哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

12 Construction for Robustness and Correctness
面向正确性与健壮性的软件构造

Wang Zhongjie
rainy@hit.edu.cn

June 23, 2021

Outline

- **What are Robustness and Correctness?**
- **How to measure Robustness and Correctness?**

- **Error and Exception in Java**
- **Exception Handling**
- **Assertions**
- **Defensive Programming**
- **The SpotBugs tool**
- **Summary**

进入软件构造最关键的质量特性——健壮性和正确性。

使用错误处理和exception提高robustness
使用断言、防御式编程提高correctness

Reading

- 代码整洁之道：第7章
- Java编程思想：第12章
- Effective Java：第9章
- MIT 6.031：09
- CMU 17-214：Oct 1
- 代码大全：第8章

1 What are Robustness and Correctness?

Robustness 健壮性

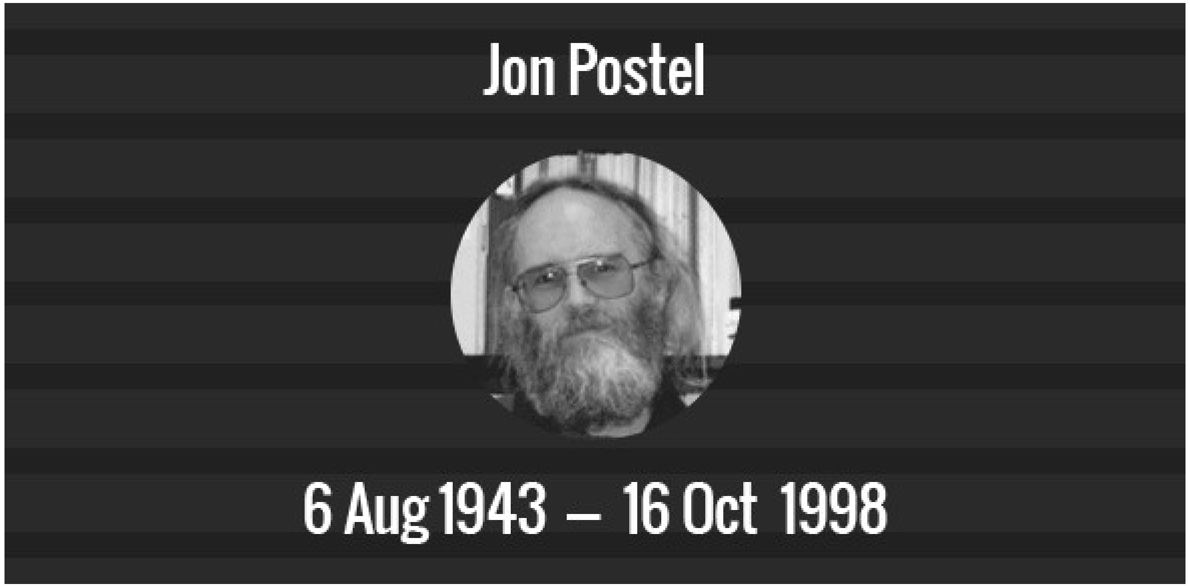
- **Robustness:** “the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions“ (IEEE Std 610.12-1990) 健壮性：系统在不正常输入或不正常外部环境下仍能够表现正常的程度
- **Robust programming** 面向健壮性的编程
  - A style of programming that focuses on handling unexpected termination and unexpected actions. 处理未期望的行为和错误终止
  - It requires code to handle these terminations and actions gracefully by displaying accurate and unambiguous error messages. 即使终止执行，也要准确/无歧义的向用户展示全面的错误信息
  - These error messages allow the user to more easily debug the program. 错误信息有助于进行debug

Robustness principle (Postel’s Law)

- **Paranoia (偏执狂) –** A programmer assumes users are out to break their code, and assumes that his own written code may fail or work incorrectly. 总是假定用户恶意、假定自己的代码可能失败
- **Stupidity -** The programmer assumes users will try incorrect, bogus and malformed inputs. 把用户想象成白痴，可能输入任何东西
  - As a consequence, the programmer returns to the user an unambiguous, intuitive error message that does not require looking up error codes.
  - The error message should try to be as accurate as possible without being misleading to the user, so that the problem can be fixed with ease. 返回给用户的错误提示信息要详细、准确、无歧义
- **Robustness principle (Postel’s Law):** 对别人宽容点，对自己狠一点
  - Be conservative in what you do; be liberal in what you accept from others.
  - “[illegible]”

对自己的代码要保守，对用户的行为要开放

Robustness principle (Postel’s Law)



Be conservative in what you do, be liberal in what you accept from others
(often reworded as "Be conservative in what you send, be liberal in what you accept").

Principles of robust programming

- **Dangerous implements** - Users should not gain access to libraries, data structures, or pointers to data structures. 封闭实现细节，限定用户的恶意行为
  - This information should be hidden from the user so that the user doesn't accidentally modify them and introduce a bug in the code.
  - When such interfaces are correctly built, users use them without finding loopholes to modify the interface.
  - The user therefore focuses solely on his or her own code.

- **Can‘t happen** - Code is modified and may introduce a possibility that an “impossible” case occurs. 考虑极端情况，没有“不可能”
  - Impossible cases are therefore assumed to be highly unlikely instead.
  - The developer thinks about how to handle the case that is highly unlikely, and implements the handling accordingly.

Correctness 正确性

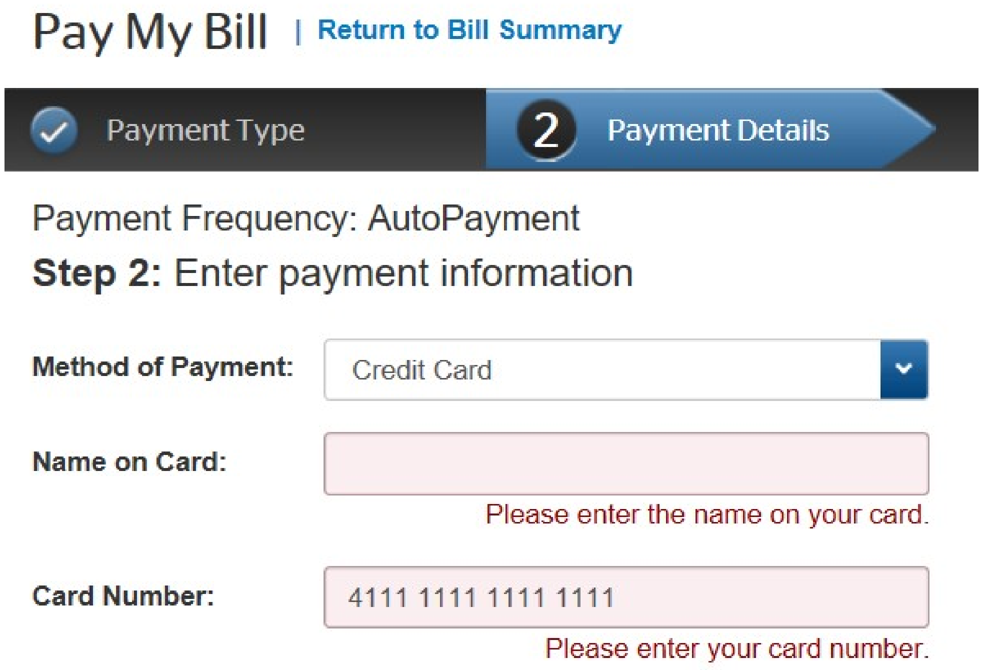
- **Correctness** is defined as the software's ability to perform according to its specification. 正确性：程序按照**spec**加以执行的能力，是最重要的质量指标！
- **Robustness vs. correctness: at opposite ends of the scale.**
  - ***Correctness*** means never returning an inaccurate result; no result is better than an inaccurate result. 正确性：永不给用户错误的结果
  - ***Robustness*** means always trying to do something that will allow the software to keep operating, even if that leads to results that are inaccurate sometimes. 健壮性：尽可能保持软件运行而不是总是退出
- **Robustness adds built-in tolerance for common and non-critical mistakes, while correctness throws an error when it encounters anything less than perfect input.**
  - 正确性倾向于直接报错(error)，健壮性则倾向于容错(fault-tolerance)

Comparison of Robustness and Correctness

Problem	Robust approach	Correct approach
A rogue web browser that adds trailing whitespace to HTTP headers. 浏览器发出包含空格的URL	Strip whitespace, process request as normal.	Return HTTP 400 Bad Request error status to client.
A video file with corrupt frames. 视频文件有坏帧	Skip over corrupt area to next playable section.	Stop playback, raise “Corrupt video file” error.
A config file with lines commented out using the wrong character. 配置文件使用了非法字符	Internally recognize most common comment prefixes, ignore them.	Terminate on startup with “bad configuration” error.
A user who enters dates in a strange format. 奇怪格式的日期输入	Try parsing the string against a number of different date formats. Render the correct format back to the user.	Invalid date error.

"No Dashes Or Spaces"

- Credit card numbers are always printed and read aloud in groups of four digits.
- Computers are pretty good at text processing, so wasting the user's time by forcing them to retype their credit card numbers in strange formats is pure laziness on behalf of developer.
- In Google Maps, you can enter just about *anything* in the search box and it will figure out a street address.
- 健壮性：避免给用户太大压力，帮助用户承担一些麻烦



Pay My Bill | Return to Bill Summary

✓ Payment Type | 2 Payment Details

Payment Frequency: AutoPayment

Step 2: Enter payment information

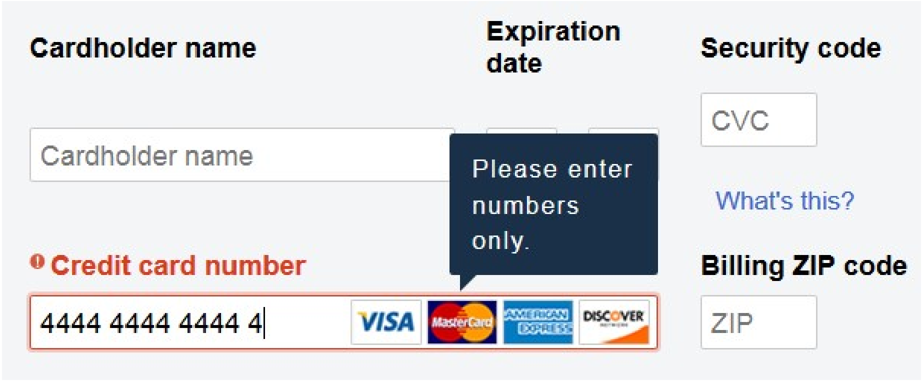
Method of Payment: Credit Card

Name on Card:

Please enter the name on your card.

Card Number: 4111 1111 1111 1111

Please enter your card number.



Cardholder name | Expiration date | Security code

Cardholder name

CVC

What's this?

Please enter numbers only.

Credit card number

4444 4444 4444 4

Billing ZIP code

ZIP

Comparison of Robustness and Correctness

- **Robustness makes life easier for users and third-party developers.**
  - By building in a bit of well thought-out flexibility, it’s going to grant a second chance for users with clients that aren’t quite compliant, instead of kicking them out cold.
- **Correctness makes life easier for your developers.**
  - Instead of bogging down checking/fixing parameters and working around strange edge cases, they can focus on the one single model where all assumptions are guaranteed.
  - Any states outside the main success path can thus be ignored—producing code that is briefer, easier to understand, and easier to

健壮性：
让用户变得更容易：出错也可以容忍，程序内部已有容错机制

正确性：
让开发者变得更容易：用户输入错误，直接结束。
（不满足precondition的调用）

Comparison of Robustness and Correctness

Internally, seek correctness;
Externally, seek robustness.

- **Externally and Internally:**
  - **External interfaces (UI, input files, configuration, API etc)** exist primarily to serve users and third parties. Make them robust, and as accommodating as possible, with the expectation that people will input garbage.
  - An application's **internal model** (i.e. domain model) should be as simple as possible, and always be in a 100% valid state. Use invariants and assertions to make safe assumptions, and just throw a big fat exception whenever you encounter anything that isn't right.
  - Protect the internal model from external interfaces with an anti-corruption layer which maps and corrects invalid input where possible, before passing it to the internal model.在内外部之间做好隔离，防止“错误”扩散
- **Make your external interfaces robust, and make your internal model correct** 对外的接口，倾向于健壮；对内的实现，倾向于正确
  - If you ignore users' needs, no one will want to use your software.
  - If you ignore programmers' needs, there won't *be* any software.

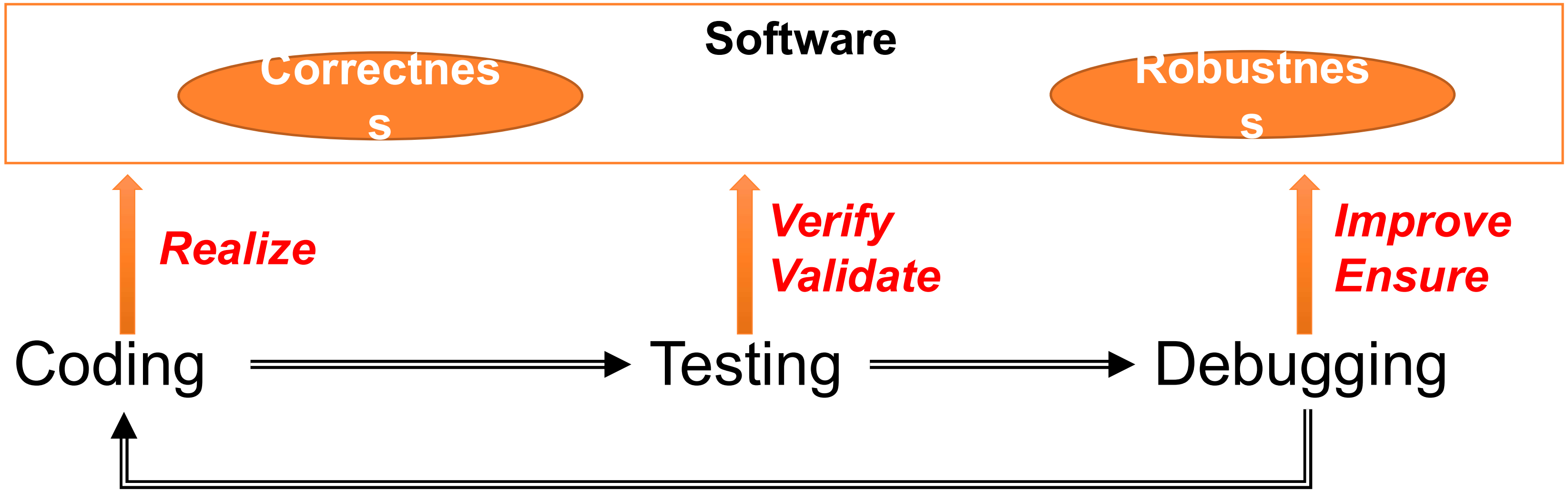
Comparison of Robustness and Correctness

- **Safety critical applications tend to favor correctness to robustness.**
  - It is better to return no result than to return a wrong result.
- **Consumer applications tend to favor robustness to correctness.**
  - Any result what so ever is usually better than the software shutting down.
- **Reliability (可靠性)**. The ability of a system to perform its required functions under stated conditions whenever required—having a long mean time between failures.
- **Reliability = Robustness + Correctness**

Steps for improving robustness and correctness

- **Step 0: To program code with robustness and correctness objectives using assertions, defensive programing, code review, formal validation, etc**
- **Step 1: To observe failure symptoms (Memory dump, stack traces, execution logs, testing)**
- **Step 2: To identify potential fault (bug localization, debug)**
- **Step 3: To fix errors (code revision)**

Techniques for Correctness and Robustness



Software

Correctness

Robustness

Realize

Verify
Validate

Improve
Ensure

Coding

Testing

Debugging

How to program with high correctness & robustness?

Assertion
Exception
Error Handling
Defensive Programming

How to judge whether a software is correct and robust or not?

Design of Test Case
Unit Testing & Integration Testing
Black-box testing & White-box Testing
Test-First Programming

How to locate and fix defects？

Techniques for Debugging
(Reproduce,Diagnose and Fix)
Debugging tools
(Printing/tracing/Logging
Debugger)

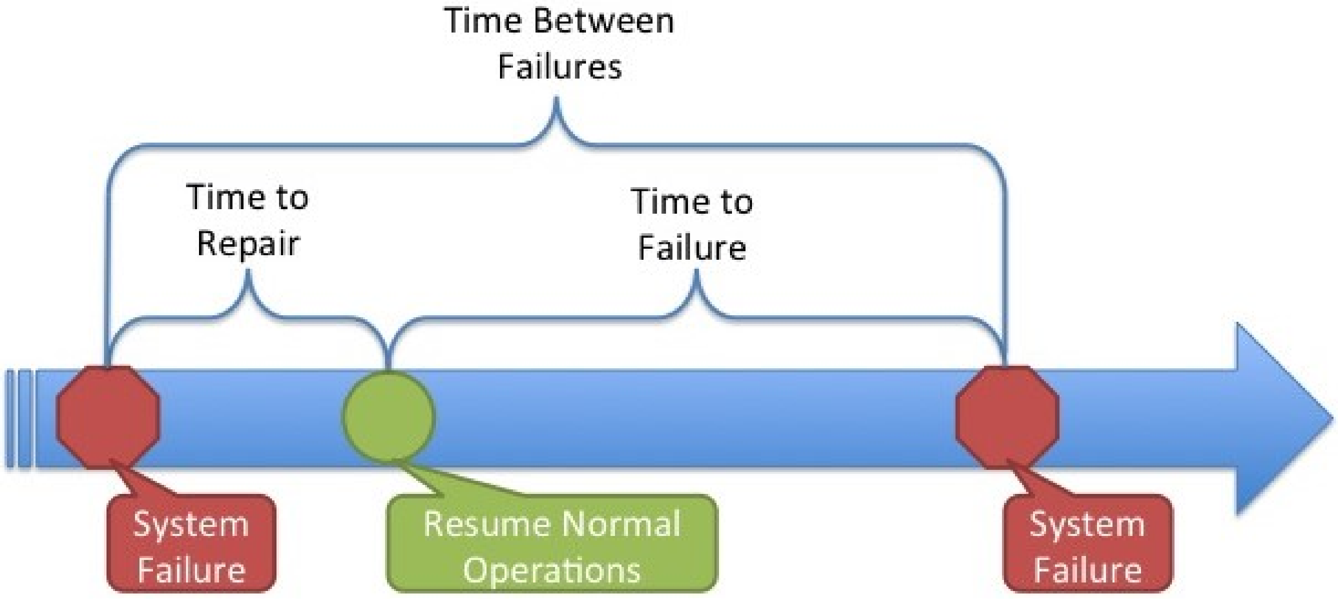
2 How to measure robustness and correctness?

Mean time between failures (MTBF) 外部观察角度

- **Mean time between failures (MTBF，平均失效间隔时间)** is the predicted elapsed time between inherent failures of a system during operation.
  - MTBF is calculated as the arithmetic mean (average) time between failures of a system.
- **The definition of MTBF depends on the definition of what is considered a system failure.**
  - For complex, repairable systems, failures are considered to be those out of design conditions which place the system out of service and into a state for repair.
  - Failures which occur that can be left or maintained in an unrepaired condition, and do not place the system out of service, are not considered failures under this definition.

Mean time between failures (MTBF)

- **Mean time between failures (MTBF)** describes the expected time between two failures **for a repairable system**, while **mean time to failure (MTTF)** denotes the expected time to failure for a **non-repairable system**.

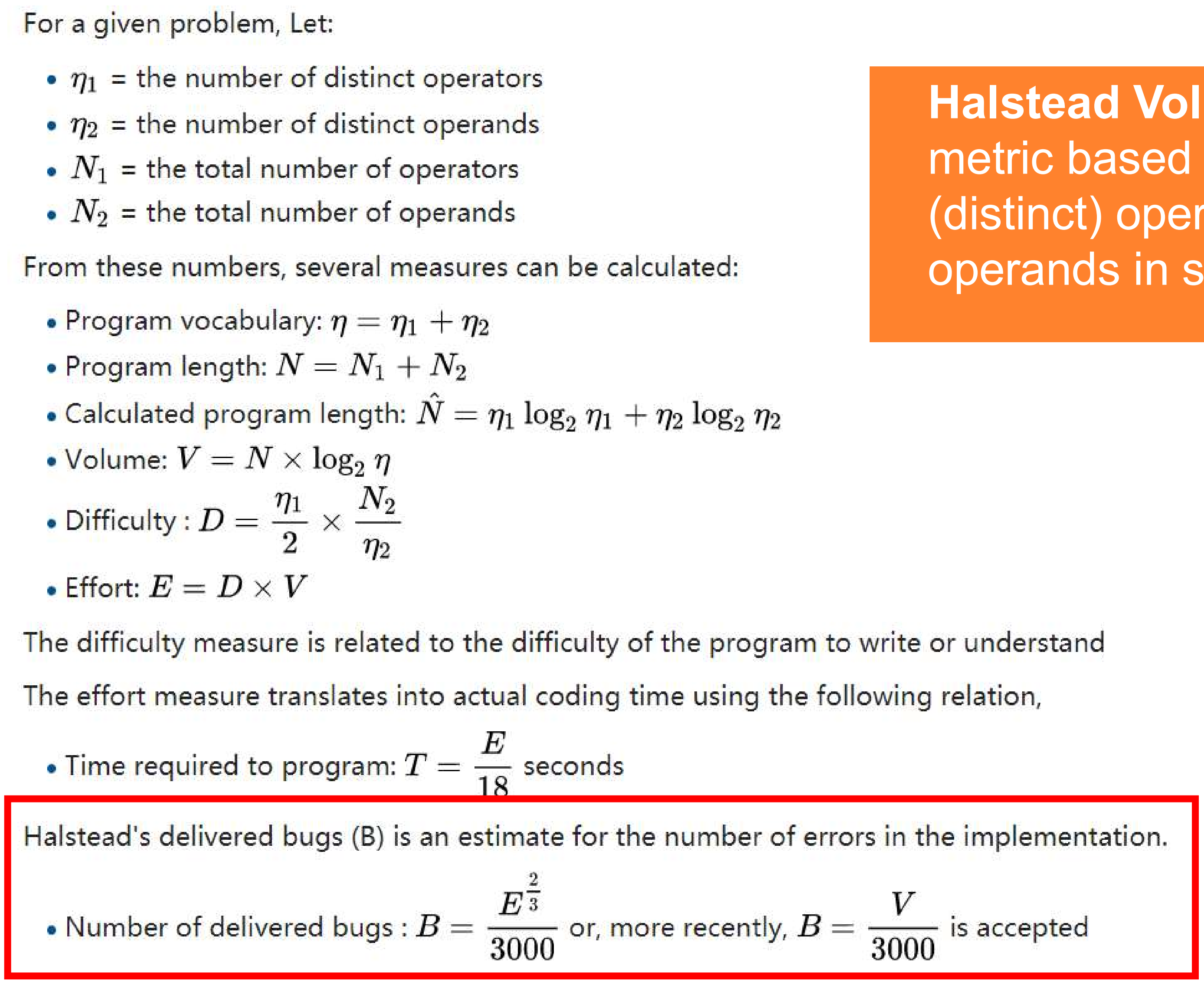


Residual defect rates 内部观察角度（间接）

- **Residual defect rates 残余缺陷率 refers to “bugs left over after the software has shipped” per KLOC: 每千行代码中遗留的bug的数量**
  - **1 - 10 defects/kloc: Typical industry software.**
  - **0.1 - 1 defects/kloc: High-quality validation. The Java libraries might achieve this level of correctness.**
  - **0.01 - 0.1 defects/kloc: The very best, safety-critical validation. NASA and companies like Praxis can achieve this level.**

- **This can be discouraging for large systems.**
  - For example, if you have shipped a million lines of typical industry source code (1 defect/kloc), it means you missed 1000 bugs!

