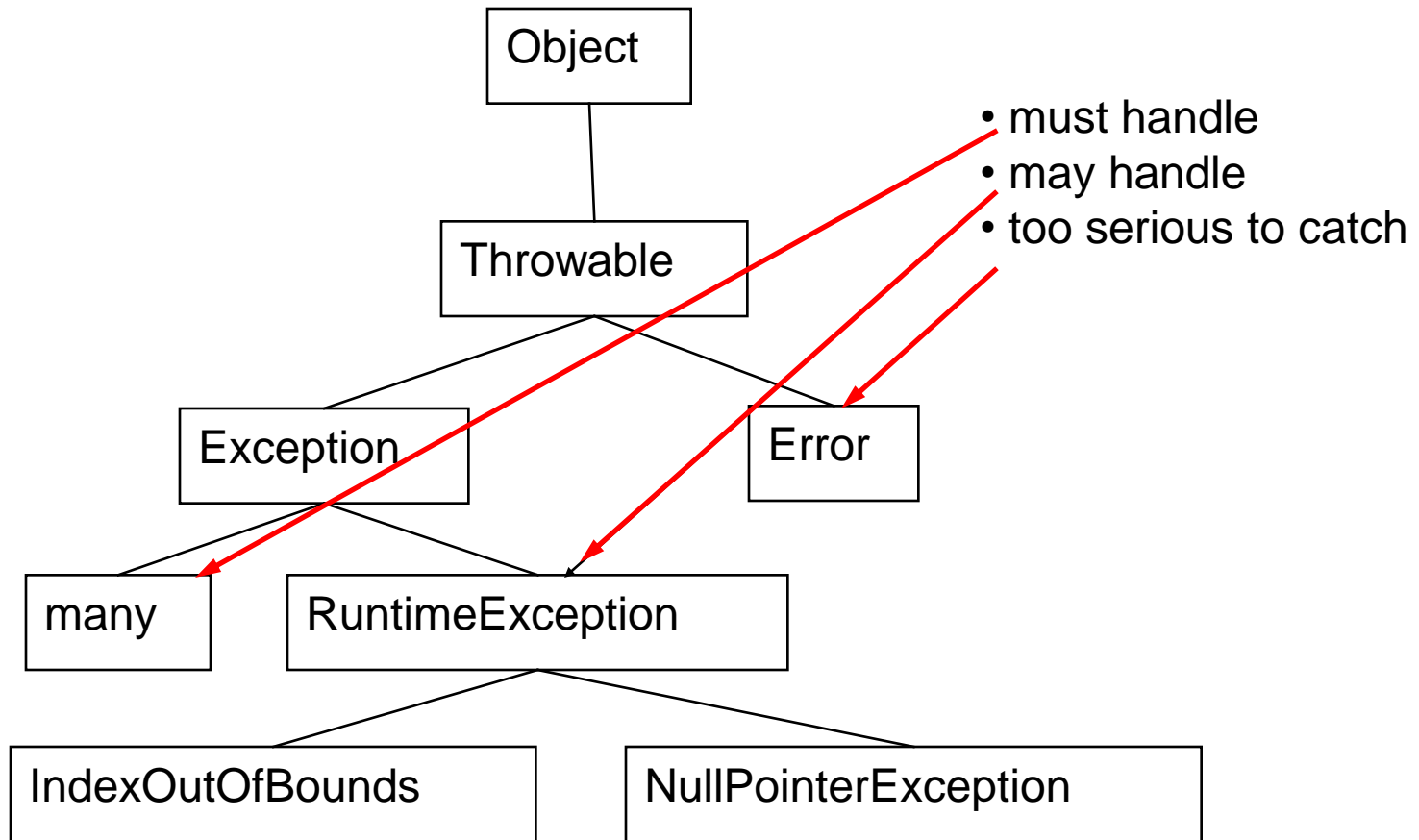


Java Exceptions

Intro to Exceptions

- What are exceptions?
 - Events that occur during the execution of a program that interrupt the normal flow of control.
- One technique for handling Exceptions is to use return statements in method calls.
- This is fine, but java provides a much more general and flexible formalism that forces programmers to consider exceptional cases.
- e.g. Opening a file, does not exist.

