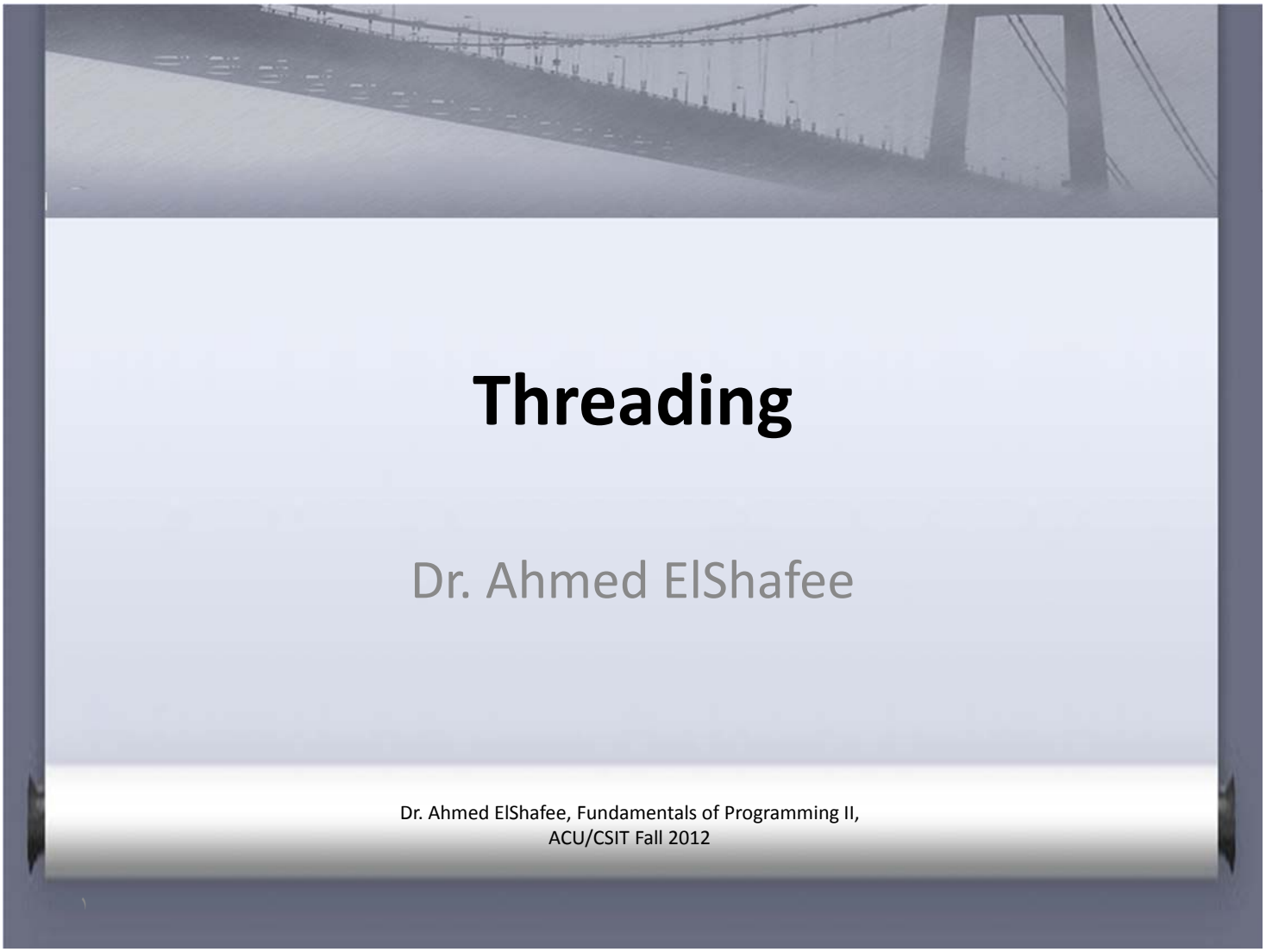


Threading

Dr. Ahmed ElShafee

Agenda

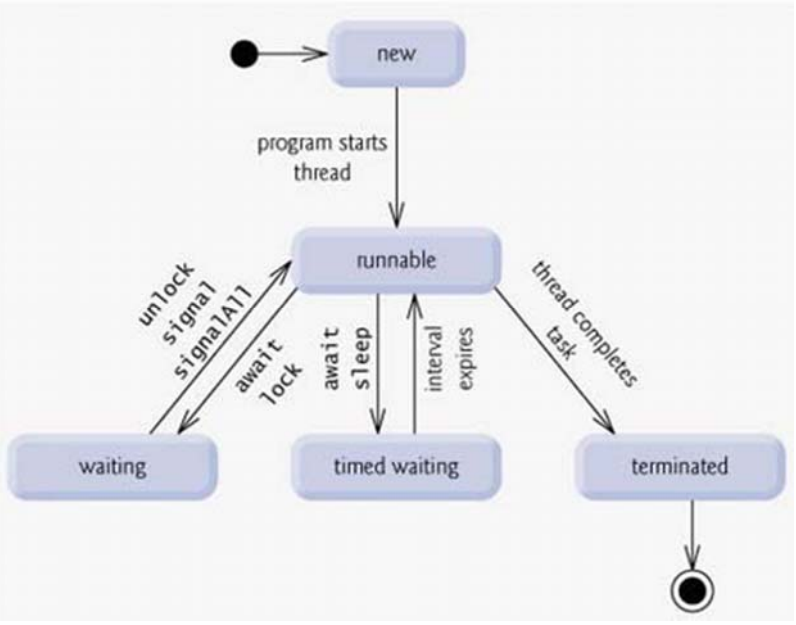
1. *
2. *
3. *

Introduction

- 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.
- Another term related to threads: **process:** A process consists of the memory space allocated by the operating system that can contain one or more threads.
- A thread cannot exist on its own; it must be a part of a process.

Life Cycle of a Thread

- A thread goes through various stages in its life cycle.
- For example, a thread is born, started, runs, and then dies.



- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Priorities

- Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.
- Java priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10).
- By default, every thread is given priority NORM_PRIORITY (a constant of 5).
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads.
- However, thread priorities cannot guarantee the order in which threads execute and very much platform dependent.

Creating a Thread

- Java defines two ways in which this can be accomplished:
  - You can implement the Runnable interface.
  - You can extend the Thread class, itself.

Create Thread by Implementing Runnable

- The easiest way to create a thread is to create a class that implements the **Runnable** interface.
- To implement Runnable, a class need only implement a single method called **run( )**,

```
public void run( )
```


- You will define the code that constitutes the new thread inside ***run()*** method.
- It is important to understand that ***run()*** can call other methods, use other classes, and declare variables, just like the main thread can.
- 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);
```

- Here ***threadOb*** is an instance of a class that implements the Runnable interface and the name of the new thread is specified by ***threadName***.

- After the new thread is created, it will not start running until you call its **start( )** method, which is declared within Thread.

```
void start( );
```



```
class NewThread implements Runnable {
    Thread t;
    NewThread() { //constructor
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);
// Let the thread sleep for a while.
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }}
```

```
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.");

    }
```


Lecture0901

- Check lab manual

Lecture0902

- Check lab manual

Lecture0903

- Check lab manual

Lecture0904

- Check lab manual

Create Thread by Extending Thread

- 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.

```
class NewThread extends Thread {
    NewThread() { //constructor
        super("Demo Thread");
        System.out.println("Child thread: " + this);
        start(); // Start the 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.");
    }
}
```



```
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.");

