Introduction to Exceptions in Java

Exceptions

- ***Exception***: an abnormal or erroneous situation at runtime
- Examples:
  - Array index out of bounds
  - Division by zero
  - Illegal input number format ←.

    public int a (int b).

    a("Hello")
  - Following a null reference

    e.g. not initialized.

Exceptions

- These erroneous situations ***throw an exception***
- Exceptions can be thrown by the *Java virtual machine* or by the *program*

Catching and re-Throwing Exceptions

The calling method can either catch an exception or it can re-throw it.

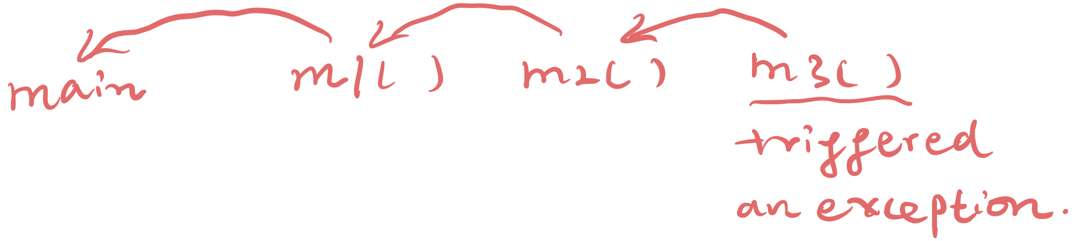
- The method catches the exception if it knows how to deal with the error.
- Otherwise the exception is re-thrown.

Catching Exceptions

- How try-catch works:
  - When the try-catch statement is executed, the statements in the try block are executed
  - If no exception is thrown:
    - Processing continues as normal
  - If an exception is thrown:
    - Program enters in "panic mode" and control is immediately passed to the first catch clause whose specified exception corresponds to the class of the exception that was thrown

Catching Exceptions

- If an exception is *not* caught and handled inside the method where it occurs:
  - Control is immediately returned to the *method that invoked the method* that produced the exception
  - If that method does not handle the exception (via a try statement with an appropriate catch clause) then control returns to the method that called it …
- This process is called ***propagating the exception***



1) m3 has try catch : m3 is done

2) m3 has no try catch

if m2 has try catch : done in m2

⋮

if the main still has no try catch, then an error occurs.

an exception could only get caught once.

Catching Exceptions

- Exception propagation continues until
  - The exception is caught and handled
  - Or until it is propagated out of the *main* method, resulting in the termination of the program

Catch Blocks

- A single catch block can handle more than one type of exception.
- This can reduce code duplication.
- In the catch clause we specify the types of exceptions that the block can handle and separate each exception type with a vertical bar (|).

A Try-Catch Example with Multiple Catch Statements

The try-catch syntax:

```
try {
        code
}
catch(exception1 e) {statements}
catch(exception2 e) {statements}
catch(exception3|exception4 e){statements}
```

Java Exceptions

- In Java, an exception is an ***object***
- There are Java predefined ***exception classes,*** like
  - ArithmeticException 2/0
  - IndexOutOfBoundsException
  - IOException
  - NullPointerException

