第三章
类的方法

目 录

3.1 方法的控制流程

- 方法的控制流程
  - ✓ **Java**程序通过控制语句来控制方法的执行流程
  - ✓ **Java**中的流程控制结构主要有三种
    - › 顺序结构
    - › 选择结构
      - – **if**语句（二路选择结构）
      - – **switch**语句（多路选择结构）
    - › 循环结构
      - – **for**语句
      - – **while**语句
      - – **do-while**语句

请自行复习此部分内容！

3.2 异常处理简介

- 异常处理
  - ✓在进行程序设计时，错误的产生是不可避免的。所谓错误，是在程序运行过程中发生的异常事件，这些事件的发生将阻止程序的正常运行
  - ✓例如：
    - › 打开一个不存在的文件、网络连接中断、操作数越界、装载一个不存在的类等。

3.2 异常处理简介

```
class ExceptionDemo1
{       public static void main( String args[ ] )
        {       int a = 0;
                System.out.println( 5/a );
        }
}
```

```
class ExceptionDemo1
{       public static void main( String args[ ] )
        {       try
                {       int a = 0;
                        System.out.println( 5/a );
                }
                catch(Exception e)
                {       System.out.println("发生了除0的错误");          }
        }
}
```

3.2.1 异常处理的意义

- 异常的基本概念
  - ✓是特殊的运行错误对象
  - ✓是异常类的对象
  - ✓Java中声明了很多异常类，每个异常类都代表了一种运行错误，类中包含了
    - › 该运行错误的信息
    - › 处理错误的方法
  - ✓每当Java程序运行过程中发生一个可识别的运行错误时，即该错误有一个异常类与之相对应时，系统都会产生一个相应的该异常类的对象，即产生一个异常

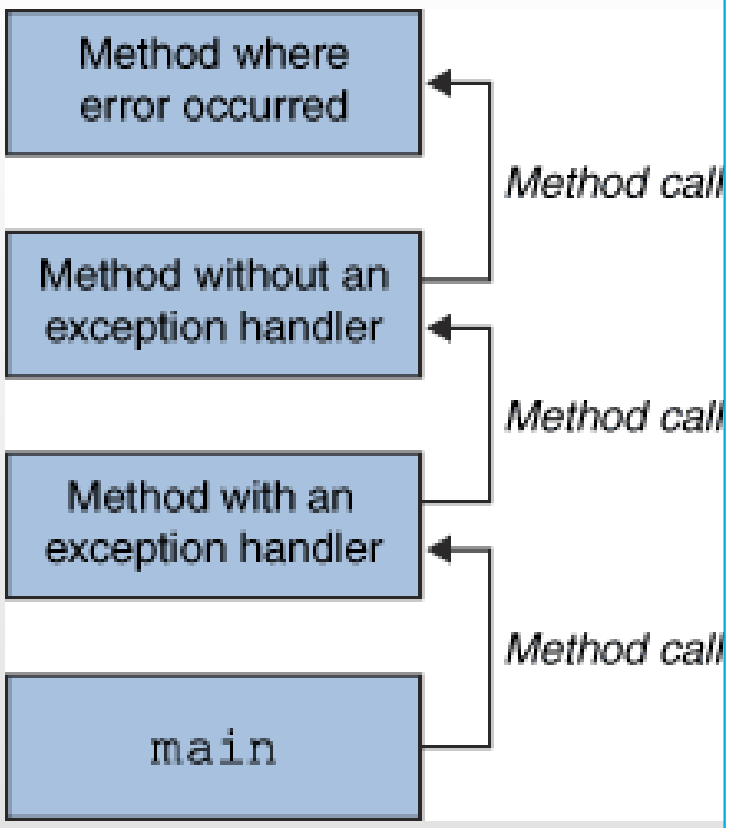
Definition: An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

