异常处理

孙聪

网络与信息安全学院

2019-10-28

课程内容

- Java概述
- 面向对象程序设计概念
- Java语言基础
- Java面向对象特性
- Java高级特征
- 容器类
- 常用预定义类
- 异常处理
- 输入输出
- 线程

提要

提要

异常的概念

- 在程序运行时，打断正常程序流程的不正常情况分两类
  - 错误(Error)：应用程序无法捕获的严重问题
  - 异常(Exception)：应用程序可捕获的一般问题
- 例：
  - 试图打开的文件不存在
  - 网络连接中断
  - 数组越界
  - 要加载的类找不到
  - ...

声明一个字符串数组，通过while循环输出数组中的各字符串

```
1   public class HelloWorld{
2       public static void main(String args[]){
3           int i=0;
4           String greetings[]={ “Hello World!”, “Hello!”,
5                                “HELLO WORLD!”};
6           while (i<4){
7               System.out.println(greetings[i]);
8               i++;
9           }
10          System.out.println( “end!” );
11      }
12  }
```

声明一个字符串数组，通过while循环输出数组中的各字符串

```
1   public class HelloWorld{
2       public static void main(String args[]){
3           int i=0;
4           String greetings[]={ “Hello World!”, “Hello!”,
5                                “HELLO WORLD!”};
6           while (i<4){
7               System.out.println(greetings[i]);
8               i++;
9           }
10          System.out.println( “end!” );
11      }
12  }
```

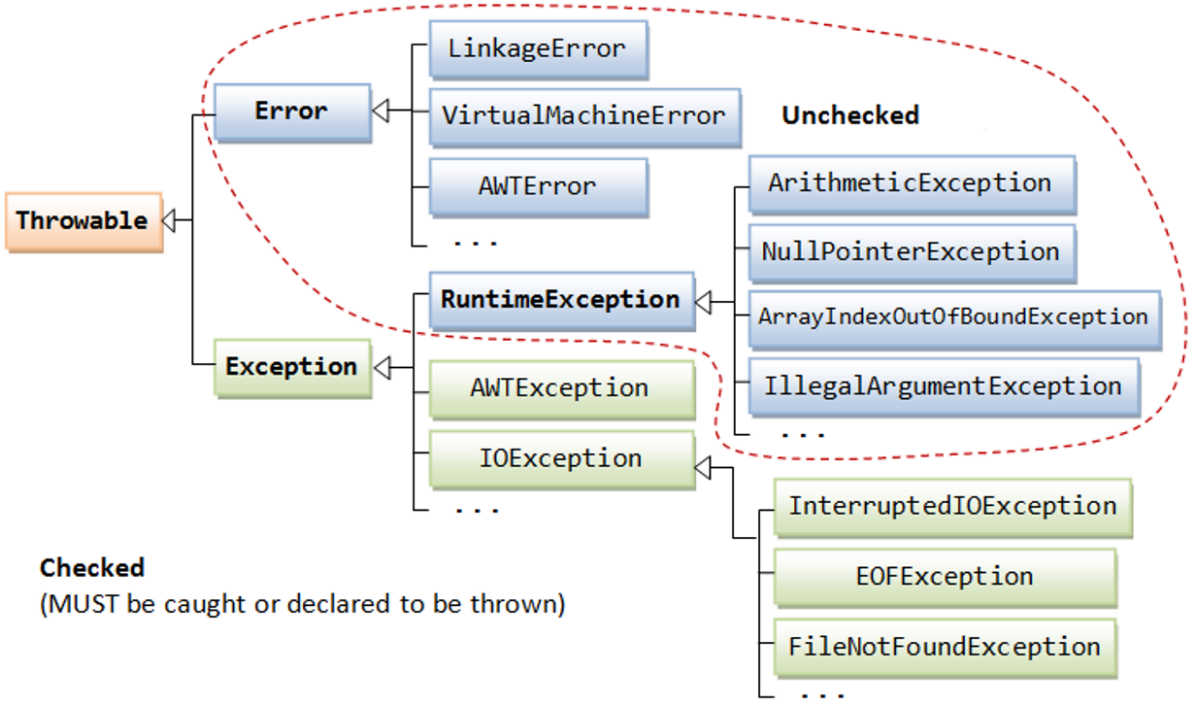
```
Hello World!
Hello!
HELLO WORLD!
Exception in thread “main” Java.lang.ArrayIndexOutOfBoundsException
        at HelloWorld.main(HelloWorld.java:7)
```

