Design Patterns

MSc in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

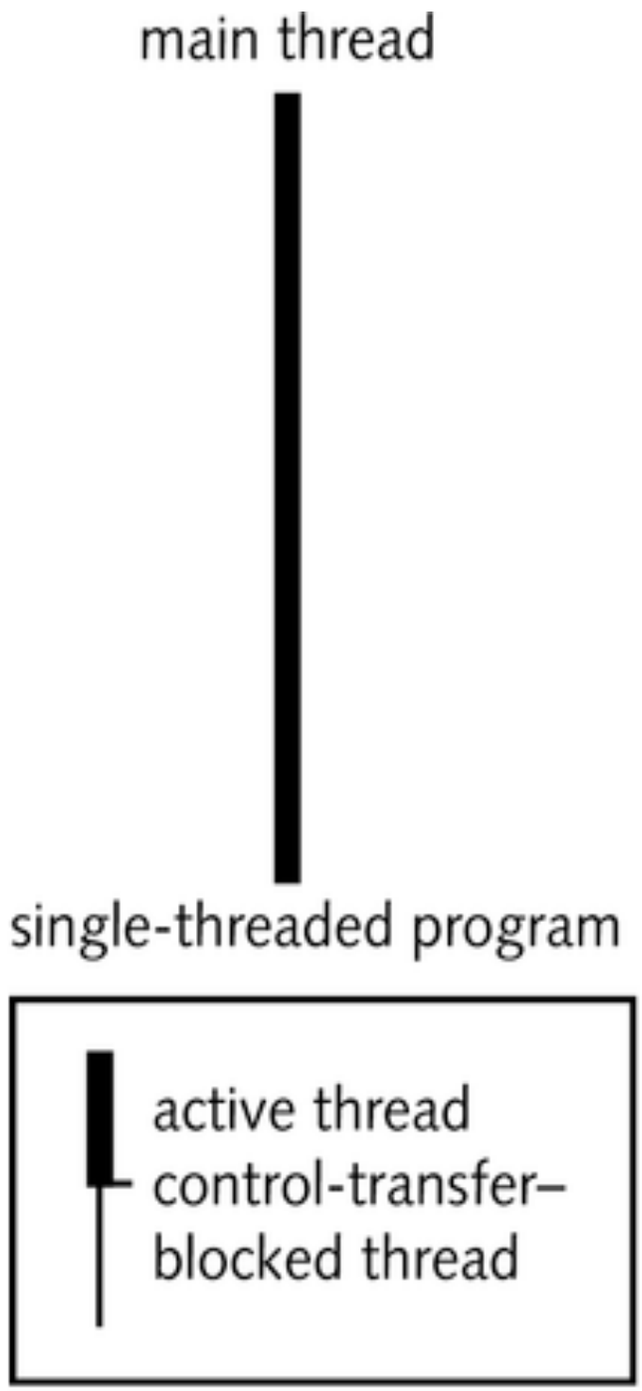
http://elearning.wit.ie

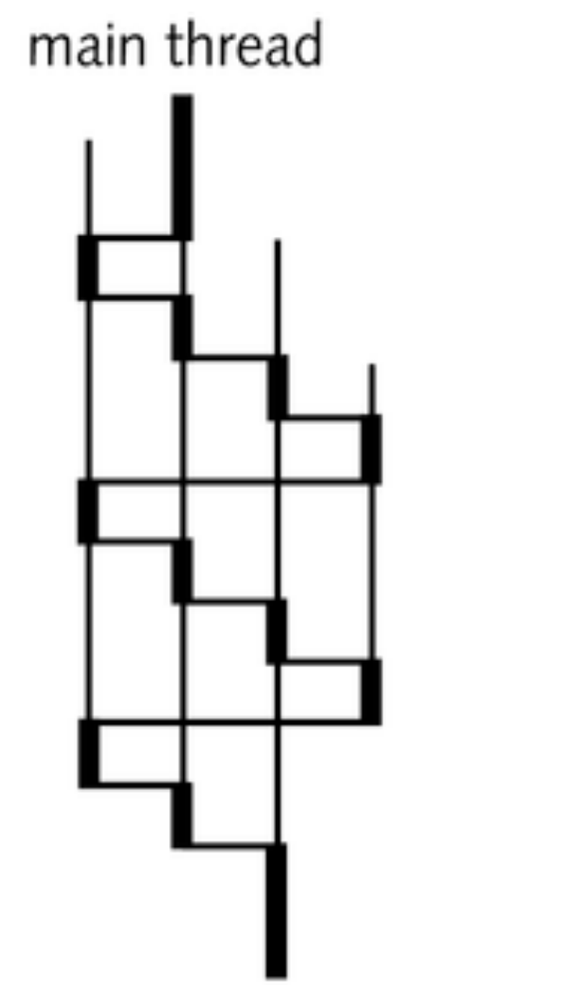
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Threads