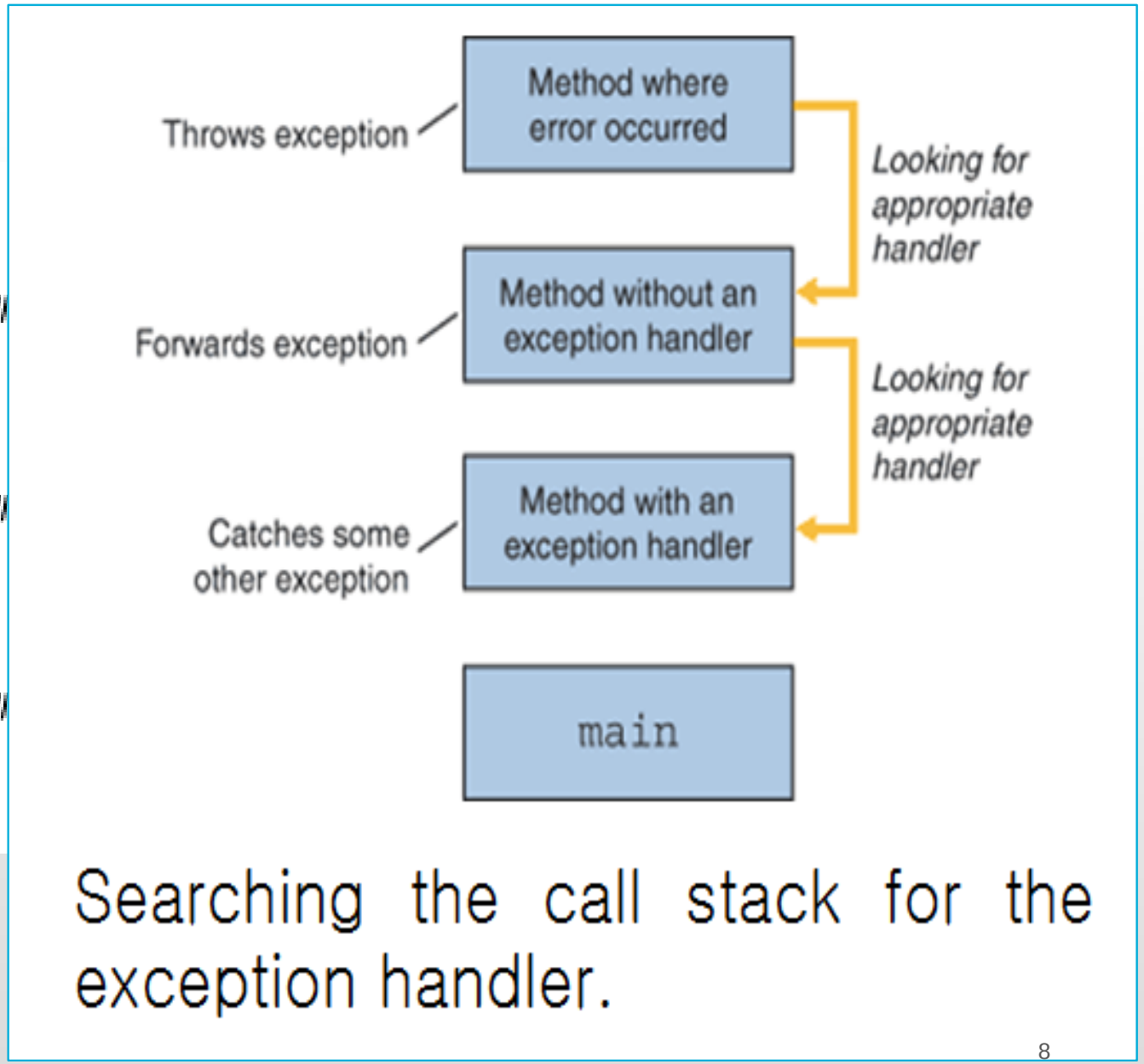
3.2.1 异常处理的意义(续)

- **Java处理错误的方法**
  - ✓抛出(**throw**)异常
    - › 在方法的运行过程中，如果发生了异常，则该方法生成一个代表该异常的对象并把它交给运行时系统，运行时系统便寻找相应的代码来处理这一异常
  - ✓捕获(**catch**)异常　积极的异常处理机制
    - › 运行时系统在方法的调用栈中查找，从生成异常的方法开始进行回溯，直到找到包含相应异常处理的方法为止
    - › 如果**JAVA**运行时系统找不到可以捕获异常的方法，则运行时系统将终止，**JAVA**程序也将退出

3.2.1 异常处理的意义(续)

