第10章 异常处理

第十章 异常处理

- Java异常处理
- Exception类
- 使用异常处理
  - try/catch/finally块
  - catch块
  - 抛出异常
- 自定义异常

10.1 Java异常处理

- Object类的子类Throwable 类是所有错误或异常的父类。

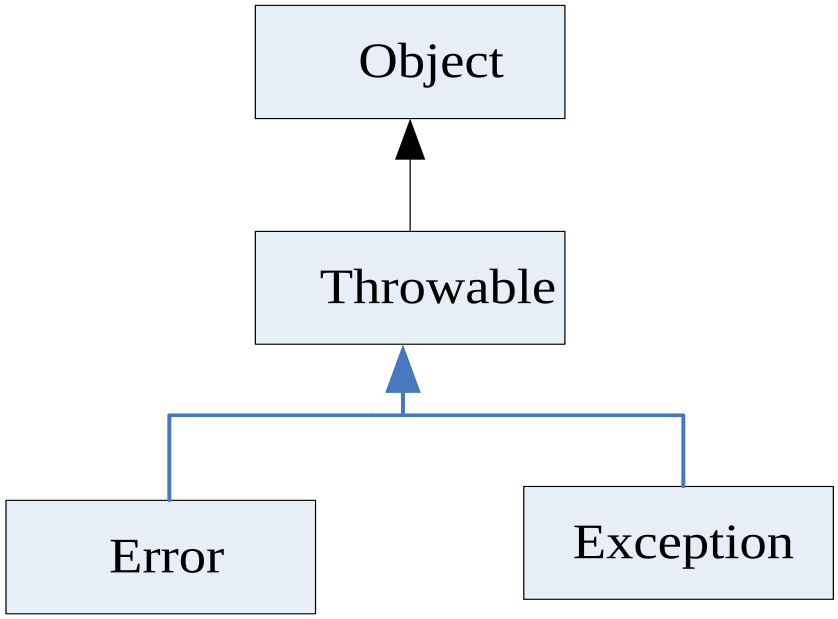
- 只有当对象是此类（或子类之一）的实例时，才能通过Java虚拟机或者Java throw语句抛出。

