Java Exception

Very slightly modified from K.P. Chow

University of Hong Kong

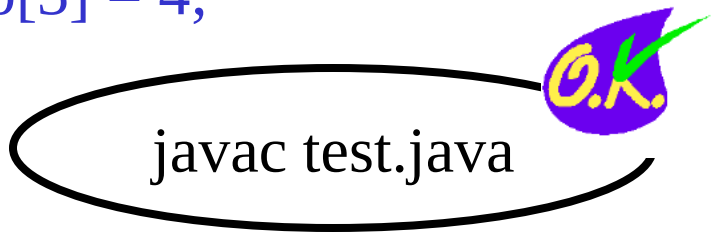
(some slides from S.M. Yiu)

Exception

An event that occurs during the execution of a program that disrupts the normal flow of instructions

Example

```
public class test {
        public static void main(String[] args) {
                int temp[] = new int[3];
                temp[3] = 4;
                …..
        }
}
```



Error detected in runtime.

```
C:\> java test
Exception in thread “main” java.lang.ArrayIndexOutOfBoundsException
        at test.main(test.java:4)
```

Example

```
import java.io.*;
public class test2 {
        public static void main(String[] args) {
                File input = new File("input.txt");
                FileInputStream in = new FileInputStream(input);
                …..
        }
}
```

C:\> javac test.java

test2.java:5: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown

FileInputStream in = new FileInputStream(input);