- 产生异常的语句是第7行，异常的名称是数组越界
- 程序运行中出现了异常，导致了程序的非正常终止

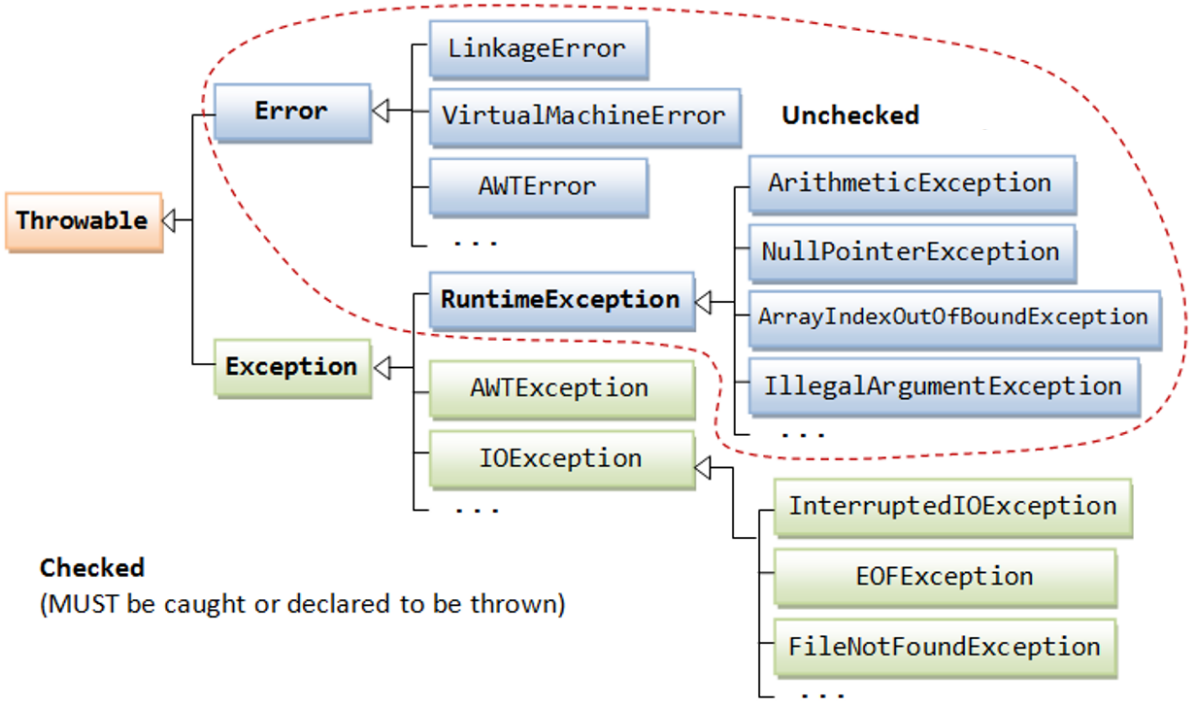
异常的概念

- 系统错误：虚拟机相关的问题，如虚拟机崩溃、动态链接失败、低层资源错误等
  - 总是不受编译器检查的（Unchecked）
  - 可以被抛出，但无法恢复，不可能被捕获
- 异常
  - Runtime异常（免检异常）：由Runtime异常类及其子类表示的异常，如数组越界、算术运算异常、空指针异常等
    - 不受编译器检查（Unchecked），不需要显式地处理（捕获或抛出）该异常就能编译通过
  - 必检异常：除Runtime异常类及其子类之外的所有异常，如文件不存在、无效URL等
    - 编译器检查并强制程序对其进行异常处理（Checked）

