
Exceptions

Your computer takes exception

- Exceptions are errors in the logic of a program (run-time errors).
- Examples:

```
Exception in thread "main" java.io.FileNotFoundException:  
student.txt (The system cannot find the file specified.)
```

```
Exception in thread "main" java.lang.NullPointerException:  
at FileProcessor.main(FileProcessor.java:9)
```

- Question: do all run-time errors cause Exceptions?
-

Causes of Exceptions

- Most exceptions happen because of “corner cases”:
 - your program does something at the boundaries of what Java knows how to handle.
 - For example:
 - Java knows how to open files for reading, mostly.
 - But if you tell it to open a file that doesn't exist, it doesn't know how it should behave.
 - It throws an exception, and gives the programmer an opportunity to define how the program should react.
-

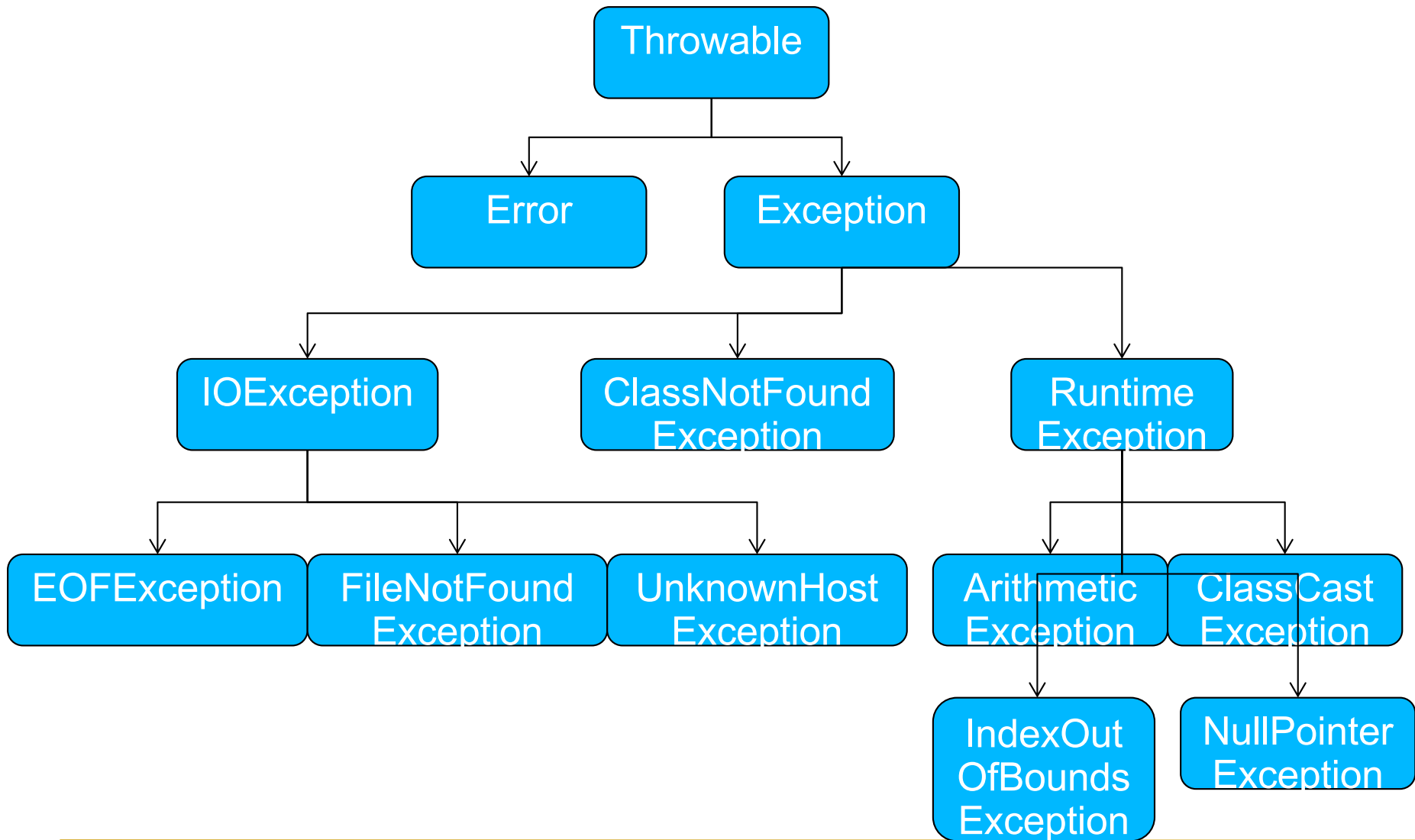
The Exception Class

- As with anything in Java, `Exception` is a class

Method	What it does
<code>void printStackTrace()</code>	Prints the sequence of method calls leading up to the statement that caused the <code>Exception</code> .
<code>String getLocalizedMessage()</code>	Returns a “detail” message.
<code>String toString()</code>	Returns the <code>Exception</code> class name and detail message.

The methods in the `Exception` class can be useful for debugging, as we will see.

The Exception class hierarchy (partial)



Pitch and catch

- When a Java statement causes an Exception (called throwing the Exception), by default Java abruptly ends the program.
 - To stop this default behavior, you can write code that catches the thrown Exception.
-

Catch: An example

```
import java.util.*; // For Scanner class
import java.io.*;

public class FileProcessor
{
    public static void main(String [] args)
    {
        try {
            File inputFile = new File("student.txt");
            Scanner input = new Scanner(inputFile);

            while(input.hasNextLine()) {
                System.out.println("> " + input.nextLine());
            }
        }
        catch(FileNotFoundException exception) {
            System.out.println("Could not find the file 'student.txt'.");
        }
    }
}
```

try/catch syntax

try block indicates that the enclosed statements have exception handlers associated with them.

```
try {
```

```
    <statements that might cause an exception>;
```

```
}
```

catch block is an exception handler for *one type* of exception.

```
catch (<ExceptionType1> e1) {
```

```
    <statements>
```

```
}
```

The type of exception that the catch block handles is indicated with a parameter.

```
...
```

```
catch (<ExceptionTypeN> eN) {
```

```
    <statements>
```

```
}
```

You can have as many catch blocks for one try block as you like. They must each handle a different type of exception.

Control Flow with try/catch

```
try {