异常处理示意图

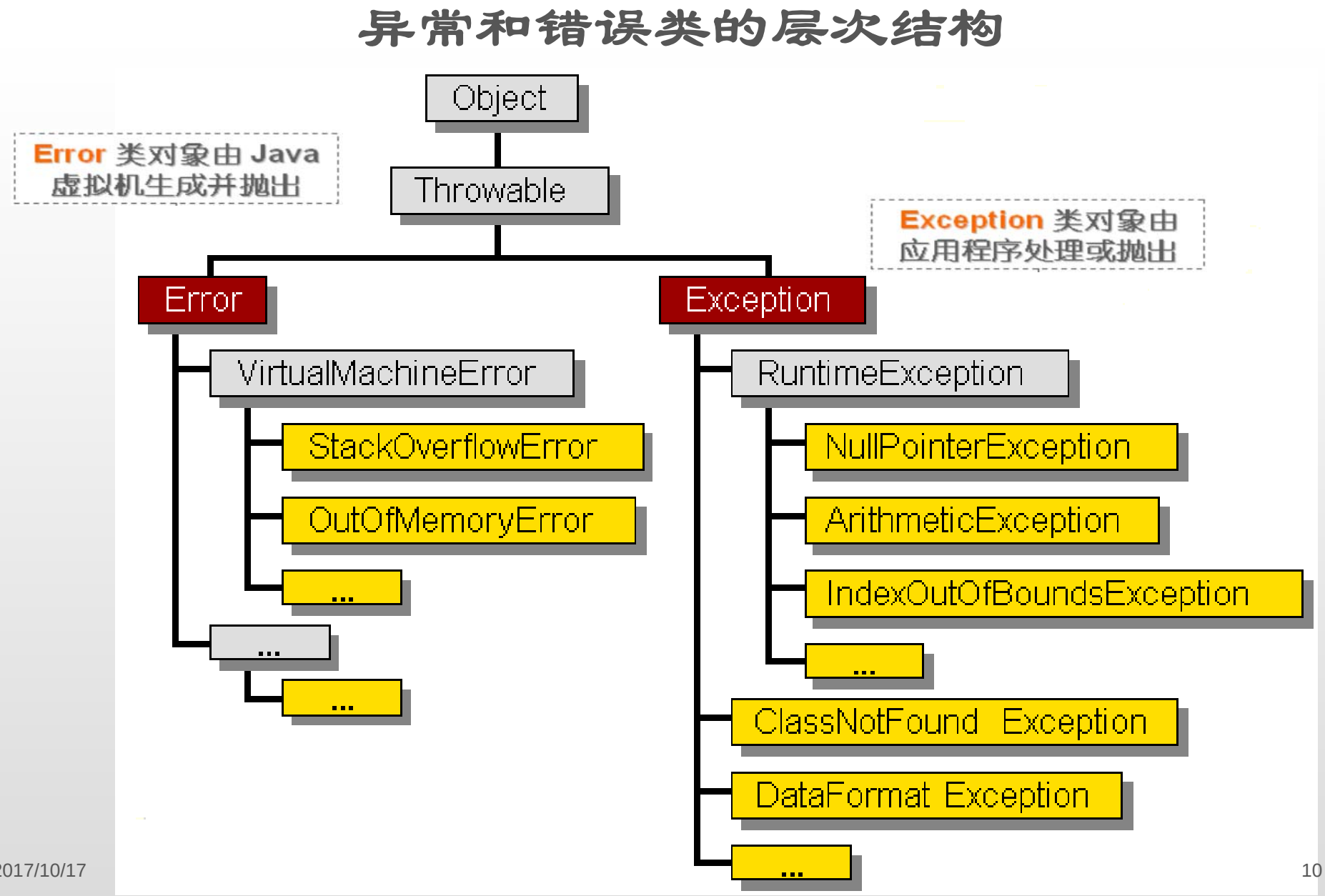




3.2.1 异常处理的意义(续)

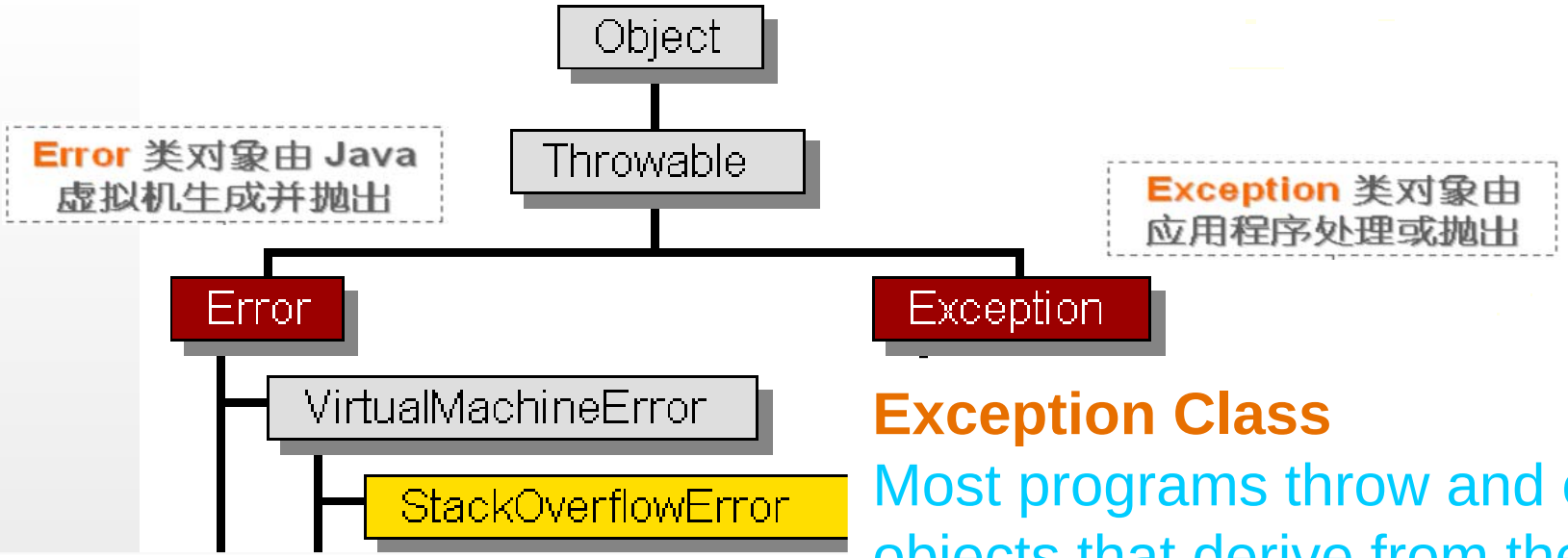
- **Java异常处理机制的优点**
  - ✓将错误处理代码从常规代码中分离出来
  - ✓按错误类型和差别分组
  - ✓对无法预测的错误的捕获和处理
  - ✓克服了传统方法的错误信息有限的问题
  - ✓把错误传播给调用堆栈

