21/09/23.

Unit - 2.

Exception handling.

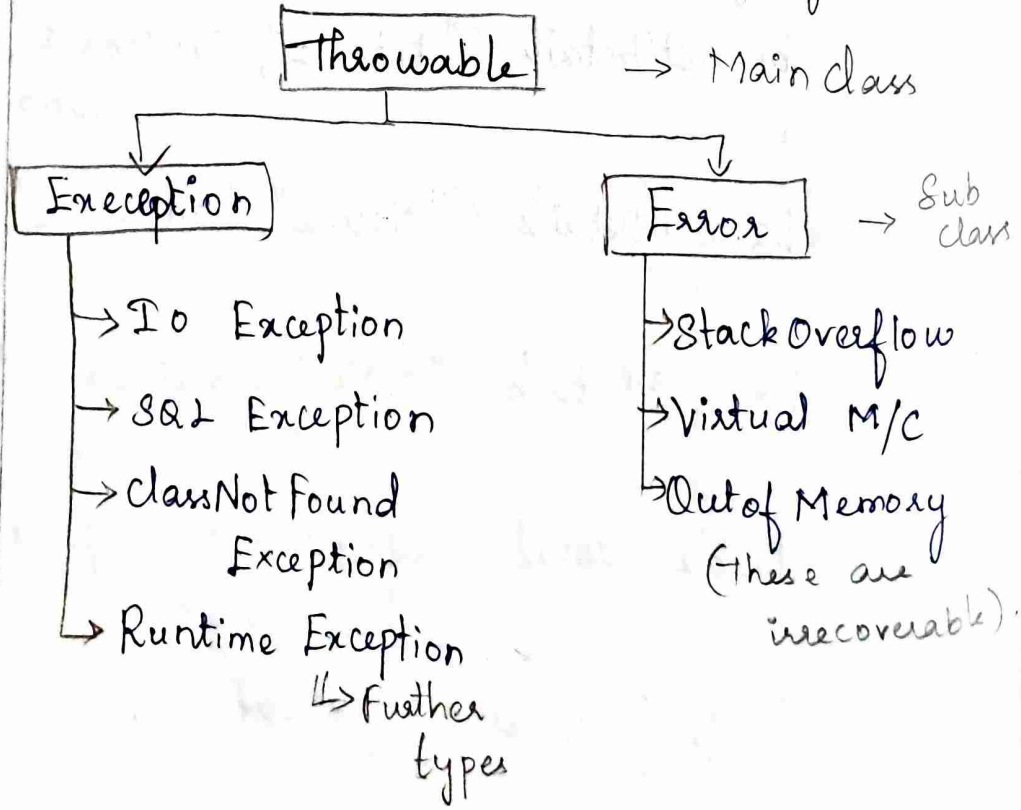
Runtime errors - Exception.

⇒ Exception is an abnormal condition happening at the runtime.

- Exception is the runtime error.
- It is of two types:

1. Checked Exceptions - Errors that occurs at Compile time.
2. Unchecked Exceptions - Error that occurs at Runtime.

Handling the Unchecked Exceptions is called Exception handling.



Runtime Exception

- Arithmetic Exception
- Nullpointer Exception
- NumberFormat Exception
- IndexOutofBounds Exception.
  - ArrayIndexOut of Bounds Exception
  - StringIndex Out of Bounds Exception.

5 keywords to Handle Exception :-

2m

1) try
2) catch → always paired
3) throw
4) throws
5) finally → Sometimes with try & Catch.

Exception cases :-

```
Class A
{
    public static void main (String[] args)
    {
        int a = 50/0;
        System.out.println("a: "+a);
    }
}
```

O/P :

```
Exception in thread "main":
Arithmetic Exception: / by 0.
```

Syntax for try-catch statement :-

```
try
{
   // exception statements;
}
catch (exception handler variable / object)
{
}
```

Implementing try-catch in exceptional cases :-

```
class A {
   Public static void main (String[] args)
   {
      try {
         int a = 50/0;
      }
      catch (Arithmetic Exception e)
      {
         System.out.println(e);
      }
   }
}
```

S.o.p ("Here statements are printed");

O/P :

```
Exception in thread "main":
Arithmetic Exception.
Here statements are printed
```

Syntax for Multiple catch statement :-

```
try
{
    // Exception statements;
} catch (Exception type 1 e1)
  {
  } catch (Exceptiontype 2 e2)
    {
    } catch (Exceptiontype n en)
      {
      }
```

Rules :

-> Any no. of catch stmts. can be given under one try.

-> The 1st catch stmt. must be specific and the other catch stmts. may be declared as general.

