Date 29/11/2016
Page 1

Unit IV

# Exception handling & Multithreaded Programming

## Introduction

In every programming language, exceptions are nothing but an errors. which can occurs during the execution of every program. An error may be produce an incorrect output or may be terminate the execution of the program. They may also cause the system to be crash. it is therefore important to detect and manage properly all the possible errors in the program so that the program will not terminate or crash during the execution.

Errors may be basically classified into two main categories

1) Compiletime errors
2) Runtime errors

1) Compiletime errors-

The errors which are generally occured or arised during the compilation time, of every programs then such errors are known as compile-time errors.

All syntax errors will be detected and displayed by the java compiler ; and therefore these errors are compile time errors. Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we can fix all the errors before we can successfully compile and run every java programs.

The most commanly arised compile-time errors are

i) Missing semicolons
ii) Missing or mismatch of the brackets in the class and methods.
iii) misspelling of identifiers and keywords
iv) Missing double quotes in the strings
v) Use of undeclared variables and so on.

2) Runtime Errors :-

As the name suggests, the errors which can occur or arised at the runtime of a every program then such errors are known as runtime errors in java.

Sometimes a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors.

Most comman runtime errors are -

i) Dividing an integer by zero.
ii) Accessing an element which is out of bounds of an array.
iii) Trying to store a value into an array of an unproper class or type.
iv) Passing a parameters which is

not in a valid range or value for a method.

v.) converting invalid string to a number.

vi) Accessing a character which is out of bounds of a string and so on.

## * Concept of Exceptions

An exception is nothing but an abnormal condition which can be cause by runtime error within our program. When the java interpretor generates an error, it creates an exceptions " if the exceptions are not handled properly, it will display an error message shown in the output of the program and it will also terminate the program.

If we want the program to continue with the execution of the remaining code, then we should try to catch the exception thrown by the error condition and then display an appropriate message for taking corrective actions. Such a task is known as exception handling."

The purpose of exception handling mechanism is to provide a means to detect and to report an exceptional circumstance so that the appropriate action can be taken. The mechanism mainly performs following main task

1) Find the problem (Hit the exception)
2) Inform that an error has occurred (Throw the exception)
3) Receive the error information (Catch the exception)
4) Take corrective actions (Handle the exception)

An exception handling basically consist of two segments, one to detect an errors and thr

ow an exception and another is to catch exceptions and take the appropriate actions.

Depending upon the above task, some comman exception which are generally occured in java are listed below

| Exception Type | Cause of Exception |
|---|---|
| 1) Arithmetic-Exception | It is caused by mathematical error such as division by zero |
| 2) ArrayIndexofBoundsException | It is caused by bad array indexes. |
| 3) ArrayStoreException | It is caused when a program tries to store the wrong type of data in an array. |
| 4) FileNotFoundException | It is caused by an attempt to access a non-existent file. |
| 5) IOException | It is caused by general input-output failures such as inability to read from a file |
| 6) NullPointerException | It is caused by referencing a null object |
| 7) Numberformat Exception | It is caused when a conversion between strings and numbers fail |
| 8) OutOfMemoryException | It is caused when there is not enough memory to allocate a new object. |
| 9) SecurityException | It is caused when an applet tries to perform an action which is not allowed by the browser's security setting |
| 10) StackOverflowException | It is caused when the system a runs out of stack space. |

What is exception? list out different type of exception in java (with & types)

| | |
|---|---|
| 11) StringIndexOutOfBoundsException | It is caused when a program attempts to access a non-existant character position in a string |

