Multithreading in Java

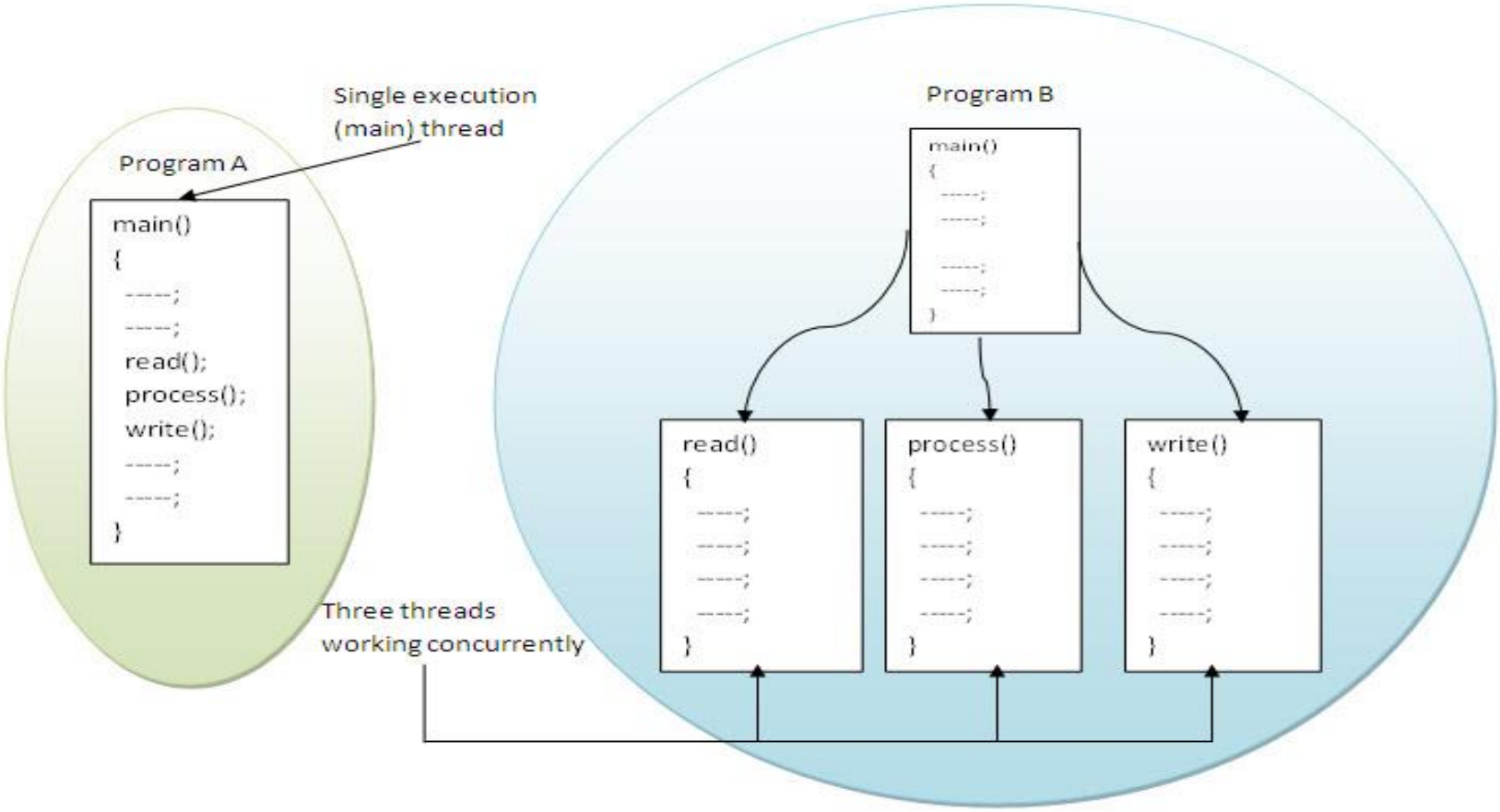
Contents

- Introduction to threads and multi-threading
- Thread Creation
  - Extending Thread class
  - Implementing Runnable interface
- Creating multiple threads
- Thread states
- Thread synchronization
- Thread scheduling and priority

Introduction to Threads and Multi-threading

- Threads are light weight processes.
- A thread is a path of code execution through a program.
- Thread consists of:
  - Its own local variables
  - Program counter
  - Lifetime
- In every Java program, we've at least one thread running which is the "main" thread.
- In a multithreaded environment, you can create more than one thread inside an application.