异常类层次

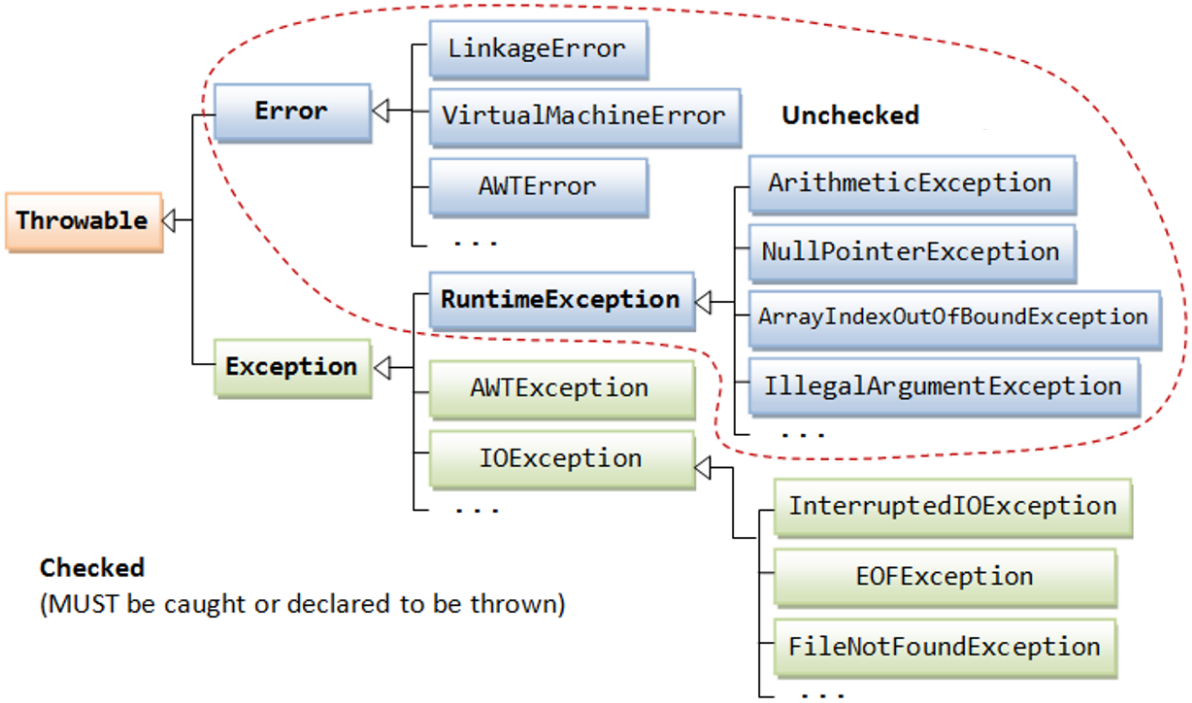

Throwable
Error
LinkageError
VirtualMachineError
AWTError
...
Exception
RuntimeException
ArithmeticException
NullPointerException
ArrayIndexOutOfBoundException
IllegalArgumentException
...
AWTException
IOException
...
InterruptedIOException
EOFException
FileNotFoundException
...
Unchecked
Checked
(MUST be caught or declared to be thrown)