np 618
M

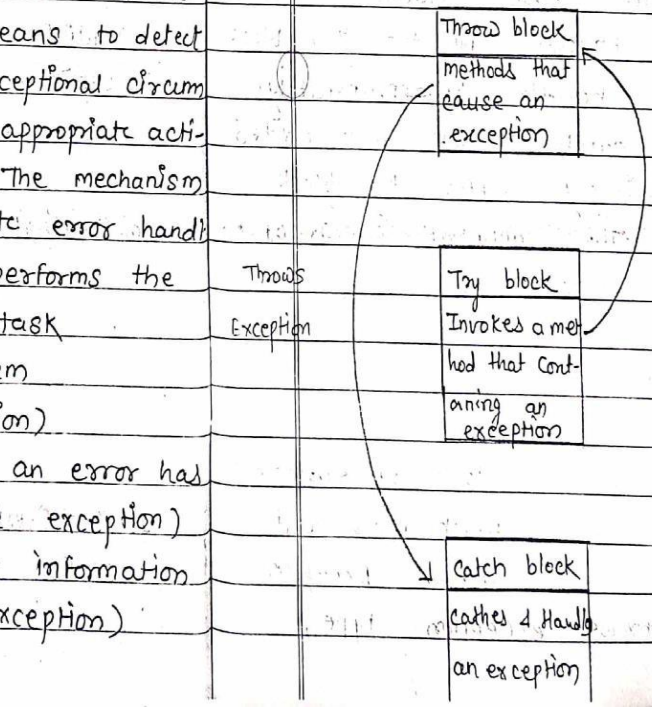
## Exception Handling Mechanism (Try - Throw - Catch Mechanism)

The main purpose behind using the exception handling (Try - Throw - Catch) mechanism is to provide a means to detect and report an exceptional circumstances so that appropriate action can be taken. The mechanism suggest a seperate error handling code that performs the following main task

1) Find the problem (Hit the exception)
2) Inform that an error has occured (Throw the exception)
3) Receive the error information (catch the exception)
4) Take corrective actions (Handle the exception)

An exception handling basically consist of two segments one to detect an error and throw an exception and another is to catch exceptions and take appropriate actions.

following diagram shows the exception handling or Try - Throw - Catch mechanism



The exception handling Try-Throw - Catch mechanism is shown in above diagram the basic concept of exception handling are throwing an exception and catching it.

Java uses a try block of code that is likely to cause an error condition and throws an exception. A throw block basically contains different methods that causes an exception. A catch block defined by the catch keyword which catches the exception thrown by the Try block and handle it appropriately. The catch block will added immediately after try block.

The general syntax of try Throw- Catch is

```
try
{
    --- // block of statement
    --- which check and
    ---- Throws Exception
throws exception type;
---
}
catch ( Type arg ) // catch execution
{
    ---- // block of statement
}                    that handles exception
```

The try block can have one or more statement that could generate an exception. If any one statement generate an exception the remaining statement in the block are skipped and the execution jumps to the throw block and then after to the catch block which is placed after next to the try block.

The catch block contains one or more statements which are necessary to process the exception. Remember that every try block should always followed by atleast one catch statement otherwise compilation error will occured.

Note that the catch

statement works like a method defination. The catch statement is pas a single parameter, which is the reference to the exception objects thrown by the try block. If the catch parameter matches with the type of exception object then the exception is caught and the statement in catch block will be executed. If the exception is not caught, the default exception handler will cause the execution to terminate.

Date 6/12/16
Page 7

* Multiple Catch blocks

According to the exception handling mechanism, it is possible to implement one or more than one catch statements can be implement inside the catch block. The general representation for multiple catch blocks are:-

```
---
try
{
    statement; // generates Exception
}

catch (Exception-Type1 e)
{
    statement; // Processes Exception
                 Type1
}

catch (Exception-Type2 e)
{
    statement; // Processes Exception
                 Type2
}
----
```

when an exception in a try block is generated, the java treats the multiple catch statements like cases in switch statement The 1st statement whose parameters matches with the exception object will be executed and the remaining statements will skipped.

Note that java does not require any processing of the exception at all. We can simply declare a catch statement with an empty block to avoid program ambigutation (complexness).

The catch statement simply ends with semicolon, which does nothing this statement will just catch an exception and ignore it.