Recall Maintainability



For a given problem, Let:

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

The difficulty measure is related to the difficulty of the program to write or understand

The effort measure translates into actual coding time using the following relation,

- Time required to program: $T = \frac{E}{18}$ seconds

Halstead's delivered bugs (B) is an estimate for the number of errors in the implementation.

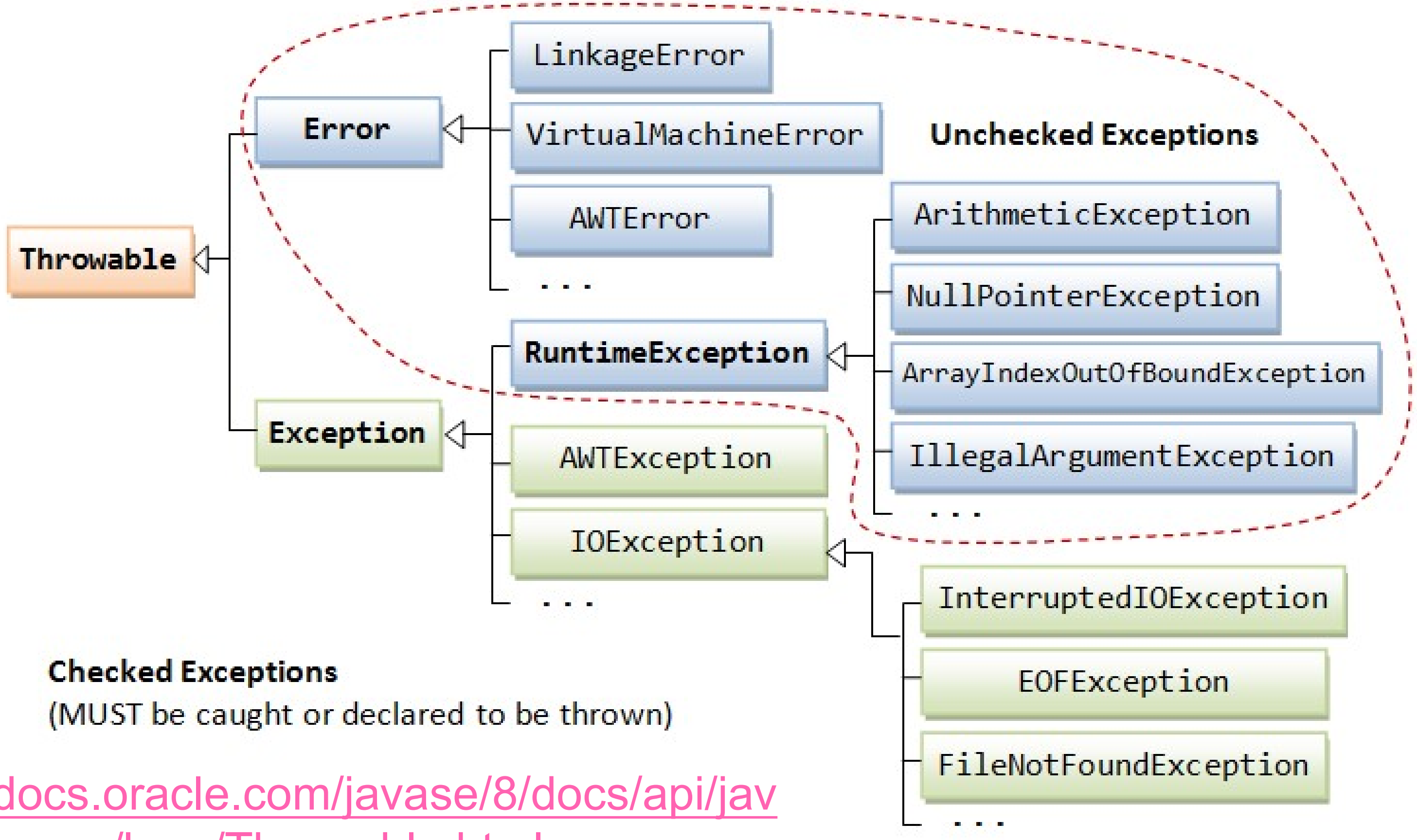
- Number of delivered bugs : $B = \frac{E^{\frac{2}{3}}}{3000}$ or, more recently, $B = \frac{V}{3000}$ is accepted

Halstead Volume: a composite metric based on the number of (distinct) operators and operands in source code.

3 Error and Exception in Java

“Abnormals” in Java

- **The base class for all Exception objects is `java.lang.Throwable`, together with its two subclasses `java.lang.Exception` and `java.lang.Error`.**



Checked Exceptions
(MUST be caught or declared to be thrown)

https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html

Error and Exception

- **The** `Error` class describes internal system errors and resource exhaustion situations inside the Java runtime system (e.g., `VirtualMachineError`, `LinkageError`) that rarely occur.
  - You should not throw an object of this type.
  - There is little you can do if such an internal error occurs, beyond notifying the user and trying to terminate the program gracefully.

内部错误：程序员通常无能为力，一旦发生，想办法让程序优雅的结束

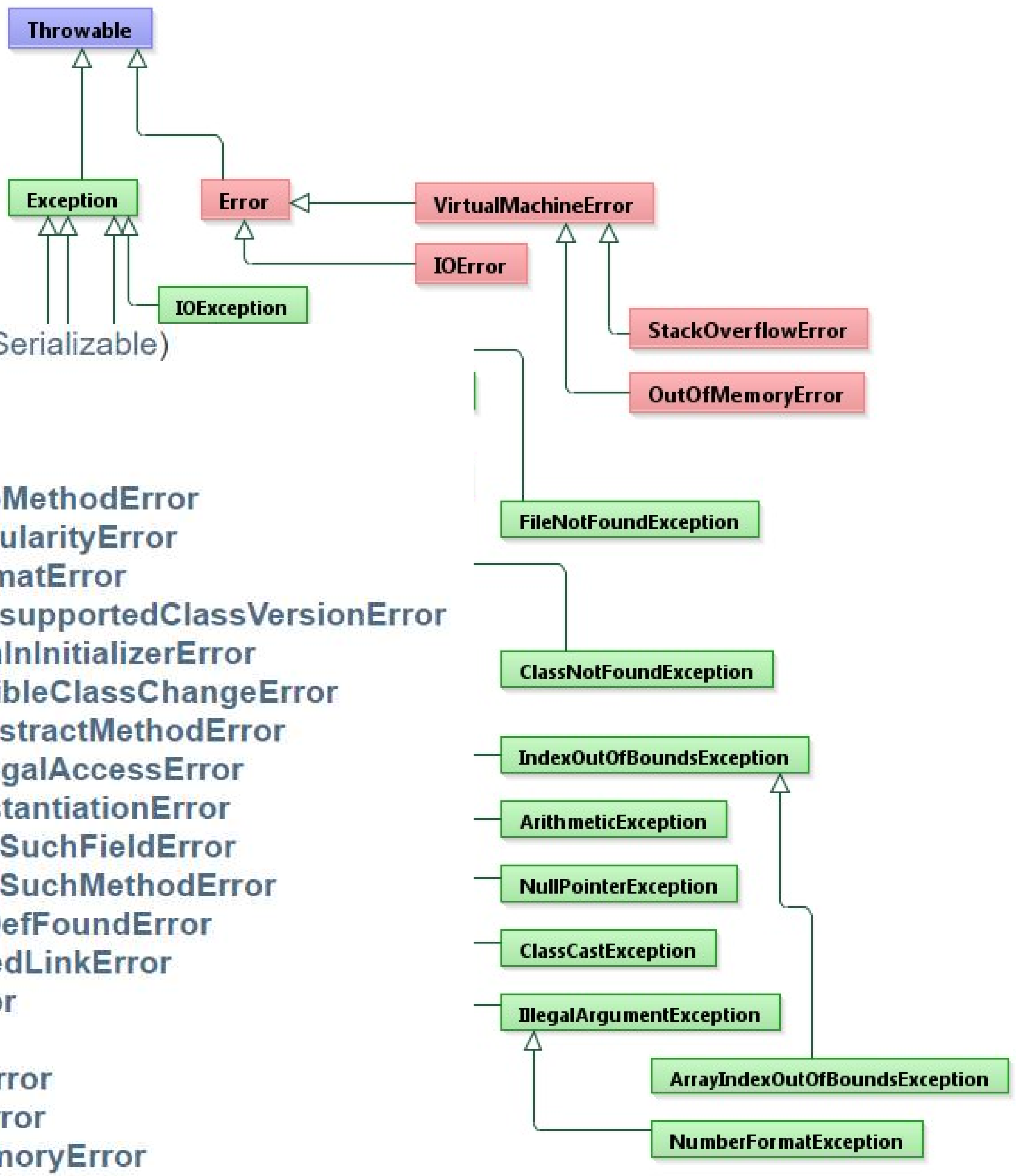
- **The** `Exception` **class describes the error caused by your program (e.g.** `FileNotFoundException`**,** `IOException`**).**
  - These errors could be caught and handled by your program (e.g., perform an alternate action or do a graceful exit by closing all the files, network and database connections).

异常：你自己程序导致的问题，可以捕获、可以处理

Sorts of errors

- **User input errors** 用户输入错误
  - In addition to the inevitable typos, some users like to blaze their own trail instead of following directions.
  - E.g. a user asks to connect to a URL that is syntactically wrong, the network layer will complain.
- **Device errors** 设备错误
  - Hardware does not always do what you want it to.
  - The printer may be turned off.
  - A web page may be temporarily unavailable.
  - Devices will often fail in the middle of a task.
- **Physical limitations** 物理限制
  - Disks can fill up
  - You can run out of available memory

Error



- java.lang.Throwable (implements java.io.Serializable)
  - java.lang.Error
    - java.lang.AssertionError
    - java.lang.LinkageError
      - java.lang.BootstrapMethodError
      - java.lang.ClassCircularityError
      - java.lang.ClassFormatError
        - java.lang.UnsupportedClassVersionError
      - java.lang.ExceptionInInitializerError
      - java.lang.IncompatibleClassChangeError
        - java.lang.AbstractMethodError
        - java.lang.IllegalAccessError
        - java.lang.InstantiationError
        - java.lang.NoSuchFieldError
        - java.lang.NoSuchMethodError
      - java.lang.NoClassDefFoundError
      - java.lang.UnsatisfiedLinkError
      - java.lang.VerifyError
    - java.lang.ThreadDeath
    - java.lang.VirtualMachineError
      - java.lang.InternalError
      - java.lang.OutOfMemoryError
      - java.lang.StackOverflowError
      - java.lang.UnknownError

在大多数时候，程序员不需要实例化Error

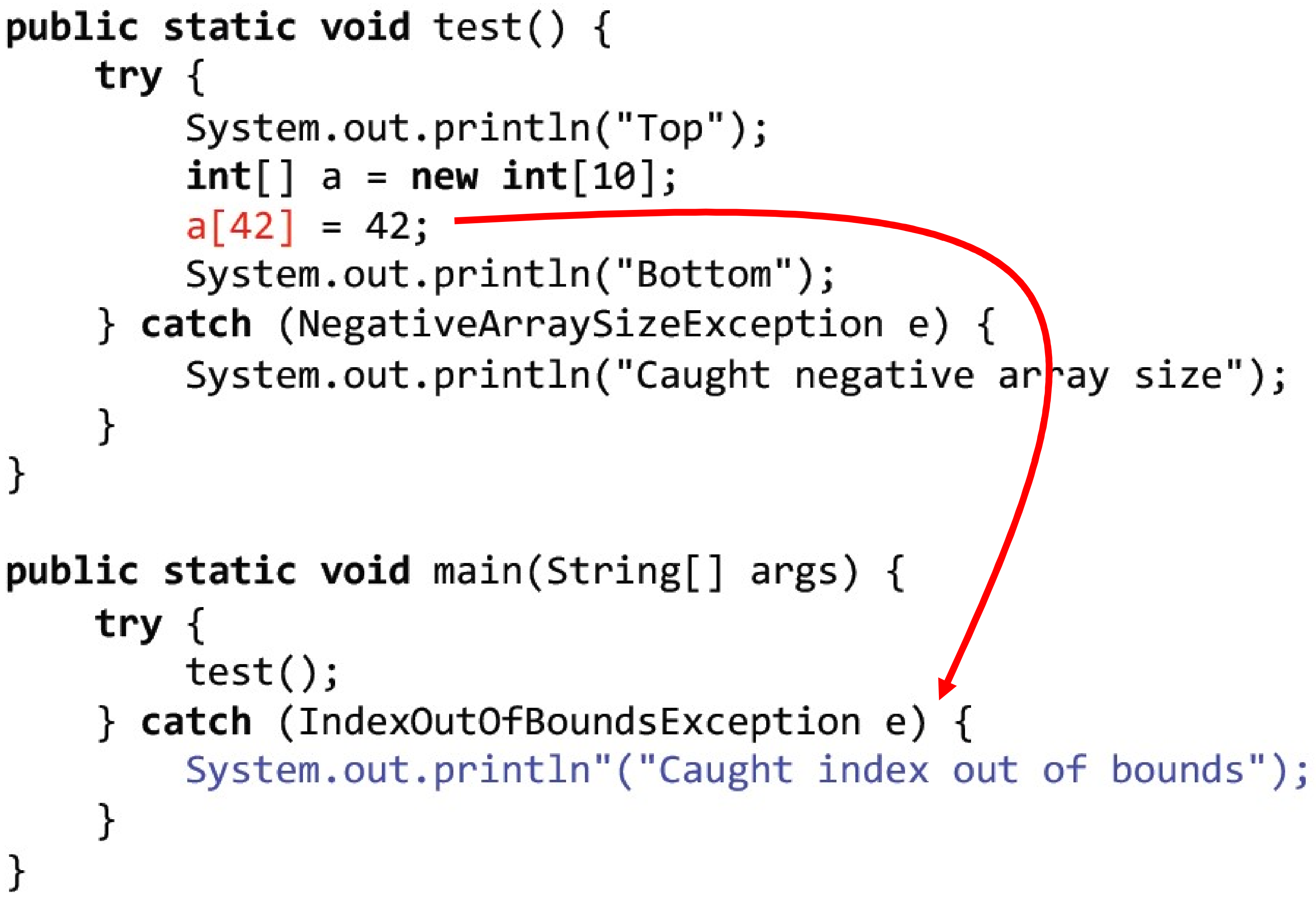
Some typical Errors

- **`VirtualMachineError`: thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.**
  - `OutOfMemoryError`: thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.
  - `StackOverflowError`: thrown when a stack overflow occurs because an application recurses too deeply.
  - `InternalError`: Thrown to indicate some unexpected internal error has occurred in the JVM.
- **`LinkageError`: a class has some dependency on another class; however, the latter class has incompatibly changed after the compilation of the former class.**
  - `NoClassDefFoundError`: Thrown if JVM or a `ClassLoader` instance tries to load in the definition of a class but no definition could be found.

4 Exception Handling

既然Error我们无能为力，
那就转向关注我们能处理的Exception

An Example



```
public static void test() {
    try {
        System.out.println("Top");
        int[] a = new int[10];
        a[42] = 42;
        System.out.println("Bottom");
    } catch (NegativeArraySizeException e) {
        System.out.println("Caught negative array size");
    }
}

public static void main(String[] args) {
    try {
        test();
    } catch (IndexOutOfBoundsException e) {
        System.out.println"("Caught index out of bounds");
    }
}
```

(1) What is Exception?

Exceptions

- **An exception** is an *abnormal event* that arises during the execution of the program and disrupts the normal flow of the program. 异常：程序执行中的非正常事件，程序无法再按预想的流程执行
- **Exceptions** are a specific means by which code can pass along errors or exceptional events to the code that called it. 将错误信息传递给上层调用者，并报告“案发现场”的信息
- **Java allows every method an alternative exit path if it is unable to complete its task in the normal way**. return之外的第二种退出途径
  - The method ***throws*** an object that encapsulates the error information.
  - The method exits immediately and does not return any value.
  - Moreover, execution does not resume at the code that called the method;
  - Instead, the ***exception-handling*** mechanism begins its search for an exception handler that can deal with this particular error condition. 若找不到异常处理程序，整个系统完全退出

Benefits of exceptions

Code that does not use Exceptions

```
FileInputStream fIn = new FileInputStream(fileName);
if (fIn == null) {
   switch (errno) {
     case _ENOFILE:
         System.err.println(“File not found: “ + …);
         return -1;
     default:
         System.err.println(“Something else bad happened: “ + …);
         return -1;
   }
}

DataInput dataInput = new DataInputStream(fIn);
if (dataInput == null) {
   System.err.println(“Unknown internal error.”);
   return -1; // errno > 0 set by new DataInputStream
}
int i = dataInput.readInt();
if (errno > 0) {
   System.err.println(“Error reading binary data from file”);
   return -1;
}
return i;
```

Benefits of exceptions

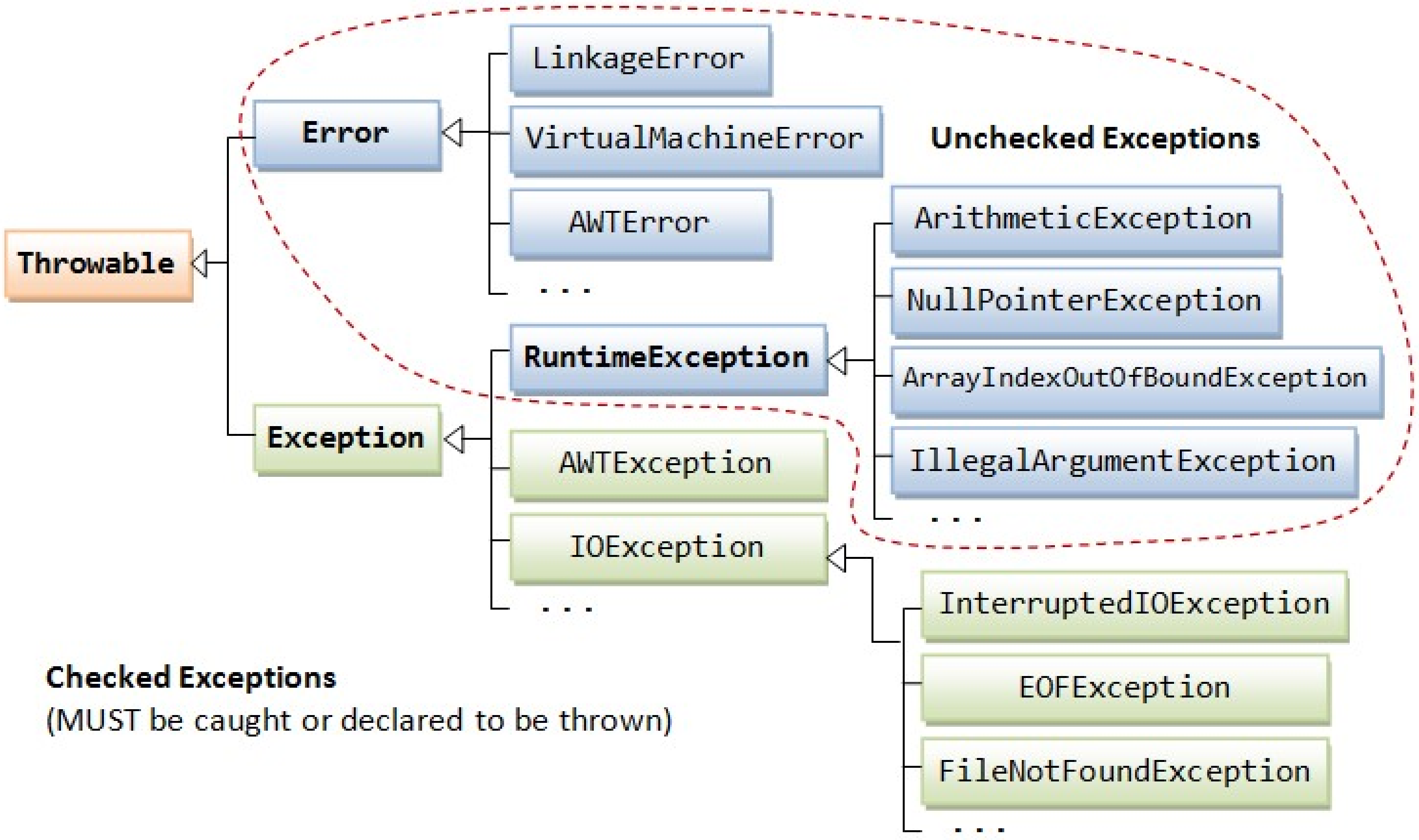
Code that use Exceptions

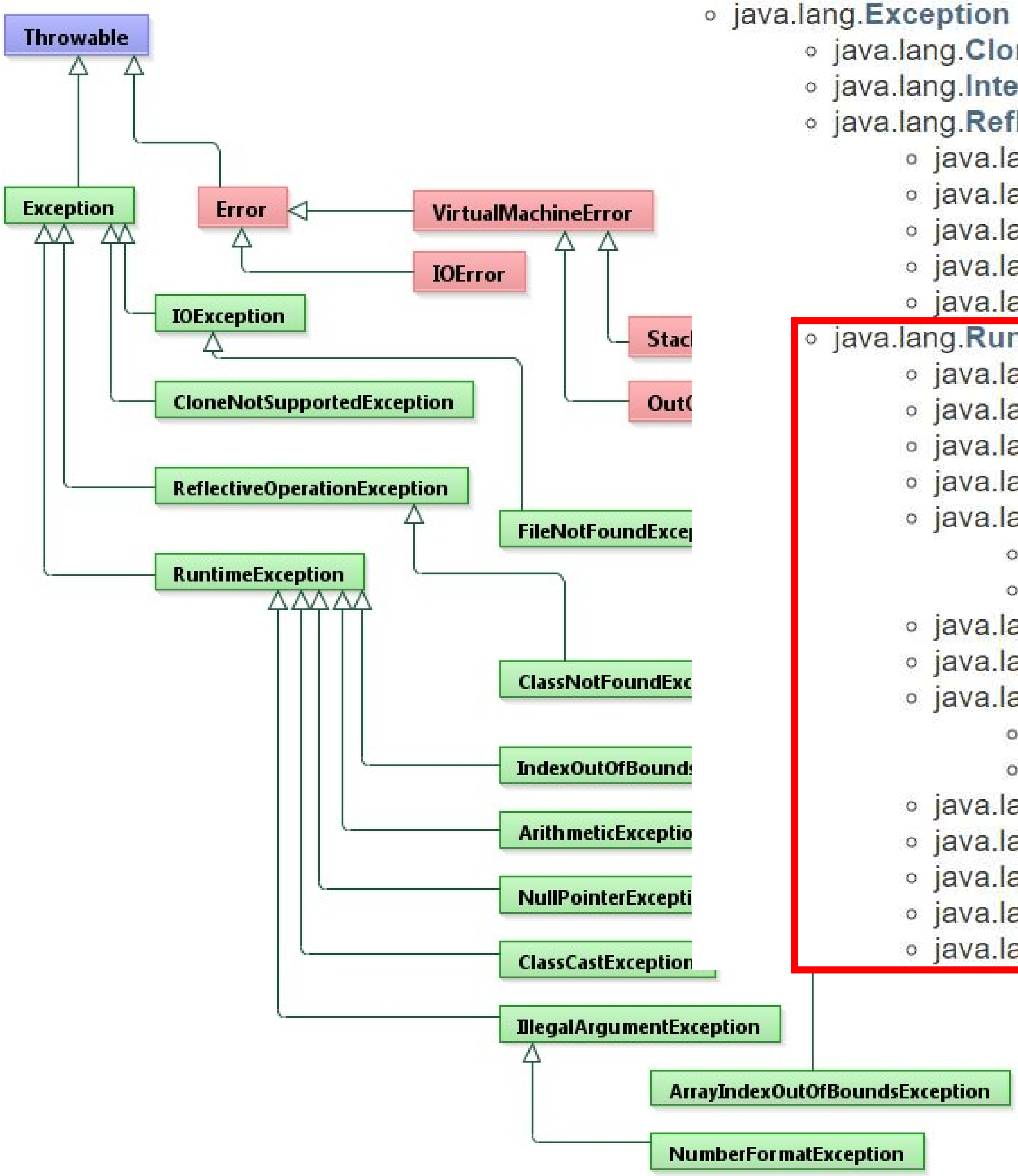
```
FileInputStream fileInput = null;

try {
        fileInput = new FileInputStream(fileName);
        DataInput dataInput = new DataInputStream(fileInput);
        return dataInput.readInt();
}
catch (FileNotFoundException e) {
        System.out.println("Could not open file " + fileName);
}
catch (IOException e) {
        System.out.println("Couldn’t read file: " + e);
}
finally {
        if (fileInput != null) fileInput.close();
}
```

