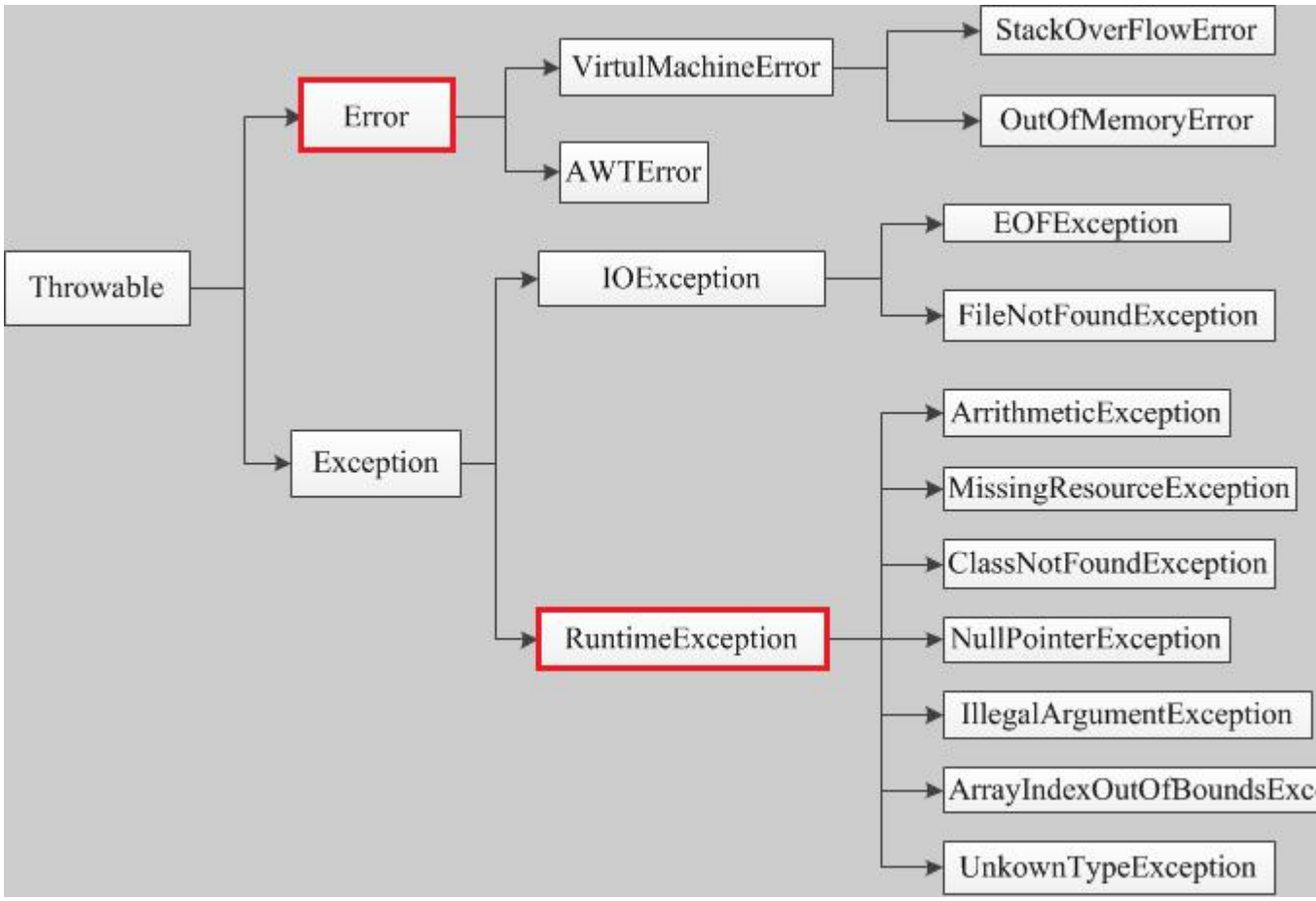
课 后 作 业

课程名称	Java 应用与开发	开课学期	2018 年秋季学期
班　　级	计算机一班	姓　　名	陈扬
学　　号	17150011001	联系方式	
完成情况	（不要填）		

简答题

1. 总结 Java 的异常处理机制。



Java 的异常分为 Error 类异常和 Exception 类异常 ①Error 类异常是指 java 运行时系统的内部错误和资源耗尽错误。应用程序不会抛出该 类对象。如果出现了这样的错误，除了告知用户，剩下的就是尽力使程序安全的终止。 ②Exception 又有两个分支，一个是运行时异常 RuntimeException，一个是检查异常 CheckedException。 处理 Exception 类异常可以用以下的方式： ①遇到问题不进行具体处理，而是继续抛给调用者，即 throw,

throws，和系统自动抛异 常。 ②针对性处理方式：捕获异常，即 try，catch 语句

2. 什么是运行时异常？ 运行时异常是不需要捕获的，程序员可以不去处理，当异常出现时，虚拟机会处理。常 见的运行时异常有空指针异常。 常见的运行时异常包括： (1) ClassCastException(类 转 换 异 常) (2) IndexOutOfBoundsException(数 组 越 界) (3) NullPointerException(空指针) (4) ArrayStoreException(数据存储异常，操作数组时类型不一致) (5) 还有 IO 操作的 BufferOverflowException 异常

3. 若 try 语句结构中有多个 catch() 子句，这些子句的排列顺序与程序执行效果是否有关？ 有关，异常处理时程序只会按顺序寻找第一个匹配的 catch()句子，即最多只会执行多个 catch()中的一个。所以在写 try，catch 语句时需要考虑异常处理顺序。

