + "正在吃");
    }

    public void introduce() {
        System.out.printf("大家好! 我是%d号, %s.\n", id, name);
    }

    public static void main(String[] args) {
        Penguin peng = new Penguin("p1", 1);
        Mouse mouse = new Mouse("m1", 2);
        peng.introduce();
        mouse.introduce();
    }
}

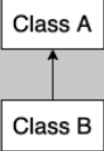
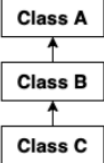
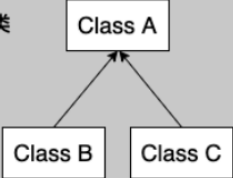
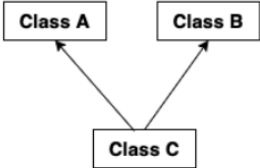
class Penguin extends Animal {
    public Penguin(String myName, int myId) {
        super(myName, myId);
    }
}

class Mouse extends Animal {
    public Mouse(String myName, int myId) {
        super(myName, myId); // 通过super关键字实现对父类的访问
    }
}
```

- super: super关键字用来引用当前对象的父类，实现对父类成员/方法的访问
- this: 指向当前类自己的引用
- Java访问控制符的含义:

	类内部	本包	子类	外部包
public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
private	√	√	×	×

- Java不支持多继承，但支持多重继承，如下图所示

<div>单继承</div>  <pre> public class A { } public class B extends A { } </pre>
<div>多重继承</div>  <pre> public class A {.....} public class B extends A {.....} public class C extends B {.....} </pre>
<div>不同类继承同一个类</div>  <pre> public class A {.....} public class B extends A {.....} public class C extends A {.....} </pre>
<div>多继承 (不支持)</div>  <pre> public class A {.....} public class B {.....} public class C extends A, B { } // Java 不支持多继承 </pre>

继承特性

- 子类将拥有父类中（符合对应访问控制的）属性、方法
- 子类可以拥有自己的属性和方法，即子类可以对父类进行扩展
- 子类可以用自己的方式实现父类的方法

继承类型

- task1:** 设计一个名为StopWatch（秒表）的类，该类继承Watch（表）类：

Watch类包含：私有数据域startTime和endTime，并包含对应访问方法，一个名为 `start()` 的方法，将startTime重设为当前时间，一个名为 `stop()` 的方法，将endTime设置为当前时间；
StopWatch类另包含：一个名为 `getElapsedTime()` 的方法，以毫秒为单位返回秒表记录的流逝时间；
在Main函数测试StopWatch类功能。

- **bonus task1 (optional):** 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？

2. 多态

多态可以简单理解为同一个接口，使用不同的实例或者参数列表表现出不同的行为

重写 (Override)

重写是子类对父类的允许访问的方法的实现过程进行重新编写，返回值和形参都不能改变

```
public class Animal {
    public void move() {
        System.out.println("动物可以移动");
    }

    public static void main(String[] args) {
        Animal a = new Animal();
        Animal b = new Dog();
        a.move(); // 执行 Animal 类的方法
        b.move(); // 执行 Dog 类的方法
    }
}

class Dog extends Animal {
    public void move() {
        System.out.println("狗可以跑和走");
    }
}
```

方法的重写规则包括：

- 参数列表与被重写方法的参数列表必须完全相同
- 访问权限不能比父类中被重写的方法的访问权限更低
- 父类的成员方法只能被它的子类重写
- 声明为final的方法不能被重写，声明为static的方法不能被重写
- 构造方法不能被重写

重载 (Overload)

重载 (Overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。每个重载的方法（或构造函数）都必须有一个独一无二的参数类型列表。

```
public class Animal {
    public void move() {
        System.out.println("动物可以移动");
    }
}
```

```

public void move(int _num) {
    System.out.printf("动物向前移动 %d 步", _num);
}

public static void main(String[] args) {
    Animal a = new Animal();
    a.move(10);
}
}

```

附：方法重写是子类与父类间一种多态性表现，方法重载是一个类的多态性表现

3. 接口

接口（Interface）是抽象方法的集合，通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。

- 接口不是类，类描述对象的属性和方法，接口则包含类要实现的方法
- 一个实现接口的类，必须实现接口内所描述的所有方法，接口中所有的方法必须是抽象方法
- 可以用接口类型引用实现接口的对象，接口引用可见范围仅限被实现的函数

下面是接口声明的一个简单例子

```

public interface Animal {
    void eat();
    void travel();
}

```

类使用implements关键字实现接口。在类声明中， Implements关键字放在class声明后面

```

public class Fish implements Animal{
    public void eat(){
        System.out.println("Fish eats");
    }
    public void travel(){
        System.out.println("Fish travels");
    }
    public void move(){
        System.out.println("Fish moves");
    }
    public static void main(String args[]){
        Fish m = new Fish();
        m.eat();
        m.travel();
        m.move();
    }

    //    for task2

```

```

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

```

- 通过同时实现多个接口，可以变相地使Java的类具有多继承的特性（接口跟接口之间采用逗号分隔）

