

Java Programming Language SE – 6

Module 15: Threads

Team Emertxe

ORACLE®

Certified Professional

Java SE 6 Programmer



Objectives

- Define a thread
- Create separate threads in a Java technology program, controlling the code and data that are used by that thread
- Control the execution of a thread and write platform-independent code with threads
- Describe the difficulties that might arise when multiple threads share data
- Use wait and notify to communicate between threads
- Use synchronized to protect data from corruption





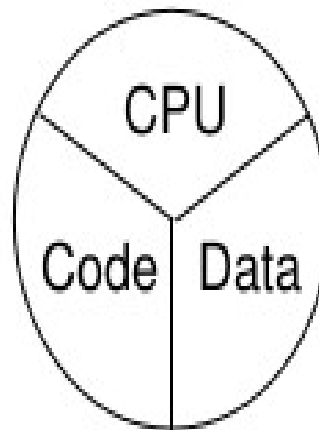
Relevance



- How do you get programs to perform multiple tasks concurrently?

Threads

- What are threads?
 - Threads are a virtual CPU.
- The three parts of a thread are:
 - CPU
 - Code
 - Data



A thread or
execution context

Creating the Thread

```
public class ThreadTester {  
    public static void main(String args[]) {  
        HelloRunner r = new HelloRunner();  
        Thread t = new Thread(r);  
        t.start();  
    }  
}  
class HelloRunner implements Runnable {  
    int i;  
    public void run() {  
        i = 0;  
        while (true) {  
            System.out.println("Hello " + i++);  
            if ( i == 50 ) {  
                break;  
            }  
        }  
    }  
}
```

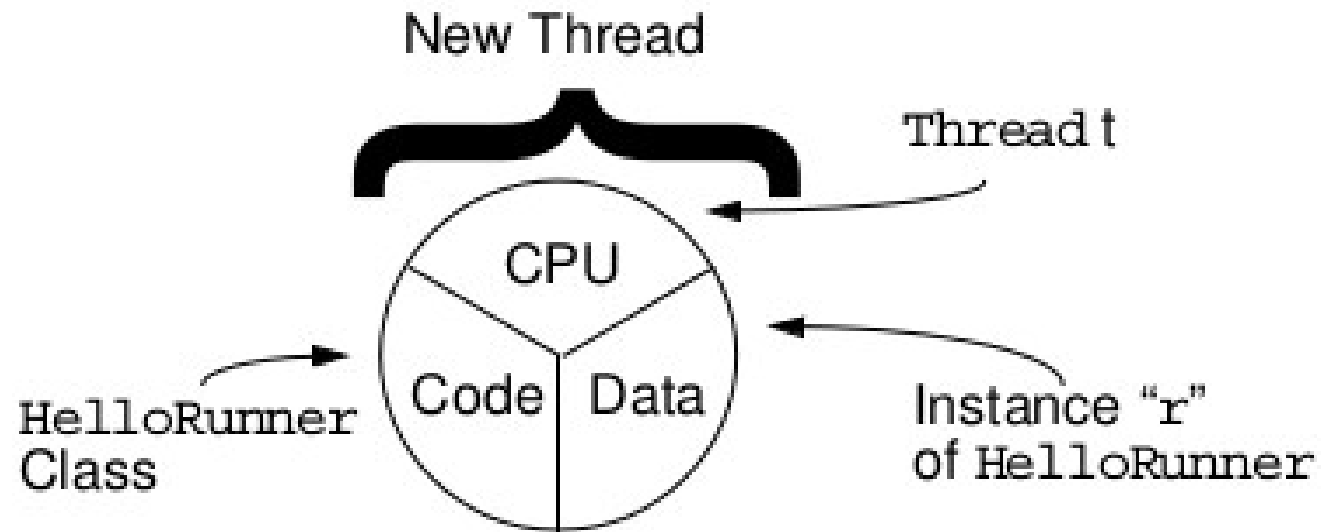
Creating the Thread

- Multithreaded programming has these characteristics:
- Multiple threads are from one Runnable instance.
- Threads share the same data and code.
- For example:

```
Thread t1 = new Thread(r);
```

```
Thread t2 = new Thread(r);
```

