讲义07：异常处理

学习目标

- 需要掌握的内容：
  - 理解异常和异常处理及其作用
  - 区别异常的类型：Error/Exception, 必检异常和免检异常
  - 声明、抛出、处理异常
  - 了解何时使用异常
  - 声明自定义异常
- 资料来源：
  - 第9章

1 异常处理的基本概念

- 语法错误
  - 程序中违反了语言的规则.
  - 可以由编译器发现.
- 逻辑错误
  - 程序没有按照预期的方案执行.
  - 通过测试和调试程序来发现和改正.
- 运行错误(教材称为语义错误)
  - 程序运行过程中, 运行环境发现了不可执行的操作.
  - Java通过异常处理来解决运行错误.

1 异常处理的基本概念

- 运行错误会引起异常. 不对异常进行处理会导致程序非正常结束, 并引起严重问题.
- 产生异常的原因有很多:
  - 用户输入无效的值
  - 程序试图打开一个不存在的文件
  - 网络连接断开
  - 试图访问一个越界的数组元素
  - ……
- 当产生异常时, 程序的正常执行流程会中断. Java给程序员处理运行异常的功能, 称为 “异常处理”.

1 异常处理的基本概念

```
5    public static void main(String[] args) {
6        int n = m1(10, 0, 5);
7        System.out.println(n);
8    }
9    public static int m1(int a, int b, int c) {
10       int x = m2(a, b) + c;
11       return x;
12   }
13   public static int m2(int x, int y) {
14       int z = x / y;
15       return z;
16   }
```

异常发生

在发生异常事件处产生一个异常对象，并交给运行环境，称为**异常抛出**。
异常抛出后，从生成异常对象的代码开始，沿着方法调用栈回溯查找，直到找到包含异常处理的方法，称为**异常捕获**。

