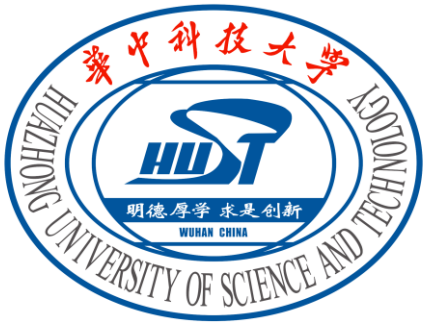
基于Java的面向对象程序设计

陈维亚

weiya_chen@hust.edu.cn

华中科技大学软件学院

第23讲：Java异常处理



目录

1. 概述

❑ 定义

Exception 是 “exceptional event“ 的缩写

Exception是程序运行时发生的异常事件，会中断程序的正常流程，比如IO时找不到文件，访问数组越界等等。

异常处理分离了一般代码和异常处理代码，更易阅读和维护；
对异常的捕获和处理会增加程序的健壮性，使程序不因发生异常而终止。

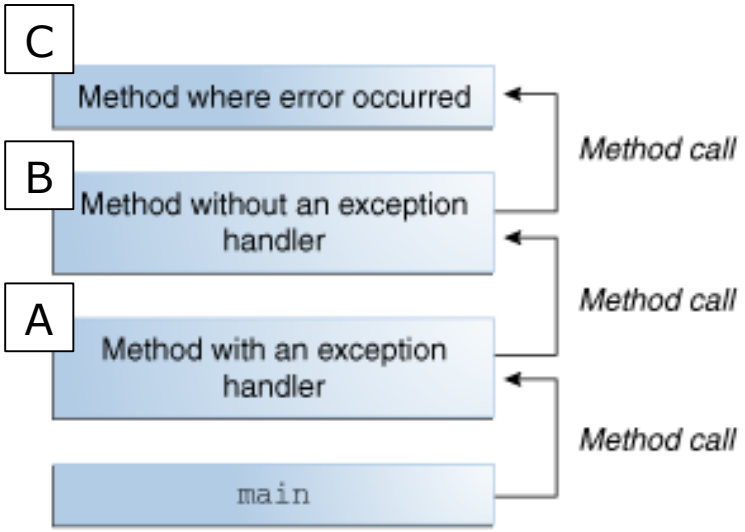
异常可以分为三类：

① **检查性异常**（checked exceptions）：程序正确，由于外在环境条件不足引发。Java编译器强制要求处理这类异常。

② **运行时异常**（runtime exception）：程序存在bug，需修改程序。

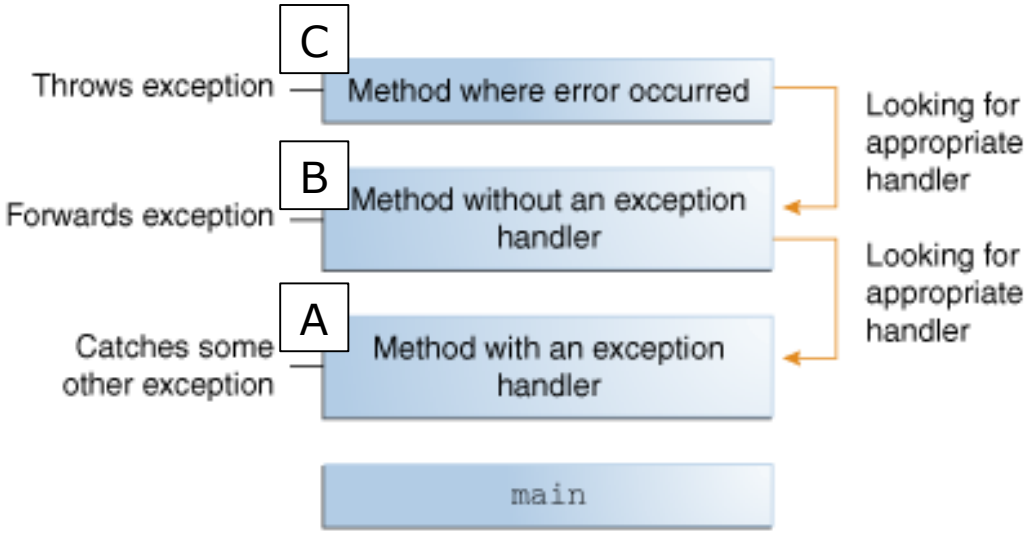
③ **运行时错误**（error）：极少见情况，非程序本身问题。

❑ Java方法调用栈



方法调用栈

❑ Java方法调用栈



方法调用栈

1. 概述

❑ 异常处理机制

Catch or Specify Requirement

会抛出某个异常的方法必须满足以上要求，即以下两条要求之一：

- 有一个try语句来捕获该异常
- 某个方法声明其抛出该异常

检查性异常必须被处理，可由产生异常的代码块自行处理，也可丢给别人处理

1. 概述

```
public class ListOfNumbers {
    private List<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() {
         // FileWriter 的构造函数会抛出 IOException.
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));

        for (int i = 0; i <= SIZE; i++) {
            // get(int) 方法会抛出 IndexOutOfBoundsException.
            out.println("Value at: " + i + " = " + list.get(i));
        }
        out.close();
    }
}
```