- 只有此类或其子类之一才可以是catch子句中的参数类型。

10.1 Java异常处理

Throwable 类有两个子类：Error类和Exception类

- Error类是对应严重的问题，应用程序不处理Error。
- Exception类称作异常，应用程序需要处理异常。

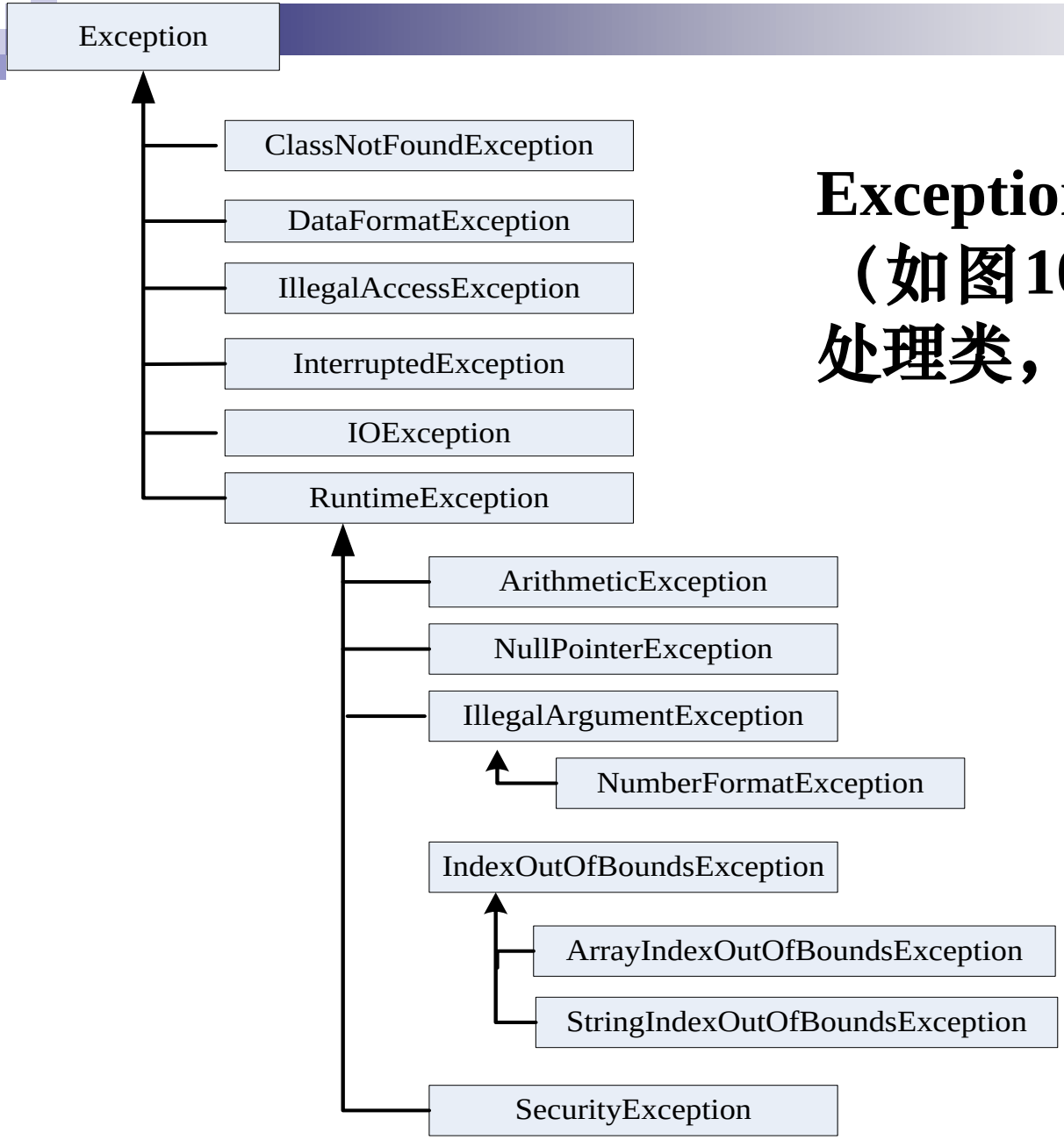


- Throwable类包含多个构造方法和操作方法，下面列出一些常用方法。

方法	说明
public Throwable()	构造方法
public String getMessage()	返回此throwable的详细消息字符串
public void printStackTrace()	将此throwable及其追踪输出至标准错误流
public String toString()	返回此throwable的简短描述

10.2 Exception类

- Exception类是异常处理的父类。
- 异常有多种类型，例如I/O异常、数字格式异常、文件未找到异常、数组越界异常等。Java将这些异常分成不同的类，放在不同的包中。
- Java预定义异常分为两种：检查异常（checked exception）和非检查异常（unchecked exception）。RuntimeException类及其子类是非检查异常，即程序不一定要对异常进行处理。
- 检查异常是非RuntimeException类及其子类的异常。例如IOExeption、SQLException等。程序必须要对检查异常进行处理。



Exception 类及其子类（如图10.2所示）是异常处理类，使程序更稳定。

图10.2

Java常见的异常

1. RuntimeException子类

序号	异常类型	异常描述
1	java.lang.ArrayIndexOfBoundsException	数组索引越界异常。当对数组的索引值为负数或大于等于数组大小时抛出。
2	java.lang.ArithmeticException	算数条件异常。例如：整数除零等。
3	java.lang.SecurityException	安全性异常
4	java.lang.IllegalArgumentException	非法参数异常
5	java.lang.ArryStoreException	数组中包含不兼容的值抛出的异常
6	java.lang.NegativeArraySizeException	数组长度为负异常
7	java.lang.ClassNotFoundException	找不到类异常。当应用试图根据字符串形式的类型的类名构造类，而在遍历CLASSPATH最后找不到对应名称的Class文件时，抛出该异常
8	java.lang.NullPointerException	空指针异常。当应用试图要求使用对象的地方使用了null时，抛出该异常。

Java常见的异常

2. IOException子类

序号	异常类型	异常描述
1	IOException	操作输入流和输出流时可能出现的异常
2	EOFException	文件已结束异常
3	FileNotFoundException	文件未找到异常

3. 其他子类

序号	异常类型	异常描述
1	ClassCastException	类型转换异常类
2	ArrayStoreException	数组中包含不兼容的值抛出的异常
3	SQLException	操作数据库异常类
4	NoSuchFieldException	字段未找到异常
5	NoSuchMethodException	方法未找到抛出的异常
6	NumberFormatException	字符串转换为数字跑出的异常
7	StringIndexOutOfBoundsException	字符串索引超出范围抛出的异常
8	IllegaAccessException	不允许访问某类异常
9	InstantiationException	当应用程序视图使用Class类中的newInstance()方法创建一个类的实例，而指定的类对象无法被实例化时，抛出该异常

10.3 使用异常处理

10.3.1 try/catch/finally块

```
try{
        //可能出现异常的程序段
}
catch(ExceptionName1 e){
        //异常处理程序段1
}
catch(ExceptionName3 e){
        //异常处理程序段2
}
…….
finally{
        //最后异常处理程序段
}
```

10.3 使用异常处理

try…catch…finally 需要注意以下几点

- try语句之后必须存在catch语句或者finally语句，或者两者同时存在。
- try语句不可以脱离catch语句和finally语句而独立存在。
- 如果try块抛出异常，try块中的异常类型与其中一个catch块的参数类型相匹配，则执行此catch块的代码。
- finally子句一般完成释放资源的任务，作为异常处理的统一出口，可以省略。如果有finally子句，则不管是否发生异常都会执行finally块。

