Cairo University
Faculty of Computers & Artificial Intelligence
Computer Science Department
Operating Systems 1 Course

Lab 3: Threads

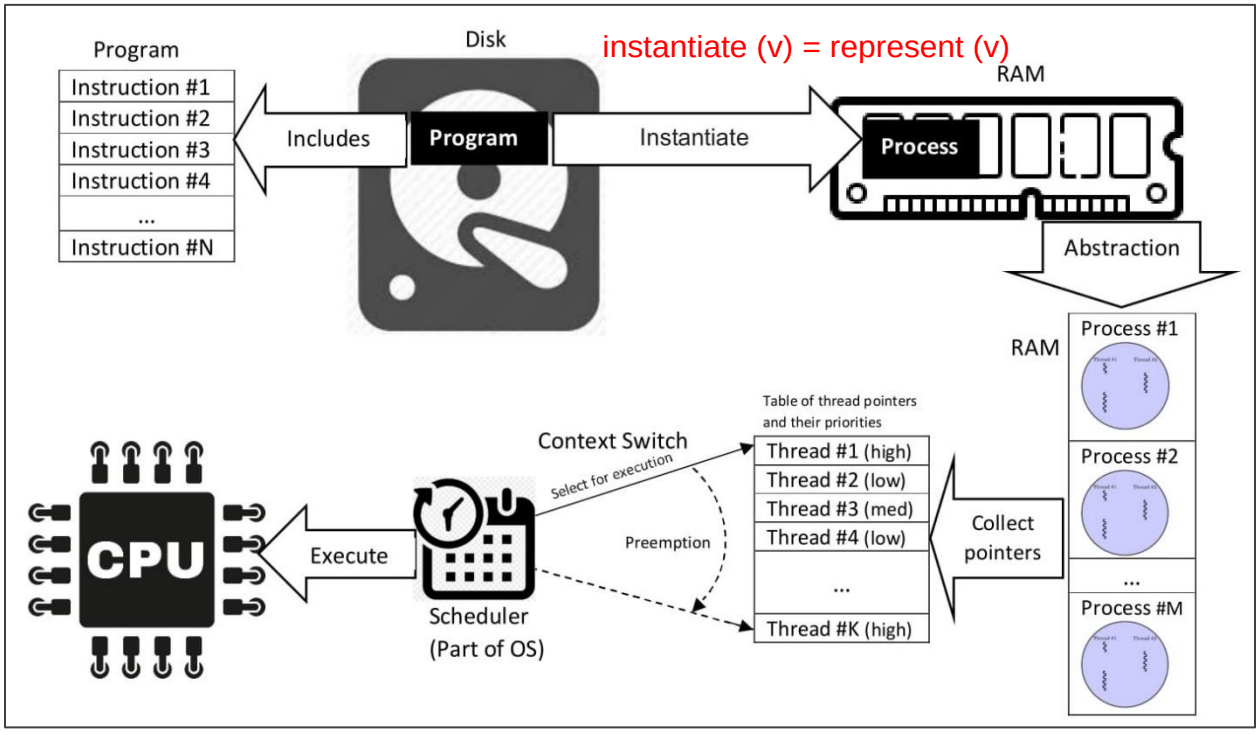
Theoretical part:

One of the functions of an operating system is handling program execution.

A ***program*** is an executable file *(stored in secondary storage)* that contains instructions to perform a certain task. When we double click a program to run it, the OS loads the program into memory and starts a ***process***.

A ***process*** is an ***instance of a program being executed***. Several processes may be related to same program.

A ***thread*** is the ***smallest executable unit of a process***. A single process can have multiple threads sharing the process' memory, each having its own task and execution path.

