

Multithreading

"Understanding concurrency, threads, creating and executing threads"
Advanced in Programming

Shakirullah Waseeb
shakir.waseeb@gmail.com

Nangarhar University

March 29, 2017



Agenda

- 1 Concurrency Overview
 - Understanding Concurrency and Parallelism
- 2 Java Multithreading
 - Java Threads Model
- 3 The Main Thread
- 4 Questions and Discussion



Agenda

- 1 Concurrency Overview
 - Understanding Concurrency and Parallelism
- 2 Java Multithreading
 - Java Threads Model
- 3 The Main Thread
- 4 Questions and Discussion



Overview

- **Concurrency:** tasks that all are making *progress at once* (by rapidly switching between them)
- Two tasks operating in parallel, means they are executing simultaneously
- In this sense, parallelism is a subset of concurrency
- Concurrent programming has several usages consider the scenario of online audio or video streaming
 - One thread to download the stream
 - Another thread to play the downloaded stream
 - These activities proceed concurrently
- Two distinct types of multitasking are: *process-based* and *thread-based*



Agenda

- 1 Concurrency Overview
 - Understanding Concurrency and Parallelism
- 2 Java Multithreading
 - Java Threads Model
- 3 The Main Thread
- 4 Questions and Discussion



Introduction

- Multithreading enables us to write efficient programs that utilize the CPU usage, by keeping idle time at minimum
- Java uses threads to enable the entire environment to be asynchronous, hence prevent the CPU's waste cycles
- Single-thread systems used approach called *event loop* with *polling*
- Java eliminates this main *loop/polling* mechanisms by multithreading; one thread can pause without stoping other parts of the program



States and Life cycle of threads

- At any time a thread is said to be in one of several states as shown in Figure 2 (UML state diagram)

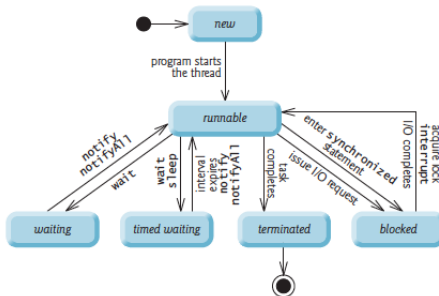


Figure: Thread life cycle [1, Page 960]



Threads priorities

- To each thread a priority is assigned, determining how this thread execution should be treated with respect to others
- Priorities are integers specifying the relative priority of one thread to another
- Thread's priority is used to decide when to **switch** from one **running** thread to the next, called *context switch*
 - A thread can voluntarily release control
 - A thread can be preempted by a higher-priority thread



Synchronization

- As multithreading introduces asynchronous behavior to program, there are situation where synchronicity is required
- In situation where threads share a common resource and perform operations on given source simultaneously
- This is required to avoid conflicts, e.g. one thread should not be allowed to write data while another thread is in the middle of reading it
- Control mechanism called *monitor* is used to model interprocess synchronization
- In Java each object has its own implicit *monitor* that is automatically entered when one of the object's synchronized methods is called
- Once a thread is inside a *synchronized method*, no other thread can call any other synchronized method on the *same object*



Threads inter-communication

- Java provides easy way for two or more threads to communicate
- This can be done via calls to predefined methods that all objects have
- Java's messaging system allows a thread to enter a synchronized method on an object, and then wait there until some other thread explicitly notifies it to come out



Thread class and the Runnable interface

- Java's multithreading system is built upon the Thread class, its methods, and its companion interface, Runnable
- To create a new thread, your program will either extend Thread or implement the Runnable interface
- The Thread class defines several methods that help manage threads listed below

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.

Figure: Thread life cycle [3, Page 277]



The main thread

- The **main** thread starts running immediately when a java program starts up
- Important for two reasons
 - ① It is the main thread from which **other threads will be spawned**
 - ② Often it must be the **last thread to finish execution** because it performs **various shutdown actions**
- To get reference of current executing thread we can use following static method of class **Thread** as:
Thread thread = Thread.currentThread();



Getting reference of Main Thread

Accessing Main Thread

```
//  
public class MainThreadDemo{  
    public static void Main(String args[]) {  
        Thread thread = Thread.currentThread();  
        System.out.println("Current running thread : " + thread);  
        thread.setName("Demo Main Thread");  
        System.out.println("After name changed : " + thread);  
    }  
}
```



Next Topics

- Creating threads by *implementing Runnable interface*
- Creating threads *extending Thread class*
- Creating *multiple threads*



Your Turn: Time to hear from you!



1

¹<https://fensafitters.files.wordpress.com/2013/07/3d095.jpg>



References

-  P.J. Deitel, H.M. Deitel
Java How to program, 10th Edition .
Prentice Hall, 2015.
-  P.J. Deitel, H.M. Deitel
Java How to program, 9th Edition .
Prentice Hall, 2012.
-  Herbert Schildt
The complete reference Java2, 5th Edition .
McGraw-Hill/Osborne, 2002.

