Exceptions & File Handling

EXCEPTION HANDLING

What is an Exception?

Exceptional events are problems that arise during the execution of a program that **disrupt the normal or expected flow of the program.**

EXCEPTION HANDLING

What Kinds of Exceptions Are There?

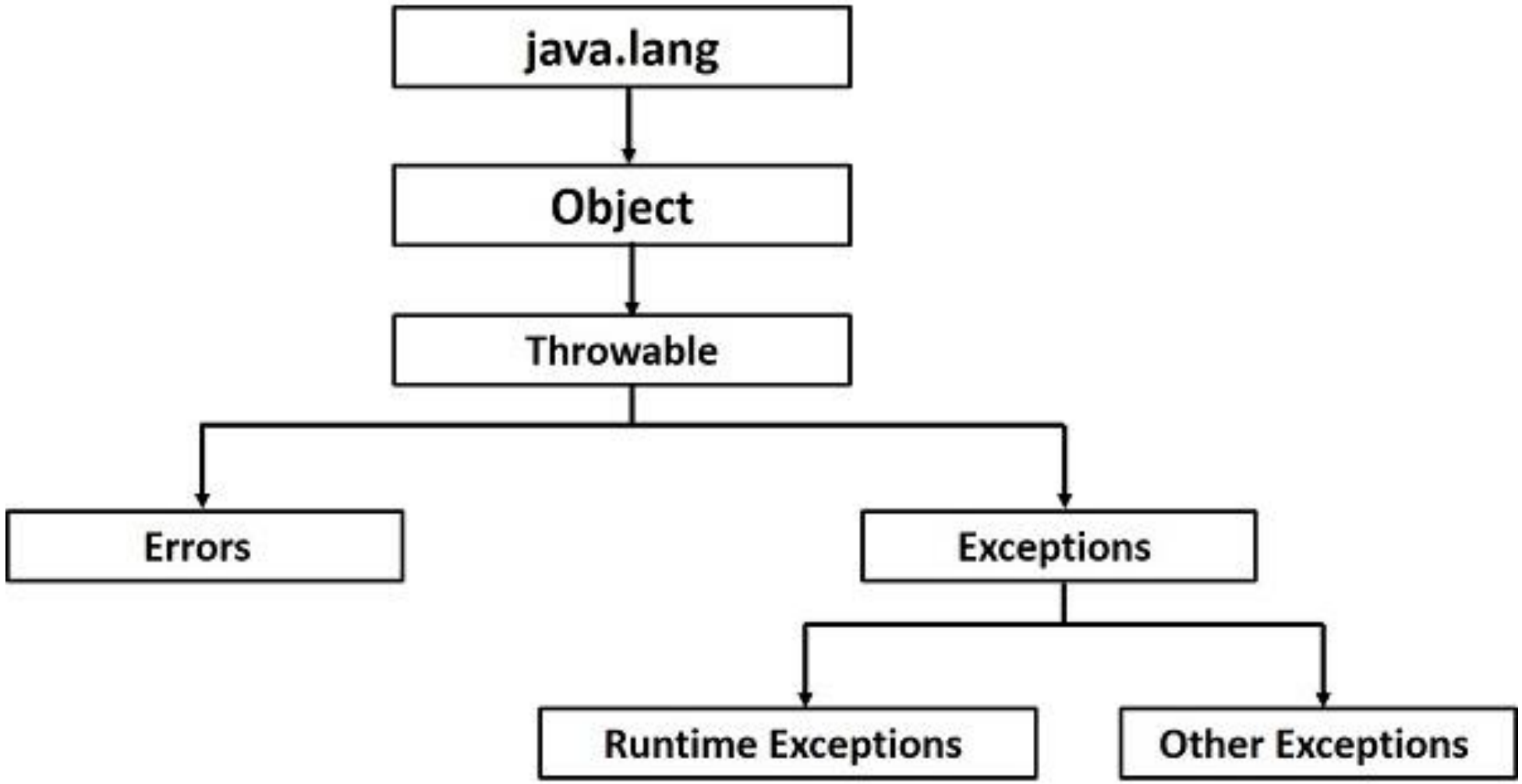
1. Checked Exceptions – occurs at compile time
2. Unchecked Exceptions – runtime errors
3. Errors – unavoidable, unforeseen errors

EXCEPTION EXAMPLES

Checked Exceptions – variable declaration / initialization, missing semi-colons; block mismatches,

Unchecked Exceptions – *ArrayIndexOutOfBoundsException*, *ArithmeticException*, *FileNotFoundException*

Errors – JVM Out of Memory


java.lang
Object
Throwable
Errors
Exceptions
Runtime Exceptions
Other Exceptions

EXCEPTION HANDLING

When a program encounters an exception (“error”), the block-level method where the exception occurs will attempt to deal with the error by throwing it “up the stack” until it reaches a block that properly handles that exception. If no such block occurs, the main function will throw the error and stop the program.

EXCEPTION HANDLING

Using Try-Catch Block to Handle Exceptions

The developer can handle exceptions by surrounding the potential problem statements with a **Try**-block and anticipate exceptions in the **Catch**-block. This handles the exception locally and allows the application to continue to execute.

```
…
   try {
       statements that may throw and exception…
   } catch (AnticipatedException e) {
       statements to execute if exception found
   }
…
```

FILE HANDLING

What is a File Handling?

JDK has a File IO library that enables us to work heavily with files. We can open, read, and write files.

FILE HANDLING

- Read external data source (Excel, CSV)
- Read project plan instructions (XML)
- Write data to a file

READING A FILE

1. Use the File class (java.io.File) to create File object
2. Open the File
3. Read the File
   Add Appropriate Exceptions
   Perform business logic
4. Close the File

SCANNER CLASS

Library: java.util.Scanner

- Lightweight way to read data from known text file
- Reads a File object
- Run a loop to read data (exit at end of file)

SCANNER CLASS

Available Methods

.next() – reads next character

.nextLine() – reads next line

.nextInt() – many other next data types

.hasNext() – boolean expression evaluating T / F

1. Opening the File

```
String fileName = “filedirectory.txt”;
File textFile = new File(fileName);
Scanner in = new Scanner(textFile);
```

2. Reading Data from File

```
String data = in.nextLine();
```

3. Closing the File

```
in.close
// close the Scanner object
```

File Directories

1. Direct Path:
   “C:/Users/..../filename.txt”
   “C:\\Users\\....\\filename.txt”
2. Local Path
   Place Inside Project Root Folder

FILEREADER & BUFFEREDREADER

Library: java.io.FileReader, java.io.BufferedReader

- Versatile method to write from diverse data
- FileReader reads the File
- BufferedReader is a wrapper that efficiently reads

FILEREADER & BUFFEREDREADER

Available Methods

.readLine()

several more…

WRITING TO A FILE

1. Use the File class (java.io.File) to create File object
2. Open the File
3. Write to the File
   Add Appropriate Exceptions
   Perform business logic
4. Close the File

WRITING TO A FILE

Libraries:

1. Try-Catch Blocks
2. Throws Declarations