Programming Language II
CSE-215

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

Multithreaded Programming-2

Thread Scheduler in Java

- **Thread scheduler** in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Difference between preemptive scheduling and time slicing

- Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.

- Under time slicing, a task executes for a predefined slice of time and then re-enters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

Sleep method in java

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- Syntax of sleep() method in java
  - The Thread class provides a method for sleeping a thread:
  - **public static void sleep(long miliseconds)throws InterruptedException**

Example of sleep method in java

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
       try{
          Thread.sleep(500);
         }catch(Exception e){System.out.println(e);}
       System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();
  t2.start();
 }
}
```

Output:
1
1
2
2
3
3
4
4

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

Can we start a thread twice?

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

```
public class TestThreadTwice1 extends Thread{

 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  TestThreadTwice1 t1=new TestThreadTwice1()
;
  t1.start();
  t1.start();
 }
}
```

Output:
running
Exception in thread "main" java.lang.IllegalThreadStateException

What if we call run() method directly instead start() method?

- The start method makes sure the code runs in a new thread context. If you called run directly, then it would be like an ordinary method call and it would run in the context of the *current* thread instead of the new one. The start method contains the special code to trigger the new thread; run obviously doesn't have that ability because *you* didn't include it when you wrote the run method.

What if we call run() method directly instead start() method?

- Each thread starts in a separate call stack.
- Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

```
class TestCallRun1 extends Thread{
 public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
  TestCallRun1 t1=new TestCallRun1();
  t1.run();//fine, but does not start a separate call stack
 }
}
```

Output:
running...

Problem if you direct call run() method

```
class TestCallRun2 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestCallRun2 t1=new TestCallRun2();
  TestCallRun2 t2=new TestCallRun2();

  t1.run();
  t2.run();
 }
}
```

```
Output:1
       2
       3
       4
       5
       1
       2
       3
       4
       5
```

As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object.

The join() method

- **Java join()** method can be used to pause the current thread execution until unless the specified thread is dead. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.
- **Syntax:**
  - public void join()throws InterruptedException
  - public void join(long milliseconds)throws InterruptedException

Example of join(long miliseconds) method (Next slide)

```
class TestJoinMethod1 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try{
  t1.join()
 }catch(Exception e){System.out.println(e);}

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

Example 1

```
Output:1
2
3
4
5
1
1
2
2
3
3
4
4
5
5
```

In the above example, when t1 completes its task then t2 and t3 starts executing.

```
class TestJoinMethod2 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod2 t1=new TestJoinMethod2();
 TestJoinMethod2 t2=new TestJoinMethod2();
 TestJoinMethod2 t3=new TestJoinMethod2();
 t1.start();
 try{
  t1.join(1500);
 }catch(Exception e){System.out.println(e);}
 t2.start();
 t3.start();
 }
}
```

```
Output:1
       2
       3
       1
       4
       1
       2
       5
       2
       3
       3
       4
       4
       5
       5
```

Example 2

In the above example, when t1 is completes its task for 1500 miliseconds (3 times) then t2 and t3 starts executing.

getName(),setName(String) and getId() method:

- public String getName()
- public void setName(String name)
- public long getId()

```
class TestJoinMethod3 extends Thread{
 public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
 TestJoinMethod3 t1=new TestJoinMethod3();
 TestJoinMethod3 t2=new TestJoinMethod3();
 System.out.println("Name of t1:"+t1.getName());
 System.out.println("Name of t2:"+t2.getName());
 System.out.println("id of t1:"+t1.getId());

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

 t1.setName("ICT");
 System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

Output:
Name of t1:Thread-0
Name of t2:Thread-1
id of t1:8
running...
After changling name of t1: ICT
running...

Creating Thread by Implementing Runnable

1) The easiest way to create a thread is to create a class that implements the **Runnable** interface.

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

3)To implement Runnable, a class need only implement a single method called **run(), which is declared like this:**

public void run( )

Creating Thread by Implementing Runnable

- After you create a class that implements **Runnable, you will instantiate an object of** type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

  Thread(Runnable *threadOb, String threadName)*

- In this constructor, *threadOb is an instance of a class that implements the* ***Runnable*** interface. The name of the new thread is specified by *threadName.*

Creating Thread by Implementing Runnable

- After the new thread is created, it will not start running until you call its **start()** method, which is declared within **Thread.**
- **In essence, start() executes a call to run().**
- The **start() method is shown here:**

```
void start()
```

```
class RunnableDemo implements Runnable
 { private Thread t;
   private String threadName;
   RunnableDemo( String name) {
    threadName = name;
    System.out.println("Creating " + threadName );
    }
    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50); }
        }catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
       System.out.println("Thread " + threadName + " exiting.");
    }
    public void start () {
        System.out.println("Starting " + threadName );
        if (t == null) {
            t = new Thread (this, threadName);
            t.start (); }
    }
}
```

Example that creates a new thread and starts it running

Example that creates a new thread and starts it running

```
public class TestThread {
    public static void main(String args[]) {
            RunnableDemo R1 = new RunnableDemo( "Thread-1");
            R1.start();
            RunnableDemo R2 = new RunnableDemo( "Thread-2");
            R2.start();
    }
}
```

Example that creates a new thread and starts it running

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   ThreadDemo( String name) {
      threadName = name;
      System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
      }catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");}
      System.out.println("Thread " +  threadName + " exiting.");
   }
   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      } }
}
```

preceding program rewritten to extend the Thread

Preceding program rewritten to extend the Thread

```
public class TestThread {
   public static void main(String args[]) {
      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();
      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

Preceding program rewritten to extend the Thread

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Example that creates a new thread and starts it running