^

1 error

Detected in compile time

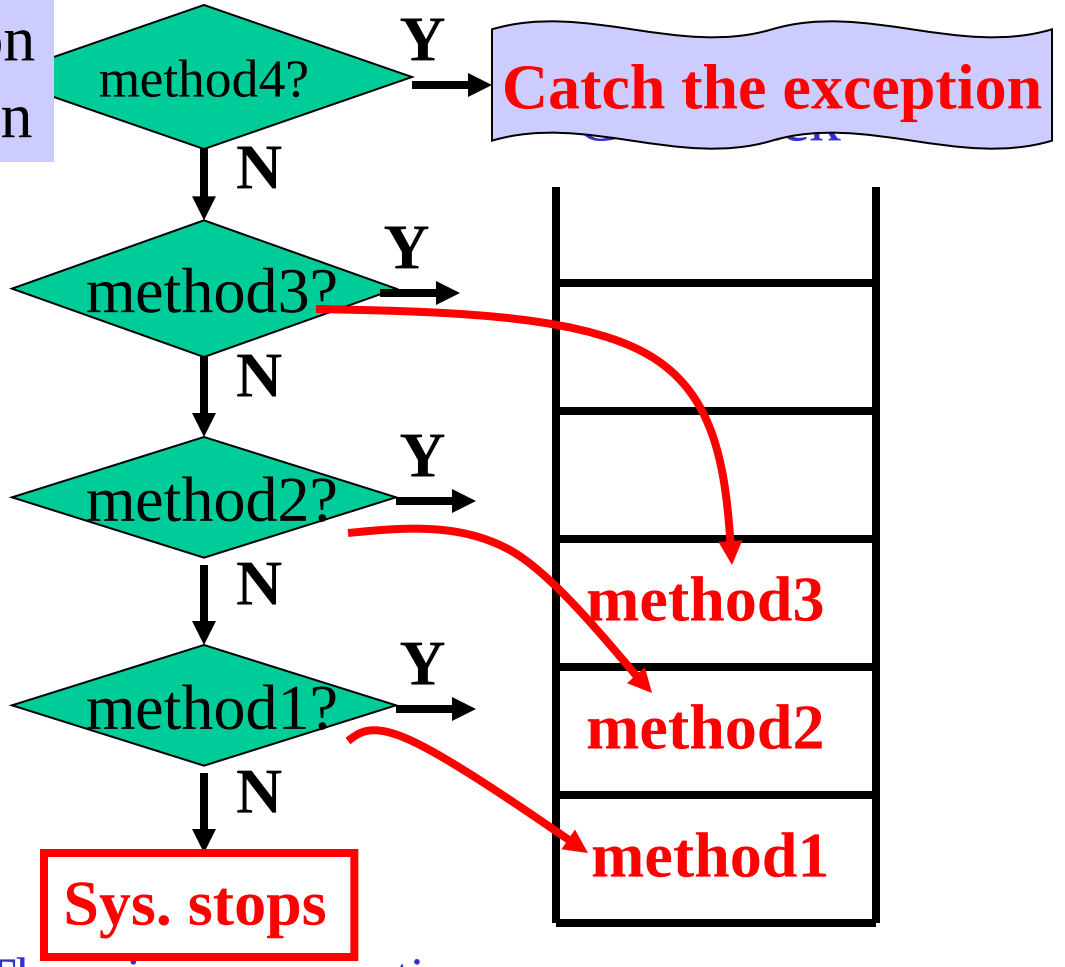
Mechanism for handling exceptions

Exception handler in

Example

```
method1 {
        call method2;
}
method2 {
        call method3;
}
method3 {
        call method4;
}
method4 {
        ……
}
```

Exception!!