Exception Class hierarchy



Throwable hierarchy

- All exceptions extend the class `Throwable`, which splits into two branches:

`Error` and `Exception`

- `Error`: internal errors and resource exhaustion inside the Java runtime system. Little you can do.
- `Exception`: splits further into two branches. `RuntimeException` and other exceptions

Focus on the Exception branch

- Two branches of **Exception**
 - exceptions that derived from **RuntimeException**
 - examples: a bad cast, an out-of-array access
 - happens because errors exist in your program. Your fault.
 - those not in the type of **RuntimeException**
 - example: trying to open a malformed URL
 - program is good, other bad things happen. Not your fault.
- Checked exceptions vs. unchecked exceptions
 - *Unchecked exceptions*: exceptions derived from the class **Error** or the class **RuntimeException**
 - *Checked **exceptions***: all other exceptions that are not unchecked exceptions
 - If they occur, they must be dealt with in some way.
 - The compiler will check whether you provide exception handlers for checked exceptions which may occur

Exception Handling Basics

- Three parts to Exception handling
 1. claiming exception
 2. throwing exception
 3. catching exception
- A method has the option of *throwing* one or more exceptions when specified conditions occur. This exception must be *claimed* by the method. Another method calling this method must either *catch* or re*throw* the exception. (unless it is a RuntimeException)

Claiming Exceptions

- Method declaration must specify every exception that the method potentially throws

MethodDeclaration throws Exception1,
Exception2, ..., ExceptionN

- Exceptions themselves are concrete subclasses of Throwable and must be defined and locatable in regular way.

Throwing Exception

- To throw an Exception, use the *throw* keyword followed by an instance of the Exception class

```
void foo() throws SomeException{  
    if (whatever) {...}  
    else{ throw new SomeException(...)}  
}
```

- Note that if a method foo has a throw clause within it, that the Exception that is thrown (or one of its superclasses) must be claimed after the signature.

Declaring to throw checked exceptions

- A java method or constructor should be declared to throw exceptions under two situations:
 1. It calls another method that throws a checked exception
 2. It throws a checked exception with the throw statement inside its body
- Declare a method or constructor to throw an exception (or exceptions) by using throws clause in its header

Single exception:

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

Multiple exception:

```
public Image load(String s) throws EOFException,  
    IOException
```

Using `throw` to throw an exception

- Throw an exception under some bad situations

E.g: a method named `readData` is reading a file whose header says it contains 700 characters, but it encounters the end of the file after 200 characters. You decide to throw an exception when this bad situation happens by using the `throw` statement

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

the entire method will be

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

Using `throw` to throw an exception (cont.)

- In the method body you can only **throw** exceptions which are of the same type or the subtype of the exceptions declared after **throws** in the method header
- If you can find an appropriate exception class in the library, make an object of that class and throw it; otherwise, you design your own exception class

Catching Exceptions

- The third piece of the picture is catching exceptions.
- This is what you will do with most commonly, since many of java's library methods are defined to throw one or more runtime exception.
- Catching exceptions:
 - When a method is called that throws an Exception e.g. `SomeException`, it must be called in a try-catch block:

```
try
{
    foo();
}
catch(SomeException se)
{
    ...
}
```

Catching Exceptions, cont.

- Note that if a method throws an Exception that is NOT a RuntimeException, you **must** do one of two things:
 - try-catch it (often called *handling it*)
 - rethrow it
- In the latter case, responsibility then moves up the calling chain to handle it, and so on all the way up to main.

