异常处理

宿宝臣 <subaochen@126.com>

山东理工大学

October 27, 2020

概述

从一个例子说起

- 输出两个参数的商:

```
public static void main(String[] args) {
    int a = Integer.parseInt(args[0]);
    int b = Integer.parseInt(args[1]);
    System.out.println("a/b = " +  a/b);
}
```

Let’s try:步步惊心

- 没有给出足够数量的参数
- 参数格式不合法:不能转换成整数
- 除数为0

Let’s try:步步惊心

- 没有给出足够数量的参数
- 参数格式不合法:不能转换成整数
- 除数为0

Let’s try:步步惊心

- 没有给出足够数量的参数
- 参数格式不合法:不能转换成整数
- 除数为0

C style解决方案

```
public static void main(String[] args) {
  if(args.length != 2){
    System.out.println("请提供2个参数!");
    return;
  }
  if(args[1].equalsIgnoreCase("0")){
    System.out.println("除数不能为0!");
    return;
  }
  int a = Integer.parseInt(args[0]);
  int b = Integer.parseInt(args[1]);
  System.out.println("a/b = " +  a/b);
}
```

完整示例参见:DivTestCLike.java

C style解决方案有什么问题吗?

- any problems?
- 语法正确,语义明确,思路清晰,考虑周全
- 值得赞赏
- 但是,Java可以让生活更完美,更轻松

C style解决方案有什么问题吗?

- any problems?
- 语法正确,语义明确,思路清晰,考虑周全
- 值得赞赏
- 但是,Java可以让生活更完美,更轻松

C style解决方案有什么问题吗?

- any problems?
- 语法正确,语义明确,思路清晰,考虑周全
- 值得赞赏
- 但是,Java可以让生活更完美,更轻松

C style解决方案有什么问题吗?

- any problems?
- 语法正确,语义明确,思路清晰,考虑周全
- 值得赞赏
- 但是,Java可以让生活更完美,更轻松

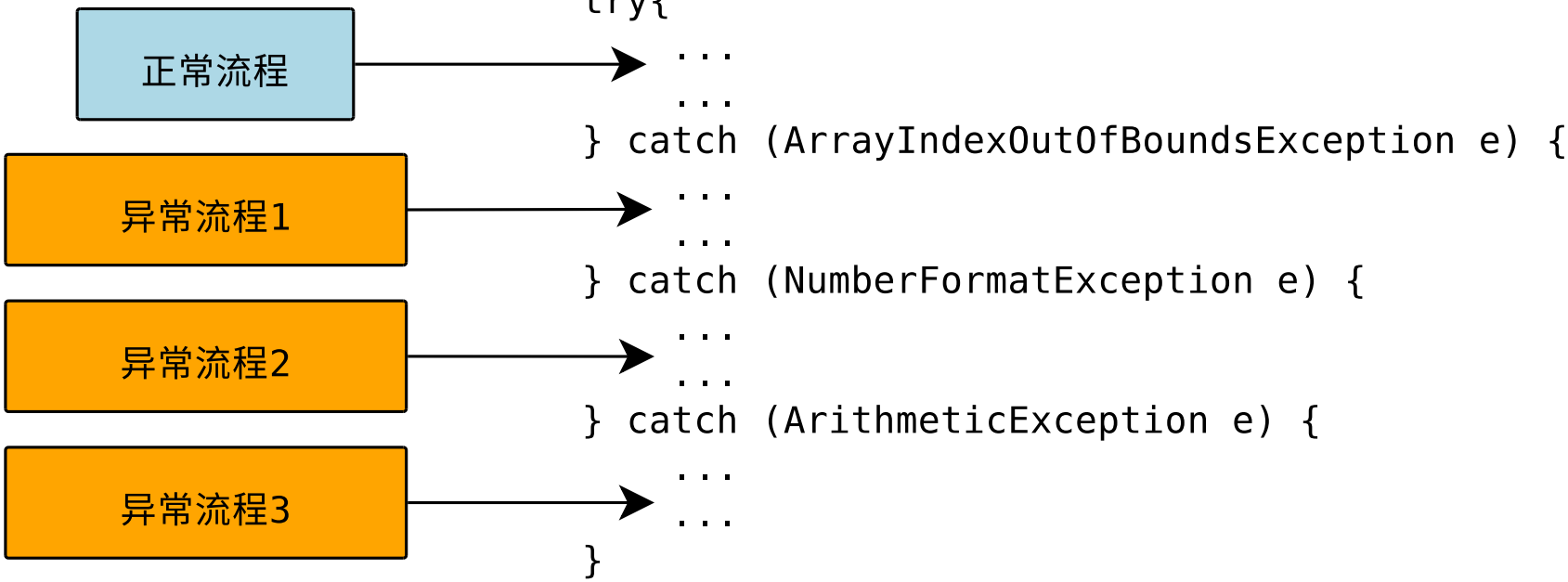
Java style解决方案

```
public static void main(String[] args) {
    try {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        System.out.println("a/b = " + a / b);
    }
```