(2) Classification of exceptions

Exceptions are derived from `Throwable`

- **In Java programming language, an exception object is always an instance of a class derived from `Throwable`.**





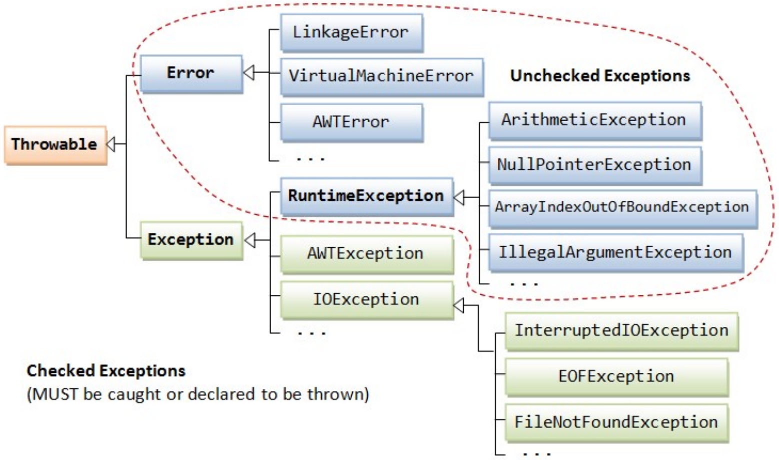
- java.lang.Exception
  - java.lang.CloneNotSupportedException
  - java.lang.InterruptedException
  - java.lang.ReflectiveOperationException
    - java.lang.ClassNotFoundException
    - java.lang.IllegalAccessException
    - java.lang.InstantiationException
    - java.lang.NoSuchFieldException
    - java.lang.NoSuchMethodException
  - java.lang.RuntimeException
    - java.lang.ArithmeticException
    - java.lang.ArrayStoreException
    - java.lang.ClassCastException
    - java.lang.EnumConstantNotPresentException
    - java.lang.IllegalArgumentException
      - java.lang.IllegalThreadStateException
      - java.lang.NumberFormatException
    - java.lang.IllegalMonitorStateException
    - java.lang.IllegalStateException
    - java.lang.IndexOutOfBoundsException
      - java.lang.ArrayIndexOutOfBoundsException
      - java.lang.StringIndexOutOfBoundsException
    - java.lang.NegativeArraySizeException
    - java.lang.NullPointerException
    - java.lang.SecurityException
    - java.lang.TypeNotPresentException
    - java.lang.UnsupportedOperationException

Runtime Exception and Other Exceptions

- **When doing Java programming, focus on the *Exception* hierarchy.**
- **The *Exception* hierarchy also splits into two branches:**
  - Exceptions that derive from `RuntimeException` 运行时异常
  - Those that do not. 其他异常
- **General rule:**
  - A `RuntimeException` happens because you made a programming error. 运行时异常：由程序员在代码里处理不当造成
  - Any other exception occurs because a bad thing, such as an I/O error, happened to your otherwise good program. 其他异常：由外部原因造成

Runtime Exception and Other Exceptions

- **Exceptions that inherit from `RuntimeException` include such problems as:**
  - A bad cast
  - An out-of-bounds array access
  - A `null` pointer access
  - ……



Checked Exceptions
(MUST be caught or declared to be thrown)

- **Exceptions that do not inherit from `RuntimeException` include**
  - Trying to read past the end of a file
  - Trying to open a file that doesn't exist
  - Trying to find a `Class` object for a string that does not denote an existing class
  - ……

RuntimeException

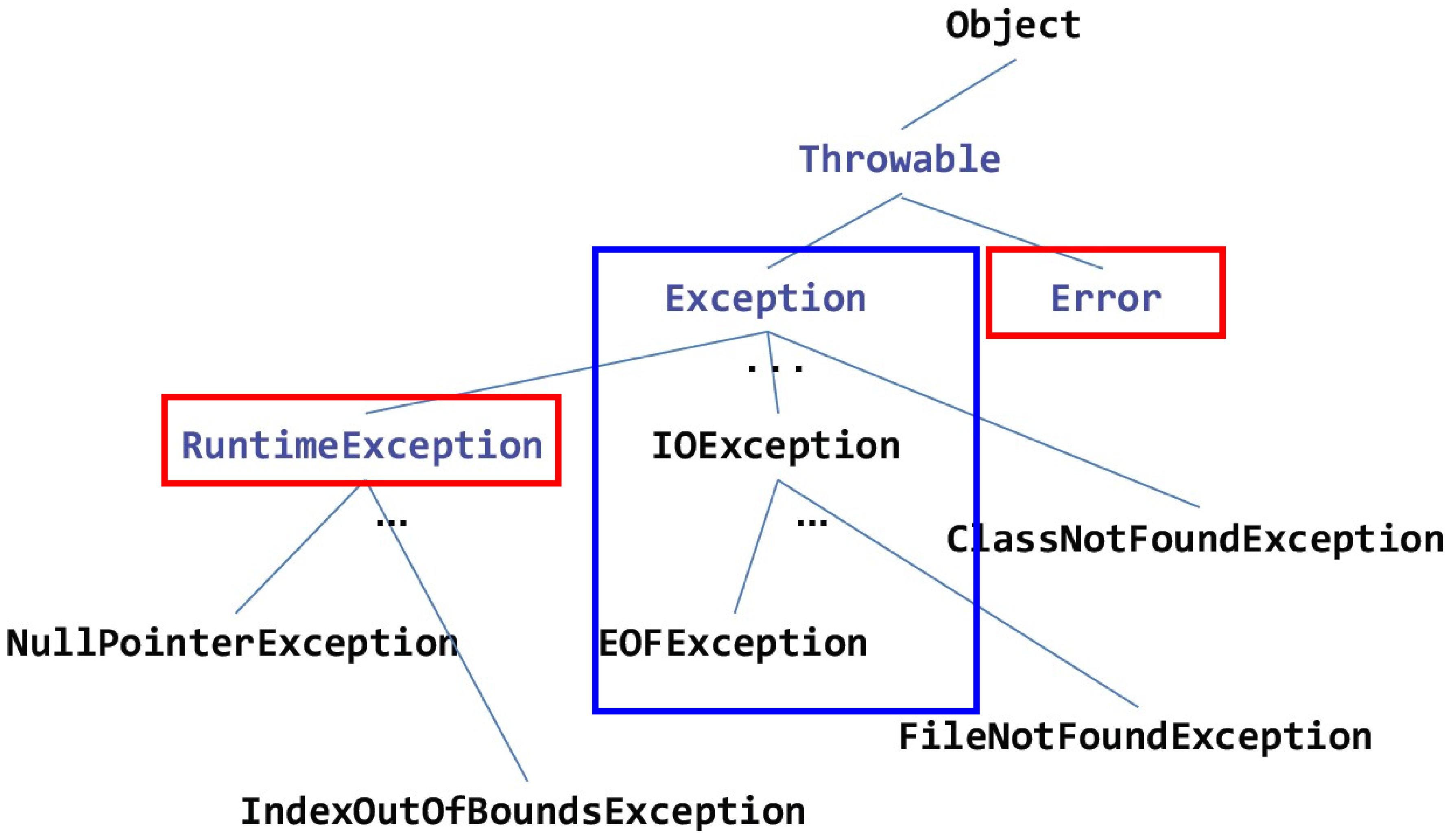
- **If it is a `RuntimeException`, it was your fault** 运行时异常，是程序源代码中引入的故障所造成的
  - You could have avoided `ArrayIndexOutOfBoundsException` by testing the array index against the array bounds.
  - The `NullPointerException` would not have happened had you checked whether the variable was null before using it.
  - 如果在代码中提前进行验证，这些故障就可以避免
- **How about a file that doesn’t exist?**
  - Can’t you first check whether the file exists, and then open it?
  - Actually, the file might be deleted right after you checked for its existence. Thus, the notion of “existence” depends on the environment, not just on your code. 非运行时异常，是程序员无法完全控制的外在问题所导致的
  - 即使在代码中提前加以验证（文件是否存在），也无法完全避免失效发生。

(3) *Checked* and *unchecked* exceptions

这是从异常处理机制的角度所做的分类

异常被谁check？——编译器、程序员

Again, the exception hierarchy in Java



How is an exception handled?

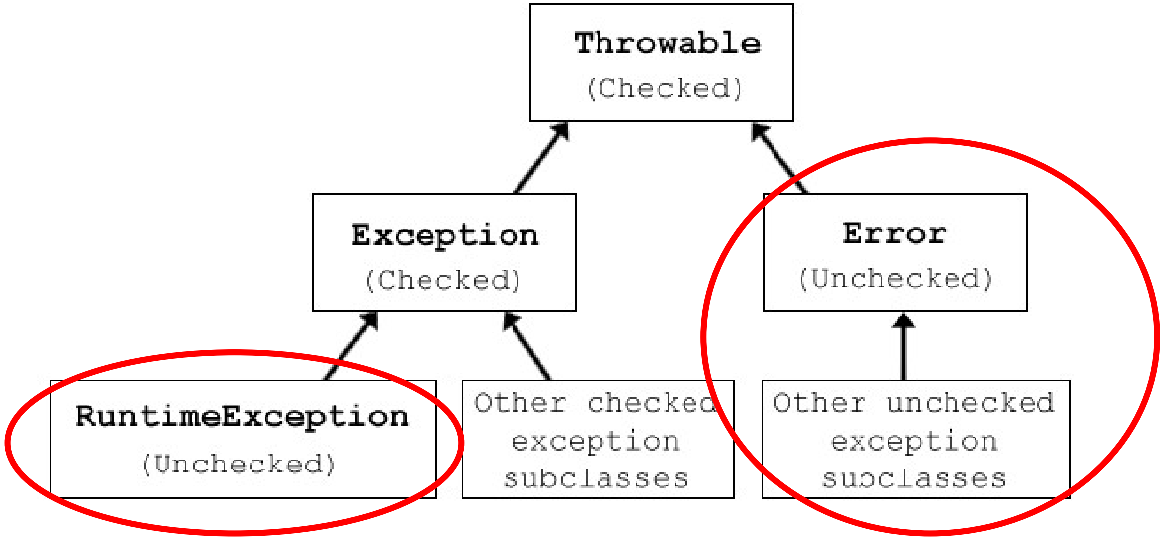
Checked Exceptions

- **When an exception occurs**
  - You have to either catch and handle the exception, or tell compiler that you can't handle it by declaring that your method throws that exception,
  - Then the code that uses your method will have to handle that exception (may choose to declare that it throws the exception if it can't handle it).
  - Compiler will check that we have done one of the two things (catch, or declare). 编译器可帮助检查你的程序是否已抛出或处理了可能的异常
- **Errors and Runtime Exceptions are not checked by compiler**
  - Errors represent conditions occurring outside the application, such as crash of the system. Runtime exceptions are usually occur by fault in the application logic.
  - You can't do anything in these situations, but have to re-write your program code. So these are not checked by compiler.
  - These runtime exceptions will uncover in development, and testing period. Then we have to refactor our code to remove these errors.

Unchecked Exceptions

Unchecked exceptions

- **Unchecked exception: Programming error, other unrecoverable failure (`Error` + `RuntimeException`)**
  - No action is required for program to compile, but uncaught exception will cause program to fail
  - 不需要在编译的时候用try…catch等机制处理
- 从RuntimeException派生出子类型

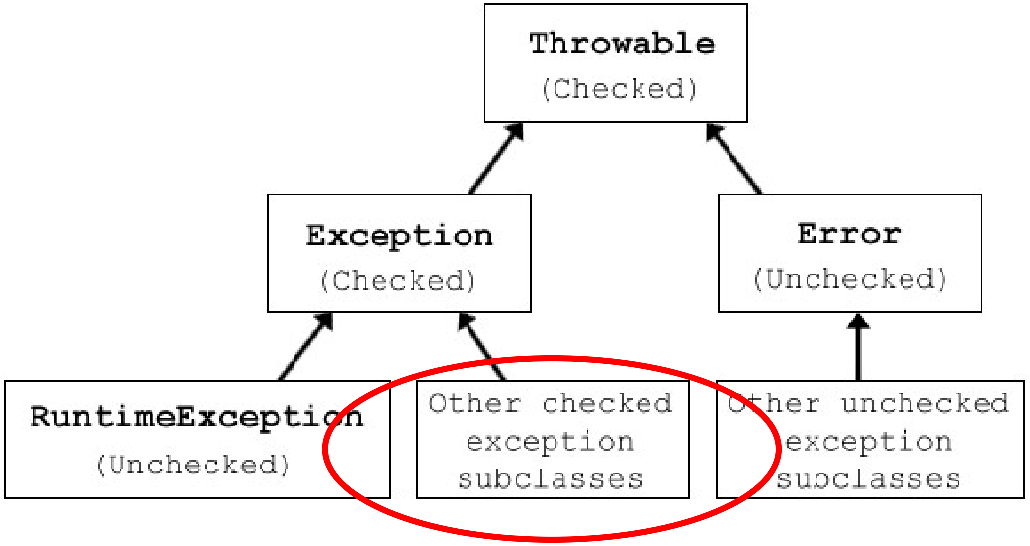


可以不处理，编译没问题，但执行时出现就导致程序失败，代表程序中的潜在bug

类似于编程语言中的dynamic type checking

Checked exceptions

- **Checked exception: An error that every caller should be aware of and handle**
- **Must be caught or propagated, or program won’t compile (The compiler checks that you provide exception handlers for all checked exceptions)**
- 需要从Exception派生出子类型



必须捕获并指定错误处理器handler，否则编译无法通过

类似于编程语言中的static type checking

Examples of unchecked exceptions

```
public class NullPointerExceptionExample {
   public static void main(String args[]){
      String str=null;
      System.out.println(str.trim());
   }
}
```

```
Exception in thread "main" java.lang.NullPointerException
```

```
public class ArrayIndexOutOfBoundExceptionExample {
   public static void main(String args[]){
      String strArray[]={"Arpit","John","Martin"};
      System.out.println(strArray[4]);
   }
}
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
```

Common Unchecked Exception Classes

- `ArrayIndexOutOfBoundsException`: **thrown by JVM when your code uses an array index, which is outside the array's bounds.**

```
int[] anArray = new int[3];
System.out.println(anArray[3]);
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
3
```

- `NullPointerException`: **thrown by the JVM when your code attempts to use a null reference where an object reference is required.**

```
String[] strs = new String[3];
System.out.println(strs[0].length());
Exception in thread "main" java.lang.NullPointerException
```

在编程和编译的时候，IDE与编译器均不会给出任何错误提示

Common Unchecked Exception Classes

- `NumberFormatException`: **Thrown programmatically (e.g., by `Integer.parseInt()`) when an attempt is made to convert a string to a numeric type, but the string does not have the appropriate format.**

```
Integer.parseInt("abc");
```

```
Exception in thread "main" java.lang.NumberFormatException:
For input string: "abc"
```

在编程和编译的时候，IDE与编译器均不会给出任何错误提示

- `ClassCastException`: **thrown by JVM when an attempt is made to cast an object reference fails.**

```
Object o = new Object();
Integer i = (Integer)o;
```

```
Exception in thread "main" java.lang.ClassCastException:
java.lang.Object cannot be cast to java.lang.Integer
```

Checked Exception Handling Operations

- **Five keywords are used in exception handling:**
  - `try`
  - `catch`
  - `finally`
  - `throws`
  - `throw`

- **Java's exception handling consists of three operations:**
  - Declaring exceptions (`throws`) 声明“本方法可能会发生XX异常”
  - Throwing an exception (`throw`) 抛出XX异常
  - Catching an exception (`try, catch, finally`) 捕获并处理XX异常

Examples of checked exceptions

```
public static void main(String args[]) {

  FileInputStream fis = null;

  try {
     fis = new FileInputStream("sample.txt");
     int c;
     while ((c = fis.read()) != -1)
        System.out.print((char) c);
     fis.close();
  } catch (FileNotFoundException e) {
     e.printStackTrace();
  } catch (IOException e) {
     e.printStackTrace();
  } catch (Exception e) {
     e.printStackTrace();
  }
}
```

Examples of checked exceptions

```
public static void main(String args[]) throws IOException {

    FileInputStream fis = null;
    fis = new FileInputStream("sample.txt");

    int k;
    while ((k = fis.read()) != -1)
        System.out.print((char) k);

    fis.close();
}
```

A note

- **Checked exceptions have to be either caught or declared in the method signature using** throws**, whereas with unchecked ones this is optional. Unchecked**异常也可以使用throws声明或try/catch进行捕获，但大多数时候是不需要的，也不应该这么做——掩耳盗铃，对发现的编程错误充耳不闻

```
public class ArrayIndexOutOfBoundExceptionExample {
   public static void main(String args[]){

     try {
        String strArray[]={"Arpit","John","Martin"};
        System.out.println(strArray[4]);

     } catch(ArrayIndexOutOfBoundsException e) {...}
   }
}
```

Use checked or unchecked exceptions in your code?

- **When deciding on checked exceptions vs. unchecked exceptions, ask yourself, “What action can the client code take when the exception occurs?** 当要决定是采用checked exception还是unchecked exception的时候，问一个问题：“如果这种异常一旦抛出，client会做怎样的补救？”

Client's reaction when exception happens	Exception type
Client code cannot do anything	Make it an unchecked exception
Client code will take some useful recovery action based on information in exception	Make it a checked exception

- 如果客户端可以通过其他的方法恢复异常，那么采用checked exception；
- 如果客户端对出现的这种异常无能为力，那么采用unchecked exception；
- 异常出现的时候，要做一些试图恢复它的动作而不要仅仅的打印它的信息。

Use checked or unchecked exceptions in your code?

- 尽量使用**unchecked exception**来处理编程错误：因为unchecked exception不用使客户端代码显式的处理它们，它们自己会在出现的地方挂起程序并打印出异常信息。
  - 充分利用Java API中提供的丰富unchecked exception，如`NullPointerException`，`IllegalArgumentException`和`IllegalStateException`等，使用这些标准的异常类而不需亲自创建新的异常类，使代码易于理解并避免过多消耗内存。
- 如果**client**端对某种异常无能为力，可以把它转变为一个**unchecked exception**，程序被挂起并返回客户端异常信息

```
try{
    ..some code that throws SQLException
}catch(SQLException ex){
    throw new RuntimeException(ex);
}
```

不同的程序员有不同的观点，关于该如何选择checked和unchecked exceptions，在技术人员中存有很大的争论。

Use checked or unchecked exceptions in your code?

- **Try not to create new custom exceptions if they do not have useful information for client code**
  - 不要创建没有意义的异常，client应该从checked exception中获取更有价值的信息(案发现场具体是什么样子)，利用异常返回的信息来明确操作失败的原因。
  - 如果client仅仅想看到异常信息，可以简单抛出一个unchecked exception:

```
throw new RuntimeException("Username already taken");
```

- **Summary**
  - Checked exception应该让客户端从中得到丰富的信息。
  - 要想让代码更加易读，倾向于用unchecked exception来处理程序中的错误

Use checked or unchecked exceptions in your code?

- **Checked Exceptions should be used for expected, but unpreventable errors that are reasonable to recover from.** 错误可预料，但无法预防，但可以有手段从中恢复，此时使用**checked exception**。

- **Unchecked Exceptions should be used for everything else.** 如果做不到这一点，则使用**unchecked exception**
  - Expected but unpreventable: 可预料但不可预防
    - The caller did everything within their power to validate the input parameters, but some condition outside their control has caused the operation to fail.
    - For example, you try reading a file but someone deletes it between the time you check if it exists and the time the read operation begins.
    - By declaring a checked exception, you are telling the caller to anticipate this failure.
    - 不可预防：脱离了你的程序的控制范围

Use checked or unchecked exceptions in your code?

- **Reasonable to recover from:** 可合理的恢复
  - There is no point telling callers to anticipate exceptions that they cannot recover from.
  - If a user attempts to read from an non-existing file, the caller can prompt them for a new filename.
  - On the other hand, if the method fails due to a programming bug (invalid method arguments or buggy method implementation) there is nothing the application can do to fix the problem in mid-execution.
  - The best it can do is log the problem and wait for the developer to fix it at a later time.
  - 如果读文件的时候发现文件不存在了，可以让用户选择其他文件；但是如果调用某方法时传入了错误的参数，则无论如何都无法在不中止执行的前提下进行恢复。
- **Unless the exception you are throwing meets all of the above conditions it should use an Unchecked Exception.**

Exception design considerations

- Use checked exceptions for special results (i.e., anticipated situations)
- Use unchecked exceptions to signal bugs (unexpected failures)

- You should use an unchecked exception only to signal an unexpected failure (i.e. a bug), or if you expect that clients will usually write code that ensures the exception will not happen, because there is a convenient and inexpensive way to avoid the exception;
- Otherwise you should use a checked exception.