Throwing an exception

Create an exception object and runtime system take over

Runtime system

Two Types of Exception

runtime exceptions

occurs within Java runtime system

run time checking

checked exceptions

occurs outside the *control* of Java runtime system

compile time checking

e.g. Division by zero
ArrayIndexOutofBounds

e.g. IO exception
FileNotFoundException

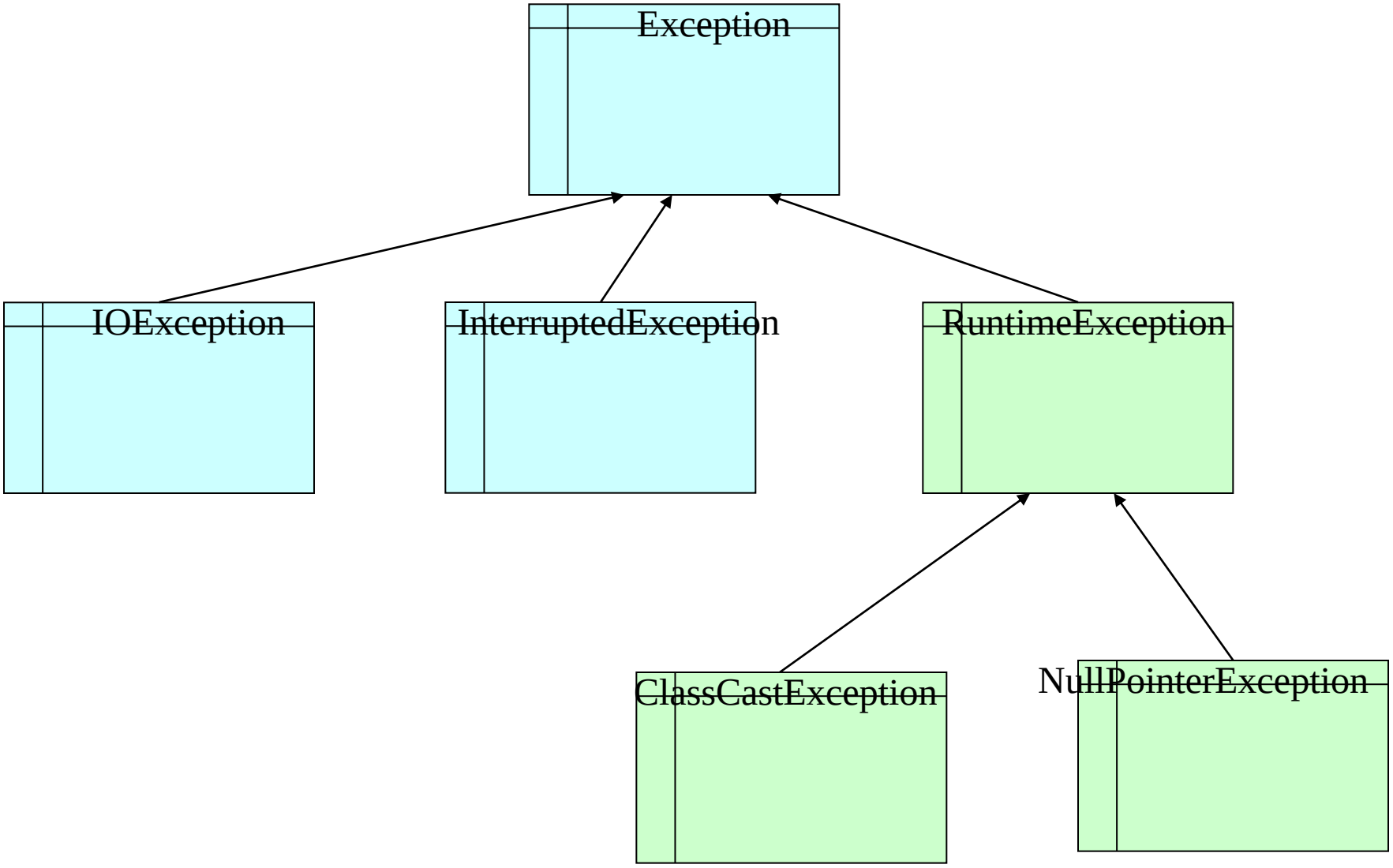
implement exception handler

Must be caught or thrown

Q: what exception to worry For each method?

Notify calling methods

Exception Hierarchy (part of)



Exceptions can be thrown by a method

Exception thrown by this method

Exception thrown by called methods

Leave it to calling method to handle

Example(1)
in java.io.InputStream class,
public int read(byte[], b) throws IOException

similar for read( )

(Exception Type,
must be a class inherits from
the "Throwable" class)

Example(2)

```
import java.io.*;
public void m1( ){
        int b = System.in.read( );
}
```

Error!!!
m1 has to either catch or throw IOException

Example(3)

```
import java.io.*;
public void m1( ){
        m2( );
}
public void m2( ){
        m3( );
}
public void m3( ) throws IOException {
        int b = System.in.read( );
}
```

```
public void m1( ) throws IOException {
        m2( );
}
```

```
public void m2( ) throws IOException {
        m3( );
}
```

Compile ok, but do not handle the exception….

Error!!
m2 has to either catch or throw IOException

Error!!
m1 has to either catch or throw IOException

```
import java.io.*;
import java.util.Vector;
public class ListOfNumbers {
    private Vector v;
    private static final int SIZE = 10;

    public ListOfNumbers( ) {
        v = new Vector(SIZE);
        for (int i=0; i < SIZE; i++) {
            v.addElement(new Integer(i));
        }
    }

    public void writeList() {
        PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
        for (int i=0; i<SIZE; i++) {
            out.println(v.elementAt(i));
        }
        out.close();
    }
```

NOT required to catch or you can

REQUIRED to catch or throw

public FileWriter(String fileName) throws IOException

runtime exception: ArrayIndexOutOfBoundsException

Catch the exception – exception handler

statements that may throw exceptions

```
try {
  block of statements
} catch (ExceptionType name) {
    exception handler 1
} catch (ExceptionType name) {
    exception handler 2
}
```

Each catch block is an exception handler taking care of different exceptions

```
public void writeList() {
    PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
    for (int i=0; i<SIZE; i++) {
        out.println(v.elementAt(i));
    }
```

```
public void writeList() {
try {
        PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
                for (int i=0; i<SIZE; i++) {
                        out.println(v.elementAt(i));
                }
        out.close( );
} catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("Caught ArrayIndexOutOfBoundsException");
} catch (IOException e) {
        System.err.println("Caught IOException");
}
}
```

```
public void writeList() {
try {
        PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
                for (int i=0; i<SIZE; i++) {
                        out.println(v.elementAt(i));
                }
        out.close( ):
} catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("Caught ArrayIndexOutOfBoundsException");
} catch (IOException e) {
        System.err.println("Caught IOException");
}
}
```

May not get executed!

use a finally block
(always will execute, even if we jump out of try block)

```
public void writeList() {
try {
        PrintWriter out = new PrintWriter(new FileWriter("out.txt"));
                for (int i=0; i<SIZE; i++) {
                        out.println(v.elementAt(i));
                }
} catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("Caught ArrayIndexOutOfBoundsException");
 } catch (IOException e) {
        System.err.println("Caught IOException");
 } finally {
        if (out != null) {
                out.close( );
        }
}
```