4. 总结 Java 异常处理机制随 Java 版本的更新不断加入的新特性，并附参考文献或网站链 接。
1 try-with-resources 语句

Java7 中提供了一种更为简单实用的用于处理资源使用异常处理的特性，称为 try-with-resources，这个所谓的 try-with-resources，是个语法糖。实际上就是自动调用资源的 close()函数。使用 try-with-resources 的语法可以实现资源的自动回收处理，是代码更为简洁，运行更为稳定。

2 捕获多个异常

在 Java7 中，可以在同一个 catch 分支中捕获多个异常类型

3 异常重新抛出

另一个升级是编译器对重新抛出异常（rethrown exceptions）的处理。这一特性允许在一个方法声明的 throws 从句中指定更多特定的异常类型。Java SE 7 的编译器能够对再次抛出的异常(rethrown exception)做出更精确的分析。这使得你可以在一个方法声明的 throws 从句中指定更具体的异常类型。

4 更简单的处理反射方法的异常

在 Java7 之前，当调用一个反射方法时，不得不捕获多个不相关的检查期异常。

1. 概述 Java I/O 流的分类。 节点流(Node Stream) 直接与节点(如文件)相连 可以从 / 向一个特定的地方(节点)读 / 写数据 如文件流 FileInputStream，内存流 ByteArrayInputStream 字节流：以字节为单位进行操作 ：InputStream / OutputStream 字符流：以字符为单位进行操作 ： Reader / Writer 处理流(Processing Stream) 是对一个已存在的流的连接和封装，处理流又称为过滤流(Filter) 如缓冲处理流 BufferedReader 对节点流或其他流进一步进行处理，如缓冲，组装成对象，等等 过滤流只能建立在节点流的基础上，即处理流的构造方法要以一个流作为参数。

2. 总结补全幻灯片中基础 I/O 流部分各方法的功能和用法。
字符流：

输入流：FileReader

1. Reader 类常用方法

【作用和用法都和 InputStream 一样，正常使用就可以】：

int read( )

int read(char[ ] c)

read(char[ ] c,int off,int len)

void close( )

为了解决中文乱码使用子类 InputStreamReader

2. 子类 InputStreamReader 常用的构造方法：

InputStreamReader(InputStream in)

InputStreamReader(InputStream in,String charsetName)

【支持输入的字符流，并且可以规定输入的字符编码格式】

3. 孙子类 FileReader 类是 InputStreamReader 的子类【作用和用法都和 FileInputStream 一样，正常使用就可以】：

FileReader(File file)

FileReader(String name)

该类只能按照系统默认的字符编码来读取数据，用户不能指定其他的字符编码类型

System.out.println(System.getProperty("file.encoding"));获得本地平台的字符编码类型

有中文乱码的缺陷所以一般使用 InputStreamReader 并且和 FileInputStream fis = new FileInputStream("……"); 配合着来用

输出流：FileWriter

1. Writer 类常用方法：

write(String str)

write(String str,int off,int len)

void close()

void flush()

2. 子类 OutputStreamWriter（可以指定字符编码格式）常用的构造方法：

OutputStreamWriter(OutputStream out)

OutputStreamWriter(OutputStream out, String charsetName)

3. 孙子类 FileWriter：以下两种构造都可以重载，指定一个 boolean 类型的参数，用来指定追加还是覆盖文件内容

new FileWriter (File file)

new FileWriter (String path)

一定要记住使用.flush()和.close()，这样才能把存在缓冲区中的数据冲出来。

https://blog.csdn.net/qq_37267015/article/details/58653406

小编程：1. 编程实践任意类型文件和文本文件复制代码。

```
import java.io.*;
public class FileCopy{
    public static void main(String[] args)throws IOException{
        FileInputStream read = new FileInputStream(new File("/*要复制的文件地址及文件名*/"));
        FileOutputStream wr = new FileOutputStream(new File("/*要输出的文件地址及文件名*/"));
        byte[] b = new byte[1024];
```

```
            int len = 0;
            while((len=read.read(b))!=-1){
                wr.write(b,0,len);
                wr.flush();
            }
            wr.close();
            read.close();
        }
} 2. 编程实践对象序列化代码。
package java.com.iostream;

public class Person {
  String name = "K";
  String sex = "M";
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getSex() {
    return sex;
  }
  public void setSex(String sex) {
    this.sex = sex;
  }
  @Override
  public String toString() {
    return "Person [name=" + name + ", sex=" + sex + "]";
  } }
package java.com.iostream;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Read {
  private static ObjectInputStream inputStream;
  public static void main (String[] args) {
    try {
      inputStream                                    =                                    new
ObjectInputStream(Files.newInputStream(Paths.get("clients.ser")));
    } catch (IOException e) {
```

```
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    try {
      Person person = (Person) inputStream.readObject();
      System.out.println(person.getName()+person.getSex());
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    try {
      inputStream.close();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
package java.com.iostream;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Read {
  private static ObjectInputStream inputStream;
  public static void main (String[] args) {
    try {
      inputStream                                    =                                    new
ObjectInputStream(Files.newInputStream(Paths.get("clients.ser")));
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    try {
      Person person = (Person) inputStream.readObject();
      System.out.println(person.getName()+person.getSex());
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
```

```
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    try {
      inputStream.close();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
```