Java threads

Banu Prakash C

Thread basics

- What is a process ?
  - It is a independently running program
- What is a Thread?
  - Threads are independent, concurrent paths of execution through a program, and each thread has its own stack, its own program counter.
  - They share memory, file handles, and other per-process state.
  - Threads are sometimes referred to as *lightweight processes*.
- *A Process supports multiple threads.*

Every Java Program uses threads.

- Every Java program has at least one thread
  - When a Java program starts, the JVM creates the main thread and places the program's main() method on the stack created for that thread.
- The JVM also creates other threads that are mostly invisible to you
  - Example: threads associated with garbage collection, object finalization, and other JVM housekeeping tasks.

  - Refer Code Snippet in next slide.

Main thread information

```
public static void main(String[] args) {
    System.out.println("JVM creates Main Thread.");

    System.out.println("Thread Information :"
            + Thread.currentThread());
```

```
Thread Information :Thread[main,5,main]
```

```
    System.out.println("Is main Thread Alive :  "
            + Thread.currentThread().isAlive());
```

```
Is main Thread Alive :  true
```

```
    /*
     * Daemon threads are background threads.
     * JVM does not wait for Daemon threads to
     * finish its execution.
     * If No Non-Daemon threads exist JVM terminates.
     */
    System.out.println("Is it a Daemon Thread :"
            + Thread.currentThread().isDaemon());

}
```

```
Is it a Daemon Thread :false
```

Why use threads?

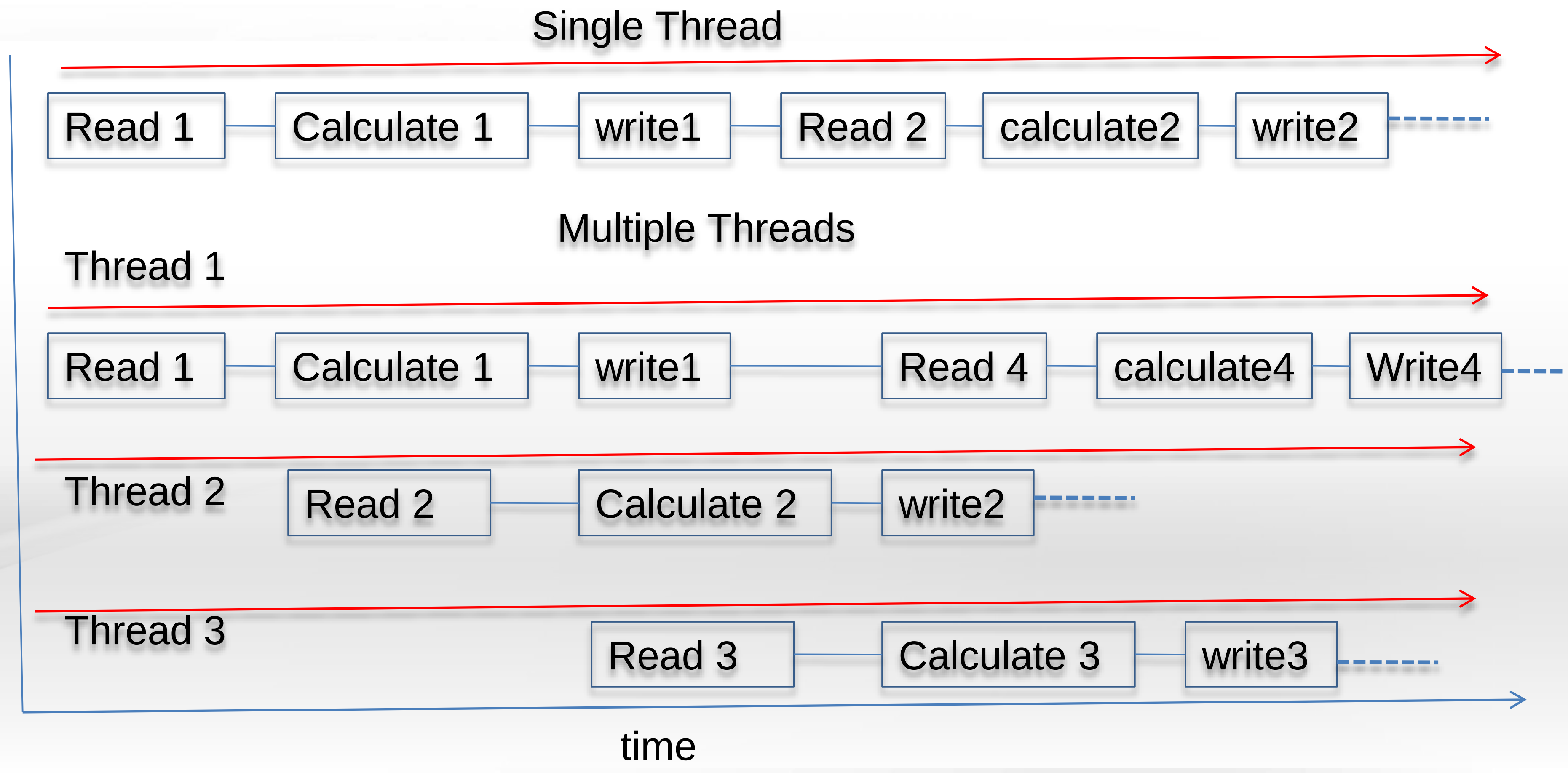
- Take advantage of multiprocessor systems
  - If a program has only one active thread, it can only run on one processor at a time.
  - If a program has multiple active threads, then multiple threads may be scheduled at once to run on different processor.
- Perform asynchronous or background processing
  - Server applications get their input from remote sources, such as sockets. When you read from a socket, if there is no data currently available, the call to read() will block until data is available.
  - If a single-threaded program were to read from the socket, and the entity on the other end of the socket were never to send any data, the program would simply wait forever, and no other processing would get done.