1. 概述

- try 语句

```
try {

    code with exception

}
catch and finally blocks
```

```
public void writeList() {
    PrintWriter out = null;

    try{
        System.out.println("Entered try statement");
        out = new PrintWriter(new FileWriter("OutFile.txt"));

        for (int i = 0; i <= SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
    }
    catch and finally blocks
}
```

1. 概述

- catch 语句

```
try {
    code with exception
} catch (ExceptionType name){
    …
} catch (ExceptionType name){
    …
}
```

```
try{
    System.out.println("Entered try statement");
    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("IndexOutOfBoundsException: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
```

1. 概述

- catch 语句

```
try {
    code with exception
} catch (ExceptionType name){
    …
} catch (ExceptionType name){
    …
}
```

JDK 7 之后，一个catch块可以处理多个类型的异常

```
catch (IOException | SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

➢ finally 语句

writeList 的 try 语句块可能以3种方式退出：

1）new FileWriter 语句失败并抛出一个IOException

2）list.get(i) 语句失败并抛出一个IndexOutOfBoundsException

3）无异常抛出，try 语句块正常退出

```
public void writeList() {
     PrintWriter out = null;

     try{
          System.out.println("Entered try statement");
          out = new PrintWriter(new FileWriter("OutFile.txt"));

          for (int i = 0; i <= SIZE; i++) {
              out.println("Value at: " + i + " = " + list.get(i));
          }
     }
     …
}
```

➢ finally 语句

当退出try语句块时，finally块包含总是被执行的语句，除非当前线程或整个程序意外停止。

```
try {
    code with exception
} catch (ExceptionType name){
    …
} finally {
    …
}
```

```
try{
    …
} catch(){
    …
} finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

try-catch-finally 组合

```
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 {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        } else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

- 情景一 - try 语句块执行失败并抛出异常

```
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 (IndexOutOfBoundsException e) {
        System.err.println("Caught IndexOutOfBoundsException: "
                           +  e.getMessage());
    } catch (IOException e) {
        System.err.println("Caught IOException: " +  e.getMessage());
    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        } else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

```
Caught IOException: OutFile.txt
PrintWriter not open
```

❑ 情景二 - try 语句块正常执行

Closing PrintWriter

```
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 {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        } else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

练习 1

一段程序中要进行对数组的访问和除法操作，请对该段程序中可能出现的异常进行处理。

```
public double average(int[] array) {
    double result = 0.0;
    for (int i = 0; i < array.length; i++) {
        result += array[i];
    }
    return result/array.length;
}
```

❑ 标明异常 throws

用 throws 关键字指出可能出现的异常类型，自己不进行处理，由该函数的调用者处理

```
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();
}
```

❑ 标明异常 throws

用 throws 关键字指出可能出现的异常类型，自己不进行处理，由该函数的调用者处理

```
public void writeList() throws IOException, IndexOutOfBoundsException {
    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 void writeList() throws IOException {
```

2. 异常类

- 抛异常 throw

public class **Throwable**
extends Object
implements Serializable

throw someThrowableObject;

```
public Object pop() {
    Object obj;

    if (size == 0) {
        throw new EmptyStackException();
    }

    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

2. 异常类

- 抛异常 throw

public class **Throwable**
extends Object
implements Serializable

throw someThrowableObject;