Some Java Error and Exception Classes

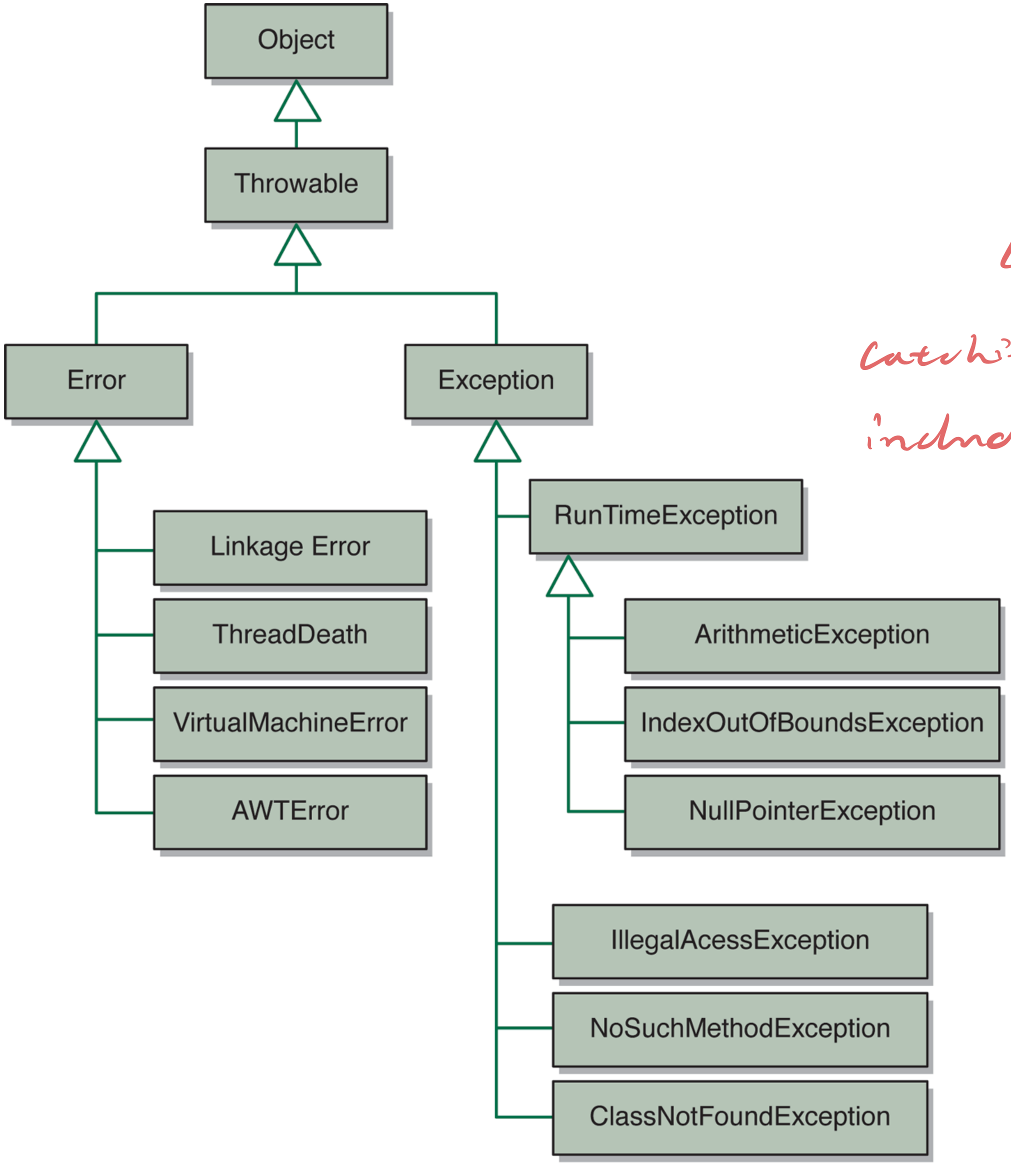


FIGURE 2.7 Part of the `Error` and `Exception` class hierarchy

catch (exception e)

catching generating exception

includes all exception.

Runtime Errors

- Java differentiates between ***runtime errors*** and ***exceptions***
  - Errors are unrecoverable situations, so the program must be terminated
    - Example: running out of memory

Exceptions

Exception: an abnormal or erroneous situation at runtime

Examples:

- Division by zero
- Array index out of bounds
- Null pointer exception

Exceptions can be thrown by the program or by the Java virtual machine, for example for this statement

```
i = size / 0;
```

The virtual machine will throw an exception

Declaring Exception Classes

```
public class EmptyStackException extends
                                    RuntimeException {

    public EmptyStackException (String msg) {
        super (msg);
    }
}
```

=> RuntimeException (msg)

you only need to make the constructure.

Checked and Unchecked Exceptions

- Checked exceptions are checked by the compiler
- Unchecked exceptions are not

Example: Checked Exception

```
import java.io.*;

class Main {
    public static void main(String[] args) {
        try {
           FileReader file = new FileReader("test.txt");
           BufferedReader fileInput = new BufferedReader(file);
           System.out.println(fileInput.readLine());
           fileInput.close();
        }
        catch (FileNotFoundException e) { ... }
        catch (IOException e) { ... }
    }
}
```

Example: Checked Exception

```
import java.io.*;

class Main {
    public static void main(String[] args) {
        FileReader file = new FileReader("test.txt");
        BufferedReader fileInput = new BufferedReader(file);
        System.out.println(fileInput.readLine());
        fileInput.close();
    }
}
```