异常类层次



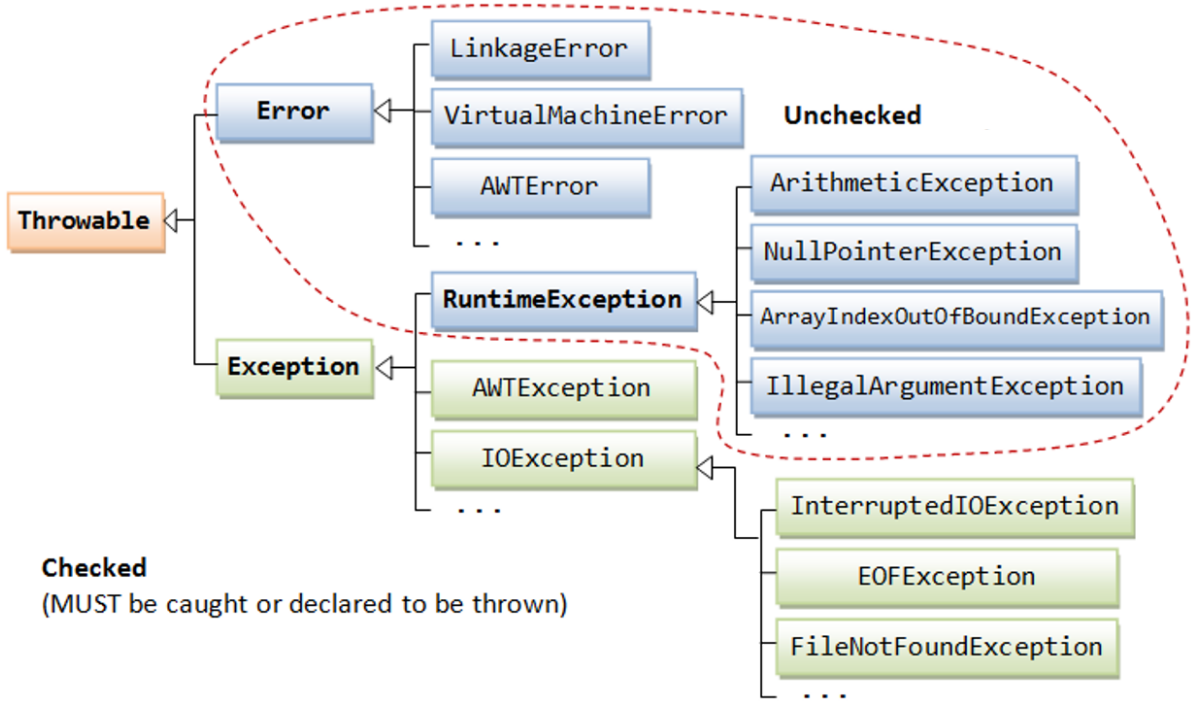
- Java.lang.Throwable: 所有错误类和异常类的父类
  - 检索异常的相关信息
  - 输出显示异常发生位置的堆栈追踪轨迹（PrintStackTrace()）

异常类层次



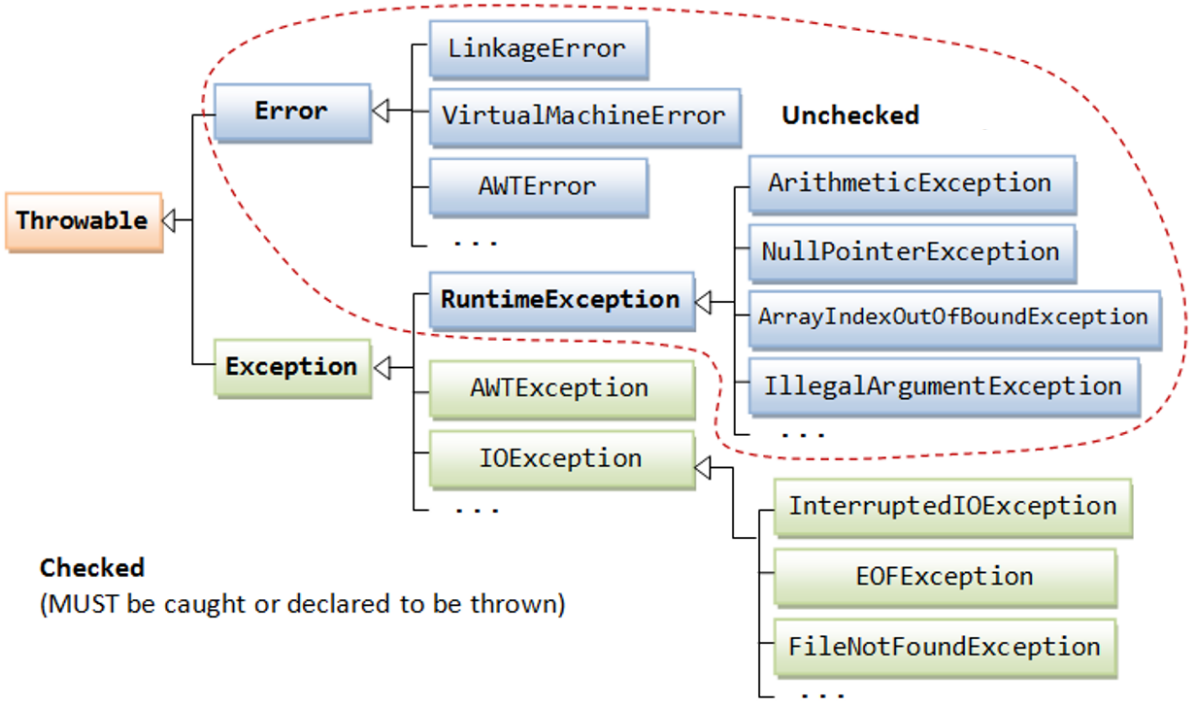
- java.lang.ArithmeticException: 算数运算异常
  - 例: 整数的除0操作导致的异常, int i=10/0;