3.2.1 异常处理的意义(续)



3.2.1 异常处理的意义(续)

异常和错误类的层次结构



Error Class

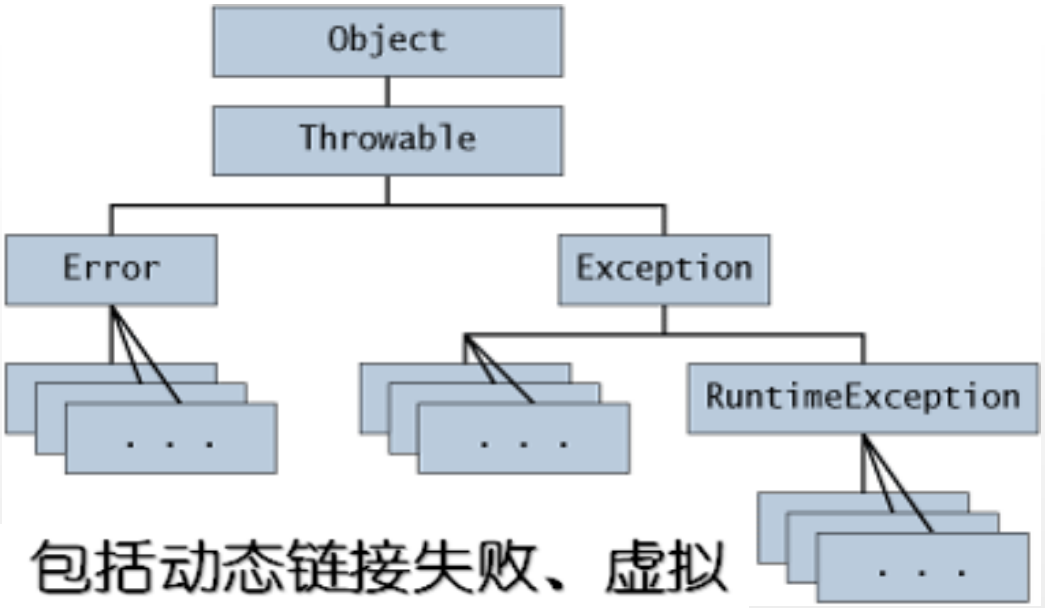
When a dynamic linking failure or other hard failure in the Java virtual machine occurs, the virtual machine throws an Error. Simple programs typically do *not* catch or throw Errors.

Exception Class

Most programs throw and catch objects that derive from the Exception class. An Exception indicates that a problem occurred, but it is not a serious system problem. Most programs you write will throw and catch Exceptions as opposed to Errors.

The Three Kinds of Exceptions

1. checked exception
2. error
3. runtime exception



Error：由**Java虚拟机生成并抛出**，包括动态链接失败、虚拟机错误等，**Java**程序不做处理。

Runtime Exception：**Java**虚拟机在运行时生成的异常，如被**0**除等系统错误、数组下标超范围等，其产生比较频繁，处理麻烦，对程序可读性和运行效率影响太大。

