《Java 程序设计》实验报告七

学生姓名：李季鸿　　班级：2021 级计本（3）班　　学号：20213002624

实验地点：9 教 404　　指导教师：张春元

实验日期：2023-04-17　　共 2 学时

实验环境：Win10+JDK1.8+IntelliJ IDEA 2022.1.1

1. 实验目的

学习异常类、常用类，进一步巩固面向对象编程思想。

2. 实验内容

（1）用集成化开发工具完成实验教材 P63 实验 2 内容。

（2）用集成化开发工具完成实验教材 P68 实验 1 内容。

（3）用集成化开发工具完成实验教材 P70 实验 2 内容。

3. 实验过程

报告撰写具体要求：截屏显示或直接写出实验 1 至实验 3 的源码和运行结果，并完成每个实验的实验后练习。

实验内容（1）：

```
1. package _7_.shiyan1;
2.
3. /**
4.  * \* Created with IntelliJ IDEA.
5.  * \* User: lijihong
6.  * \* Date: 2023-04-17
7.  * \* Time: 16:29
8.  * \
9.  */
10.public class Goods {
11.    boolean isDanger;
12.    String name;
13.
14.    public void setIsDanger(boolean boo) {
15.        isDanger = boo;
16.    }
17.
18.    public boolean isDanger() {
19.        return isDanger;
20.    }
21.
22.    public void setName(String s) {
23.        name = s;
```

```
    }

    public String getName() {
        return name;
    }
}
package _7_.shiyan1;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-17
 * \* Time: 16:30
 * \
 */
public class DangerException extends Exception {
    String message;

    public DangerException() {
        message = "危险品";
    }

    public void toShow() {
        System.out.print(message + " ");
    }
}
package _7_.shiyan1;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-17
 * \* Time: 16:30
 * \
 */
public class Machine {
    public void checkBag(Goods goods) throws DangerException {
        if (goods.isDanger()) {
            DangerException danger = new DangerException();
            throw danger;//抛出 danger
        } else {
            System.out.print(goods.getName() + "不是危险品！ ");
        }
```

```
    }
}
package _7_.shiyan1;

/**
 * \* Created with IntelliJ IDEA.
 * \* User: lijihong
 * \* Date: 2023-04-17
 * \* Time: 16:31
 * \
 */
public class Check {
    public static void main(String[] args) {
        Machine machine = new Machine();
        String name[] = {"苹果", "炸药", "西服", "硫酸", "手表", "硫磺"};
        Goods[] goods = new Goods[name.length];
        for (int i = 0; i < name.length; i++) {
            goods[i] = new Goods();
            if (i % 2 == 0) {
                goods[i].setIsDanger(false);
                goods[i].setName(name[i]);
            } else {
                goods[i].setIsDanger(true);
                goods[i].setName(name[i]);
            }
        }
        for (int i = 0; i < goods.length; i++) {
            try {
                machine.checkBag(goods[i]);
                System.out.println(goods[i].getName() + "检查通过");
            } catch (DangerException e) {
                e.toShow();//e 调用 toShow()方法
                System.out.println(goods[i].getName() + "被禁止!");
             }
         }
    }
}
```

```
苹果不是危险品！苹果检查通过
危险品 炸药被禁止！
西服不是危险品！西服检查通过
危险品 硫酸被禁止！
手表不是危险品！手表检查通过
危险品 硫磺被禁止！

进程已结束,退出代码0
```

课后练习 1：

不可以，DangerException 是自定义的错误类型，Exception 中没有 show()方法，会报错：

```
java: 找不到符号
  符号:   方法 toShow()
  位置: 类型为java.lang.Exception的变量 e
```

课后练习 2：

不可以，DangerException 是自定义的错误类型，java.io.IOException 中没有 show()方法，会报错：

```
java: 找不到符号
  符号:   方法 toShow()
  位置: 类型为java.lang.Exception的变量 e
```

此次实验我们主要学习的是 try-catch 语句，在 Java 程序语句中，我们经 常使用 try-catch 语句来处理异常，将可能出现的异常操作方法 try-catch 语句的 try 部分，当 try 部分出现异常对象，则 try 部分会立刻结束执行， 而转向执行相应的 catch 部分；try-catch 语句中可以由几个 catch 组成， 分别处理发生的相应异常；实验后练习 1 中不能将实验 try-catch 语句中 catch 捕获的异常更改为 Exception；实验后练习 2 中不能将实验 try-catch 语句中 catch 捕获的异常更改为 java.io.IOException

