



Object Oriented Programming

ICT2122

Introduction to JAVA - Threads

P.H.P. Nuwan Laksiri
Department of ICT
Faculty of Technology
University of Ruhuna.

Lesson 08

Recap – Database Connectivity

- What is an API
- JDBC Introduction
- Why JDBC
- JDBC Components
- JDBC Architecture
- JDBC Drivers
- Basic steps to use a database in Java
- Handling Exceptions

Outline

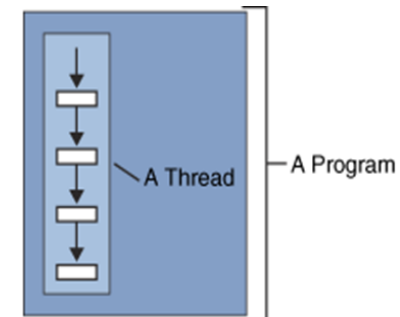
- Introduction to Java threads
- Multithreading in Java
- Threads vs Processes
- Life cycle of a Thread
- Creating Threads
 - By extending
 - By implementing

?

- ICT2122 - Mid Term
02 Quizzes
- ICT2132 - Mid Term
Mini Project Evaluation


Introduction – Java Thread

- A thread is similar to the sequential programs : a single thread also has a beginning, an end, a sequence, and at any given time during the runtime of the thread there is a single point of execution.
- A thread itself is not a program. It cannot run on its own, but runs within a program.
- Definition:
 - A thread is a single sequential flow of control within a program.



Introduction – Java Thread

- Facility to allow multiple activities within a single process
- Referred as lightweight process
- A thread is a series of executed statements
- Each thread has its own program counter, stack and local variables
- A thread is a nested sequence of method calls
- Its shares memory, files and per-process state



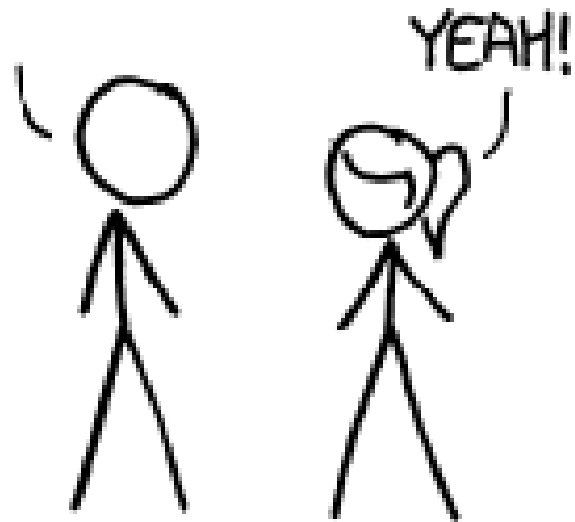
What's the need of a thread or why we use Threads?

- To perform asynchronous or background processing
- Increases the responsiveness of GUI applications
- Take advantage of multiprocessor systems
- Simplify program logic when there are multiple independent entities

SITUATION:

There is a
problem.

Let's use
multithreading.



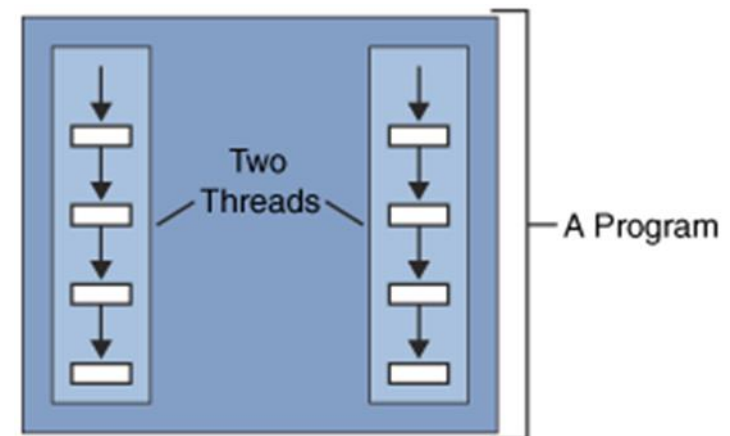
SOON:

SITUATION:

the
are
97
prms.oble

Multithreading in Java

- Multithreading is the concept of breaking a program into two or more parts called threads and run in parallel.
- Multi threading enables you to write in a way where multiple activities can proceed concurrently in the same program
- This makes programs more responsive and effective.
- Multithreading increases the performance.