```

interface A {
    void eat();
    void sleep();
}
interface B {
    void show();
}
public class C implements A, B {
}

```

- **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

Comparable是排序接口。若一个类实现了Comparable接口，就意味着该类支持排序。实现了Comparable接口的类的对象的列表或数组可以通过 Collections.sort() 或 Arrays.sort() 进行自动排序。

```

public interface Comparable<T> {
    public int compareTo(T o);
}

```

4. 抽象类

包含了抽象方法的类，即为抽象类。抽象方法只有方法声明，没有具体实现

```

abstract void fun();

```

抽象方法必须用abstract关键字进行修饰，如果一个类含有抽象方法，则称这个类为抽象类，抽象类必须在类前用abstract关键字修饰

在Java语言中使用abstract class来定义抽象类。如下示例：

```

public abstract class Employee
{
    private String name;
}

```

```

private String address;
private int number;
public Employee(String name, String address, int number)
{
    System.out.println("Constructing an Employee");
    this.name = name;
    this.address = address;
    this.number = number;
}
public String getName()
{
    return name;
}
public String getAddress()
{
    return address;
}
abstract public double computePay();
}

```

注意：抽象类不能实例化对象，抽象类必须被继承并实现方法，才能被使用

- **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：

某公司的雇员分为以下若干类：

Employee：所有员工类的父类

属性：员工的姓名，员工的生日月份。

方法：getSalary(int month)，根据月份来确定工资。

SalesEmployee：Employee的子类，销售人员。工资 = 月销售额*提成率 + 基本月薪

属性：月销售额、提成率

SalariedEmployee：Employee的子类，拿固定工资的员工。如果该月员工过生日，则公司会额外奖励100元。

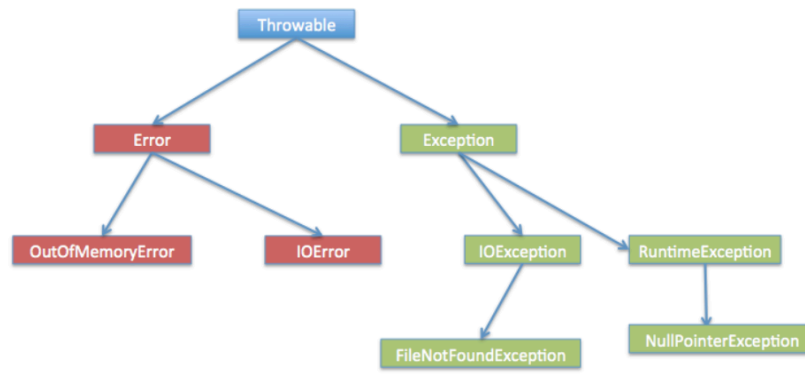
属性：月薪

HourlyEmployee：Employee的子类，拿小时工资的员工。

属性：每小时工资、每月工作的小时数

5. 异常处理

Exception（异常）是应用程序中可能的可预测、可恢复问题。异常一般是在特定环境下产生的，通常出现在代码的特定方法和操作中。所有的异常类是从java.lang.Exception类继承的子类，Exception类又是Throwable类的子类。



捕获异常

```
try
{
    // code
} catch (ExceptionName e)
{
    // logics after catch
}
```

下面的例子中声明有两个元素的一个数组，当代码试图访问数组的第四个元素的时候就会抛出一个异常。

```
public class ExceptionTest {
    public static void main(String args[]) {
        try {
            int a[] = new int[2];
            System.out.println("Access third element :" + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown :" + e);
        }
        System.out.println("Out of the block");
    }
}
```

throws/throw关键字： 定义一个方法的时候可以使用throws关键字声明。使用throws关键字声明的方法表示此方法不处理异常，而交给外部（方法调用处）进行处理

```
public class ClassName
{
    public void deposit(double amount) throws RemoteException
    {
        // Method implementation
        throw new RemoteException();
    }
    //Remainder of class definition
}
```


finally关键字： finally关键字用来创建在try代码块后面执行的代码块。无论是否发生异常，finally代码块中的代码总会被执行。在finally代码块中，可以运行清理类型等收尾善后性质的语句。

```
try {
    // code
} catch (Exception1 e1){
    // code
} catch (Exception2 e2){
    // code
} finally {
    // code
}
```

- **task4:** 请在实验报告中列举出Error和Exception的区别
- **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

6. 枚举

Java枚举是一个特殊的类，表示用户定义的一组常量。Java枚举类使用enum关键字来定义，各个常量使用逗号来分割。

```
enum Color{
    RED,
    GREEN,
    BLUE
}

public class Test {
    public static void main(String[] args) {
        Color c1 = Color.RED;
        System.out.println(c1);
    }
}
```

- **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

```
public class Test {
    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);  
//        TODO  
//        switch () {  
//  
//        }  
    }  
}  
  
enum Color {  
    RED(1),  
    GREEN(2),  
    BLUE(3);  
  
    int type;  
  
    Color(int _type) {  
        this.type = _type;  
    }  
}
```