Imp

* Use of finally Statement

Java supports a statement known as finally statement which can be used to handle an exception that is not caught by any of the previous catch statements. finally block can be used to handle any exception generated within try block. It may be added immediately after the try block or after the last catch block shown below.

```
try                    try
{                      {
---                    --
---                    }
}                      catch(--)
Finally                {---
{                      }
--                     catch(---)
--                     {
}                      ----
                       }
                       Finally(---)
                       {
                       ---
                       }
```

when finally block is defined this is compulsarily executes regardless whether an exception is is thrown. As a result we can use it to perform certain operation such as closing a files and releasing system resources.

Generally the statements which are occured inside a finally block is represented below

```
finally
{
    int a = 10, b = 5;
    int y = b / a;
    System.out.println("y = "+y);
}
```

* Catch all statement (Exception)

In some situation, we may not able to handle all possible exceptions and therefore may not able to design independant catch statements to catch them.

This can be achieved by using catch all exception. It may be as good idea to use the catch block as a default statement along with other catch handlers so that it can catch all those exceptions which are not handled properly

* Uncaught exceptions

As discussed earlier, for handling an exception we can use a catch block. A catch block generally looks like a function def" it catches an exception whose type matches with the type of the catch argument when it is caught the code in the catch block is executed.

But in some situations, the compiler is unable to handle some exception with catch statements then such exceptions are known as uncaught exception.

```
/**

*/
class Error
{
  public static void main(String args[])
  {
    int a=10, b=5, c=5;
    int x, y;
    try
    {
      x = a / (b-c); // exception here
    }
    catch (ArithmeticException e)
    {
      System.out.println("Division by zero");
    }

    y=a / (b+c);

    System.out.println("y = "+y);
  }
}
```

Above program will generate o/p as-

```
Division by zero
y=1
```

Date 12/12/16
Page

# * Multithreaded programming

## * Introduction to thread

In a every operating system, we can execute several programs simultaneously. This ability of a every system is known as multitasking. In a system's terminology, it is known as multithreading.

The concept of multithreading is totally depends upon the concept of thread. A thread is nothing but one process which can run in every operating system. Secondly, we can also say that a thread is similar to a program which has a single flow of control. It has a beginning, a body an end and executes every command simultaneously. All main programs generally called as a single threaded program. Every program will have atleast one thread.

A unique property of a java

Date
Page 11

is, it supports the concept of multithreading. Java enables us to use multiple flow of controls in a developing a programms. Each flow of contr may be seperate with tiny modules known as thread. that runs parallel to others Internally in every system. A program that contains multiple flow of controls are known as multithreaded programs.

A multithreading is a powerfull programming tool that makes java different from its flow of programming. Multithreading is useful in number of ways. It enables the programmers to do multiple things in a number of ways. The, can divide a long program into a threads and execute them in a parallel.

for example, we can send task of printing and it contineou to perform the same task in the background. This approach would considerably improves the speed of our program. Threds are extensively

used in java enabled web browsers to the local computers for displaying the output and so on. Any application requires two or more things to be done at the same time for the use of the thread.

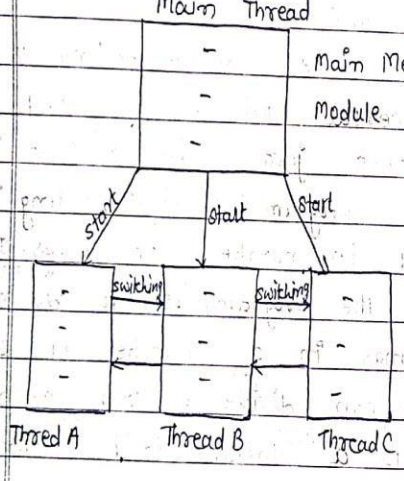



fig multithreaded Program

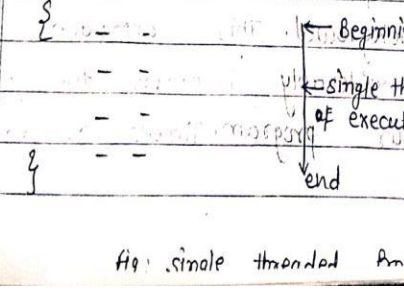



fig: single threaded Program

## * Creating a threads

In java, creating a threads are very simple. Threads are created and implemented in the form of objects that contains a method called run(). The run() method is the heart and soul of any thread. It makes a entire body of an thread and is the only method in which the thread's behaviour can be implemented.