Java style解决方案(续)

```
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("数组越界，运行程序时输入的
                参数个数不对。应该输入2个参数，您输入的参数
                个数是：" + args.length);
            //e.printStackTrace();
        } catch (NumberFormatException e) {
            System.out.println("数字格式异常，
                程序只能接受整数形式的参数");
            //e.printStackTrace();
        } catch (ArithmeticException e) {
            System.out.println("数学异常，除数不能为0");
            //e.printStackTrace();
        } catch (Exception e) {
            System.out.println("天知道发生了什么，
                总之情况不对");
            //e.printStackTrace();
        }
    }
```

Java异常处理的标准模板



敲重点

- 合理规划正常流程和异常流程
- 正常流程不一定能够完全执行：
  一旦遇到异常即转入异常流程，异常流程执行完毕后退出程序。
- 只有一个异常流程会被执行到
- catch块的最后无需return语句

敲重点

- 合理规划正常流程和异常流程
- 正常流程不一定能够完全执行：
  一旦遇到异常即转入异常流程，异常流程执行完毕后退出程序。
- 只有一个异常流程会被执行到
- catch块的最后无需return语句

敲重点

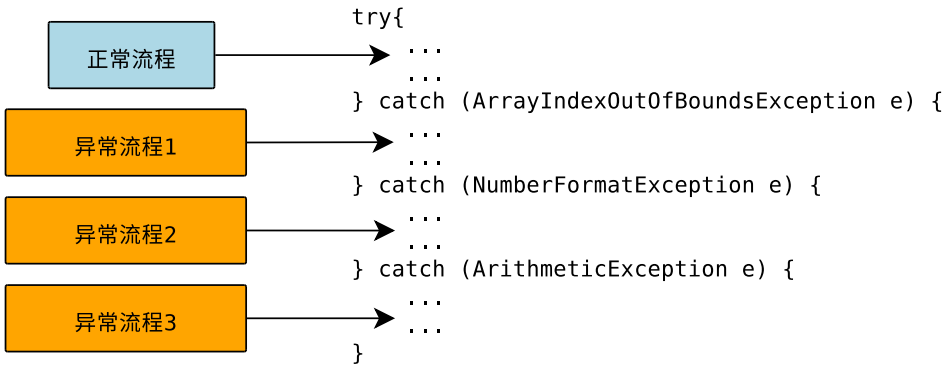
- 合理规划正常流程和异常流程
- 正常流程不一定能够完全执行：
  一旦遇到异常即转入异常流程，异常流程执行完毕后退出程序。
- 只有一个异常流程会被执行到
- catch块的最后无需return语句

敲重点

- 合理规划正常流程和异常流程
- 正常流程不一定能够完全执行：
  一旦遇到异常即转入异常流程，异常流程执行完毕后退出程序。
- 只有一个异常流程会被执行到
- catch块的最后无需return语句