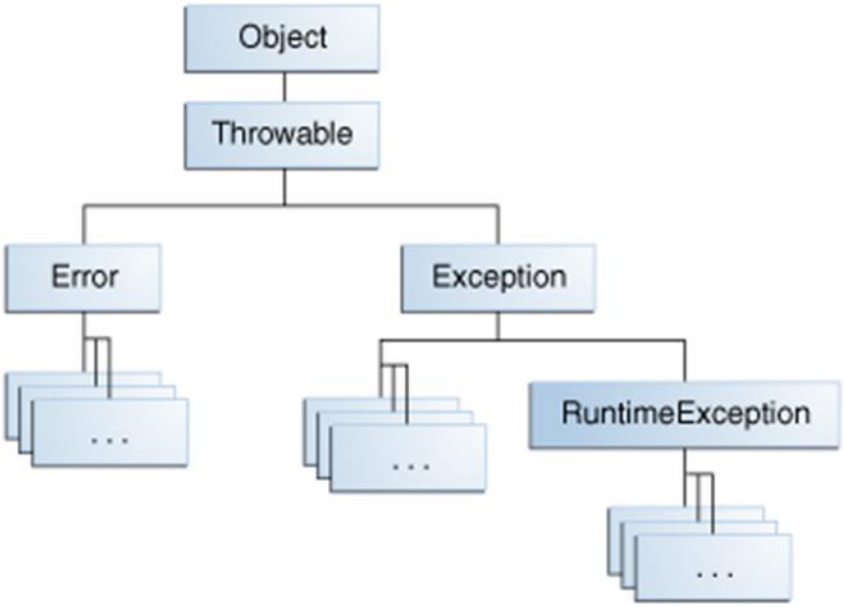
```
public Object pop() throws EmptyStackException{
    Object obj;

    if (size == 0) {
        throw new EmptyStackException();
    }

    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

2. 异常类

❑ Throwable



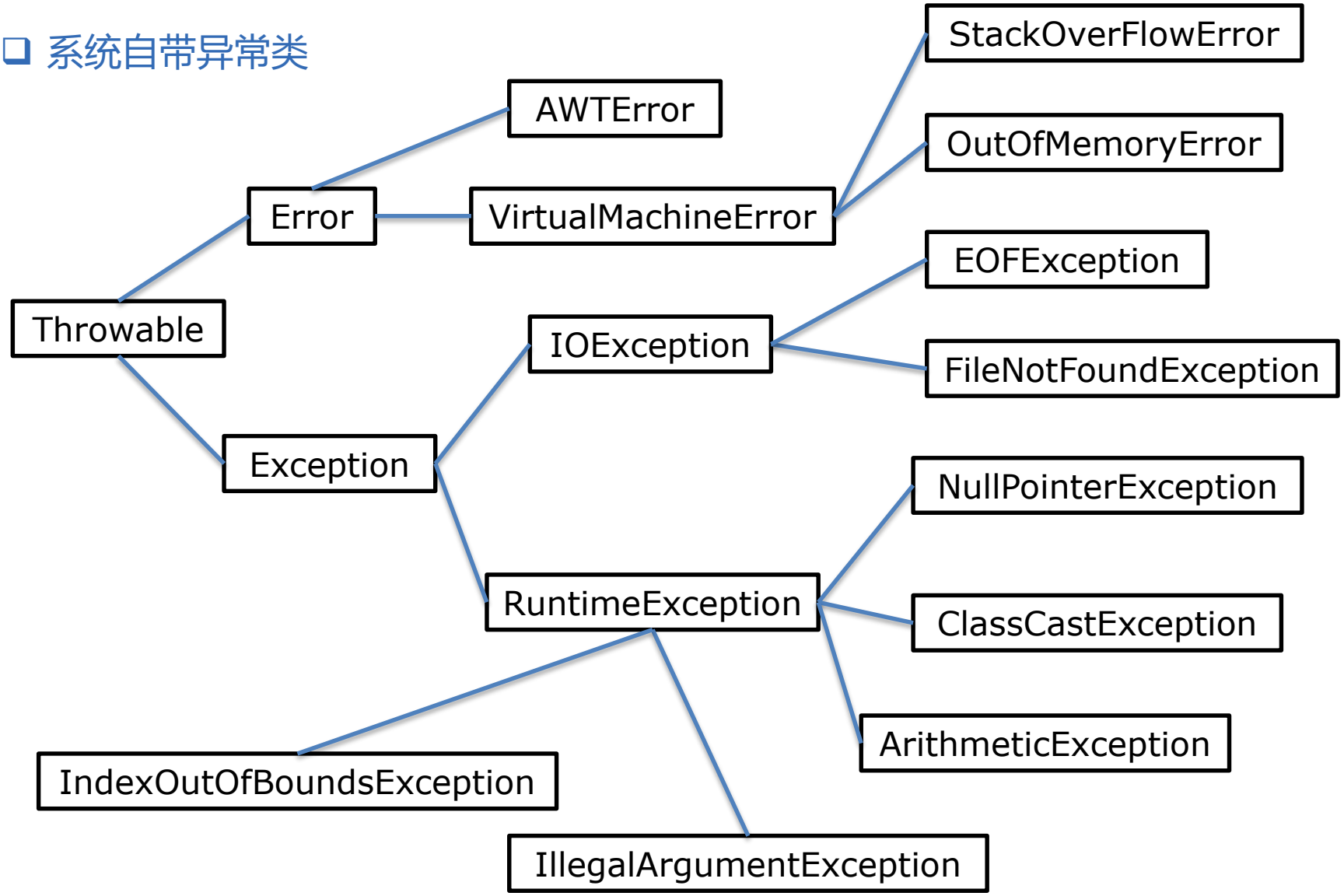
① **检查性异常**（checked exceptions）：程序正确，由于外在环境条件不足引发。Java编译器强制要求处理这类异常。

② **运行时异常**（RuntimeException）：程序存在bug，需修改程序。

③ **运行时错误**（Error）：极少见情况，非程序本身问题。

2. 异常类

- 系统自带异常类



2. 异常类

❑ 用户定义异常类

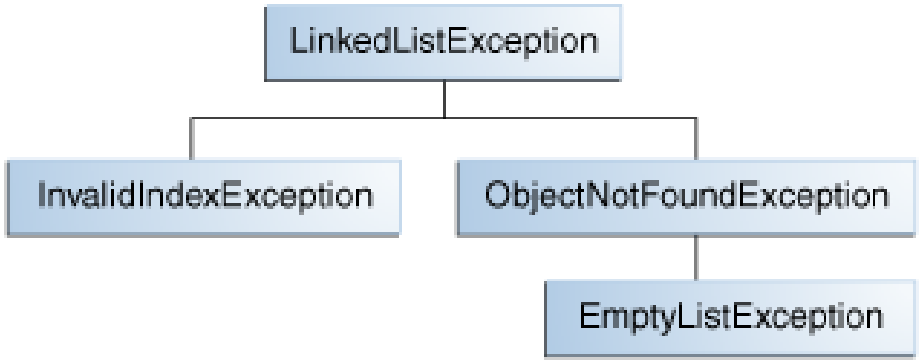
- 你需要定义的异常类型无法由Java自带类型代表吗？
- 用户能从你定义的异常类型中获得更多的信息吗？
- 你的代码会抛出多个异常吗？
- 你是否可以接受使用其他人（第三方）定义的异常类？

❑ 用户定义异常类 – 举例

实现一个自定义的链表数组类MyLinkedList，包含如下方法：

- objectAt(int n) - 如果n<0或者n超过了当前数组长度，抛出异常
- firstObject() - 如果数组为空，抛出异常
- indexOf(Object o) – 如果o不在数组中，抛出异常

请为MyLinkedList类定义相应的异常类。



2. 异常类

❑ 用户定义异常类 – 举例

Throwable 类中定义的方法：

- getMessage()：获得详细的异常信息
- toString()：获得异常的简短描述
- printStackTrace()：打印异常发生处堆栈跟踪信息，包括类名、方法名和所在行数