The general syntax for run method is

```
public void run()
{
    (Statements for
    implementing thread)
}
```

The run() should be invoked by an object of the concerned thread. This can be achieved by creating the thread

and initiating it with the help of another thread method called start().

The thread can be created in java in two ways

1) By creating a thread class

2) By converting a class to a thread

1) By creating a thread class:

Define a class that extends thread class and override its run() method with the code required by a thread class.

2) By converting a class to a thread:

Define a class that implements runnable interface, the runnable interface has only one run() method which is to be defined in the method with the code to be executed by the thread.

import java.lang.thread

This approaches are to be used depends upon what kind of class require for creating it. If it requires to extend another class then we have to implement the runnable interfaces because java class does not have two super classes.

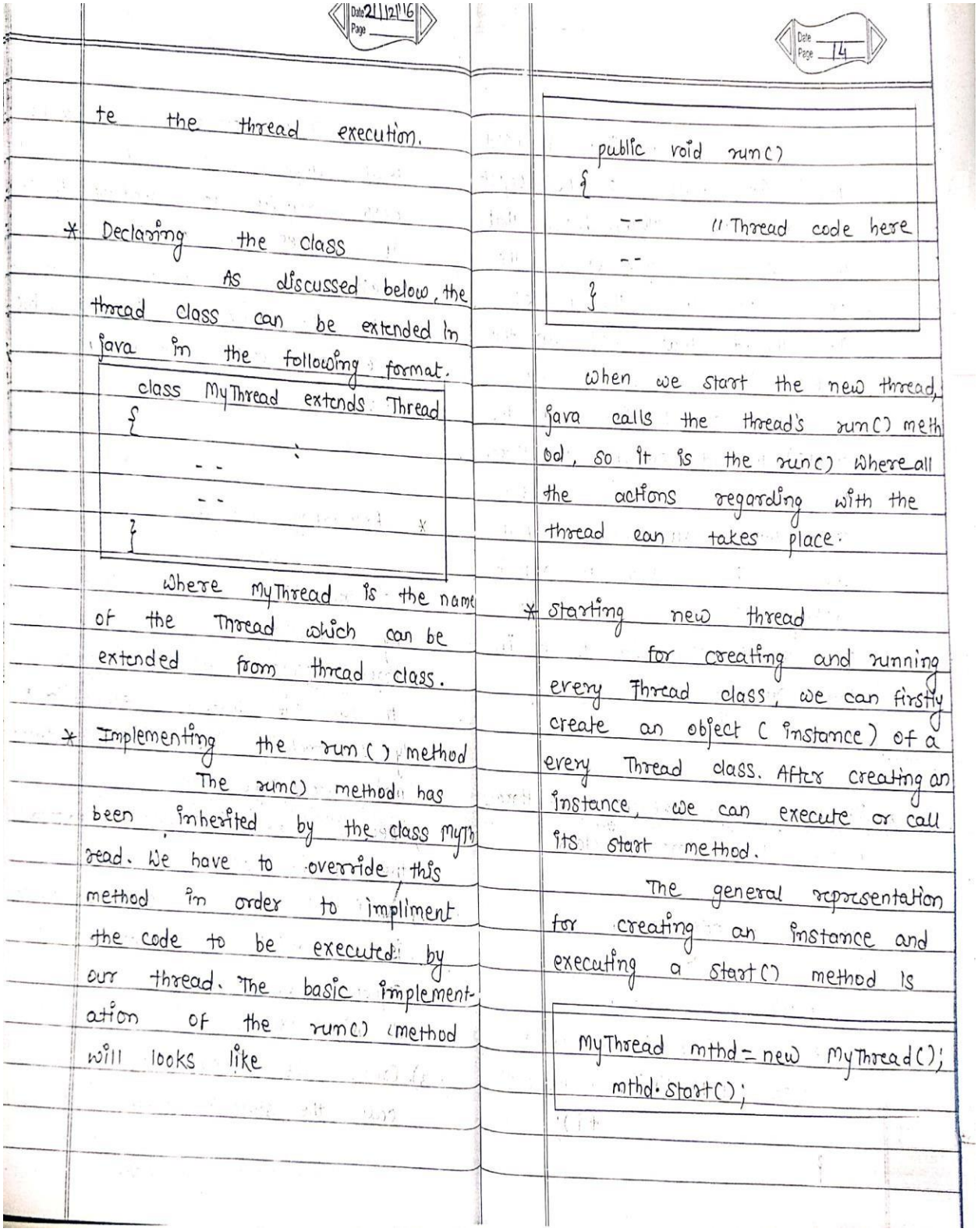
* Extending Threads

We can make our class as runnable for a thread by extending the class java.lang.thread. This gives us access to all the thread methods directly. It includes the following three main steps. They are

1) Declare the class as extending the thread class

2) Impliment the run() method which is responsible for executing the sequence of code which the thread will execute.

3) Create the thread object and call the start() method to initi



te the thread execution.

* Declaring the class

As discussed below, the thread class can be extended in java in the following format.

```
class MyThread extends Thread
{
  :
  --
  --
}
```

Where myThread is the name of the Thread which can be extended from thread class.

* Implementing the run() method

The run() method has been inherited by the class myThread. We have to override this method in order to impliment the code to be executed by our thread. The basic implementation of the run() method will looks like