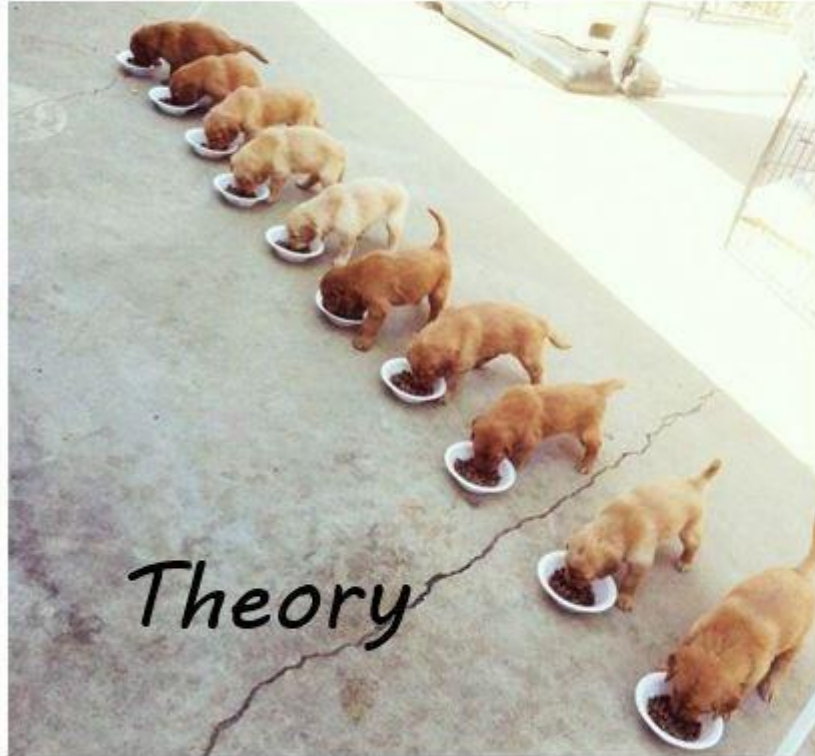
Without multithreading in Java

- When a particular resource is available, the process dispatches the request to the appropriate event handler.
- Until this event handler returns, nothing else can happen.
- Example:
 - In MS Excel, while executing a complex formula scrolling operation is not allowed
 - Most of the time tasks of the same program have to wait for other resources.

Why Multithreading in Java

- If calculations and scrolling are executed in two independent threads, they can run independently.
- One does not need to wait for other thread to complete
- One can automatically switch from one task (ready to wait) to another (ready to run)
- More work can be done in the same time
- With multi processors, threads can run simultaneously.
- With a single processor, CPU time is shared among threads

Multithreaded programming



Multitasking Vs. Multithreading

- These two have lot of differences.
- Multitasking is running several different programs parallel
 - Heavy weight processes
 - Each runs on different address space
- Multithreading is running different components of a same program parallel
 - Light-weight processes
 - Runs in the same address space