Single-Threaded and Multithreaded Programs





The Lifecycle of a Thread

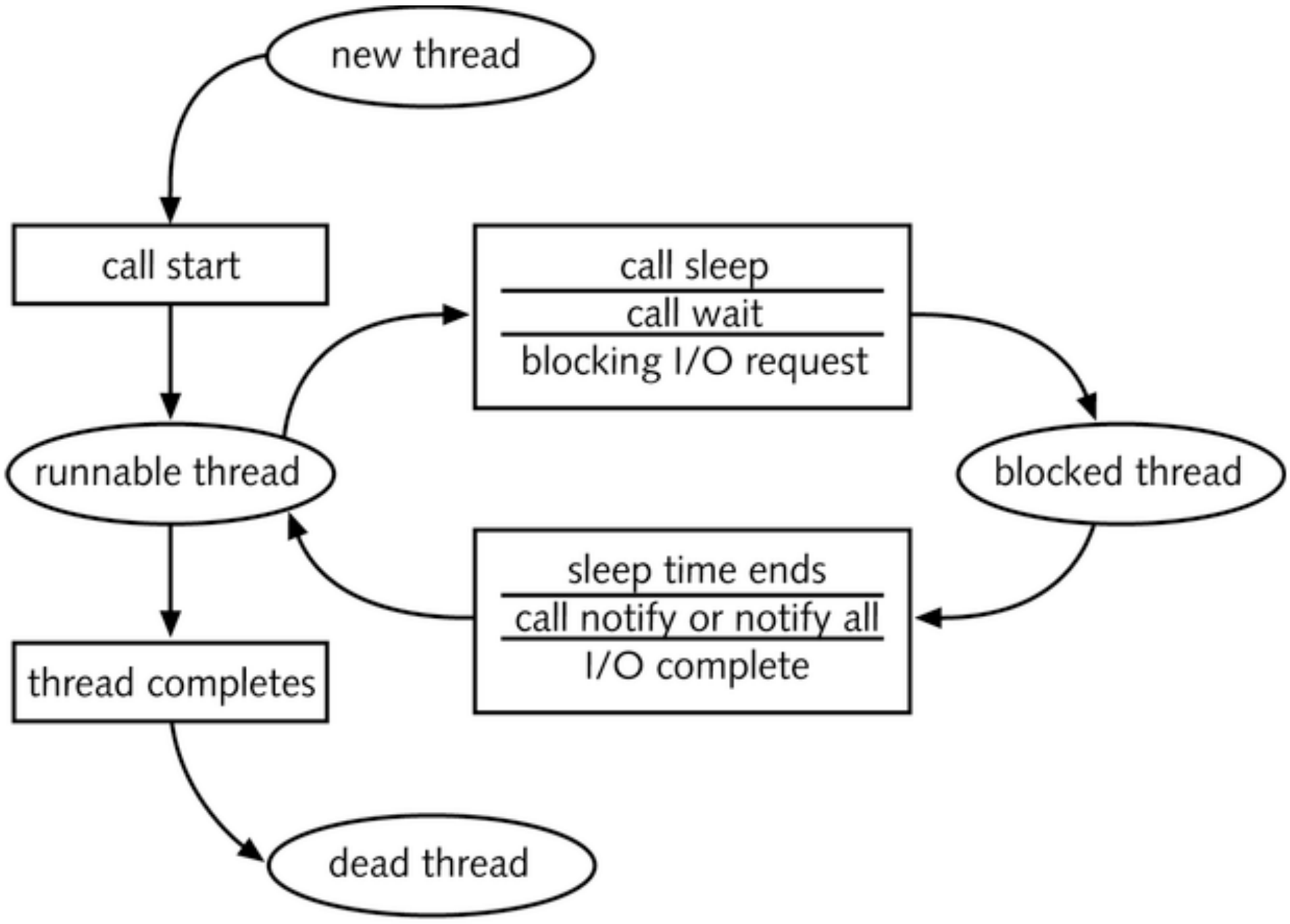
- Threads start running, pause, resume, and finally stop dynamically during program execution