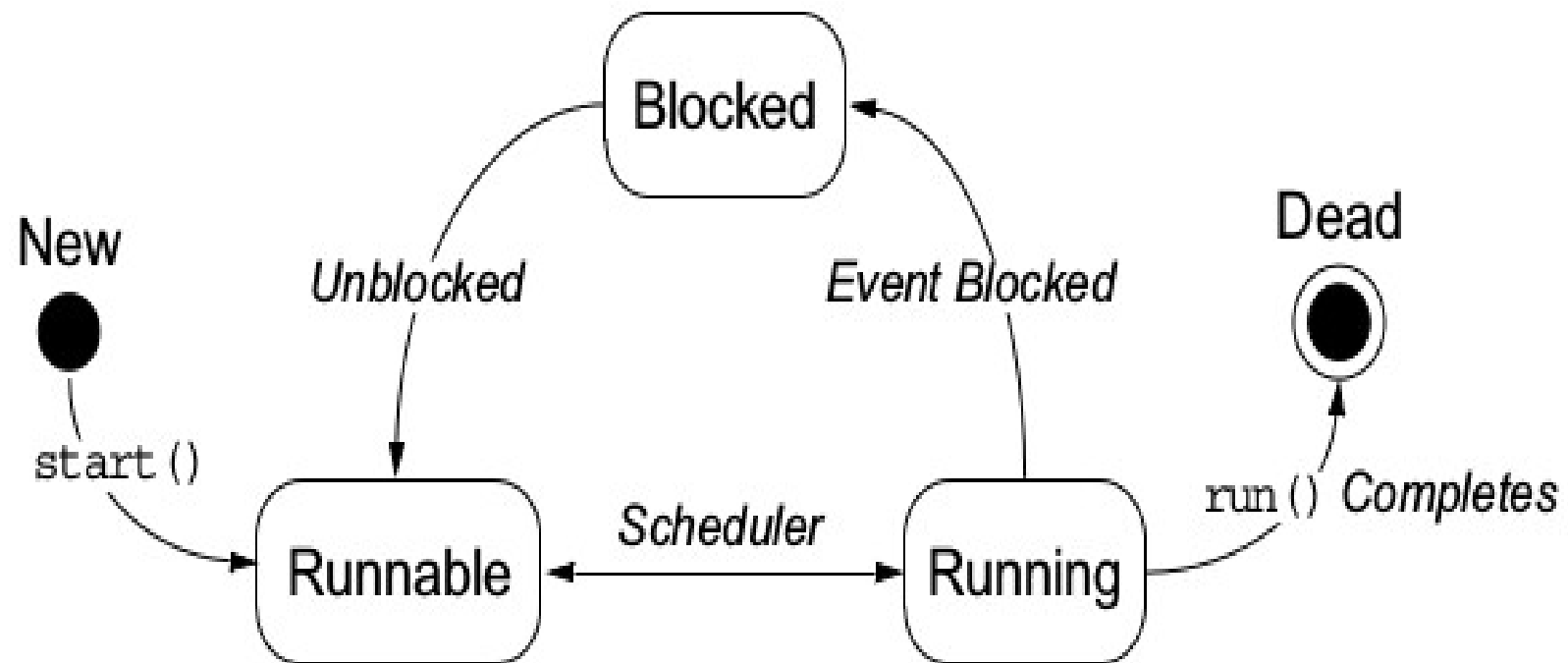
Creating the Thread



Starting the Thread

- Use the start method.
- Place the thread in a runnable state.

