Chapter 12 Exception Handling and Text I/O

异常的一个例子：运行时错误（runtime error）

```
import java.util.Scanner;
public class Quotient {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter two integers: ");
    int number1 = input.nextInt();
    int number2 = input.nextInt();
    System.out.println(number1 + " / " +
number2 + " is " + (number1 / number2));
  }
}
```

某两次运行结果

```
Enter two integers: 5 2 ↵Enter
5 / 2 is 2
```

```
Enter two integers: 3 0 ↵Enter
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Quotient.main(Quotient.java:11)
```

- 显然除数为0是不行的，所以程序引发异常后就崩溃了。
- 顺便提一下，如果是除数为0.0的浮点除法，例如**double f = 5.0/0.0;** 则不会引发异常，**f**会得到Infinity的特殊值。

使用if和方法来改写程序

- 为了让程序不至于异常退出，首先想到的方法是多一个判断，然后除数为0时提示并主动退出程序：

```
public static int quotient(int number1, int number2) {
  if (number2 == 0) {
    System.out.println("Divisor cannot be zero");
    System.exit(1); //主动退出程序
  }

  return number1 / number2;
}
```

```
Enter two integers: 5 3 ↵Enter
5 / 3 is 1
```

```
Enter two integers: 5 0 ↵Enter
Divisor cannot be zero
```

有没有更好的解决方法？

- 上述的方案并不完美，因为在方法中主动退出程序是大忌，相当于方法的编写者自作主张。退出程序的任务，应该始终由调用者决定。
- 现在问题来了，能否在不退出程序，并且hold住程序不挂掉的情况下，告诉调用者程序出错了呢？
- 异常处理终于出场了……

先改写求商的方法

```
public static int quotient(int number1, int number2) {
if (number2 == 0)
  throw new ArithmeticException("Divisor cannot be
zero");

  return number1 / number2;
}
```

```java
public static void main(String[] args) {
  Scanner input = new Scanner(System.in);
  // Prompt the user to enter two integers
  System.out.print("Enter two integers: ");
  int number1 = input.nextInt();
  int number2 = input.nextInt();

  try {
    int result = quotient(number1, number2);
    System.out.println(number1 + " / " + number2 + " is
" + result);
  } catch (ArithmeticException ex) {
    System.out.println("Exception: an integer "
    + "cannot be divided by zero ");
  }

  System.out.println("Execution continues ...");
}
```

某两次的执行结果

```
Enter two integers: 5 3 ↵Enter
5 / 3 is 1
Execution continues ...
```

```
Enter two integers: 5 0 ↵Enter
Exception: an integer cannot be divided by zero
Execution continues ...
```

- 可以看到程序通过了除数为0的考验。
- 原因在于除数为0时，引发了这个异常：**throw new** ArithmeticException(**"Divisor cannot be zero"**); 然后这个异常被程序自己捕获了（这点很重要，否则程序又要挂了）：
  catch (ArithmeticException ex)

总结一下

- 自己捕捉异常的代码模版如下：

```
try {
  Code to run;
  A statement or a method that may throw an exception;
  More code to run;
}
catch (type ex) {
  Code to process the exception;
}
```

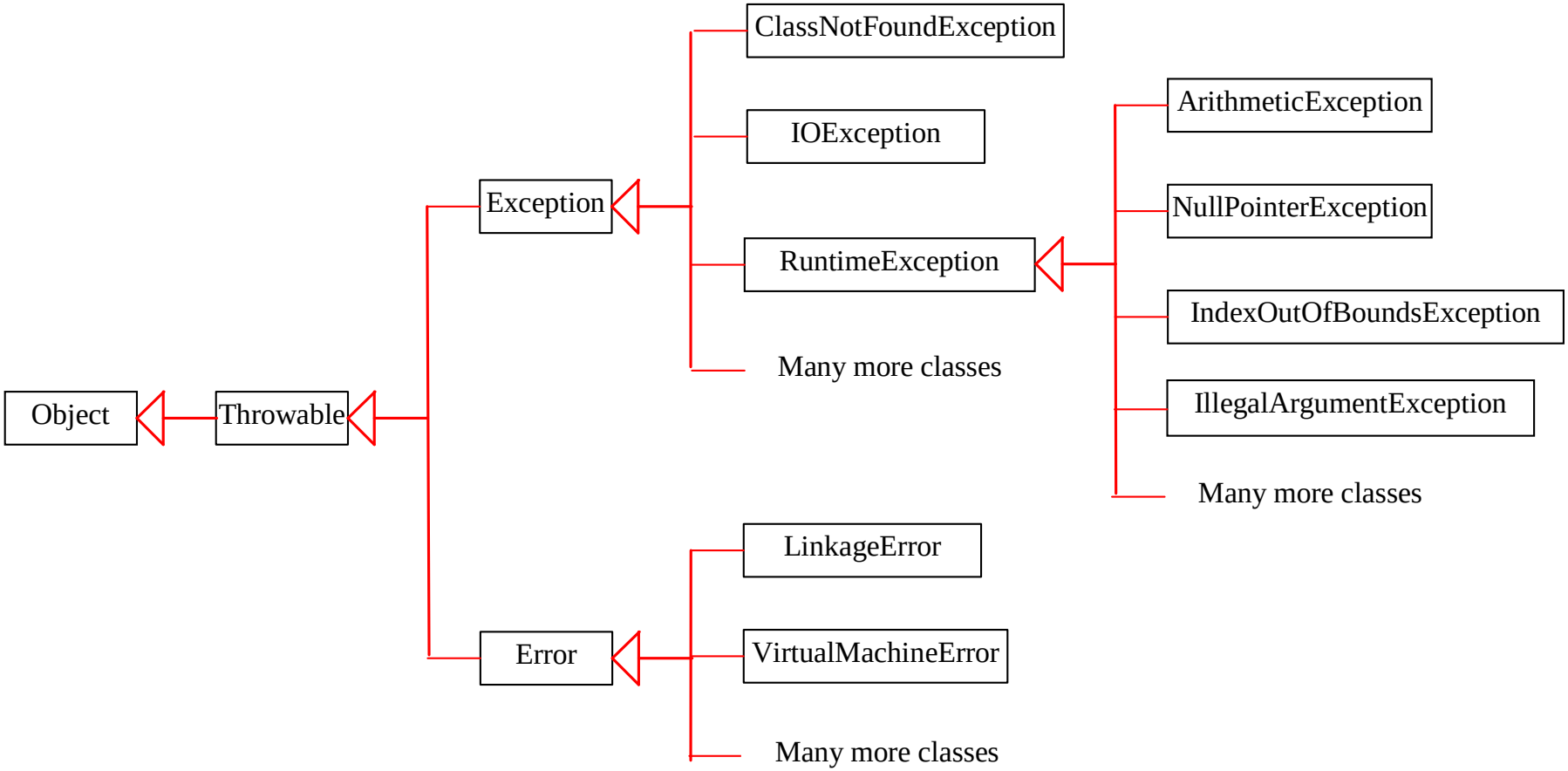
再来一个例子：输入不匹配异常

```java
import java.util.*;

public class InputMismatchExceptionDemo {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean continueInput = true;

    do {
      try {
        System.out.print("Enter an integer: ");
        int number = input.nextInt();

        // Display the result
        System.out.println(
          "The number entered is " + number);

        continueInput = false;
      }
      catch (InputMismatchException ex) {
        System.out.println("Try again. (" +
          "Incorrect input: an integer is required)");
        input.nextLine(); // Discard input
      }
    } while (continueInput);
  }
}
```