Checked vs. Unchecked Exceptions

	Checked exception	Unchecked exception
Basic	The compiler checks the checked exception. If we do not handle the checked exception, then the compiler objects. 必须被显式地捕获或者传递（try-catch-finally-throw），否则编译器无法通过	The compiler does not check the Unchecked exception. Even if we do not handle the unchecked exception, the compiler doesn't object.异常可以不必捕获或抛出，编译器不去检查
Class of Exception	Except RuntimeException class, all the child classes of the class "Exception", and the "Error" class and its child classes are Checked Exception. 继承自Exception类	RuntimeException class and its child classes, are Unchecked Exceptions.继承自RuntimeException类
Handling	从异常发生的现场获取详细的信息，利用异常返回的信息来明确操作失败的原因，并加以合理的恢复处理	简单打印异常信息，无法再继续处理
Appearance	代码看起来复杂，正常逻辑代码和异常处理代码混在一起	清晰，简单

(4) Declaring Checked Exceptions by `throws`

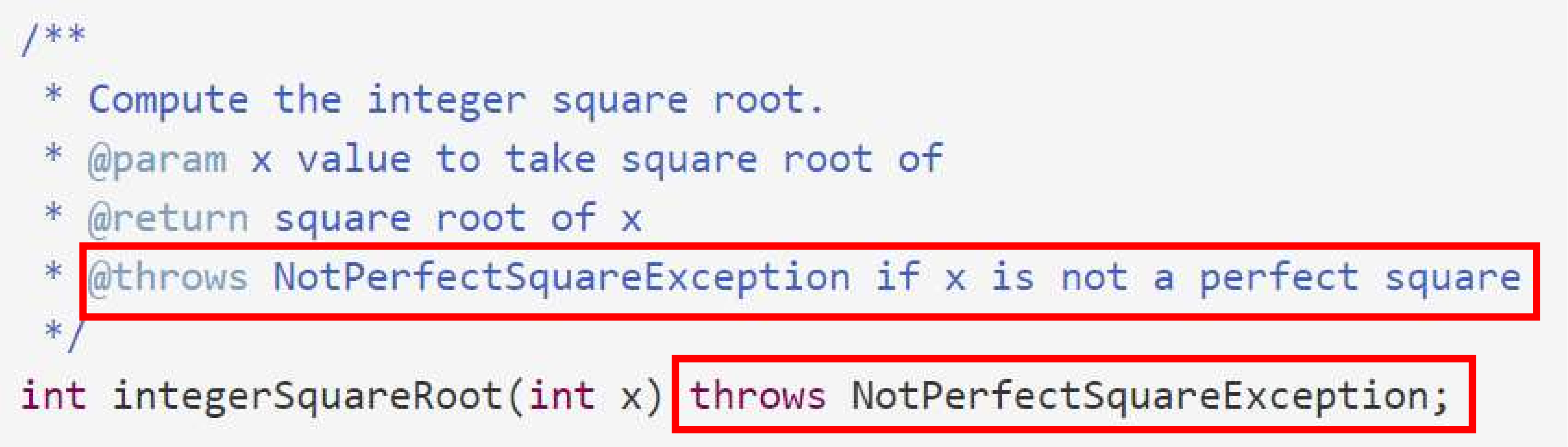
Declaring Checked Exceptions by `throws`

- **A Java method can throw an exception if it encounters a situation it cannot handle.**
  - A method will not only tell the Java compiler what values it can return, it is also going to tell the compiler what can go wrong. “异常”也是方法和client端之间spec的一部分，在post-condition中刻画
  - E.g. code that attempts to read from a file knows that the file might not exist or that it might be empty. The code that tries to process the information in a file therefore will need to notify the compiler that it can throw some sort of `IOException`.
- **The place in which you advertise that your method can throw an exception is the header of the method;** the header changes to reflect the checked exceptions the method can throw.

```
public FileInputStream(String name)
       throws FileNotFoundException
```

How to declare exceptions in a specification

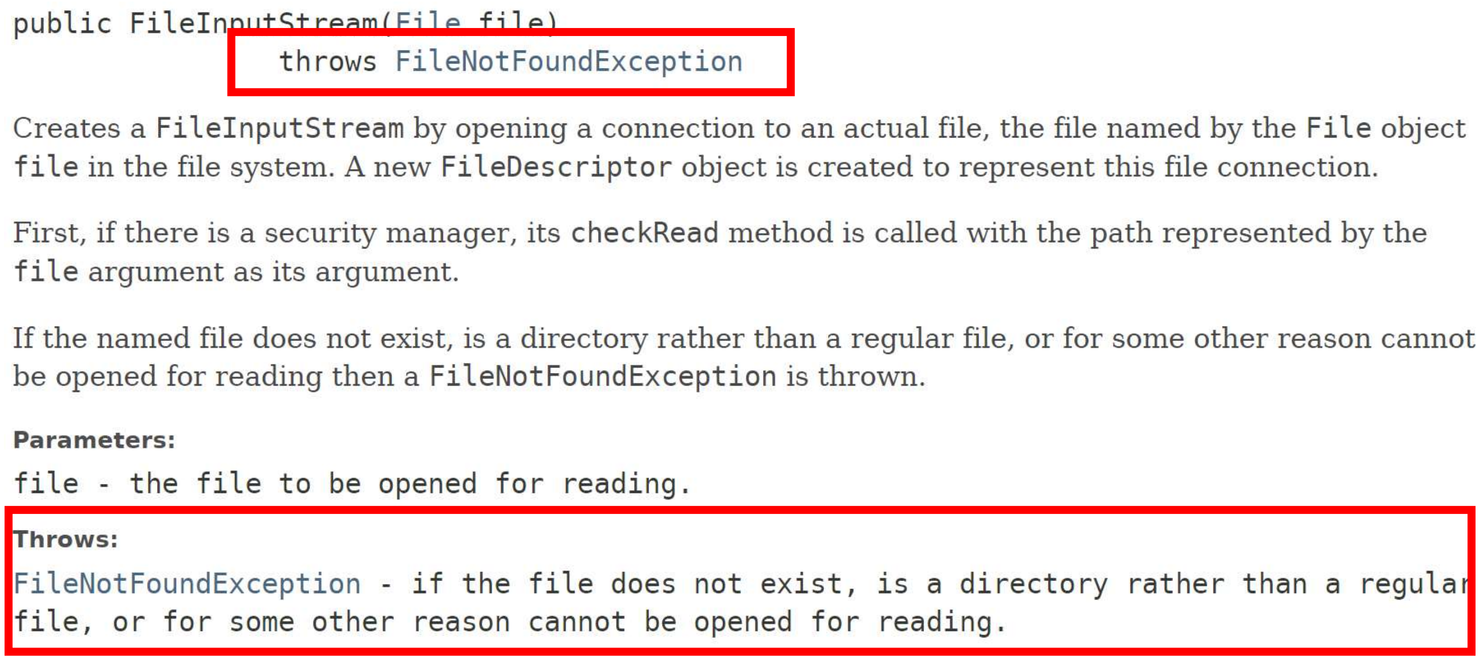
- **Checked exceptions that signal a special result are always documented with a Javadoc `@throws` clause, specifying the conditions under which that special result occurs.**
- **Java may also require the exception to be included in the method signature, using a `throws` declaration.**



```
/**
 * Compute the integer square root.
 * @param x value to take square root of
 * @return square root of x
 * @throws NotPerfectSquareException if x is not a perfect square
 */
int integerSquareRoot(int x) throws NotPerfectSquareException;
```

How to declare exceptions in a specification

- **As with Java methods that are part of the supplied classes, you declare that your method may throw an exception with an *exception specification* in the method header and in spec. 程序员必须在方法的spec中明确写清本方法会抛出的所有checked exception，以便于调用该方法的client加以处理**



```
public FileInputStream(File file)
                throws FileNotFoundException
```

Creates a `FileInputStream` by opening a connection to an actual file, the file named by the `File` object `file` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method is called with the path represented by the `file` argument as its argument.

If the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a `FileNotFoundException` is thrown.

Parameters:

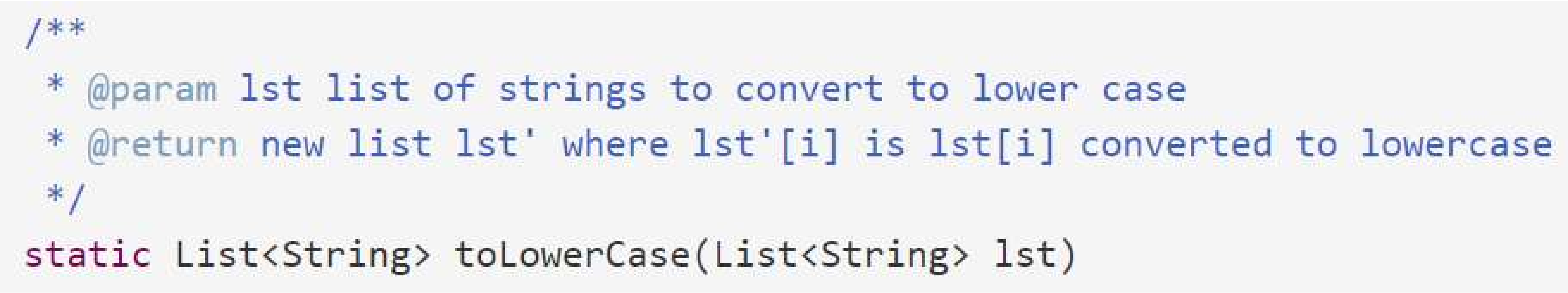
`file - the file to be opened for reading.`

Throws:

`FileNotFoundException - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.`

How to declare exceptions in a specification

- **Unchecked exceptions that are used to signal unexpected failures – bugs in either the client or the implementation – are not part of the postcondition of a method, so they should not appear in either `@throws` or `throws`.**
- **For example, `NullPointerException` need never be mentioned in a spec.**



```
/**
 * @param lst list of strings to convert to lower case
 * @return new list lst' where lst'[i] is lst[i] converted to lowercase
 */
static List<String> toLowerCase(List<String> lst)
```

Declare more than one Checked Exceptions

- **If a method might throw more than one checked exception type, you must list all exception classes in the header.**

```
class MyAnimation {
    . . .
    public Image loadImage(String s)
              throws FileNotFoundException, EOFException
    {
      . . .
    }
}
```

Declaring Checked Exceptions by `throws`

- **When you write your own methods, you don’t have to advertise every possible `throwable` object that your method might actually throw.** 你的方法应该**throws**什么异常？
  - You call a method that throws a checked exception—for example, the `FileInputStream` constructor. 你所调用的其他函数抛出了一个checked exception——从其他函数传来的异常
  - You detect an error and throw a checked exception with the `throw` statement. 当前方法检测到错误并使用throws抛出了一个checked exception——你自己造出的异常
- **Then you must tell the programmers who will use your method about the possibility of an exception.** 此时需要告知你的**client**需要处理这些异常
  - If no handler catches the exception, the current thread of execution terminates. 如果没有handler来处理被抛出的checked exception，程序就终止执行

Don’t throw `Error` and unchecked exceptions

- **Do not need to advertise internal Java errors— exceptions inheriting from `Error`.**
  - Any code could potentially throw those exceptions, and they are entirely beyond your control.
  - An internal error occurs in the virtual machine or runtime library.
- **You should not advertise unchecked exceptions inheriting from `RuntimeException`.**
  - These runtime errors are completely under your control.
  - If you are so concerned about array index errors, you should spend your time fixing them instead of advertising the possibility they can happen.
  - You make a programming error, such as `a[-1] = 0` that gives rise to an unchecked exception (`ArrayIndexOutOfBoundsException`).

```
void drawImage(int i) throws ArrayIndexOutOfBoundsException

// bad style!
```

Considering subtyping polymorphism

- **If you override a method from a superclass, the checked exceptions that the subclass method declares cannot be more general than those of the superclass method.** 如果子类型中**override**了父类型中的函数，那么子类型中方法抛出的异常不能比父类型抛出的异常类型更宽泛
- **It is OK to throw more specific exceptions, or not to throw any exceptions in the subclass method.** 子类型方法可以抛出更具体的异常，也可以不抛出任何异常
- **In particular, if the superclass method throws no checked exception at all, neither can the subclass.** 如果父类型的方法未抛出异常，那么子类型的方法也不能抛出异常。

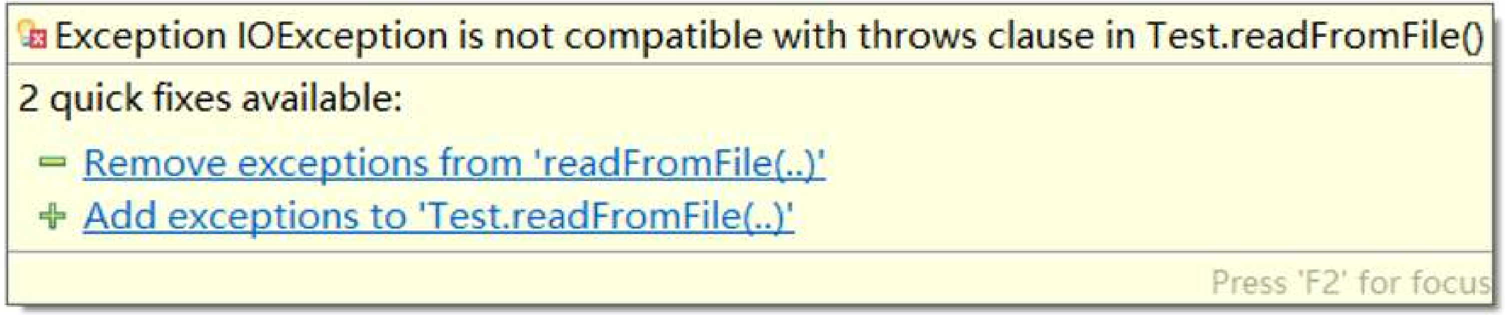
参见LSP原则
目标是子类型多态：客户端可用统一的方式处理不同类型的对象，子类型可替代父类型

Liskov Substitution Principle (LSP)

- **LSP is a particular definition of a subtyping relation, called (strong) behavioral subtyping** 强行为子类型化
- **In programming languages, LSP is relied on the following restrictions:**
  - Preconditions cannot be strengthened in a subtype.
  - Postconditions cannot be weakened in a subtype.
  - Invariants of the supertype must be preserved in a subtype.
  - Contravariance of method arguments in a subtype 子类型方法参数：逆变
  - Covariance of return types in a subtype. 子类型方法的返回值：协变
  - No new exceptions should be thrown by methods of the subtype, except where those exceptions are themselves subtypes of exceptions thrown by the methods of the supertype.

Considering subtyping polymorphism

```
public class Test {
  public boolean readFromFile() throws FileNotFoundException {
    ...
  }
}

class SubType extends Test {
  @Override
  public boolean readFromFile() throws IOException {
    ...
  }
}
```



Exception IOException is not compatible with throws clause in Test.readFromFile()

2 quick fixes available:

- Remove exceptions from 'readFromFile(..)'
- Add exceptions to 'Test.readFromFile(..)'

Press 'F2' for focus

(5) How to Throw an Exception

How to Throw an Exception

- **Suppose you have a method,** `readData`**, that is reading in a file. Something terrible has happened in your code.**
- **You may decide this situation is so abnormal that you want to throw an exception** `EOFException` **with the description "Signals that an EOF has been reached unexpectedly during input."**

```
throw new EOFException();
```

or

```
EOFException e = new EOFException();
throw e;
```

How to Throw an Exception

声明：本函数可能发生该异常

```
String readData(Scanner in) throws EOFException
{
    . . .
    while (. . .)
    {
        if (!in.hasNext()) // EOF encountered
        {
            if (n < len)
                  throw new EOFException();
         }
            . . .
     }
      return s;
}
```

问题：还有可能是在哪里抛出EOFException？

异常在这里发生了

How to Throw an Exception

- **The** `EOFException` **has a second constructor that takes a** `string` **argument.**
- **You can put this to good use by describing the exceptional condition more carefully.** 利用Exception的构造函数，将发生错误的现场信息充分的传递给**client**。

```
String gripe = "Content-length: " + len + ", Received: " + n;

throw new EOFException(gripe);
```

How to Throw an Exception

- **Throwing an exception is easy if one of the existing exception classes works for you:**
  - Find an appropriate exception class 找到一个能表达错误的Exception类/或者构造一个新的Exception类
  - Make an object of that class 构造Exception类的实例，将错误信息写入
  - Throw it 抛出它

- **Once a method throws an exception, it does not return to its caller. This means you do not have to worry about cooking up a default return value or an error code** 一旦抛出异常，方法不会再将控制权返回给调用它的client，因此也无需考虑返回错误代码

(6) Creating Exception Classes

Creating Exception Classes

- **Your code may run into a problem which is not adequately described by any of the standard exception classes.**
- **In this case, it is easy enough to create your own exception class.**
- 如果**JDK**提供的**exception**类无法充分描述你的程序发生的错误，可以创建自己的异常类
- **Just derive it from `Exception`, or from a child class of `Exception` such as `IOException`.**
- **It is customary to give both a default constructor and a constructor that contains a detailed message.**
  - The `toString` method of the `Throwable` superclass returns a string containing that detailed message, which is handy for debugging.

An example for checked exception

- **To define a checked exception you create a subclass (or hierarchy of subclasses) of `java.lang.Exception`:**

```
public class FooException extends Exception {
  public FooException() { super(); }
  public FooException(String message) { super(message); }
  public FooException(String message, Throwable cause) {
       super(message, cause);
  }
  public FooException(Throwable cause) { super(cause); }
}
```

- **Methods that can potentially throw or propagate this exception must declare it:**

```
public void calculate(int i) throws FooException, IOException;
```

An example for checked exception

- **Code calling this method must either handle or propagate this exception (or both):**

```
try {
  ...

} catch(FooException ex) {

  ex.printStackTrace();
  System.exit(1);

} catch(IOException ex) {
  throw new FooException(ex);
}
```

An example for unchecked exception

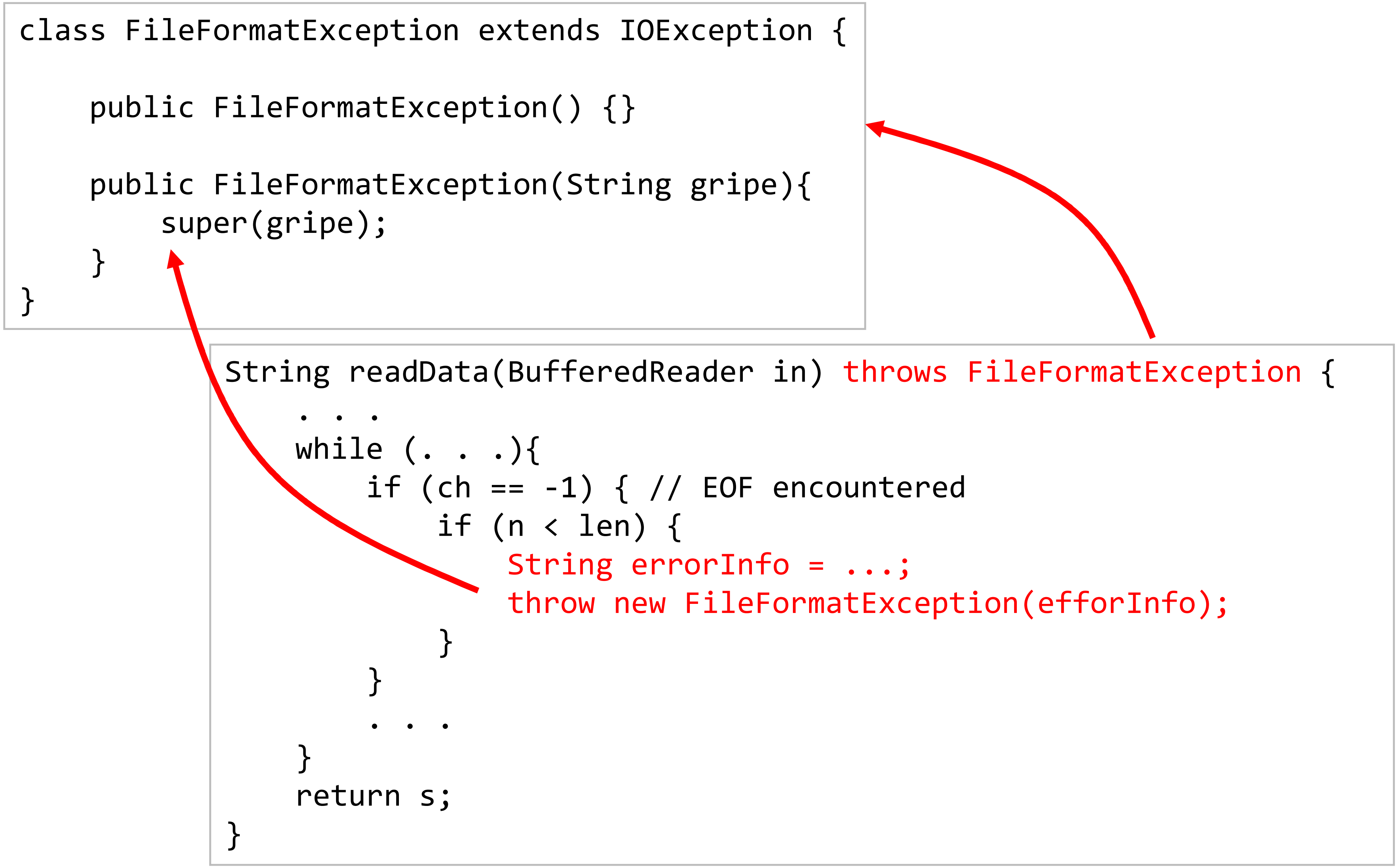
- **Sometimes there will be situations where you don‘t want to force every method to declare your exception implementation in its `throws` clause. In this case you can create an unchecked exception that extends `java.lang.RuntimeException`.**

```
public class FooRuntimeException extends RuntimeException {
  ...
}
```

- **Methods can throw or propagate `FooRuntimeException` exception without declaring it.**

```
public void calculate(int i) {
  if (i < 0) {
    throw new FooRuntimeException("i < 0: " + i);
  }
}
```

Creating more specific unchecked exception



```
class FileFormatException extends IOException {

    public FileFormatException() {}

    public FileFormatException(String gripe){
        super(gripe);
    }
}
```

```
String readData(BufferedReader in) throws FileFormatException {
    . . .
    while (. . .){
        if (ch == -1) { // EOF encountered
            if (n < len) {
                String errorInfo = ...;
                throw new FileFormatException(efforInfo);
            }
        }
        . . .
    }
    return s;
}
```

An exception class can contain more information

```
public class InsufficientFundsException extends Exception {
   private double amount;
   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }
   public double getAmount(){
      return amount;
   }
}

...
double needs = amount - balance;
throw new InsufficientFundsException(needs);
...

try{  ...
}catch(InsufficientFundsException e){
   System.out.println(“Money is short for "+ e.getAmount());
}
```

包含更多“案发现场信息”的异常类定义和辅助函数

抛出异常的时候，将现场信息记入异常

在异常处理时，利用这些信息给用户更有价值的帮助

(7) Catching Exceptions

Catching Exceptions

- **If an exception occurs that is not caught anywhere, the program will terminate and print a message to the console, giving the type of the exception and a stack trace.** 异常发生后，如果找不到处理器，就终止执行程序，在控制台打印出**stack trace**。
  - GUI programs catch exceptions, print stack trace messages, and then go back to the user interface processing loop.
- **To catch an exception, set up a** `try/catch` **block :**

```
try {
    code
    more code
    more code
}
catch (ExceptionType e) {
    handler for this type
}
```

Catching Exceptions

- **If any code inside the `try` block throws an exception of the class specified in the `catch` clause, then**
  - The program skips the remainder of the code in the ***try block***.
  - The program executes the handler code inside the ***catch clause***.

  - If none of the code inside the `try` block throws an exception, then the program skips ***the catch clause***.
  - If any of the code in a method throws an exception of a type other than the one named in the ***catch clause***, this method exits immediately.
  - Hopefully, one of its callers has already provided a catch clause for that type.

Catching Exceptions

```
public void read(String filename) {

    try {
        InputStream in = new FileInputStream(filename);
        int b;
        while ((b = in.read()) != -1) {
            process input...
        }
    }
    catch (IOException exception) {
        exception.printStackTrace();
    }
}
```

Pass the exception on to the caller

- **Another choice to handle the exceptions: do nothing at all and simply pass the exception on to the caller.** 也可以不在本方法内处理，而是传递给调用方，由**client**处理（“推卸责任”）
  - Let the caller of the read method worry about it!
  - Might be the best choice (prefer more correctness to robustness).
- **If we take that approach, then we have to advertise the fact that the method may throw an** IOException.

```
public void read(String filename) throws IOException {
     InputStream in = new FileInputStream(filename);
     int b;
     while ((b = in.read()) != -1) {...}
}
```

- **The compiler strictly enforces the** throws **specifiers. If you call a method that throws a checked exception, you must either handle it or pass it on.**

Try/catch and throw an exception?

- **As a general rule, you should catch those exceptions that you know how to handle and propagate those that you do not know how to handle.** 尽量在自己这里处理，实在不行就往上传——要承担责任！
- **When you propagate an exception, you must add a** `throws` **specifier to alert the caller that an exception may be thrown.** 但有些时候自己不知道如何处理，那么提醒上家，由**client**自己处理
- **Notice:**
  - If you are writing a method that overrides a superclass method which throws no exceptions, then you must catch each checked exception in the method's code. 如果父类型中的方法没有抛出异常，那么子类型中的方法必须捕获所有的**checked exception**——为什么？
  - You are not allowed to add more throws specifiers to a subclass method than are present in the superclass method. 子类型方法中不能抛出比父类型方法更多的异常！

Get detailed information from an `Exception`

- **The exception object may contain information about the nature of the exception.**

- **To find out more about the object, try** `e.getMessage()` **to get the detailed error message (if there is one)**
- **Use** `e.getClass().getName()` **to get the actual type of the exception object.**

Catching Multiple Exceptions

- **You can catch multiple exception types in a `try` block and handle each type differently.**
- **Use a separate `catch` clause for each type as in the following example:**

```
try {
    code that might throw exceptions
}
catch (FileNotFoundException e) {
    emergency action for missing files
}
catch (UnknownHostException e) {
    emergency action for unknown hosts
}
catch (IOException e) {
    emergency action for all other I/O problems
}
```

(8) Rethrowing and Chaining Exceptions

Rethrowing and Chaining Exceptions

- **You can throw an exception in a `catch` clause** 本来catch语句下面是用来做exception handling的，但也可以在catch里抛出异常

- **Typically, you do this when you want to change the exception type.**

- **If you build a subsystem that other programmers use, it makes a lot of sense to use an exception type that indicates a failure of the subsystem.**
  - E.g., `ServletException`, the code that executes a servlet may not want to know in minute detail what went wrong, but it definitely wants to know that the servlet was at fault.

