接口

1. 接口概念

> **接口 相当于一种约定 接口只关注约定本身 不关注具体实现**

> 1.接口中所有的方法默认为全局抽象方法 即使用public abstract修饰
>
> 2.接口不能直接对象 必须通过new实现类(子类)的方式创建对象 多态向上转型
>
> 3.实现类(子类)必须实现(重写)接口中所有的抽象方法 除非子类也是抽象类 或者 子类也是接口
>
> 4.接口中不能书写普通属性(默认为全局静态常量) 普通方法(JDK8开始可以使用default编写普通方法) 构造方法 静态方法
>
> 5.接口实现多态的方式与之前一致
>
> 6.一个类只能继承一个父类 但是可以实现多个接口
>
> 7.一个接口 可以 继承 多个接口

```
package com.atguigu.test4;

/**
 *  打印机类
 *  墨盒 :  彩色 黑色
 *  纸张 :  A4 B5
 */
public class Printer {
    private InkBox inkBox;
    private Paper paper;

    public void setInkBox(InkBox inkBox){
        this.inkBox = inkBox;
    }

    public InkBox getInkBox(){
        return inkBox;
    }

    public void setPaper(Paper paper){
        this.paper = paper;
    }
    public Paper getPaper(){
        return paper;
    }

    public void print(){
```

```
        System.out.println("使用" + this.getInkBox().getInkBoxType() + "在 " +
this.getPaper().getPaperType() + "纸张上打印");
    }









}
```

```
package com.atguigu.test4;

public interface InkBox {
    String getInkBoxType();
}
```

```
package com.atguigu.test4;

public class BlackInkBox implements  InkBox{
    @Override
    public String getInkBoxType() {
        return "黑色墨盒";
    }
}
```

```
package com.atguigu.test4;

public class ColorInkBox implements  InkBox {
    @Override
    public String getInkBoxType() {
        return "彩色墨盒";
    }
}
```

```
package com.atguigu.test4;

public interface Paper {
    String getPaperType();
}
```

```java
package com.atguigu.test4;

public class A4 implements  Paper{

    @Override
    public String getPaperType() {
        return "A4";
    }
}
```

```java
package com.atguigu.test4;

public class B5 implements  Paper{
    @Override
    public String getPaperType() {
        return "B5";
    }
}
```

```java
package com.atguigu.test4;



/**
 *  当你关注事物的本质 使用抽象类
 *  当你关注某个功能 使用接口
 */
public class Test {
    public static void main(String[] args) {
        Printer hp = new Printer();

        InkBox black = new BlackInkBox();
        InkBox color = new ColorInkBox();

        hp.setInkBox(black);

        Paper a4 = new A4();
        Paper b5 = new B5();

        hp.setPaper(a4);

        hp.print();

        InkBox [] inkBoxes = new InkBox[2];
        inkBoxes[0] = new BlackInkBox();
        inkBoxes[1] = new ColorInkBox();


    }
}
```

2. 接口和抽象类

> **当你关注事物的本质 使用抽象类**
>
> **当你关注某个功能 使用接口**

> 相同点
>
> 代表系统的抽象层 都不能被实例化 都能包含抽象方法 用于描述系统提供的服务，不必提供具体实现
>
> 不同点
>
> 在抽象类中可以为部分方法提供默认实现，而接口中只能包含抽象方法 抽象类便于复用，接口便于代码维护 一个类只能继承一个直接的父类，但可以实现多个接口

> 使用原则
>
> 接口做系统与外界交互的窗口 接口提供服务 接口本身一旦制定，就不允许随意修改 抽象类可完成部分功能实现，还有部分功能可作为系统的扩展点

> 多用组合，少用继承 针对接口编程 针对扩展开放，针对改变关闭

异常

1. try-catch处理异常

1.1 情况1

> 使用try-catch处理异常情况1： 正常捕获到异常
>
> try代码块中保存可能出现异常的代码
>
> catch用于捕获异常
>
> try 或者 catch 均不能单独出现 try必须结合catch 或者 catch-finally 或者 finally

```
package com.atguigu.test5;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 *  使用try-catch处理异常情况1 ： 正常捕获到异常
 *  try代码块中保存可能出现异常的代码
 *  catch用于捕获异常
 *
 *  try 或者 catch 均不能单独出现 try必须结合catch  或者 catch-finally  或者 finally
 */
public class TestTryCatch1 {
```

```
    public static void main(String[] args) {
        try{
            Scanner in = new Scanner(System.in);
            System.out.print("请输入被除数:");
            int num1 = in.nextInt();
            System.out.print("请输入除数:");
            int num2 = in.nextInt();
            System.out.println(num1+"/"+ num2 +"="+ num1/ num2);
        }catch(InputMismatchException e){
            e.printStackTrace();
        }

        System.out.println("感谢使用本程序! ");
    }
}
```

1.2 情况2

> 使用try-catch处理异常情况2： 使用多个catch块捕获多种异常
>
> 使用多个catch块捕获多种异常 注意先写子类 然后再写父类 不推荐只写一个父类的方式处理异常

```
package com.atguigu.test5;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 *  使用try-catch处理异常情况2 ： 使用多个catch块捕获多种异常
 *  使用多个catch块捕获多种异常  注意先写子类  然后再写父类  不推荐只写一个父类的方式处理异常
 */
public class TestTryCatch2 {
    public static void main(String[] args) {
        try{
            Scanner in = new Scanner(System.in);
            System.out.print("请输入被除数:");
            int num1 = in.nextInt();
            System.out.print("请输入除数:");
            int num2 = in.nextInt();
            System.out.println(num1+"/"+ num2 +"="+ num1/ num2);
        }catch(InputMismatchException e){
            e.printStackTrace();
        }catch(ArithmeticException e){
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }

        System.out.println("感谢使用本程序! ");
    }
```

```
}
```