More on try-catch

The general form of the try-catch structure is:

```
try{
    /* any number of lines of code
       that call any number of methods
       with any thrown Exceptions */
}
catch(Exception1 e1){
    /* do anything you want here
       e.g. change value and try again.
       print error and quit
       print stacktrace
    */
}
catch (Exception2 e2){
    /* any number of exceptions can be handled ... */
}
```

Example1

```
import java.io.*;
```

```
public class Exception1{  
    public static void main(String[] args){  
        InputStream f;  
        try{  
            f = new FileInputStream("foo.txt");  
        }  
        catch(FileNotFoundException fnfe){  
            System.out.println(fnfe.getMessage());  
        }  
    }  
}
```

Example2

```
import java.io.*;
public class Exception2{
    public static void main(String[] args){
        InputStream fin;
        try{
            fin = new FileInputStream("foo.txt");
            int input = fin.read();
        }
        catch(FileNotFoundException fnfe){
            System.out.println(fnfe.getMessage());
        }
        catch(IOException ioe){
            System.out.println(ioe.getMessage());
        }
    }
}
```


Catching exceptions

- Checked exceptions handling is strictly enforced. If you invoke a method that lists a checked exception in its throws clause, you have three choices
 1. Catch the exception and handle it
 2. Declare the exception in your own `throws` clause, and let the exception pass through your method (you may have a `finally` clause to clean up first)
 3. Catch the exception and map it into one of your exceptions by throwing an exception of a type declared in your own `throws` clause

try/catch clause (1)

- If no exception occurs during the execution of the statements in the `try` clause, it finishes successfully and all the `catch` clauses are skipped
- If any of the code inside the `try` block throws an exception, either directly via a `throw` or indirectly by a method invoked inside it
 1. The program skips the remainder of the code in the `try` block
 2. The `catch` clauses are examined one by one, to see whether the type of the thrown exception object is compatible with the type declared in the `catch`.
 3. If an appropriate `catch` clause is found, the code inside its body gets executed and all the remaining `catch` clauses are skipped.
 4. If no such a `catch` clause is found, then the exception is thrown into an outer `try` that might have a `catch` clause to handle it
- A `catch` clause with a superclass `exceptionType` cannot precede a `catch` clause with a subclass `exceptionType`

try/catch clause (2)

another choice for this situation is to do nothing but simply pass the exception on to the caller of the method

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

- If you call a method that throws a checked exception, you must either handle it or pass it on.

finally clause

- You can use a `finally` clause without a `catch` clause
- Sometimes the `finally` clause can also throw an exception

Example

```
public boolean searchFor(String file,
String word)
    throws StreamException
{
    Stream input = null;
    try {
        some code which may throw an
        StreamException
    } finally {
        input.close(); // this may throw anIOException
    }
}
```

finally clause

- You may want to do some actions whether or not an exception is thrown. `finally` clause does this for you

```
Graphics g = image.getGraphics();
try {
    //1
    code that might throw exceptions
    //2
} catch (IOException e) {
    //3
    show error dialog (// some code which may throw
exceptions)
    //4
} finally {
    g.dispose(); (// some code which will not throw
exceptions)
    //5
} //6
```

- No exception is thrown: 1, 2, 5, 6
- An exception is thrown and caught by the `catch` clause
 - The `catch` clause doesn't throw any other exception: 1, 3, 4, 5, 6
 - The `catch` clause throws an exception itself: 1, 3, 5, and the exception is thrown back to the caller of this method
- An exception is thrown but not caught by the `catch` clause: 1, 5, and the exception is thrown back to the caller of this method

Recommendations

- Do not use Exceptions to handle normal conditions in the program that can be checked with if statements. For example:
 - to find the end of an array
 - to check if an object is null

Creating your own Exceptions

- You can follow this procedure exactly when creating your own Exception.
- Create a class that subclasses Exception (or RuntimeException).
- You may also add functionality so that a relevant message is stored when the error is thrown, and any other customized functionality you choose.

Creating new exception types

- Exceptions are objects. New exception types should extend `Exception` or one of its subclasses
- Why creating new exception types?
 1. describe the exceptional condition in more details than just the string that `Exception` provides

