UNIT IV - EXCEPTION AND MULTITHREADING

Exception handling –Exception types – try catch and finally block, throws –Runtime exception – Introduction to Multithreading - Thread Creation – Thread control and priorities - Thread synchronization, Inter-thread communication.

Introduction to Multithreading

Multithreading is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. **Multiprocessing and multithreading, both are used to achieve multitasking.**

- use multithreading than multiprocessing because **threads use a shared memory area.** They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the 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 an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously.

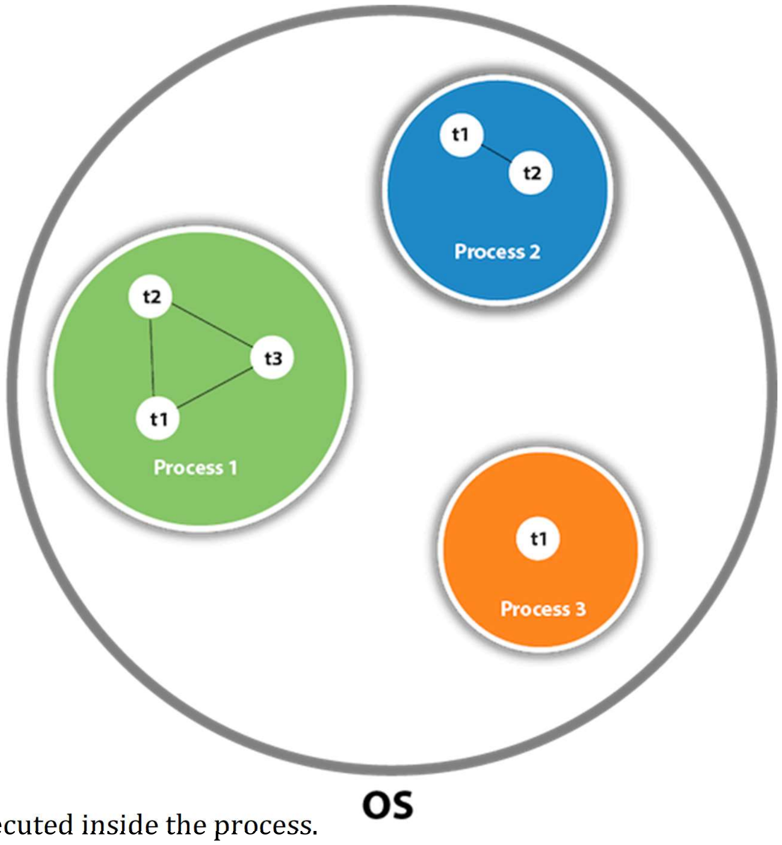
Use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- **Thread-based Multitasking (Multithreading)**
  - Threads share the same address space.
  - A thread is lightweight.
  - Cost of communication between the thread is low.

What is Thread in java?

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



a thread is executed inside the process.

There is context-switching between the threads.

There can be multiple processes inside the OS, and one process can have multiple threads.

Thread Creation

Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

Java provides **Thread class** to achieve thread programming.

Thread class provides constructors and methods to create and perform operations on a thread.

Thread class extends Object class and implements Runnable interface.

S.No	Modifier and Type	Method	Description

1)	void	start()	It is used to start the execution of the thread.
2)	void	run()	It is used to do an action for a thread.
3)	static void	sleep()	It sleeps a thread for the specified amount of time.
4)	static Thread	currentThread()	It returns a reference to the currently executing thread object.
5)	void	join()	It waits for a thread to die.
6)	int	getPriority()	It returns the priority of the thread.
7)	void	setPriority()	It changes the priority of the thread.
8)	String	getName()	It returns the name of the thread.
9)	void	setName()	It changes the name of the thread.
10)	long	getId()	It returns the id of the thread.
11)	boolean	isAlive()	It tests if the thread is alive.
12)	static void	yield()	It causes the currently executing thread object to pause and allow other threads to execute temporarily.
13)	void	suspend()	It is used to suspend the thread.
14)	void	resume()	It is used to resume the suspended thread.
15)	void	stop()	It is used to stop the thread.
16)	void	destroy()	It is used to destroy the thread group and all of its subgroups.
17)	boolean	isDaemon()	It tests if the thread is a daemon thread.
18)	void	setDaemon()	It marks the thread as daemon or user thread.
19)	void	interrupt()	It interrupts the thread.
20)	boolean	isinterrupted()	It tests whether the thread has been interrupted.
21)	static boolean	interrupted()	It tests whether the current thread has been interrupted.
22)	static int	activeCount()	It returns the number of active threads

