

UNIT III EXCEPTION HANDLING AND I/O

Exceptions - exception hierarchy - throwing and catching exceptions – built-in exceptions, creating own exceptions, Stack Trace Elements. Input / Output Basics – Streams – Byte streams and Character streams – Reading and Writing Console – Reading and Writing Files.

3.1 EXCEPTION

Exception is an abnormal condition. An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time, *which disrupts the normal flow of the program's instructions.*

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an 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.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

3.1.1 Exception Handling

The exception handling in java is one of the powerful mechanisms *to handle the runtime errors* so that *normal flow of the application can be maintained.*

Java smoothly handles various types of exceptions using well-defined exception handling mechanism such as

- ClassNotFoundException
- RuntimeException
- ArrayIndexOutOfBoundsException
- IOException
- FileNotFoundException
- NumberFormatException

3.1.2 Steps Involved In Exception Handling

- Hit the exception
- Throw the exception
- Catch the exception
- Handle the exception

3.1.3 Advantage of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

- statement 1;
- statement 2;
- statement 3;//exception occurs
- statement 4;
- statement 5;

Suppose there is 5 statements in your program and there occurs an exception at statement 3, rest of the code will not be executed i.e. statement 4 & 5 will not run. If we perform

exception handling, rest of the statement will be executed. That is why we use exception handling in java.

3.1.4 Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

1. Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2. Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3. Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

3.1.5 Uncaught Exceptions

When we don't handle the exceptions, they lead to unexpected program termination. Lets take an example for better understanding.

```
class UncaughtException
{
    public static void main(String args[])
    {
        int a = 0;
        int b = 7/a;    // Divide by zero, will lead to exception
    }
}
```

This will lead to an exception at runtime, hence the Java run-time system will construct an exception and then throw it. As we don't have any mechanism for handling exception in the above program, hence the default handler will handle the exception and will print the details of the exception on the terminal.

name and description of Exception

java.lang.ArithmeticException: / by zero

at UncaughtException.main(UncaughtException.java:4)