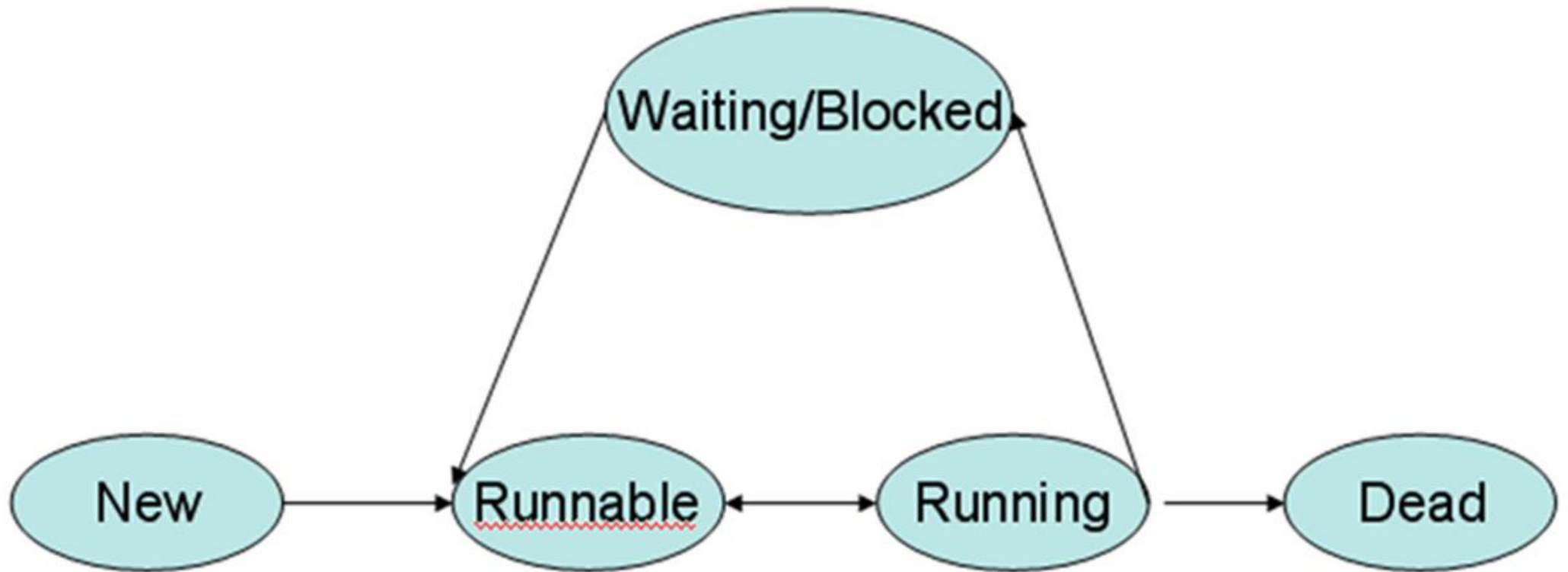
Thread vs Process

- A program in execution is often referred as process.
 - A thread is a subset(part) of the process.
- A process consists of multiple threads.
 - A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.
- A process is sometime referred as task.
 - A thread is often referred as lightweight process.
- A process has its own address space.
 - A thread uses the process's address space and share it with the other threads of that process.