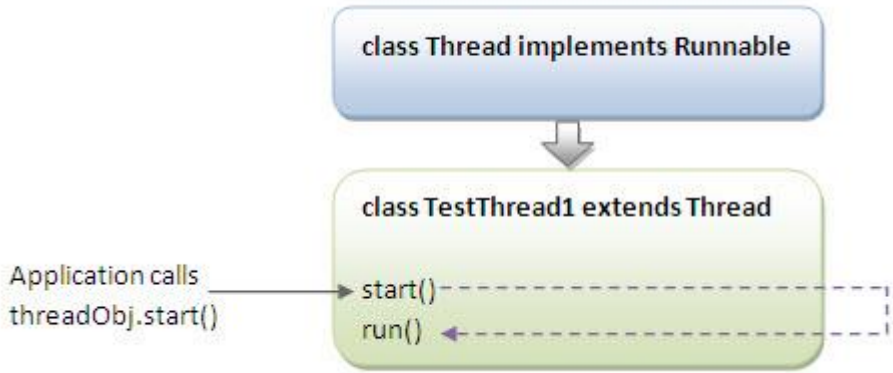
Introduction to Threads and Multi-threading (continued…)



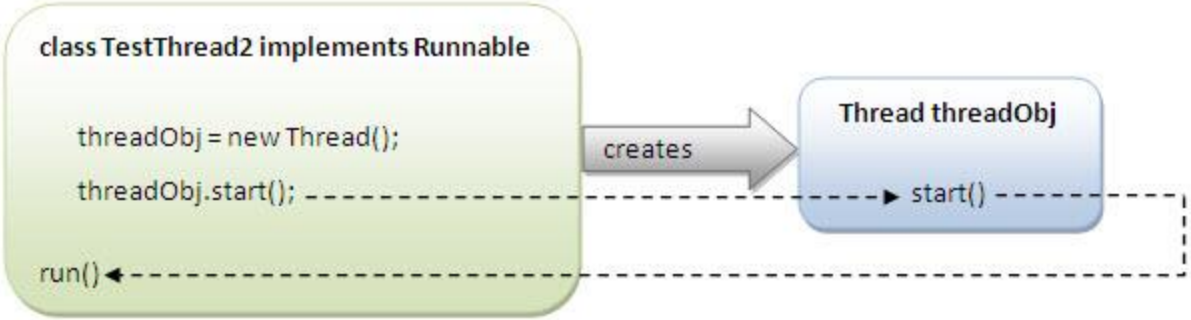
Thread Creation

- Every thread of execution, in Java, begins as an instance of class Thread.
- Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
- Threads can be created in any of the two known ways:
  - By extending the Thread class.
  - By implementing the Runnable interface.
- Runnable interface contains a single method run()
- You need to define run() method while creating the thread in any of the two ways.

Thread Creation (continued…)



Thread creation by extending java.lang.Thread class



Thread creation by implementing Runnable interface

Thread Creation Extending Thread Class

- Thread can be created by extending the java.lang.Thread class
- It's considered as the easiest method among the two.
- It restricts the programmer from extending any other class.
- Four steps should be followed to create thread in this way:
  - Create a class extending java.lang.Thread class.
  - Override java.lang.Thread's run() method in your class.
  - Instantiate the thread by instantiating your class.
  - Invoke the start() method that your class inherited from the Thread class.