- The states of all threads except the thread that the JVM is currently executing cannot change

- By default, the JVM tries to distribute control equally to all threads

- Threads with high priority values tend to preempt lower priority threads

State Transition Diagram of a Thread



Running a Thread – Method 1

Steps:

1: Implement a class that derives from Thread class

2: Place the code for task into the `run` method of class

```
public class MyThread extends Thread
{
   public void run()
   {
      // Task statements go here
      . . .
   }
}
```

Running a Thread

3. Create an object of class

```
MyThread t = new MyThread();
```

4. Call the `start` method to start the thread.

```
t.start();
```

Running a Thread – Method 2

1: Implement a class that implements the `Runnable` interface

```
public interface Runnable
{
   void run();
}
```

2: Place the code for task into the `run` method of class

```
public class MyRunnable implements Runnable
{
   public void run()
   {
      // Task statements go here
      . . .
   }
}
```

Running a Thread

3. Create an object of class

```
Runnable r = new MyRunnable();
```

4. Construct a `Thread` object for the runnable object.

```
Thread t = new Thread(r);
```

5. Call the `start` method to start the thread.

```
t.start();
```

Example

- A program to print a time stamp and "Hello World" once a second for ten seconds

```
Thu Dec 28 23:12:03 PST 2004 Hello, World!
Thu Dec 28 23:12:04 PST 2004 Hello, World!
Thu Dec 28 23:12:05 PST 2004 Hello, World!
Thu Dec 28 23:12:06 PST 2004 Hello, World!
Thu Dec 28 23:12:07 PST 2004 Hello, World!
Thu Dec 28 23:12:08 PST 2004 Hello, World!
Thu Dec 28 23:12:09 PST 2004 Hello, World!
Thu Dec 28 23:12:10 PST 2004 Hello, World!
Thu Dec 28 23:12:11 PST 2004 Hello, World!
Thu Dec 28 23:12:12 PST 2004 Hello, World!
```

`GreetingRunnable` Outline

```
public class GreetingRunnable implements Runnable
{
   private String greeting;

   public GreetingRunnable(String aGreeting)
   {
      greeting = aGreeting;
   }

   public void run()
   {
      // Task statements go here
   }
}
```

Thread Action for `GreetingRunnable`

- Print a time stamp
- Print the greeting
- Wait a second

- We can get the date and time by constructing a Date object

```
Date now = new Date();
```

Thread Action for `GreetingRunnable`

- To wait a second, use the sleep method of the Thread class

```
sleep(milliseconds)
```

```
Date now = new Date();
```

- A sleeping thread can generate an **`InterruptedException`**
  - Catch the exception
  - Terminate the thread

Running Threads

- `sleep` puts current thread to sleep for given number of milliseconds

```
Thread.sleep(milliseconds)
```

- When a thread is interrupted, most common response is to terminate `run`

Generic `run` Method

```
public void run()
{
    try
    {
        Task statements
    }
    catch (InterruptedException exception)
    {
    }
    Clean up, if necessary
}
```

