UNIT IV

10 Exception Handling and Multithreading

Page No.
Date

4.1] Errors and Exceptions :- Exceptions in java are issuees that can occur during program execution and can be handled by the program itself. They provide a way to gracefully handle unexepected situations.

eg. Let's say you have a program that needs data from a file. If the file is not found, a file NOT-FOUND Exception occurs. You can catch this exception and handle it appropriately, such as displaying an error message to the user.

* Exception Handling :- Exception Handling is a mechanism to handle runtime errors such as classNOTFOUND Exception, IOException, SQLException RemoteException, etc.

- Advantages of Exception Handling :- The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disturbs the normal ~~throw~~ flow of the application; that is why we need to handle exceptions.

Let's consider a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5; //exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed.

4.1.1] try-catch :-

- The try statement allows you to define a block of code to be tested for errors while it is being executed. It contains a set of statements where an exception can occur.
- The catch block is used to handle the uncertain condition of a try block. A try block is always followed by a catch block, which handles the exception that occurs in the associated try block.

Syntax-
```
try
{
  //block of code that may raise exception
}
catch (Exception e)
{
  //rest of the program
}
```

- Program for Try-catch :-

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

```
try
{
  int data = 50/0; // may throw exception
}
catch (Arithmetic Exception e) //handling excep.
{
  System.out.println ("Rest of the code");
}
}
```

- Output :-

```
rest of the code
```

4.1.2] throw :-

*. The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked and unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception.

Syntax- throw new exception_class ("error message");

- Program for throw :-

```
public class ThrowDemo
{
public static void main (String args[])
{
    validate (13);
    System.out.println ("Rest of the code...");
}
```

```
public static void validate (int age)
{
  if (age < 18)
  {
throw new ArithmeticException ("Person is not
 eligible to vote");
 }
 else
 {
 System.out.println ("Person is eligible to vote");
 }
}
```

- output :-

```
Person is not eligible to vote
Rest of the code...
```

4.1.3] throws :-

* The Java 'throws' keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception. So, its better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
- Throws keyword is required only to convince the compiler and usage of the throws keyword does not prevent abnormal termination of the program.

Syntax-

```
return_type method_name () throws exception-
class_name
{ //method code
}
```

Program for throws :-

```
public class ThrowsDemo
{
  static void CheckAge (int a) throws ArithmeticException
  {
    if (age <18)
  {
   throw new ArithmeticException ("Access denied- you must be at leaSt 18 years old");
  }
  else
  {
  system.out.println ("Access granted- You are old enough!");
  }

  public static void main (String args[])
  {
     CheckAge (15);
  }
}
```

Output :-

Access denied - you must be at least 18 years old.

4.1.4] finally :-

Java finally block is a block used to execute important code such as closing the connection, etc. It is always executed whether an exception is handled or not.

- Therefore it contain all the necessary statements that need to be printed regardless of exception occurs or not. The Finally block follows the try-catch blocks.

syntax-
```
Finally
{
  //Block code
}
```

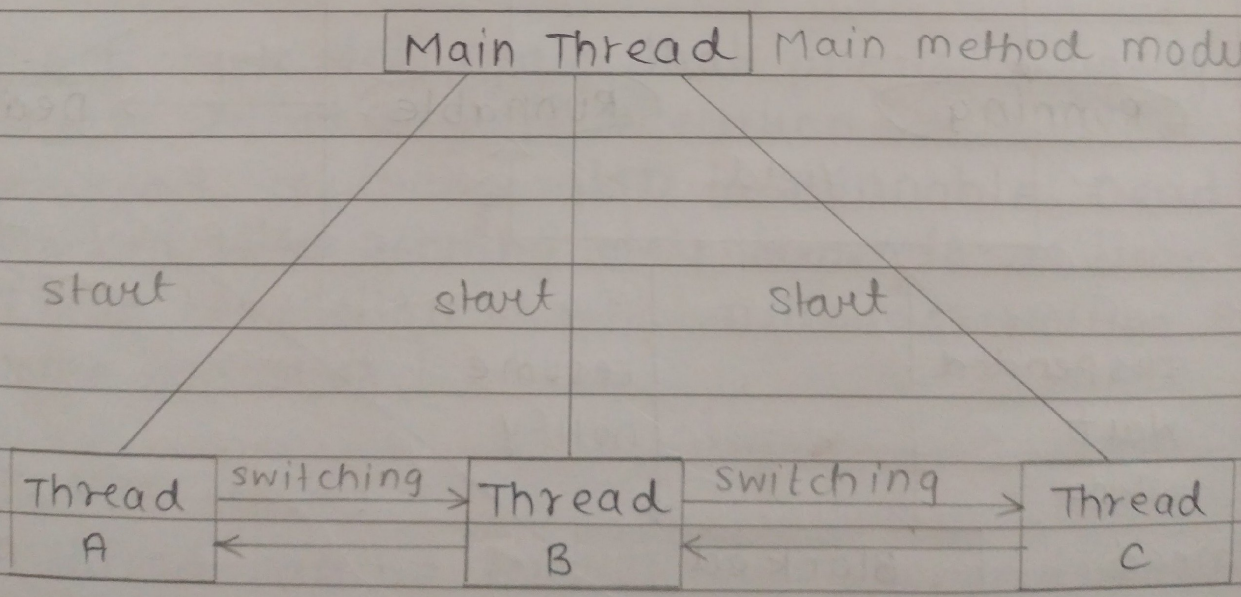
- Program for Finally:-

```
public class FinallyDemo
{
 public static void main (String args[])
 {
  try
  {
   int [] myNumbers = {1, 2, 3};
   System.out.println (myNumbers [10]);
  }
  catch (Exception e)
  {
  System.out.println ("something went wrong");
  }
  Finally {
  System.out.println ("The try-catch is Finished");
  }
 }
}
```

