WEBONISE LAB

Error Handling & Java I/O

Exception handling

What Is an Exception?

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

An exception can occur for many different reasons, including the following:

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Why its important?

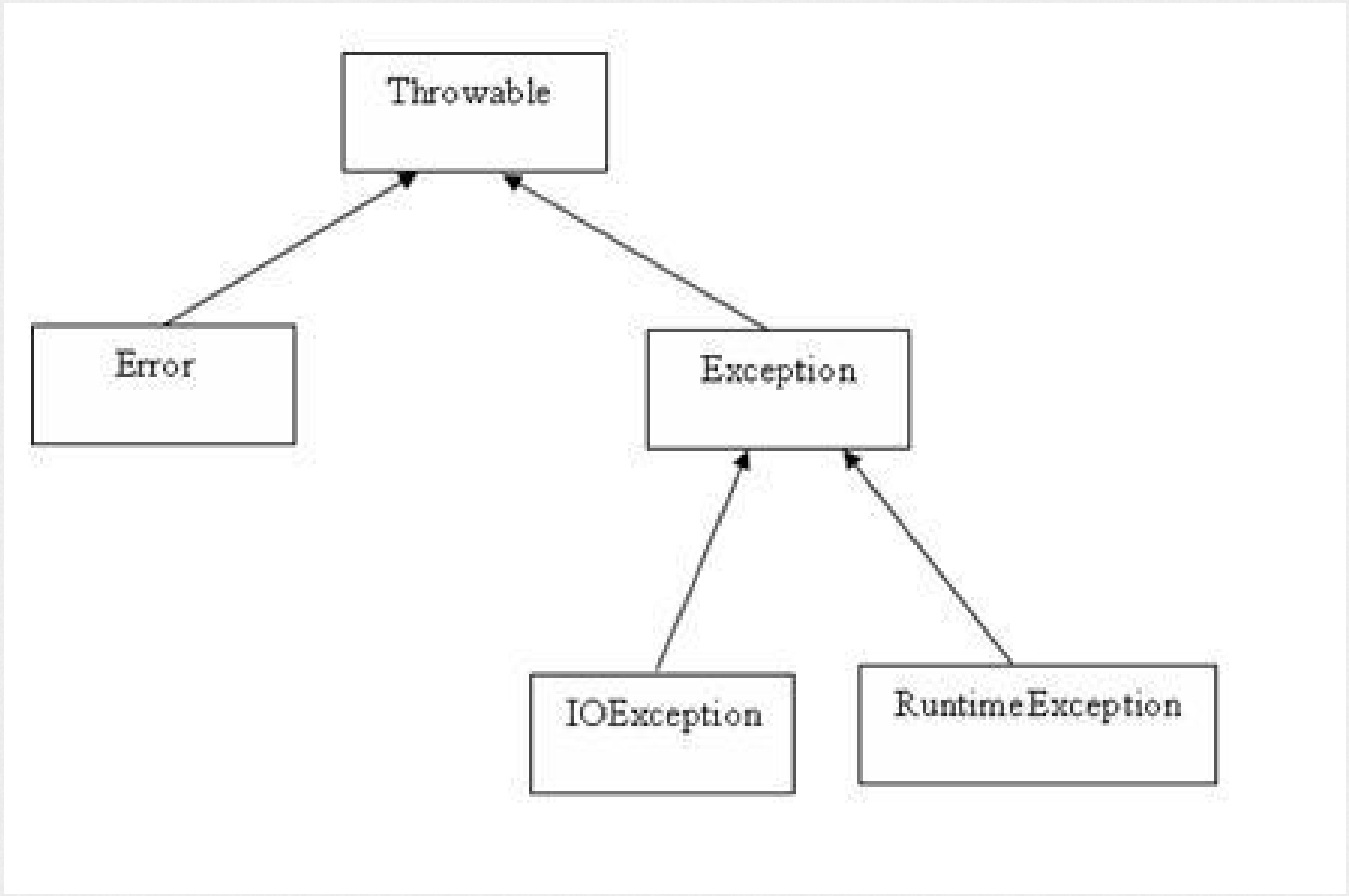
By using exceptions to manage errors, Java programs have the following advantages over traditional error management techniques:

1. Separating Error Handling Code from "Regular" Code
2. Propagating Errors Up the Call Stack
3. Grouping Error Types and Error Differentiation

Categories of exception

- Checked Exception:-
  checked at compile time.
  mostly occurs due to unforeseen conditions.
  Ex:- IOException, SQLException etc.
- Unchecked Exception:-
  Not checked at compile time but at runtime.
  mostly caused due to negligence of programmer.
  Ex:- ArithmeticException, NullPointerException,
  ArrayIndexOutOfBoundException.
- Error:-
  Error is irrecoverable
  Ex:- OutOfMemoryError, VirtualMachineError.
  try, catch, finally, throw, throws.

Exception handling



Catching exceptions

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code.

```
try {
   //Protected code
} catch(ExceptionName e1) {
   //Catch block
}
```

The throw/throws keywords

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.
- You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

```
public class className {
   public void deposit(double amount) throws RemoteException {
      // Method implementation
      throw new RemoteException();
   }
}
```

The finally keyword

- The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.
- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

```
try {
   //Protected code
} catch(ExceptionType ex) {
   //Catch block
} finally {
   //The finally block always executes.
}
```

Java I/O

- The java.io package contains classes to perform input and output (I/O) in Java.

- Java's I/O package mostly concerns itself with the reading of raw data from a source and writing of raw data to a destination. The most typical sources and destinations of data are these:
  - Files
  - Pipes
  - Network Connections
  - In-memory Buffers (e.g. arrays)
  - System.in, System.out, System.error

- A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

How to do I/O?

- Open the stream
  A FileReader is used to connect to a file that will be used for input:

```
FileReader file = new FileReader(fileName);
```

- Use the stream (read, write, or both)

  Manipulate the data as it comes in or goes out

```
BufferedReader reader = new BufferedReader(file);

String s = reader.readLine( );

BufferedWriter writer = new BufferedWriter(output);
writer.write(s);
```

- Close the stream

  A stream is an expensive resource. You must close a stream before you can open it again

Readers and writers

- Streams use bytes as the unit which can be read and written. Bytes are good for hardware, but not good for software.   So need some abstractions Readers and Writers.
- They provide the ability to perform input and output using characters (Strings)

  Readers work with InputStreams

  Writers work with OutputStreams
- InputStreamReader converts an InputStream into a Reader

```
InputStreamReader ISR = new
InputStreamReader(new FileInputStream("..."));
```

- OutputStreamWriter converts an OutputStream into a Writer

```
OutputStreamWriter OSW = new
OutputStreamWriter(new FileOutputStream("..."));
```

Serialization

If you want to read/write an object, it needs to implement the java.io. Serializable interface

- ObjectOutputStream & ObjectInputStream
  - Works like other input-output streams
  - They can write and read Objects.
  - ObjectOutputStream: Serializes Java Objects into a byte-encoded format, and writes them onto an OutputStream.
  - ObjectInputStream: Reads and reconstructs Java Objects from a byte-encoded format read from InputStream

Serialization

- Serialization can be used in.
    - Remote Method Invocation (RMI), communication between objects via sockets. (Marshaling and unmarshaling objects)
    - Archival of an object for use in a later invocation of the same program.

- Objects to be serialized
    - Must implement Serializable interface
    - Non-persistent fields can be marked with transient keyword

Assignment

1. Write a program to search string in a file.

2. Write a program to to copy the content of one file to another

Thank You