			in the current thread's thread group.
23)	void	checkAccess()	It determines if the currently running thread has permission to modify the thread.
24)	static boolean	holdLock()	It returns true if and only if the current thread holds the monitor lock on the specified object.
25)	static void	dumpStack()	It is used to print a stack trace of the current thread to the standard error stream.
26)	StackTraceElement[]	getStackTrace()	It returns an array of stack trace elements representing the stack dump of the thread.
27)	static int	enumerate()	It is used to copy every active thread's thread group and its subgroup into the specified array.
28)	Thread.State	getState()	It is used to return the state of the thread.
29)	ThreadGroup	getThreadGroup()	It is used to return the thread group to which this thread belongs
30)	String	toString()	It is used to return a string representation of this thread, including the thread's name, priority, and thread group.
31)	void	notify()	It is used to give the notification for only one thread which is waiting for a particular object.
32)	void	notifyAll()	It is used to give the notification to all waiting threads of a particular object.
33)	void	setContextClassLoader()	It sets the context ClassLoader for the Thread.
34)	ClassLoader	getContextClassLoader()	It returns the context ClassLoader for the thread.
35)	static Thread.UncaughtExceptionHandler	getDefaultUncaughtExceptionHandler()	It returns the default handler invoked when a thread abruptly terminates due to an uncaught exception.
36)	static void	setDefaultUncaughtExcept	It sets the default handler invoked when

		ionHandler()	a thread abruptly terminates due to an uncaught exception.

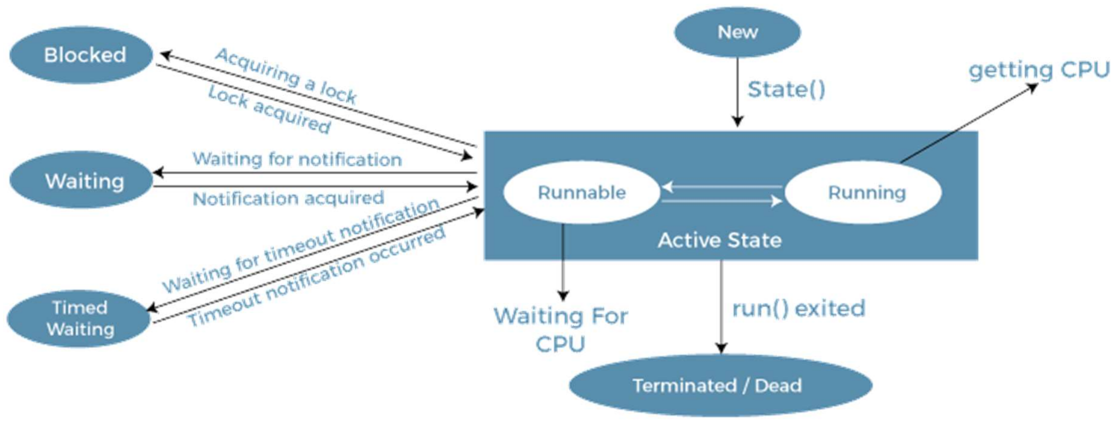
Constructors of Thread Class

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Life cycle of a Thread (Thread States)

A thread always exists in any one of the following states. These states are:

- New
- Active
- Blocked / Waiting
- Timed Waiting
- Terminated



Life Cycle of a Thread

Thread creation by extending the Thread class

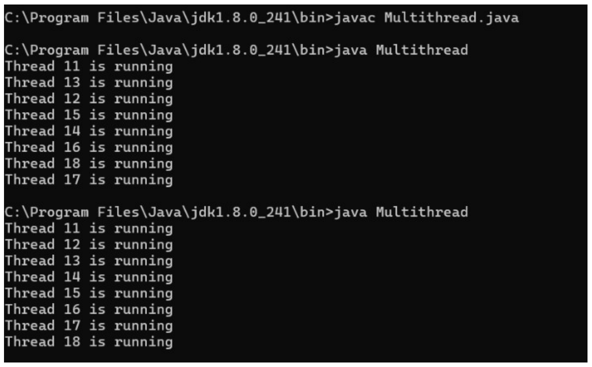
We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class.