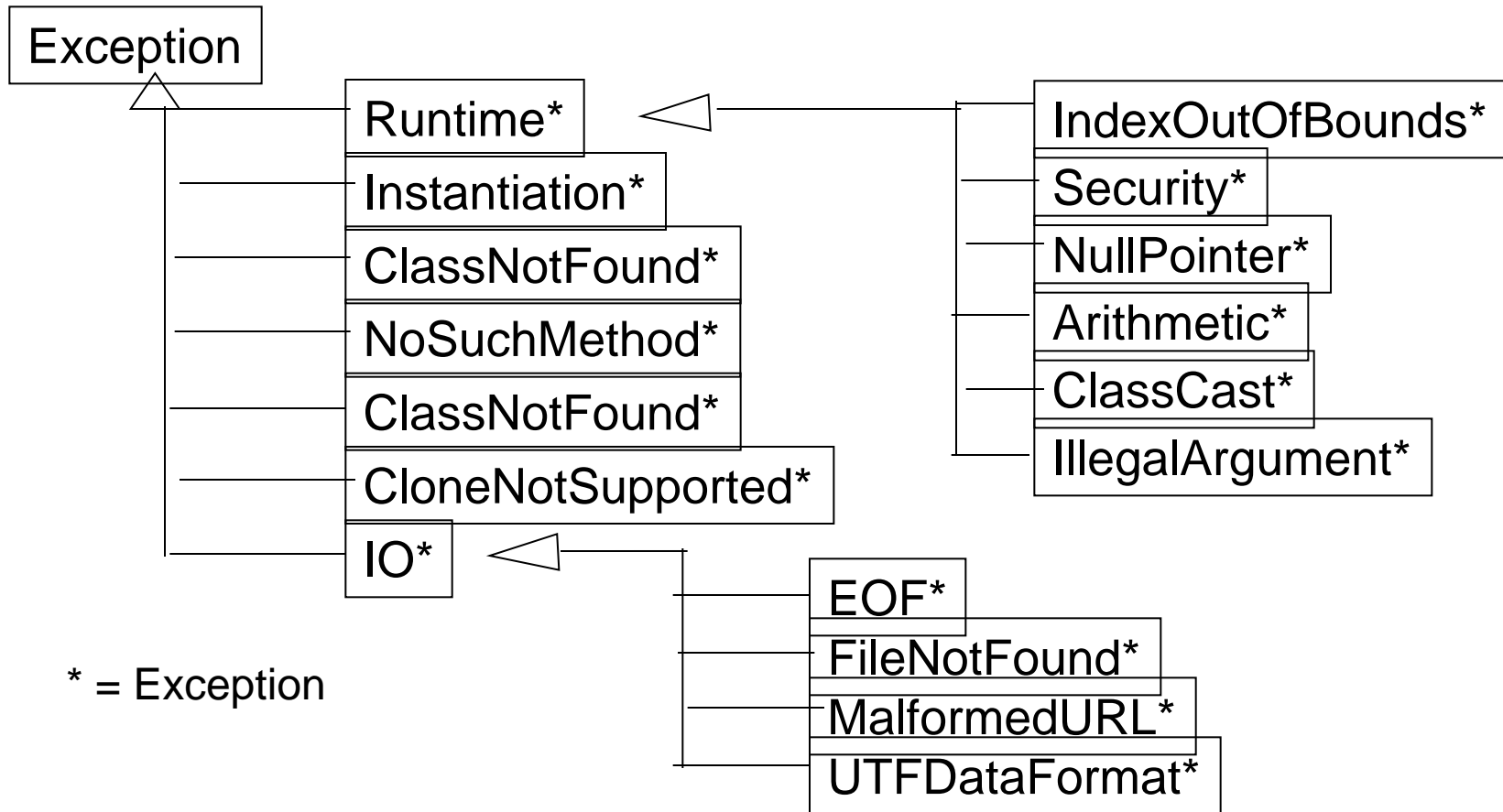
E.g. suppose there is a method to update the current value of a named attribute of an object, but the object may not contain such an attribute currently. We want an exception to be thrown to indicate the occurring of if such a situation

```
public class NoSuchAttributeException extends Exception {  
    public String attrName;  
    public NoSuchAttributeException (String name) {  
        super("No attribute named \"" + name + "\" found");  
        attrName = name;  
    }  
}
```

2. the type of the exception is an important part of the exception data – programmers need to do some actions **exclusively** to one type of exception conditions, not others

Throwing Your Own Exceptions

- There are many exceptions and you should get to know them



Exception Handling

Exception Handling

```
try {  
    // try block  
}  
catch (ExceptionType1 param1) {  
    // Exception Block  
}  
catch (ExceptionType2 param2) {  
    // Exception Block  
}  
.....  
catch (ExceptionTypeN paramN) {  
    // Exception Block  
}  
finally {  
    // finally Block  
}
```

Codes that have some possibilities to generate exception(s).