实验内容（2）：

```
1. package _7_.shiyan2;
2.
3. /**
4.  * \* Created with IntelliJ IDEA.
5.  * \* User: lijihong
```

```
6.  * \* Date: 2023-04-17
7.  * \* Time: 16:40
8.  * \
9.  */
10.public class FindMess {
11.    public static void main(String args[]) {
12.        String mess = "姓名:张三 出生时间:1989.10.16 个人网
   站:http://www.zhang.com。身高:185cm,体重:72kg";
13.        int index = mess.indexOf(":"); //mess 调
   用 indexOf(String s)方法返回字符串中首次出现冒号的位置
14.        String name = mess.substring(index + 1);
15.        if (name.startsWith("张")) {
16.            System.out.println("简历中的姓名姓\"张\"");
17.        }
18.        index = mess.indexOf(":", index + 1); //mess 调
   用 indexOf(Strings,int start)方法返回字符串中第 2 次出现冒号的位置
19.        String date = mess.substring(index + 1, index + 11);
20.        System.out.println(date);
21.        index = mess.indexOf(":", index + 1);
22.        int heightPosition = mess.indexOf("身高"); // mess 调用
   indexOf(String s) 方法返回字符串中首次出现 "身高" 的位置
23.        String personNet = mess.substring(index + 1, heightPositi
   on - 1);
24.        System.out.println(personNet);
25.        index = mess.indexOf(":", heightPosition); //mess 调 用
   indexOf(String s, int start)方法返回字符串中 "身高" 后面的冒号位置
26.        int cmPosition = mess.indexOf("cm");
27.        String height = mess.substring(index + 1, cmPosition);
28.        height = height.trim();
29.        int h = Integer.parseInt(height);
30.        if (h >= 180) {
31.            System.out.println("简历中的身高" + height + "大于或等
   于 180 cm");
32.        } else {
33.            System.out.println("简历中的身高" + height + "小
   于 180 cm");
34.        }
35.        index = mess.lastIndexOf(":"); //mess 调
   用 lastIndexOf(Strings)返回字符串中最后一个冒号位置
36.        int kgPosition = mess.indexOf("kg");
37.        String weight = mess.substring(index + 1, kgPosition);
38.        weight = weight.trim();
39.
```

```
        int w = Integer.parseInt(weight);
        if (w >= 75) {
            System.out.println("简历中的体重" + weight + "大于或等于 75 kg");
        } else {
            System.out.println("简历中的体重" + weight + "小于 75 kg");
        }
        String str1 = new String("ABCABC");
        String str2 = null;
        String str3 = null;
        String str4 = null;
        str2 = str1.replaceAll("A", "First");
        str3 = str2.replaceAll("B", "Second");
        str4 = str3.replaceAll("C", "Third");
        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(Long.toBinaryString(12345));
        System.out.println(Long.toOctalString(12345));
        System.out.println(Long.toHexString(12345));
        System.out.println(Long.toString(8, 2));
    }
}
```

```
简历中的姓名姓"张"
1989.10.16
http://www.zhang.com
简历中的身高185大于或等于 180 cm
简历中的体重72小于 75 kg
```

课后练习 1：

```java
String str1 = new String( original: "ABCABC");
String str2 = null;
String str3 = null;
String str4 = null;
str2 = str1.replaceAll( regex: "A",  replacement: "First");
str3 = str2.replaceAll( regex: "B",  replacement: "Second");
str4 = str3.replaceAll( regex: "C",  replacement: "Third");
System.out.println(str1);
System.out.println(str2);
System.out.println(str3);
System.out.println(str4);
```

```
ABCABC
FirstBCFirstBC
FirstSecondCFirstSecondC
FirstSecondThirdFirstSecondThird
```

课后练习 2：

```java
System.out.println(Long.toBinaryString( i: 12345));
System.out.println(Long.toOctalString( i: 12345));
System.out.println(Long.toHexString( i: 12345));
System.out.println(Long.toString( i: 8,  radix: 2));
```

```
1100000111001
30071
3039
1000
```

通过此次实验我们了解了 String 类的常用方法，String 类是 final 类，不 可以有子类，在 Java 中，使用 java.lang 包中的 String 类创建一个字符串 变量，因此字符串变量是一个对象