    <statements that might cause an exception>;
}
catch(<ExceptionType1> e1) {
    <statements>
}
...
catch(<ExceptionTypeN> eN) {
    <statements>
}
<statements after try/catch>
```

- If no exception occurs during the try block:
 - jump to statements after all the catch blocks.
 - If an exception occurs in the try block:
 - jump to the first handler for that type of exception.
-
- After the catch finishes, jump to the statement after all the catch blocks.

Catch: An example

```
import java.util.*; // For Scanner class
import java.io.*;

public class FileProcessor
{
    public static void main(String [] args)
    {
        try {
            File inputFile = new File("student.txt");
            Scanner input = new Scanner(inputFile);

            while(input.hasNextLine()) {
                System.out.println("> " + input.nextLine());
            }
        } catch(FileNotFoundException exception) {
            System.out.println("Could not find the file 'student.txt'.");
        }
    }
}
```

finally

```
try {  
    <statements that might cause an exception>;  
}  
catch (<ExceptionType1> e1) {  
    <statements>  
}  
...  
catch (<ExceptionTypeN> eN) {  
    <statements>  
} finally {  
    <statements in here are done whether  
    an exception occurred or not>  
}
```

Remember

- When an exception occurs
 - you jump to the appropriate catch block
 - you do not ever jump back to the try block
 - If you absolutely must complete the try block
 - you need to put it inside a loop
-

Example

```
String filename = null;
Scanner inFromFile = null;
try {
    Scanner inFromKbd = new Scanner(System.in);
    System.out.print("Enter file name> ");
    filename = inFromKbd.nextLine();
    inFromFile = new Scanner(new File(filename));
} catch (FileNotFoundException e) {
    System.out.println("Error opening file " +
        filename);
}
/* but the file might not be open */
```

Example

```
String filename = null;
Scanner inFromFile = null;
boolean successfulOpen=false;
do {
    try {
        Scanner inFromKbd = new Scanner(System.in);
        System.out.print("Enter file name> ");
        filename = inFromKbd.nextLine();
        inFromFile = new Scanner(new File(filename));
        successfulOpen=true;
    } catch (FileNotFoundException e) {
        System.out.println("Error opening file " + filename);
    }
} while (!successfulOpen);
/* if we get this far, the file is open */
```

Stack Traces

- How do you know what went wrong?
- All exceptions have methods that return information about the cause of the Exception:

Method	Description
<code>getLocalizedMessage()</code>	Returns a String containing a description of the error
<code>getStackTrace()</code>	Returns an array of <code>StackTraceElement</code> objects, each of which contains info about where the error occurred
<code>printStackTrace()</code>	Displays the Stack Trace on the console.