Execute the codes here when the corresponding exception occurred.

Do always

Exception Handling

```
class Divider {  
  
    public static void main(String[] args) {  
  
        try {  
  
            System.out.println("Before Division");  
            int i = Integer.parseInt(args[0]);  
            int j = Integer.parseInt(args[1]);  
            System.out.println(i / j);  
            System.out.println("After Division");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("ArithmeticException");  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("ArrayIndex" +  
                "OutOfBoundsException");  
        }  
        catch (NumberFormatException e) {  
            System.out.println("NumberFormatException");  
        }  
    }  
}
```

```
finally {  
    System.out.println("Finally block");  
}  
}  
}
```

Catch Block Searches

```
class CatchSearch {  
  
    public static void main(String[] args) {  
        try {  
            System.out.println("Before a");  
            a();  
            System.out.println("After a");  
        }  
        catch (Exception e) {  
            System.out.println("main: " + e);  
        }  
        finally {  
            System.out.println("main: finally");  
        }  
    }  
  
    public static void a() {  
        try {  
            System.out.println("Before b");  
            b();  
            System.out.println("After b");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("a: " + e);  
        }  
        finally {  
            System.out.println("a: finally");  
        }  
    }  
}
```

```
    public static void b() {  
        try {  
            System.out.println("Before c");  
            c();  
            System.out.println("After c");  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("b: " + e);  
        }  
        finally {  
            System.out.println("b: finally");  
        }  
    }  
  
    public static void c() {  
        try {  
            System.out.println("Before d");  
            d();  
            System.out.println("After d");  
        }  
        catch (NumberFormatException e) {  
            System.out.println("c: " + e);  
        }  
        finally {  
            System.out.println("c: finally");  
        }  
    }  
}
```

```
    public static void d() {  
        try {  
            int array[] = new int[4];  
            array[10] = 10;  
        }  
        catch (ClassCastException e) {  
            System.out.println("d: " + e);  
        }  
        finally {  
            System.out.println("d: finally");  
        }  
    }  
}
```

The throw Statement

The throw Statement

```
throw object;
```

Use the statement when we generate an exception

Exception Object in Argument

```
catch (ExceptionType param) {  
    ....  
    throw param;  
    ....  
}
```

Throw to new Exception Object

```
throw new ExceptionType(args);
```

The throw Statement

```
class ThrowDemo {  
  
    public static void main(String[] args) {  
        try {  
            System.out.println("Before a");  
            a();  
            System.out.println("After a");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("main: " + e);  
        }  
        finally {  
            System.out.println("main: finally");  
        }  
    }  
  
    public static void a() {  
        try {  
            System.out.println("Before b");  
            b();  
            System.out.println("After b");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("a: " + e);  
        }  
        finally {  
            System.out.println("a: finally");  
        }  
    }  
}
```


```
    public static void b() {  
        try {  
            System.out.println("Before c");  
            c();  
            System.out.println("After c");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("b: " + e);  
        }  
        finally {  
            System.out.println("b: finally");  
        }  
    }  
  
    public static void c() {  
        try {  
            System.out.println("Before d");  
            d();  
            System.out.println("After d");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("c: " + e);  
            throw e;  
        }  
        finally {  
            System.out.println("c: finally");  
        }  
    }  
}
```

```
    public static void d() {  
        try {  
            int i = 1;  
            int j = 0;  
            System.out.println("Before division");  
            System.out.println(i / j);  
            System.out.println("After division");  
        }  
        catch (ArithmeticException e) {  
            System.out.println("d: " + e);  
            throw e;  
        }  
        finally {  
            System.out.println("d: finally");  
        }  
    }  
}
```