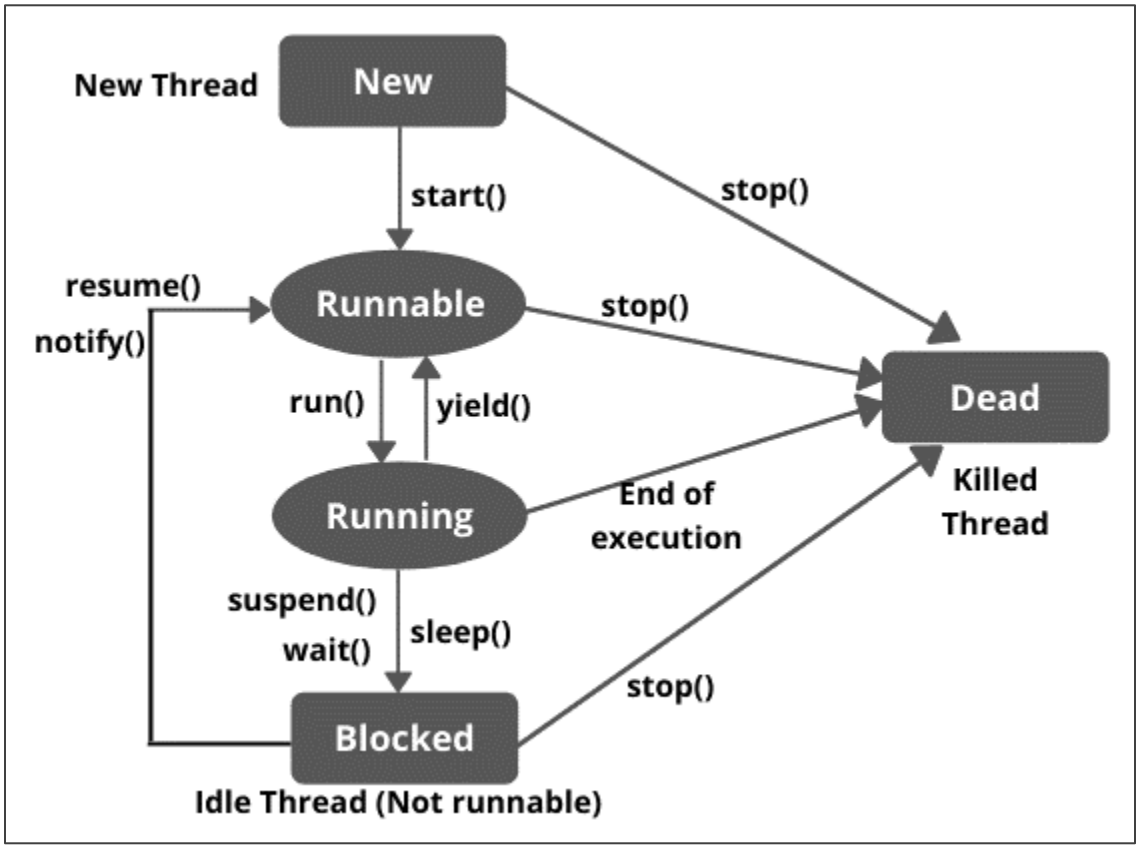


In ***concurrent programming***, ***multiple processes are being executed during the same period of time*** (not necessarily executing at the same physical instant as in parallel programming). This means that in concurrent programming ***the processes' lifetimes overlap***, but the execution doesn't have to happen at the same instant.

In the ***Java*** programming language, concurrent programming is mostly concerned with ***threads***. The ***Java Virtual Machine (JVM)*** allows an application to have multiple threads of execution running concurrently. When a JVM starts up, there is usually a single thread (which typically calls the ***"main"*** method of some class). The JVM continues to execute threads until either of the following occurs:

- The exit method of class Runtime has been called and the security manager has permitted the exit operation to take place.
- All threads that are not daemon threads have died, either by returning from the call to the run method or by throwing an exception that propagates beyond the run method.

The thread lifecycle in java:



Technical part:

✓ *Defining a thread:*

An application that creates an instance of "Thread" must provide the code that will run in that thread. ***There are two ways to do this:***

1. **Implementing the *"Runnable"* interface:**
   The "Runnable" interface defines a single method ***"run"*** meant to contain the code to be executed in the thread. The new runnable object is passed to the "Thread" constructor.

```
public class MyRunnable implements Runnable{
	@Override
	public void run() {
		for (int i = 0; i < 1000; i++)
			System.out.println("MyRunnable is running!");
	}
}
```

2. **Extending the *"Thread"* class:**
   The "Thread" class itself implements Runnable, though its "run" method does nothing. An application can subclass "Thread", providing its own implementation of "run".

```
public class MyThread extends Thread{
	String name;

	MyThread(String name) { this.name = name;}

	@Override
	public void run() {
		for (int i = 0; i < 1000; i++)
			System.out.println(name + " is running!");
	}
}
```

