DOCUMENTATION

ON

THREAD

&

EXCEPTIONS

- **THREAD**

  A thread is a lightweight sub process. It is a separate path of execution because each thread runs in a separate stack from the memory area of process.

- **THREAD CREATION**

  We can create thread in two ways

  1. By extending Thread class: A class that extends Thread class and override its run() method with the code required by the thread.
  2. By implementing Runnable interface: A class that implements Runnable interface. The Runnable interface has only one method, run( ), that is to be defined in the method with the code to be executed by the thread.

  We can make our class runnable as thread by extending the class java.lang .Thread. This gives us access to all the thread methods directly.

  We should call start method instead of directly calling run method

  We cannot directly call start method inside runnable thread - because we don't have start method inside runnable interface

- **Declaring the Class**

  Class Bhanu extends Thread {

  }

- **Implementing the run( ) Method**

  The run( ) method has been inherited by the class Bhanu. We have to override this method in order to implement the code to be executed by our thread.

  public void run( ) {

  }

- **Starting New Thread**

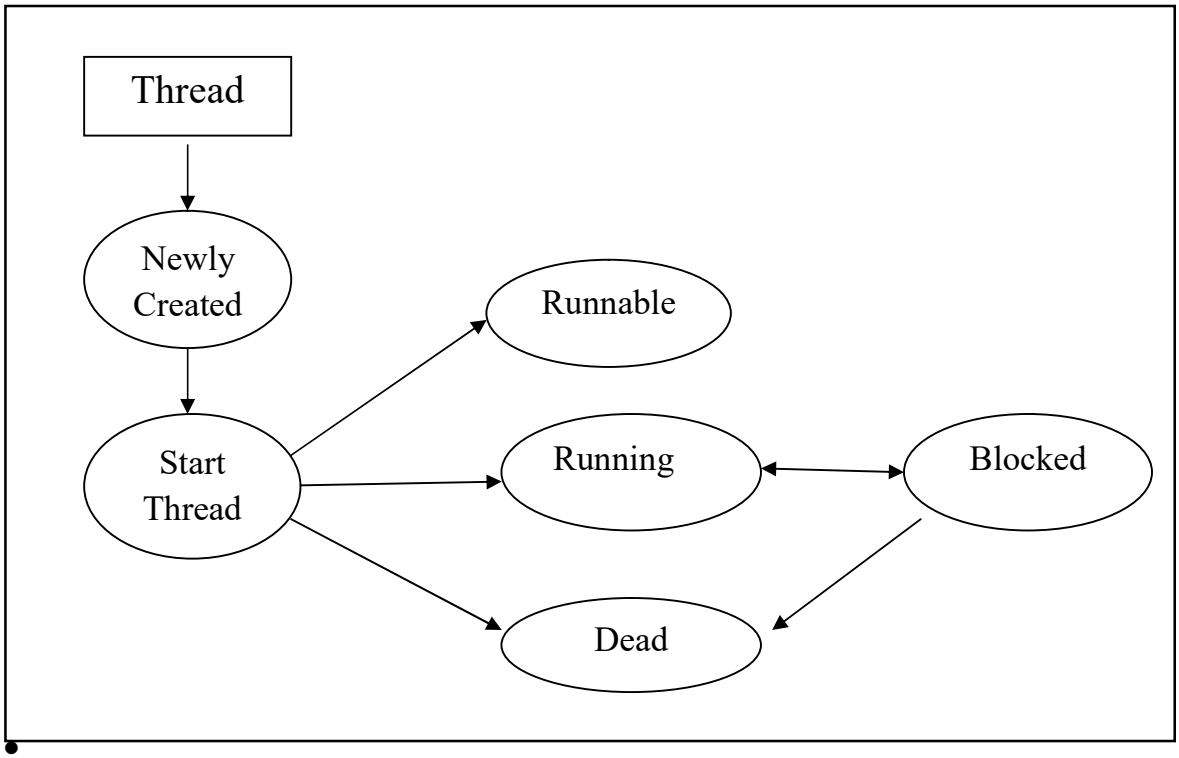
  Bhanu bThread = new Bhanu( );

  bThread.start( );

  The first line instantiates a new object of class Bhanu. That this statement just creates the object. The thread that will run this object is not yet running. The thread is in a newborn state.

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

- **Thread Life Cycle**



- **New state**: After the creations of Thread is in this state but before the start( ) method invocation. At this point, the thread is considered as not alive.
- **Runnable (Ready – to – run) state:** A thread starts its life from Runnable state. A thread first enters runnable state after the invoking of start( ) method but a thread can return this state after either running, waiting, sleeping or coming back from the blocked state too. On this state a thread is waiting for a turn on the processor.

- **Running state:** A thread is in running state which means the thread is currently executing. There are several ways to enter in Runnable state but there is only one way to enter in running state. The scheduler selects a thread from runnable pool.
- **Dead state:** Athread can be considered as dead, when its run ( ) method completes. If any thread comes in this state, which means it cannot ever run again.
- **Blocked:** A thread can enter in this state, because of waiting for the resources that are held by another thread.

➢ **Methods for preventing Thread Execution:**

We can prevent the Thread Execution by using three (3) methods of threads class

- yield
- join
- sleep

yield ( ) : This is to pause current executing thread, for giving chance to remaining waiting threads of some priority.

If there is no thread waiting or all the threads waiting have low priority, then the same thread is once again given a chance for the execution.

The once again chance given to the thread is decided by thread scheduler and we cannot expect anything exactly.

Public static native void yield( );

join ( ) : If a thread wants to wait until completion of other thread, then we should go for join( ) method.

sleep ( ) : If a thread don't want to perform any operation for the particular amount of time when a pausing is required, then we should go for sleep( ) method.

➢ **Exception :**

Exceptions are run – time errors caused due to logical mistakes occurred during program execution because of its wrong input.

Creating an array with negative number as size of the array can't be negative. Hence, the exception "java.lang.NegativeArraySizeException" is thrown represents the logical mistake in giving the size of array.

In java, exception is an object. In other words it is the instance of the subclasses of java.lang.Throwable class.

“java.lang” package to represent different logical mistakes. All these classes are subclasses of java.lang. Throwable class.

1. For handling wrong operations on arrays
    - NegativeArraySizeException
    - ArrayIndexOutOfBoundException
    - ArrayStoreException
2. If we divide integerwith zero (0)
    - ArithmeticException
3. If we try to covert alpha numeric String data to number
    - NumberFormatException

There are two (2) types of exceptions

- Checked exceptions
- Unchecked exceptions

Checked Exceptions: The Checked Exceptions are recognized by the compiler. In the Exception Hierarchy, except the Runtime Exception and its subclasses errors all the remaining Exceptions are called Checked Exceptions

Ex: FileNotFoundException, InterruptedException,IOException

UnChecked Exception: The Unchecked Exceptions are not recognized by the compiler. They can occur suddenly in the Exception Heiarachy the Runtime Exception and its subclasses are called as UnCheckedExceptions.

Ex: NullPointerException, ArithmeticException, ClassCastException, ArrayIndexOutOfBoundsExceptions.