异常类层次



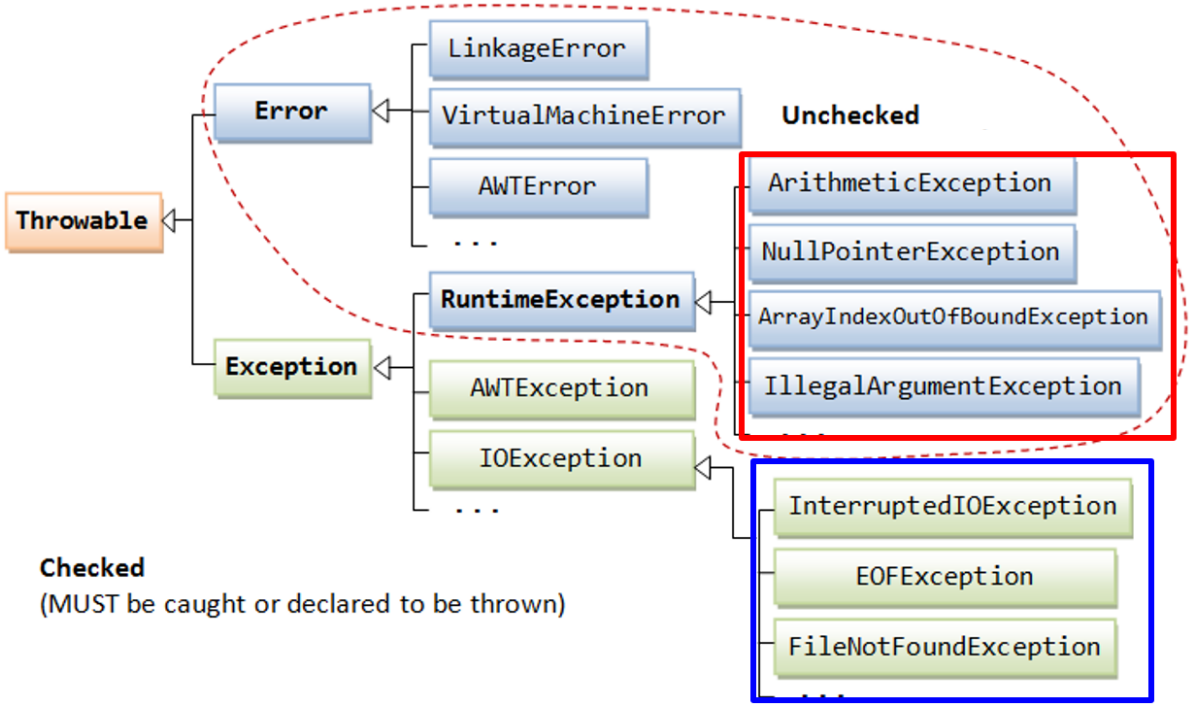
- java.lang.NullPointerException: 需要使用对象但仅能获得null时抛出。具体情况包括:
  - 调用一个null对象的实例方法
  - 访问或修改一个null对象的成员变量
  - 在数组变量引用到null对象时, 访问数组的length或具体的数组元素
  - 将null作为Throwable对象抛出
  - 例: Date d=null; System.out.println(d.toString());