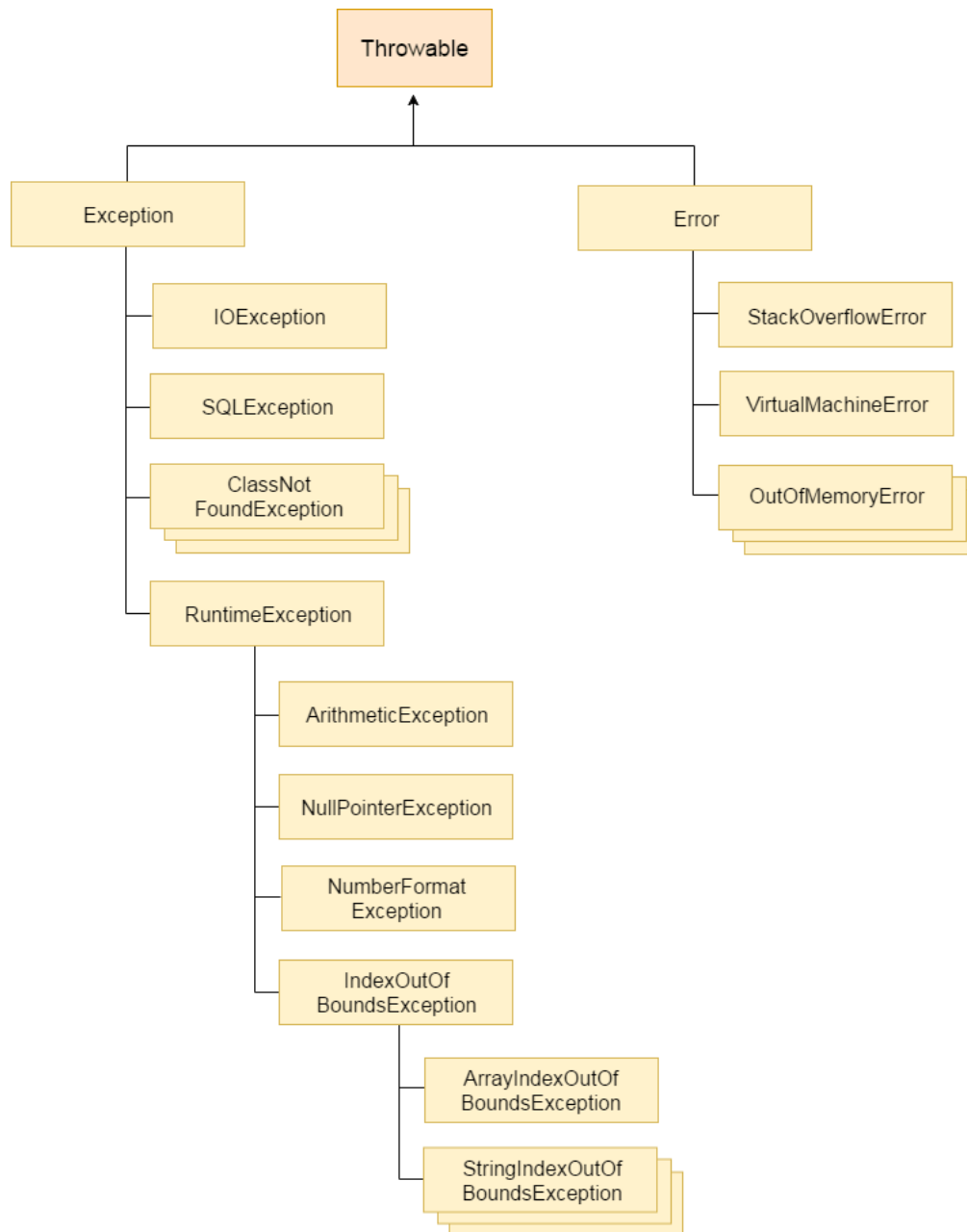
class name file name Stack Trace
(line at which exception occurred)

3.2 EXCEPTION HIERARCHY

All exception classes are subtypes of the `java.lang.Exception` class. The exception class is a subclass of the `Throwable` class. Other than the exception class there is another subclass called `Error` which is derived from the `Throwable` class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.

The `Exception` class has two main subclasses: `IOException` class and `RuntimeException` Class.



3.3 THROWING AND CATCHING EXCEPTIONS

Syntax

```

try
{
//block of code to monitor for an error
}
catch(ExceptionType object)
{
//Exception Handling code here
}
.
.
finally
{
// code placed here will always executed
}

```

Example Program: Command Line Argument

```

import java.io.*;
class DividerDemo
{
public static void main(String args[])
{
try
{
int a=Integer.parseInt(args[0]);
int b=Integer.parseInt(args[1]);
System.out.println("Quotients" + a/b);
}
catch(ArithmeticException e)
{
System.out.println("Error in denominator");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Error in index values");
}
catch(NumberFormatException n)
{
System.out.println("Data type error");
}
finally
{
System.out.println("Finally Block");
}
}
}

```

Output:

```
C:\java\jdk\bin>javac DividerDemo.java
```

```
C:\java\jdk\bin>java DividerDemo 1 0
Error in denominator
Finally Block
```

```
C:\java\jdk\bin>java DividerDemo
Error in index value
Finally Block
```

```
C:\java\jdk\bin>java DividerDemo 2 2
Quotient=1
Finally Block
```

```
C:\java\jdk\bin>javac DividerDemo a b
Data type error
Finally Block
```

3.4 CREATING OWN EXCEPTIONS or USER-DEFINED EXCEPTION IN JAVA

Java provides us facility to create our own exceptions which are basically derived classes of Exception. For example MyException in below code extends the Exception class.

