Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf
yousuf@juniv.edu

Multithreaded Programming-1

Introduction

- 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.

Introduction

- Two distinct types of multitasking:
  - process-based Multitasking(Multiprocessing)
  - thread-based Multitasking(Multithreading)

Introduction

Process-based Multitasking(Multiprocessing) :

- A *process is, in essence, a program that is executing.*
- Each process have **its own address in memory** i.e. each process allocates separate memory area.
- *Thus, process-based* multitasking is the feature that allows your computer to run two or more programs concurrently.
  - For example, process-based multitasking enables you to run the Java compiler at the same time that you are using a text editor or visiting a web site.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

Introduction

Thread-based Multitasking (Multithreading):

- Threads share the **same address space**.
- In a *thread-based multitasking environment, the thread is the smallest unit of* dispatchable code. This means that a single program can perform two or more tasks simultaneously.
  - For instance, a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads.
- Cost of communication between the thread is low.

Introduction

- Multiprocessing and multithreading, both are used to achieve multitasking. But we use multithreading than multiprocessing because **threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.**
- **Note: At least one process is required for each thread.**

Introduction

- **Advantages of Java Multithreading:**

  1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

  2) You **can perform many operations together so it saves time**.

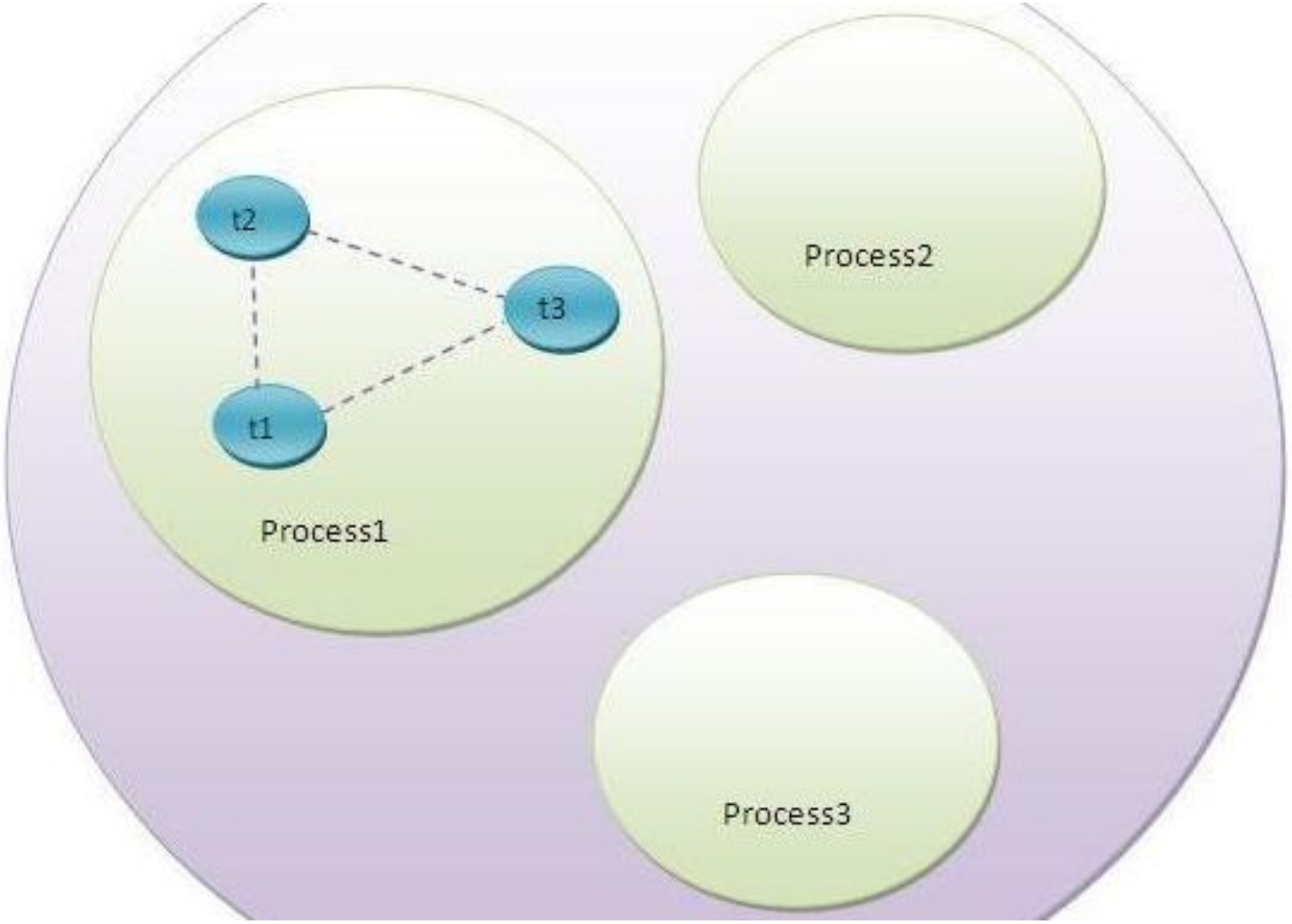
  3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

Introduction

What is Thread in java:

- A thread is a lightweight sub process, a 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 shares a common memory area.
- **Note: At a time one thread is executed only.**

Introduction



As shown in the above figure, 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.