-> At a time, only one catch statement can be executed.

Example :

```
class A
{
    public static void main (String[] args)
    {
```

```
try {
    int b = 0, c = 5;
    int a = c/b;
}
catch (Arithmetic Exception e)
{
    System.out.println(e);
}
catch (ArrayIndexOutofBounds Exception
{                                    e1)
    System.out.println(e1);
}
catch (NuMemberFormat Exception e3)
{
    System.out.println(e3);
}
}
}
```

o/p :

```
Exception in "thread" main :
Arithmetic Exception : /by0 .
ArrayIndexOutof Bounds Exception :
NumberFormat Exception.
```

Try- Catch - Finally block :-

Syntax :

```
try
  {
} catch (exception handler e1)
  {
  }
finally
{
    // finally block;
}
```

Example :

```
class A {
    public static void main (String[] args) {
        try {
            String b[3] = {"a", "b", "c"};
            System.out.println (b[4]);
        }
    catch (StringOutofBounds Exception e) {
        System.out.println (e);
    }
    finally
      {
```

```
        System.out.println("Final block");
    }
  }
}
```

O/P :-

```
StringOutofBounds Exception : b[4].
Final block.
```

27/09/2023.

Nested Try statement.

Syntax :

```
try  // outer block
 {
    statement 1;
    statement 2;
   try  // inner block
   {
     statement 3;
     statement 4;
   try
   {
     statement 5;
     statement 6;
   } catch (Exception e1)
     {
     }
   } catch (Exception e2)
```

```
        {
        }
    } catch (Exception e3)
        {
        }
```

Example :

```
class A
{
  public static void main (String [] args)
  {
    try {
        try {
            try{
                int arr[] = {1, 2, 3, 4};
                System.out.println (arr [10]);
            }
            catch (Arithmetic Exception e)
            {
              System.out.println ("Arithmetic
                                   Exception");
              System.out.println ("Inside try
                                   block 2");
            }
        }
        catch (Arithmetic Exception e1)
        {
          System.out.println ("Arithmetic Exception");
```

```
            System.out.println("Inside try block 1");
        }
    }
    catch (ArrayIndexOutofBoundException e)
    {
      System.out.println (e);
      System.out.println ("Inner block try");
    }
  }
}
```

Output :

```
Exception in java thread "main":
java.lang.ArrayIndexOutofBound
Index 4 Out of Bound 10
Inner block.
```

Throwable.

(2M)

Syntax :

```
throw throwableInstance;
```

throwableInstance → Object of throwable class

throwable → Superclass for Exception

Eg.:

```
throw new ArithmeticException ("demo");
```

throw is given below try & catch

Eg. program :

```
class A {
    static void throwdemo ( )
    {
      try
       {
        throw new NullPointer Exception ("Sample");
       } catch (NullPointer Exception e)
        {
         System.out.println ("e);
        } throw e;
    Public static void main (Sting [] args)
    {
       try
       {
          throw demo( );
       }
       catch (NullPointer Exception e)
       {
         System.out.println ("Recaught:" + e);
       }
    }
    }
}
```

O/P:

```
Exception in "thread "main":
java.lang.NullPointer#
Sample
Exception in thread "main":
Recaught : java.lang.NullPointer
```

10/9/23.

Throws

- The keyword "throws" will be associated with the method.

- Syntax:

```
public returntype fn_name() throws Exception
    {                                  handler
    }
```

- Example:

```
public void demo() throws IOException
    {
    }
```

- Should not given inside try & catch

- Eg. Program:

```
class A
{
 public void throwsdemo() throws
                    IllegalAccessException
  {
    System.out.println("Inside throws demo");
    throw new IllegalAccessException("Sample");
  }
public static void main(String[] args)
 {
```

```
try {
    throwsdemo();
}
catch (IllegalAccessException e)
{
    System.out.println(e);
}
}
}
```

o/p:

```
Exception in thread "main":
java.lang.IllegalAccessException
Inside throwsdemo
Sample
```

30/09/2023.

String Handling.

① - keyword : String.

- Eg.: String Constructor

```
String s = new String();
```

```
char ch[] = {'a', 'b', 'c'};
string s = new string(ch);
System.out.println(s);
```

o/p :-

abc

(2). - To execute the sub range of a character array as an initializer, use the following constructor.

Syntax: `String (char ch[], int start, int numofchars);`

Eg.: To print c, d & e from {a, b, c, d, e}, the program is,

```
string (ch, 2, 3);
```

o/p: cde.