We pass the string to the constructor of the super class- Exception which is obtained using “getMessage()” function on the object created.

```
class MyException extends Exception
{
    public MyException(String s)
    {
        super(s);
    }
}
public class Main
{
    public static void main(String args[])
    {
        try
        {
            throw new MyException("CSECSE");
        }
        catch (MyException ex)
        {
            System.out.println("Caught");
            System.out.println(ex.getMessage());
        }
    }
}}
```

OUTPUT:
Caught
CSECSE

3.5 STACK TRACE ELEMENTS

- A stack trace is a listing of all pending method calls at a particular point in the execution of a program.
- Stack trace listings are displayed whenever a Java program terminates with an uncaught exception.
 - **PrintStackTrace** method of the Throwable class- to access the text description of a stack trace.
 - **getStackTrace** method to get an array of StackTraceElement objects that can be analyse in the program.
- For example:

```
Throwable t = new Throwable();
StackTraceElement[] frames = t.getStackTrace();
for (StackTraceElement frame : frames)
```
- The StackTraceElement class has methods to obtain the file name and line number, as well as the class and method name, of the executing line of code.
- The **toString** method yields a formatted string containing all of this information.

3.6 STREAMS & I/O (INPUT/ OUTPUT)

- In java, a stream is flow of data (bytes).
- Every stream has source and destination.
- Two fundamental types of streams are input stream and output stream.
 - Output stream writes data into a file (or program or device)
 - Input stream reads data from a file(or program or device)
 - Error stream – eg: *System.err.println("error message");*
- Java program performs Input and Output (I/O) operation through streams.
- The java.io package has plenty of predefined stream classes.
- The java.io package classify the stream classes into two categories such as
 - **Byte Stream**- classes handling input and output in the form of bytes.
 - **Character stream**- classes handling input and output in the form of characters.

3.6.1 ByteStream Classes

1. BufferedInputStream – Buffering Input
2. BufferedOutputStream – Buffering output
3. DataInputStream – Reading primitive streams
4. DataOutputStream – Writing primitive types
5. FileInputStream – Reading from file
6. FileOutputStream – Writing to file
7. InputStream – Performing Input operation
8. OutputStream – Predefined output operations
9. PrintStream – Output Stream that contains print() and println() method.

3.6.2 The Character stream Classes

1. BufferedReader – Buffering Input
2. BufferedWriter- Buffering Output
3. FileReader - Input stream that reads from file.
4. FileWriter - Output stream that writes to file.
5. InputStreamReader – Translating byte stream into a character stream
6. OutputStreamReader – Translating character stream into a byte stream
7. PrintWriter - Output Stream that contain print() and println() method.
8. Writer – Performing output operation
9. Reader – Performing input operation

3.6.3 Syntax for reading character streams

```
BufferedReader br= new BufferedReader( new InputStreamReader (System.in));
                                     (or)
InputStreamReader reader = new InputStreamReader(System.in);
BufferedReader br= new BufferedReader (reader);
```

Example:

```
Class examp
{
public static void main(String args[])
{
int age;
String name;
InputStreamReader reader = new InputStremReader (System.in);
BufferedReader br = new BufferedReader (reader);
age=Integer.parseInt (br.readLine ());
name=br.readLine ();
}}

```

3.6.4 Syntax for Reading byte Stream

```
DataInputStream din = new DataInputStream (System.in);
```

Example:

```
DataInputStream din = new DataInputStream (System.in);  
int a= Integer.parseInt(din.readLine());  
String name=din.readLine();
```