States of a thread

Threads exist in several states.

- A thread can be *running*.
- It can be *ready to run as soon as it gets CPU time.*
- *A running thread can be suspended,* which temporarily halts its activity.
- A suspended thread can then be *resumed, allowing it* to up where it left off.
- A thread can be *blocked when waiting for a resource.*
- *At any* time, a thread can be terminated, which halts its execution immediately. Once terminated, a thread cannot be resumed.

States of a thread

- The Java **Thread** **class defines several methods that help manage threads. Several of those** used are shown here:

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.
join	Wait for a thread to terminate.
run	Entry point for the thread.
sleep	Suspend a thread for a period of time.
start	Start a thread by calling its run method.

The Main Thread

- When a Java program starts up, one thread begins running immediately.
- This is usually called the *main thread of your program, because it is the one that is executed when your* program begins.
- The main thread is important for two reasons:

  1) It is the thread from which other "child" threads will be spawned.

  2) Often, it must be the last thread to finish execution because it performs various shut down actions.

The Main Thread

- Although the main thread is created automatically when your program is started, it can be controlled through a **Thread object.**
- **To do so, you must obtain a reference to it** by calling the method **currentThread(), which is a public static member of Thread.**
- This method returns a reference to the thread in which it is called. Once you have a reference to the main thread, you can control it just like any other thread.

Example 1

```
// Controlling the main Thread.
class CurrentThreadDemo {
  public static void main(String args[]) {
     Thread t = Thread.currentThread();

    System.out.println("Current thread: " + t);
```

```
    // change the name of the thread
    t.setName("My Thread");
    System.out.println("After name change: " + t);

    try {
      for(int n = 5; n > 0; n--) {
        System.out.println(n);
        Thread.sleep(1000);
      }
     } catch (InterruptedException e) {
      System.out.println("Main thread interrupted'
    }
  }
}
```

Explanation of previous program

- In this program, a reference to the current thread (the main thread, in this case) is obtained by calling **currentThread(), and this reference is stored in the local variable t.**
- **Next, the program displays information about the thread.**
- The program then calls **setName() to change the internal name of the thread. Information about the thread is** then redisplayed.
- Next, a loop counts down from five, pausing one second between each line. The pause is accomplished by the **sleep() method.**

Explanation of previous program

- Here is the output generated by this program:

  Current thread: Thread[main,5,main]

  After name change: Thread[My Thread,5,main]

  5

  4

  3

  2

  1

Explanation of previous program

- This displays, in order: the name of the thread, its priority, and the name of its group.
- By default, the name of the main thread is **main. Its priority is 5, which is the default value, and main** is also the name of the group of threads to which this thread belongs.
- A *thread group is a* data structure that controls the state of a collection of threads as a whole.
- After the name of the thread is changed, **t is again output. This time, the new name of the thread** is displayed.

Explanation of previous program

- The **sleep( ) method causes the thread from which it is called to suspend** execution for the specified period of milliseconds.
- you can set the name of a thread by using **setName().**

Creating a Thread

- In the most general sense, you create a thread by instantiating an object of type **Thread.**
- **Java defines two ways to create thread:**
  - By extending **Thread class.**
  - By implementing **Runnable interface.**

Creating a Thread

Thread class:

- Thread class provide **constructors and methods to create and perform operations on a thread**. Thread class extends Object class and implements Runnable interface.
- Commonly used **Constructors of Thread** class:
  - Thread()
  - Thread(String name)
  - Thread(Runnable r)
  - Thread(Runnable r, String name)

Creating a Thread

- Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.

Creating a Thread

- Commonly used methods of Thread class:

11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Creating a Thread

- **Java Thread Example by extending Thread class**

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output: thread is running...

Creating a Thread

- Java Thread Example by implementing Runnable interface:

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

If you are not extending the Thread class, **your class object would not be treated as a thread object**. So you need **to explicitely create Thread class object**. We are passing the object of your class that implements Runnable so that your class run() method may execute.

Output: thread is running...

Thank you