Thread Scheduling



Thread Scheduling Example

```
public class Runner implements Runnable {  
    public void run() {  
        while (true) {  
            // do lots of interesting stuff  
            // ...  
            // Give other threads a chance  
            try {  
                Thread.sleep(10);  
            } catch (InterruptedException e) {  
                // This thread's sleep was interrupted  
                // by another thread  
            }  
        }  
    }  
}
```

Terminating a Thread

```
public class Runner implements Runnable {  
    private boolean timeToQuit=false;  
    public void run() {  
        while ( ! timeToQuit ) {  
            // continue doing work  
        }  
        // clean up before run() ends  
    }  
    public void stopRunning() {  
        timeToQuit=true;  
    }  
}
```

Terminating a Thread

```
public class ThreadController {  
    private Runner r = new Runner();  
    private Thread t = new Thread(r);  
    public void startThread() {  
        t.start();  
    }  
    public void stopThread() {  
        // use specific instance of Runner  
        r.stopRunning();  
    }  
}
```

Basic Control of Threads



- Test threads:
 - `isAlive()`
- Access thread priority:
 - `getPriority()`
 - `setPriority()`
- Put threads on hold:
 - `Thread.sleep()`// static method
 - `join()`
 - `Thread.yield()`// static method

The join Method

```
public static void main(String[] args) {  
    Thread t = new Thread(new Runner());  
    t.start();  
    ...  
    // Do stuff in parallel with the other thread for a while  
    ...  
    // Wait here for the other thread to finish  
    try {  
        t.join();  
    } catch (InterruptedException e) {  
        // the other thread came back early  
    }  
    ...  
    // Now continue in this thread  
    ...}
```



Other Ways to Create Threads

```
public class MyThread extends Thread {  
    public void run() {  
        while ( true ) {  
            // do lots of interesting stuff  
            try {  
                Thread.sleep(100);  
            } catch (InterruptedException e) {  
                // sleep interrupted  
            }  
        }  
    }  
    public static void main(String args[]) {  
        Thread t = new MyThread();  
        t.start();  
    }  
}
```

Selecting a Way to Create Threads



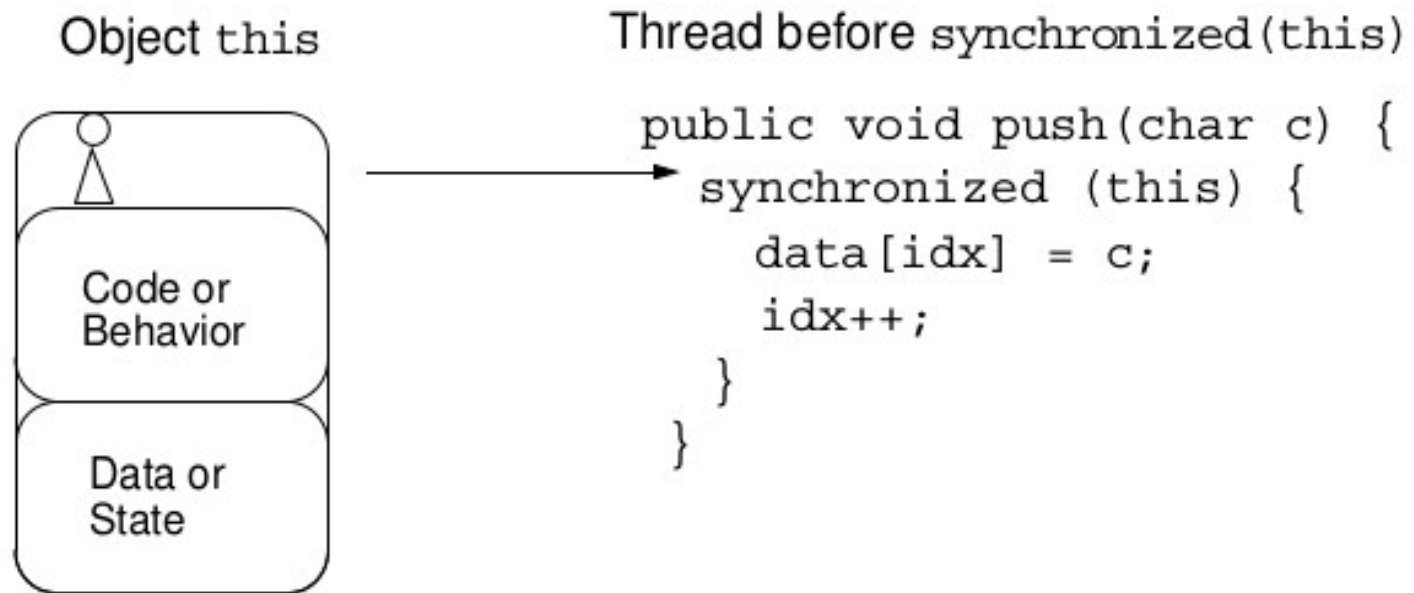
- Implement Runnable:
 - Better object-oriented design
 - Single inheritance
 - Consistency
- Extend Thread:
 - Simpler code