Example:

```
class A
{
  public static void main (String[] args)
  {
     char ch[] = {'J', 'a', 'v', 'a'};
     String s1 = new string (ch);
     String s2 = new String (s1);
     System.out.println (s1);
     System.out.println (s2);
  }
}
```

o/p:- Java
Java

③ - String class provides constructor that initialize a string when given a byte array.

Syntax:
```
string (byte ch[]);
string (byte ch[], int start,
                   int no-of chars);
```

Example:

```
class A
{
  public static void main (String[] args)
  {
     byte ascii[] = {65, 66, 67, 68, 69, 70};
     String s1 = new String (ascii);
     System.out.println (s1);
     String s2 = new String (ascii, 2, 4);
     System.out.println (s2);
  }
}
```

O/P:-
```
ABCDEF
CDEF
```

---

④ - String length

Example:

```
class A
{
  Public static void main (String[] args)
  {
```

```
        char ch = {'J', 'a', 'y', 'a'};
        String s1 = new String (ch);
        S.O.p (s1. length ());
    }
}
```

O/P: 4

---

(5) - String Concatenation :-

Eg.:

```
class A
{
    public static void main (String[] args)
    {
        String age = "18";
        System. out. println ("My age is " +
                              age + "years");
    }
}
```

O/P: May age is 18 years.

---

(6). String Concatenation with other Datatypes :-

Eg.:

```
String s = "four" + 2 + 2;
System.out.println (s);
```

O/P: four22 (instead of four4).

⑦. String Comparison :-

- To Compare 2 strings for equality, use "equals()".

- To perform a comparison that ignores case differences, use "equalsIgnoreCase();"

- Eg :-

```
class A
{
  public static void main (String[] args)
  {
    String s1 = "Hello";
    String s2 = "HELLO";
    String s3 = "Hello";
    String s4 = "goodbye";
    String s5 = "GOODBYE";
    System.out.println ("s1.equals(s2));
    System.out.println (s1.equals(s3));
    System.out.println (s4.equals(s5));
    System.out.println (s1.equalsIgnoreCase
                        (s2));
    System.out.println (s4.equalsIgnoreCase
                        (s5));
  }
}
```

O/P :-

```
False
True
False
True
```

(8). - Java provides a string object that contains :-

a) the method to toLowerCase() - Converts all the characters in a string from uppercase to lowercase.

b) the method toUpperCase() - Converts all the characters in a string from lowercase to uppercase.

Eg. :-

```
class A {
    public static void main (String[] args) {
        String s1 = "jayabharathi";
        String s2 = "HARINI";
        System.out.println (toUppercase(s1));
        System.out.println (toLowercase(s2));
    }
}
```

O/p :- JAYABHARATHI
harini

9. String handling function also has 2 methods : i) starts_With() method - determines whether a given string begins with the specified string.

ii) endsWith() - determines whether a given string ends with the specified string.

→ boolean startsWith (String str)
→ boolean endsWith (String str)

10. equals Vs ==

equals () method compares the characters inside the string object

the == operator compares two object references to see whether they refer to the same instance.

11. Searching strings :

the String class provides two methods to search a string for a specified character or a substring.

indexOf (String str)
lastIndexOf ()

12. Concat

s1.concat(s2);

The concatenation of two strings can be done using concat().

Syntax:

String.concat(str);

ex:

s1 = "Hello";

s2 = s1.concat("hai");

13. Modifying a string:

We can extract a substring using substring() function.

It has 2 forms the first one

i) String substring(int indexOf())

Here, the indexOf denotes the index at which the substring starts

ii) The second form of substring allows to specify both the beginning and ending index of the substring

String substring(int indexof, int end

5) String buffer:

String buffer supports modifyable string the string buffer defines 4 constructors

→ String Buffer()
→ String Buffer (int size)
→ String Buffer (String str)
→ String Buffer (char ch)

String Buffer has the following functions:

i) length() and capacity ()
ii) Ensure capacity ()
iii) setlength()
iv) append ()
v) insert ()
vi) reverse ()
vii) charAt (), setcharAt ()
viii) delete (), deletecharAt ()

i) length () and capacity ()

The length() method gives the current length of the string buffer. The capacity method gives the total allocated capacity.

Syntax: int length ()

ex:

```
Class A
{
  Public static void main (String[]args)
  {
    StringBuffer sb = new StringBuffer ("Hello");
    System.out.println ("the String is " + sb);
    System.out.println ("length of sb"
                        sb.length());
    System.out.println ("Capacity "
                        sb.capacity());
  }
}
```

ii) Ensure Capacity ()

Ensure capacity method is used to set the size of the buffer

Syntax: void EnsureCapacity (int capacity)

Here, capacity specifies the minimum size of the buffer.