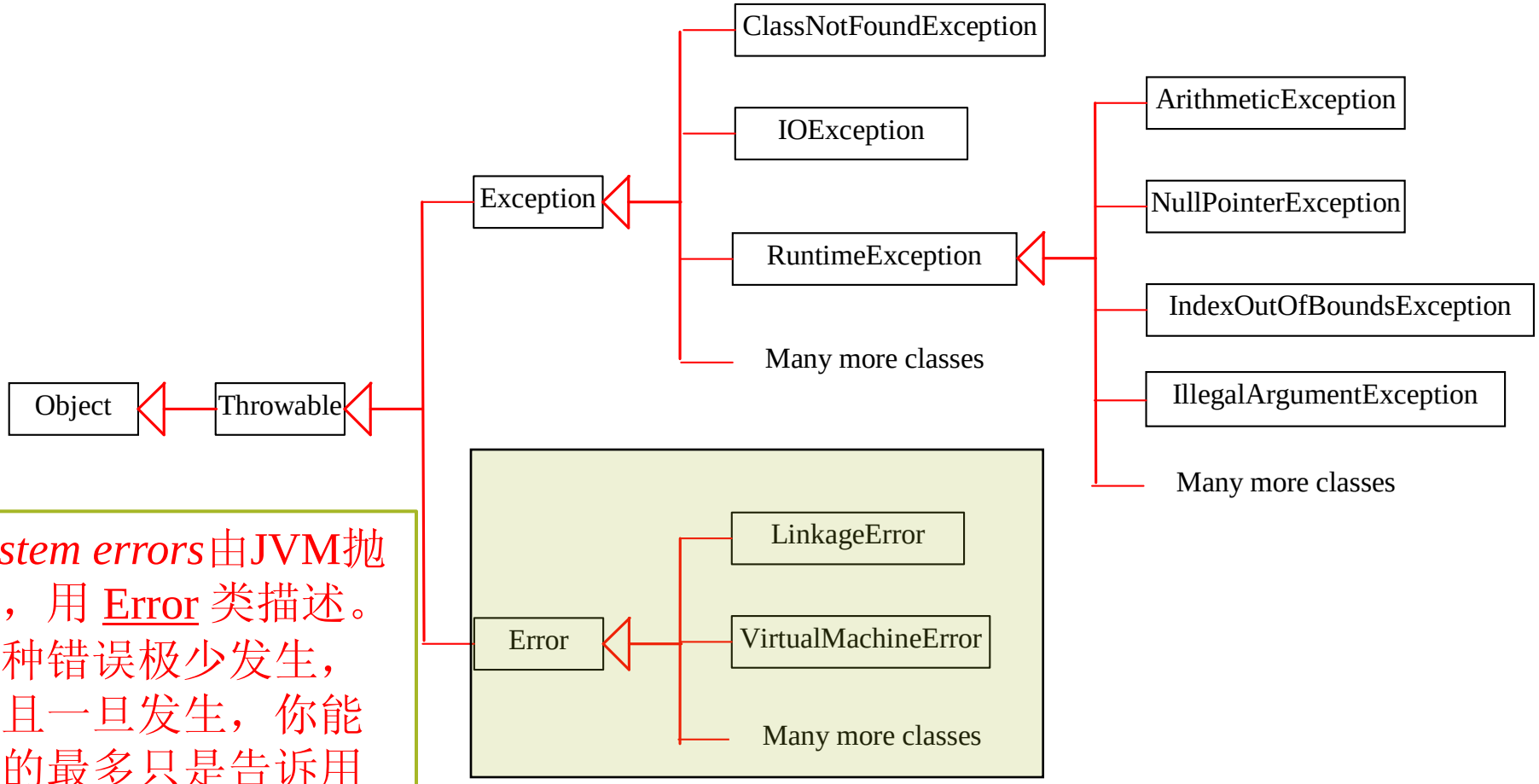
If an InputMismatchException occurs

```
Enter an integer: 3.5 ↵Enter
Try again. (Incorrect input: an integer is required)
Enter an integer: 4 ↵Enter
The number entered is 4
```