Using the synchronized Keyword

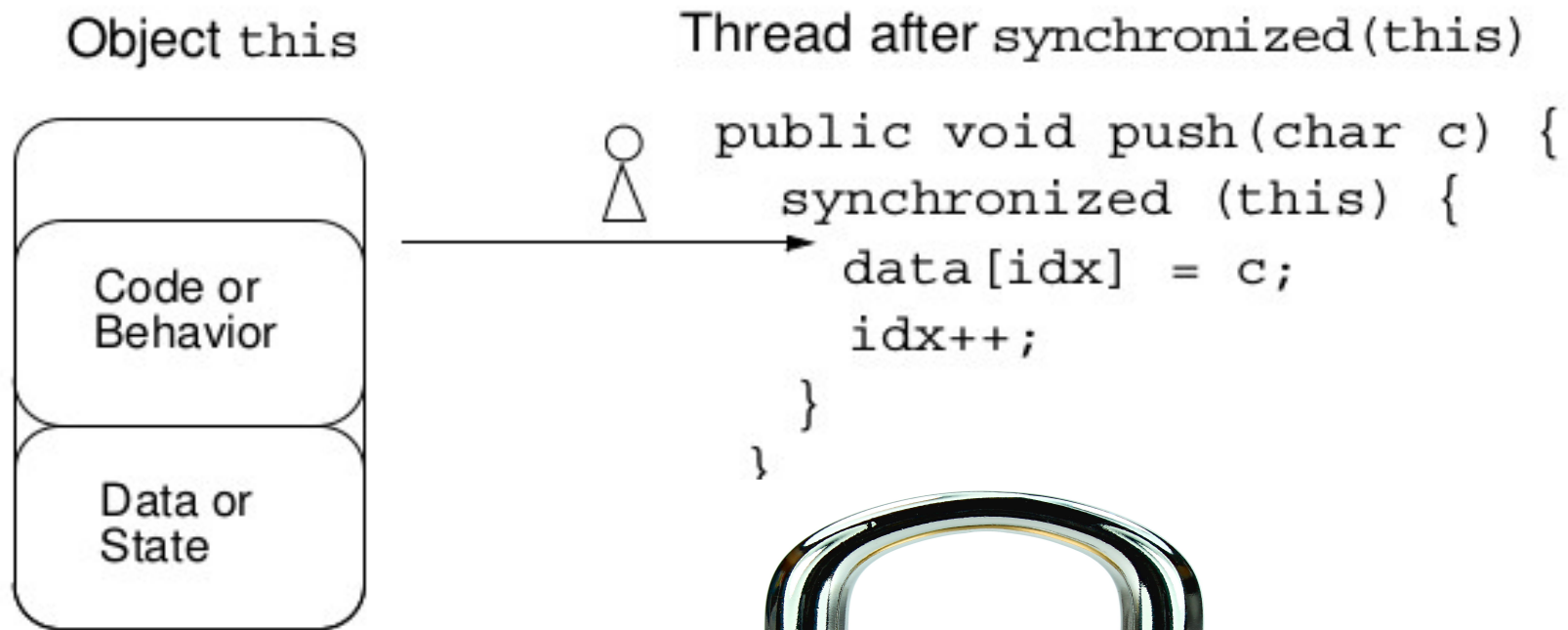
```
public class MyStack {  
    int idx = 0;  
    char [] data = new char[6];  
    public void push(char c) {  
        data[idx] = c;  
        idx++;  
    }  
    public char pop() {  
        idx--;  
        return data[idx];  
    }  
}
```