C style vs. Java style

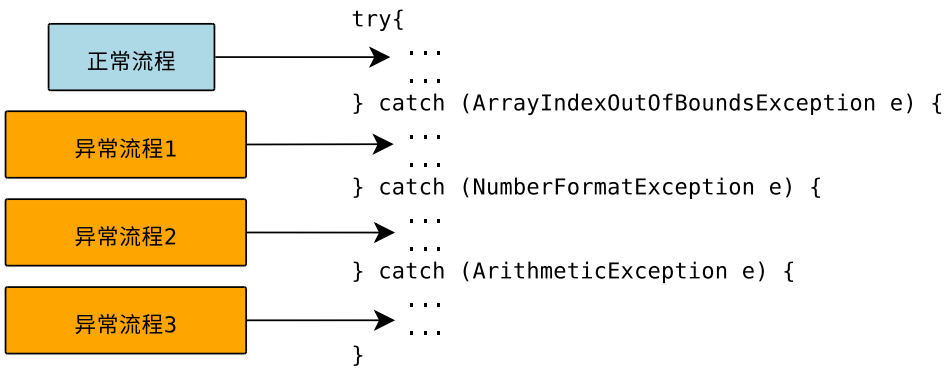
- 层次更清晰,很容易辨识主流程和异常流程



- 异常对象化,可以更好的描述异常流程和现象

C style vs. Java style

- 层次更清晰,很容易辨识主流程和异常流程

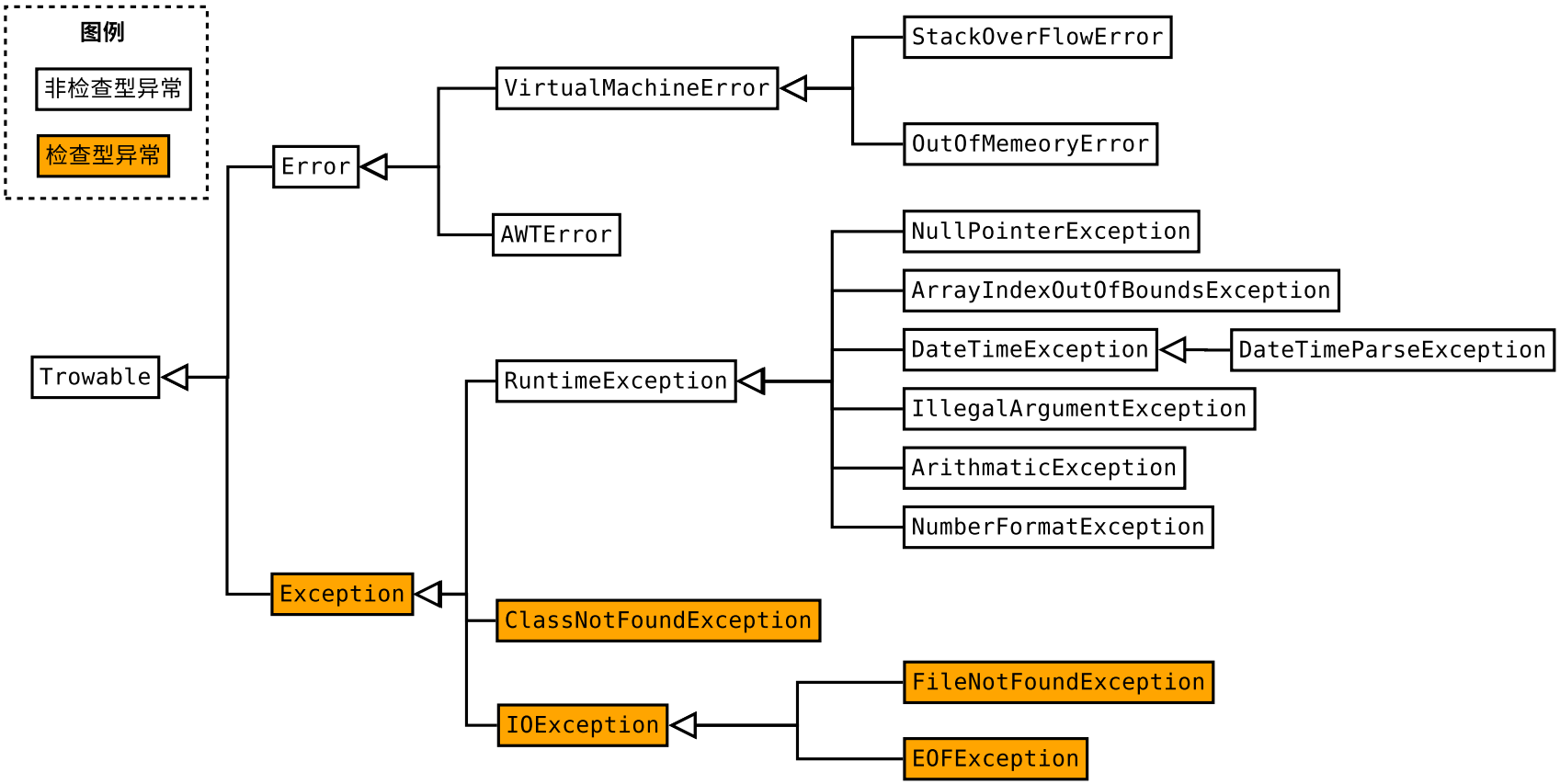


- 异常对象化,可以更好的描述异常流程和现象

异常对象从哪里来？

- Java API中定义了常见的异常类
- 用户自定义异常类

Java API内置的异常类



两种异常类

非检查型异常

JVM知道如何处理此类异常，
可自动处置，无需捕获（catch）；
也可以捕获（catch）此类异常，给出更明确的提示信息

检查型异常

必须捕获（catch），否则为语法错误

用户自定义异常

- 定义方法：extends Exception
- 定义原则：依业务而定
- 如果所有的异常都叫Exception？
- 示例：SomeException

用户自定义异常全部是检查型异常！

So,必须捕获所有的用户自定义异常

用户自定义异常

- 定义方法：extends Exception
- 定义原则：依业务而定
- 如果所有的异常都叫Exception？
- 示例：SomeException

用户自定义异常全部是检查型异常！

So,必须捕获所有的用户自定义异常

用户自定义异常

- 定义方法：extends Exception
- 定义原则：依业务而定
- 如果所有的异常都叫Exception？
- 示例：SomeException

用户自定义异常全部是检查型异常！

So,必须捕获所有的用户自定义异常

用户自定义异常

- 定义方法：extends Exception
- 定义原则：依业务而定
- 如果所有的异常都叫Exception？
- 示例：SomeException

用户自定义异常全部是检查型异常！

So,必须捕获所有的用户自定义异常

用户自定义异常

- 定义方法：extends Exception
- 定义原则：依业务而定
- 如果所有的异常都叫Exception？
- 示例：SomeException

用户自定义异常全部是检查型异常！

So,必须捕获所有的用户自定义异常

如何以及为何抛出异常：throw/throws

- catch的参数：异常对象从哪里来？全部来自于throw new...
- 如何throw？配合if语句
- 为什么要使用throws？
- 如何/在哪里catch？责任链机制

如何以及为何抛出异常：throw/throws

- catch的参数：异常对象从哪里来？全部来自于throw new...
- 如何throw？配合if语句
- 为什么要使用throws？
- 如何/在哪里catch？责任链机制