异常的类型


Object
Throwable
Exception
Error
ClassNotFoundException
IOException
RuntimeException
Many more classes
ArithmeticException
NullPointerException
IndexOutOfBoundsException
IllegalArgumentException
Many more classes
LinkageError
VirtualMachineError
Many more classes

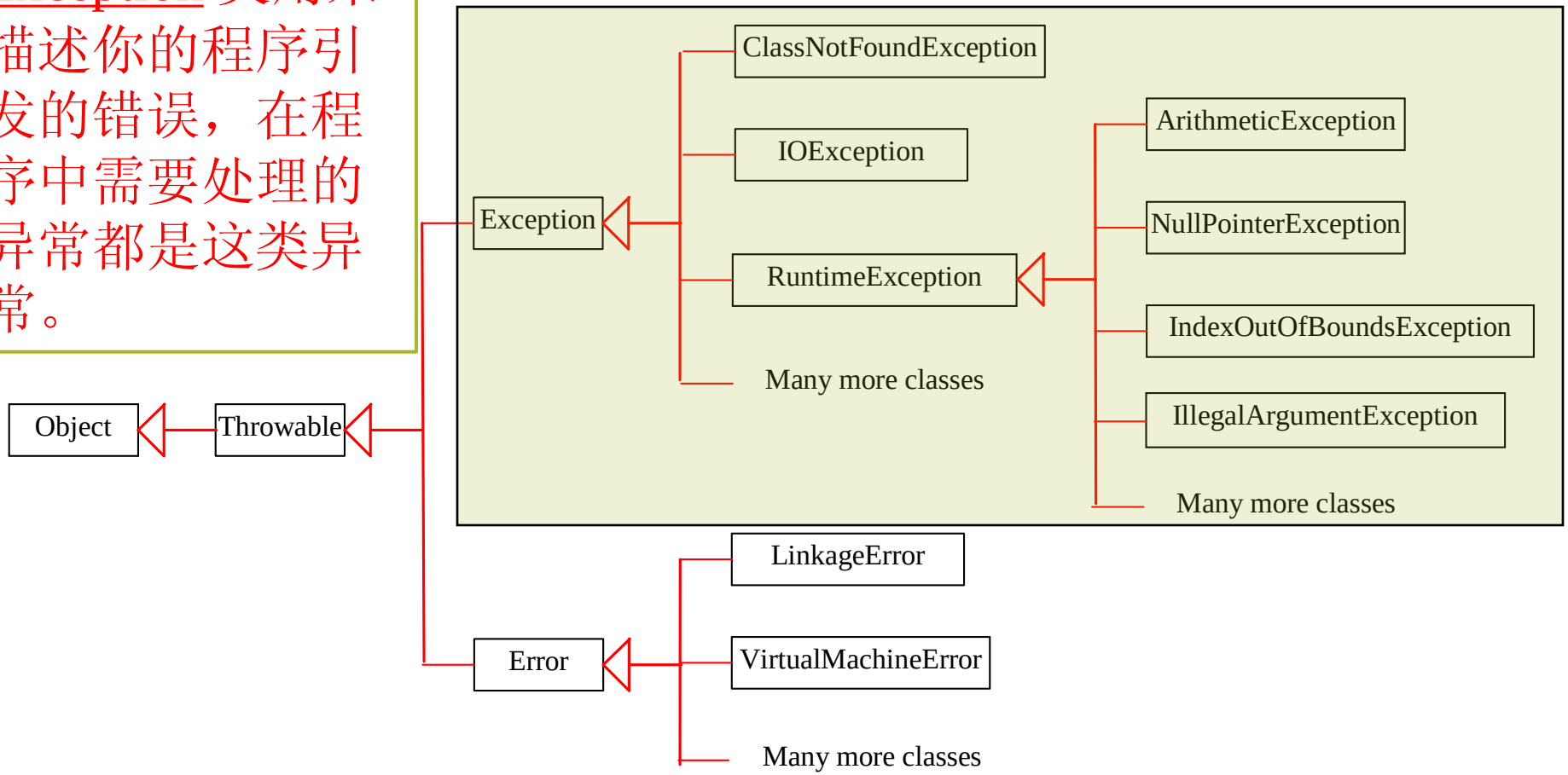
系统错误



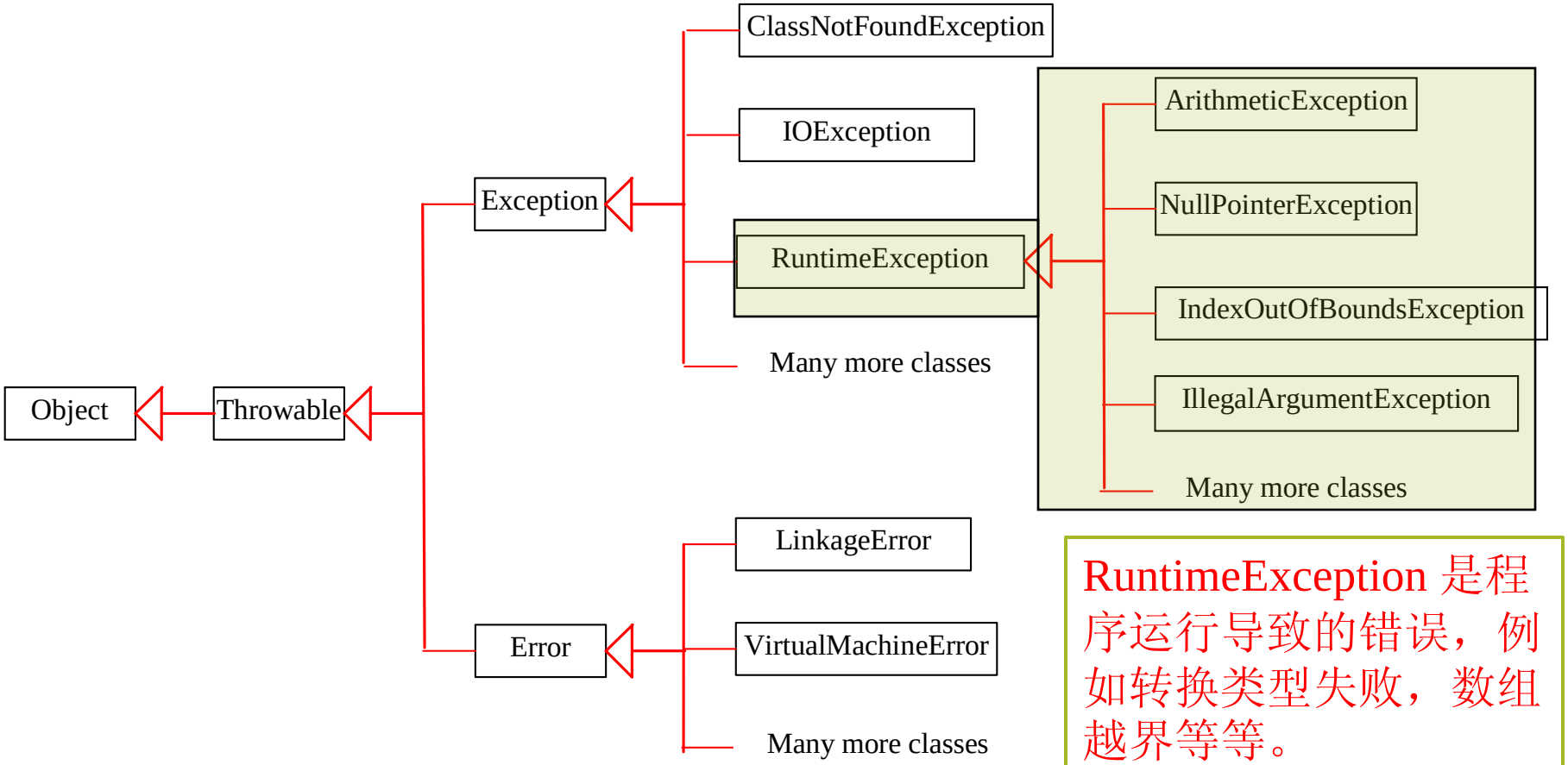