Why use threads?

- More responsive UI
  - Event-driven UI toolkits, have an event thread that processes UI events such as keystrokes and mouse clicks
  - If an event listener were to perform a lengthy task, such as checking spelling in a large document, the event thread would be busy running the spelling checker, and thus would not be able to process additional UI events until the event listener completed. This would make the program appear to freeze, which is disturbing to the user.
  - Example : See How Microsoft Word Application works.

Execution time for Single Thread and Multiple Threads.

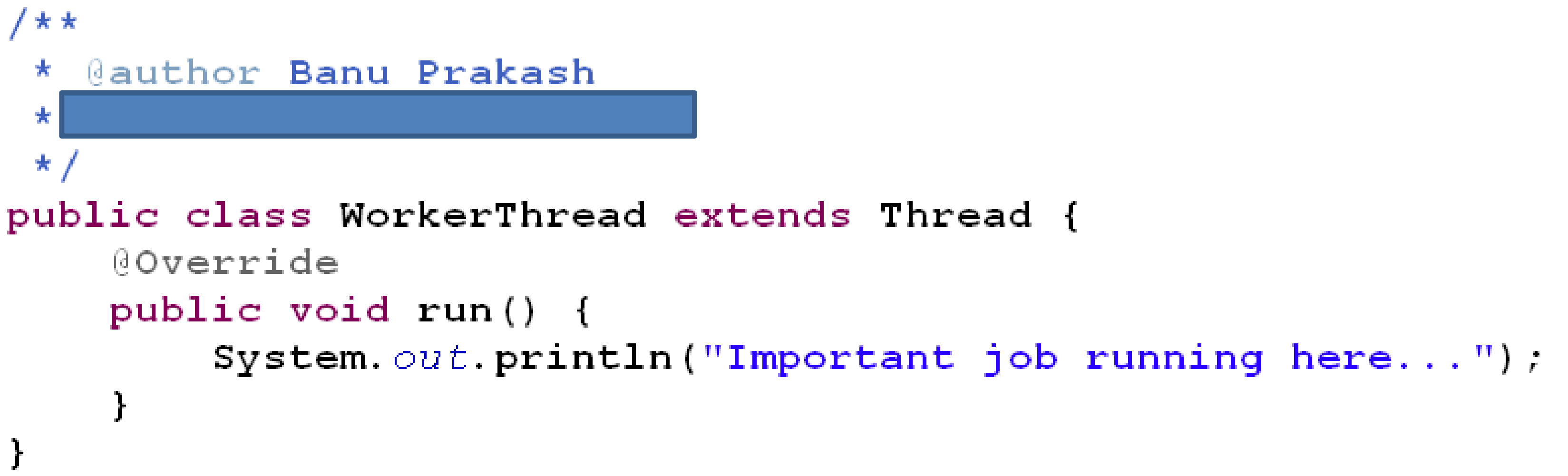
- Consider a very simple program that consists of three activities:
  - Reading a number of blocks of data from a file.
  - Performing some calculation on each block of data.
  - Writing the results of the calculation to another file.



How to define a Thread?

- You can define a thread in java using two ways:

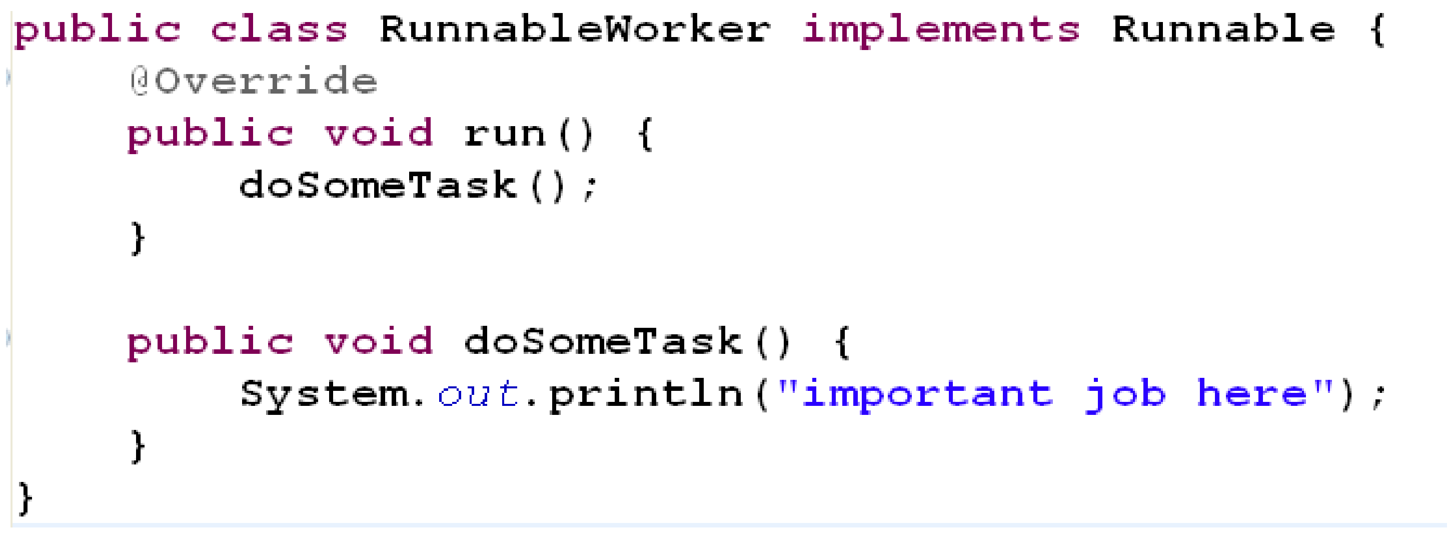
1. Extending java.lang.Thread class
   - Override the run( ) method.
   - The run() method is to every thread, what main() is to Main thread.



```
/**
 * @author Banu Prakash
 * [redacted]
 */
public class WorkerThread extends Thread {
    @Override
    public void run() {
        System.out.println("Important job running here...");
    }
}
```

Runnable interface.

2. Implementing java.lang.Runnable interface.
    - Runnable interface has only one method public void run()
    - Override the run() method



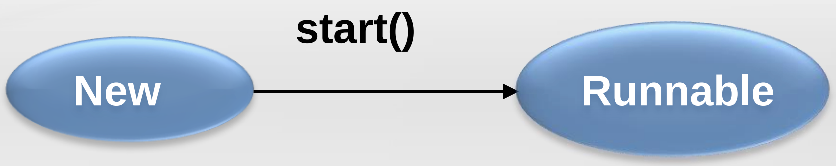
```
public class RunnableWorker implements Runnable {
    @Override
    public void run() {
        doSomeTask();
    }

    public void doSomeTask() {
        System.out.println("important job here");
    }
}
```

Thread Life Cycle.

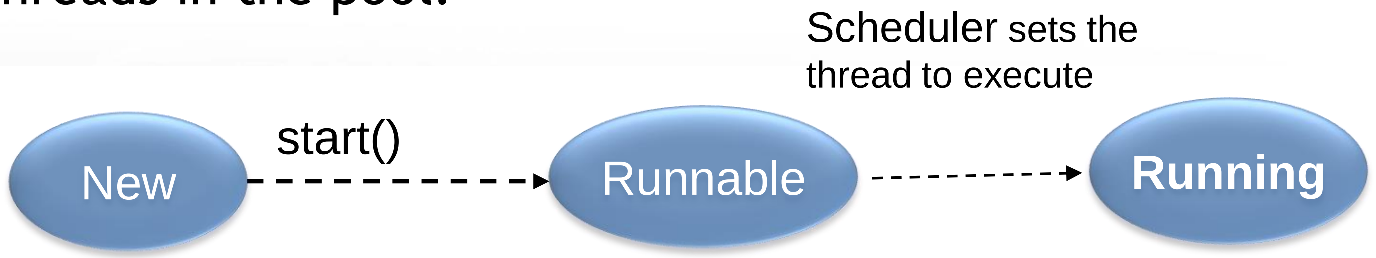
- When an instance of Thread Object is created it is in new state.

- When that thread invokes the start( ) method, it is now alive and is placed in a runnable pool where all the living threads wait their turn to be run by the scheduler.