2. finally

> finally： 表示不管是否出现异常 以及异常是否被捕获到 都将执行的代码
>
> finally 不能单独出现 必须结合 try 或者 try-catch
>
> finally不执行的唯一情况： 在执行之前退出JVM虚拟机
>
> System.exit(int status) : 0表示正常退出 非0表示非正常退出

```
package com.atguigu.test6;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 *  finally ： 表示不管是否出现异常 以及异常是否被捕获到 都将执行的代码
 *  finally 不能单独出现 必须结合 try 或者 try-catch
 *  finally不执行的唯一情况 ： 在执行之前退出JVM虚拟机
 *
 *  System.exit(int status) : 0表示正常退出 非0表示非正常退出
 *
 */
public class TestFinally {
    public static void main(String[] args) {
        try{
            Scanner in = new Scanner(System.in);
            System.out.print("请输入被除数:");
            int num1 = in.nextInt();
            System.out.print("请输入除数:");
            int num2 = in.nextInt();
            System.out.println(num1+"/"+ num2 +"="+ num1/ num2);

//            System.exit(12);

        }catch(InputMismatchException e){
            e.printStackTrace();
        }finally{
            System.out.println("感谢使用本程序! ");
        }


    }
}
```

> finally面试题 : 注意区分返回值为基本数据类型 还是引用数据类型

> 回顾 值传递 和 引用传递

```
package com.atguigu.test5;


import java.util.Arrays;

/**
 *  finally面试题 ： 注意区分返回值为基本数据类型 还是引用数据类型
 *  回顾 值传递 和 引用传递
 */
public class TestFinallyInterview {
    public static int getNum(){
        int num = 10;
        try{
            num++;
            return num;
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            num++;
        }
        return num;
    }

    public static int [] getNums(){
        int []nums = {1,2,3,4,5};
        try{
            return nums;
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            for (int i = 0; i < nums.length; i++) {
                nums[i]++;
            }

        }
        return nums;
    }


    public static void main(String[] args) {
        System.out.println(getNum());

        int[] nums = getNums();

        System.out.println(Arrays.toString(nums));
    }



}
```

3. throw和throws

> throws 表示在方法声明的位置 声明异常
>
> 根据声明异常类型的不同 调用者需要做出不同的处理
>
> 运行时异常：RuntimeException 及其子类属于运行时异常 调用者不必处理
>
> 检查异常(CheckedException)：除了RuntimeException之外的其他异常 都属于检查异常
>
> 调用者必须处理 可以继续往后声明给JVM虚拟机 也可以 使用try-catch处理
>
> throw 表示在方法体内抛出异常
>
> 针对抛出的异常 需要做出不同的处理
>
> 运行时异常 直接抛出 无需任何处理
>
> 检查异常 抛出以后 必须声明

```
    package com.atguigu.test6;

import java.io.IOException;

/**
 *  throws 表示在方法声明的位置 声明异常
 *  根据声明异常类型的不同 调用者需要做出不同的处理
 *  运行时异常：RuntimeException 及其子类属于运行时异常 调用者不必处理
 *  检查异常(CheckedException)：除了RuntimeException之外的其他异常 都属于检查异常
 *  调用者必须处理 可以继续往后声明给JVM虚拟机 也可以 使用try-catch处理
 *
 *  throw 表示在方法体内抛出异常
 *  针对抛出的异常 需要做出不同的处理
 *  运行时异常 直接抛出 无需任何处理
 *  检查异常 抛出以后 必须声明
 *
 */
public class TestThrowAndThrows {
    public static void m1() throws NullPointerException{

    }

    public static void m2() throws NullPointerException,ArithmeticException,ClassCastException{

    }

    public static void m3() throws ClassNotFoundException{

    }

    public static void m4() throws IOException{

    }

    public static void inputAge(int age){
```

```
        // ……
        if(age < 0 || age > 130){
           throw new RuntimeException("年龄不合法，必须在1 ~ 130 之间");
        }
        // ……
    }

    public static void m5() throws IOException {
        throw new IOException();
    }




    public static void main(String[] args) throws ClassNotFoundException, IOException {
        m1();
        m2();

        m3();

        try {
            m4();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        inputAge(155);


        m5();
    }


}
```

4. 自定义异常

> 自定义异常步骤
>
> 1.继承 Throwable 或者 Exception 或者 RuntimeException 任意其中一个
>
> 继承Throwable 或者 Exception 表示 检查异常 继承RuntimeException 表示为运行时异常
>
> 2.调用父类的有参构造方法完成异常信息初始化

```
package com.atguigu.test6;

/**
 * @author WHD
 * @description TODO
 * @date 2023/6/9 16:06
```

```
 */
public class Person {
    private int age;

    public int getAge() {
        return age;
    }

    public void setAge(int age)  {
        if(age > 0 && age < 130){
            this.age = age;
        }
        throw new InputAgeOutOfBoundsException("" + age);
    }

    public static void main(String[] args) {
        Person p1 = new Person();
        p1.setAge(200);
    }

}
```

```
package com.atguigu.test6;

/**
 *  自定义异常步骤
 *  1.继承 Throwable 或者 Exception  或者 RuntimeException 任意其中一个
 *  继承Throwable 或者  Exception 表示 检查异常 继承RuntimeException 表示为运行时异常
 *  2.调用父类的有参构造方法完成异常信息初始化
 */
public class InputAgeOutOfBoundsException extends RuntimeException{
    public InputAgeOutOfBoundsException(String message){
        super(message);
    }
}
```