异常类层次



- java.io.IOException: 输入/输出时可能产生的各种异常

异常类层次



- 异常一般是由程序员的疏忽或者环境的变化所导致的
- 若不对异常进行处理，则会导致程序的不正常终止，为保证程序正常运行，Java提供了异常处理机制

提要

异常处理方法

- 捕获并处理异常
- 将方法中产生的异常抛出

捕获并处理异常

通过try-catch-finally语句来实现，基本格式：

```
try{ /** 监控区域 */
    //一条或多条可能抛出异常的Java语句
} catch (ExceptionType1 e1){  /** 异常处理程序 */
    //捕获到ExceptionType1类型的异常时执行的代码
} catch (ExceptionType2 e2){  /** 异常处理程序 */
    //捕获到ExceptionType2类型的异常时执行的代码
} ...
finally{
    //执行最终清理的语句
}
```

- try
  - 把可能出现异常的语句都放在try语句块中
  - try语句块之后必须紧跟至少一个catch语句块

捕获并处理异常

通过try-catch-finally语句来实现，基本格式：

```
try{ /** 监控区域 */
    //一条或多条可能抛出异常的Java语句
} catch (ExceptionType1 e1){  /** 异常处理程序 */
    //捕获到ExceptionType1类型的异常时执行的代码
} catch (ExceptionType2 e2){  /** 异常处理程序 */
    //捕获到ExceptionType2类型的异常时执行的代码
} ...
finally{
    //执行最终清理的语句
}
```

- catch (ThrowableType objRef) {...}
  - ThrowableType：当前catch语句块能够处理的异常类型，必须是Throwable类的子类
  - objRef：异常处理程序中使用的指向被捕获异常对象的引用

捕获并处理异常

通过try-catch-finally语句来实现，基本格式：

```
try{ /** 监控区域 */
    //一条或多条可能抛出异常的Java语句
} catch (ExceptionType1 e1){  /** 异常处理程序 */
    //捕获到ExceptionType1类型的异常时执行的代码
} catch (ExceptionType2 e2){  /** 异常处理程序 */
    //捕获到ExceptionType2类型的异常时执行的代码
} ...
finally{
    //执行最终清理的语句
}
```

- finally
  - 用于将除内存之外的资源恢复到初始状态。需清理的资源包括：已打开的文件或网络连接，在屏幕上画的图形等
  - finally语句块可以省略
  - 若finally语句块存在，则无论是否发生异常均执行

```
//创建一个保存10个Integer对象的容器，并通过writeList方法将其中数据保存到OutFile.txt中
public class ListOfNumbers {
    private ArrayList<Integer> list;
    private static final int size = 10;
    public ListOfNumbers() {
        list = new ArrayList<Integer>(size);
        for (int i = 0; i < size; i++)
            list.add(new Integer(i));    }
    public void writeList(){
        PrintWriter out=new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i=0;i<size;i++)
            out.println("Value at: "+i+" = "+list.get(i));
        out.close();    }
    public static void main(String args[]) {
        ListOfNumbers list = new ListOfNumbers();
        list.writeList();    }
}
```

```
//创建一个保存10个Integer对象的容器，并通过writeList方法将其中数据保存到OutFile.txt中
public class ListOfNumbers {
    private ArrayList<Integer> list;
    private static final int size = 10;
    public ListOfNumbers() {
        list = new ArrayList<Integer>(size);
        for (int i = 0; i < size; i++)
            list.add(new Integer(i));    }
    public void writeList(){
        PrintWriter out=new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i=0;i<size;i++)
            out.println("Value at: "+i+" = "+list.get(i));
        out.close();    }
    public static void main(String args[]) {
        ListOfNumbers list = new ListOfNumbers();
        list.writeList();    }
}
```

```
F:\>javac ListOfNumbers.java
ListOfNumbers.java:16: 错误: 未报告的异常错误IOException; 必须对其进行捕获或声明
以便抛出
                PrintWriter out=new PrintWriter(new FileWriter("OutFile.txt"));
                                                ^
1 个错误
```

```
//创建一个保存10个Integer对象的容器，并通过writeList方法将其中数据保存到OutFile.txt中
public class ListOfNumbers {
    private ArrayList<Integer> list;
    private static final int size = 10;
    public ListOfNumbers() {
        list = new ArrayList<Integer>(size);
        for (int i = 0; i < size; i++)
            list.add(new Integer(i));    }
    public void writeList(){
        PrintWriter out=new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i=0;i<size;i++)
            out.println("Value at: "+i+" = "+list.get(i));
        out.close();    }
    public static void main(String args[]) {
        ListOfNumbers list = new ListOfNumbers();
        list.writeList();    }
}
```