Example program for reading input and displaying output:

```
import java.io.*;  
class StudInfo  
{  
public static void main(String args[])throws IOException  
{  
String name;  
int age,m1,m2,m3,total;  
float avg;  
DataInputStream din = new DataInputStream (System.in);  
System.out.println("Enter the name: ");  
name=din.readLine() ;  
System.out.println("Enter the Age: ");  
age=Integer.parseInt(din.readLine()) ;  
System.out.println("Enter the Marks: ");  
m1=Integer.parseInt(din.readLine());  
m2=Integer.parseInt(din.readLine());  
m3=Integer.parseInt(din.readLine());  
total=m1+m2+m3;  
avg=total/3;  
System.out.println("Name: " + name);  
System.out.println("Age: " + age);  
System.out.println("Average: "+avg);  
}  
}
```

Input:

```
Enter the Name : Raj  
Enter the Age: 21  
Enter the Marks: 69 72 88
```

Output:

```
Name : Raj  
Age : 21  
Avg: 76
```

3.7 FILE HANDLING

File handling in java comes under IO operations. Java IO package `java.io` classes are specially provided for file handling in java. Some of the common file handling operations are;

1. Create file
2. Delete file
3. Read file
4. Write file
5. Change file permissions

3.7.1 Create File

File f = new File("abc.txt");

- This line won't create any physical file, First it will check is there any physical file already available with abc.txt name or not.
- If it is already available then f pointing to that file.
- If it is not already available this line won't create any physical file and just it create a java file object to represent the name abc.txt.

- Code for to create file

```
File f = new File("abc.txt");
System.out.println(f.exists()); //false
f.createNewFile();
System.out.println(f.exists()); //true
```

- Code for to directory file

```
File f = new File("foldername");
System.out.println(f.exists()); //false
f.mkdir();
System.out.println(f.exists()); //true
```

- Code for to create file in specific directory

```
File f = new File("foldername");
f.mkdir();
//File f1 = new File("foldername","demo.txt"); or
File f1 = new File(f,"demo.txt");
f1.createNewFile();
```

```
File f = new File("E:\\xyz","demo.txt");
f.createNewFile();
```

File Class Constructors

1. File f = new File(String name);
2. File f = new File(String subdir, String name);
3. File f = new File(File subdir, String name);

Methods

1. boolean exists();
2. boolean createNewFile();
3. boolean mkdir();
4. boolean isDirectory();
5. String list[]();
6. Long length();
7. Boolean delete();

Write a program to print the names of all files and sub directories present in C:\\folder

```
import java.io.File;
class demo{
public static void main(String args[]) {
int count=0;
File f = new File("C:\\foldername");
String s[]=f.list();
for(String s1:s){
count++;
System.out.println(s1);
}
System.out.println("The total Number : " +count);
}
}
```

Write a program to display only file name

```
import java.io.File;
class demo{
public static void main(String args[]) {
int count=0;
File f = new File("C:\\foldername");
String s[]=f.list();
for(String s1:s){
File f1=new File(f,s1);
If(f1.isFile()){          // for directory isDirectory()
count++;
System.out.println(s1);
}
System.out.println("The total Number : " +count);
}
}
```

3.7.2 FileWriter

We can use FileWriter Object to write character data to the file.

Constructor

1. FileWriter fw = new FileWriter(String name);
2. FileWriter fw = new FileWriter(File f);

The above 2 constructor meant for overwriting existing data. Instead of overwriting if we want to perform append operation then we have to use the following 2 constructor

3. FileWriter fw = new FileWriter(String name, boolean append);
4. FileWriter fw = new FileWriter(File f, boolean append);

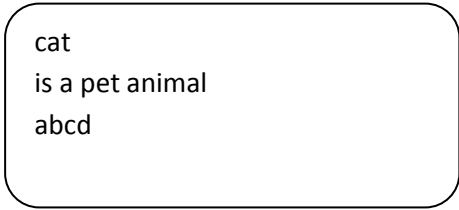
Note: If the specified file is not available then all the above constructors will create that file.