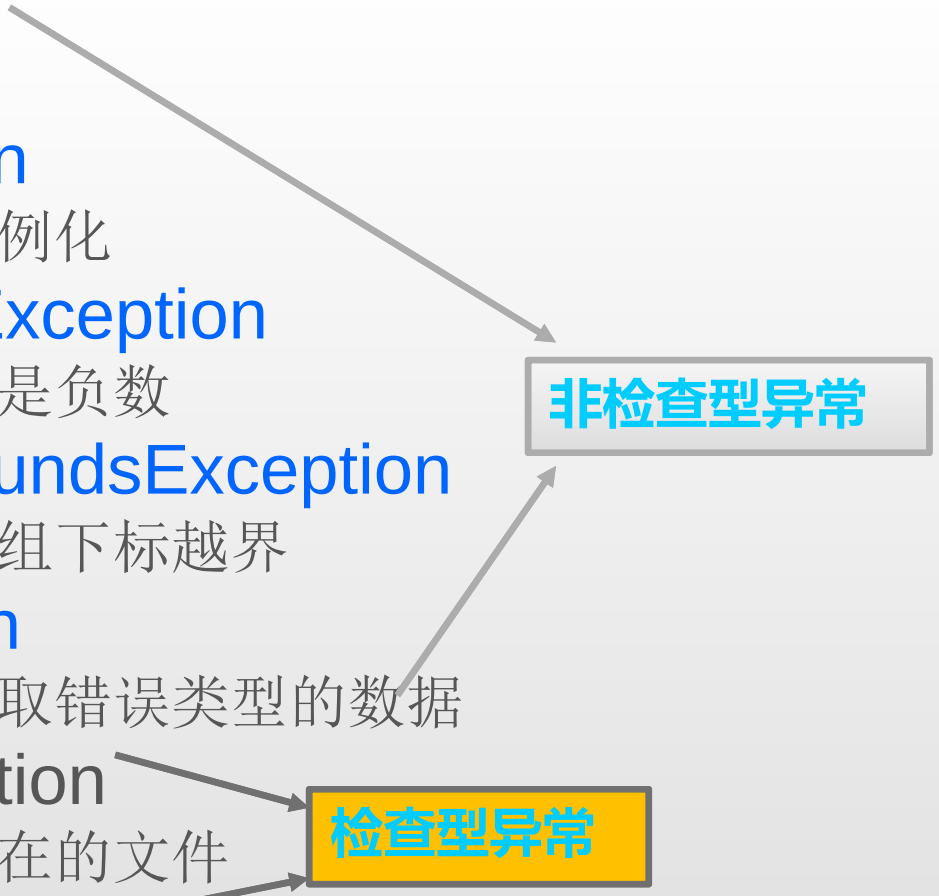
- 由系统检测，用户可不做处理，系统将它们交给**缺省的异常处理程序**，必要时，用户可对其处理。

Exception：一般程序中可预知的问题，其产生的异常可能会带来意想不到的结果，因此**Java**编译器要求**Java**程序必须捕获或声明所有的非运行时异常。

checked exception

3.2.1 异常处理的意义(续)

- **Java预定义的一些常见异常**
  - ✓ ArithmeticException
    - › 整数除法中除数为0
  - ✓ NullPointerException
    - › 访问的对象还没有实例化
  - ✓ NegativeArraySizeException
    - › 创建数组时元素个数是负数
  - ✓ ArrayIndexOutOfBoundsException
    - › 访问数组元素时，数组下标越界
  - ✓ ArrayStoreException
    - › 程序试图向数组中存取错误类型的数据
  - ✓ FileNotFoundException
    - › 试图存取一个并不存在的文件
  - ✓ IOException
    - › 通常的I/O错误



3.2.1 异常处理的意义(续)

- 测试系统定义的运行异常——数组越界出现的异常

```
public class HelloWorld {
    public static void main (String args[ ]) {
      int i = 0;
      String greetings [ ] = {"Hello world!", "No, I mean it!",
                              "HELLO WORLD!!"};
      while (i < 4) {
            System.out.println (greetings[i]);
            i++;
      }
    }
}
```

3.2.1 异常处理的意义(续)

- 运行结果

```
Hello world!
No, I mean it!
HELLO WORLD!!
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException
        at HelloWorld.main(HelloWorld.java:7)
```

- 说明
  - ✓ 访问数组下标越界，导致ArrayIndexOutOfBoundsException异常
  - ✓ 该异常是系统定义好的类，对应系统可识别的错误，所以Java虚拟机会自动中止程序的执行流程，并新建一个该异常类的对象，即抛出数组出界异常

3.2.3 异常的处理

- 对于检查型异常，Java强迫程序必须进行处理。处理方法有两种：
  - ✓(1)声明抛出异常
    - › 不在当前方法内处理异常，而是把异常抛出到调用方法中
  - ✓(2)捕获异常
    - › 调用方法中使用try{}catch(){}块，捕获到所发生的异常，并进行相应的处理

3.2.3 异常的处理

(1)声明抛出异常　消极的异常处理机制

✓如果程序员不想在当前方法内处理异常，可以使用throws子句声明将异常抛出到调用方法中

✓如果所有的方法都选择了抛出此异常，最后JVM将捕获它，输出相关的错误信息，并终止程序的运行。在异常被抛出的过程中，任何方法都可以捕获它并进行相应的处理

3.2.3 异常的处理(续)—一个例子

```
public void openThisFile(String fileName) throws java.io.FileNotFoundException

{
      //code  for method

}

public void getCustomerInfo() throws java.io.FileNotFoundException

{
        // do something
       this.openThisFile("customer.txt");
        // do something
}
```

如果在openThisFile中抛出了FileNotfoundException异常，getCustomerInfo将停止执行，并将此异常传送给它的调用者

3.2.3 异常的处理(续)—一个例子

```
public class TestException
{   public static void main(String [] args)
    {       int reslut = new Test().devide( 3, 1 );
            System.out.println("the result is" + reslut );
    }
}
class Test
{   public int devide(int x, int y) throws Exception
    {       int result = x/y;
            return x/y;
    }
}
```

3.2.3 异常的处理(续)

(2)捕获异常

- 语法格式

```
try{
}
catch (ExceptionType name) {
}
catch (ExceptionType name) {
}
finally{
}
```

3.2.3 异常的处理(续)