```
    // Create a second thread.
    class NewThread implements Runnable {
      Thread t;

      NewThread() {
        // Create a new, second thread
        t = new Thread(this, "Demo Thread");
        System.out.println("Child thread: " + t);
        t.start(); // Start the thread
      }

      // This is the entry point for the second thread.
      public void run() {
        try {
          for(int i = 5; i > 0; i--) {
            System.out.println("Child Thread: " + i);
            Thread.sleep(500);
          }
      } catch (InterruptedException e) {
        System.out.println("Child interrupted.");
      }
      System.out.println("Exiting child thread.");
    }
}
```

Example that creates a new thread and starts it running

```
class ThreadDemo {
  public static void main(String args[ ] ) {
    new NewThread(); // create a new thread

    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Main Thread: " + i);
        Thread.sleep(1000);
      }
    } catch (InterruptedException e) {

      System.out.println("Main thread interrupted.");
    }
    System.out.println("Main thread exiting.");
  }
}
```

Example that creates a new thread and starts it running

- Inside **NewThread's constructor, a new Thread object is created by the following** statement:

  t = new Thread(this, "Demo Thread");

- Passing **this as the first argument indicates that you want the new thread to call the run( ) method on this object.**
- **Next, start() is called, which starts the thread of** execution beginning at the **run( ) method.**

Example that creates a new thread and starts it running

- This causes the child thread's **for loop to** begin.
- After calling **start(), NewThread's constructor returns to main().**
- **When the** main thread resumes, it enters its **for loop. Both threads continue running, sharing the** CPU in single-core systems, until their loops finish.

Example that creates a new thread and starts it running

- Out put of previous program:

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
```

```
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

Example that creates a new thread and starts it running

- As mentioned earlier, in a multithreaded program, **often the main thread must be the last thread to finish running.** In fact, for some older JVMs, if the main thread finishes before a child thread has completed, then the Java run-time system may "hang."
- The preceding program ensures that the main thread finishes last, because the main thread sleeps for 1,000 milliseconds between iterations, but the child thread sleeps for only 500 milliseconds. This causes the child thread to terminate earlier than the main thread.
- Shortly, you will see a better way to wait for a thread to finish.

Creating Thread by Extending Thread Class

- The second way to create a thread is to create a new class that extends **Thread, and** then to create an instance of that class.
- The extending class must override the **run()** method, which is the entry point for the new thread.
- It must also call **start() to begin** execution of the new thread.

Preceding program rewritten to extend **Thread**

```
// Create a second thread by extending Thread
class NewThread extends Thread {

  NewThread() {
    // Create a new, second thread
    super("Demo Thread");
    System.out.println("Child thread: " + this);
    start(); // Start the thread
  }

  // This is the entry point for the second thread.
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
        Thread.sleep(500);
      }
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}
```

Preceding program rewritten to extend **Thread**

```
class ExtendThread {
  public static void main(String args[]) {
    new NewThread(); // create a new thread

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

Preceding program rewritten to extend **Thread**

- This program generates the same output as the preceding version.
- Here, child thread is created by instantiating an object of **NewThread, which is derived from Thread.**
- Notice the call to **super( ) inside NewThread. This invokes the following form of the Thread constructor:**

  public Thread(String *threadName)*

  Here, *threadName specifies the name of the thread.*

Creating Multiple Threads

- So far, you have been using only two threads: the main thread and one child thread.

- However, your program can spawn as many threads as it needs. For example, the following program creates three child threads:

```
// Create multiple threads.
class NewThread implements Runnable {
  String name; // name of thread
  Thread t;

  NewThread(String threadname) {
    name = threadname;
    t = new Thread(this, name);
    System.out.println("New thread: " + t);
    t.start(); // Start the thread
  }

  // This is the entry point for thread.
  public void run() {
   try {
      for(int i = 5; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(1000);
      }
    } catch (InterruptedException e) {
        System.out.println(name + "Interrupted");
      }
      System.out.println(name + " exiting.");
    }
```

Creating Multiple Threads

```
class MultiThreadDemo {
  public static void main(String args[]) {
    new NewThread("One"); // start threads
    new NewThread("Two");
    new NewThread("Three");

    try {
      // wait for other threads to end
      Thread.sleep(10000);
    } catch (InterruptedException e) {
      System.out.println("Main thread Interrupted");
    }
    System.out.println("Main thread exiting.");
  }
}
```

- Sample output from this program is shown here.

```
New thread: Thread[One, 5, main]
New thread: Thread[Two, 5, main]
New thread: Thread[Three, 5, main]
One: 5
Two: 5
Three: 5
One: 4
Two: 4
Three: 4
One: 3
Three: 3
Two: 3
One: 2
Three: 2
Two: 2
One: 1
Three: 1
Two: 1
One exiting.
```

```
Two exiting.
Three exiting.
Main thread exiting
```

Creating Multiple Threads

- As you can see, once started, all three child threads share the CPU.
- Notice the call to **sleep(10000) in main( ). This causes the main thread to sleep for ten seconds and** ensures that it will finish last.

Thank you