A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Example

class MultithreadingDemo extends Thread {

```
public void run()
{
try {
// Displaying the thread that is running
System.out.println("Thread " + Thread.currentThread().getId()+ " is running");
}
catch (Exception e) {
// Throwing an exception
System.out.println("Exception is caught");
}
}
}
```



```
C:\Program Files\Java\jdk1.8.0_241\bin>javac Multithread.java

C:\Program Files\Java\jdk1.8.0_241\bin>java Multithread
Thread 11 is running
Thread 13 is running
Thread 12 is running
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 18 is running
Thread 17 is running

C:\Program Files\Java\jdk1.8.0_241\bin>java Multithread
Thread 11 is running
Thread 12 is running
Thread 13 is running
Thread 14 is running
Thread 15 is running
Thread 16 is running
Thread 17 is running
Thread 18 is running
```

```
// Main Class
public class Multithread{
public static void main(String[] args)
{
int n = 8; // Number of threads
for (int i = 0; i < n; i++) {
MultithreadingDemo object= new MultithreadingDemo();
object.start();
}
}
}
```

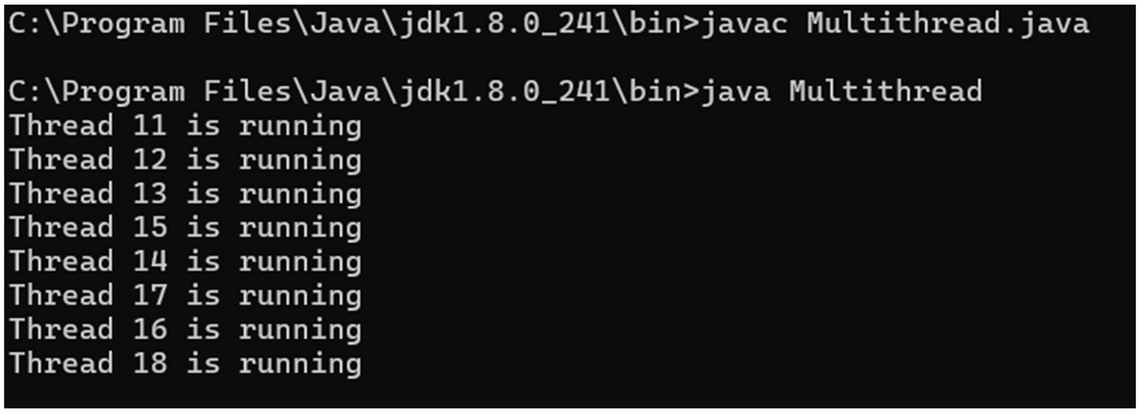
Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

Example

```
class MultithreadingDemo implements Runnable {
public void run()
{
try {
// Displaying the thread that is running
System.out.println("Thread " + Thread.currentThread().getId()+ " is running");
}
catch (Exception e) {
// Throwing an exception
System.out.println("Exception is caught");
}
}
}

// Main Class
class Multithread {
public static void main(String[] args)
{
int n = 8; // Number of threads
for (int i = 0; i < n; i++) {
Thread object= new Thread(new MultithreadingDemo());
object.start();
}
}
```



```
C:\Program Files\Java\jdk1.8.0_241\bin>javac Multithread.java

C:\Program Files\Java\jdk1.8.0_241\bin>java Multithread
Thread 11 is running
Thread 12 is running
Thread 13 is running
Thread 15 is running
Thread 14 is running
Thread 17 is running
Thread 16 is running
Thread 18 is running
```

Sr. No.	**Key**	**Thread**	**Runnable**
1	Basic	Thread is a class. It is used to create a thread	Runnable is a functional interface which is used to create a thread
2	Methods	It has multiple methods including start() and run()	It has only abstract method run()

Sr. No.	Key	Thread	Runnable
3		Each thread creates a unique object and gets associated with it	Multiple threads share the same objects.
4	Memory	More memory required	Less memory required
5	Limitation	Multiple Inheritance is not allowed in java hence after a class extends Thread class, it can not extend any other class	If a class is implementing the runnable interface then your class can extend another class.