06/10/2022.

(iii) SetLength :-

```
class A
{
  public static void main (String[] args)
  {
    StringBuffer sb = new StringBuffer ("Jaya");
    String s1 = sb.setLength(2);
    System.out.println ("The String after
                setlength is : " + s1);
  }
}
```

O/P : Ja

Syntax : `void setLength (int len)`

(iv) append :-

The append() method concatenates the string representation of any other type of data to the end of the invoking string buffer object. method.

Syntax :

```
StringBuffer append (String str)
StringBuffer append (int num)
StringBuffer append (object obj).
```

Eg.:

```
class A {
   public static void main (String [] args)
   {
      String A;
      int a = 42;
      StringBuffer sb = new StringBuffer(40);
      s = sb.append("a = ");
      System.out.println (s);
   }
}
```

O/p :- a = 40

(v) Insert :-

The insert() method inserts one string into another.

Syntax : StringBuffer insert (int index, String str)

StringBuffer insert (int index, int num)

StringBuffer insert (int index, char ch)

StringBuffer insert (int index, object obj)

Eg :-

```
Class A {
   Public static void main (String[] args) {
      StringBuffer sb = new StringBuffer ("I java");
```

```
        sb.insert (2, "like");
        System.out.println (sb);
    }
}
```

O/P: I like java

Eg 2:

```
class A {
    public static void main (String [] args)
    {
        StringBuffer sb = new StringBuffer ("I java");
        sb.insert (7, 's');
        System.out.println (sb);
    }
}
```

O/P: I javas.

vii) Reverse

The reverse() method reverses the characters within a StringBuffer object using the StringBuffer object.

Syntax:

StringBuffer reverse ()

Eg.:

```
class A {
    public static void main (String[] args)
```

```
    {
      StringBuffer sb = new StringBuffer("ABCDE");
      sb.reverse();
      System.out.println(sb);
    }
}
```

o/p: EDCBA.

vii) charAt() & setcharAt():-

The value of single character can be obtained from a StringBuffer via charAt() method.

Syntax: `char charAt(int where);`

We can set the value of a character within a StringBuffer object via setcharAt() method.

Syntax: void setcharAt(int where, char ch)

Eg:-

```
class A {
  public static void main(String[] args){
     StringBuffer sb = new StringBuffer("Jaya");
     String s1 = sb.setcharAt(3,'v');
     System.out.println(s1);
  }  char
     String s2 = sb.charAt(1);
}    System.out.println(s2);
  }
}
```

o/p :- Java

J

viii) deleteAt() & deletecharAt().

We can delete the characters within the stringBuffer by using the methods.

Syntax :

StringBuffer delete (int startindex, int endindex)

StringBuffer deletecharAt (int where)

Eg :

```
class A {
  public static void main (String[] args) {
    StringBuffer sb = new StringBuffer ("ABC");
    sb. deletecharAt (1);
    System.out. println (sb);
  }
}
```

o/p :- AC

Unit - 3.

Multithreading.

Thread: A small program in execution.

Multithreading: Many threads can be executed at a time.

Thread can created in 2 ways :-

1. class classname extends Thread.
2. class classname implements Runnable.

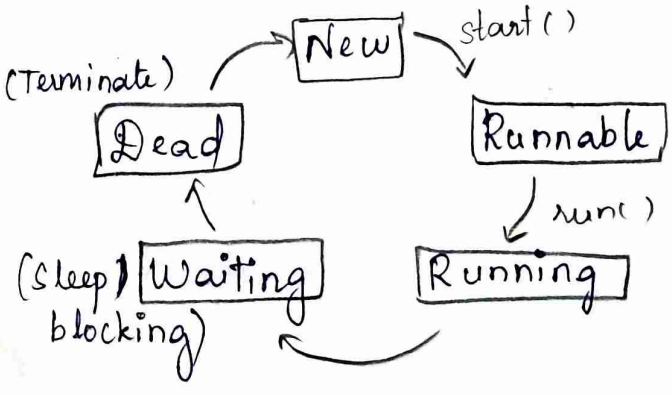
eg: 1. class A extends Thread.
↳ Super class

2. class A implements Runnable.
↳ Interface

Thread methods :

1. setName ( )
2. getName ( )
3. setID ( )
4. getID ( )
5. run ( )
6. start ( )
7. isAlive ( )
8. join ( )
9. setPriority ( ).