- 说明
  - ✓ **try** 语句
    - › 其后跟随可能产生异常的代码块
  - ✓ **catch**语句
    - › 其后跟随异常处理语句，通常用到几个方法
      - – **getMessage()** – 返回一个字符串对发生的异常进行描述。
      - – **toString()**-返回此异常的简短描述
      - – **printStackTrace()** – 给出方法的调用序列，一直到异常的产生位置
  - ✓ **finally**语句
    - › 不论在**try**代码段是否产生异常，**finally** 后的程序代码段都会被执行。通常在这里释放内存以外的其他资源
    - › 例如：读文件后，都需要关闭文件

3.2.3 异常的处理(续)

```
public class DivTest{
   public static void main(String args[]){
   try
    {
       int a=Integer.parseInt(args[0]);
       int b=Integer.parseInt(args[1]);
       int c=a/b;
      System.out.println("您输入的两个数相除的结果是："+c);
     }
     catch( IndexOutOfBoundsException ie)
     {
        System.out.println("数据越界：运行程序时输入的参数个数不够");
      }
     catch( NumberFormatException ne)
     {
        System.out.println("数字格式异常：程序只能接收整数参数");
      }
     catch( ArithmeticException ae)
     {
        System.out.println("算术异常");
      }
     catch( Exception e)
     {
        System.out.println("未知异常");
      }
   }
}
```

注意事项：

在类层次树中，一般的异常类型放在后面，特殊的放在前面

3.2.4 生成异常对象

- 生成异常对象
  - ✓三种方式
    - › 由Java虚拟机生成
    - › 由Java类库中的某些类生成
    - › 在程序中生成自己的异常对象，即异常可以不是出错产生，而是人为地抛出
  - ✓生成异常对象都是通过throw语句实现，生成的异常对象必须是Throwable或其子类的实例
    - › throw new Throwable();
    - › ArithmeticException e = new ArithmeticException();
      throw e;

- 生成异常对象举例

```
class ThrowTest
{
    public static void main(String args[])
    {
      try {  throw new ArithmeticException();
      } catch(ArithmeticException ae){
         System.out.println(ae);
      }
      try {  throw new ArrayIndexOutOfBoundsException();
      } catch(ArrayIndexOutOfBoundsException ai){
         System.out.println(ai);
      }
      try {  throw new StringIndexOutOfBoundsException();
      } catch(StringIndexOutOfBoundsException si){
         System.out.println(si);
      }
    }
}
```

3.2.5 声明自己的异常类

- 声明自己的异常类
  - ✓除使用系统预定义的异常类外，用户还可声明自己的异常类
  - ✓自定义的所有异常类都必须是Exception的子类
  - ✓一般的声明方法如下

```
public class MyExceptionName extends SuperclassOfMyException
{
    public MyExceptionName() {
        super("Some string explaining the exception");
    }
}
```

3.2.5 声明自己的异常类

- 声明当除数为零时抛出的异常类DivideByZeroException

```
public class DivideByZeroException extends ArithmeticException{
    public DivideByZeroException() {
       super("Attempted to divide by zero");
    }
}
```

```
import java.io.*;
public class Examp3_16 {
   private static int quotient(int numerator, int denominator)
                          throws DivideByZeroException {
         if (denominator == 0)  throw new  DivideByZeroException();
         return(numerator / denominator);
   }
```

示 例

```
public class Tran { // 创建类
    // 定义方法，抛出异常
    static int avg(int number1, int number2) throws MyException {
        if (number1 < 0 || number2 < 0) { // 判断方法中参数是否满足指定条件
            throw new MyException("不可以使用负数"); // 错误信息
        }
        if (number1 > 100 || number2 > 100) { // 判断方法中参数是否满足指定条件
            throw new MyException("数值太大了"); // 错误信息
        }
        return (number1 + number2) / 2; // 将参数的平均值返回
    }

    public static void main(String[] args) { // 主方法
        try { // try代码块处理可能出现异常的代码
            int result = avg(102, 150); // 调用avg()方法
            System.out.println(result); // 将avg()方法的返回值输出
        } catch (MyException e) {
            System.out.println(e); // 输出异常信息
        }
    }
}
```

运行结果：
MyException：数值太大了

本章小结

- 本章内容
  - Java程序中类方法的控制结构，包括顺序、分支及循环三种基本结构
  - Java的异常处理机制，包括对错误的分类方法，如何抛出异常、捕获异常
  - 方法的重载
- 本章要求
  - 掌握三种流程控制语法，并熟练应用
  - 了解Java的异常处理机制，会编写相应程序
  - 掌握方法重载的含义，并熟练应用