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at demo.Main.m2(Main.java:14)
        at demo.Main.m1(Main.java:10)
        at demo.Main.main(Main.java:6)
```

代码：examples/lecture07/exception_01

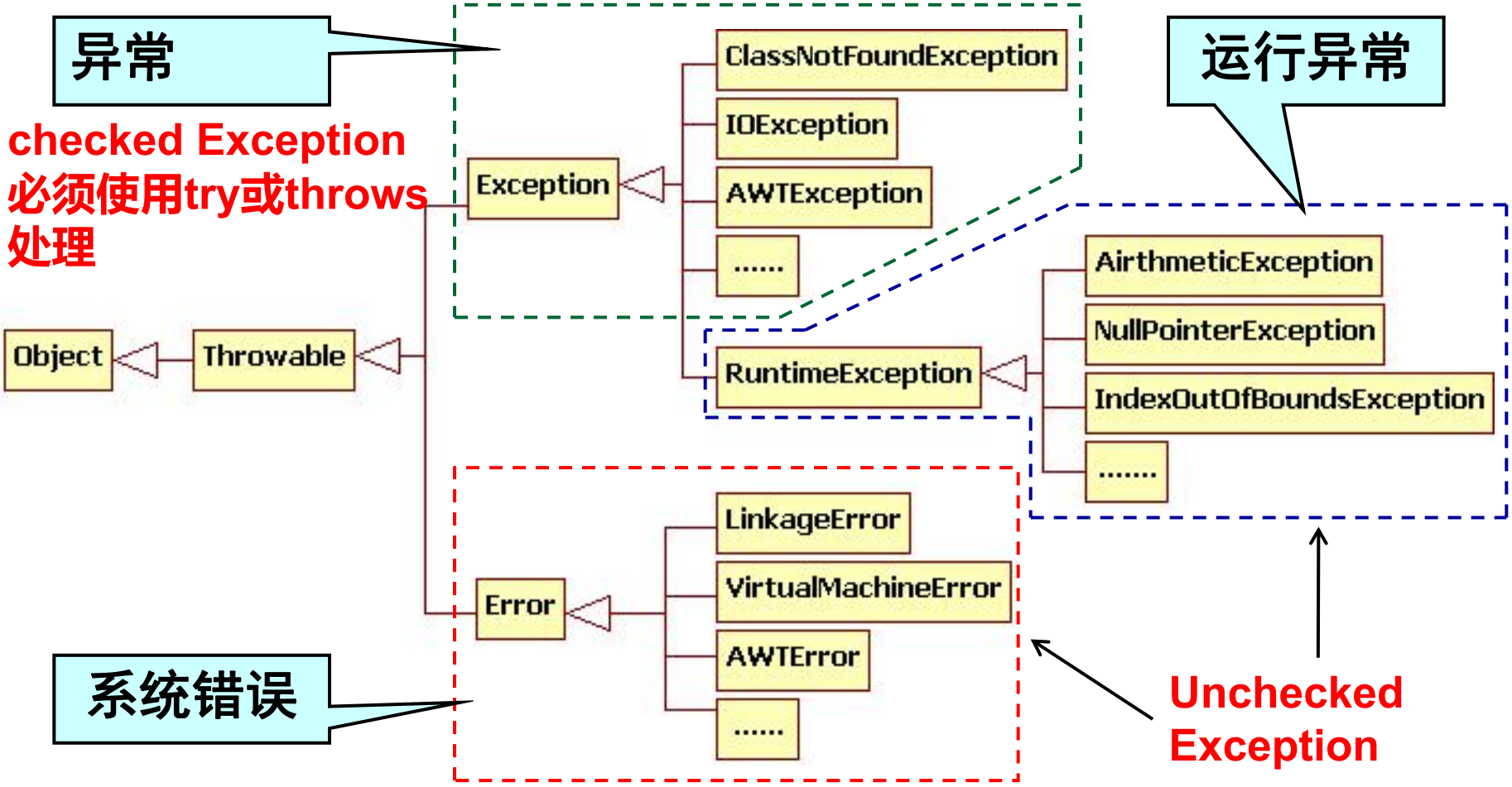
2 异常类

Java异常是java.lang.Throwable的后代类的对象实例.

- 系统错误(System Error)
  - 由JVM抛出并在Error类中描述.
  - 很少发生, 如果发生只能通知用户及尽量稳妥地结束程序.
- 运行异常(Runtime Exception)
  - 由RuntimeException类描述的编程错误,
  - 由JVM抛出, 如: 类型转换失败、数组下标越界、……
- 异常(Exception)
  - 由Exception类描述, 由程序和外部环境引起的错误.
  - 能通过程序捕获和处理.

2 异常类

Java的异常类



2 异常类

异常对象中包含有价值的信息。
Java.lang.Throwable 的如下实例方法用以获取这些信息:

- **public String getMessage()**
  返回异常对象的详细信息
- **public String toString()**
  返回: 异常类名全称 + “:” + getMessage()
- **public void printStackTrace()**
  在控制台输出异常对象及跟踪信息
- **public StackTraceElement[] getStackTrace()**
  返回该异常对象相关的堆栈跟踪元素的数组

3 异常处理

异常处理的基本过程

- 异常的抛出
  程序代码运行中出现了错误，由JRE或程序代码创建并抛出异常对象。
- 异常的捕获和处理
  异常对象在方法调用栈中传递，并使用try-catch语句进行捕获和处理。处理后异常消失。
  如果最终没有捕获和处理则由JRE中止程序运行。

3 异常处理

```
try {
  语句组; //可能抛出异常的语句组
}
catch(Exception1 ex1) {
  异常Exception1的处理代码;
}
……
catch(ExceptionN  exN) {
  异常ExceptionN的处理代码;
}
finally { // finally部分是可选的
}
```

如果try块中某条语句抛出异常, 则跳过try块中剩下的语句.

程序按照catch子句的排列顺序逐个检查其参数类型是否与抛出异常的类型匹配.

如果有匹配的catch子句, 则将异常对象传递给该子句的参数, 并执行其中的语句.

如果没有匹配的catch子句, 则Java退出方法, 把异常对象传递给调用该方法的方法并重复以上过程.

如果最终不能找到合适的处理器, 则终止程序运行并显示错误信息.

任何情况下, finally中的代码一定执行。

3.1 抛出异常

- 程序检查到运行错误后, 创建一个适当类型的异常实例并用throw语句抛出它, 称为 “抛出异常”.
- 例如:

```
IllegalArgumentException ex =
                              new IllegalArgumentException();
throw ex;
```

或者:

```
throw new IllegalArgumentException();
```

- Java系统抛出的异常也是通过这种方式进行的。

3.2 声明异常

- Java方法中如果抛出必检异常(checked exception)而方法没有使用try-catch语句捕获该异常，则该方法都必须说明它可能抛出的必检异常.
- 方法用throws声明其可能抛出的异常, 语法为:

```
public void method()
    throws Exception1, Exception2, … , ExceptionN
```

- 例如:

```
public void method1() throws IOException
```

3.3 捕获和处理异常

示例：method2中可能产生3种必检异常，分析每种异常的处理。

```
main method {
  …
  try {
    …
    invoke method1;
    statement1;
  }
  catch(Exception1 ex1) {
    process ex1;
  }
  statment2;
}
```

```
method1
        throws Exception1{
  …
  try {
    …
    invoke method2;
    statement3;
  }
  catch(Exception2 ex2) {
    process ex2;
  }
  statment4;
}
```

```
method2 throws
    Exception1, Exception2{
  …
  try {
    …
    可能产生异常的语句;
    statement5;
  }
  catch(Exception3 ex3) {
    process ex3;
  }
  statment6;
}
```

3.3 捕获和处理异常

捕获异常编程时注意事项:

- 一个异常父类可以派生若干子类, 捕获异常父类的catch子句可以捕获其所有子类的异常对象.
- catch子句的排列顺序非常重要, 不能将父类的catch子句排列在子类catch子句之前.
- Java强制要求程序员处理必检异常, 如果一个方法中某些语句可能抛出必检异常, 则:
  - 该方法中处理该必检异常, 即使用try-catch语句.
  - 该方法必须声明抛出该异常, 即方法声明中使用throws关键字声明该异常.

3.4 重新抛出异常

- 如果捕获并处理该异常, 则异常对象被清除.此时如果希望继续将异常对象传递给调用该方法的方法, 则需要将异常对象重新抛出. 语法为:

```
try {
  语句;
}
catch(Exception ex) {
  处理异常语句;
  throw ex;  //异常对象的重新抛出
}
```

3.5 异常处理的使用

一个方法出现**必检异常**时，有2种策略进行处理：

(1) 让调用者处理：

```
public void method()
    throws TheException {
    ……
    可能抛出异常的语句;
    ……
}
```

(2) 方法内部处理：

```
public void method()  {
    ……
    try {
       可能抛出异常的语句;
    }
    catch(TheException ex){
       处理异常;
    }
    ……
}
```

3.5 异常处理的使用

- 对于RuntimeException及其子类，尽量不使用try-catch，而是使用检测来避免异常发生。例如：

```
try{
  System.out.println(refVar.toString());
}
catch(NullPointerException e) {
  System.out.println("refVar is null");
}
```

```
if(refVar != null) {
  System.out.println(refVar.toString());
}
else {
  System.out.println("refVar is null");
}
```

```
try{
  a[i] = 100;
}
catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("下标越界");
}
```

```
if(i>=0 &&i<a.length) {
  a[i] = 100;
}
else {
  System.out.println("下标越界");
}
```

4 自定义异常类

- Java提供了很多异常类, 应该尽量使用它们而不要创建自己的异常类.
- 如果遇到预定义的异常类不能恰当描述错误信息时, 可以创建自定义异常类.
- 通常通过继承Exception类或除RuntimeException外的子类来创建自定义的异常类.

4 自定义异常类

```
// RadiusException.java: 描述非法半径的异常类
public class RadiusException extends Exception {
  /** Information to be passed to the handlers */
  private double radius;
  /** Construct an exception */
  public RadiusException(double radius) {
    this.radius = radius;
  }
  /** Return the radius */
  public double getRadius() {
    return radius;
  }
  /** Override the "toString" method */
  public String toString() {
    return "Radius is " + radius;
  }
}
```

5 自动关闭资源的try语句

JDK7之后，提供try-with-resource语句，把打开资源、异常处理和关闭资源集成在一个语句模块中。

```
try(初始化资源对象的语句) {
    使用资源对象的语句
} catch(Exception e) {
    异常捕获和处理
}
```

多个资源对象初始化使用分号;分隔

可以有多个catch，如果初始化和使用资源对象时不会有异常产生，可以省略

例如：

```
try (Scanner scanner = new Scanner(new File("src/data.txt"));
     PrintWriter writer = new PrintWriter("src/output.txt")) {
    // 使用资源对象scanner和writer
} catch (FileNotFoundException e) {
    System.out.println("文件不存在！ ");
}
```

代码：examples/lecture07/exception_02

课后工作

- 结合教材内容，复习课堂讲授内容
- 作业：