Thread states :



Sample program:

```
class A extends Thread {
    public static void run(String[] args)
    {
        try {
            System.out.println("Thread" +
            Thread.currentThread().getID();
                    + "is running");
        } catch (Exception e)
        {
            System.out.println("Exception is
                                caught");
        }
    }
}
public class MultiThread {
    public static void main() {
        int n=8;
        for(int i=0; i<n; i++) {
            A a1 = new A();
            a1.Start(); → invoke the
                          run method
        }
    }
}
```

o/p:
```
Thread 15 is running
Thread 6 is running
Thread 27 is running
Thread 18 is running
Thread 20 is running
Thread 5 is running
Thread 23 is running
Thread 78 is running.
```

Syntax:

```
Thread objname = new Thread(new class());
```

Creating a thread using Runnable Interface:

```
class A implements Runnable
{ public void run()
  {
    try
    {
      System.out.println("Thread" + Thread.
      currentThread().getID(); + " is
                                running");
    }
    catch (Exception e) {
      System.out.println("Exception is
                          caught");
    }
```

```
    }
}
public class Multithread {
    public static void main (String[] args) {
        int n = 8;
        for (int i=0; i<n; i++)
        {
            Thread a1 = new Thread (new A());
            a1.start();
        }
    }
}
```

o/p :- Thread 15 is running
⋮ → 8 times.

11/10/2023

Thread Synchronization.

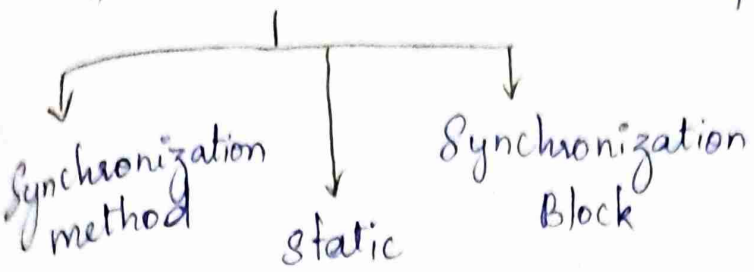
In thread, multiple threads access shared resources. Java synchronization is the better option where we want to allow only one thread to access the shared resource..

(2m) The Synchronization advantages are:

1. To prevent the thread interference.
2. To prevent the consistency problem.

There are 2 types of thread Synchronization :

i) Mutual exclusive ii) Cooperation



Synchronization

Without Synchronization method :-

```
class Table
{ public void ~~stat~~printTable (int n)
  {
    for (i=1; i<=5; i++)
    {
      System.out.println (i * n);
      try {
          thread.sleep (400);
      } catch (Exception e)
        {
          System.out.println (e);
        }
  }
}

class mythread1 extends Thread
{
    Table t;
```

```
    Mythread1 ( Table t)
    {
      this.t = t;
    }
    public void run()
    {
      t.printTable (5);
    }
}
class mythread2 extends Thread
{
    Table t;
    Mythread2(Table t)
    {
      this.t = t;
    }
    public void run()
    {
      t.printTable (100);
    }
}
class TestSyn {
   public static void main (String[] args)
   {
      Table obj = new Table();
      mythread1 t1 = new mythread1 (obj);
      mythread2 t2 = new mythread2 (obj);
```

```
        t1.start();
        t2.start();
    }
}
```

o/p:

```
5
100
10
200
15
300
20
400
25
500
```

Synchronized method :-

```
class Table {
    Synchronized public void printTable(int n)
    {
        :
}
```

o/p:

```
5
10
15
20
25
100
200
300
400
500
```

Synchronization is implemented by internal entity known as lock or monitor. Every object has a lock associated with it. If you declare any method as synchronized it is known as Synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically occurs a lock for that object & releases it. When the thread completes it task.

Synchronized Block:

Suppose you have 50 lines of code in your method but you want to ~~sing~~ a synchronize only 5 lines you can use Synchronized block.

Eg: class Table {

Synchronized (this){

Public void printTable()

:

}

Synchronize block is used to lock an object for any shared resource

Static Synchronization :-

If you make any static method as synchronized, the lock will be on the class not on the object.

Example :

```
class Table {

   Synchronized static public void printTable
                                    (int n)
   {
      for (i=1; i<=5; i++)
      {
         System.out.println (i * n);
      }
      try {
          thread.sleep (400);
      } catch (Exception e)
        {
          System.out.println(e);
        }
   }
}

class mythread1 extends Thread
{
    public void run()
    {
```

```
            t1.printTable(5);
        }
    }

class mythread2 extends Thread
{
    public void run()
    {
        t2.printTable(100);
    }
}

class TestSyn {
    public static void main(String[] args)
    {
        mythread1 t1 = new mythread1();
        mythread2 t2 = new mythread2();
        t1.start();
        t2.start();
    }
}
```