实验内容（3）：

```
1. package _7_.shiyan3;
2.
3. import java.util.InputMismatchException;
4. import java.util.Scanner;
5. import java.util.regex.Matcher;
6. import java.util.regex.Pattern;
7.
8. /**
9.  * \* Created with IntelliJ IDEA.
10. * \* User: lijihong
11. * \* Date: 2023-04-17
12. * \* Time: 16:53
13. * \
14. */
15.public class ComputerPrice {
16.    public static void main(String[] args) {
17.        String menu = "北京烤鸭:189元 西芹炒肉:12.9元 酸菜鱼:69元 铁板牛肉：32元";
18.        Scanner scanner = new Scanner(menu);
19.        String regex = "[^0123456789.]+";
20.        scanner.useDelimiter(regex);
21.        double sum = 0;
22.        while (scanner.hasNext()) {
23.            try {
24.                double price = scanner.nextDouble();
25.                sum = sum + price;
26.                System.out.println(price);
27.            } catch (InputMismatchException exp) {
28.                String t = scanner.next();
29.            }
30.        }
31.        System.out.println("菜单总价格：" + sum + "元");
32.        System.out.println("-----------------------------------");
33.        String wangzhi = "中央电视台:www.cctv.com 清华大学:www.tsinghua.edu.cn";
34.//        String regex_1 = "[^(http//|www)\56?\\w+\56{1}\\w+56{1}\\p{Alpha}]+";
35.//        String regex_1 = "[^(http//|www)\\.+\\w+\\.+\\w+\\.\\p{Alpha}]+";
36.        String regex_1 = "(?i)\\b(https?://|www\\.)\\S+\\b";
```

```
        Pattern pattern = Pattern.compile(regex_1);
        Matcher matcher = pattern.matcher(wangzhi);
        while (matcher.find()) {
            String link = matcher.group();
            System.out.println(link);
        }
    }
}
```

```
189.0
12.9
69.0
32.0
菜单总价格：302.9元
------------------------------------------------
www.cctv.com
www.tsinghua.edu.cn
```

实验后的练习：

解释：

这里使用了一个正则表达式 (?i)\\b(https?://|www\\.)\\S+\\b 来匹配网址链接。该正则表达式包含以下组成部分：

1. (?i)：表示忽略大小写。
2. \\b：表示一个单词的边界，用于确保匹配的是完整的网址链接。
3. (https?://|www\\.)：表示匹配以 http:// 或 https:// 或 www. 开头的网址链接。
4. \\S+：表示匹配一个或多个非空白字符，用于匹配网址链接的其余部分。
5. \\b：表示单词边界，用于确保匹配的是完整的网址链接。

在代码中，我们首先将输入字符串和正则表达式存储在变量 wangzhi 和 regex_1 中。然后，我们使用 Pattern.compile() 方法将正则表达式编译成一个模式对象。接着，我们使用模式对象的 matcher() 方法创建一个匹配器对象，用于在输入字符串中查找匹配的文本。最后，我们使用 matcher.find() 方法循环遍历所有的匹配结果，并将它们打印到控制台上。

4. 实验总结

写出实验中的心得体会（对第 7 章理论课重点简述）。

内部类与异常：

Java 中的内部类和异常是两个不同的概念，它们之间没有直接的连接。但是，它们都是 Java 语言中比较重要的概念，下面分别介绍它们的相关知识点和易错细节。

内部类知识点总结

知识点

Java 中的内部类是定义在另一个类中的类。内部类可以访问包含它们的外部类的私有数据，这使得内部类具有一些特殊的能力和灵活性。在 Java 中，有四种类型的内部类：

成员内部类：定义在另一个类的内部，并且是该类的成员。

静态内部类：定义在另一个类的内部，并且被声明为 static。

局部内部类：定义在方法或代码块中的类。

匿名内部类：没有类名的内部类，通常用于实现接口或继承抽象类。

在 Java 中，内部类与外部类之间存在一些关系和限制。例如，内部类可以访问外部类的私有成员，但是外部类不能访问内部类的私有成员。此外，内部类可以访问它所在方法中的 final 变量，但是不能修改它们的值。

易错细节

在使用内部类时，可能会遇到一些常见的错误和注意事项，例如：

内部类的实例化方式：由于内部类是定义在另一个类中的，因此要创建内部类的实例，必须先创建外部类的实例。例如，如果要创建成员内部类的实例，可以使用以下代码：

```
OuterClass outer = new OuterClass();
OuterClass.InnerClass inner = outer.new InnerClass();
```