    }
```

Lecture0905

- Check lab manual

Lecture0906

- Check lab manual

Thread Methods

public void start()

Starts the thread in a separate path of execution, then invokes the **run()** method on this Thread object.

public void run()

If this Thread object was instantiated using a separate Runnable target, the **run()** method is invoked on that Runnable object.

public final void setDaemon(boolean on)

A parameter of true denotes this Thread as a daemon thread. (service)

public final void setName(String name)

Changes the name of the Thread object. There is also a **getName()** method for retrieving the name.

public final void setPriority(int priority)

Sets the priority of this Thread object. The possible values are between 1 and 10.

public final void join(long millisec)

The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.

public void interrupt()

Interrupts this thread, causing it to continue execution if it was blocked for any reason.

public final boolean isAlive()

Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

public static void sleep(long millisec)

Causes the currently running thread to block for at least the specified number of milliseconds

public static Thread currentThread()

Returns a reference to the currently running thread, which is the thread that invokes this method.

Lecture0907

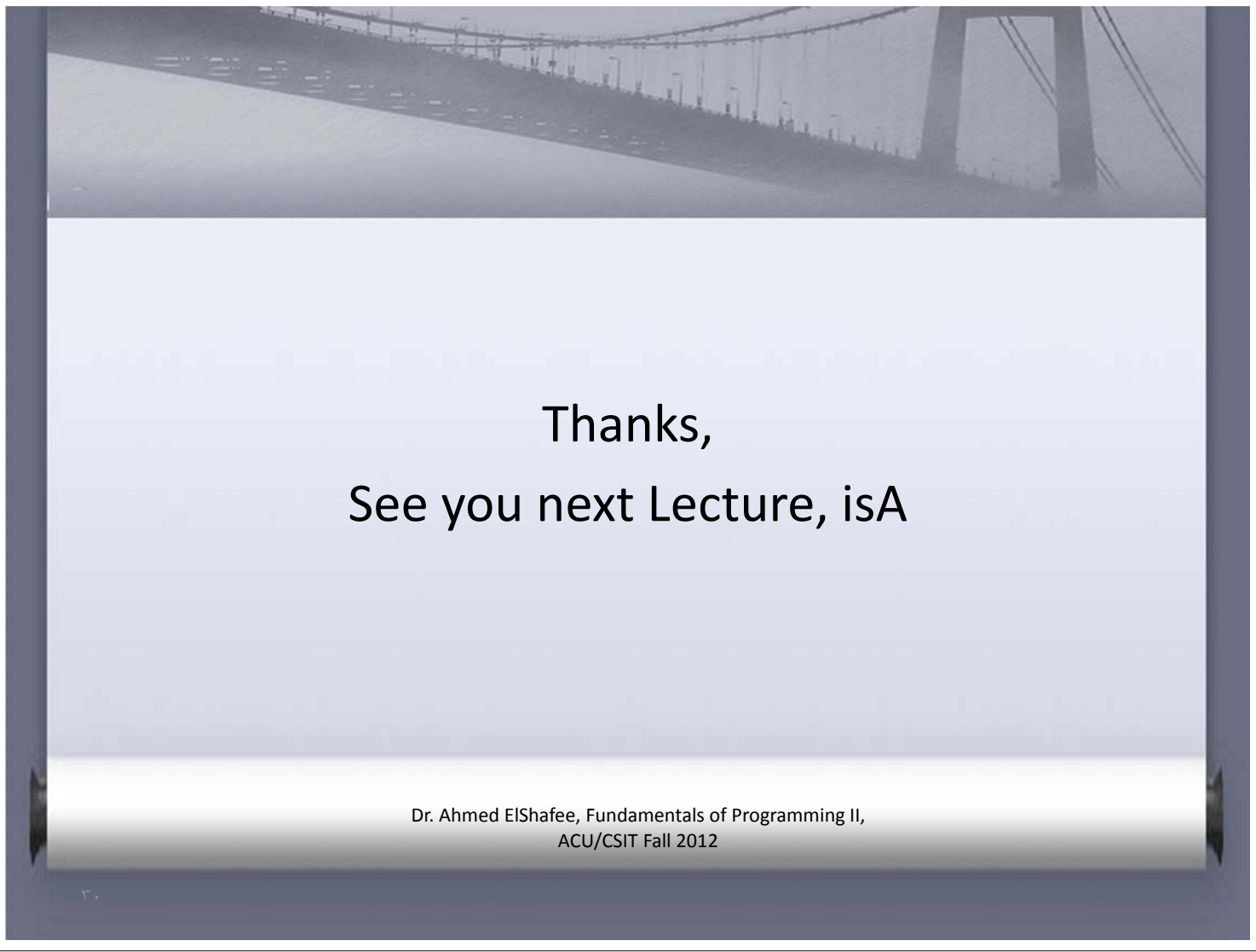
- Check lab manual

Lecture0908

- Check lab manual

Lecture0909

- Check lab manual



Thanks,

See you next Lecture, isA