The Object Lock Flag

- Every object has a flag that is a type of lock flag.
- The synchronized enables interaction with the lock flag.

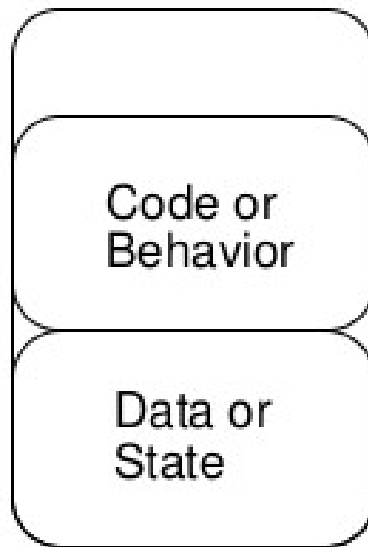


The Object Lock Flag



The Object Lock Flag

Object `this`
lock flag missing



Waiting for
object lock

Another thread, trying to
execute `synchronized(this)`

```
public char pop() {  
    synchronized (this) {  
        idx--;  
        return data[idx];  
    }  
}
```

Releasing the Lock Flag

The lock flag is released in the following events:

- Released when the thread passes the end of the synchronized code block
- Released automatically when a break, return, or exception is thrown by the synchronized code block



dreamstime.com

Using synchronized – Putting It Together



- All access to delicate data should be synchronized.
- Delicate data protected by synchronized should be private.