*System errors*由JVM抛出，用 Error 类描述。这种错误极少发生，并且一旦发生，你能做的最多只是告诉用户直接关闭程序重启机器再次尝试……

异常

Exception 类用来描述你的程序引发的错误，在程序中需要处理的异常都是这类异常。



运行时异常（Runtime Exceptions）



RuntimeException 是程序运行导致的错误，例如转换类型失败，数组越界等等。

受检异常（Checked Exceptions）VS. 非受检异常（Unchecked Exceptions）

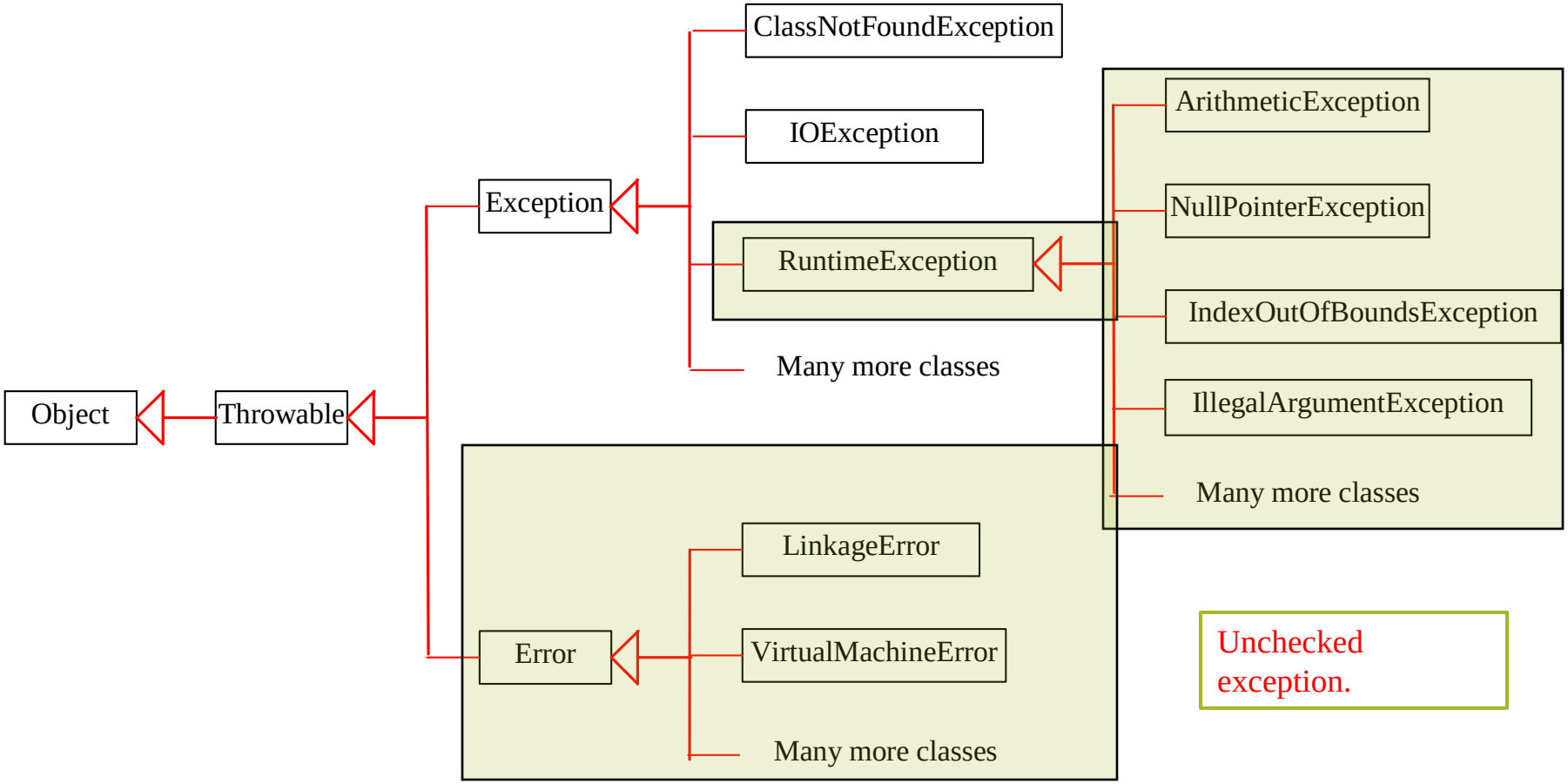
RuntimeException, Error以及它们的子类，都属于非受检异常，其余异常属于受检异常。