【例10.1】try-finally 捕获异常实例。

```
public class Ex91{
    public static void methodA() {
        try{
            System.out.println("abcd");    //可能出错的语句块
        }
        finally {
            System.out.println("123456");  //最后异常处理程序段
        }
    }
    public static void main(String[] args){
        methodA();
    }
}
```

运行结果:

```
abcd
123456
```

【例10.2】try-catch-finally捕获异常实例

```
import java.io.*;
public class Ex92{
    public static void main(String[] args){
        try{   //可能产生异常的代码段
            FileInputStream in = new
FileInputStream("test.txt");
            System.out.println("in proc try");
        }
        catch(FileNotFoundException e)     //捕获文件没有找
到异常
        {
            System.out.println("in proc catch");
        }
        finally{                          // 最后异常处理程序段
            System.out.println("in proc finally");
        }
    }
}
```

【例10.2】try-catch-finally捕获异常实例

运行结果:

如果存在文件test.txt时将输出:

in proc try

in proc finally

如果文件test.txt文件不存在时将输出:

in proc catch

in proc finally

10.3.2 catch块的顺序

- catch语句中处理异常类型和生成异常对象完全一致；
- catch语句中处理异常类型是生成异常对象的父类。

```
try{              //line1
      //statements
}
catch(Exception  eRef)   //line2
 {
       //statements
 }
 catch(ArithmeticException)  //line3
 {
       //statements
 }
```

```
public class CatchTest{
    static public void methodA() {
        try{
            System.out.println("abcd");
        }
        catch(FileNotFoundException e){}
        }
    public static void main(String[] args){
        methodA();
    }
}
```

上例编译不能通过，编译的错误信息如下：

CatchTest.java:10: exception java.io.FileNotFoundException is never thrown in body of corresponding try statement

}catch(FileNotFoundException e){}

如果将FileNotFoundException替换为Exception就可以编译通过。

10.3.3 抛出异常

如果方法确实引发了异常，那么在方法中必须写明相应的处理代码。

处理异常有两种方法：

- 一种是使用try-catch块捕获所发生的异常，并进行相应的处理。
- 另一种方法，是不在当前方法内处理异常，而是把异常抛出，由调用方法处理。

10.3.3 抛出异常

格式如下：

返回类型 方法名（参数） throws 异常列表

- 关键字throws后是方法内可能发生且不进行处理的所有异常列表，各异常之间以逗号分隔。

```
public void throubleSome() throws IOException
```

- 如果方法引发了异常，而它自己又不处理，就需要由调用方法处理。

10.4 自定义异常

除了使用Java预定义的异常外，用户还可以创建自己的异常。

自定义类必须是Exception类的子类。

```
public class MyException extends Exception{…}
```

在程序中发现异常情况时，程序可以抛出（throw）异常实例，将其放到异常队列中，并激活Java的异常处理机制。

如：

```
throw  new  MyException();
```

【例10.3】自定义异常实例

问题说明：

第8章编写了银行账户处理程序。
在用户进行取钱操作时需要输入银行账户和取钱金额，有可能发生输入错误账号或取钱金额大于余额的异常。发生异常后，程序应该有提示信息，这就需要使用自定义异常。

【例10.3】自定义异常实例

```
class NumberException extends Exception{
        //自定义异常，继承Exception类
        String info;
        public NumberException () {        //不带参数构造方法
                info ="It is a wrong number";
        }
        public NumberException(int number) {  //带参数构造方法
                info="Number "+number+" is not permitted";
        }
        //重写父类toString方法，返回自定义异常内容。
        public String toString(){
                return this.info;
        }
}
```

【例10.3】自定义异常实例

```
class ExceptionDemo {
    static void check(int i, int balance)   throws
    NumberException{
        if(i > balance){
              throw new NumberException();
        }else if ( i==0 ){
              throw new NumberException(i);
        }else{
              System.out.println("exit without exception");
        }
    }
```

【例10.3】自定义异常实例

```
public static void main(String[] args) {
        int balance = 1000;
        try{
            check(300, balance);
        }catch(NumberException e){e.printStackTrace();}
        try{
            check(2000, balance);
        }catch(NumberException e){e.printStackTrace();}
        try{
            check(0, balance);
        }catch(NumberException e){e.printStackTrace();}
    }
}
```

异常处理的优势

- 体现了良好的层次结构，提供了良好的接口。
- 异常处理机制使得处理异常的代码和常规代码分离，减少了代码数量，增强了可读性。

异常处理时遵循的原则

- 在程序内部进行异常的捕获和处理，尽量不要让Java运行时环境来处理异常对象。
- 把异常处理的代码与正常代码分开，简化程序并增加可读性。
- 利用finally语句作为异常处理的统一出口。
- 可以用简单条件测试解决的问题不要用异常控制来解决，以提高程序运行的效率。

异常处理时遵循的原则

- 对异常处理不要分的太细，也不要压制，要充分利用异常的传递。
- 自定义异常类一定要是Throwable的直接或间接子类，一般不用自定义异常类作父类。
- 捕获或声明异常时要选取合适的类型，注意捕获异常的顺序。