CHAPTER 11

EXCEPTION HANDLING IN JAVA

An exception is an unusual (but generally expected!) event that can be detected either by hardware or by software and that may require special processing. An exception is an abnormal condition that arises in a code sequence at run time. In other words, an exception is a run-time error. An event is a notification that something specific has occurred, such as a mouse click on a graphical button. An event handler is a segment of code that is executed in response to the occurrence of an event.

11.1. INTRODUCTION

Exception is a problem that arises in the program during execution. An exception is an event, which occurs during the execution of a program. When such an event occurs within a java method, the method creates an exception object and hands it off to the runtime system. An exception can occur for many different reasons, some of them are caused by user error, programmer error or by physical resources that have failed in some manner, and some of them are as follows:

(i) Entered invalid data: if a user has entered invalid data.
(ii) File not found: if a file that needs to be opened cannot be found.
(iii) Errors generated by the runtime environment: JVM has run out of memory.
(iv) Connection fail: A network connection has been lost in the middle of communications.

All exception classes are packaged in java.lang package. The Exception class is a subclass of the Throwable class. Other subclass is called Error which is derived from the Throwable class.

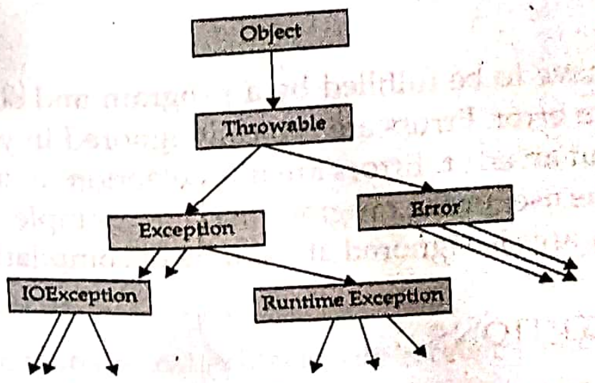


Fig 11.1: Throwable class

Following is the list of important methods available in the Throwable class.

Method	Description
getMessage()	Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
getCause()	Returns the cause of the exception as represented by a Throwable object
toString()	Returns the name of the class concatenated with the result of getMessage()
printStackTrace()	Prints the result of toString() along with the stack trace to System.err, the error output stream.
public StackTraceElement [] getStackTrace()	Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
public Throwable fillInStackTrace()	Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

11.2. TYPES OF EXCEPTION HANDLING

Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. There are three types of exceptions:

(i) Checked Exception

Checked exceptions are subclasses of Exception that represent invalid conditions in areas outside the immediate control of the program for example invalid user input, database problems, network outages and absent files. These exceptions cannot simply be ignored at the time of compilation. Checked exceptions in Java extend the java.lang.Exception class.

(ii) Unchecked Exception

This is also known as Runtime exceptions. A runtime exception is an exception that occurs anywhere in a program that probably could have been avoided by the programmer. Runtime exceptions are ignored at the time of compilation. Thus, the compiler does not require that you catch or specify runtime exceptions.

(iii) Errors

Some conditions that have to be fulfilled by a program and sometimes are not fulfilled, which is called program error. Errors are typically ignored in your code because you can rarely do anything about an error. Errors are not exceptions at all, but problems that arise beyond the control of the user or the programmer. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

11.3 COMMON EXCEPTIONS

The following are the common exception classes.

(i) **Arithmetic Exception:** it is thrown, when a program attempts to perform divide by zero.

(ii) **String Index Out of Bounds Exception:** it is thrown when a program attempts to access a character at a non-existent index in a String.

(iii) **Null Pointer Exception:** it is thrown when JVM attempts to perform an operation on an Object that points to null or no data.

(iv) **Class Not Found Exception:** it is thrown when a program can not find a class it depends at runtime.

(v) **Array Index Out of Bounds Exception:** it is thrown when a program attempts to access an index of an array that does not exist.

(vi) **Number Format Exception:** it is thrown when a program is attempting to convert a string to a numerical data type.

(vii) **IOException:** IOException class is contained in java.io, but it is thrown when the JVM failed to open an I/O stream.

11.4 CATCHING AND HANDLING EXCEPTIONS

Exception handling is managed by five keywords: try, catch, throw, throws and finally. Briefly, here we have to explain the work.

11.4.1 Try/catch block

A try/catch block is placed around the code that might generate an exception. Program statements that you want to monitor for exceptions are contained within a try block. The try block is immediately followed by one or more catch blocks. Code within a try/catch block is protected code. If an exception occurs within the try block, it is thrown. A method catches an exception using a combination of the try and catch. Each catch block specifies the type of exception it can catch and handle.