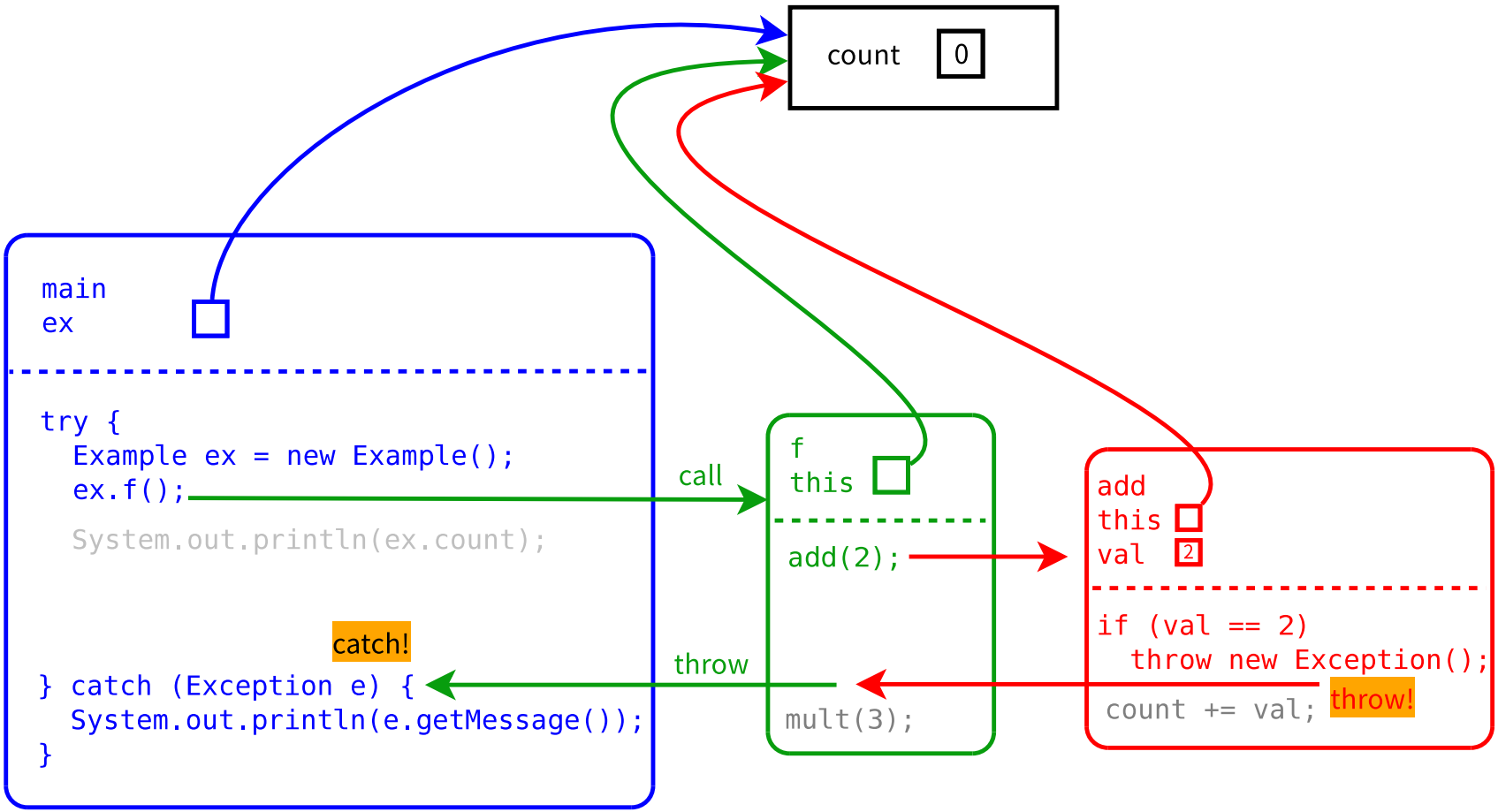
如何以及为何抛出异常：throw/throws

- catch的参数：异常对象从哪里来？全部来自于throw new...
- 如何throw？配合if语句
- 为什么要使用throws？
- 如何/在哪里catch？责任链机制