- 调用了java.io.FileWriter的构造方法创建文件输出流，该方法声明如下：
  - public FileWriter(String fileName) throws IOException;
- FileWriter()构造方法可以抛出必检异常IOException
- writeList()方法中没有对该必检异常IOException进行处理，故程序编译时报错
- 此外，容器对象的get()方法还可能抛出免检异常IndexOutOfBoundsException

更改writeList()方法

```
public void writeList() {
    PrintWriter out = null;
    try {
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < size; i++)
            out.println("Value at: " + i + " = " + list.get(i));
    } catch (IndexOutOfBoundsException e) {
        System.err.println("Caught IndexOutOfBoundsException: "
                           + e.getMessage());
    } catch (IOException e) {
        System.err.println("Caught IOException: " + e.getMessage());
    } finally {/*执行程序的最后清理操作，关闭程序打开的文件流*/
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        } else { System.out.println("PrintWriter not open"); }
    }
}
```

习题7-1. 以下程序的输出是什么？

```
public static void main(String args[]){
    int i = 0 ;
    String greetings[]={ “Hello World!” , “Hello!” , “HELLO!” };
    while (i < 4){
        try {
            System.out.println(greetings[i]);
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println( “Re-setting Index Value” );
            i = -1;
        }finally{
            System.out.println( “This is always printed” );
        }
        i++;
    }
}
```

多种异常同时处理

- 可编写针对Exception的任何子类的catch块
- 子类的异常对象可与父类的异常处理程序匹配
  - 若catch块针对叶节点，则是专用的异常处理，捕获一种特定的异常
  - 若catch块针对中间节点，则是通用的异常处理，捕获该节点及其所有子类表示的异常

处理多种异常

```
public void writeList() {
    PrintWriter out = null;
    try {
        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < size; i++)
            out.println("Value at: " + i + " = " + list.get(i));
    } catch (IOException e) {
        System.err.println("Caught IOException: " + e.getMessage());
    } catch (Exception e){ /*所有非IOException*/
        System.err.println("Caught Exception: " + e.getMessage());
    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        } else { System.out.println("PrintWriter not open"); }
    }
}
```

将方法中产生的异常抛出

- 可能产生异常的方法不处理该异常，而是将该异常抛出到调用该方法的程序
  - 例：`public void troublesome() throws IOException { ... }`

将方法中产生的异常抛出

- 可能产生异常的方法不处理该异常，而是将该异常抛出到调用该方法的程序
  - 例：`public void troublesome() throws IOException { ... }`
- 声明抛出异常（方法声明的throws子句）
  - `retType mtdName ([paralist]) throws [exceptionList] {...}`