Displaying the stack trace info

```
import java.util.*; // For Scanner class
import java.io.*;

public class FileProcessor
{
    public static void main(String [] args)
    {
        try {
            File inputFile = new File("student.txt");
            Scanner input = new Scanner(inputFile);

            while(input.hasNextLine()) {
                System.out.println("> " + input.nextLine());
            }
        } catch(FileNotFoundException exception) {
            System.out.println("Could not find the file 'student.txt'.");
            System.out.println(exception.getLocalizedMessage());
            exception.printStackTrace();
        }
    }
}
```


Multiple catch blocks

```
import java.util.*; // For Scanner class
import java.io.*;

public class FileProcessor
{
    public static void main(String [] args)
    {
        try {
            File inputFile = new File("student.txt");
            Scanner input = new Scanner(inputFile);
            PrintWriter pw = new PrintWriter(new File("quoted.txt"));

            while(input.hasNextLine()) {
                pw.println("> " + input.nextLine());
            }
        }
        catch(FileNotFoundException exception) {
            System.out.println("Could not find the file 'student.txt'.");
        }
        catch(IOException exception) {
            System.out.println("Could not write to file 'quoted.txt'.");
        }
    }
}
```

Multiple catch blocks

```
import java.util.*; // For Scanner class
import java.io.*;

public class FileProcessor
{
    public static void main(String [] args)
    {
        try {
            File inputFile = new File("student.txt");
            Scanner input = new Scanner(inputFile);
            PrintWriter pw = new PrintWriter(new File("quoted.txt"));

            while(input.hasNextLine()) {
                pw.println("> " + input.nextLine());
            }
        }
        catch(FileNotFoundException exception) {
            System.out.println("Could not find the input file.");
            System.out.println(exception.getLocalizedMessage());
            exception.printStackTrace();
        }
        catch(IOException exception) {
            System.out.println("Could not write to file 'quoted.txt'.");
        }
    }
}
```