- 这么做的目的是：更改**exception**的类型，更方便**client**端获取错误信息并处理

Rethrowing and Chaining Exceptions

- **Here is how you can catch an exception and rethrow it:**

```
try {
    access the database
}
catch (SQLException e) {
    throw new ServletException("database error: " + e.getMessage());
}
```

- **Here, the** `ServletException` **is constructed with the message text of the exception.**

Rethrowing and Chaining Exceptions

- **However, it is a better idea to set the original exception as the "cause" of the new exception** 但这么做的时候最好保留“根原因”

```
try {
    access the database
}
catch (SQLException e) {
    Throwable se = new ServletException("database error");
    se.initCause(e);
    throw se;
}
```

- **When the exception is caught, the original exception can be retrieved**

```
Throwable e = se.getCause();
```

- **This wrapping technique is highly recommended. It allows you to throw high level exceptions in subsystems without losing the details of the original failure.**

(9) `finally` Clause

finally Clause

- **When your code throws an exception, it stops processing the remaining code in your method and exits the method.** 当异常抛出时，方法中正常执行的代码被终止
- **This is a problem if the method has acquired some resource (files, database connections,…), which only this method knows about, and that resource must be cleaned up.** 如果异常发生前曾申请过某些资源，那么异常发生后这些资源要被恰当的清理
- **One solution is to catch and rethrow all exceptions.**
- **But this solution is tedious because you need to clean up the resource allocation in two places—in the normal code and in the exception code.**

- **Java has a better solution:** ***the*** `finally` ***clause.***

Try-Catch-Finally

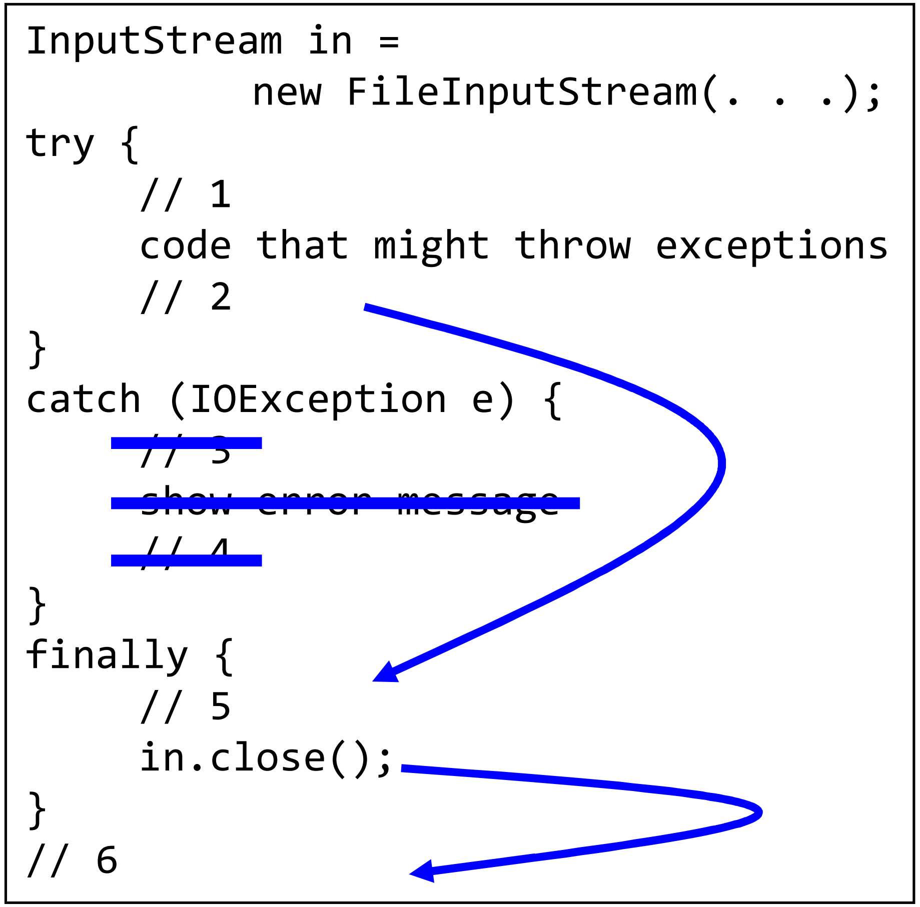
- **The code in the finally clause executes whether or not an exception was caught.**
- **In the following example, the program will close the file *under all circumstances*:**

```
InputStream in = new FileInputStream(. . .);
try {
     // 1
     code that might throw exceptions
     // 2
}
catch (IOException e) {
     // 3
     show error message
     // 4
}
finally {
     // 5
     in.close();
}
// 6
```

Try-Catch-Finally: case 1

不管程序是否碰到异常，finally都会被执行

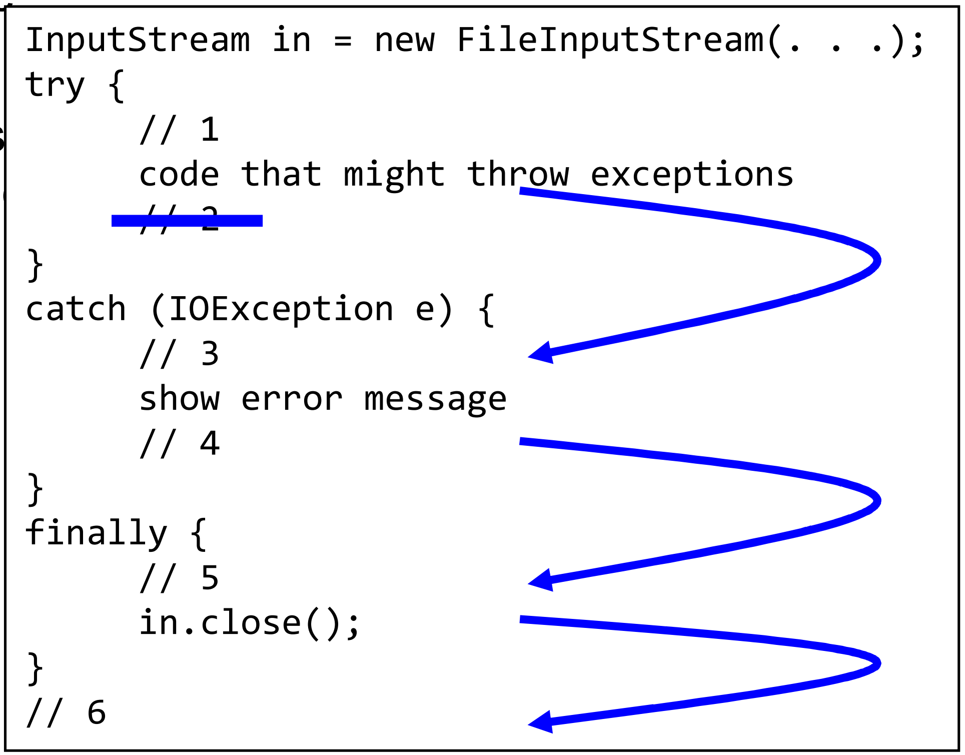
- **Three possible situations in which the program will execute the finally clause.**
- **Case 1: The code throws no exceptions.**
  - The program first executes all the code in the try block.
  - Then, it executes the code in the finally clause.
  - Afterwards, execution continues with the first statement after the finally clause.
  - In other words, execution passes through points 1, 2, 5, and 6.



```
InputStream in =
         new FileInputStream(. . .);
try {
     // 1
     code that might throw exceptions
     // 2
}
catch (IOException e) {
     ~~// 3~~
     ~~show error message~~
     ~~// 4~~
}
finally {
     // 5
     in.close();
}
// 6
```

`Try-Catch-Finally`: case 2

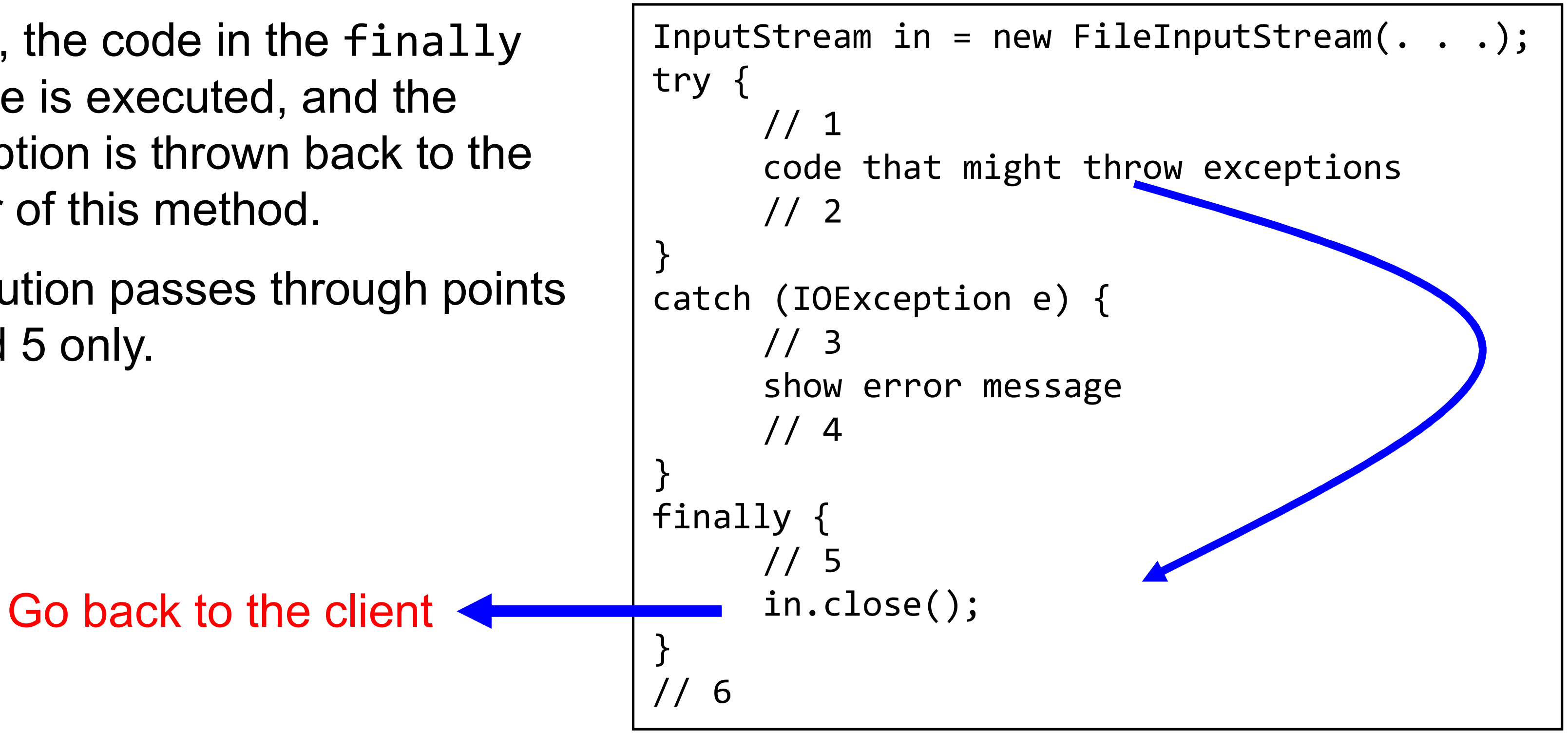
- **Case 2: The code throws an exception that is caught in a `catch` clause.**
  - The program executes all code in the `try` block, up to the point at which the exception was thrown. The remaining code in the `try` block is skipped. The program then executes the code in the matching `catch` clause, and then the code in the `finally` clause.
  - If the `catch` clause does not throw an exception, the program executes the first line after the `finally` clause.
  - Execution passes through points 1, 3, 4, 5, and 6.
  - If `catch` clause throws an exception, then the exception is thrown back to the caller, and execution passes through points 1, 3, and 5 only.



```
InputStream in = new FileInputStream(. . .);
try {
      // 1
      code that might throw exceptions
      ~~// 2~~
}
catch (IOException e) {
      // 3
      show error message
      // 4
}
finally {
      // 5
      in.close();
}
// 6
```

`Try-Catch-Finally`: case 3

- **Case3: The code throws an exception that is not caught in any catch clause.**
  - Here, the program executes all code in the `try` block until the exception is thrown.
  - The remaining code in the `try` block is skipped.
  - Then, the code in the `finally` clause is executed, and the exception is thrown back to the caller of this method.
  - Execution passes through points 1 and 5 only.



```
InputStream in = new FileInputStream(. . .);
try {
      // 1
      code that might throw exceptions
      // 2
}
catch (IOException e) {
      // 3
      show error message
      // 4
}
finally {
      // 5
      in.close();
}
// 6
```

Go back to the client

Using finally without a Catch

- You can use the finally clause without a catch clause.
- For example, consider the following try statement:

```
InputStream in = . . .;
try {
     code that might throw exceptions
}
finally {
     in.close();
}
```

- The in.close() statement in the finally clause is executed whether or not an exception is encountered in the try block.
- If an exception is encountered, it is rethrown and must be caught in another catch clause.

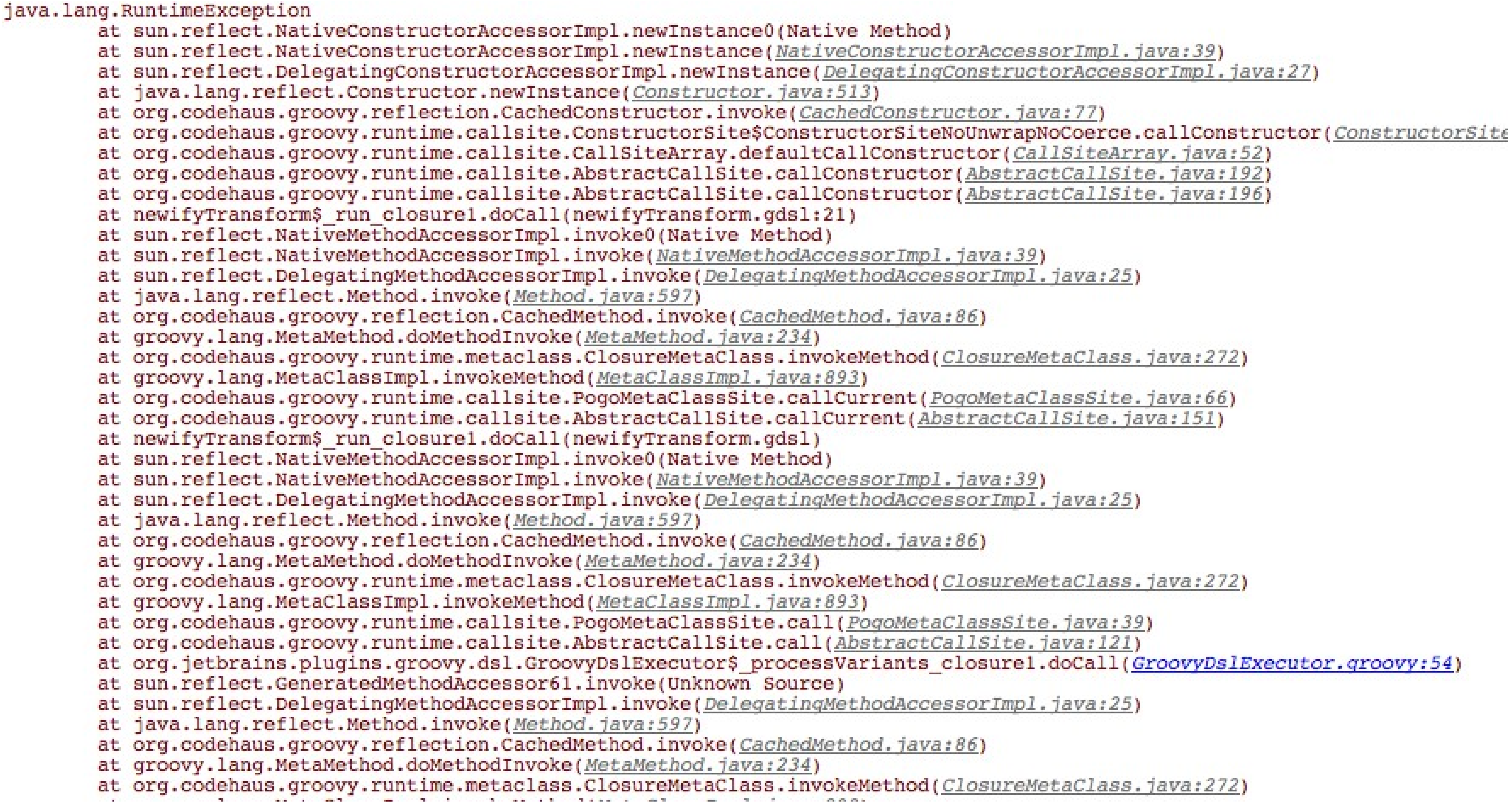
Problem

```
class Indecisive {

  public static void main(String[] args) {
    System.out.println(decision());
  }

  static boolean decision() {
    try {
      return true;
    } finally {
      return false;
    }
  }

}
```

The `finally` is processed after the `try`.

(10) Analyzing Stack Trace Elements

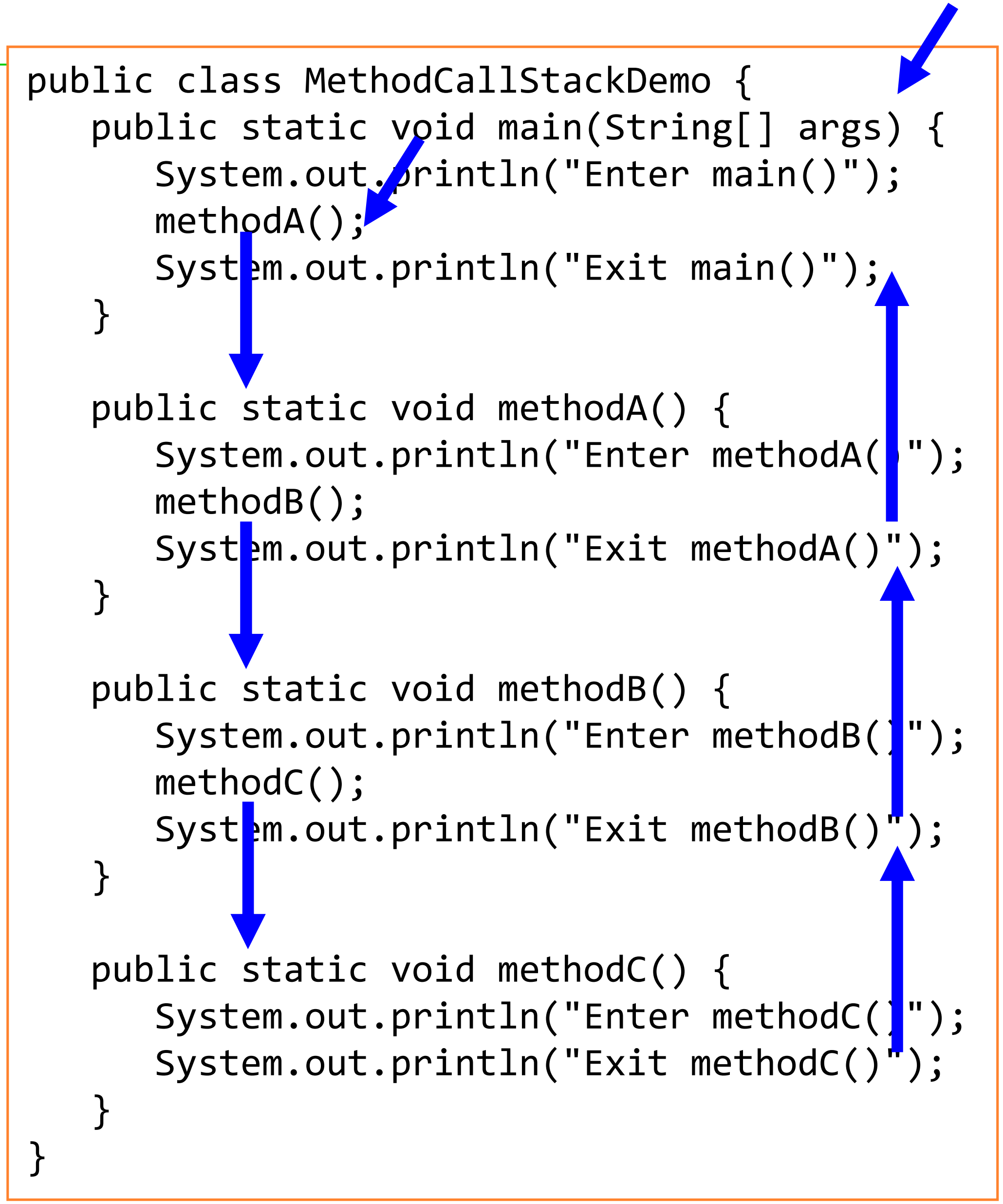
Stack trace (or call stack trace)



```
java.lang.RuntimeException
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:39)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:27)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:513)
        at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:77)
        at org.codehaus.groovy.runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.callConstructor(ConstructorSite
        at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:52)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:192)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:196)
        at newifyTransform$_run_closure1.doCall(newifyTransform.gdsl:21)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:86)
        at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:234)
        at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(ClosureMetaClass.java:272)
        at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:893)
        at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.callCurrent(PogoMetaClassSite.java:66)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callCurrent(AbstractCallSite.java:151)
        at newifyTransform$_run_closure1.doCall(newifyTransform.gdsl)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:86)
        at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:234)
        at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(ClosureMetaClass.java:272)
        at groovy.lang.MetaClassImpl.invokeMethod(MetaClassImpl.java:893)
        at org.codehaus.groovy.runtime.callsite.PogoMetaClassSite.call(PogoMetaClassSite.java:39)
        at org.codehaus.groovy.runtime.callsite.AbstractCallSite.call(AbstractCallSite.java:121)
        at org.jetbrains.plugins.groovy.dsl.GroovyDslExecutor$_processVariants_closure1.doCall(GroovyDslExecutor.groovy:54)
        at sun.reflect.GeneratedMethodAccessor61.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.codehaus.groovy.reflection.CachedMethod.invoke(CachedMethod.java:86)
        at groovy.lang.MetaMethod.doMethodInvoke(MetaMethod.java:234)
        at org.codehaus.groovy.runtime.metaclass.ClosureMetaClass.invokeMethod(ClosureMetaClass.java:272)
```

Method Call Stack

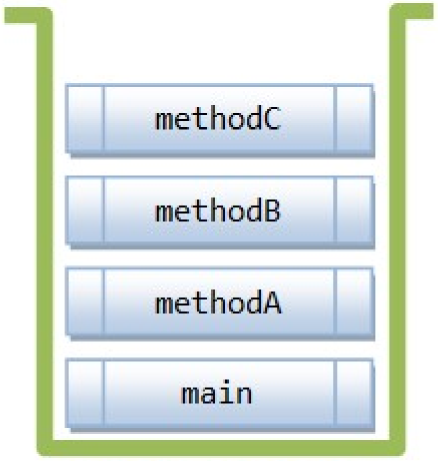
- **A typical application involves many levels of method calls, which is managed by a so-called method call stack.**
- **A stack is a last-in-first-out queue.**
- **What’s the result?**
  - `Enter main()`
  - `Enter methodA()`
  - `Enter methodB()`
  - `Enter methodC()`
  - `Exit methodC()`
  - `Exit methodB()`
  - `Exit methodA()`



```
public class MethodCallStackDemo {
   public static void main(String[] args) {
      System.out.println("Enter main()");
      methodA();
      System.out.println("Exit main()");
   }

   public static void methodA() {
      System.out.println("Enter methodA()");
      methodB();
      System.out.println("Exit methodA()");
   }

   public static void methodB() {
      System.out.println("Enter methodB()");
      methodC();
      System.out.println("Exit methodB()");
   }

   public static void methodC() {
      System.out.println("Enter methodC()");
      System.out.println("Exit methodC()");
   }
}
```

Method Call Stack

- **The sequence of events:**
  - JVM invoke the `main()`.
  - `main()` pushed onto call stack, before invoking `methodA()`.
  - `methodA()` pushed onto call stack, before invoking `methodB()`.
  - `methodB()` pushed onto call stack, before invoking `methodC()`.
  - `methodC()` completes.
  - `methodB()` popped out from call stack and completes.
  - `methodA()` popped out from the call stack and completes.
  - `main()` popped out from the call stack and completes. Program exits.