GreetingRunnable.java

```
public class GreetingRunnable implements Runnable
{
  private static final int REPETITIONS = 10;
  private static final int DELAY = 1000;
  private String greeting;

  public GreetingRunnable(String aGreeting)
  {
    greeting = aGreeting;
  }

  public void run()
  {
    try
    {
      for (int i = 1; i <= REPETITIONS; i++)
      {
        Date now = new Date();
        System.out.println(now + " " + greeting);
        Thread.sleep(DELAY);
      }
    }
    catch (InterruptedException exception)
    {
    }
  }
}
```

To Start the Thread

- Construct an object of your runnable class

```
Runnable t = new GreetingRunnable("Hello World");
```

- Then construct a thread and call the `start` method.

```
Thread t = new Thread(r);
t.start();
```

GreetingThreadTester

```
public class GreetingThreadTester
{
  public static void main(String[] args)
  {
    GreetingRunnable r1 = new GreetingRunnable("Hello, World!");
    GreetingRunnable r2 = new GreetingRunnable("Goodbye, World!");
    Thread t1 = new Thread(r1);
    Thread t2 = new Thread(r2);
    t1.start();
    t2.start();
  }
}
```

Output

```
Thu Dec 28 23:12:03 PST 2004 Hello, World!
Thu Dec 28 23:12:03 PST 2004 Goodbye, World!
Thu Dec 28 23:12:04 PST 2004 Hello, World!
Thu Dec 28 23:12:05 PST 2004 Hello, World!
Thu Dec 28 23:12:04 PST 2004 Goodbye, World!
Thu Dec 28 23:12:05 PST 2004 Goodbye, World!
Thu Dec 28 23:12:06 PST 2004 Hello, World!
Thu Dec 28 23:12:06 PST 2004 Goodbye, World!
Thu Dec 28 23:12:07 PST 2004 Hello, World!
Thu Dec 28 23:12:07 PST 2004 Goodbye, World!
Thu Dec 28 23:12:08 PST 2004 Hello, World!
Thu Dec 28 23:12:08 PST 2004 Goodbye, World!
Thu Dec 28 23:12:09 PST 2004 Hello, World!
Thu Dec 28 23:12:09 PST 2004 Goodbye, World!
Thu Dec 28 23:12:10 PST 2004 Hello, World!
Thu Dec 28 23:12:10 PST 2004 Goodbye, World!
Thu Dec 28 23:12:11 PST 2004 Goodbye, World!
Thu Dec 28 23:12:11 PST 2004 Hello, World!
Thu Dec 28 23:12:12 PST 2004 Goodbye, World!
Thu Dec 28 23:12:12 PST 2004 Hello, World!
```

Thread Scheduler

- The thread scheduler runs each thread for a short amount of time (a *time slice*)

- Then the scheduler activates another thread

- There will always be slight variations in running times especially when calling operating system services (e.g. input and output)

- There is no guarantee about the order in which threads are executed

Terminating Threads

- A thread terminates when its `run` method terminates
- Do not terminate a thread using the deprecated `stop` method
- Instead, notify a thread that it should terminate

```
t.interrupt();
```

- `interrupt` does not cause the thread to terminate–it sets a boolean field in the thread data structure

Terminating Threads

- The run method should check occasionally whether it has been interrupted
  - Use the `interrupted` method
  - An interrupted thread should release resources, clean up, and exit

```
public void run()
{
   for (int i = 1;
         i <= REPETITIONS && !Thread.interrupted(); i++)
   {
      Do work
   }
   Clean up
}
```

Terminating Threads

- The sleep method throws an InterruptedException when a sleeping thread is interrupted
  - Catch the exception
  - Terminate the thread

```
public void run()
{
    try
    {
       for (int i = 1; i <= REPETITIONS; i++)
       {
          Do work
       }
    }
    catch (InterruptedException exception)
    {
    }
    Clean up
}
```