受检异常强制要求程序员在源代码中显式处理这些异常，否则编译器会报错。

非受检异常

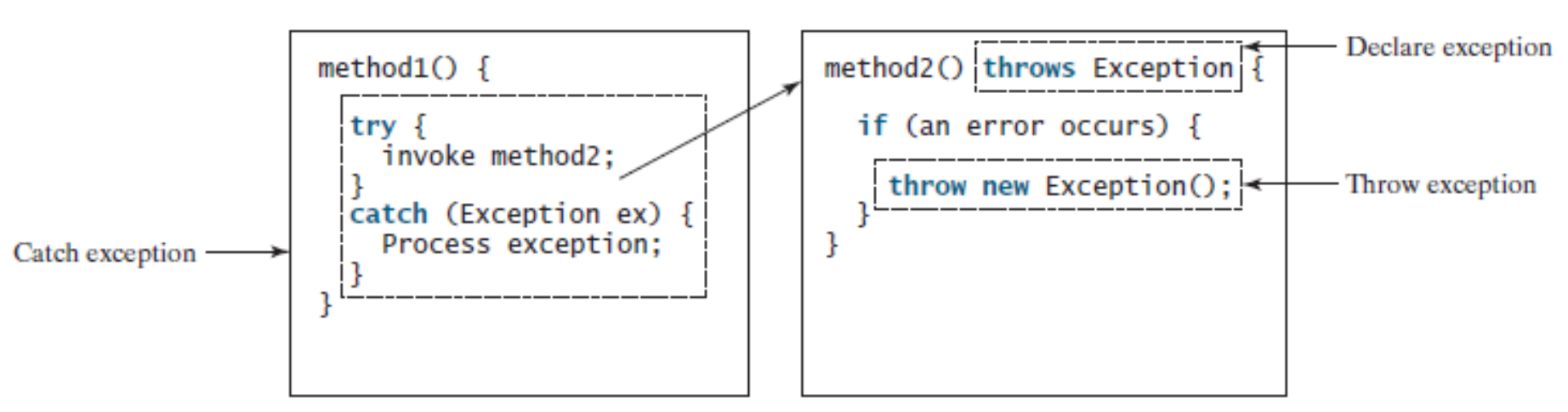
- ✓ 大部分情况下，非受检异常反映的是程序的逻辑错误，这是无法挽回的错误（除非修改源代码重编译，否则无法避免再次运行出错）。
- ✓ 例如空指针异常NullPointerException，表示你访问了一个空对象，下标越界IndexOutOfBoundsException，表示你的数组下标超出范围，这些逻辑错误应该在开发过程中就修改。
- ✓ 如果全部借助try-catch机制来保证程序的稳定是不现实的，不但代码冗长并且效率低下，所以 Java不会强制你捕捉这类异常。

非受检异常



Unchecked exception.

声明，抛出及捕捉异常


method1() {
try {
invoke method2;
}
catch (Exception ex) {
Process exception;
}
}
Catch exception
method2() throws Exception {
if (an error occurs) {
throw new Exception();
}
}
Declare exception
Throw exception

声明异常

每一个方法可以声明几种受检异常，以强制方法的调用者显式捕获这些异常。这种叫做声明异常（*declaring exceptions*）。下面是两个例子：

```
public void myMethod()
  throws IOException

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

抛出异常

当程序检测到异常时，可以创建一个能够恰当描述该异常的实例并将其抛出，至于由谁来处理这个异常，它就不管了。这种叫做抛出异常（*throwing an exception*），例如下面这两个例子：

```
throw new IllegalArgumentException("Wrong Argument");
```

或者：

```
IllegalArgumentException ex =
new IllegalArgumentException("Wrong Argument");
throw ex;
```

捕捉异常

- 抛出的异常一定要自己捕获，否则等到操作系统检测到异常，你的程序已经挂了。语法如下：

```
try {
  statements;
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

从JDK 1.7版本开始，不同类型的异常可以一起被活捉：

```
catch (Exception1
| Exception2 | ...
| Exceptionk ex) {
// Same code for
handling these
exceptions
}
```

