Software Practice 1 - Multithreading

- What is the thread
- Life cycle of thread
- How to create thread
- Thread method
- Lab practice

Prof. Joonwon Lee

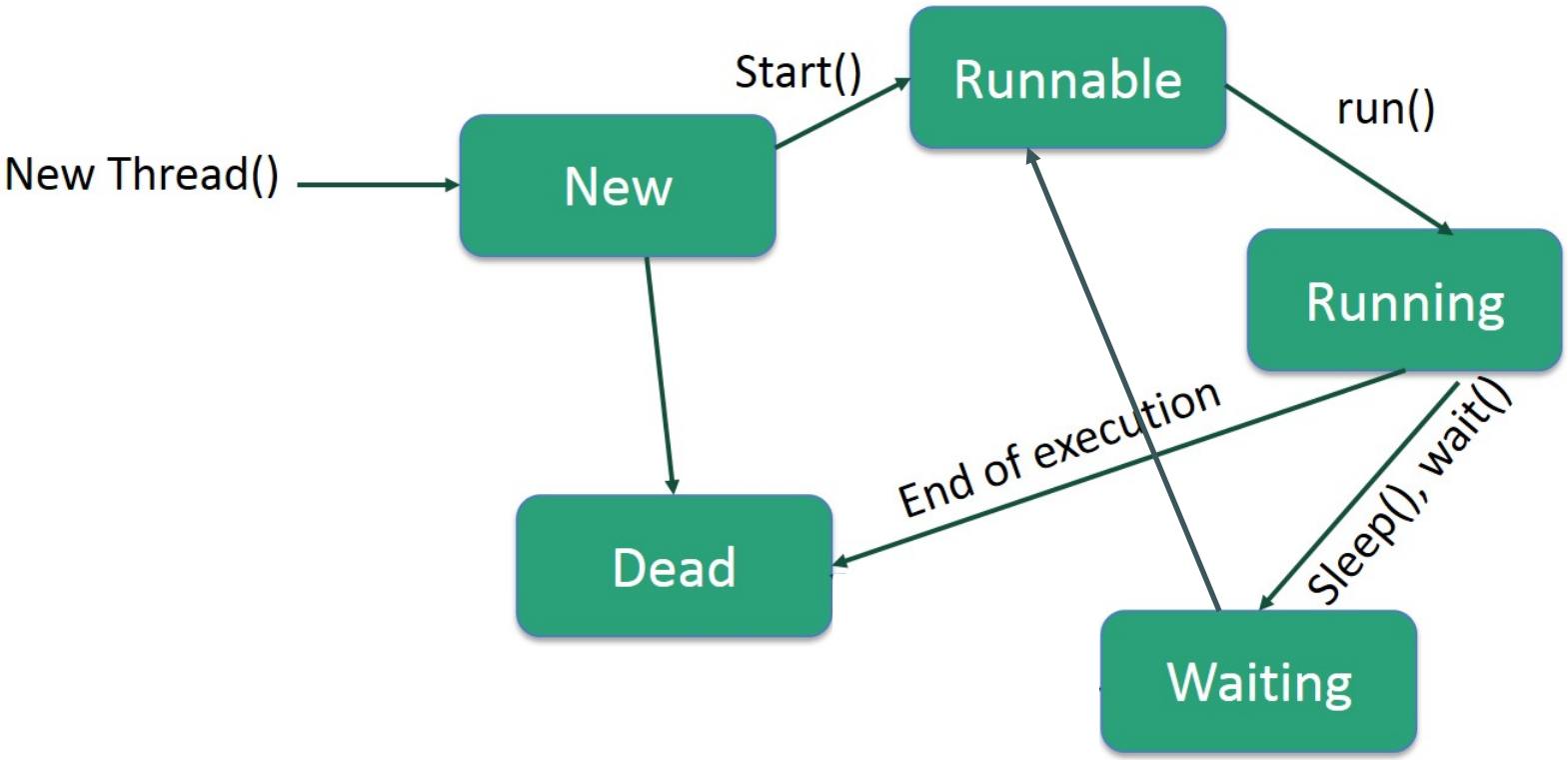
T.A. Jaehyun Song (42)
Jongseok Kim

T.A. Sujin Oh (43)
Junseong Lee

Thread

- The smallest sequence of programmed instructions that can be managed independently by a scheduler of OS
- The implementation of threads and process differs between operating systems, but in most cases a thread is a component of a process

Life Cycle of Thread



Life Cycle of Thread

- **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 (Dead)** − 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 thread 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 are very much platform dependent.

Create a Thread
by Implementing a Runnable Interface

- If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface
- Step 1 : implement a run() method
  - Public void run()
- Step 2 : instantiate a Thread object
  - Thread(Runnable threadObj, String threadName);
- Step 3 : Call start()
  - void start();

Example

```
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 (cont'd)

```
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();
  }
}
```

Create a Thread by Extending a Thread Class

- create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.
- Step 1 : override run() method
  - public void run()
- Step 2 : call start() method
  - void start();

Example

```
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 ();
    }
  }
}
```

Example (cont'd)

```
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();
  }
}
```

Thread Methods

Sr.No.	Method & Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this T hread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**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.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**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.

Thread Static Method

Sr.No.	Method & Description
1	**public static void yield()** Causes the currently running thread to yield to any other threads of the same priority t hat are waiting to be scheduled.
2	**public static void sleep(long millisec)** Causes the currently running thread to block for at least the specified number of millis econds.
3	**public static boolean holdsLock(Object x)** Returns true if the current thread holds the lock on the given Object.
4	**public static Thread currentThread()** Returns a reference to the currently running thread, which is the thread that invokes th is method.
5	**public static void dumpStack()** Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Major Java Multithreading Concepts

- While doing Multithreading programming in Java, you would need to have the following concepts very handy −
- What is thread synchronization?
- Handling interthread communication
- Handling thread deadlock
- Major thread operations

Synchronized

- Synchronized method

```
public synchronized void set() {
        …
}
```

- Synchronized block

```
public void method() {
        synchronized (object) {
                …
        }
}
```

Wait & Notify

- **Wait()**
  - The wait() method causes the current thread to wait indefinitely until another thread either invokes notify () for this object or notifyAll().

- **Notify()**
  - The notify() method is used for waking up thread th at are waiting for an access to this object's monitor.

[Lab – Practice]

- Producer - Consumer Problem

- There are two process(thread) producer and consumer, who are share common fixed-size buffer.
- The producer's job is to generate data and put it into buffer
- At the same time, consumer is consuming the data, one piece at a time.

Producer - Consumer

- **Buffer size == 5**
- **Produce (consume) 10 data. (0 to 9)**

- **Warning!**
  - Cannot consume the data if buffer is empty
  - Cannot produce the data if buffer is full

- **Hint**
  - Wait and Notify
  - BlockingQueue

Example of Output

Output:

Start: Producer
Start: Consumer
Produced: 0
Produced: 1
Consumed: 0
Produced: 2
Consumed: 1
...
Consumed: 7
Produced: 9
Consumed: 8
Consumed: 9

[Submit]

- **Upload to i-Campus**
  - Compress your Producer.java & Consumer.java to zip file
  - File name: studentID_lab8.zip

- **Due date**
  - Today 23:59:59
    - Class 42 (05/21 Monday)
    - Penalty: -10% of each lab score per one day