内部类的访问控制：内部类可以访问包含它们的外部类的私有成员，但是外部类不能访问内部类的私有成员。因此，在使用内部类时，需要注意访问控制的限制。

内部类的作用域：内部类可以访问其包含类的成员，但是其作用域范围限制在其包含类的作用域内。因此，在使用内部类时，需要注意其作用域的限制。

内部类的名称：由于内部类是定义在另一个类中的，因此其名称必须遵循特定的命名规则。例如，成员内部类的名称必须使用形如 OuterClass.InnerClass 的语法。

异常知识点总结

知识点

在 Java 中，异常是一种表示程序错误或异常情况的机制。当程序发生异常时，它会抛出一个异常对象，可以使用 try-catch 块来捕获和处理这些异常。

Java 中的异常分为两种类型：受检异常（checked exception）和非受检异常（unchecked exception）。受检异常是指在编译时必须处理的异常，例如文件不存在、网络连接失败等。如果不处理这些异常，编译器会报错。非受检异常是指运行时可能发生的异常，例如空指针异常、数组下标越界等。这些异常不需要在编译时处理，但是如果不处理它们，程序将在运行时崩溃。

在 Java 中，可以使用 try-catch-finally 块来捕获和处理异常。try 块中包含可能抛出异常的代码，catch 块用于捕获和处理异常，finally 块用于执行清理操

作，例如关闭文件或释放资源。try-catch-finally 块的基本语法如下：

```
try {
    // 可能抛出异常的代码
} catch (ExceptionType1 e1) {
    // 处理 ExceptionType1 异常
} catch (ExceptionType2 e2) {
    // 处理 ExceptionType2 异常
} finally {
    // 清理代码，无论是否发生异常都会执行
}
```

在捕获和处理异常时，需要注意以下几点：

捕获异常的顺序：如果多个 catch 块可以处理同一类型的异常，应该把最具体的异常类型的 catch 块放在前面，最一般的异常类型的 catch 块放在后面。

抛出异常的语句：可以使用 throw 语句手动抛出异常。在抛出异常时，需要选择合适的异常类型，并提供清晰的异常信息。

自定义异常：可以通过继承 Exception 或 RuntimeException 类来创建自定义异常。自定义异常应该提供清晰的异常信息，并尽可能地提供详细的堆栈跟踪信息，以便快速诊断和解决问题。

try-with-resources 语句：在 Java 7 中引入了 try-with-resources 语句，可以用于自动关闭资源，例如文件或数据库连接。try-with-resources 块中可以包含一个或多个资源对象，这些对象必须实现 AutoCloseable 接口。在 try-with-resources 块结束时，Java 会自动关闭这些资源。

内部类和异常的易错细节

在使用异常时，可能会遇到一些常见的错误和注意事项，例如：

捕获异常的类型不够具体：如果捕获的异常类型过于宽泛，可能会导致错误的处理方式。例如，如果捕获 Exception 类型的异常，可能会屏蔽真正的异常信息，使问题难以诊断。

忘记处理异常：受检异常必须在编译时处理，否则编译器会报错。因此，在使用可能抛出受检异常的方法时，需要考虑如何处理这些异常。

不正确地使用 finally：finally 块用于执行清理代码，例如关闭文件或释放资源。在 finally 块中应该避免抛出异常，否则可能会导致更严重的问题。另外，如果 try 或 catch 块中包含 return 或 throw 语句，finally 块仍然会执行。

不正确地抛出异常：在抛出异常时，需要选择合适的异常类型，并提供清晰的异常信息。如果异常信息不够清晰，可能会导致问题难以诊断。另外，在抛出异常时，应该提供详细的堆栈跟踪信息，以便快速定位问题。

没有关闭资源：在使用文件、网络连接等资源时，需要手动关闭这些资源，否则可能会导致资源泄漏或占用过多的系统资源。可以使用 try-with-resources 语句来自动关闭资源，以减少这种问题的发生。

自定义异常不符合规范：如果要创建自定义异常，需要继承 Exception 或 RuntimeException 类，并提供清晰的异常信息。另外，自定义异常应该符合 Java 命名规范，并以 Exception 结尾。