异常捕捉的层次

- 假设method3引发异常，则异常的捕捉顺序是：Exception3, Exception2, Exception1，某一层方法捕捉失败，将导致该方法退栈，并由下一层继续尝试捕捉。

```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

An exception is thrown in method3

Call stack

捕捉或声明受检异常

Java强制你处理受检异常。如果一个方法声明为会抛出受检异常 (非 Error 或 RuntimeException 及其子类)，你必须显式写明 try-catch ，或者选择把这个异常继续向外抛出。例如，假设方法 p1 调用了 p2 ，并且 p2 可能抛出受检异常 IOException， 这时候你的代码只有写成(a)或(b)的形式，才能够通过编译。

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

例题：声明，抛出并捕捉异常

```java
public class CircleWithException {
  /** The radius of the circle */
  private double radius;

  /** The number of the objects created */
  private static int numberOfObjects = 0;

  /** Construct a circle with radius 1 */
  public CircleWithException() {
    this(1.0);
  }

  /** Construct a circle with a specified radius */
  public CircleWithException(double newRadius) {
    setRadius(newRadius);
    numberOfObjects++;
  }

  /** Return radius */
  public double getRadius() {
    return radius;
  }

```

```java
    /** Set a new radius */
    public void setRadius(double newRadius)
        throws IllegalArgumentException {
      if (newRadius >= 0)
        radius = newRadius;
      else
        throw new IllegalArgumentException(
          "Radius cannot be negative");
    }

    /** Return numberOfObjects */
    public static int getNumberOfObjects() {
      return numberOfObjects;
    }

    /** Return the area of this circle */
    public double findArea() {
      return radius * radius * 3.14159;
    }
  }
```

测试程序和运行结果

```java
public class TestCircleWithException {
  public static void main(String[] args) {
    try {
      CircleWithException c1 = new CircleWithException(5);
      CircleWithException c2 = new CircleWithException(-5);
      CircleWithException c3 = new CircleWithException(0);
    }
    catch (IllegalArgumentException ex) {
      System.out.println(ex);
    }

    System.out.println("Number of objects created: " +
      CircleWithException.getNumberOfObjects());
  }
}
```

```
java.lang.IllegalArgumentException: Radius cannot be negative
Number of objects created: 1
```

再次抛出异常

```
try {
  statements;
}
catch(TheException ex) {
  perform operations before exits;
  throw ex;
}
```

finally子句

- finally的作用是确保语句总能被执行，无论异常是否发生。finally的语法如下：

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

单步执行一下

假设这堆语句没有引发异常。

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

单步执行一下

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

try中的语句执行完后，finally语句块会被执行。

单步执行一下

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

异常处理块执行完毕，进入下一个语句。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

假设语句2引发了Exception1的异常

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

异常被捕捉到，进入catch处理。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

finally语句块还是会被执行到。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

异常处理块执行完毕，进入下一个语句。注意语句3已被跳过。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

假设语句2引发Exception2类型的异常。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

异常被正确捕捉，进入相应处理。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

finally语句块还是会被执行。

单步执行一下

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

再次抛出异常，交给调用者处理。注意此时finally已经执行完毕。

关于异常处理

- 由于将出错处理的代码和常规代码分开，异常处理的可读性和可维护性都很好。
- 不过异常处理需要更多的运行资源和运行时间，因为它需要一个异常的实例，回滚调用堆栈，并调用出错处理方法。

何时需要主动抛出异常

- 异常是在方法中发生的。如果希望异常被调用者自行处理，那就需要创建一个异常并抛出；如果方法可以自己处理异常，那就不需要抛出异常。

何时需要使用异常处理

由于效率问题，try-catch块不应该到处使用。应该使用它来处理意外的错误条件。如果是简单的条件，最好不要使用，例如：

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

何时需要使用异常处理

因为出错的情况太简单，只有一种可能性，所以上面那个语句块最好写成下面这样：

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

File类

- 这个类提供平台无关的，针对文件的操作和文件属性的读取。例如删除文件，查看是否只读，查看文件路径及创建日期等。
- 注意File类提供的操作，都是在不需要打开文件就能够进行的。这个类不能读写文件内容。

一个例子

```
public class TestFileClass {
  public static void main(String[] args) {
    java.io.File file = new java.io.File("image/us.gif");
    System.out.println("Does it exist? " + file.exists());
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead());
    System.out.println("Can it be written? " + file.canWrite());
    System.out.println("Is it a directory? " + file.isDirectory());
    System.out.println("Is it a file? " + file.isFile());
    System.out.println("Is it absolute? " + file.isAbsolute());
    System.out.println("Is it hidden? " + file.isHidden());
    System.out.println("Absolute path is " +
      file.getAbsolutePath());
    System.out.println("Last modified on " +
      new java.util.Date(file.lastModified()));
  }
}
```

create a File
exists()
length()
canRead()
canWrite()
isDirectory()
isFile()
isAbsolute()
isHidden()
getAbsolutePath()
lastModified()

文件输入输出

- 可以用PrintWriter写文本文件。用法和System.out.print差不多，除了需要先打开文件。

java.io.PrintWriter	
`+PrintWriter(file: File)`	Creates a `PrintWriter` object for the specified file object.
`+PrintWriter(filename: String)`	Creates a `PrintWriter` object for the specified file-name string.
`+print(s: String): void`	Writes a string to the file.
`+print(c: char): void`	Writes a character to the file.
`+print(cArray: char[]): void`	Writes an array of characters to the file.
`+print(i: int): void`	Writes an `int` value to the file.
`+print(l: long): void`	Writes a `long` value to the file.
`+print(f: float): void`	Writes a `float` value to the file.
`+print(d: double): void`	Writes a `double` value to the file.
`+print(b: boolean): void`	Writes a `boolean` value to the file.
Also contains the overloaded `println` methods.	A `println` method acts like a `print` method; additionally, it prints a line separator. The line-separator string is defined by the system. It is `\r\n` on Windows and `\n` on Unix.
Also contains the overloaded `printf` methods.	The `printf` method was introduced in §4.6, "Formatting Console Output."

一个例子