Method Call Stack
(Last-in-First-out Queue)

Call Stack Trace

- **Suppose** `methodC()` **carries out a "**`divide-by-0`**" operation, which triggers an** `ArithmeticException`**.**
- **The exception message clearly shows the *method call stack trace* with the relevant statement line numbers:**

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MethodCallStackDemo.methodC(MethodCallStackDemo.java:22)
        at MethodCallStackDemo.methodB(MethodCallStackDemo.java:16)
        at MethodCallStackDemo.methodA(MethodCallStackDemo.java:10)
        at MethodCallStackDemo.main(MethodCallStackDemo.java:4)
```

Call Stack Trace

- **Process:**
  - `MethodC()` triggers an `ArithmeticException`. As it does not handle this exception, it popped off from the call stack immediately.
  - `MethodB()` also does not handle this exception and popped off the call stack. So does `methodA()` and `main()` method.
  - The `main()` method passes back to JVM, which abruptly terminates the program and print the call stack trace.

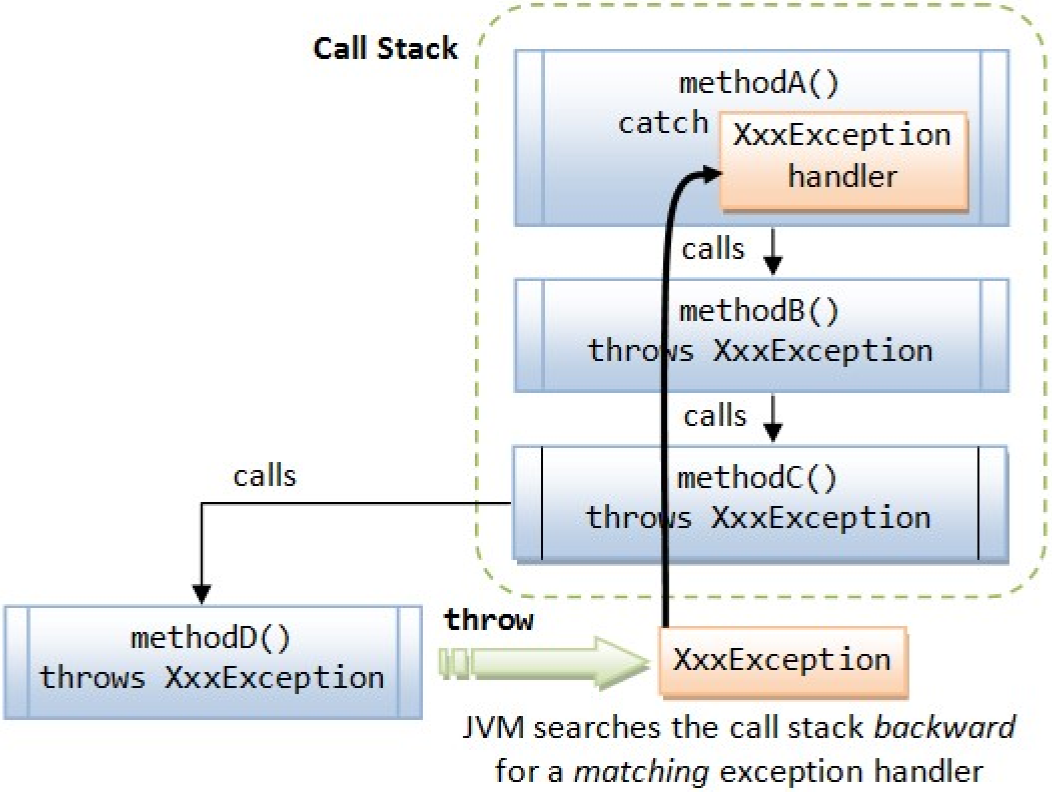
```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MethodCallStackDemo.methodC(MethodCallStackDemo.java:22)
        at MethodCallStackDemo.methodB(MethodCallStackDemo.java:16)
        at MethodCallStackDemo.methodA(MethodCallStackDemo.java:10)
        at MethodCallStackDemo.main(MethodCallStackDemo.java:4)
```

Exception & Call Stack

- **When an exception occurs inside a Java method, the method creates an `Exception` object and passes the `Exception` object to the JVM (i.e., the method "`throw`" an Exception).**
- **The `Exception` object contains the type of the exception, and the state of the program when the exception occurs.**
- **The JVM is responsible for finding an exception handler to process the Exception object.**
  - It searches backward through the call stack until it finds a matching exception handler for that particular class of `Exception` object (in Java term, it is called "`catch`" the Exception).
  - If the JVM cannot find a matching exception handler in all the methods in the call stack, it terminates the program.

Exception & Call Stack

- **Suppose that** `methodD()` **encounters an abnormal condition and throws a** `XxxException` **to the JVM.**
- **The JVM searches backward through the call stack for a matching exception handler**
- **It finds** `methodA()` **having a** `XxxException` **handler and passes the exception object to the handler.**
  - Notice that `methodC()` and `methodB()` are required to declare "`throws XxxException`" in their method signatures in order to compile the program.



JVM searches the call stack *backward* for a *matching* exception handler

Analyzing Stack Trace Elements

- **A *stack trace* is a listing of all pending method calls at a particular point in the execution of a program.**
- **You have almost certainly seen stack trace listings—they are displayed whenever a Java program terminates with an uncaught exception.**
- **You can access the text description of a stack trace by calling the `printStackTrace` method of the `Throwable` class.**

```
Throwable t = new Throwable();
StringWriter out = new StringWriter();
t.printStackTrace(new PrintWriter(out));
String description = out.toString();
```

Analyzing Stack Trace Elements

- **A more flexible approach is the `getStackTrace` method that yields an array of `StackTraceElement` objects, which you can analyze in your program.**

```
Throwable t = new Throwable();
StackTraceElement[] frames = t.getStackTrace();
for (StackTraceElement frame : frames)
    analyze frame
```

- **The `StackTraceElement` class has methods to obtain the file name and line number, as well as the class and method name, of the executing line of code.**
- **The `toString` method yields a formatted string containing all of this information.**

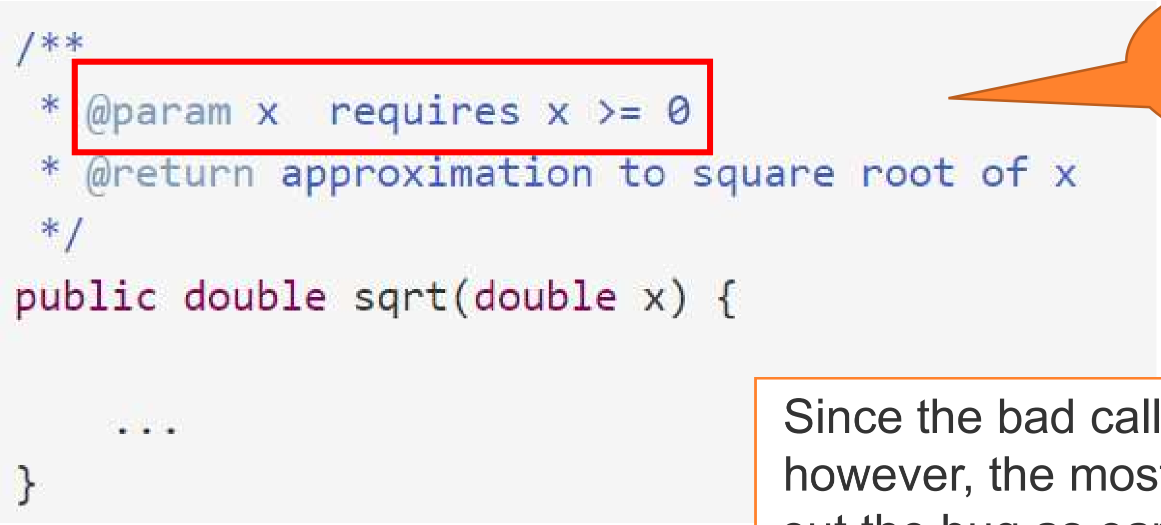
5 Assertions

First Defense: Make Bugs Impossible

- **The best defense against bugs is to make them impossible by design.** 最好的防御就是不要引入**bug** ☺
  - **Static checking**: eliminates many bugs by catching them at compile time.
  - **Dynamic checking**: Java makes array overflow bugs impossible by catching them dynamically. If you try to use an index outside the bounds of an array or a `List`, then Java automatically produces an error. --- unchecked exception / runtime error
  - **Immutability**: An immutable type is a type whose values can never change once they have been created.
  - **Immutable values**: by `final`, which can be assigned once but never reassigned.
  - **Immutable references:** by `final`, which makes the reference unreassignable, but the object that the reference points to may be mutable or immutable.

Second Defense: Localize Bugs

- If we can’t prevent bugs, we can **try to localize them to a small part of the program**, so that we don’t have to look too hard to find the cause of a bug. 如果无法避免，尝试着将bug限制在最小的范围内
  - When localized to a single method or small module, bugs may be found simply by studying the program text. 限定在一个方法内部，不扩散
  - **Fail fast**: the earlier a problem is observed (the closer to its cause), the easier it is to fix. 尽快失败，就容易发现、越早修复



```
/**
 * @param x  requires x >= 0
 * @return approximation to square root of x
 */
public double sqrt(double x) {

    ...
}
```

Pre-condition
如果违反，该方法
可以做任何事

Since the bad call indicates a bug in the caller, however, the most useful behavior would point out the bug as early as possible. 应该尽可能早的指出client的bug

Second Defense: Localize Bugs

- **Assertions 断言:** When the precondition is not satisfied, this code terminates the program by throwing an `AssertionError` exception. The effects of the caller's bug are prevented from propagating. Fail fast，避免扩散

- **Checking preconditions is an example of defensive programming** 检查前置条件是防御式编程的一种典型形式
  - Real programs are rarely bug-free.
  - Defensive programming offers a way to mitigate the effects of bugs even if you don't know where they are.

(1) What and Why Assertions?

What is assertion?

- **An assertion is code that's used during development that allows a program to check itself as it runs, i.e.,** to *test your assumptions* about your program logic (such as pre-conditions, post-conditions, and invariants). 断言：在开发阶段的代码中嵌入，检验某些“假设”是否成立。若成立，表明程序运行正常，否则表明存在错误。
  - When an assertion is true, that means everything is operating as expected.
  - When it's false, that means it has detected an unexpected error in the code.
- **An example**

```
public class AssertionTest {
   public static void main(String[] args) {
      int number = -5;    // assumed number is not negative
      // This assert also serve as documentation
      assert (number >= 0) : "number is negative: " + number;
      // do something
      System.out.println("The number is " + number);
   }
}
```

What is assertion?

- **Each assertion contains a** `boolean` **expression that you believe will be true when the program executes.**
  - If it is not true, the JVM will throw an `AssertionError`.
  - This error signals you that you have an invalid assumption that needs to be fixed. 出现AssertionError，意味着内部某些假设被违反了
  - The assertion confirms your assumptions about the behavior of your program, increasing your confidence that the program is free of errors. 增强程序员对代码质量的信心：对代码所做的假设都保持正确

- **Assertion is much better than using** `if-else` **statements, as it serves as proper documentation on your assumptions, and it does not carry performance liability in the production environment.**
  断言即是对代码中程序员所做假设的文档化，也不会影响运行时性能 (在实际使用时，**assertion**都会被**disabled**)

What is assertion?

- **An assertion usually takes two arguments**
  - **A boolean expression** that describes the assumption supposed to be true
  - **A message** to display if it isn't.
- **The Java language has a keyword** `assert` **with two forms:**
  - `assert condition;`
  - `assert condition : message;`
  - Both statements evaluate the condition and throw an `AssertionError` if the boolean expression evaluates to false.
  - In the second statement, the expression is passed to the constructor of the `AssertionError` object and turned into a message string. The description is printed in an error message when the assertion fails, so it can be used to provide additional details to the programmer about the cause of the failure.
    所构造的message在发生错误时显示给用户，便于快速发现错误所在

Assertion in Java

- To assert that x is non-negative, you can simply use the statement

```
assert x >= 0;
```

- Or pass the actual value of x into the `AssertionError` object, so that it gets displayed later:

```
assert x >= 0 : “x is ” + x;
```

- If `x == -1`, then this assertion fails with the error message

```
x is -1
```

- This information is often enough to get started in finding the bug.

Assertion Example

```java
public class AssertionSwitchTest {
   public static void main(String[] args) {
      // assumed either '+', '-', '*', '/' only
      char operator = '%';
      int operand1 = 5, operand2 = 6, result = 0;
      switch (operator) {
         case '+': result = operand1 + operand2; break;
         case '-': result = operand1 - operand2; break;
         case '*': result = operand1 * operand2; break;
         case '/': result = operand1 / operand2; break;
         default: assert false : "Unknown operator: " + operator;
      }
      System.out.println(operand1 + " " + operator + " "
              + operand2 + " = " + result);
   }
}
```

(2) What to Assert and What not to?

When use assertions?

- **Assertion can be used for verifying:**
  - **Internal Invariants** 内部不变量: Assert that a value is within a certain constraint, e.g., `assert x > 0`.
  - **Rep Invariants** 表示不变量: Assert that an object's state is within a constraint. What must be true about each instance of a class before or after the execution of a method? Class invariants are typically verified via private `boolean` method, e.g., `checkRep()`.
  - **Control-Flow Invariants** 控制流不变量: Assert that a certain location will not be reached. For example, the default clause of a `switch-case` statement.
  - **Pre-conditions of methods** 方法的前置条件: What must be true when a method is invoked? Typically expressed in terms of the method's arguments or the states of its objects.
  - **Post-conditions of methods** 方法的后置条件: What must be true after a method completes successfully?

What to Assert？

- **Pre-condition: method argument requirements**
- **Post-condition: Method return value requirements**
  - This kind of assertion is sometimes called a ***self check*** .

```
public double sqrt(double x) {
     assert x >= 0;
     double r;
     ... // compute result r
     assert Math.abs(r*r - x) < .0001;
     return r;
}
```

What to Assert？

- **Control-flow: covering all**
  – If a conditional statement or switch does not cover all the possible cases, it is good practice to use an assertion to block the illegal cases.

```
void foo() {
    for (...) {
      if (...)
        return;
    }
    assert false;
    // Execution should never reach this point!
}
```

```
switch (vowel) {
     case 'a':
     case 'e':
     case 'i':
     case 'o':
     case 'u': return "A";
     default: assert false;
}
```

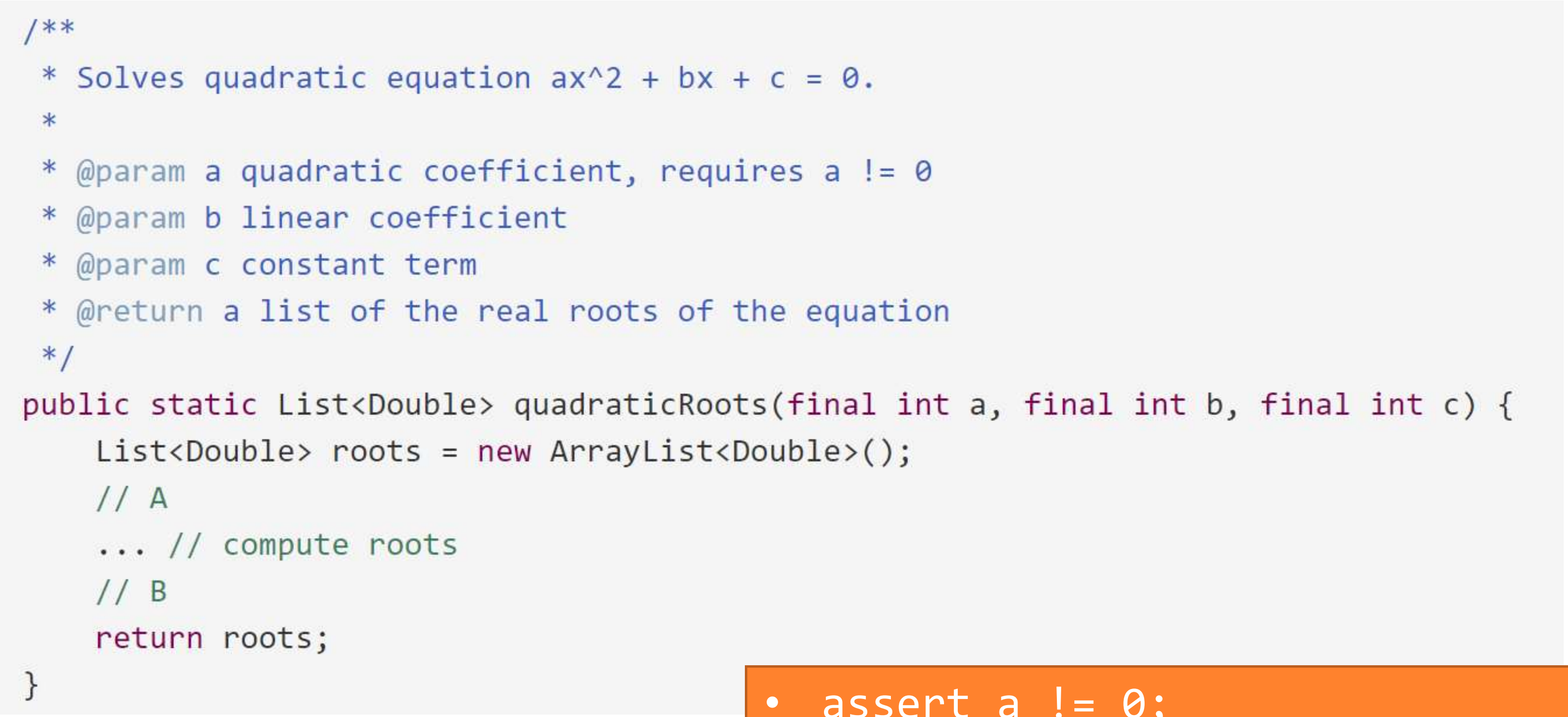
But don’t use the assert statement here, because it can be turned off. Instead, throw an exception in the illegal cases, so that the check will always happen:

```
    default: throw new AssertionError("must be a vowel, but was: "
                                       + vowel);
```

What to Assert: more scenarios

- **The value of an input-only variable is not changed by a method**
- **A pointer is non-NULL**
- **An array or other container passed into a method can contain at least X number of data elements**
- **A table has been initialized to contain real values**
- **A container is empty (or full) when a method begins executing (or when it finishes)**
- **The results from a highly optimized, complicated method match the results from a slower but clearly written routine**

Example



```
/**
 * Solves quadratic equation ax^2 + bx + c = 0.
 *
 * @param a quadratic coefficient, requires a != 0
 * @param b linear coefficient
 * @param c constant term
 * @return a list of the real roots of the equation
 */
public static List<Double> quadraticRoots(final int a, final int b, final int c) {
    List<Double> roots = new ArrayList<Double>();
    // A
    ... // compute roots
    // B
    return roots;
}
```

What statements would be reasonable to write at position A?

And at position B?

- `assert a != 0;`
- `assert b != 0;`
- `assert c != 0;`
- `assert roots.size() >= 0;`
- `assert roots.size() <= 2;`
- `for (double x : roots) { assert Math.abs(a*x*x + b*x + c) < 0.0001; }`

When to use assertions?

- **Normally, you don't want users to see assertion messages in production code; assertions are primarily for use during development and maintenance.** 断言主要用于开发阶段，避免引入和帮助发现**bug**
- **Assertions are normally compiled into the code at development time and compiled out of the code for production.** 实际运行阶段，不再使用断言
- **During production, they are compiled out of the code so that the assertions don't degrade system performance.** 避免降低性能
- **When should you write runtime assertions?** 使用断言的主要目的是为了在开发阶段调试程序、尽快避免错误
  - As you write the code, not after the fact. When you're writing the code, you have the invariants in mind.
  - If you postpone writing assertions, you're less likely to do it, and you're liable to omit some important invariants.

Avoid putting executable code in assertions

- **Since assertions may be disabled, the correctness of your program should never depend on whether or not the assertion expressions are executed.**
- **In particular, asserted expressions should not have *side-effects .***
  - For example, if you want to assert that an element removed from a list was actually found in the list, don't write it like this:
- **If assertions are disabled, the entire expression is skipped, and x is never removed from the list. Write it like this instead:**

```
// don't do this:
assert list.remove(x);
```

```
// do this:
boolean found = list.remove(x);
assert found;
```

Don’t Assert External Conditions

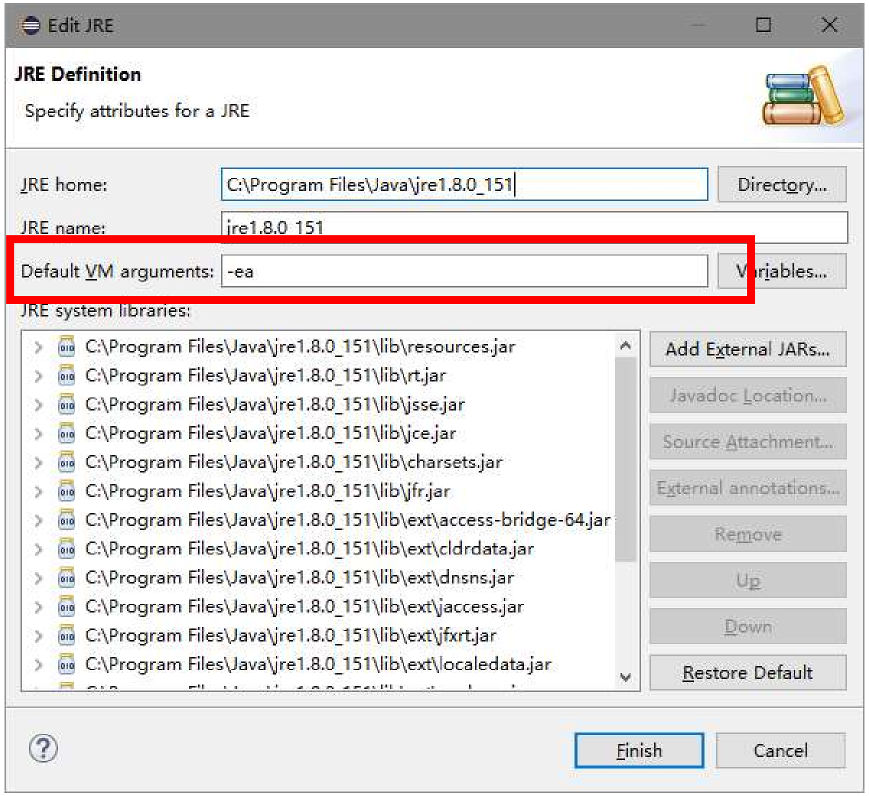
- **Never use assertions to test conditions that are external to your program.** 程序之外的事，不受你控制，不要乱断言
  - Such as the existence of files, the availability of the network, or the correctness of input typed by a human user. 文件/网络/用户输入等
  - Assertions test the internal state of your program to ensure that it is within the bounds of its specification. 断言只是检查程序的内部状态是否符合规约
  - When an assertion fails, it indicates that the program has run off the rails in some sense, into a state in which it was not designed to function properly. Assertion failures therefore indicate bugs. 断言一旦false，程序就停止执行
  - External failures are not bugs, and there is no change you can make to your program in advance that will prevent them from happening. 你的代码无法保证不出现此类外部错误 (recall section 7-2)
  - External failures should be handled using exceptions instead. Avoid trivial assertions, just as you would avoid uninformative comments. 外部错误要使用Exception机制去处理

Turn on/off Assert in different phases

- **Many assertion mechanisms are designed so that assertions are executed only during testing and debugging, and turned off when the program is released to users.**
  - Assertions are a great tool for keeping your code safe from bugs, but Java has them off by default! Java缺省关闭断言，要记得打开(-ea)
- **The advantage of this approach is that you can write very expensive assertions that would otherwise seriously degrade the performance of your program.** 断言非常影响运行时的性能
  - For example, a procedure that searches an array using binary search has a requirement that the array be sorted.
  - Asserting this requirement requires scanning through the entire array, however, turning an operation that should run in logarithmic time into one that takes linear time.
  - You should be willing to pay this cost during testing, since it makes debugging much easier, but not after the program is released to users.

Enable & Disable assertions in Java

- **Enable assertions**
  - Running the program with the `-enableassertions` or `-ea` option:
    - `java -enableassertions MyApp`
  - The option -ea... turns on assertions in all classes of the default package.
    - `java -ea:MyClass MyApp`
- **Disable assertions**
  - Running the program with the `-disableassertions` or `-da` option:
    - `java -ea:... -da:MyClass MyApp`
- **By default, assertions are disabled.**
- **Enable assertions in Eclipse:**
  - In preferences, go to Java → Installed JREs . Click “Java SE 8”, click “Edit…”, and in the “Default VM arguments” box enter: `-ea`



JRE Definition

Specify attributes for a JRE

JRE home: C:\Program Files\Java\jre1.8.0_151

JRE name: jre1.8.0 151

Default VM arguments: -ea

JRE system libraries:

```
C:\Program Files\Java\jre1.8.0_151\lib\resources.jar
C:\Program Files\Java\jre1.8.0_151\lib\rt.jar
C:\Program Files\Java\jre1.8.0_151\lib\jsse.jar
C:\Program Files\Java\jre1.8.0_151\lib\jce.jar
C:\Program Files\Java\jre1.8.0_151\lib\charsets.jar
C:\Program Files\Java\jre1.8.0_151\lib\jfr.jar
C:\Program Files\Java\jre1.8.0_151\lib\ext\access-bridge-64.jar
C:\Program Files\Java\jre1.8.0_151\lib\ext\cldrdata.jar
C:\Program Files\Java\jre1.8.0_151\lib\ext\dnsns.jar
C:\Program Files\Java\jre1.8.0_151\lib\ext\jaccess.jar
C:\Program Files\Java\jre1.8.0_151\lib\ext\jfxrt.jar
C:\Program Files\Java\jre1.8.0_151\lib\ext\localedata.jar
```

(3) Guidelines for Using Assertions

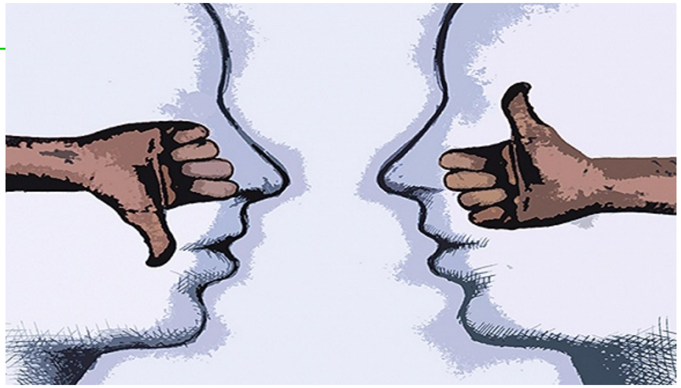
Assertion vs. Exception?

- **Assertions generally cover *correctness* issues of program.**
  - If an assertion is fired for an anomalous condition, the corrective action is not merely to handle an error gracefully—the corrective action is to change the program's source code, recompile, and release a new version of the software. 断言→Correctness
- **Exceptions generally cover *robustness* issues of program.**
  - If error handling code is used to address an anomalous condition, the error handling will enable the program to respond to the error gracefully. 错误/异常处理→Robustness
- **Assertions are especially useful in large, complicated programs and in high reliability programs.**
  - They enable programmers to more quickly flush out mismatched interface assumptions, errors that creep in when code is modified, and so on.

Assertion vs. Exception?

- **Use error handling code (exception) for conditions you expect to occur** 使用异常来处理你“预料到可以发生”的不正常情况
  - Error handling code checks for off-nominal circumstances that might not occur very often, but that have been anticipated by the programmer who wrote the code and that need to be handled by the production code.
- **Use assertions for conditions that should never occur** 使用断言处理“绝不应该发生”的情况
  - Assertions check for bugs in the code.

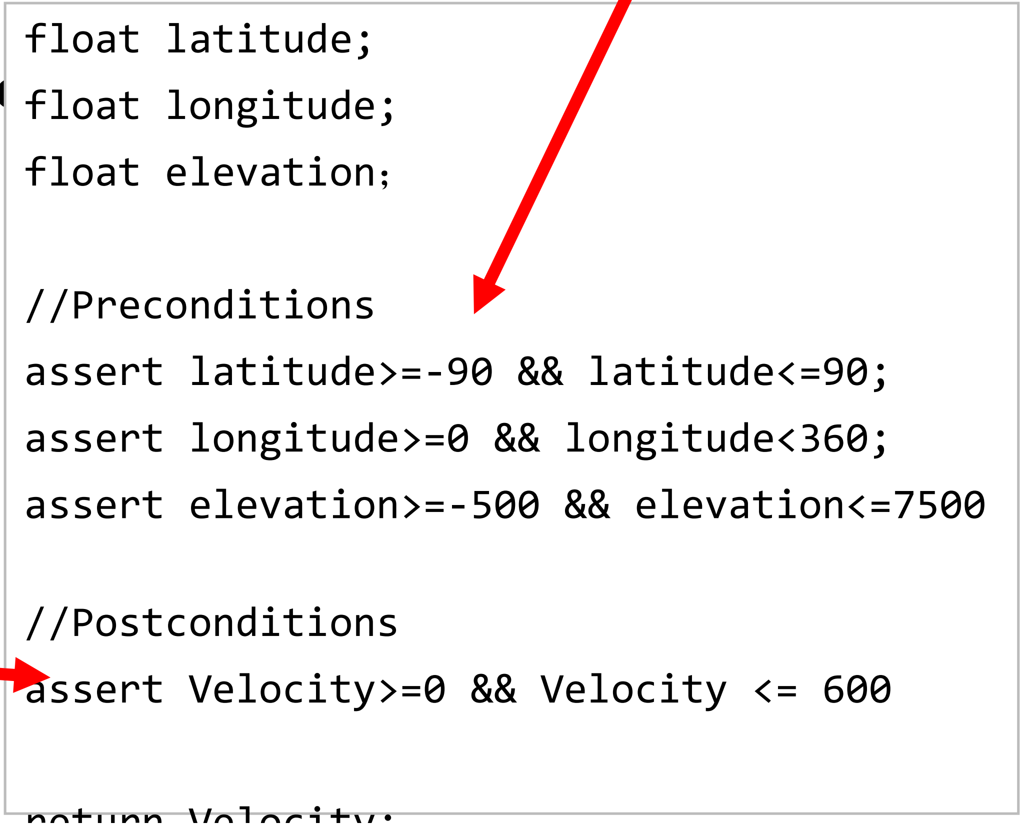
Should pre-/post-condition be asserted?



- **Another viewpoint: Do not use assertions for argument checking in public methods.** 在其他一些开发者眼里，不应该针对参数的合法性使用断言。
- **Reasons:**
  - Argument checking is typically part of the published specifications (or contract) of a method, and these specifications must be obeyed whether assertions are enabled or disabled. 不管是否-ea，spec中的pre-/post-conditions都能够被保证
  - Another problem with using assertions for argument checking is that erroneous arguments should result in an appropriate runtime exception (such as `IllegalArgumentException`, `IndexOutOfBoundsException`, or `NullPointerException`). An assertion failure will not throw an appropriate exception. 即使spec被违反，也不应通过assert直接fail，而是应抛出具体的runtime异常

Use Assertions for pre-/post- conditions

- If the variables `latitude`, `longitude`, and `elevation` were coming from an external source, invalid values should be checked and handled by error handling code rather than assertions. 如果参数来自于外部（不受自己控制），使用异常处理
- If the variables are coming from a trusted, internal source, however, and the routine's design is based on the assumption that these values will be within their valid ranges, then assertions are appropriate. 如果来自于自己所写的其他代码，可以使用断言来帮助发现错误（例如post-condition就需要）



```
float latitude;
float longitude;
float elevation;

//Preconditions
assert latitude>=-90 && latitude<=90;
assert longitude>=0 && longitude<360;
assert elevation>=-500 && elevation<=7500

//Postconditions
assert Velocity>=0 && Velocity <= 600


return Velocity;
```

Should pre-/post-condition be asserted?

```
/**
  * Sets the refresh rate.
  *
  * @param  rate refresh rate, in frames per second.
  * @throws IllegalArgumentException if rate <= 0 or
  * rate > MAX_REFRESH_RATE.
*/

public void setRefreshRate(int rate) {
  // Enforce specified precondition in public method
  if (rate <= 0 || rate > MAX_REFRESH_RATE)
    throw new IllegalArgumentException("Illegal rate: " + rate);

  setRefreshInterval(1000/rate);
}
```

Should pre-/post-condition be asserted?

- **You can use an assertion to test a nonpublic method's precondition that you believe will be true no matter what a client does with the class.**

```
/**
 * Sets the refresh interval (which must correspond to a legal
rate).
 *
 * @param  interval refresh interval in milliseconds.
*/

 private void setRefreshInterval(int interval) {
  // Confirm adherence to precondition in nonpublic method
  assert interval > 0
         && interval <= 1000/MAX_REFRESH_RATE : interval;

  ... // Set the refresh interval
 }
```

Should pre-/post-condition be asserted?

- **You can test postcondition with assertions in both public and nonpublic methods.**

```
/**
  * Returns a BigInteger whose value is (this-1 mod m).
  *
  * @param  m the modulus.
  * @return this-1 mod m.
  * @throws ArithmeticException  m <= 0, or this BigInteger
  *  has no multiplicative inverse mod m (that is, this BigInteger
  *  is not relatively prime to m).
  */
public BigInteger modInverse(BigInteger m) {
  if (m.signum <= 0)
    throw new ArithmeticException("Modulus not positive: " + m);
  ... // Do the computation
  assert this.multiply(result).mod(m).equals(ONE) : this;
  return result;
}
```

Combine assert & exception handling for robustness

- **Both assertions and exception handling code might be used to address the same error.** 断言和异常处理都可以处理同样的错误
  - In the source code for Microsoft Word, for example, conditions that should always be true are asserted, but such errors are also handled by error-handling code in case the assertion fails.
  - For extremely large, complex, long-lived applications like Word, assertions are valuable because they help to flush out as many development-time errors as possible.
- **But the application is so complex (million of lines of code) and has gone through so many generations of modification that it isn't realistic to assume that every conceivable error will be detected and corrected before the software ships, and so errors must be handled in the production version of the system as well.**

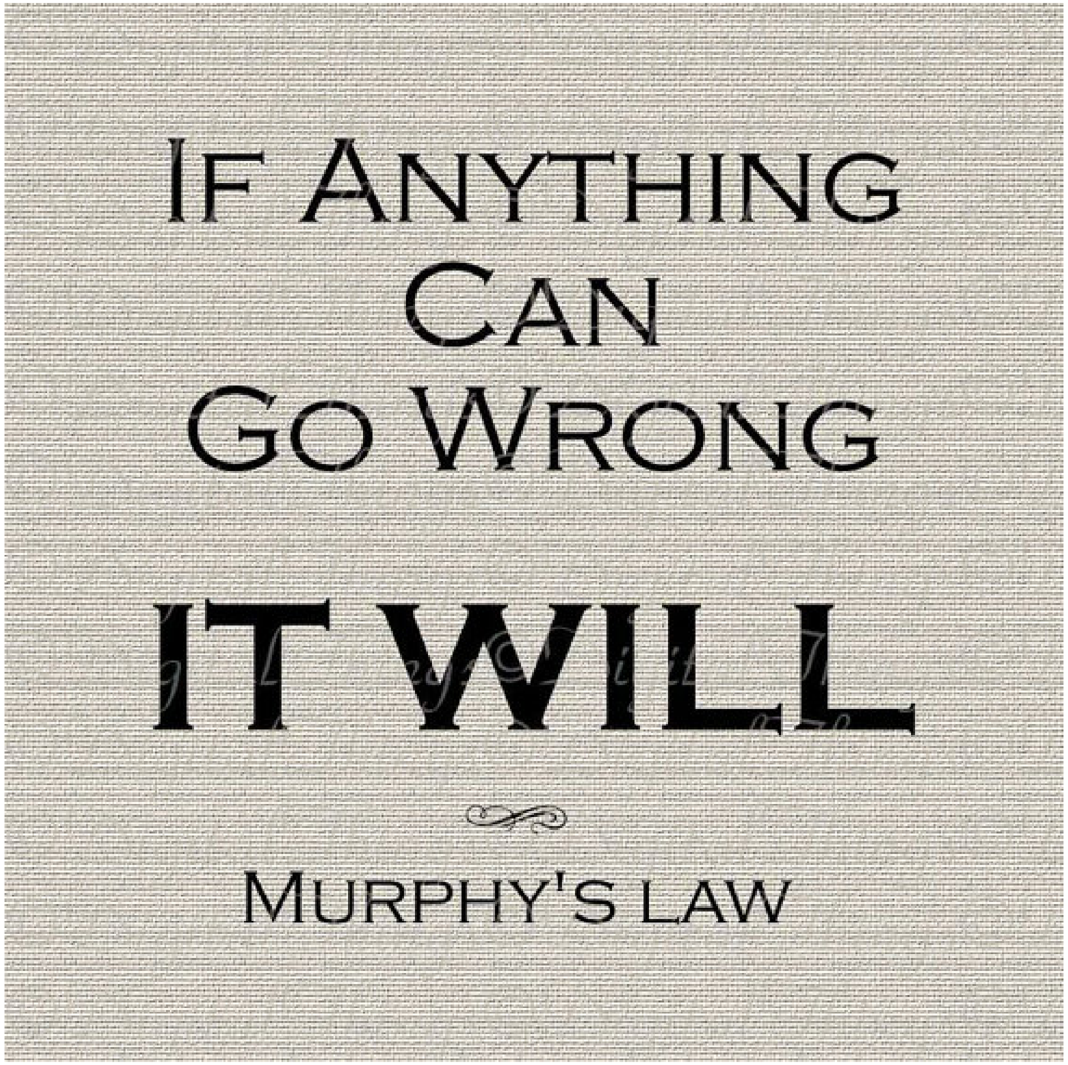
开发阶段用断言尽可能消除**bugs**
在发行版本里用异常处理机制处理漏掉的错误

6 Defensive Programming

What is defensive programming?

- **Defensive programming is a form of defensive design intended to ensure the continuing function of a piece of software under unforeseen circumstances.**
  - Defensive programming practices are often used where high availability, safety or security is needed.
- **The idea can be viewed as reducing or eliminating the prospect of Murphy's Law having effec**
- **"Your code should fail as early as possible!"**

Murphy’s Law



What is defensive programming?

- The idea is based on **defensive driving**
  - You adopt the mind-set that you're never sure what the other drivers are going to do.
  - That way, you make sure that if they do something dangerous you won't be hurt.
  - You take responsibility for protecting yourself even when it might be the other driver's fault.
  - 眼观六路，耳听八方，一旦其他车辆有对你产生危险的症状，马上采取防御式行动

Techniques for defensive programming

- **Protecting programs from invalid inputs**
- **Assertions**
- **Exceptions**
- **Specific error handling techniques**
- **Barricade**
- **Debugging aids**

- **The best form of defensive coding is not inserting errors in the first place.**
- **You can use defensive programming in combination with the other techniques.**

(1) Protecting Programs From Invalid Inputs

- **"Garbage in, garbage out"**
  - That expression is essentially software development's version of caveat emptor: let the user beware. 货物出门概不退换
- **For production software, garbage in, garbage out isn't good enough.**
- **A good program never puts out garbage, regardless of what it takes in.**
  - "Garbage in, nothing out"
  - "Garbage in, error message out"
  - "No garbage allowed in"

✓

- **"Garbage in, garbage out" is the mark of a sloppy, non-secure program.**

Protecting Programs From Invalid Inputs

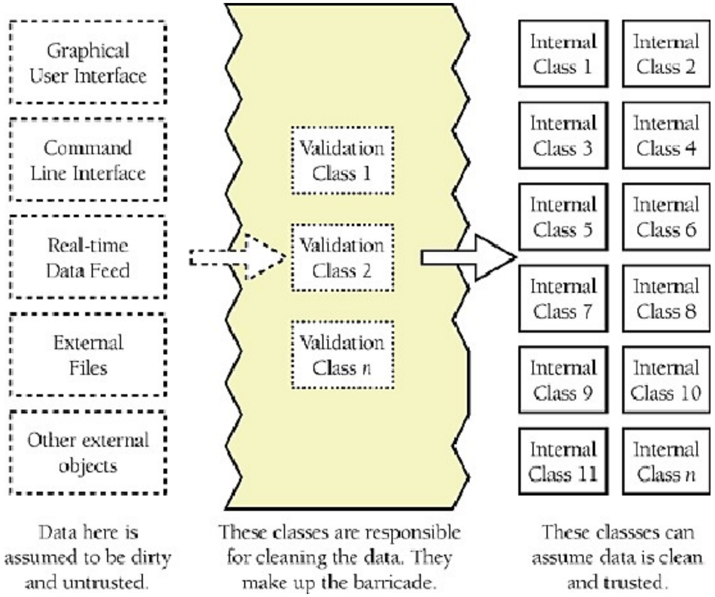
- **Check the values of all data from external sources** 对来自外部的数据源要仔细检查，例如：文件、网络数据、用户输入等
  - When getting data from a file, a user, the network, or some other external interface, check to be sure that the data falls within the allowable range.
- **Examples:**
  - Make sure that numeric values are within tolerances and that strings are short enough to handle.
  - If a string is intended to represent a restricted range of values (such as a financial transaction ID or something similar), be sure that the string is valid for its intended purpose; otherwise reject it.
  - If you’re working on a secure application, be especially leery of data that might attack your system: attempted buffer overflows, injected SQL commands, injected html or XML code, integer overflows, and so on.

Protecting Programs From Invalid Inputs

- **Check the values of all routine input parameters** 对每个函数的输入参数合法性要做仔细检查，并决定如何处理非法输入
  - Checking the values of routine input parameters is essentially the same as checking data that comes from an external source, except that the data comes from another routine instead of from an external interface.
- **Decide how to handle bad inputs**
  - Once you’ve detected an invalid parameter, what do you do with it?
  - Depending on the situation, you might choose any of a dozen different approaches, which are described in detail later in this chapter.

(2) Barricade 设置路障

- **Barricades are a damage-containment strategy.**
  - The reason is similar to that for having isolated compartments in the hull of a ship and firewalls in a building.
  - One way to barricade for defensive programming purposes is to designate certain interfaces as boundaries to “safe” areas.
  - Check data crossing the boundaries of a safe area for validity and respond sensibly if the data isn’t valid.



Defining some parts of the software that work with dirty data and some that work with clean can be an effective way to relieve the majority of the code of the responsibility for checking for bad data.

Barricade

- **The class's public methods assume the data is unsafe, and they are responsible for checking the data and sanitizing it.** 类的**public**方法接收到的外部数据都应被认为是**dirty**的，需要处理干净再传递到**private**方法——隔离舱
  - Once the data has been accepted by the class's public methods, the class's private methods can assume the data is safe.
- **Another way is as an operating-room technique.** 操作间技术
  - Data is sterilized before it's allowed to enter the operating room. Anything that's in the operating room is assumed to be safe.
  - The key design decision is deciding what to put in the operating room, what to keep out, and where to put the doors—which routines are considered to be inside the safety zone, which are outside, and which sanitize the data.
  - The easiest way to do this is usually by sanitizing external data as it arrives, but data often needs to be sanitized at more than one level, so multiple levels of sterilization are sometimes required.

Relationship between Barricades and Assertions

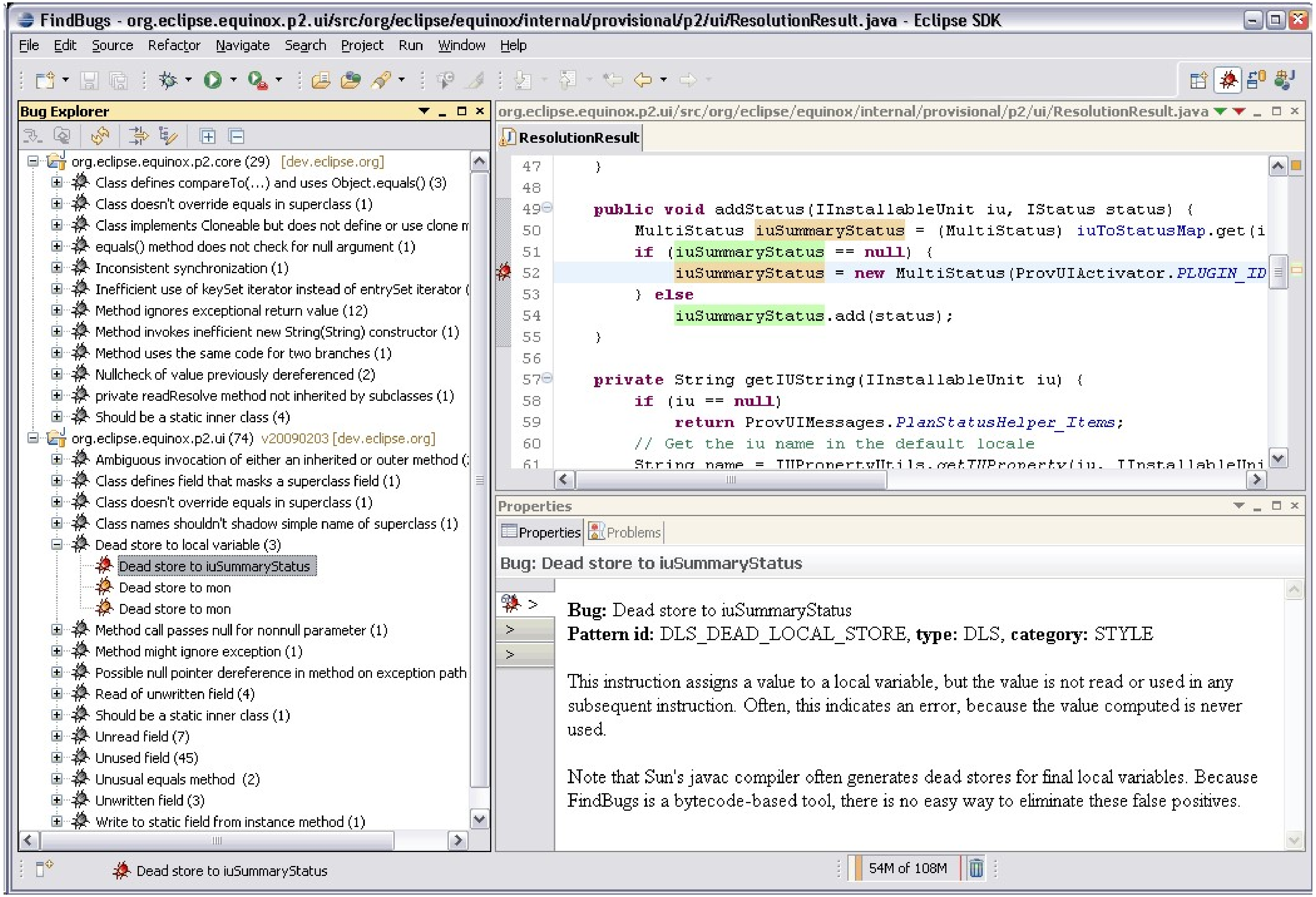
- **The use of barricades makes the distinction between assertions and error handling clean cut.**
  - Routines that are outside the barricade should use error handling because it isn’t safe to make any assumptions about the data.
  - Routines inside the barricade should use assertions, because the data passed to them is supposed to be sanitized before it’s passed across the barricade. If one of the routines inside the barricade detects bad data, that’s an error in the program rather than an error in the data.
- “隔离舱”外部的函数应使用异常处理，“隔离舱”内的函数应使用断言。
  - The use of barricades also illustrates the value of deciding at the architectural level how to handle errors.
  - Deciding which code is inside and which is outside the barricade is an architecture-level decision.
- **Proxy**设计模式？——隔离

7 The SpotBugs tool

早期版本：FindBugs

- **FindBugs** is a program which uses static analysis to look for bugs in Java code. Java静态代码分析工具
  - It operates on Java bytecode.
- Potential errors are classified in four ranks: scariest, scary, troubling and of concern. This is a hint to the developer about their possible impact or severity.
  - Bug list can be found in http://findbugs.sourceforge.net/bugDescriptions.html
- It is distributed as a stand-alone GUI application, but also plug-ins available for Eclipse, Gradle, Maven, and Jenkins.
  - http://findbugs.sourceforge.net/
  - http://findbugs.cs.umd.edu/eclipse/

FindBugs



FindBugs - org.eclipse.equinox.p2.ui/src/org/eclipse/equinox/internal/provisional/p2/ui/ResolutionResult.java - Eclipse SDK

Bug Explorer

- org.eclipse.equinox.p2.core (29) [dev.eclipse.org]
  - Class defines compareTo(...) and uses Object.equals() (3)
  - Class doesn't override equals in superclass (1)
  - Class implements Cloneable but does not define or use clone m
  - equals() method does not check for null argument (1)
  - Inconsistent synchronization (1)
  - Inefficient use of keySet iterator instead of entrySet iterator (
  - Method ignores exceptional return value (12)
  - Method invokes inefficient new String(String) constructor (1)
  - Method uses the same code for two branches (1)
  - Nullcheck of value previously dereferenced (2)
  - private readResolve method not inherited by subclasses (1)
  - Should be a static inner class (4)
- org.eclipse.equinox.p2.ui (74) v20090203 [dev.eclipse.org]
  - Ambiguous invocation of either an inherited or outer method (
  - Class defines field that masks a superclass field (1)
  - Class doesn't override equals in superclass (1)
  - Class names shouldn't shadow simple name of superclass (1)
  - Dead store to local variable (3)
    - Dead store to iuSummaryStatus
    - Dead store to mon
    - Dead store to mon
  - Method call passes null for nonnull parameter (1)
  - Method might ignore exception (1)
  - Possible null pointer dereference in method on exception path
  - Read of unwritten field (4)
  - Should be a static inner class (1)
  - Unread field (7)
  - Unused field (45)
  - Unusual equals method (2)
  - Unwritten field (3)
  - Write to static field from instance method (1)

org.eclipse.equinox.p2.ui/src/org/eclipse/equinox/internal/provisional/p2/ui/ResolutionResult.java

ResolutionResult