Thread Creation Extending Thread Class cont..

```
class A extends Thread{
public void run(){     //Method of Thread class
int i;
for (i=0; i<=10; i++)
System.out.println("Thread A Iteration:" +i);
}
}
public class ThreadExample2{
public static void main(String[] args){
A t1=new A();
t1.start();   //Method of Thread class to begin the execution
}
}
```

Thread Creation Extending Thread Class

C:\windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\shiv8>cd OneDrive\Desktop\Java Programs

C:\Users\shiv8\OneDrive\Desktop\Java Programs>javac ThreadExample2.java

C:\Users\shiv8\OneDrive\Desktop\Java Programs>java ThreadExample2
Thread A Iteration:0
Thread A Iteration:1
Thread A Iteration:2
Thread A Iteration:3
Thread A Iteration:4
Thread A Iteration:5
Thread A Iteration:6
Thread A Iteration:7
Thread A Iteration:8
Thread A Iteration:9
Thread A Iteration:10

C:\Users\shiv8\OneDrive\Desktop\Java Programs>
```

Thread Creation Implementing Runnable Interface

- Thread can be created by implementing java.lang.Runnable interface.
- This provides the programmer liberty to extend any class.
- Four steps needs to followed to create thread in this way:
  - Create a class implementing the Runnable interface.
  - Implement the run() method of Runnable interface your class.
  - Create an object of the Thread class by passing an instance of the above created class.
  - Invoke the start() method on your Thread object.

```
class A implements Runnable{
public void run(){
int i;
for (i=0; i<=10; i++)
System.out.println("Thread A Iteration:" +i);  }    }
class B implements Runnable{
public void run(){
int i;
for(i=0; i<=10; i++)
System.out.println("Thread B Iteration:" +i);   }    }

public class ThreadExample{
public static void main(String[] args){
Thread t1=new Thread(new A());
Thread t2=new Thread(new B());
t1.start();
t2.start();   }    }
```

Thread Creation Implementing Runnable Interface contd..

C:\windows\system32\cmd.exe

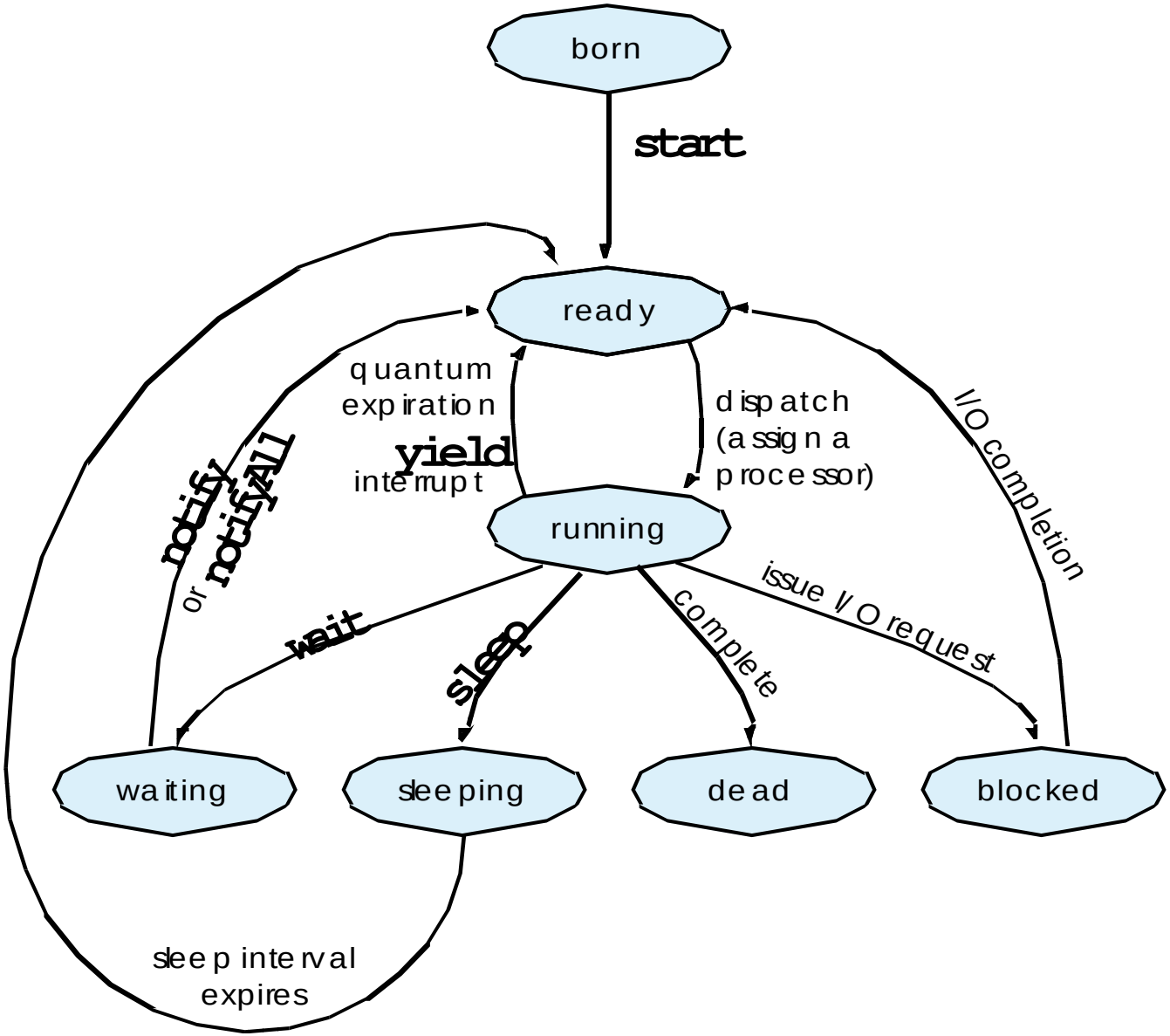
```
C:\Users\shiv8\OneDrive\Desktop\Java Programs>Java ThreadExample
Thread A Iteration:0
Thread B Iteration:0
Thread A Iteration:1
Thread B Iteration:1
Thread B Iteration:2
Thread B Iteration:3
Thread B Iteration:4
Thread B Iteration:5
Thread B Iteration:6
Thread A Iteration:2
Thread B Iteration:7
Thread A Iteration:3
Thread B Iteration:8
Thread A Iteration:4
Thread B Iteration:9
Thread A Iteration:5
Thread B Iteration:10
Thread A Iteration:6
Thread A Iteration:7
Thread A Iteration:8
Thread A Iteration:9
Thread A Iteration:10

