# Unit V
# Multithreaded Programming & Garbage Collection

Tutor: Pema Galey

## Learning Outcomes

In this session, you will learn about:

- Multithreaded Programming
  - Runnable interface
  - Thread Priorities
- Garbage Collection in Java

## Implementing Runnable Interface

- When a Java program needs to inherit from a class other than the Thread class, you need to implement the `Runnable` interface.
- The following code shows the syntax of the `run()` method:

```
public void run()
```

- The `run()` method contains the code that defines the new thread.
- When the `run()` method is called, another thread starts executing concurrently with the main thread in the program.
- The class that implements the Runnable interface creates an instance of the `Thread` class.

## Creating Thread by Implementing Runnable Interface

- The new thread object starts executing by calling the `start()` method.
- The `start()` method is declared in the `Thread` class and when the `start()` method executes, the `run()` method is called.

## Creating Thread by Extending Thread Class

- You can use the following code to create a thread by extending the Thread class:

```
class ThreadDemo extends Thread {
   ThreadDemo() {
         super("ChildThread");
         System.out.println("ChildThread:" + this);
         start();
   }
}
```

## Creating Thread by Implenting Runnable Interface

- You can use the following code to create a thread by implementing Runnable interface:

```
class NewThreadClass implements Runnable{
    String ThreadName;
    NewThreadClass(String name) {
        ThreadName = name;
        Thread t = new Thread(this, ThreadName);
        System.out.println("Thread created: " + t);
        t.start();
    }
}
```

## Creating Threads

- Using the **isAlive()** method:
  - The isAlive() method is used to check the existence of a thread.
  - The following code shows the syntax of the isAlive()method:

```
public final boolean isAlive()
```

- Using the **join()** method:
  - The join() method waits until the thread on which it is called, terminates.
  - In addition, the join()method enables you to specify a maximum amount of time that you need to wait for the specified thread to terminate.
  - The following code shows how to declare the join()method:

```
public final void join() throws InterruptedException
```

## Thread Priorities

- Thread priorities are integer values in the range of ***1 to 10*** that specify the priority of one thread with respect to the priority of another thread.
- If the processor encounters another thread with a higher priority, the current thread is pushed back; and the thread with the higher priority is executed.
- The next thread of a lower priority starts executing if a higher priority thread stops or becomes not runnable.
- A thread is pushed back in the queue by another thread if it is waiting for an I/O operation.
- A thread can also be pushed back in the queue when the time for which the sleep()method was called on another higher priority thread is over.

## Setting the Thread Priorities

- Once a thread is created, its priority is set by using the `setPriority()` method declared in the Thread class.
- The following code shows how to declare the setPriority()method:

```
public final void setPriority(int newPriority)
```

- The thread can be set to a default priority by specifying the `NORM_PRIORITY` constant in the `setPriority()` method.
  - `MAX_PRIORITY` is the highest priority (=10) that a thread can have,
  - `MIN_PRIORITY` is the lowest priority (=1) a thread can have; and
  - `NORM_PRIORITY` is the default priority (=5) set for a thread.

## Synchronizing Threads

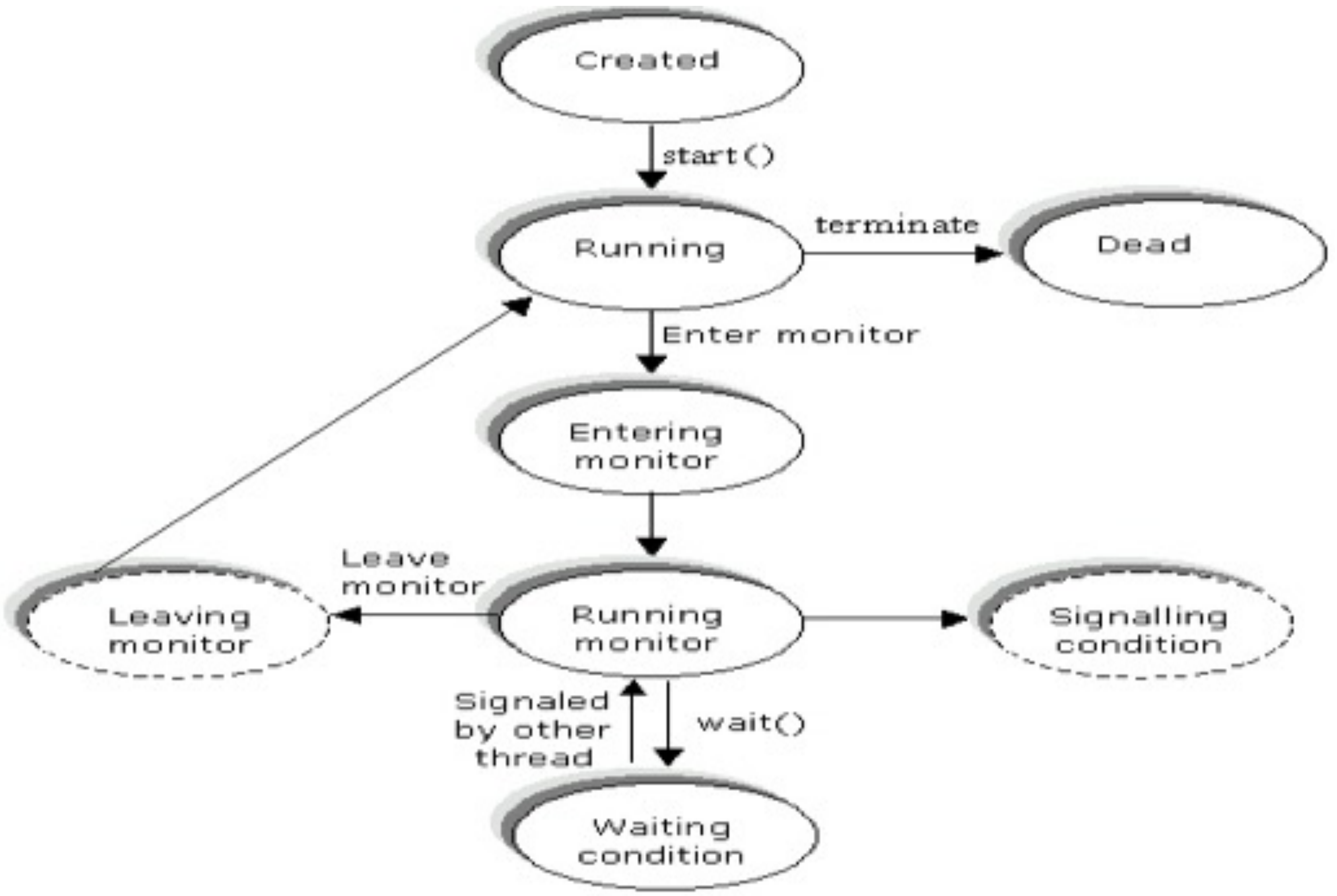
- The synchronization of threads ensures that if two or more threads need to access a shared resource then that resource is used by only one thread at a time.
- You can synchronize your code by using the **synchronized** keyword.
- Synchronization is based on the concept of monitor.
- A monitor, also known as a semaphore, is an object that is used as a mutually exclusive lock.
- All objects and classes are associated with a monitor and only one thread can own a monitor at a given time.
- To enter an object's monitor, you need to call a method that has been modified with the synchronized keyword.