- Output:- something went wrong
The try-catch is Finished

4.3] Multithreading :-

- Multithreading is a Java feature that allows concurrent execution of two or more parts of a program. Each part of such program is called a thread. (Thread is a lightweight process)
- The process of executing multiple threads simultaneously is known as Multithreading.
- Multithreading is a specialized form of multitasking. Multiprocessing and multithreading both are used to achieve multitasking.
- Process is heavy weight, so we use Multithreading than Multiprocessing.
- Java Multithreading is mostly used in games, animation, etc.
- Due to Multithreading, multiple activities can proceed concurrently in the same program.



- Advantages of Java Multithreading :-

1) It doesn't block the user because, threads are independent and you can perform multiple operations at same time.

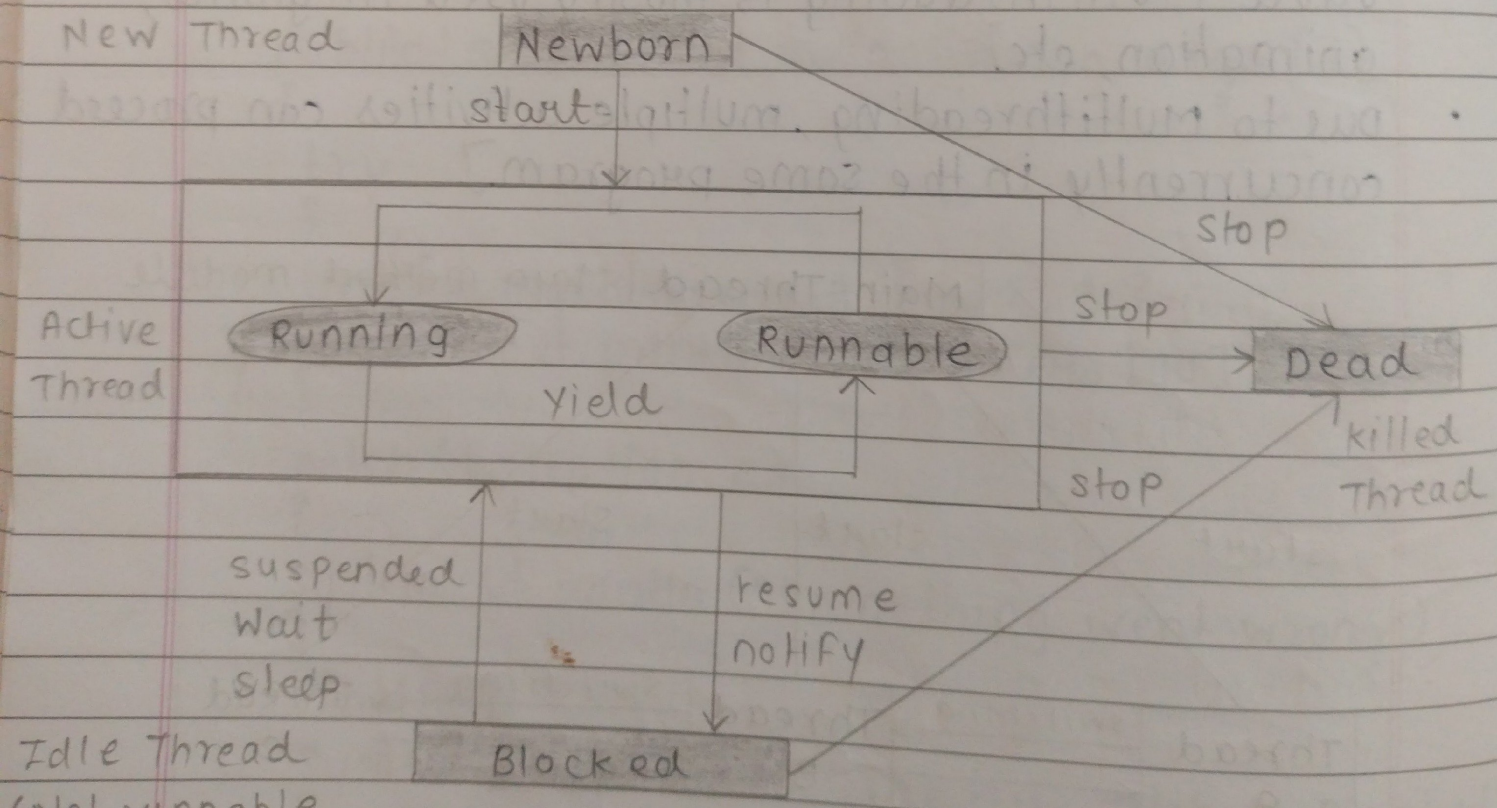
2) You can perform many operations together, so it saves time.