Using synchronized – Putting It Together

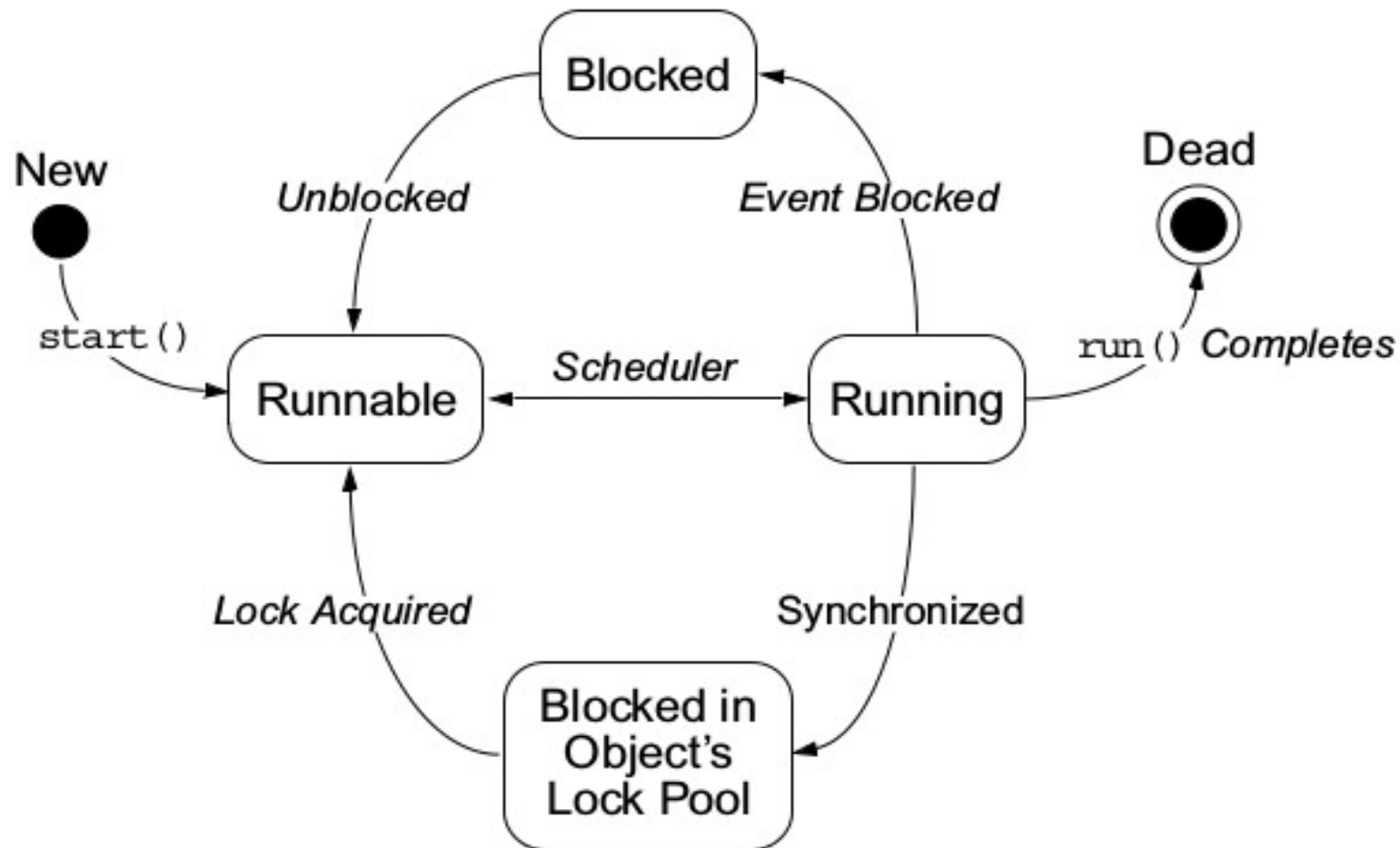


The following two code segments are equivalent:

```
public void push(char c) {  
    synchronized(this) {  
        // The push method code  
    }  
}
```

```
public synchronized void push(char c) {  
    // The push method code  
}
```

Thread State Diagram With Synchronization



Deadlock

A deadlock has the following characteristics:

- It is two threads, each waiting for a lock from the other.
- It is not detected or avoided.
- Deadlock can be avoided by:
 - Deciding on the order to obtain locks
 - Adhering to this order throughout
 - Releasing locks in reverse order