Syntax:

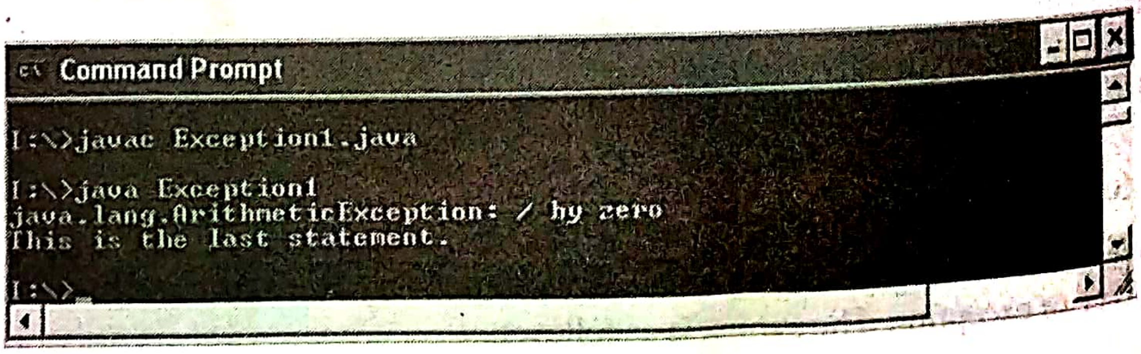
Syntax of try/catch block is as follows:

```
try
{
    //Statements
}
catch (Exception e)
{
    //exception handler statements
}
```

Multiple catch Blocks

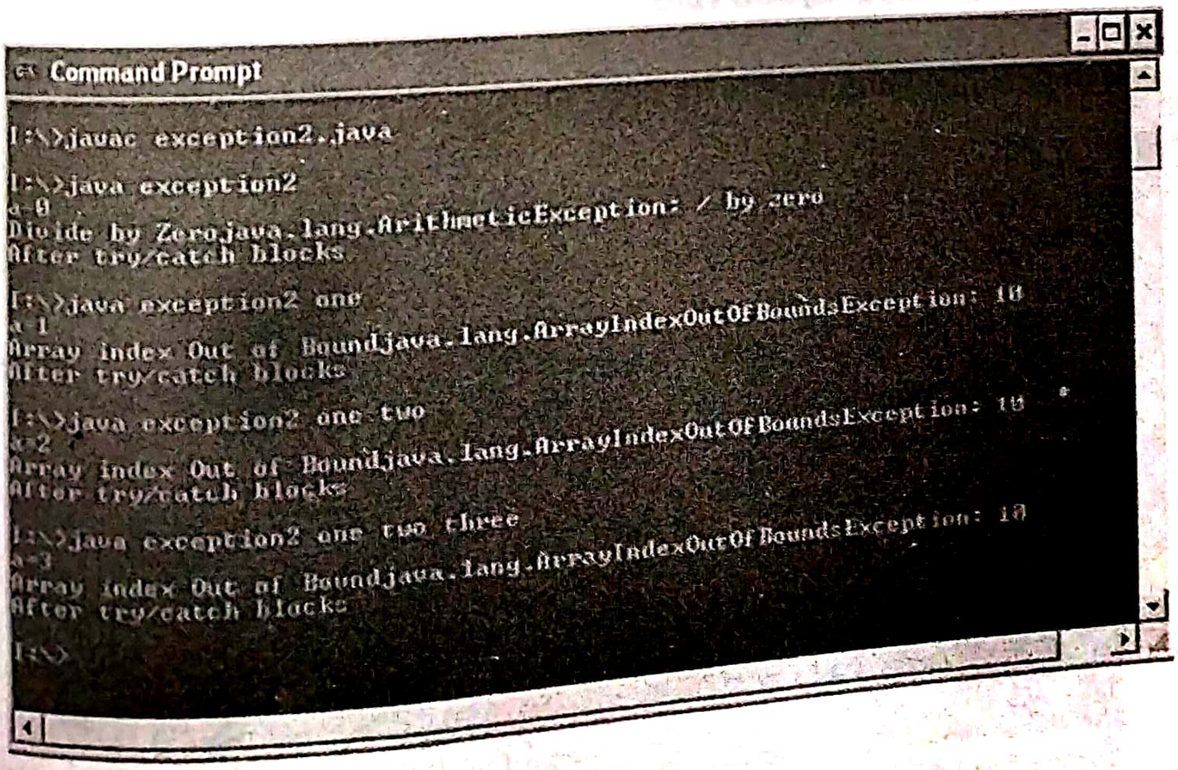
We can use more than one catch block within one try block, each of them catching a specific type of exception. The syntax is as follows:

```
try
{
   //Statements
}
catch (Exception1 e1)
{
   //exception handler statements
}
catch (Exception2 e2)
{
   //exception handler statements
}
catch (Exception3 e3)
{
   //exception handler statements
}
//program for exception
class Exception1
{
public static void main(String args[])
{
int x, y;
try
{
x = 0;
y = 10 / x;
System.out.println("Error in output");
}
catch (ArithmeticException e)
{
System.out.println(e);
}
System.out.println("This is the last statement.");
}
}
```



```
I:\>javac Exception1.java

I:\>java Exception1
java.lang.ArithmeticException: / by zero
This is the last statement.

I:\>
```