3) Threads are independent, so it doesn't affect other threads if exception occurs in a single thread.

4.3.1] Thread - Life Cycle :-

* A thread goes through various stages in its life cyle. The life cycle of the thread in java is controlled by JVM.

- State Transition Diagram of a Thread :-



- Following are the stages of the life cycle of a Thread :-

1] Newborn state:

- When a thread object is created, the thread is born then called as thread is in Newborn state.
- A new thread begin's its life cycle in this state, you can pass to the Running state by invoking start() method or kill it by using stop() method.

2] Runnable state:

- The thread is in runnable state after invocation of start() method, it means that the thread is ready for the execution & waiting for the availability of the processor.
- The treads which are ready for execution are managed in a queue.
- Same priority threads are processed on the basis of first-come-first serve.

3] Running state:

- The Running state means that processor has given its time to thread for their execution.
- A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

4] Blocked state:

- A thread can be temporarily suspended or blocked from entering into runnable or running state.
- Thread can be blocked by using following methods:

i) suspended()
ii) wait()
iii) sleep()

- Methods used to enter thread into Runnable state From blocked state:

i) resume() : invoked in case of suspended().
ii) notify() : invoked in case of wait().
iii) When the specified time is elapsed in the case of sleep().

5] Dead State:

- The thread will move to the dead state after completion of its execution. It means thread is in terminated or dead state when its run() method exits.
- Also Thread can be explicitly moved to the dead state by invoking stop() method.

4.3.2] Creating Thread:-

☆ • There are two different ways of creating the Thread in Java programming language:

1] By extending the Thread class:-Thread class provide constructors and methods to create and perform operations on a thread.

- Program for creating a thread by extending the thread class :-

```
import java.io.*;
class A extends Thread
{
public void run()
{
```

```
System.out.println ("Welcome");
}
{ class Creatingthread1 {
public static void main (String args[])
{
  A a1 = new A();
  a1.start(); // starting thread
```

2] By implementing Runnable Interface :- We can create a thread by implementing the Runnable interface.

Program for creating a thread by implementing runnable interface :-

```
class B implements Runnable
{
 public static void main (String args[])
 {
  B b1 = new B();
  Thread t = new Thread (b1 = "b1");
  t.start();
  System.out.println (t.getName());
 }

 public void run()
 {
 System.out.println ("Inside run method");
 }
}
```

4.3.3] Thread priority :-

- Each thread have a priority, Priorities are represented by a number between 1 and 10.
- Thread priorities are the integers which decide how one should be treated with repect to others.
- Thread priority decides when to switch from one running thread to another, process is called context switching.
- A thread can voluntarily release control and the highest priority thread that is ready to run is given the CPU.
- A thread can be voluntarily release / preempted by a higher priority thread no matter what the lower priority thread is doing. Whenever a higher priority thread wants to run it does.
- To set the priority of the thread below setPriority() method is used which is a method of the class Thread
- ThreadName.setPriority(int number);
  Where, number is an integer value which is between 1 to 10.
- getPriority() is used to retrieve the priority of the thread.
- In place of defining priority in integers, we can use below three constants:

→ MIN_PRIORITY
→ NORM_PRIORITY
→ MAX_PRIORITY

- Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 & MAX_PRIORITY is 10.

4.2] User Defined Exceptions :-

- We know the different exception classes such as Arithmetic_Exception, Null_Pointer_Exception, etc. But all these exception classes are predefined which will be executed when particular condition is occured.
- For example, when you divide a number by zero it executes the Arithmetic Exception.
- In Java we can create our own exception class and throw that exception using throw keyword.
- These exceptions are known as user-defined exceptions.
- User-defined exception must extend Exception class.
- The exception is thrown by using throw keyword.

- Example :-

```
class MyException extends Exception
{
 Myexception(String msg)
 {
    super(msg);
 }
}

class ExceptionDemo
{
 public static void main(String args[])
 {
   try
   {
    //MyException m1 = new MyException("User
     defined Exception");
```

```
        //throw m1;
        throw new MyException ("User defined
                                Exception");
    }
    catch (MyException e)
    {
        System.out.println(e);
    }
  }
}
```