Thread Interaction – wait and notify



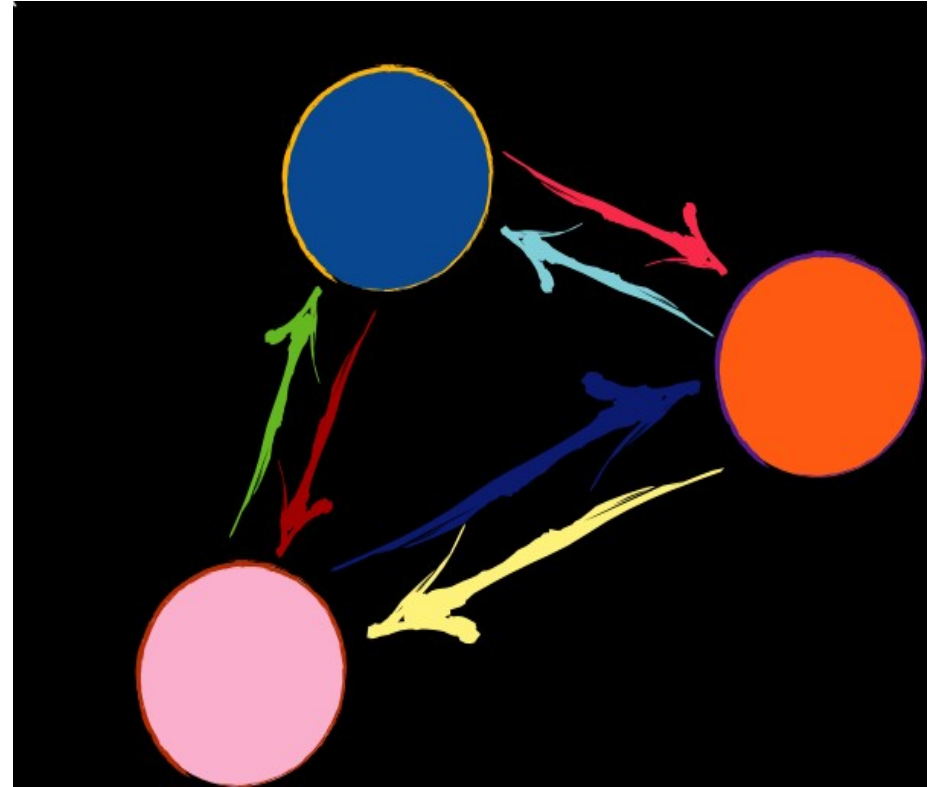
- Scenario:
 - Consider yourself and a cab driver as two threads.
- The problem:

How do you determine when you are at your destination?
- The solution:
 - You notify the cab driver of your destination and relax.
 - The driver drives and notifies you upon arrival at your destination.