```
import java.io.*;
public class WriteData {
  public static void main(String[] args) throws IOException {
    File file = new File("scores.txt");
    if (file.exists()) {
      System.out.println("File already exists");
      System.exit(1); //强行退出程序
    }
    PrintWriter output = new PrintWriter(file); //文件打开
    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);
    output.close(); //文件关闭
  }
}
```

文件scores.txt内容：
John T Smith 90
Eric K Jones 85

try-with-resources

- Java支持一种自动关闭资源（例如文件）的写法，格式为：

```
try (declare and create resources) {
  Use the resource to process the file;
}
```

- 小括号内是声明并打开资源的代码，大括号内是针对资源的操作。离开大括号后，资源自动关闭。

重写上一个例子

- 现在可以不用close语句了。这种写法特别适合总是忘记关闭文件的同学。

```
try (
  // Create a file
  java.io.PrintWriter output = new java.io.PrintWriter(file);
) {
  // Write formatted output to the file
  output.print("John T Smith ");
  output.println(90);
  output.print("Eric K Jones ");
  output.println(85);
}
```

declare/create resource

use the resouce

读文本文件

- 文本文件可以使用Scanner类读取。记得当初从键盘读取输入，我们也用过Scanner。是的，你没有看错，它们就是同一个Scanner。
- 从键盘读取输入，是这样子的：

```
Scanner input = new Scanner(System.in);
```

- 从文件读取输入，是这样子的：

```
Scanner input = new Scanner(new File(filename));
```

- 也就是说，Scanner的构造方法用于指定**源**。

读文件的例子

```
import java.util.Scanner;
import java.io.*;
public class ReadData {
  public static void main(String[] args) throws Exception {
    Scanner input = new Scanner(new File("scores.txt"));
    while (input.hasNext()) {
      String firstName = input.next();
      String mi = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      System.out.println(
        firstName + " " + mi + " " + lastName + " " + score);
    }
    input.close(); //关闭文件
  }
}
```

文件scores.txt内容：
John T Smith 90
Eric K Jones 85

例题：替换文本

- 写一个程序，将一个文本文件中的字符串替换成目标字符串，并将结果写入到一个新文件中。命令行格式为：

java ReplaceText sourceFile targetFile oldString newString

- 用法举例：

java ReplaceText FormatString.java t.txt StringBuilder StringBuffer

代码段一：检查命令行参数

```
import java.io.*;
import java.util.*;
public class ReplaceText {
 public static void main(String[] args) throws Exception {
  // Check command line parameter usage
  if (args.length != 4) {
   System.out.println(
    "Usage: java ReplaceText sourceFile targetFile oldStr newStr");
   System.exit(1);
  }
```

代码段二：检查文件

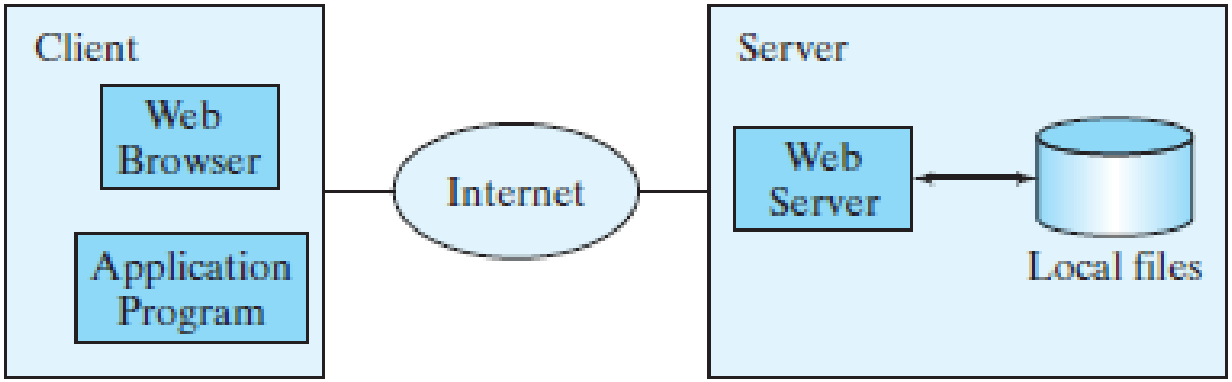
```
// Check if source file exists
File sourceFile = new File(args[0]);
if (!sourceFile.exists()) {
  System.out.println("Source file " + args[0] + " does not exist");
  System.exit(2);
}
// Check if target file exists
File targetFile = new File(args[1]);
if (targetFile.exists()) {
  System.out.println("Target file " + args[1] + " already exists");
  System.exit(3);
}
```

代码段三：读取、替换、写入

```
try ( // Create input and output files
      Scanner input = new Scanner(sourceFile);
      PrintWriter output = new PrintWriter(targetFile);
    ) {
    while (input.hasNext()) {
      String s1 = input.nextLine();
      String s2 = s1.replaceAll(args[2], args[3]);
      output.println(s2);
    }
  }
 }
}
```

从网上读取数据

- 其实，Scanner比你想象的强大，因为它还可以从网上读取数据，只要修改**源**就可以了。
- 当然，网络数据是以**流**（stream）的形式传输的，所以与打开文件的方式略有不同。传输示意如下：



URL (Uniform Resource Locator）

- 网络资源是用URL来定位的，所以读取之前，需要用URL指定**源**。例如：