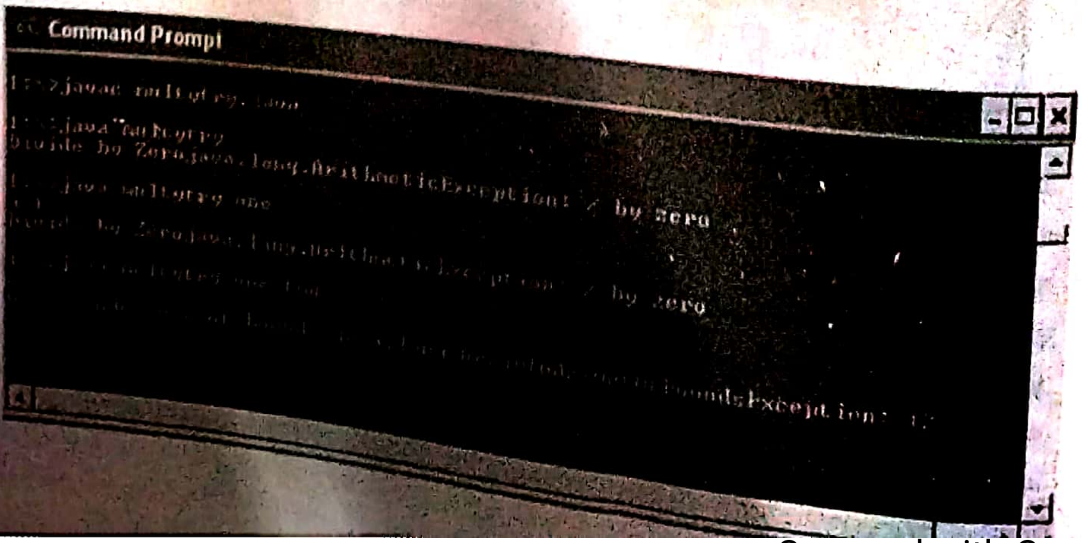
```
//program for exception
class exception2
{
public static void main(String args[])
{
try
{
int a=args.length;
System.out.println("a="+a);
int b = 10/a;
int c[]={3};
c[10]=87;
}
catch(ArithmeticException e)
{
System.out.println("Divide by Zero" + e);
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Array index Out of Bound"+e);
}
System.out.println("After try/catch blocks");
}
}
```



```
I:\>javac exception2.java

I:\>java exception2
a=0
Divide by Zerojava.lang.ArithmeticException: / by zero
After try/catch blocks

I:\>java exception2 one
a=1
Array index Out of Boundjava.lang.ArrayIndexOutOfBoundsException: 10
After try/catch blocks

I:\>java exception2 one two
a=2
Array index Out of Boundjava.lang.ArrayIndexOutOfBoundsException: 10
After try/catch blocks

I:\>java exception2 one two three
a=3
Array index Out of Boundjava.lang.ArrayIndexOutOfBoundsException: 10
After try/catch blocks

I:\>
```

//mult

```
public static void main(String args[])
{
try
{
int a=args.length;
int b=10/a;
System.out.println("a="+a);
try
{
if(a==1)
{
a = a/(a-a);
}
if(a==2)
{
int c[]={1};
c[12]=87;
}
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Array index out-of-bounds:"+e);
}
}
catch(ArithmeticException e)
{
System.out.println("Divide by Zero"+e);
}
}
}
```



```
[illegible]>javac [illegible].java

[illegible]>java [illegible]
Divide by Zerojava.lang.ArithmeticException: / by zero

[illegible]>java [illegible] one
[illegible]
Divide by Zerojava.lang.ArithmeticException: / by zero

[illegible]
[illegible]
[illegible]OutOfBoundsException: [illegible]
```

11.4.2. The throw clause

Throw keyword is used to manually throw an exception. To throw an exception find the appropriate exception class. Now create an object of that class and use the throw statement. The flow of execution immediately stops after the throw statement; any subsequent statements are not executed.