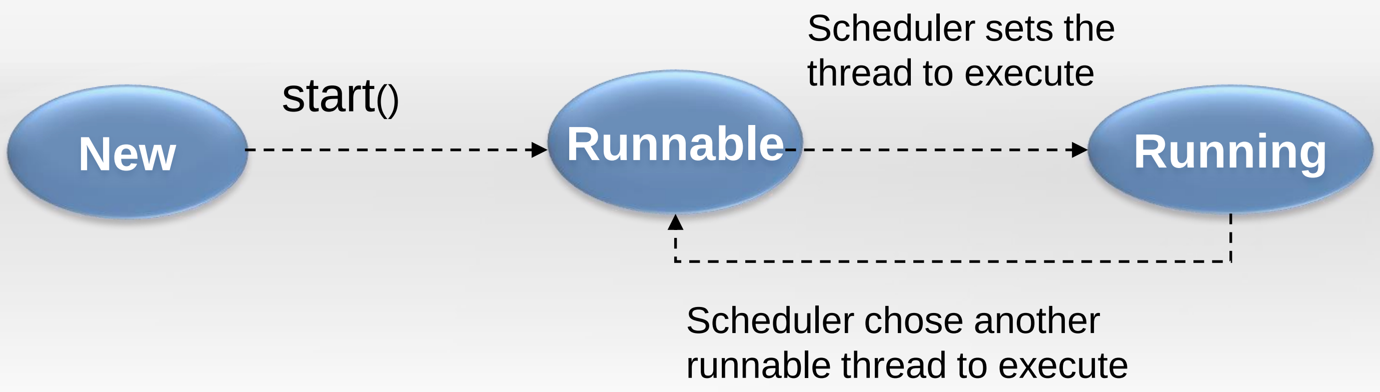


Thread Life Cycle.

- The thread scheduler chooses a thread to execute from the threads in the pool.

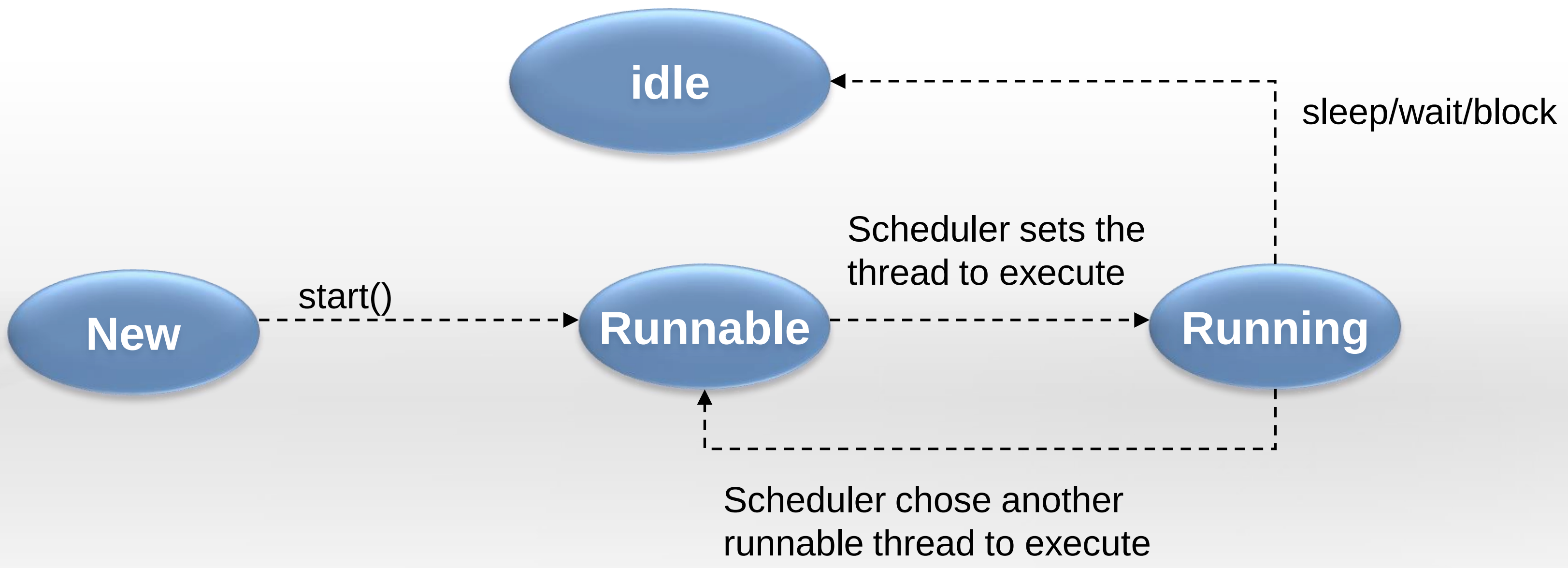


- If another thread is chosen by the scheduler to execute before it finishes, it goes back to the thread pool to await its turn again.