The compiler gives the error:

```
Main.java:5: error: unreported exception FileNotFoundException; must be
caught or declared to be thrown
                        FileReader file = new FileReader("test.txt");
```

Example: UnChecked Exception

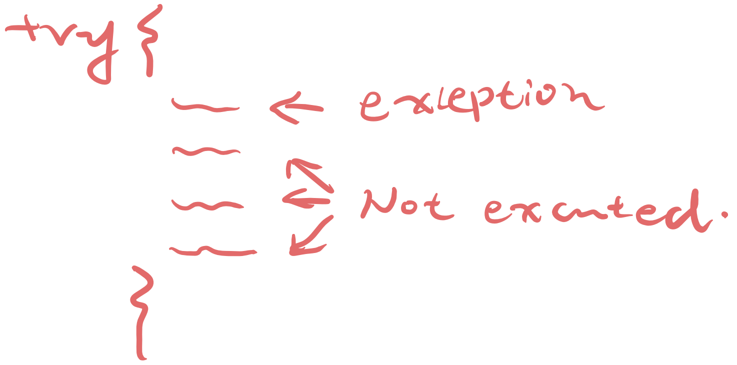
```
class Main {

    public static void main(String[] args) {
        int x = 10;
        int y = 0;
        int z = x / y;
    }
}
```

The compiler does not give an error even though we are dividing by zero.

How do Exceptions Affect the Program Execution?

trigger exception.

- If the offending line is within a try-statement:
  - Lines of code after this line, within the try area, will not execute
  - Line of code below the entire try-catch structure will only execute if the exception is caught



How do Exceptions Affect the Program Execution?

- If the offending line is not within a try-statement:
  - Lines of code after this line will not execute

- An exception can only be caught once. It cannot be caught and then propagate to be re-caught.

A Try-Catch Example with Multiple Catch Statements and a *finally* Block

The try-catch-finally syntax:

```
try {
        code
}
catch(exception1 e) {statements}
catch(exception2 e) {statements}
catch(exception3|exception4 e){statements}
finally {statements}
```

done no matter exception is triggered and handle or not.

Finally Block

The finally block always executes when the try block exits, whether an exception was thrown or not (even if the exception was not caught by any of the catch statements!)

The finally block is executed even if there is a return statement inside the try or catch blocks or if a new exception is thrown.

No *finally* Block

```
PrintWriter out;
try {
    out = new PrintWriter(new FileWriter("OutFile.txt"));
    out.println("Data");
    int x = 5 / 0;
}
catch(FileNotFoundException e) {…}
catch(IOException e) {…}
if (out != null) out.close();
```

← not excuted because the exception haven't been handled yet.

The exception caused by dividing 5 by 0 will not be caught, so the statement out.close() will not be executed, so the file will not be closed, and the data will not be stored in it.

Code with *finally* Block

```
PrintWriter out = null;
try {
    out = new PrintWriter(new FileWriter("OutFile.txt"));
    out.println("Data");
    int x = 5 / 0;
}
catch(FileNotFoundException e) {…}
catch(IOException e) {…}
finally  {
  if (out != null) out.close();
}
```

The file will be closed.

```
public class ExceptionExample {
    private static int x = 1;
    private static String s = "";
    public static void main(String[] args) {
        try {
            method1(2);
            method1(1);
            x = x + 3;
        }
        catch (Exception1 e) {x = 0;}
        catch (Exception2 ex) {x = x + 5;}
        System.out.println(x+", "+s);
    }
```

not called. because triggered exception.

x=7.

```
private static void method2(String str) {
    if (str.length() > 0) ++x;
    else
        throw new Exception1("Empty string");
    s = "hello";
    throw new Exception2("Long string");
}
```

"", "hello"

x=2

is passed to method1

Because no try-catch, the exception

passed to method 1.

2. 1

```
    private static void method1(int param) throws Exception1, Exception2 {
        try {
            if (param == 1) method2("hello");
            else method2(s);
            ++x;
        }
        catch (Exception1 e) {
            System.out.println(e.getMessage());
            s = "hi";
        }
}
```

-> Because an exception throws, this line of code is never executed.

method 1 could not handle exception 2, so it is passed to main method.

Hand-trace through this code. What is printed by the program?

Empty String.
7, hello

output.