The throw Statement

```
class ThrowDemo2 {  
  
    public static void main(String[] args) {  
        try {  
            System.out.println("Before a");  
            a();  
            System.out.println("After a");  
        }  
        catch (Exception e) {  
            System.out.println("main: " + e);  
        }  
        finally {  
            System.out.println("main: finally");  
        }  
    }  
}
```

```
public static void a() {  
    try {  
        System.out.println("Before throw statement");  
        throw new ArithmeticException();  
    }  
    catch (Exception e) {  
        System.out.println("a: " + e);  
    }  
    finally {  
        System.out.println("a: finally");  
    }  
}
```



Exception and Error Classes

Throwable Constructors

Throwable()

Throwable(String message)

getMessage() Method

String getMessage()

printStackTrace() Method

void printStackTrace()

Exception Constructor

Exception()

Exception (String message)

Subclasses of Exception Class

ClassNotFoundException

IllegalAccessException

InstantiationException

InterruptedException

NoSuchFieldException

NoSuchMethodException

RuntimeException

Subclasses of RuntimeException

ArrayIndexOutOfBoundsException

ArithmeticException

ClassCastException

NegativeArraySizeException

NullPointerException

NumberFormatException

SecurityException

StringIndexOutOfBoundsException

Exception and Error Classes

```
class PrintStackTraceDemo {

    public static void main(String[] args) {
        try {
            a();
        }
        catch (ArithmeticException e) {
            e.printStackTrace();
        }
    }

    public static void a() {
        try {
            b();
        }
        catch (NullPointerException e) {
            e.printStackTrace();
        }
    }

    public static void b() {
        try {
            c();
        }
        catch (NullPointerException e) {
            e.printStackTrace();
        }
    }
}
```

```
public static void c() {
    try {
        d();
    }
    catch (NullPointerException e) {
        e.printStackTrace();
    }
}

public static void d() {
    try {
        int i = 1;
        int j = 0;
        System.out.println(i / j);
    }
    catch (NullPointerException e) {
        e.printStackTrace();
    }
}
```

Result:

```
java.lang.ArithmeticException: / by zero
```

```
at PrintStackTraceDemo.d(PrintStackTraceDemo.java:43)
at PrintStackTraceDemo.c(PrintStackTraceDemo.java:32)
at PrintStackTraceDemo.b(PrintStackTraceDemo.java:23)
at PrintStackTraceDemo.a(PrintStackTraceDemo.java:14)
at PrintStackTraceDemo.main(PrintStackTraceDemo.java:5)
```

The throws Clause

throws Statement of Constructor

```
consModifiers clsName(cparams) throws  
exceptions {  
    // constructor body  
}
```

throws Statement at a Method

```
mthModifiers rtype mthName(mparams) throws  
exceptions {  
    // constructor body  
}
```

```
class ThrowsDemo {  
    public static void main(String[] args) {  
        a();  
    }  
    public static void a() {  
        try {  
            b();  
        }  
        catch (ClassNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
public static void b() throws  
ClassNotFoundException {  
    c();  
}
```

```
public static void c() throws  
ClassNotFoundException {  
    Class cls =  
    Class.forName("java.lang.Integer");  
    System.out.println(cls.getName());  
    System.out.println(cls.isInterface());  
}
```

Custom Exceptions

Subclass of Exception

```
import java.util.*;

class ExceptionSubclass {

    public static void main(String[] args) {
        a();
    }

    static void a() {
        try {
            b();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    static void b() throws ExceptionA {
        try {
            c();
        }
        catch (ExceptionB e) {
            e.printStackTrace();
        }
    }
}
```

```
static void c() throws ExceptionA, ExceptionB {
    Random random = new Random();

    int i = random.nextInt();
    if (i % 2 == 0) {
        throw new ExceptionA("We have a problem");
    }
    else {
        throw new ExceptionB("We have a big problem");
    }
}

class ExceptionA extends Exception {
    public ExceptionA(String message) {
        super(message);
    }
}

class ExceptionB extends Exception {
    public ExceptionB(String message) {
        super(message);
    }
}
```

Execution:

ExceptionA: We have a problem

```
at ExceptionSubclass.c(ExceptionSubclass.java:31)
at ExceptionSubclass.b(ExceptionSubclass.java:20)
at ExceptionSubclass.a(ExceptionSubclass.java:11)
at ExceptionSubclass.main(ExceptionSubclass.java:6)
```