Exceptions

- Reading: Savitch, Chapter 8

Objectives

- To learn what an exception is.
- To learn how to handle an exception.

What is an Exception?

- An exception is an object that describes an unexpected situation.
- **Definition:** An *exception* is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

Example

```
int a = 4, b = 4;
int [ ] intAy = new int[4];
intAy[0] = a/(a - b);
//this will generate and throw an ArithmeticException
//object

for(int k = 1; k <= 4; k++)
{       intAy[k] = a * k;       }
//this will generate and throw an
//ArrayIndexOutOfBoundsException object
```

- Exceptions are *thrown* by a program or the run time environment, and may be *caught* and *handled* by another part of the program.
- A program can therefore be separated into a normal execution flow and an exception execution flow.

Exception Classes in API

- Many exceptions have been defined in *java.lang*.
- java.lang.Throwable is the top of the hierarchy of exception and error classes.

- Throwable has two direct subclasses
    - java.lang.Exception
    - java.lang.Error
- All exception classes are subclasses of java.lang.Exception.

Example

java.io.EOFException

java.io.FileNotFoundException

java.lang.NumberFormatException

etc.

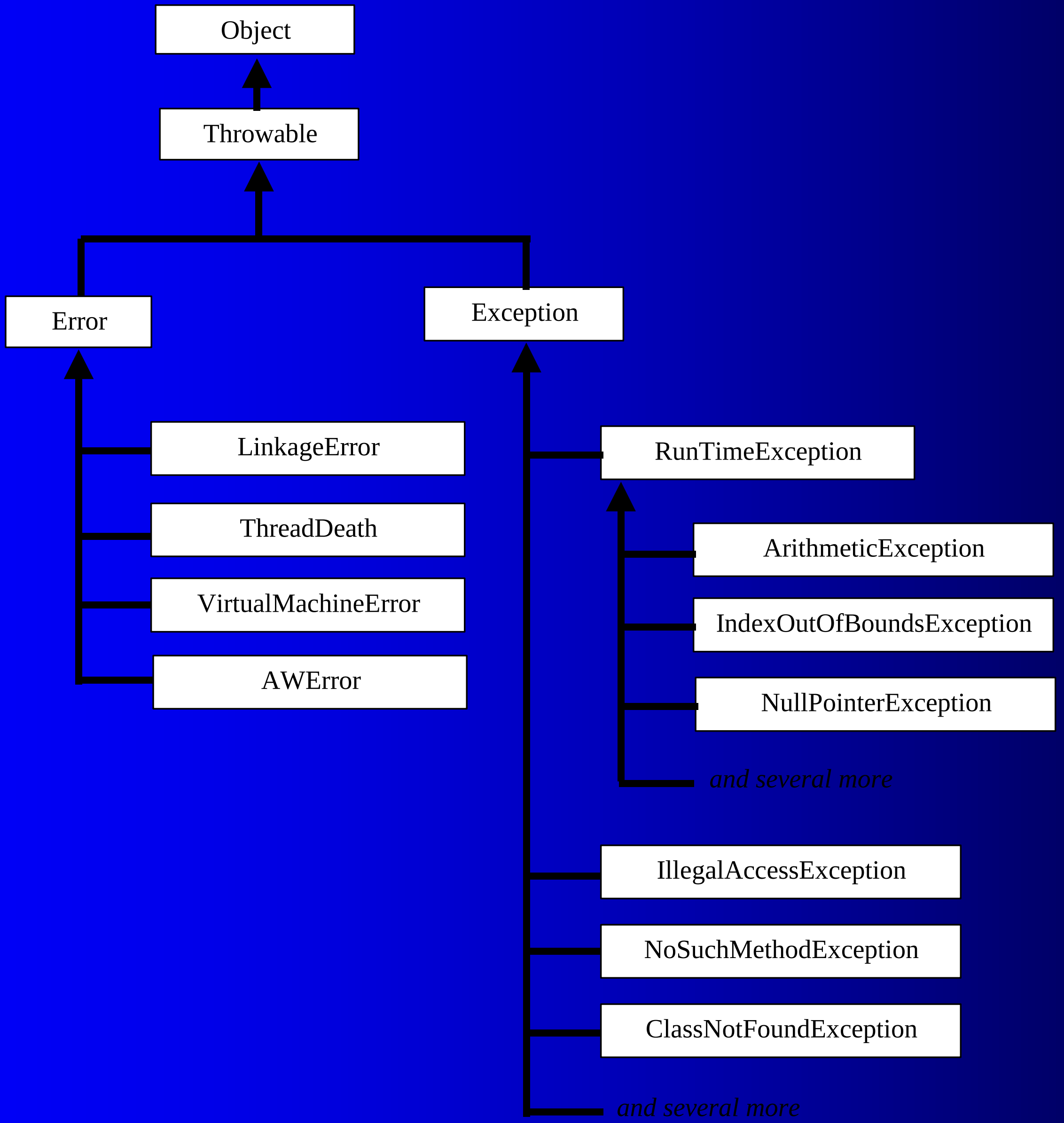

Figure 8.1 **Part of the** Error **and** Exception **class hierarchy**

Checked and Unchecked Exceptions

- Exceptions are divided into two categories – checked and unchecked.