C:\Users\shiv8\OneDrive\Desktop\Java Programs>
```

Creating Multiple Threads

- There are situations where you need to same job multiple times or do multiple jobs simultaneously.
- Multiple threads is a good option in such situations.
- CPU cycles are shared among the multiple threads.
- Each thread runs in it’s own call stack.

Thread States: Life Cycle of a Thread



Thread Synchronization

- Multithreading, if not monitored, results in asynchronous behavior to the programs.
- To protect shared resource from asynchronous access, thread should be synchronized using locks.
- Only one thread can hold the lock for the shared resource at a time.
- Other threads need to wait until the current thread releases the lock.
- There are two types of locks in Java:
  - Object Locks and
  - Class Locks
- Synchronized keyword is used protect the piece of code being shared by multiple threads.

Thread Synchronization (continued…)

- Synchronized can be implemented by two ways:
  - Synchronized Methods
  - Synchronized Blocks
- In the object locks, the monitor controls access to a Java object. Object locks do not work for static code.
- Class locks are used to control the access to synchronized static code.
- Synchronized method is used to control access to a method.
- Synchronized block is used to control access to a block of code.

Thread Scheduling and Priority

- Thread scheduler uses scheduling algorithms to decide upon the state of the thread.
- Basically, there are two main algorithms:
  - Preemptive scheduling
  - Time sharing
- Every thread has an associated Thread Priority.
- The priority can be any integer value ranging from 1 to 10.
- Java provides three constants to describe the priority range:
  - MAX_PRIORITY
  - MIN_PRIORITY
  - NORM_PRIORITY