## Synchronizing Threads

- When a thread acquires a lock, it is said to have entered the monitor.
- During the execution of a synchronized method, the object is locked so that no other synchronized method can be invoked.
- The monitor is automatically released when the method completes its execution.
- The monitor can also be released when the synchronized method executes the `wait()` method.
- When a thread calls the `wait()` method, it temporarily releases the locks that it holds.
- In addition, the thread stops running and is added to the list of waiting threads for that object.

## Synchronizing Threads

- The following figure shows how synchronization is maintained among threads.



## Synchronizing Threads

The `synchronized` statement:

- The synchronized statement is used where the synchronization methods are not used in a class and you do not have access to the source code.
- You can synchronize the access to an object of this class by placing the calls to the methods defined by it inside a synchronized block.
- The following code shows how to use the synchronized block:

```
synchronized(obj){
   /* statements to be synchronized*/
}
```

## Communication Access Threads

- Multithreading eliminates the concept of polling by a mechanism known as interprocess communication.
- A thread may notify another thread that the task has been completed.
- This communication between threads is known as inter-threaded communication.
- The various methods used in inter-threaded communication are:
  - `wait():` Informs the current thread to leave its control over the monitor and sleep for a specified time until another thread calls the notify() method. The following code shows how to declare the wait() method:

```
public final void wait() throw InterruptedException;
```

## Communication Access Threads

- **`notify():`** Wakes up a single thread that is waiting for the monitor of the object being executed. If multiple threads are waiting, one of them is chosen randomly. The following code shows how to declare the **`notify()`** method:

```
public final void notify()
```

- **`notifyAll():`** Wakes up all the threads that are waiting for the monitor of the object.

## Creating Daemon Thread

- A daemon thread is a thread that runs in the background of a program and provides services to other threads.
- The `setDaemon()` method is used to convert a thread to a daemon thread.
- The following code shows how to set a thread, Thread1 as a daemon thread:

```
Thread1.setDaemon(true);
```

- The `isDaemon()` method is used to determine if a thread is a daemon thread.
- The following code shows how to declare the isDaemon()method:

```
boolean b = Thread1.isDaemon();
```

## Garbage Collection

- **Garbage collection** refers to automatically destroying the objects created and releasing their memory for future reallocation.
- The various activities involved in garbage collection are:
  - Monitoring the objects used by a program and determining when they are not in use
  - Destroying objects that are no more in use and reclaiming their resources, such as memory space
- **JVM** performs garbage collection when it needs more memory to continue execution of programs.
- You cannot explicitly reclaim the memory of a specific dereferenced object.

## Garbage Collection

- The Runtime class in Java encapsulates the JRE.
- The various methods of the Runtime class used in memory management are:
  - `static Runtime.getRuntime():` Returns the reference of the current runtime object.
  - `void gc()`: Invokes garbage collection.
  - `long totalMemory():` Returns the total number of bytes of memory available in JVM.

## Garbage Collection Implementation

- You can explicitly run the garbage collector in Java to collect information regarding how large the object heap is or to determine the number of objects of a certain type that can be instantiated.
- The **`totalMemory()`** method returns the total memory in the JVM, and the **`freeMemory()`** method returns the amount of memory free in the JVM.
- You can run the garbage collector explicitly by calling the **`gc()`** method.

## Garbage Collection

- To run the garbage collector explicitly, you need to perform the following steps:
  - Create an object of the Runtime class. The following code shows how to create an object of the Runtime class:

```
Runtime r = Runtime.getRuntime();
```

- Invoke the `gc()` method of the `Runtime` class to request garbage collection. The following code shows how to invoke the garbage collector:

```
r.gc();
```

## Thank you!