将方法中产生的异常抛出

- 可能产生异常的方法不处理该异常，而是将该异常抛出到调用该方法的程序
  - 例：`public void troublesome() throws IOException { ... }`
- 声明抛出异常（方法声明的throws子句）
  - `retType mtdName ([paralist]) throws [exceptionList] {...}`
- 抛出异常（throw语句）
  - `throw someThrowableObj;`
  - 执行throw语句后，在当前方法中找是否有catch字句匹配抛出的异常someThrowableObj的类型：
    - 若找到，则由该catch子句处理
    - 若未找到，则转向上一层调用者程序，在调用者程序中查找是否有catch子句匹配someThrowableObj
    - 依次递归…
  - 若一个异常在转向到main()后还未被处理，则程序将非正常终止

```
class ListOfNumbersDeclared {
    private ArrayList<Integer> list;
    private static final int size = 10;
    public ListOfNumbersDeclared() {
        list = new ArrayList<Integer>(size);
        for (int i = 0; i < size; i++)
            list.add(new Integer(i));
    }
    public void writeList()
            throws IOException, IndexOutOfBoundsException {//声明抛出异常
        PrintWriter out = new PrintWriter(new FileWriter("OutFile1.txt"));
        for (int i = 0; i < size; i++)
            out.println("Value at: " + i + " = " + list.get(i));
        out.close();
    }
}
public class TestListOfNumbersDeclared {
    public static void main(String args[]) {
        try {
            ListOfNumbersDeclared list = new ListOfNumbersDeclared();
            list.writeList();
        } catch (Exception e) { }
        System.out.println("A list of numbers created and stored in OutFile1.txt");
    }
}
```

对于以下两个程序，假设s2语句会引起异常，分别说明

- s3是否会执行？
- 如果异常被捕获，哪些语句会被执行？
- 如果异常未被捕获，哪些语句会被执行？

```
try{
    s1;
    s2;
    s3;
} catch(ExceptionType1 e){
    ...
} catch(ExceptionType2 e){
    ...
}
finally{
    s4;
}
s5;
```

```
try{
    s1;
    s2;
    s3;
} catch(ExceptionType1 e){
    ...
} catch(ExceptionType2 e){
    ...
}
s5;
```

提要

自定义异常类

- 在实际程序设计中，尽可能使用预定义的异常类型能够降低程序的复杂度
- 但预定义异常类体系无法预见所有可能出现的程序错误，因为现实中存在无法用预定义异常类描述的问题，因此需要用户自己定义异常类

自定义异常类

- 在实际程序设计中，尽可能使用预定义的异常类型能够降低程序的复杂度
- 但预定义异常类体系无法预见所有可能出现的程序错误，因为现实中存在无法用预定义异常类描述的问题，因此需要用户自己定义异常类
- 定义方法
  - 自定义异常类必须从已有的异常类继承，因而必然是Exception类的子类
  - 最好选择意思相近的异常类作为新异常类的父类
  - 自定义异常类可包含普通类的内容
  - 尽可能避免从RuntimeException派生自定义异常类，因为RuntimeException免检，而我们希望自定义异常必检，这样编译器就会强制程序捕获这些异常，从而提高程序健壮性

自定义异常类的定义

描述通信中客户端和服务器连接超时的异常

```
public class ServerTimeOutException extends Exception{
    private String reason;
    private int port ;
    public ServerTimeOutException(String reason, int port){
        this.reason = reason;
        this.port = port;
    }
    public String getReason(){
        return reason;
    }
    public int getPort(){
        return port;
    }
}
```

自定义异常类的使用

抛出自定义异常

```
public void connectMe(String serverName) throws ServerTimeOutException{
    int success;
    int portToConnect = 80;
    success = open(serverName, portToConnect);
    if(success= -1)
        throw new ServerTimedOutException(“Could not connect”,80);
}
```

自定义异常类的使用

处理可能抛出的自定义异常

```
public void findServer(){
    ...
    try{
        connectMe(defaultServer);
    } catch (ServerTimeOutException e){
        System.out.println("Server timed out, try another");
        try{ //try-catch语句块是可以嵌套的
            connectMe(alternateServer);
        } catch (ServerTimeOutException e1) {
            System.out.println("No server available");
        }
    }
}
```

- 例：TestMyException.java