Arrays类

- ✓ import java.util.Arrays
  - 填充替换数组元素
  - 数组中的元素定义完成后，可通过Arrays类的静态方法fill()来对数组中的元素进行替换。该方法通过各种重载形式可完成任意类型的数组元素的替换。fill()方法有两种参数类型。下面以int型数组为例介绍fill()方法的使用方法。

```
public class Swap { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[5]; // 创建int型数组
        Arrays.fill(arr, 8); // 使用同一个值对数组进行填充
        for (int i = 0; i < arr.length; i++) { // 循环遍历数组中的元素
            // 将数组中的元素依次输出
            System.out.println("第" + i + "个元素是：" + arr[i]);
        }
    }
}
```

fill(int[] a, int value)

Arrays类

- ✓ import java.util.Arrays
  - 填充替换数组元素
  - 数组中的元素定义完成后，可通过Arrays类的静态方法fill()来对数组中的元素进行替换。该方法通过各种重载形式可完成任意类型的数组元素的替换。fill()方法有两种参数类型。下面以int型数组为例介绍fill()方法的使用方法。

```
public class Displace { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 45, 12, 2, 10 }; // 定义并初始化int型数组arr
        Arrays.fill(arr, 1, 2, 8); // 使用fill方法对数组进行初始化
        for (int i = 0; i < arr.length; i++) { // 循环遍历数组中元素
            // 将数组中的每个元素输出
            System.out.println("第" + i + "个元素是：" + arr[i]);
        }
    }
}
```

fill(int[] a, int fromIndex, int toIndex, int value)

Arrays类

- ✓ import java.util.Arrays
  - 对数组进行排序
  - 通过Arrays类的静态sort()方法可实现对数组排序，sort()方法提供了许多种重载形式，可对任意类型数组进行升序排序。

```
public class Taxis { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 23, 42, 12, 8 }; // 声明数组
        Arrays.sort(arr); // 将数组进行排序
        for (int i = 0; i < arr.length; i++) { // 循环遍历排序后的数组
            System.out.println(arr[i]); // 将排序后数组中的各个元素输出
        }
    }
}
```

Arrays.sort(object)

Arrays类

- ✓ import java.util.Arrays
  - 复制数组
  - Arrarys类的copyOf()方法与copyOfRange()方法可实现对数组的复制。copyOf()方法是复制数组至指定长度，copyOfRange()方法则将指定数组的指定长度复制到一个新数组中。

```
public class Cope { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 23, 42, 12, }; // 定义数组
        int newarr[] = Arrays.copyOf(arr, 5); // 复制数组arr
        for (int i = 0; i < newarr.length; i++) { // 循环变量复制后的新数组
            System.out.println(newarr[i]); // 将新数组输出
        }
    }
}
```

copyOf(arr, int newlength)

Arrays类

- ✓ import java.util.Arrays
  - 复制数组
  - Arrarys类的copyOf()方法与copyOfRange()方法可实现对数组的复制。copyOf()方法是复制数组至指定长度，copyOfRange()方法则将指定数组的指定长度复制到一个新数组中。

```
public class Repeat { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 23, 42, 12, 84, 10 }; // 定义数组
        int newarr[] = Arrays.copyOfRange(arr, 0, 3); // 复制数组
        for (int i = 0; i < newarr.length; i++) { // 循环遍历复制后的新数组
            System.out.println(newarr[i]); // 将新数组中的每个元素输出
        }
    }
}
```

copyOfRange(arr, int fromIndex, int toIndex)

Arrays类

- ✓ import java.util.Arrays
  - 数组查询
  - Arrays类的binarySearch()方法，可使用二分搜索法来搜索指定数组，以获得指定对象。该方法返回要搜索元素的索引值。binarySearch()方法提供了多种重载形式，用于满足各种类型数组的查找需要。

```
public class Example { // 创建类
    public static void main(String[] args) { // 主方法
        int ia[] = new int[] { 1, 8, 9, 4, 5 }; // 定义int型数组ia
        Arrays.sort(ia); // 将数组进行排序
        int index = Arrays.binarySearch(ia, 4); // 查找数组ia中元素4的索引位置
        System.out.println("4的索引位置是: " + index); // 将索引输出
    }
}
```

binarySearch（Object[].a, Object key）

Arrays类

- ✓ import java.util.Arrays
  - 数组查询
  - Arrays类的binarySearch()方法，可使用二分搜索法来搜索指定数组，以获得指定对象。该方法返回要搜索元素的索引值。binarySearch()方法提供了多种重载形式，用于满足各种类型数组的查找需要。

```
public class Rakel { // 创建类
    public static void main(String[] args) { // 主方法
        // 定义String型数组str
        String str[] = new String[] { "ab", "cd", "ef", "yz" };
        Arrays.sort(str); // 将数组进行排序
        // 在指定的范围内搜索元素"cd"的索引位置
        int index = Arrays.binarySearch(str, 0, 2, "cd");
        System.out.println("cd的索引位置是: " + index); // 将索引输出
    }
}
```

binarySearch（Object[].a, int fromIndex, int toIndex, Object key）

Arrays类

- ✓ import java.util.Arrays
  - 填充替换数组元素
  - 数组中的元素定义完成后，可通过Arrays类的静态方法fill()来对数组中的元素进行替换。该方法通过各种重载形式可完成任意类型的数组元素的替换。fill()方法有两种参数类型。下面以int型数组为例介绍fill()方法的使用方法。

```
public class Swap { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[5]; // 创建int型数组
        Arrays.fill(arr, 8); // 使用同一个值对数组进行填充
        for (int i = 0; i < arr.length; i++) { // 循环遍历数组中的元素
            // 将数组中的元素依次输出
            System.out.println("第" + i + "个元素是：" + arr[i]);
        }
    }
}
```

fill(int[] a, int value)