Methods of FileWriter class:

1. write(int ch);
2. write(char[] ch);
3. write(String s);
4. flush();
5. close();

Example program for Writing data into the file

```
import java.io.*;
class Demo{
public static void main(String args[])throws Exception{
FileWriter fw = new FileWriter("abc.txt");
fw.write(99);
fw.write("at\n is a pet animal");
char[] ch1={'a','b','c','d'};
fw.write(ch1);
bw.flush();
bw.close();
}
}
```



```
cat
is a pet animal
abcd
```

3.7.3 FileReader

- We can use FileReader to read character data from the file

Constructor

1. FileReader fr = new FileReader(String fname);
2. FileReader fr = new FileReader(File f);

Methods

1. int read();
 - It attempts to read next character from the file and return its Unicode value.
 - If there is no next character then we will get -1.
2. int read(char[] ch);
 - it attempts to read enough characters from the file into char[] and return the number of character copied from the file into char[].

```
File f = new File("abc.txt");
char ch[] = new char[(int)f.length()];
FileReader fr = new FileReader("abc.txt");
fr.read(ch);
for(char ch1:ch){
System.out.println(ch1);
}
```

Example program for reading data from the file

```
import java.io.*;
class Demo{
public static void main(String args[])throws Exception{
  FileReader fr = new FileReader ("abc.txt");
  int i=fr.read();
  while(i!=-1){
    System.out.println((char)i);
    I=fr.read();
  }
}
}
```

3.7.4 BufferedWriter

Usage of FileWriter and FileReader is not recommended because, while writing Data by FileWriter we have to insert line separator manually, which is varied from system to system. It is difficult to the programmer.

Constructor:

1. `BufferedWriter bw = new BufferedWriter(Writer w);`
2. `BufferedWriter bw = new BufferedWriter(Writer w, int buffersize);`

Methods of BufferedWriter:

1. `Write(int ch)`
2. `write(char[] ch)`
3. `write(String s)`
4. `flush()`
5. `close()`
6. `newline()`

Example program for Writing data into the file

```
import java.io.*;
class Demo{
public static void main(String args[])throws Exception{
  FileWriter fw = new FileWriter("abc.txt");
  BufferedWriter bw=new BufferedWriter(fw);
  bw.write(100);
  bw.newLine();
  char[] ch1={'a','b','c','d'};
  bw.write(ch1);
  bw.newLine();
  bw.write("Durga");
  bw.newLine();
  bw.write("Software Solutions");
  bw.flush();
  bw.close();
}
}
```

```
d
abcd
Durga
Software Solutions
```

3.7.5 BufferedReader

The main advantage of BufferedReader over FileReader is we can read data line by line in addition to character by character, which is more convenient to the programmer.

Constructor

1. BufferedReader br = new BufferedReader(Reader r);
2. BufferedReader br = new BufferedReader(Reader r, int buffersize);

Methods:

1. int read();
2. int read(char ch[]);
3. void close();
4. String readLine();

It attempts to read next line from the file and return it, if it is available. If the next line not available, then it will returns null.

Example program for reading data from the file

```
import java.io.*;
class Demo{
public static void main(String args[])throws Exception{
FileReader fr = new FileReader("abc.txt");
BufferedReader br = new BufferedReader(fr);
String line = br.readLine();
While(line!=null) {
    System.out.println(line);
    line=br.readLine();
}
br.close();
}
}
```

3.7.6 PrintWriter

- It is the most enhanced writer to write character data to the file.
- The main advantage of PrintWriter is we can write any type of primitives type data directly to the file.