```
try {
 URL url = new URL("http://www.xmu.edu.cn");
} catch (MalformedURLException ex) {
  ex.printStackTrace();
 }
```

- 然后，Scanner就可以读取这个源：

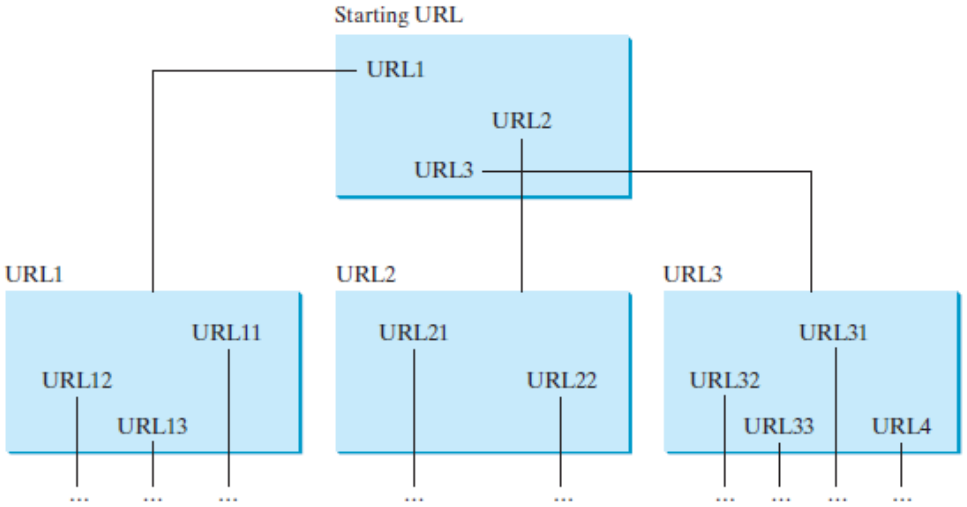
```
Scanner input = new Scanner(url.openStream());
```

一个例子

```
import java.util.Scanner;
public class ReadFileFromURL {
  public static void main(String[] args) {
    try {
      java.net.URL url = new java.net.URL("http://www.xmu.edu.cn/");
      int count = 0;
      Scanner input = new Scanner(url.openStream());
      while (input.hasNext()) {
        String line = input.nextLine();
        count += line.length();
      }
      System.out.println("The file size is " + count + " characters");
    } catch (java.net.MalformedURLException ex) {
      System.out.println("Invalid URL");
    } catch (java.io.IOException ex) {
      System.out.println("I/O Errors: no such file");
    }
  }
}
```

网络爬虫

- 网络爬虫是一个自动提取网页的程序，它为搜索引擎从WWW下载网页，是搜索引擎的重要组成。传统爬虫从一个或若干初始网页的URL开始，获得初始网页上的URL，在抓取网页的过程中，不断从当前页面上抽取新的URL放入队列，直到满足系统的一定停止条件。



网络爬虫设计

- 可以从一个网页出发，提取页面上所有http://格式的字符串，然后当作新目标继续爬行。算法如下：

```
把起始URL添加到队列listOfPendingURLs;
当listOfPendingURLs非空，且队列listOfTraversedURLs
大小<=100{
  URL从listOfPendingURLs出队;
  如果URL不在已访问队列listOfTraversedURLs中{
     URL入队listOfTraversedURLs;
    显示URL;
    读取URL所在页面，搜索页面上所有新URL{
       如果新URL没有在队列listOfTraversedURLs中出现，
将其入队listOfPendingURLs;
    }
  }
}
```

代码段：读取用户输入的URL并开始爬行

- listOfPendingURLs 待处理URL队列
- listOfTraversedURLs已访问URL队列

```
import java.util.Scanner;
import java.util.ArrayList;

public class WebCrawler {
  public static void main(String[] args) {
    java.util.Scanner input = new java.util.Scanner(System.in);
    System.out.print("Enter a URL: ");
    String url = input.nextLine();
    crawler(url); // Traverse the Web from the a starting url
  }

  public static void crawler(String startingURL) {
    ArrayList<String> listOfPendingURLs = new ArrayList<>();
    ArrayList<String> listOfTraversedURLs = new ArrayList<>();

```

代码段：网络爬虫算法的具体实现

```
    listOfPendingURLs.add(startingURL);
    while (!listOfPendingURLs.isEmpty() &&
        listOfTraversedURLs.size() <= 100) {
      String urlString = listOfPendingURLs.remove(0);
      if (!listOfTraversedURLs.contains(urlString)) {
        listOfTraversedURLs.add(urlString);
        System.out.println("Craw " + urlString);

        for (String s: getSubURLs(urlString)) {
          if (!listOfTraversedURLs.contains(s))
            listOfPendingURLs.add(s);
        }
      }
    }
  }
```

```java
  public static ArrayList<String> getSubURLs(String urlString) {
    ArrayList<String> list = new ArrayList<>();

    try {
      java.net.URL url = new java.net.URL(urlString);
      Scanner input = new Scanner(url.openStream());
      int current = 0;
      while (input.hasNext()) {
        String line = input.nextLine();
        current = line.indexOf("http:", current);
        while (current > 0) {
          int endIndex = line.indexOf("\"", current);
          if (endIndex > 0) { // Ensure that a correct URL is found
            list.add(line.substring(current, endIndex));
            current = line.indexOf("http:", endIndex);
          }
          else
            current = -1;
        }
      }
    }
    catch (Exception ex) {
      System.out.println("Error: " + ex.getMessage());
    }

    return list;
  }
}
```

代码段：页面URL提取

THE END