✓ *Starting a thread:*

In the class containing your "main" function, create an object of your thread and call its ***"start"*** function.

```
	public static void main(String[] args) {
		Thread t1 = new Thread(new MyRunnable());
		Thread t2 = new MyThread("A");
		Thread t3 = new MyThread("B");

		t1.start(); t2.start(); t3.start();

		System.out.println("In main thread!");
	}
```

the start method will execute after finishing last line of the whole program

this print line will execute first

✓ ***Setting a thread's priority:***

Every thread has a priority; threads with higher priority are executed in preference to threads with lower priority. Initially, a thread's priority is set equal to the priority of the thread that creates it. We can use ***"setPriority(int newPriority)"*** to change the priority of a thread. ***We can also use the predefined constants:***

```
public static final int MIN_PRIORITY;
public static final int NORM_PRIORITY;
public static final int MAX_PRIORITY;
```

```
	public static void main(String[] args) {
		Thread t1 = new Thread(new MyRunnable());
		Thread t2 = new MyThread("A");
		Thread t3 = new MyThread("B");

		t3.setPriority(10); //MAX_PRIORITY
		t1.setPriority(Thread.MIN_PRIORITY);

		t1.start(); t2.start(); t3.start();

		System.out.println("In main thread!");
	}
```

✓ ***Pausing a thread using sleep:***

The ***"sleep"*** method causes the current thread to ***suspend execution*** for a specified period (in milliseconds).

```
public class MyRunnable implements Runnable{
	@Override
	public void run() {
		for (int i = 0; i < 1000; i++) {
			System.out.println("MyRunnable is running!");

			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

✓ ***Interrupting a thread:***

The ***"interrupt"*** method is an indication to a thread that it should ***stop what it is doing and do something else.*** It's up to the programmer to decide exactly how a thread responds to an interrupt, but it is very common for the thread to terminate. We can use the method ***"interrupted"*** to ***test whether the current thread has been interrupted.***

```
public class MyRunnable implements Runnable{
	@Override
	public void run() {
		for (int i = 0; i < 1000; i++) {
			System.out.println("MyRunnable is running! " + i);

			if (Thread.interrupted()) {
				System.out.println("Interrupted!");
				break;
			}
		}
	}
}
```

```
	public static void main(String[] args) {
		Thread t1 = new Thread(new MyRunnable());
		Thread t2 = new MyThread("A");
		Thread t3 = new MyThread("B");

		t3.setPriority(10); //MAX_PRIORITY
		t1.setPriority(Thread.MIN_PRIORITY);

		t1.start(); t2.start(); t3.start();

		System.out.println("In main thread!");

		t1.interrupt();
	}
```

✓ ***Checking whether a thread is running:***

The ***"isAlive"*** method checks whether a thread is still running. Add this piece of code at the end of the "main" function and observe the output.

```
		while (t1.isAlive());
		System.out.println("T1 is dead!");
```

✓ ***Joining a thread:***

The ***"join"*** method ***allows one thread to wait for the completion of another***. In the following example, we called ***t1.join(); t2.join(); t3.join()*** in the main thread, so the main thread waits for t1, t2 and t3 to finish their execution before continuing its execution.

```
public static void main(String[] args) throws InterruptedException {
	Thread t1 = new Thread(new MyRunnable());
	Thread t2 = new MyThread("A");
	Thread t3 = new MyThread("B");

	t3.setPriority(10); //MAX_PRIORITY
	t1.setPriority(Thread.MIN_PRIORITY);

	t1.start(); t2.start(); t3.start();
	t1.join(); t2.join(); t3.join();

	System.out.println("In main thread!");
}
```

*Note: Like "**sleep**", "**join**" responds to an interrupt by exiting with an InterruptedException.*

✓ ***Pausing a thread using yield:***

The ***"yield"*** method ***causes the currently executing thread object to temporarily pause and allow other threads to execute***. In the following example, each thread prints a message and then gives the role to other threads to execute, ***s***o the behavior appears as if each thread prints then the CPU switches to the other thread. If we didn't use "yield", the switch between the threads will be random.

```
public class MyThread extends Thread{
	String name;
	MyThread(String name) { this.name = name;}
	@Override
	public void run() {
		for (int i = 0; i < 1000; i++) {
			System.out.println(name + " is running!");
			Thread.yield();
		}
	}
}
```

```
public static void main(String[] args) {
	//Thread t1 = new Thread(new MyRunnable());
	Thread t2 = new MyThread("A");
	Thread t3 = new MyThread("B");

	t2.start(); t3.start();
}
```

Exercises

1. Write a class ***"StopWatch"*** whose ***"main"*** function acts as a stopwatch and prints the time each second until it is terminated. *Hint: Use* ***"sleep"***.

2. Write a program that takes two matrices as input and displays their addition result. ***The program should use threads to add the matrices.***
   Hints:
   - *Make a thread for each column (or row).*
   - *Each thread adds two corresponding columns (or rows) and writes the result in the shared output matrix.*
   - *The parent thread must wait till the children threads finish their work.*