Scheduler sets the thread to execute
start()
New
Runnable
Running
Scheduler chose another runnable thread to execute

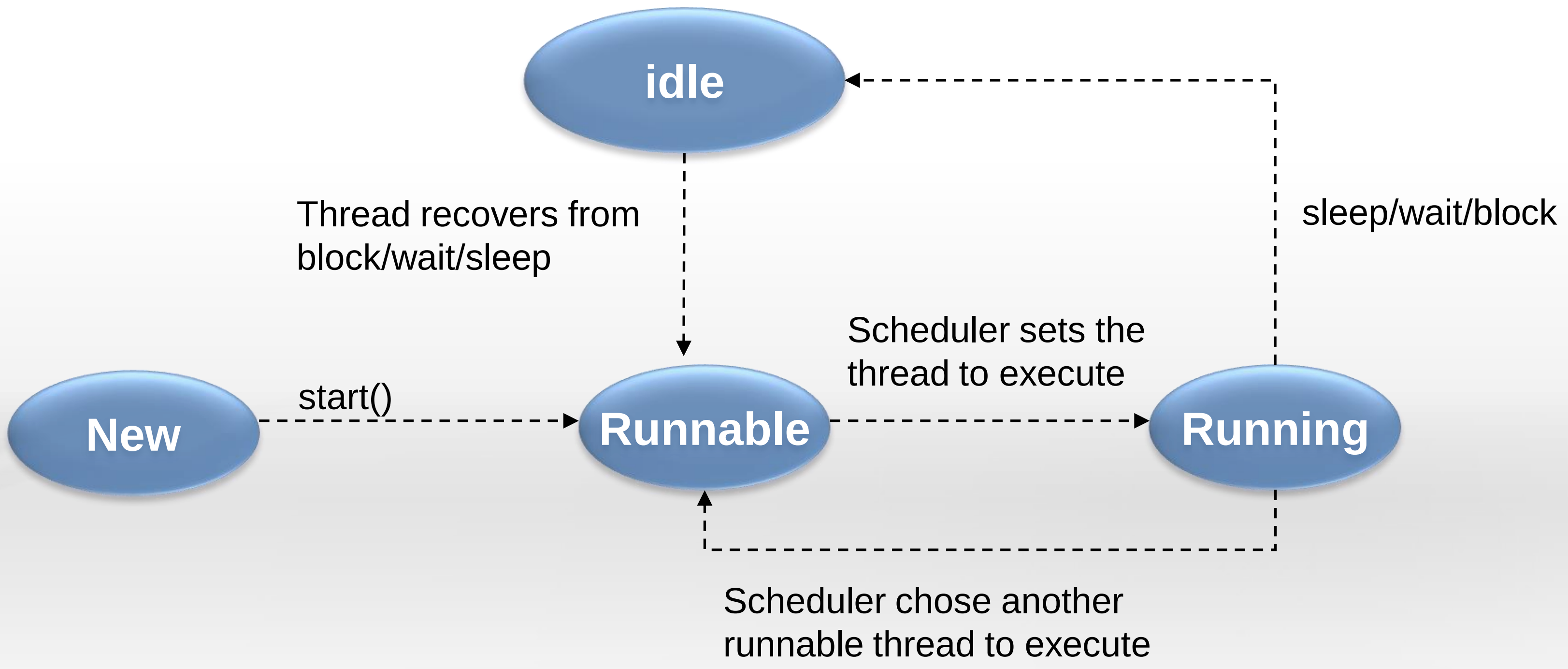
Thread Life Cycle

- A Thread may go to idle state (Not Runnable) while it was running because of sleep/block/wait.
  - Sleep : The Thread will pause itself for a specified duration
  - Block: Thread is waiting for a resource
  - Waiting: Thread is waiting for a signal from another thread.


idle
sleep/wait/block
Scheduler sets the thread to execute
New
start()
Runnable
Running
Scheduler chose another runnable thread to execute