Arrays类

- ✓ import java.util.Arrays
  - 填充替换数组元素
  - 数组中的元素定义完成后，可通过Arrays类的静态方法fill()来对数组中的元素进行替换。该方法通过各种重载形式可完成任意类型的数组元素的替换。fill()方法有两种参数类型。下面以int型数组为例介绍fill()方法的使用方法。

```
public class Displace { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 45, 12, 2, 10 }; // 定义并初始化int型数组arr
        Arrays.fill(arr, 1, 2, 8); // 使用fill方法对数组进行初始化
        for (int i = 0; i < arr.length; i++) { // 循环遍历数组中元素
            // 将数组中的每个元素输出
            System.out.println("第" + i + "个元素是：" + arr[i]);
        }
    }
}
```

fill(int[] a, int fromIndex, int toIndex, int value)

Arrays类

- ✓ import java.util.Arrays
  - 对数组进行排序
  - 通过Arrays类的静态sort()方法可实现对数组排序，sort()方法提供了许多种重载形式，可对任意类型数组进行升序排序。

```
public class Taxis { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 23, 42, 12, 8 }; // 声明数组
        Arrays.sort(arr); // 将数组进行排序
        for (int i = 0; i < arr.length; i++) { // 循环遍历排序后的数组
            System.out.println(arr[i]); // 将排序后数组中的各个元素输出
        }
    }
}
```

Arrays.sort(object)

Arrays类

- ✓ import java.util.Arrays
  - 复制数组
  - Arrarys类的copyOf()方法与copyOfRange()方法可实现对数组的复制。copyOf()方法是复制数组至指定长度，copyOfRange()方法则将指定数组的指定长度复制到一个新数组中。

```
public class Cope { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 23, 42, 12, }; // 定义数组
        int newarr[] = Arrays.copyOf(arr, 5); // 复制数组arr
        for (int i = 0; i < newarr.length; i++) { // 循环变量复制后的新数组
            System.out.println(newarr[i]); // 将新数组输出
        }
    }
}
```

copyOf(arr, int newlength)

Arrays类

- ✓ import java.util.Arrays
  - 复制数组
  - Arrarys类的copyOf()方法与copyOfRange()方法可实现对数组的复制。copyOf()方法是复制数组至指定长度，copyOfRange()方法则将指定数组的指定长度复制到一个新数组中。

```
public class Repeat { // 创建类
    public static void main(String[] args) { // 主方法
        int arr[] = new int[] { 23, 42, 12, 84, 10 }; // 定义数组
        int newarr[] = Arrays.copyOfRange(arr, 0, 3); // 复制数组
        for (int i = 0; i < newarr.length; i++) { // 循环遍历复制后的新数组
            System.out.println(newarr[i]); // 将新数组中的每个元素输出
        }
    }
}
```

copyOfRange(arr, int fromIndex, int toIndex)

Arrays类

- ✓ import java.util.Arrays
  - 数组查询
  - Arrays类的binarySearch()方法，可使用二分搜索法来搜索指定数组，以获得指定对象。该方法返回要搜索元素的索引值。binarySearch()方法提供了多种重载形式，用于满足各种类型数组的查找需要。

```
public class Example { // 创建类
    public static void main(String[] args) { // 主方法
        int ia[] = new int[] { 1, 8, 9, 4, 5 }; // 定义int型数组ia
        Arrays.sort(ia); // 将数组进行排序
        int index = Arrays.binarySearch(ia, 4); // 查找数组ia中元素4的索引位置
        System.out.println("4的索引位置是: " + index); // 将索引输出
    }
}
```

binarySearch（Object[].a, Object key）

Arrays类

- ✓ import java.util.Arrays
  - 数组查询
  - Arrays类的binarySearch()方法，可使用二分搜索法来搜索指定数组，以获得指定对象。该方法返回要搜索元素的索引值。binarySearch()方法提供了多种重载形式，用于满足各种类型数组的查找需要。

```
public class Rakel { // 创建类
    public static void main(String[] args) { // 主方法
        // 定义String型数组str
        String str[] = new String[] { "ab", "cd", "ef", "yz" };
        Arrays.sort(str); // 将数组进行排序
        // 在指定的范围内搜索元素"cd"的索引位置
        int index = Arrays.binarySearch(str, 0, 2, "cd");
        System.out.println("cd的索引位置是: " + index); // 将索引输出
    }
}
```

binarySearch（Object[].a, int fromIndex, int toIndex, Object key）