Checked and Unchecked Exceptions

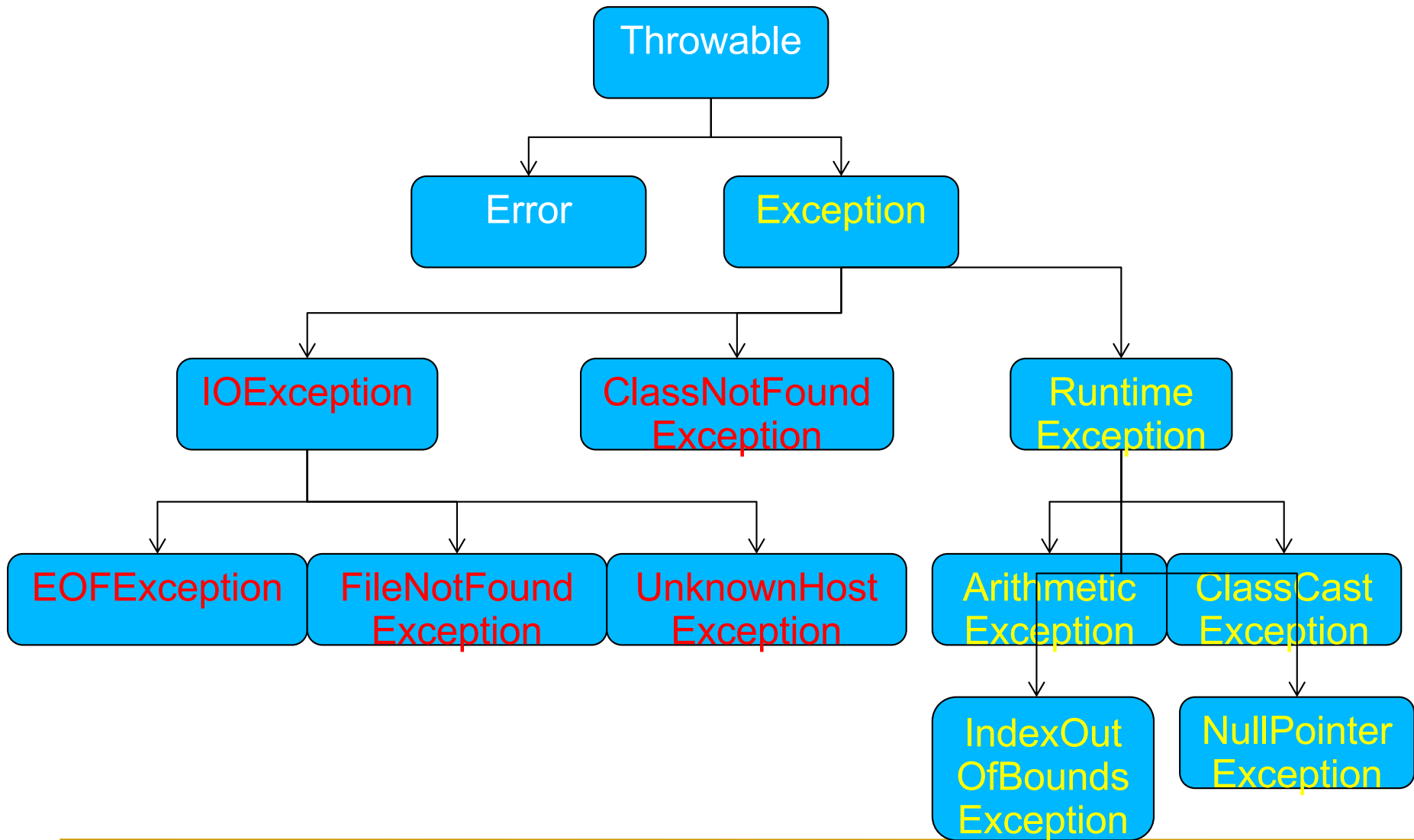
- Exceptions happen while the program is running
 - For most kinds of Exceptions, the compiler is happy to let the programmer make mistakes that could lead to an exception at run-time.

 - **Unchecked (by the compiler) Exceptions**
 - They are caused by programmer error.
 - The compiler lets the programmer screw up.
 - e.g., `NullPointerException`,
`IndexOutOfBoundsException`
-

Checked and Unchecked Exceptions

- Exceptions happen while the program is running
 - For most kinds of Exceptions, the compiler is happy to let the programmer make mistakes that could lead to an exception at run-time.
 - But, for certain kinds of exceptions, the compiler will **check** to see if your code **might** cause an exception at run-time.
 - Checked (by the compiler) Exceptions:
 - They are caused by things outside of the programmer's control (eg, a file doesn't exist).
 - The compiler requires that the programmer declare what to do if the Exception occurs.
-

Checked and Unchecked Exceptions



Options for Checked Exceptions

If the compiler detects that a statement might cause a Checked Exception, it requires the programmer to do either of the following:

1. Catch the Exception
2. Declare that crashing is acceptable
 - Use the `throws` clause in the method signature

Otherwise, the program will not compile.

Throws: An example

```
import java.util.*;    // For Scanner class
import java.io.*;

public class FileProcessor
{
    public static void main(String [] args)
        throws FileNotFoundException
    {
        File inputFile = new File("student.txt");
        Scanner input = new Scanner(inputFile);

        while(input.hasNextLine()) {
            System.out.println("> " + input.nextLine());
        }
    }
}
```

Catch or throw?

When should you catch an exception, and when should you declare that it can be thrown?

- Usually, if your catch block is not going to do anything besides print an error message and quit the program, it's better to just throw the exception
 - You should only catch an exception if you're really going to handle the error so that it won't affect the rest of the program.
-

Causing a ruckus

- Guess what ... you can create your very own Exceptions, any time you want!
 - The `throw` keyword:
(note: NOT the same as the `throws` keyword!)
 - Use it to make your code throw an exception
`throw new Exception();`
 - Mainly useful for passing messages between methods that aren't easily done with returns
-