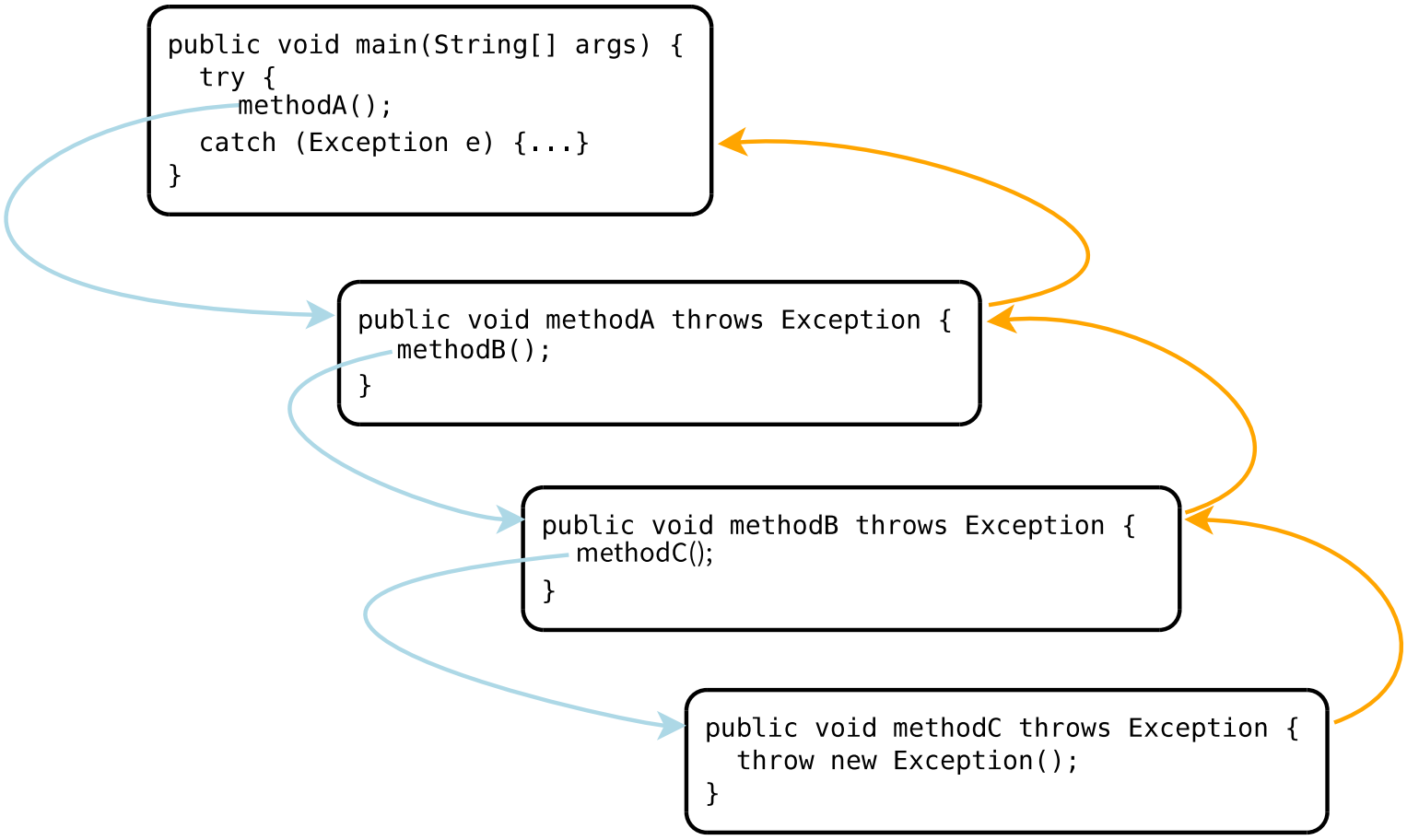
如何以及为何抛出异常：throw/throws

- catch的参数：异常对象从哪里来？全部来自于throw new...
- 如何throw？配合if语句
- 为什么要使用throws？
- 如何/在哪里catch？责任链机制

throw示例


count
0
main
ex
try {
Example ex = new Example();
ex.f();
System.out.println(ex.count);
} catch (Exception e) {
System.out.println(e.getMessage());
}
catch!
call
throw
f
this
add(2);
mult(3);
add
this
val
2
if (val == 2)
throw new Exception();
count += val;
throw!