```java
    }

    public void addStatus(IInstallableUnit iu, IStatus status) {
        MultiStatus iuSummaryStatus = (MultiStatus) iuToStatusMap.get(i
        if (iuSummaryStatus == null) {
            iuSummaryStatus = new MultiStatus(ProvUIActivator.PLUGIN_ID
        } else
            iuSummaryStatus.add(status);
    }

    private String getIUString(IInstallableUnit iu) {
        if (iu == null)
            return ProvUIMessages.PlanStatusHelper_Items;
        // Get the iu name in the default locale
        String name = IUPropertyUtils.getIUProperty(iu, IInstallableUni
```

Properties

Bug: Dead store to iuSummaryStatus

Bug: Dead store to iuSummaryStatus
Pattern id: DLS_DEAD_LOCAL_STORE, **type:** DLS, **category:** STYLE

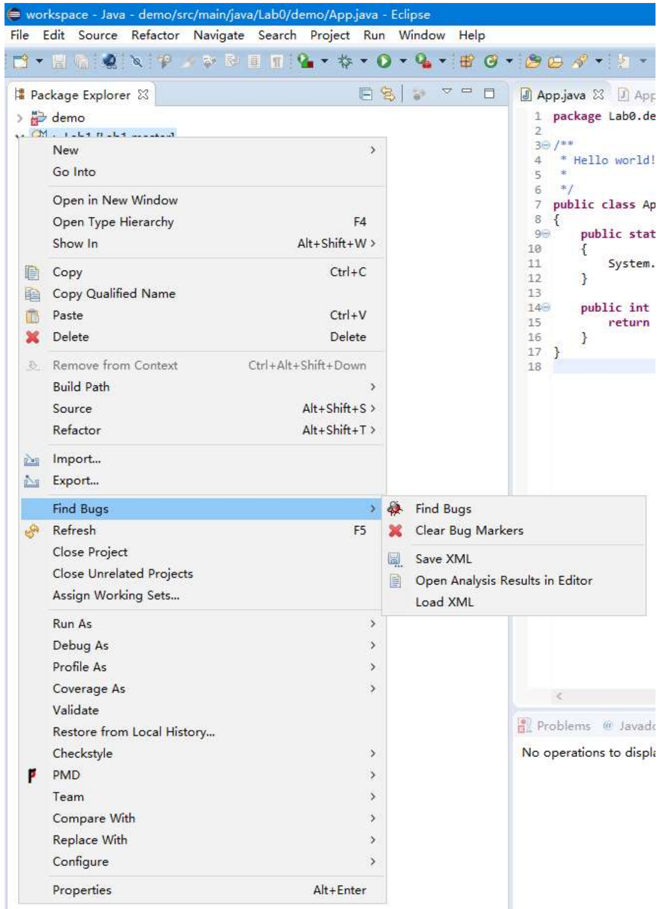
This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

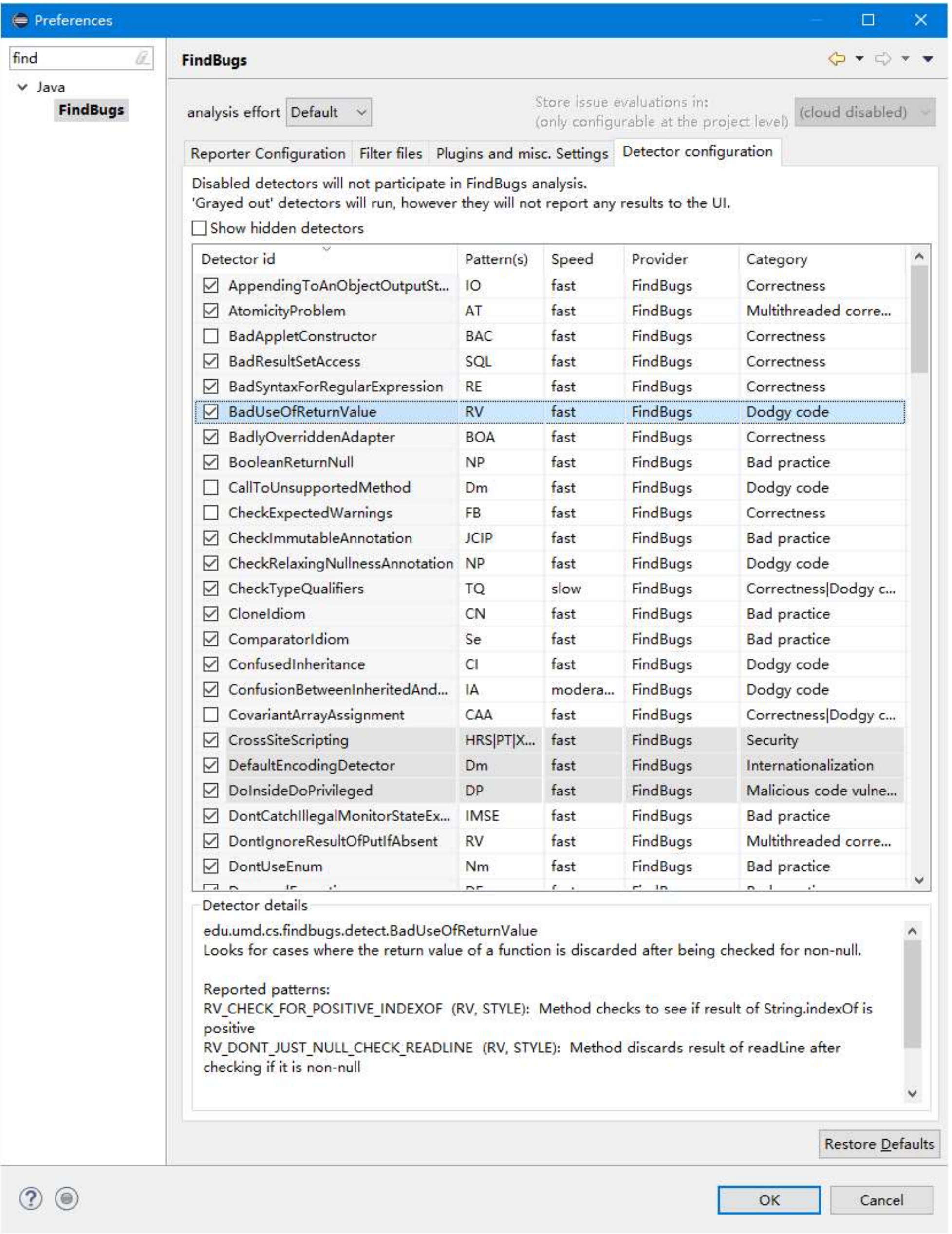
Dead store to iuSummaryStatus

54M of 108M

FindBugs



```
package Lab0.de
/**
 * Hello world!
 *
 */
public class Ap
{
    public stat
    {
        System.
    }

    public int
        return
    }
}
```



analysis effort: Default

Store issue evaluations in: (only configurable at the project level) (cloud disabled)

Disabled detectors will not participate in FindBugs analysis.
'Grayed out' detectors will run, however they will not report any results to the UI.

☐ Show hidden detectors

Detector id	Pattern(s)	Speed	Provider	Category
☑ AppendingToAnObjectOutputSt...	IO	fast	FindBugs	Correctness
☑ AtomicityProblem	AT	fast	FindBugs	Multithreaded corre...
☐ BadAppletConstructor	BAC	fast	FindBugs	Correctness
☑ BadResultSetAccess	SQL	fast	FindBugs	Correctness
☑ BadSyntaxForRegularExpression	RE	fast	FindBugs	Correctness
☑ BadUseOfReturnValue	RV	fast	FindBugs	Dodgy code
☑ BadlyOverriddenAdapter	BOA	fast	FindBugs	Correctness
☑ BooleanReturnNull	NP	fast	FindBugs	Bad practice
☐ CallToUnsupportedMethod	Dm	fast	FindBugs	Dodgy code
☐ CheckExpectedWarnings	FB	fast	FindBugs	Correctness
☑ CheckImmutableAnnotation	JCIP	fast	FindBugs	Bad practice
☑ CheckRelaxingNullnessAnnotation	NP	fast	FindBugs	Dodgy code
☑ CheckTypeQualifiers	TQ	slow	FindBugs	Correctness|Dodgy c...
☑ CloneIdiom	CN	fast	FindBugs	Bad practice
☑ ComparatorIdiom	Se	fast	FindBugs	Bad practice
☑ ConfusedInheritance	CI	fast	FindBugs	Dodgy code
☑ ConfusionBetweenInheritedAnd...	IA	modera...	FindBugs	Dodgy code
☐ CovariantArrayAssignment	CAA	fast	FindBugs	Correctness|Dodgy c...
☑ CrossSiteScripting	HRS|PT|X...	fast	FindBugs	Security
☑ DefaultEncodingDetector	Dm	fast	FindBugs	Internationalization
☑ DoInsideDoPrivileged	DP	fast	FindBugs	Malicious code vulne...
☑ DontCatchIllegalMonitorStateEx...	IMSE	fast	FindBugs	Bad practice
☑ DontIgnoreResultOfPutIfAbsent	RV	fast	FindBugs	Multithreaded corre...
☑ DontUseEnum	Nm	fast	FindBugs	Bad practice

Detector details

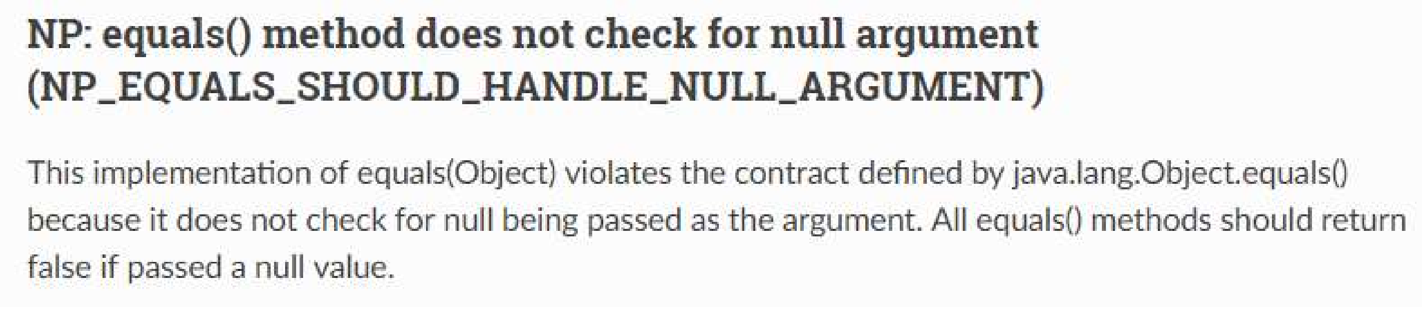
edu.umd.cs.findbugs.detect.BadUseOfReturnValue
Looks for cases where the return value of a function is discarded after being checked for non-null.

Reported patterns:
RV_CHECK_FOR_POSITIVE_INDEXOF (RV, STYLE): Method checks to see if result of String.indexOf is positive
RV_DONT_JUST_NULL_CHECK_READLINE (RV, STYLE): Method discards result of readLine after checking if it is non-null

SpotBugs

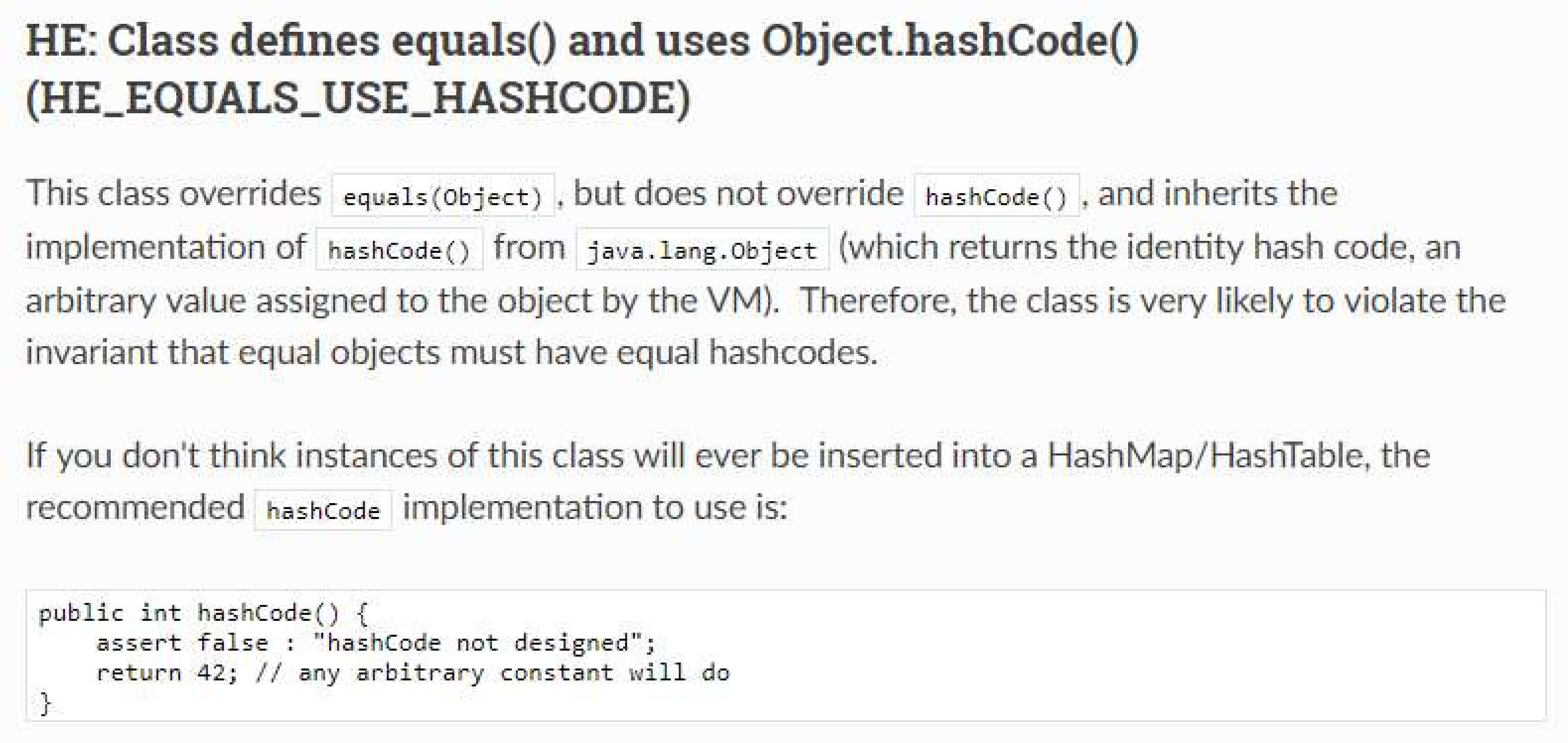
- **SpotBugs** is a program which uses static analysis to look for bugs in Java code.
  - It is the spiritual successor of FindBugs, carrying on from the point where it left off with support of its community.
- It checks for more than 400 bug patterns
  - https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html
- It can be used standalone and through several integrations, including: **Ant, Maven, Gradle, and Eclipse**
- 开源**Git**仓库：
  - https://github.com/spotbugs/spotbugs

Examples of SpotBugs patterns



NP: equals() method does not check for null argument (NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT)

This implementation of equals(Object) violates the contract defined by java.lang.Object.equals() because it does not check for null being passed as the argument. All equals() methods should return false if passed a null value.

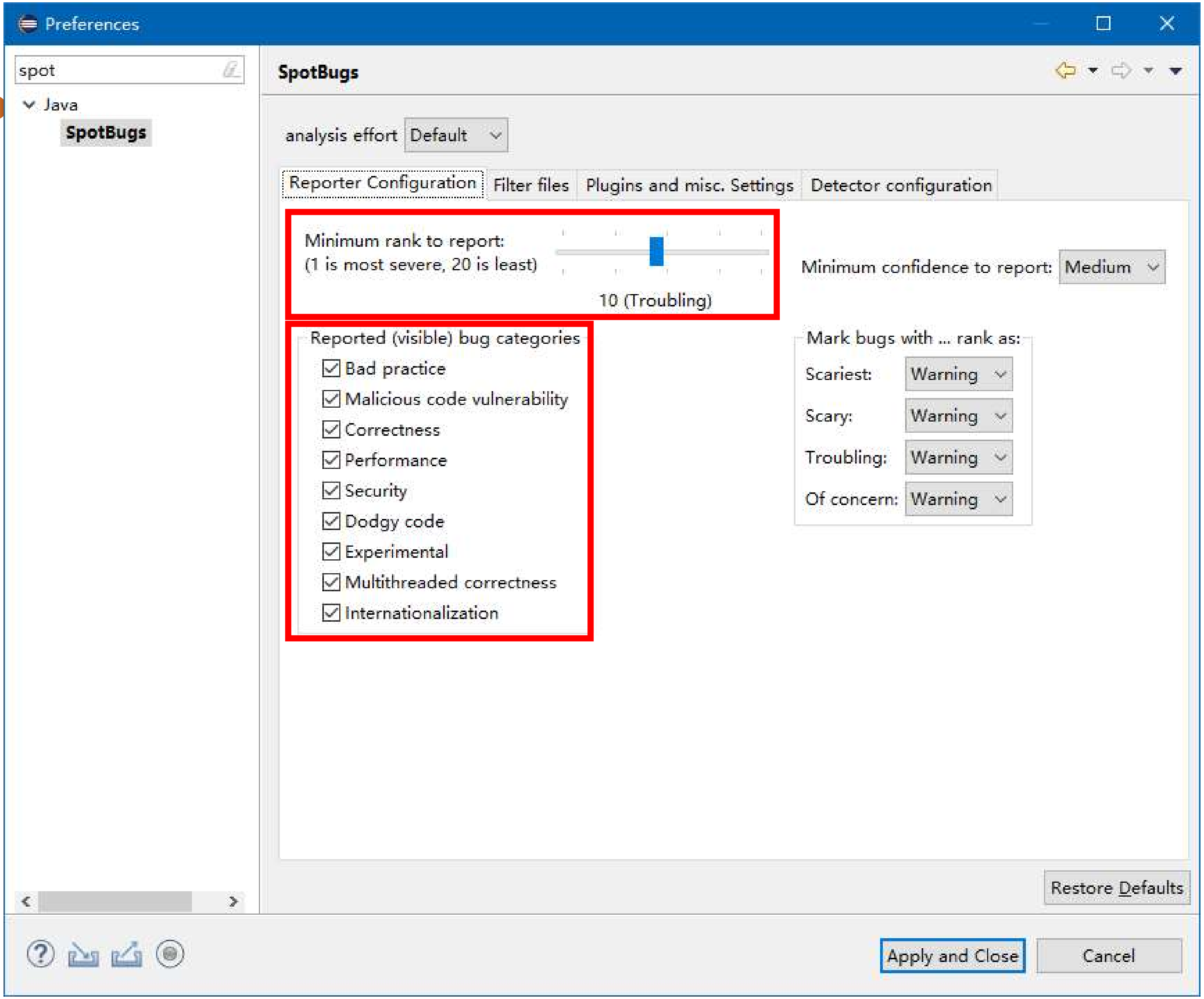


HE: Class defines equals() and uses Object.hashCode() (HE_EQUALS_USE_HASHCODE)

This class overrides `equals(Object)`, but does not override `hashCode()`, and inherits the implementation of `hashCode()` from `java.lang.Object` (which returns the identity hash code, an arbitrary value assigned to the object by the VM). Therefore, the class is very likely to violate the invariant that equal objects must have equal hashcodes.

If you don't think instances of this class will ever be inserted into a HashMap/HashTable, the recommended `hashCode` implementation to use is:

```
public int hashCode() {
    assert false : "hashCode not designed";
    return 42; // any arbitrary constant will do
}
```



SpotBugs

analysis effort Default

Minimum rank to report:
(1 is most severe, 20 is least)
10 (Troubling)

Minimum confidence to report: Medium

Reported (visible) bug categories

- ☑ Bad practice
- ☑ Malicious code vulnerability
- ☑ Correctness
- ☑ Performance
- ☑ Security
- ☑ Dodgy code
- ☑ Experimental
- ☑ Multithreaded correctness
- ☑ Internationalization

Mark bugs with ... rank as:

Scariest:	Warning
Scary:	Warning
Troubling:	Warning
Of concern:	Warning

Summary

Summary

- **What are Robustness and Correctness?**
- **How to measure Robustness and Correctness?**

- **Error and Exception in Java**
- **Exception Handling**
- **Assertions**
- **Defensive Programming**
- **The SpotBugs tool**

The end

June 23, 2021