- All subclasses of RuntimeException represent unchecked exceptions. All other exceptions are checked exceptions.

- A checked exception must be handled in the program. While handling unchecked exception is optional.

- A programmer can also define their own exception classes (will be discussed later).

- A program can deal with an exception in one of three ways:

  - ignore it (for unchecked exceptions only)

  - handle it where it occurs.

  - handle it in an another place in the program (propagate it).

Ignore an Exception

- If an unchecked exception is ignored by the program, the program execution will be terminated and an appropriate message will be displayed.

Example

```
//IgnoreEx.java
public class IgnoreEx {
    public static void main(String [ ] args) {
        int a = 4, b = 4;
        int intAy;
        System.out.println(“The start of IgnoreEx”);
        intAy  = a/(a - b);
        System.out.println(“The end of IgnoreEx”);
    }
}
```

The program execution will be

```
% java IgnoreEx
The start of IgnoreEx
Exception in thread "main"
    java.lang.ArithmeticException: / by zero
        at IgnoreEx.main(IgnoreEx.java:7)
```

- In the circumstance where a checked exception cannot be ignored, then a throws statement can be used to throw the exception.

- The syntax of the throws statement is

```
throws exceptionObject;
```

Example

```
public static void main(String [ ] args)
    throws IOException, FileNotFoundException {
... ...
}
```

Handle an Exception Whenever It is Thrown

- The try…catch…finally statement can be used to handle exceptions whenever they are thrown.

- The syntax of the statement is as follows.

```
try {
    statementTry; //exceptions may be thrown from here
}
catch(Exception1 e1) {
    statementException1; //handle the exception e1
}
catch(Exception2 e2) {
    statementException2; //handle the exception e2
}
… …
finally {
    statementFinally;   //the code will always be executed
}
```

- statementTry is the code segment which may throw exceptions.
- Each catch clause has an associated exception type.
- Once an exception is thrown, it will be compared with each catch clause. The statement in the first matched catch clause will be fired.

- The statementFinally will always be executed.

  - If no exception is generated, statementFinally is executed after statementTry in the try block.

  - If an exception is generated, statementFinally is executed after the statement in the appropriate catch clause is completed.

- catch and finally clauses are optional .

Example

```
//CatchEx.java
import java.io.*;

public class CatchEx {
    public static void main(String [ ] args) {
        System.out.println("The start of CatchEx");
        private final int N = 3;
        private int intAy[ ] = new int [N];
```

```
try {
      BufferedReader stdin = new BufferedReader
                (new InputStreamReader (System.in));
      for (int i = 0; i <= N; i++) {
            System.out.println ("Input an integer:");
            intAy[i] = Integer.parseInt (stdin.readLine());
      }
}
catch(IOException e1){
    System.out.println("IOException: " +
    e1.getMessage());
}
```

```
catch(NumberFormatException e2){
    System.out.println("NumberFormatException: " +
    e2.getMessage());
}
catch(ArrayIndexOutOfBoundsException e3){
    System.out.println("ArrayIndexOutOfBounds: " +
    e3.getMessage());
}
```

```
        finally {
            System.out.println("Printed from the finally”
                                    + “statement");
        }
        System.out.println("The end of CatchEx");
    } // end of main
} // end of class
```

public String getMessage()

is a method of the Throwable class. It returns a string which is a brief description of the exception.

The program execution:

```
% java CatchEx
The start of CatchEx
Input an integer:
8
Input an integer:
k
NumberFormatException: k
Printed from the finally statement
The end of CatchEx
```

```
% java CatchEx
The start of CatchEx
Input an integer:
1
Input an integer:
2
Input an integer:
3
```

```
Input an integer:
4
ArrayIndexOutOfBounds: 3
Printed from the finally statement
The end of CatchEx
```

Summary

- *Throwing an exception*: either Java itself or your code signals when something unusual happens
- *Handling an exception*: responding to an exception by executing a part of the program specifically written for the exception
  – also called *catching an exception*
- The normal case is handled in a `try` block
- The exceptional case is handled in a `catch` block
- The catch block takes a parameter of type `Exception`
  – it is called the *`catch`-block parameter*
  – `e` is a commonly used name for it
- If an exception is *thrown*, execution in the `try` block ends and control passes to the `catch` block(s) after the `try` block