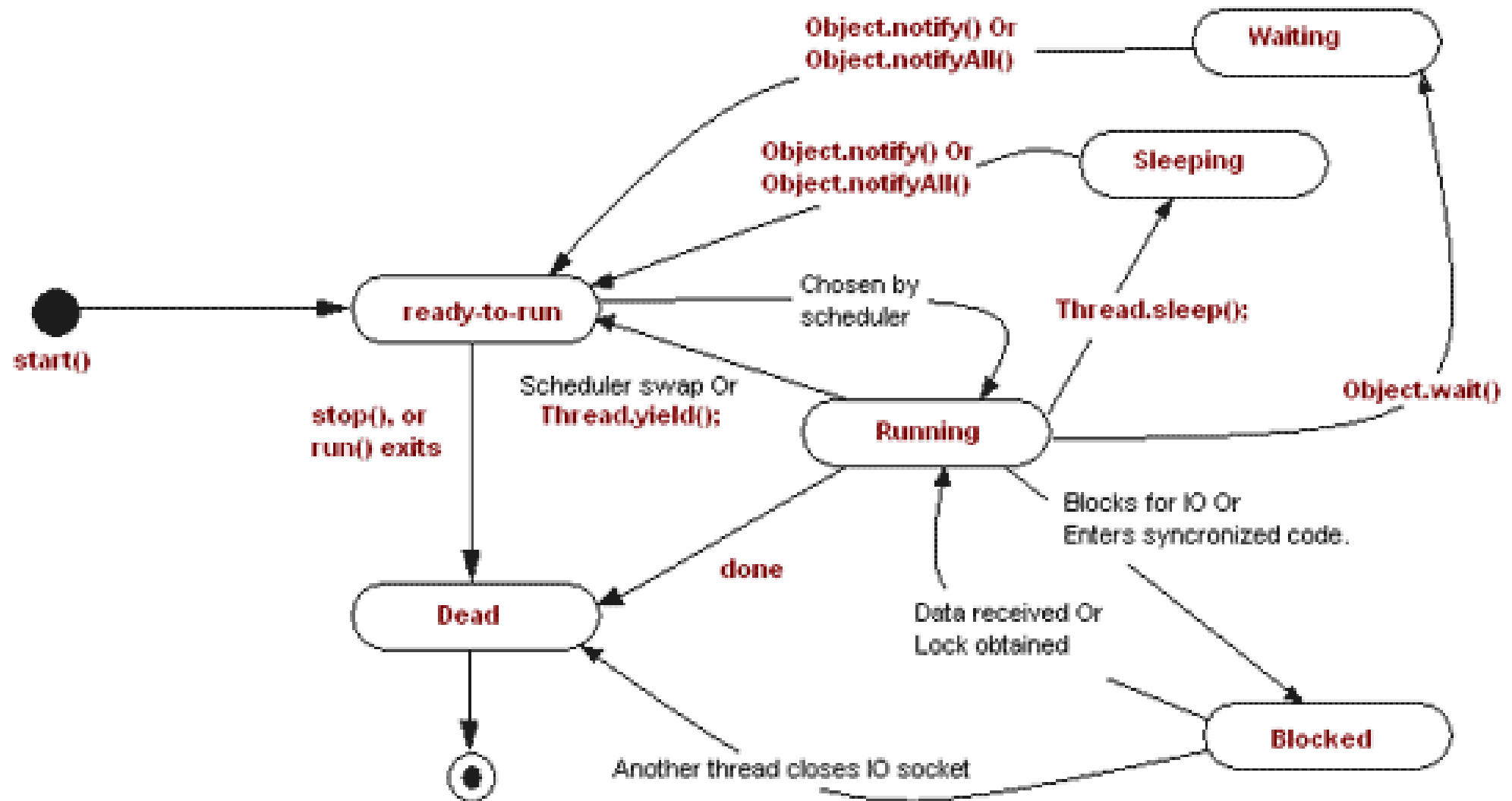
Thread vs Process

- A thread can communicate with other thread (of the same process) directly by using methods like `wait()`, `notify()`, `notifyAll()`.
 - A process can communicate with other process by using inter-process communication.
- New threads are easily created.
 - Creation of new processes require duplication of the parent process.
- Threads have control over the other threads of the same process.
 - A process does not have control over the sibling process, it has control over its child processes only.

Life cycle of a thread (Thread States)



Life cycle of a thread (Thread States)



Life cycle of a thread (Thread States)

- New
 - The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- Runnable
 - The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- Running
 - The thread is in running state if the thread scheduler has selected it.

Life cycle of a thread

(Thread States)

- Waiting/ Blocked
 - This is the state when the thread is still alive but is currently not eligible to run.
- Dead
 - A thread is in terminated or dead state when its run() method exits.

Thread creation in Java

- Threads can be created in two ways,
 - By extending Thread class
 - Create a new class that extends Thread class
 - By implementing Runnable interface

Important methods of Java thread class

Method name	Description
<code>public void run()</code>	Used to perform action for a thread.
<code>public void start()</code>	Starts the execution of the thread. JVM calls the <code>run()</code> method on the thread.
<code>public void sleep(long milisec)</code>	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
<code>public Thread currentThread()</code>	Returns the reference of currently executing thread
<code>public void join()</code>	waits for a thread to die.
<code>public String getName()</code>	Returns the name of the thread.
<code>public int getPriority()</code>	Returns the priority of the thread.

Important methods of Java thread class

Method name	Description
<code>public int setPriority(int priority)</code>	Changes the priority of the thread.
<code>public boolean isAlive()</code>	Tests if the thread is alive.
<code>public void yield()</code>	Causes the currently executing thread object to temporarily pause and allow other threads to execute.
<code>public void suspend()</code>	Used to suspend the thread(deprecated).
<code>public void resume()</code>	Used to resume the suspended thread(deprecated).
<code>public void stop()</code>	Used to stop the thread(deprecated)
<code>public void interrupt()</code>	Interrupts the thread

Thread creation in Java

Thread implementation in java can be achieved in two ways:

- Extending the `java.lang.Thread` class
- Implementing the `java.lang.Runnable` Interface
 - Note: The `Thread` and `Runnable` are available in the `java.lang.*` package

1st Method : Extend Thread class

- One way to create a thread is to create a new class that extends Thread class.
- This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

- A class extending the Thread class should overrides the run() method from the Thread class to define the code executed by the thread.
- This method provides entry point for the thread and you will put your complete business logic inside this method.