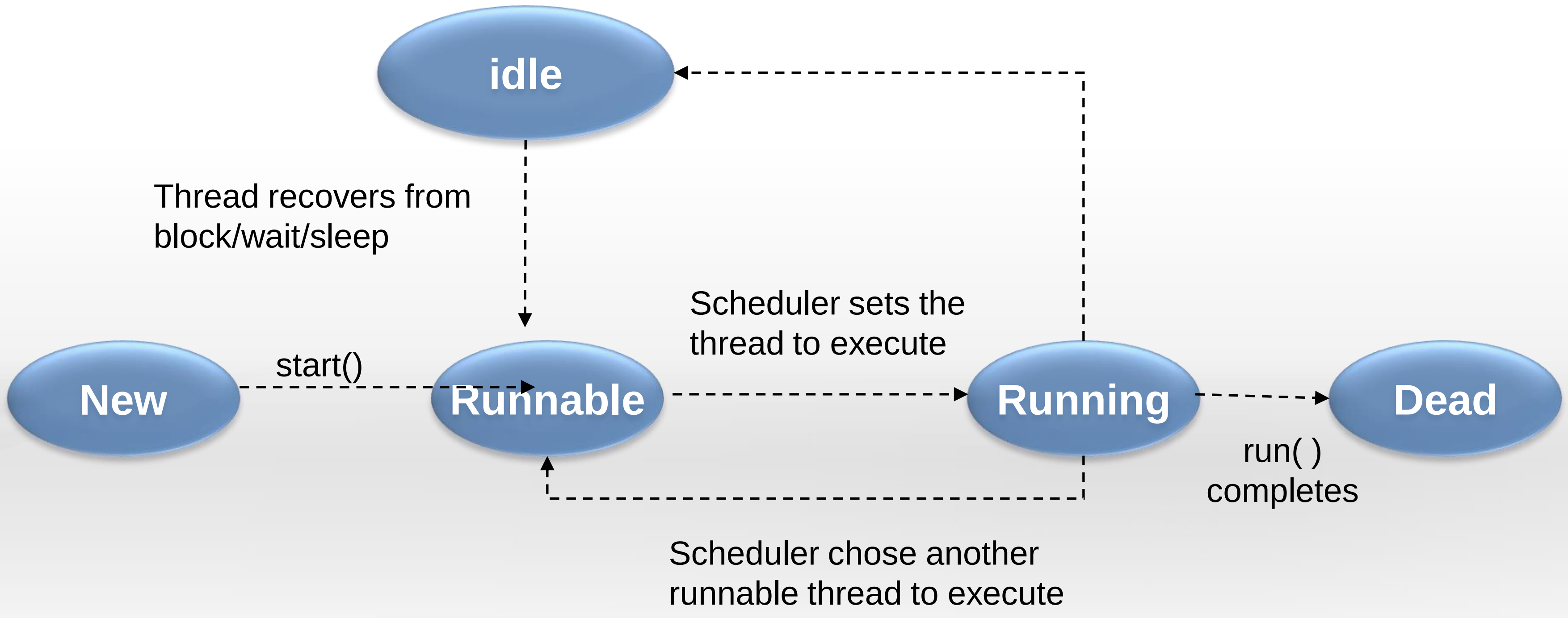
Thread Life Cycle.

- After the pause is over if it is sleeping or Thread gets a notification if it is waiting for signal or recovers from blocking mode, it returns to a *Runnable* state and waits in the pool to be executed again.


idle
Thread recovers from block/wait/sleep
sleep/wait/block
Scheduler sets the thread to execute
start()
New
Runnable
Running
Scheduler chose another runnable thread to execute

