Unit V
Multithreaded Programming

Tutor: Pema Galey

Learning Outcomes

In this session, you will learn about:

- Multithreaded Programming
- Threads in Java
- Life Cycle of a Thread

Multithreaded

- Java provides built-in support for ***multithreaded programming***.
- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a ***thread***, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.
- Example for multitasking :
  - Laptop OS, Smart Phones, etc.

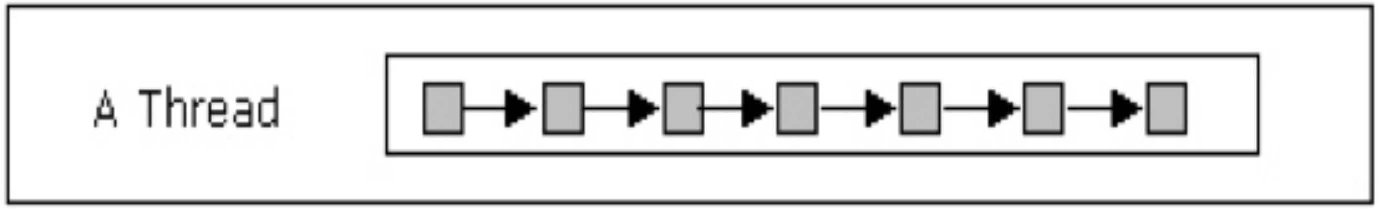
Multithreaded

- A process is broken into small chunks called ***tasks***, and tasks are further broken into smaller chunks called ***threads***.
- A process that is made of one thread is known as ***single-threaded process***.
- A process that creates two or more threads is called a ***multi-threaded process***.
- Each thread in a multi-threaded program runs at the same time and has a different execution path.

Basic Concept of Multithreading

Single-Threaded process:

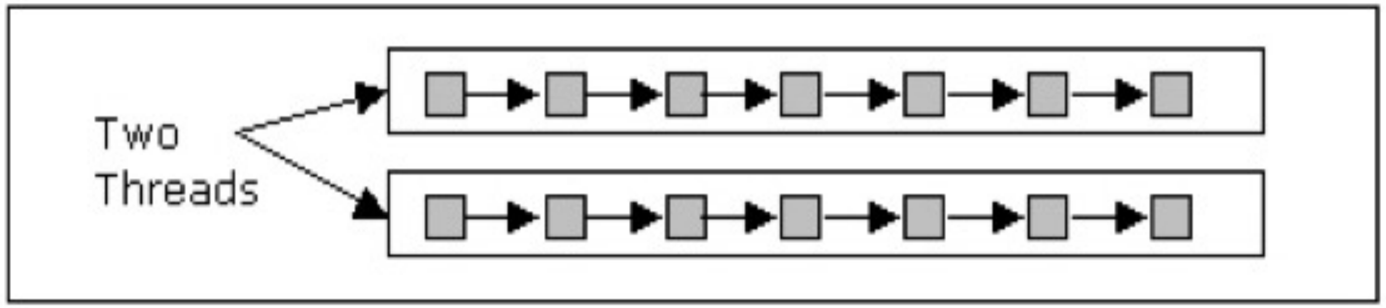
- A single-threaded application can perform only one task at a time.
- The following figure shows a single threaded program.



Basic Concept of Multithreading

Multi-Threaded process :

- Multithreading helps perform multiple operations simultaneously, saving the time of the user.
- The following figure shows a multi-threaded program.



Basic Concept of Multithreading

- **Multitasking** is the ability to execute more than one task at the same time.
- Multitasking can be divided into the following categories:
  - *Process-based multitasking*
  - *Thread-based multitasking*

- **Process-Based** multitasking:
  - Enables you to switch from one program to another so quickly that it appears as if the programs are executing at the same time.
  - Enables a computer to execute two or more processes concurrently.

Basic Concept of Multithreading

- **Thread-Based** multitasking:
  - A single program can contain two or more threads and therefore, can perform two or more tasks simultaneously.
  - Threads are called lightweight process because there are fewer overloads when the processor switches from one thread to another.

- **Benefits of multithreading**:
  - Improved performance
  - Minimized system resource usage
  - Simultaneous access to multiple applications
  - Program structure simplification

Basic Concept of Multithreading

- **Pitfalls of multithreading**:
  - **Race condition**: This condition is caused when synchronization between the two threads is not implemented.
  - **Deadlock condition**: The deadlock condition arises in a computer system when two threads wait for each other to complete their operations before performing their individual action. As a result, the two threads are locked and the program fails.
  - **Lock starvation**: Lock starvation occurs when the execution of a thread is postponed because of its low priority.

Java Thread Model

- In **single-threaded systems**, an approach called event loop with polling is used.
- Polling is the process in which a single event is executed at a time.
- In the event loop model:
  - A single thread runs in an infinite loop till its operation is completed.
  - When this operation is completed, the event loop dispatches control to the appropriate event-handler.
  - No more processing can happen in the system until the event handler returns. This results in the wastage of the CPU time.
- In a **multi-threaded Java** application, event loop/polling mechanism is eliminated.

Thread Class in Java

- The `java.lang.Thread` class is used to construct and access individual threads in a multi-threaded application.
- You can create a multi-threaded application by using the **Thread** class or the **Runnable** interface.
- The following list represents the various methods defined in the Thread class:
  - `getPriority()`: Returns the priority of a thread.
  - `isAlive`(): Determines whether a thread is running.
  - `sleep`(): Makes the thread pause for a period of time.
  - `getName`(): Returns the name of the thread.
  - `start`(): Starts a thread by calling the run()method.