1st Method : Extend Thread class

Step 2

- Once Thread object is created, you can start it by calling start () method, which executes a call to run() method to make the thread eligible for running.

Example

```
package mythread;

/**
 *
 * @author nuwan
 */
public class MyThread extends Thread{

    @Override
    public void run() {
        System.out.println("My thread is running...!!!");
    }

}
```

Example

```
6 package mythread;
7
8 /**
9  *
10  * @author nuwan
11  */
12 public class MyThreadTest {
13
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         // TODO code application logic here
19         MyThread myThread = new MyThread();
20         myThread.start();
21     }
22
23 }
```

mythread.MyThreadTest > main >

Output - MyThread (run) x

```
> run:
My thread is running...!!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

2nd Method: Implementing Runnable Interface

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.

Step I

- Implement a run() method provided by Runnable interface. This provides entry point for the thread and can put the complete business logic inside this method.

2nd Method: Implementing Runnable Interface

Step 2

- Instantiate a Thread object using either one of the following constructors.
 - Thread(Runnable threadObj, String threadName)
 - Thread(Runnable threadObj)
 - threadObj =instance of a class that implements the Runnable interface
 - threadName =name given to the new thread

Step 3

- Once Thread object is created, start it by calling start() method, which executes a call to run() method.

Example

```
package mythread;
```

```
/**
```

```
 *
```

```
 * @author nuwan
```

```
 */
```

```
public class MyThreadInter implements Runnable{
```

```
    @Override
```

```
    public void run() {
```

```
        System.out.println("My thread is running...!!!");
```

```
    }
```

```
}
```

Example

```
6 package mythread;
7
8 /**
9  *
10  * @author nuwan
11  */
12 public class MyThreadTest {
13
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         // TODO code application logic here
19         MyThreadInter myThreadInter = new MyThreadInter();
20         Thread newThread = new Thread(myThreadInter);
21         newThread.start();
22     }
23
24 }
25
```

mythread.MyThreadTest > main >

Output - MyThread (run) x

run:
My thread is running...!!!
BUILD SUCCESSFUL (total time: 0 seconds)

Extends Thread class vs Implements Runnable Interface?

- Extending the Thread class will make your class unable to extend other classes, because of the single inheritance feature in JAVA.
 - However, this will give you a simpler code structure.
 - If you implement Runnable, you can gain better object-oriented design and consistency and also avoid the single inheritance problems.
- If you just want to achieve basic functionality of a thread you can simply implement Runnable interface and override run() method.
 - But if you want to do something serious with thread object as it has other methods like suspend(), resume(), ..etc which are not available in Runnable interface then you may prefer to extend the Thread class.

The Difference

```
12 public class MyThreadTest {
13
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         //By extending the Thread Class
19         MyThread myThread = new MyThread();
20         myThread.start();
21
22         //By Implementing the Runnable Interface
23         MyThreadInter myThreadInter = new MyThreadInter();
24         Thread newThread = new Thread(myThreadInter);
25         newThread.start();
26     }
27
28 }
29
```

mythread.MyThreadTest > main >

Output - MyThread (run) x

```
run:
My thread is running, by extending...!!!
My thread is running, by Implementing...!!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Constructors in Thread class

Constructor	Meaning
<u>Thread()</u>	Constructs a new Thread
<u>Thread(String)</u>	Constructs a new Thread with the specified name.
<u>Thread(Runnable)</u>	Constructs a new Thread which applies the run() method of the specified target.
<u>Thread(Runnable, String)</u>	Constructs a new Thread with the specified name and applies the run() method of the specified target.

Starting a thread

- Steps in the starting process
 - create an instance of the thread object
 - Call start() method which spawn a new thread
 - Start() code registers the Thread with scheduler and the scheduler calls the run() method
- Difference between run() and start()
 - Thread.run() does not spawn a new thread whereas Thread.start() does.
 - Thread.run() actually runs on the same thread as that of the caller
 - Thread.start() creates a new thread on which the task is run

Hands-on

- Let's print numbers
- Let's add and multiply

Summary

- Introduction to Java threads
- Multithreading in Java
- Threads vs Processes
- Life cycle of a Thread
- Creating Threads
- By extending
- By implementing

References

- <https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html>
- How To Program (Early Objects)
 - By H .Deitel and P. Deitel
- Head First Java
 - By Kathy Sierra and Bert Bates

Questions ???





Thank You