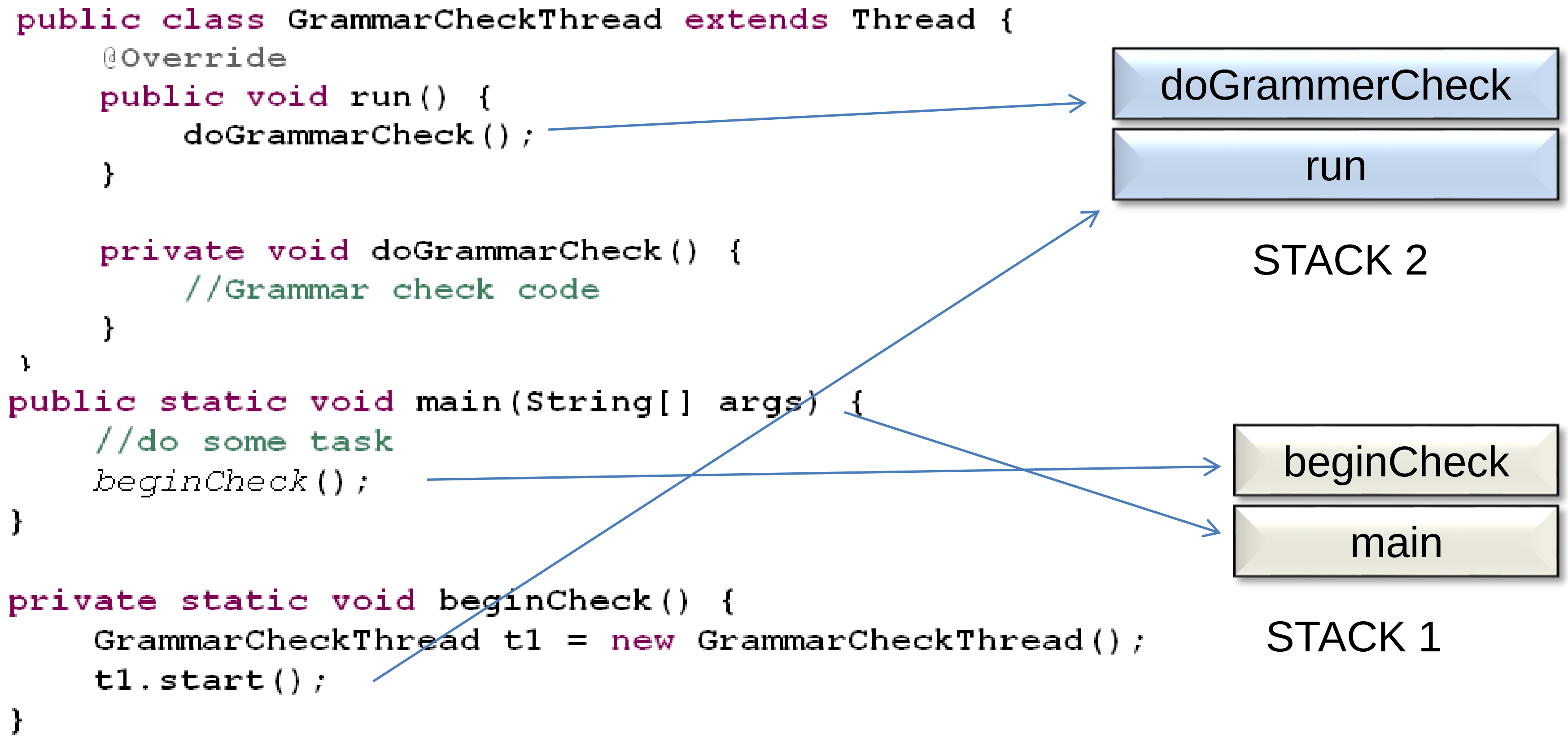
Thread Life Cycle

- If the thread finishes its execution by exiting its run() method, it is now a ‘dead’ thread.


idle
Thread recovers from block/wait/sleep
Scheduler sets the thread to execute
start()
New
Runnable
Running
Dead
run( ) completes
Scheduler chose another runnable thread to execute

Threads and Stacks

1. Main thread starts, places main() on the stack.
2. main() calls beginCheck( )
3. beginCheck() creates another thread and calls start( ).
4. start() places run() of that thread on a separate stack.
5. doGrammerCheck() is called from run( )



```
public class GrammarCheckThread extends Thread {
        @Override
        public void run() {
                doGrammarCheck();
        }

        private void doGrammarCheck() {
                //Grammar check code
        }
}
public static void main(String[] args) {
        //do some task
        beginCheck();
}

private static void beginCheck() {
        GrammarCheckThread t1 = new GrammarCheckThread();
        t1.start();
}
```

Threads and Stack

- Since each thread has a separate stack:
  - The data stored in local variables are not going to be effected in concurrent execution.
  - If any exception occurs in a thread and if not handled only stack of that thread is destroyed, the application will not be terminated.
  - Along with stacks Program Counter (PC) is also created one per thread.
  - The threads stack is destroyed as soon as the run( ) method is completed.
  - The main threads stack is destroyed when main completes.
- Refer : http://goo.gl/CZ3Ss (Inside Java Virtual Machine)

Thread Scheduler

- The thread scheduler is the part of the JVM that decides which thread should run at any given moment, and also takes threads *out of the run state*.
- *In a* single processor machine, only one thread can actually *run at a time*. *Only one stack* can ever be executing at one time.
- *Assuming a* single processor machine, any thread in the *runnable state can be chosen by the scheduler to be the one* and only running thread.

Note: ***The order in which runnable threads are chosen to run is not guaranteed.***

Thread Control Methods:

- start( ) :
  - Creates a new Thread of execution (with a new call stack).
  - Moves the thread from new state to runnable state.
  - Places the run( ) on the new call stack created for the current thread.

- sleep( ):
  - Static method of Thread class
  - use it in your code to "slow a thread down" by forcing it to go into a sleep mode before coming back to runnable.

Where can I use the sleep() method ?

Imagine a thread that runs in a loop, downloading the latest stock prices and analyzing them.

Downloading prices one after another would be a waste of time, as most would be quite similar—and even more important, it would be an incredible waste of precious bandwidth.

The simplest way to solve this is to cause a thread to pause (sleep) for five minutes after each download.

```
try {
    Thread.sleep(5 * 60 * 1000); // Sleep for 5 minutes
} catch (InterruptedException ex) {
    System.out.println(ex.getMessage());
}
```

Examples

- Code Example: thread_1.zip
  - Contains NumberThread.java and NumberThreadExample.java
  - Illustrates how to create threads by extending java.lang.Thread

- Code Example: thread_2.zip
- Contains CharacterThread.java and RunnableExample.java
  - Illustrates how to create threads by implementing java.lang.Runnable interface.

- Video: java_threads_1.swf
  - Illustrates thread creation

Quiz

- What would be the output of running the following code?

```
class TestThread extends Thread {
}

public class Test {
    public static void main(String[] args) {
        TestThread t1 = new TestThread();
        t1.start();
    }
}
```

Quiz

- What would be the output of running the following code?

```
class ExThread extends Thread {
     private String name = "";
     ExThread(String s) {
      name = s;
     }
     public void run() {
      for (int i = 0; i < 2; i++) {
       try {
        Thread.sleep(1000);
       } catch(InterruptedException e) {
       }
       System.out.println(name);
      }
     }
    }
    public class Test {
     public static void main(String[] args) {
      ExThread t1 = new ExThread("One");
      t1.run();
      ExThread t2 = new ExThread("Two");
      t2.run();
     }
    }
```

Quiz

- Assume three threads t1,t2 and t3 running, if an exception occurs in t1 thread what happens to the running application?