```
public void run()
{
  --   // Thread code here
  --
}
```

when we start the new thread, java calls the thread's run() method, so it is the run() where all the actions regarding with the thread can takes place.

* Starting new thread

for creating and running every Thread class, we can firstly create an object ( instance ) of a every Thread class. After creating an instance, we can execute or call its start method.

The general representation for creating an instance and executing a start() method is

```
MyThread mthd = new MyThread();
mthd.start();
```

The first line is used for instantiates a new object of a class MyThread. Note that this statement only creates the object of the Thread class. The Thread that will run this object is not still running.

The second line calls the start() method causing the Thread to move into the runnable state. Then java runtime will schedule the Thread to run by invoking its run() method. Now the Thread is said to be in the running state.

* Program for creating the threads using the Thread class

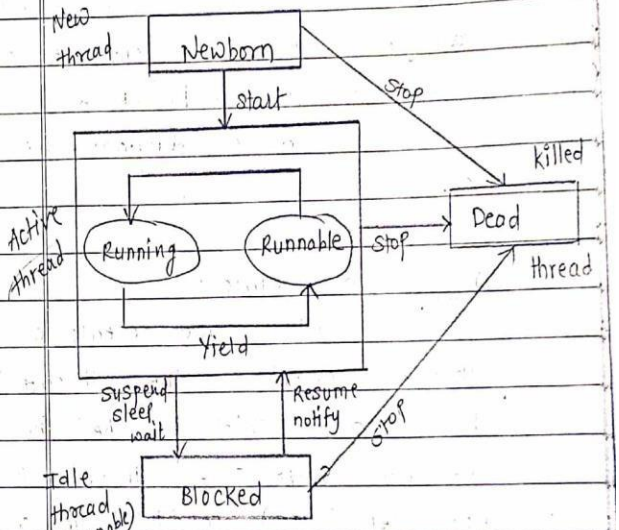
```
class A extends Thread
{
    public void run()
    { for(int i=1; i<=5; i++)
      System.out.println("From Thread A:i="
                         +i);
    }
    System.out.println("Exit from A");
}

class B extends Thread
{
    public void run()
    {
      for(int j=1; j<=5; j++)
      {
        System.out.println("from Thread B=j="
                           +j);
      }
      System.out.println("Exit from B");
    }
}

class C extends Thread
{
    public void run()
    {
      for(int k=1; k<=5; k++)
      {
        System.out.println("from Thread C: k="
                           +k);
      }
      System.out.println("Exit from C");
    }
}
```

```
class ThreadTest
{
  public static void main(String args[])
  {
    new A().start();
    new B().start();
    new C().start();
  }
}
```