Constructor

1. PrintWriter pw = new PrintWriter(String filename);
2. PrintWriter pw = new PrintWriter(File f);
3. PrintWriter pw = new PrintWriter(Writer w);

Methods

- | | |
|----------------------|------------------------|
| 1. print(char ch); | 6. println(char ch); |
| 2. print(int i); | 7. println(int i); |
| 3. print(double d); | 8. println(double d); |
| 4. print(boolean b); | 9. println(boolean b); |
| 5. print(String s); | 10. println(String s); |

Example program

```
import java.io.*;
class Demo{
public static void main(String args[])throws Exception {
PrintWriter pw = new PrintWriter("abc.txt");
pw.write(100);
pw.println(100);
pw.println(true);
pw.println('c');
pw.println("sampe");
pw.flush();
pw.close();
}
}
```

```
d100
true
c
sampe
```

Write a program to merge data from 2 files into a third file

```
import java.io.*;
class Demo{
public static void main(String args[])throws Exception {
PrintWriter pw = new PrintWriter("abc.txt");
BufferedReader br = new BufferedReader(new FileReader("file1.txt"));
String line=br.readLine();
while(line!=null){
pw.println(line);
line=br.readLine();
}
br = new BufferedReader(new FileReader("file2.txt"));
line=br.readLine();
while(line!=null){
pw.println(line);
line=br.readLine();
}
pw.flush();
br.close();
pw.close();
}
```

```
File1.txt
222
333
444
555
```

```
File2.txt
AAA
BBB
CCC
DDD
```

```
File1.txt
222
333
444
555
AAA
BBB
CCC
DDD
```

UNIT III PART-A

1. Give any two methods available in Stack trace Element. N/D 2010
2. Define Assertions and give its syntax. N/D 2010
3. What is an exception? A/M 2011
4. What are stack trace elements? N/D 2012
5. How to throw an exception. A/M 2014

Regulation 2017

Define run time exceptions

What is the use of assert keyword

What happens when the statement: int value=25/0; is executed?

Give an example for reading data from file using File Input Stream.

PART-B

1. Define Exception and explain its different types. With appropriate examples using Java. (8) N/D 2011
2. Discuss on exception handling in detail. (16) A/M 2012
3. What is exception? How to throw an exception? Give an example. (8) A/M 2013
4. What is exception? why it is needed? Describe the exception hierarchy. Write notes on stack trace elements. Give example. (16) N/D 2013 A/M 2016
5. Define Exception and explain its different types with example. (8) A/M 2015
6. Explain the different types of exceptions. (4) A/M 2018
7. Explain the exception hierarchy [Marks 8] N/D 2010
8. Describe briefly about exception hierarchy in Java. (8) A/M 2013
9. Explain the concept of throwing and catching exception in java. [Marks 8] A/M 2011
10. Explain throwing and catching exception. (16) N/D 2012
11. Explain the task of catching exception with example. (8) A/M 2011
12. What is finally class? How to catching exception? Write an example. (8) A/M 2013
13. What do you mean by error handling? Describe the throwing and catching exceptions supported in generic programming. Give example. (16) N/D 2015
14. How do you analyse the stack trace element. (8) A/M 2011
15. What are stack trace Elements? Explain. (8) N/D 2011
16. Write short notes on stack trace elements. (4) A/M 2018
17. Design a java program to handle array index out of bound exception. (8) A/M 2018

2017 Regulation**N/D 2018**

1. Explain the different types of exceptions and the exception hierarchy in java with appropriate example. (13)
2. What are the input and output streams? Explain them with illustration. (13)

A/M 2019

3. Give an example for nested try statement in java source file and explain.
4. Write a note on built-in exception
5. Create an IN file in Java to store the details of 100 students using a STUDENT class. Read the details from IN file, convert all the letters in IN file to lowercase letters and write it into OUT file.

Java Nested try block

The try block within a try block is known as nested try block in java.

Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

```

.....
try
{
    statement 1;

```

```
        statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
.....
```

Java nested try example

```
class Excep6{
    public static void main(String args[]){
        try{
            try{
                System.out.println("going to divide");
                int b =39/0;
            }catch(ArithmeticException e){System.out.println(e);}

            try{
                int a[]=new int[5];
                a[5]=4;
            }
            catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

            System.out.println("other statement");
        }
        catch(Exception e){System.out.println("handeled");}

        System.out.println("normal flow..");
    }
}
```