For example.

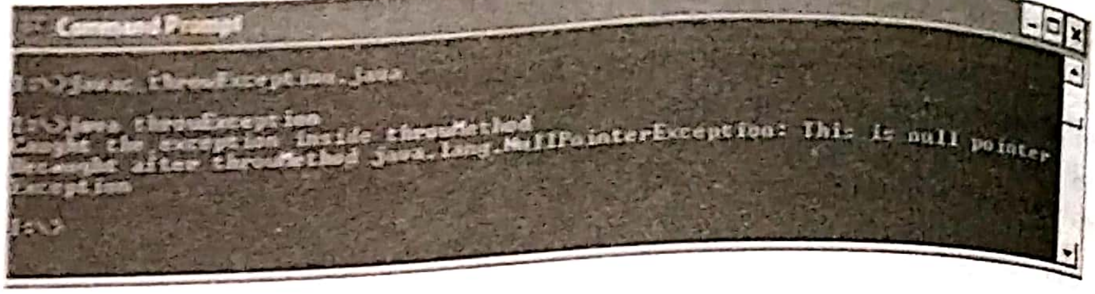
throw new ArithmeticException;

throw is a keyword used to thrown an exception .The general form of throw is shown below:

throw instance;

where instance is an object of type Throwable.

```
// throw Implementation
class throwException
{
static void throwMethod()
{
try
{
throw new NullPointerException("This is null pointer Exception");
}
catch(NullPointerException e)
{
System.out.println("Caught the exception inside throwMethod");
throw e;
}
}
public static void main(String args[])
{
try
{
throwMethod();
}
catch(NullPointerException e)
{
System.out.println("Recaught after throwMethod " + e);
}
}
}
```



```
Command Prompt
I:\>javac throwException.java

I:\>java throwException
Caught the exception inside throwMethod
Recaught after throwMethod java.lang.NullPointerException: This is null pointer
Exception

I:\>
```

11.4.3. The throws Clause

Throws is an alternate way is to indicate that a method may possibly throw an exception. Any exception that is thrown out of a method must be specified as such by a throws clause. This is possible by adding the throws keyword after the signature of the method and followed by the name of one or more exceptions. The Syntax is as follows.

```
public methodname() throws ExceptionName
{
//statements
}
```

In case of multiple exceptions are to be thrown, they are all put in the throws using commas.

```
public methodname() throws exception1,exception2, exception3
{
  //statements
}
```

11.4.4. The finally clause

The statements that absolutely must be executed before a method returns is put in a finally block. The finally clause is the optional clause of try/catch block. All statements within the finally clause are executed hundred percent. It is not depend of the fact that an exception is thrown or not. The statements within the finally clause are clean up process such as close the connection, releasing resources and closing file.

Note: try must be accomplained by at least one catch block or a finally block.

The syntax is as follows.

```
try
{
//Statement
}
catch(Exception e)
{
//exception handler statements
}
finally
```

```
{
  //Statements
}
```

11.4.5. User Define Exception

You can create your own exceptions in Java. Exceptions can be generated manually by your code or by the Java run-time system. Users can create their own exception classes and throw that exception. Manually generated exceptions are used to report some error condition to the caller of a method. The user defined exception classes must derive from the Exception class or any of its subclasses such as IOException. Keep the following points in mind when writing your own exception classes:

(i) All exceptions must be a child of Throwable.

(ii) If you want to write a checked exception that is automatically enforced by the Handle, you need to extend the Exception class.

(iii) If you want to write a runtime exception then you need to extend the RuntimeException class.

```
//user defined exception
import java.io.*;
class MyException extends IOException
{
public MyException(String s)
{
    super(s);
}
}
```

11.6. JAVA'S BUILT-IN EXCEPTIONS

Java defines several exception classes inside the package java.lang. The most general of these exceptions are subclasses of the standard type RuntimeException.

Exception	Cause
ArithmeticException	Arithmetic error condition (for example, divide by zero)
ArrayIndexOutOfBoundsException	Array index less than zero or greater than actual size of array
ArrayStoreException	Object type mismatch between array and object to be stored in array
ClassCastException	Cast of object to inappropriate type
ClassNotFoundException	Unable to load requested class
CloneNotSupportedException	Object does not implement cloneable interface
Exception	Root class of exception hierarchy
IllegalAccessException	Class is not accessible
IllegalArgumentException	Method received illegal argument