The above prg will generate an output as:

```
from thread A: i=1
from thread A: i=2
from thread B: j=1
from thread B: j=2
from thread A: i=3
from thread B: j=3
from thread C: k=1
from thread C: k=2
from thread C: k=3
from thread C: k=4
from thread C: k=5
Exit from C
from thread B: j=4
from thread B: j=5
Exit from B
from thread A: i=4
from thread A: i=5
```

* Life Cycle of Thread :-




Thread is a subprogram which executes by using computers CPU time.

The typical lifecycle of a thread includes 5 main state they are

1) Newborn State
2) Runnable state
3) Running State
4) Blocked State
5) Dead State

Date 26/12/16
Page

A thread is always works in one of this 5 states. It can move from one state to another via a verity of ways as shown in above diagram. The working

1) Newborn state :-

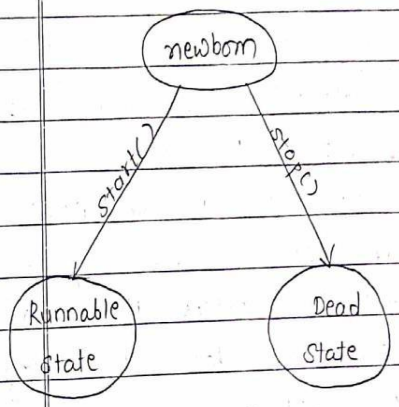
When we create a thread object, the thread is born and said to be in newborn state. The thread is not yet (still) scheduled for running. At this state, we can do only one of the following thing ie

ⓐ schedule it for running by using start() method

ⓑ kill it by using stop() method

If scheduled, it moves to the runnable state. If we attempt to use any other method, an exception will be thrown

Date
Page 17

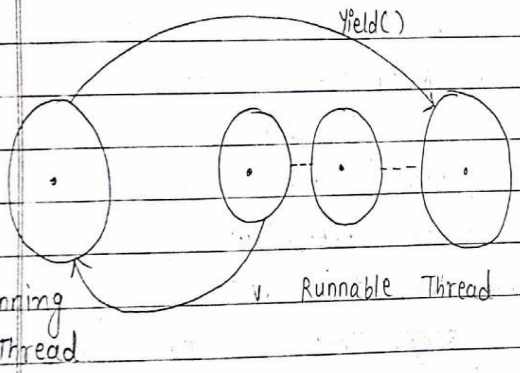



2) Runnable state :-

The runnable state means that, the thread is ready for execution and is waiting for the availability of the processor. That is the thread has joined the queue of the thread which are waiting for the execution. If all the threads have equal priority, then they are given the timeslots for execution in a round robin fashion ie first-come, first-serve manner. The thread which control the joins of the queue at the end and again waits for its run, this

process of assigning time to the threads is known as time-slicing.

However if we want a thread of a equal priority to execute before its turn comes this can be done by using the yield() method.

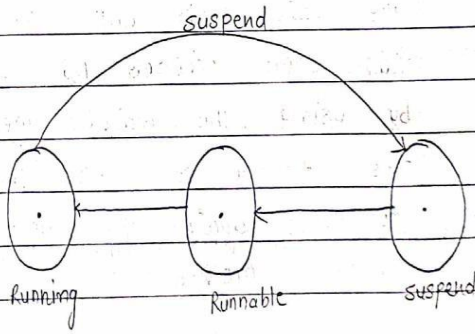



3) Running state

Running means that the processor has given its time to the thread for its execution. The thread runs untill it control on its own or it is preexecuted by the higher priority thread.