异常处理的责任链机制



原则

领导不易，最好各负其责，不要将异常处理的责任都推到上面来

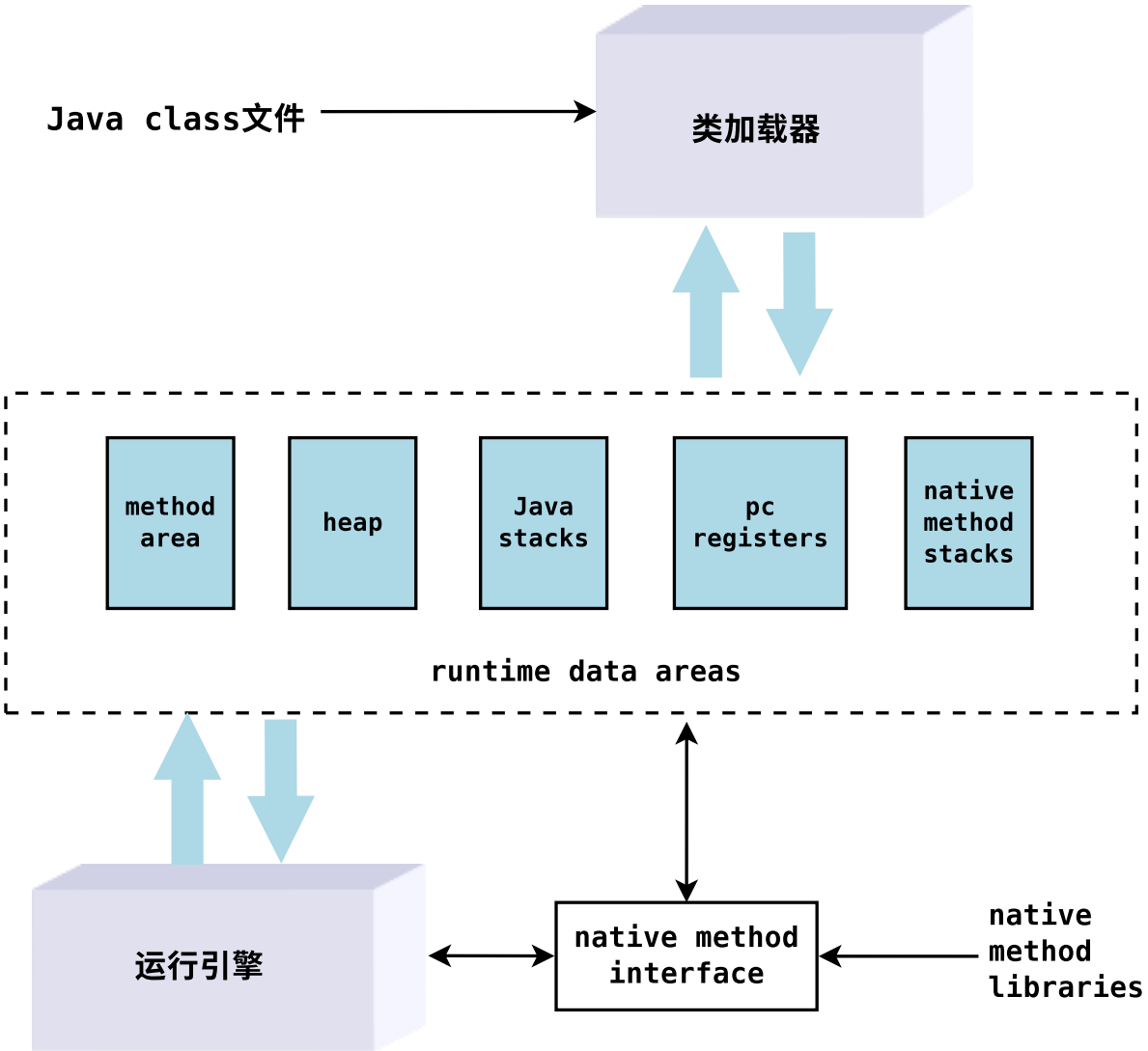
异常处理太麻烦了!

- 神助攻：IDE帮助很大
- Idea快捷键：Alt + Enter
- 注意控制好正常流程的粒度

练习

- 从键盘输入日期字符串，转换为LocalDate对象并打印
  - DateTimeParseException
  - 体会“流程”定义的粒度是如何影响代码的？
- 真的需要捕获所有异常吗？
  - 重写DivTest.java
  - 不捕获任何异常
  - 只捕获ArrayIndexOutofBoundsException

回顾Java运行机制


Java class文件
类加载器
method area
heap
Java stacks
pc registers
native method stacks
runtime data areas
运行引擎
native method interface
native method libraries

最后的清理:finally

- 常和数据库操作、网络操作等有关
- 总是被执行