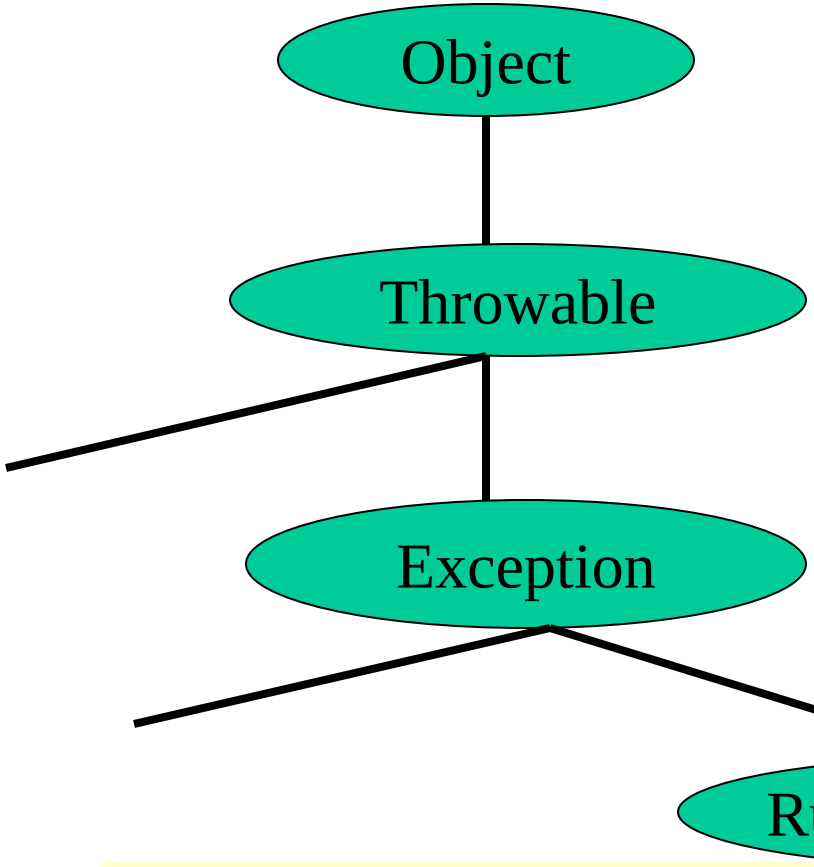
If throw, an old exception might be lost

How about if writeList do not want to handle it?

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

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

Can I create my own Exception classes?



differentiate your exceptions from existing ones….

Example

```
public class NewException
        extends Exception {
……}
```

Example

```
public class NewException
        extends RuntimeException {
……}
```

Not required to catch or throw

Example: Assign 1

```
…
public static void main( String[] args) {
        int j;
        j = Integer.parseInt(args[0]);
        …..
}
```

in Integer class:

```
public static int parseInt(String s)
                        throws NumberFormatException;
```

```
public static void main( String[] args) {
        int j;
        try {
                j = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
                System.out.println("wrong input format");
        }
…..
```

- exception.printStackTrace() prints:
  - Error message
  - Exception class name
  - Descriptive string stored in the exception
    - throw new Exception("descriptive string");
  - List of methods that had not completed execution
- exception.getMessage()
  - Returns the descriptive string stored in the exception
- exception.getStackTrace()
  - Used to output to other forms (see next page)

```
StackTraceElement[] trace = exception.getStackTrace();
For(int I=0; I<trace.length;I++){
    StackTraceElement current = trace[I];
    System.out.print(current.getClassName() + “\t” );
    System.out.print(current.getFileName() + “\t” );
    System.out.print(current.getLineNumber () + “\t” );
    System.out.print(current.getMethodName() + “\n” );
}
```

Storing chained exception info.

- Catch one- throw another, but can keep the info of the old one.

```
catch (NumberFormatException e) {
        throw new Exception("descriptive string",e);
}
```