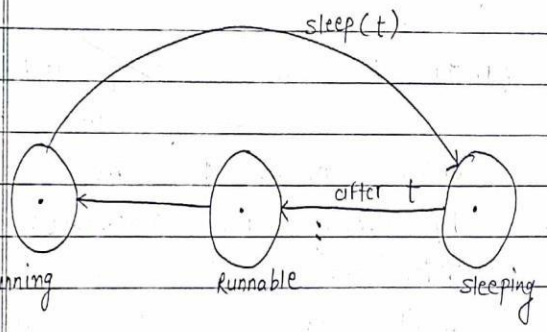
A running thread may be control in a one of the following situations -

(1) It has been suspended by using suspend() method. A suspended thread can be reviewed by using the resume() method. This approch is usefull when we want to suspend a thread for sometime due to certain reasons

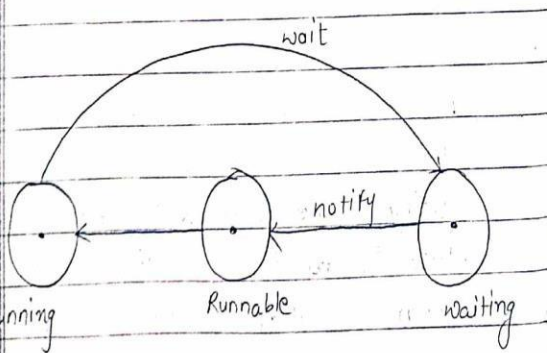



(2) We can put a thread to sleep for a specified time period using the method sleep(time) where time is in milliseconds. This means that the thread is out

of the queue during this time period the thread is reenters into the runnable state as soon as this time period has been elapsed.




(3) Every thread has been occured into the wait state untill some event has been occured this is done by using the wait() method. The thread can be scheduled to run again by using the notify() method.




4) Blocked state

A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements. A block thread is considered as not runnable but not dead and therefore after some time it run again.

5) Dead State

The fifth and the last state of thread life-cycle is an dead state. A running thread ends its working when it has completed executing its run() method. When we can kill the thread by sending the stop message to it at any state by causing a premature death to it. A thread can be killed as soon as it is born, or while it is running or even when it is in

blocked condition.

* Stopping and Blocking (Suspending) a Thread

* Stopping a Thread

Whenever we want to stop a thread from running further, we may call its stop() method. This method causes the thread to move to the thread to the dead state. A thread will also move to the dead state automatically when it reaches at the end of its method. The stop() method may also be used when the death of a thread is occured.