Thread Class in Java

- The **main** thread:
  - It is the first thread to be executed in a multi-threaded process.
  - It is created automatically when a Java program is executed.
  - You can access a thread by using the currentThread() method of the Thread class.
  - The following code shows the syntax of the currentThread() method:

```
public static Thread currentThread()
```

Thread Class : Example

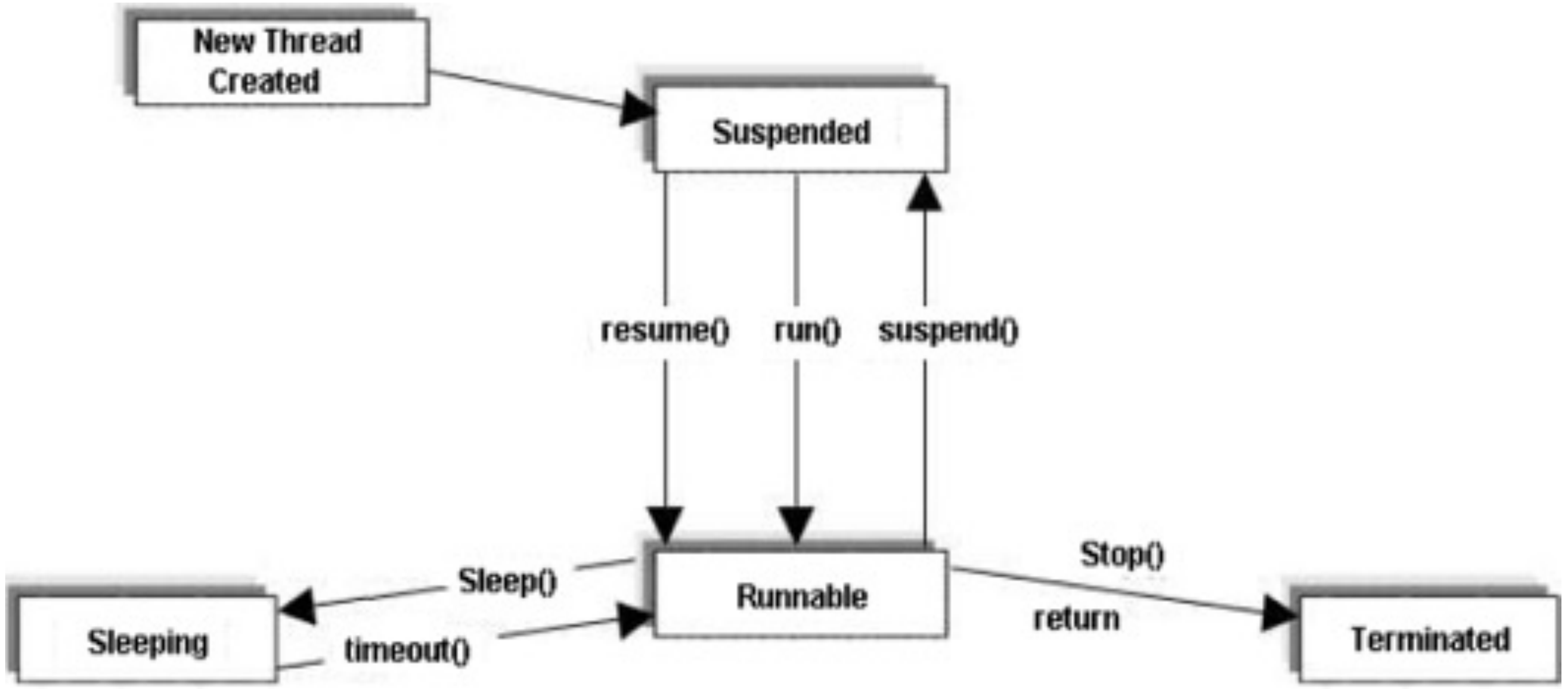
```
class MainThreadDemo{
   public static void main(String args[]){
      Thread t=Thread.currentThread();
      System.out.println("The current thread is :"+t);
      t.setName("MainThread");
      System.out.println("The current thread after name change: "+t);
      System.out.println("The current Thread is going to sleep for 5 seconds:");
      try{
         t.sleep(5000);
      }
      catch(InterruptedException e){
         System.out.println("Main Thread Interrupted!");
      }
      System.out.println("After 5 seconds..The current Thread is exiting now.");
   }
}
```

The Life Cycle of a Thread

- The various states in the life cycle of a thread are:
  1. New
  2. Runnable
  3. Not runnable
  4. Terminated or dead

The Life Cycle of a Thread

- The following figure shows the life cycle of a thread.



The Life Cycle of a Thread

1. **The new thread state**:
   - The following code shows the syntax to instantiate the Thread class:

```
Thread newThread = new Thread(this, "threadName");
```

   - The following code shows the syntax to start a thread:

```
newThread.start();
```

2. **The runnable thread state**:
   - When the `start()` method of a thread is invoked, the thread enters the runnable state.
   - The `start()` method allocates the system resources to the thread, schedules the thread, and passes the control to its `run()` method.

The Life Cycle of a Thread

3) **The not runnable thread state**:

- A thread is not in the runnable state if it is:
  - Sleeping
  - Waiting
  - Being blocked by another thread

4) **The dead thread state**:

- A thread enters the dead state when the loop in the run() method is complete.
- Assigning a null value to a thread object changes the state of the thread to dead.
- The isAlive() method of the Thread class is used to determine whether a thread has been started.
- You cannot restart a dead thread.

Thank you!