```
public class InvalidIndexException extends LinkedListException {
     private int n;
     public InvalidIndexException(int n){
          this.n = n;
     }
}
```

```
if(n<0 || n>=list.size())
     throw new InvalidIndexException(n);
```

3. 异常处理原则

➢ 异常处理的语法规则

1. try语句块不能单独存在，catch语句块可以有一个或多个，finally语句块最多一个；
2. try-catch-finally均不能单独使用；
3. 有多个catch块时，Java会按顺序匹配，如找到匹配则不会继续执行后面的catch；
4. throw语句后的语句不会被执行。

➢ 异常处理原则

1. 避免过大的try语句块；
2. 细化异常的类型，不要都用Exception；
3. 自己能处理的异常不要抛给别人；
4. 不要用try-catch实现控制流程的跳转；
5. 尽可能重用已经存在的异常类。

练习 2

1）以下语句是否合法？

```
try {

} finally {

}
```

2）以下语句是否有问题？

```
try {

} catch (Exception e) {

} catch (ArithmeticException a) {

}
```

3）请填空：

a. int[] A;
A[0] = 0;

b. JVM 找不到系统库

c. 一个程序读取输入流时遇到了结束符

d. 在关闭流之前已经读到了结束符，再次尝试读取流中的数据

b 1. __error

d 2. __checked exception

a 3. __compile error

c 4. __no exception

练习 3

修改cat方法使其可以正确编译。

```
public static void cat(File file) {
    RandomAccessFile input = null;
    String line = null;

    try {
        input = new RandomAccessFile(file, "r");
        while ((line = input.readLine()) != null) {
            System.out.println(line);
        }
        return;
    } finally {
        if (input != null) {
            input.close();
        }
    }
}
```

练习 3

修改cat方法使其可以正确编译。

```
public static void cat(File file) {
    RandomAccessFile input = null;
    String line = null;

    try {
        input = new RandomAccessFile(file, "r");
        while ((line = input.readLine()) != null) {
            System.out.println(line);
        }
        return;
    } catch(FileNotFoundException fnf) {
        System.err.format("File: %s not found%n", file);
    } catch(IOException e) {
        System.err.println(e.toString());
    } finally {
        if (input != null) {
            try {
                input.close();
            } catch(IOException io) {
            }
        }
    }
}
```

总结

- try, catch, finally blocks 组合
- 异常类的分类，Throwable
- throws 和 throw
- 自定义异常类

下节预告

Java 线程