* Blocking (Suspending) a Thread

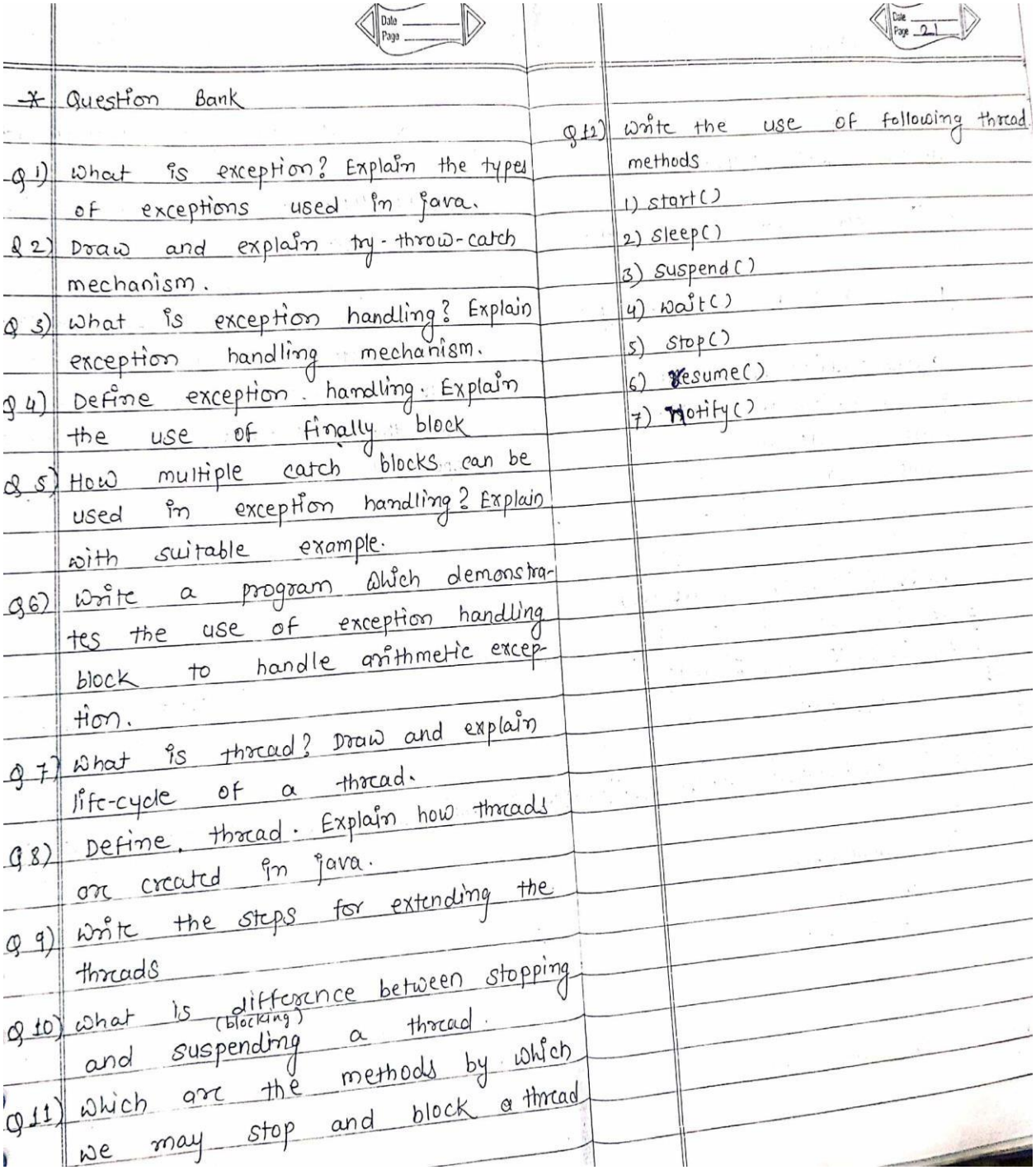
A thread can also temporarily suspended or blocked from entering into the runnable and subsequently running state by using either of the following thread methods

sleep() // blocked for specific time

Suspend() // blocked untill further order.

wait() // blocked untill certain condition occurs.

These methods causes the Thread move into the block or not runnable state. The Thread will return to the runnable state after some specified time.



## Question Bank

Q1) What is exception? Explain the types of exceptions used in java.

Q2) Draw and explain try-throw-catch mechanism.

Q3) What is exception handling? Explain exception handling mechanism.

Q4) Define exception handling. Explain the use of finally block

Q5) How multiple catch blocks can be used in exception handling? Explain with suitable example.

Q6) Write a program which demonstrates the use of exception handling block to handle arithmetic exception.

Q7) What is thread? Draw and explain life-cycle of a thread.

Q8) Define thread. Explain how threads are created in java.

Q9) Write the steps for extending the threads

Q10) What is difference between stopping (blocking) and suspending a thread.

Q11) Which are the methods by which we may stop and block a thread

Q12) Write the use of following thread methods

1) start()
2) sleep()
3) suspend()
4) wait()
5) stop()
6) resume()
7) notify()

*****