  Options:

  1. Only thread t1 terminates, other threads complete their execution.
  2. Entire application is terminated.
  3. Main thread completes its execution, Threads t1,t2 and t3 terminates.

Other Thread Control Methods

- **yield( ) :** make the currently running thread head back to runnable to allow other threads of the same priority to get their turn. So the intention is to use yield() to promote graceful turn-taking among equal-priority threads.

- **join( ) :** lets one thread "join onto the end“ of another thread.
  - If you have a thread B that can't do its work until another thread A has completed *its work, then you want thread B to "join" thread A*.
  - *This means that* thread B will not become runnable until A has finished.

Thread Safe

- Thread safety simply means that the fields of an object or class always maintain a valid state, as observed by other objects and classes, even when used concurrently by multiple threads.
- The following type of data are thread safe.
  - Local variables
    - local variables are residing in stack and each thread contains a separate stack
  - Immutable Objects
    - Once the object is created the data is not going to change.
  - Volatile variables.
    - It is a field modifier
    - every thread accessing a volatile field will read its current value before continuing, instead of using a cached value.
    - Will not work in all situations.

Example

- Data inconsistency in multi threaded application.
- Video:
    - Java_threads_2.swf
        - Illustrates how balance is inconsistent when multiple threads are accessing the same account concurrently.
    - Java_threads_2a.swf and java_threads_2b.swf
        - Illustrates how to debug multi threaded application.
- Code examples: threads_3.zip
    - Contains Account.java, DepositThread.java, Withdrawthread.java and BankingApplication.java
        - Examples used for video illustrating data inconsistency in multi threaded application.

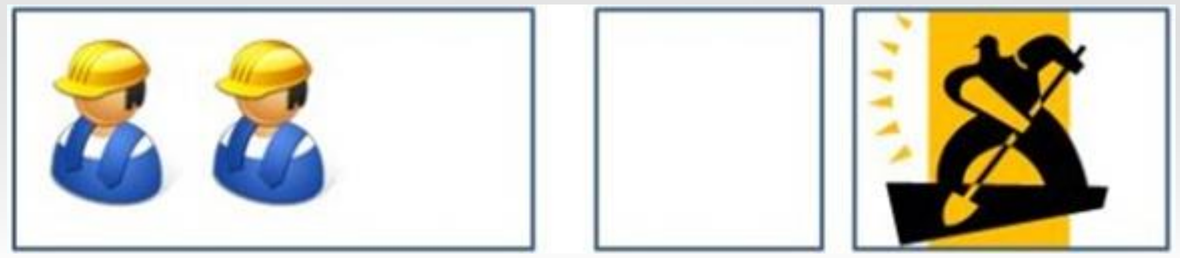
Thread Synchronization

- Monitor: Monitor is a Mutually exclusive lock.
- Every Object has one mutex.
- If a thread acquires a mutex, other thread has to wait for the mutex to be become free.

Threads

Shovel
(Monitor) is free

One Worker (Thread) grabs the shovel and starts digging.
Others wait until Shovel (monitor) is free.



Object Locks

- When a thread arrives at the first instruction in a monitor region, the thread must obtain a lock on the referenced object.
- The thread is not allowed to execute the code until it obtains the lock. Once it has obtained the lock, the thread enters the block of protected code.
- When the thread leaves the block, no matter how it leaves the block, it releases the lock on the associated object.
- You never explicitly lock an object. Object locks are internal to the Java virtual machine.

Synchronized methods

Thread T1

Thread **T1** calls synchronized method.
account.deposit( )

T1 acquire lock for object.
Account (SB123)

Execute the method completely

Account is locked, only thread **T1** can access it.

T1 releases the lock for object.
account

Continue…

Account is free for access, now other threads can deposit or withdraw.

Account
NO : SB123
Balance : xxxxx

Wait till thread gets the lock

Thread **T2** wants to call synchronized method account.withdraw( ) on same account (SB123).

Thread T2

T2 acquire lock for object.
account

Example

- Thread Synchronization
- Code Example: Modify Account class from threads_3.zip
  - In Account.java uncomment /* synchronized */ for deposit() and withdraw() methods and run BankingApplication.java.

Explore More!!

Never let your curiosity die!

Explore

- Thread Deadlocks
  - http://goo.gl/XFEmW
- Thread cooperation
  - The wait( ) , notify( ) and notifyAll( ) methods of Object class.
- Concurrent collection classes
  - ArrayBlockingQueue
  - CopyOnWriteArrayList
  - ConcurrentHashMap
- Thread Pooling
  - Using Executor and ExecutorService of java.util.concurrent package.

- synchronized versus java.util.concurrent.Lock.

References

Contains the reference that will supplement the self learning and will be needed for completing the assignments & practice questions

References

- Thread Synchronization explained
  - http://goo.gl/D0twa
- **Java theory and practice: Concurrent collections classes**
  - http://goo.gl/6B3bi
- Thinking in Java: Blocking (wait() and notify())
  - http://goo.gl/kYVez