异常处理不符合规范：在处理异常时，需要遵循 Java 异常处理的规范。例

如，捕获异常的顺序应该从具体到一般，避免屏蔽真正的异常信息。另外，可以在 finally 块中执行清理代码，避免资源泄漏或占用过多的系统资源。

不正确地捕获异常：在捕获异常时，应该捕获特定的异常类型，而不是捕获通用的 Exception 类型。这样可以更精确地定位问题，并提供更清晰的异常信息。

忽略异常：在处理异常时，不能忽略异常。即使无法处理异常，也应该记录异常信息，并考虑向上抛出异常或者返回默认值等方式来保证程序的正常运行。

处理异常的方式不当：在处理异常时，应该根据具体情况选择适当的方式，例如记录日志、抛出异常、返回默认值等。如果处理异常的方式不当，可能会导致更严重的问题。

并发异常：在多线程环境下，需要特别注意并发异常。例如，可能出现竞态条件、死锁等问题，需要通过合理的并发控制来避免这些问题的发生。

异常链：在抛出异常时，可以使用异常链来将多个异常连接起来。这样可以更精确地描述问题，帮助调试和定位问题。

异常处理效率问题：异常处理通常比较耗时，如果在高性能的系统中过度使用异常，可能会影响系统的性能。因此，在高性能的系统中，需要仔细考虑异常的使用方式，避免异常处理成为系统瓶颈。

异常处理与业务逻辑混淆：在处理异常时，需要避免将异常处理和业务逻辑混淆在一起。应该将异常处理和业务逻辑分离开来，使代码更加清晰易懂。

异常处理与代码复杂度增加：在过度使用异常处理时，可能会导致代码的复杂度增加，使代码难以维护和调试。因此，在使用异常处理时，需要适当控制异常的数量和使用方式。

异常处理与测试覆盖率：异常处理可能会对测试覆盖率产生影响。例如，可能需要编写额外的测试代码来覆盖异常处理的分支。因此，在编写测试代码时，需要考虑异常处理的情况，并尽可能地覆盖所有分支。

异常处理与系统安全性：异常处理可能会对系统安全性产生影响。例如，可能会在异常处理中暴露敏感信息，或者使系统容易受到攻击。因此，在编写异常处理代码时，需要考虑系统的安全性，并采取相应的措施来保护系统安全。

异常处理与可读性：异常处理可能会对代码的可读性产生影响。例如，可能会导致代码冗长，或者使代码难以理解。因此，在编写异常处理代码时，需要尽可能地使代码简洁易懂，提高代码的可读性。

异常处理与设计模式：在使用设计模式时，需要考虑异常处理的情况，并结合具体的设计模式来实现异常处理。例如，可以使用策略模式来处理异常，或者使用模板方法模式来定义异常处理流程等。

异常处理与代码规范：异常处理需要符合代码规范和最佳实践。例如，应该避免在 try-catch 块中放置过多的代码，应该使用 try-with-resources 语句来自动关闭资源等。

异常处理与代码重构：在进行代码重构时，需要考虑异常处理的情况，并尽可能地将异常处理与业务逻辑分离开来。同时，需要遵循重构原则，保证代码的可读性、可维护性和可扩展性。

异常处理与开发团队协作：在多人协作开发时，需要统一异常处理的方式和规范，避免出现不一致的情况。同时，需要进行异常处理的培训和知识分享，提高整个团队的异常处理能力。

总之，异常处理是编程中不可避免的一部分，需要程序员们认真对待。正确

使用异常处理可以提高程序的可靠性和健壮性，而不当使用异常处理则可能会导致程序的错误和异常。因此，程序员们需要不断学习和实践，提高异常处理的技能和能力。

内部类与多线程编程：内部类可以用于多线程编程中的实现，例如可以定义一个实现 Runnable 接口的内部类来实现线程功能。内部类还可以与线程池一起使用，提高程序的性能和效率。

内部类与代码可读性：使用内部类可以提高代码的可读性，特别是当内部类只用于一个外部类时。使用内部类可以将相关的代码组织在一起，减少类的数量，同时也可以避免外部类被其他类访问，提高代码的封装性。

内部类与访问控制：内部类可以访问外部类的私有成员和方法，但是外部类无法直接访问内部类的私有成员和方法。因此，在设计类的访问控制时，需要考虑内部类的访问权限。

内部类与继承：内部类可以被继承，而且可以被声明为 private 或 protected。当一个内部类被继承时，子类可以使用父类的内部类，但是子类无法访问父类内部类的私有成员和方法。

总之，内部类是 Java 语言的一个重要特性，具有许多优点和应用场景。程序员们需要了解内部类的特性和使用方法，以便在编写代码时充分发挥内部类的优势。