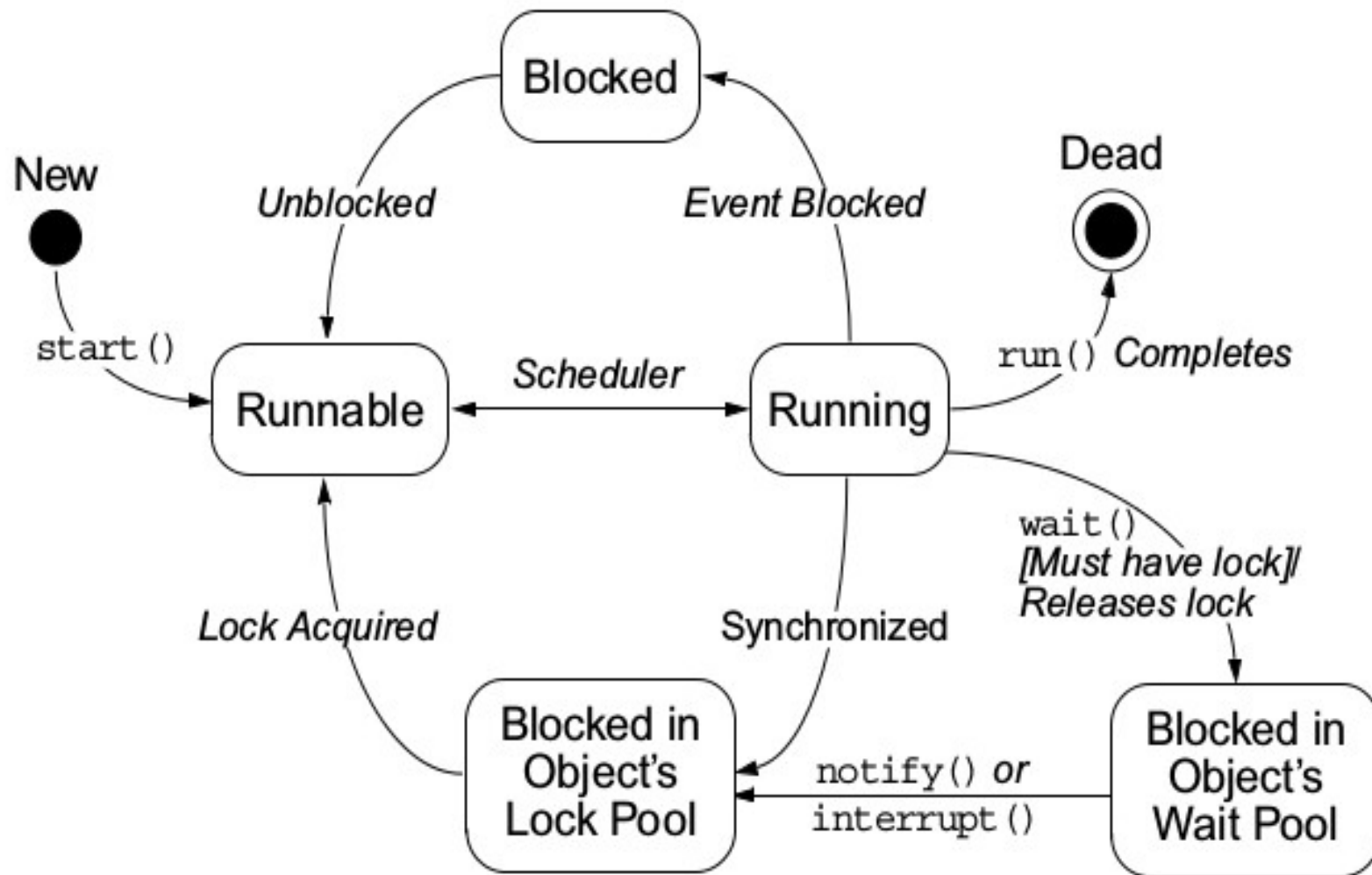
Thread Interaction

Thread interactions include:

- The wait and notify methods
- The pools:
 - Wait pool
 - Lock pool



Thread State Diagram With wait and notify



Monitor Model for Synchronization



- Leave shared data in a consistent state.
- Ensure programs cannot deadlock.
- Do not put threads expecting different notifications in the same wait pool.

The Producer Class

```
package mod13;

public class Producer implements Runnable {
    private SyncStack theStack;
    private int num;
    private static int counter = 1;
    public Producer (SyncStack s) {
        theStack = s;
        num = counter++;
    }
}
```

The Producer Class

```
public void run() {  
    char c;  
    for (int i = 0; i < 200; i++) {  
        c = (char)(Math.random() * 26 + 'A');  
        theStack.push(c);  
        System.out.println("Producer" + num + ": " + c);  
        try {  
            Thread.sleep((int)(Math.random() * 300));  
        } catch (InterruptedException e) {  
            // ignore it  
        }  
    }  
}
```

The Consumer Class

```
package mod13;  
  
public class Consumer implements Runnable {  
    private SyncStack theStack;  
    private int num;  
    private static int counter = 1;  
    public Consumer (SyncStack s) {  
        theStack = s;  
        num = counter++;  
    }  
}
```


The Consumer Class

```
public void run() {  
    char c;  
    for (int i = 0; i < 200; i++) {  
        c = theStack.pop();  
        System.out.println("Consumer" + num + ": " + c);  
        try {  
            Thread.sleep((int)(Math.random() * 300));  
        } catch (InterruptedException e) {  
            // ignore it  
        }  
    }  
} // END run method
```

Stay connected

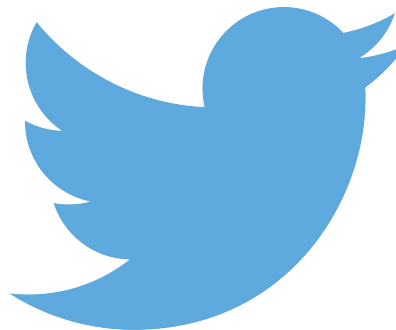


About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046
T: +91 80 6562 9666
E: training@emertxe.com



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



slideshare
Present Yourself

<https://www.slideshare.net/EmertxeSlides>



Thank You