

Synchronization

CS 272 Software Development

Providing Consistency

- If **multithreading**...
 - If **sharing data** between threads...
 - If shared data not already **thread safe**...
 - must **synchronize** access to that data



Synchronization

- Using the **synchronized** keyword and intrinsic (or monitor) lock objects to protect blocks of code
- Using the **volatile** keyword to protect* variables
- Using **wait()** and **notifyAll()** to coordinate threads
- Using **conditional synchronization** via lock objects



Synchronization

- Using the **synchronized** keyword and intrinsic (or monitor) lock objects to protect blocks of code
- Using the **volatile** keyword to protect* variables
- Using **wait()** and **notifyAll()** to coordinate threads
- Using **conditional synchronization** via lock objects



Synchronized Keyword

```
private Object lock;  
private int a;
```

```
public void increment {  
    synchronized (lock) {  
        a++;  
    }  
}
```

```
public void decrement {  
    synchronized (lock) {  
        a--;  
    }  
}
```



Synchronized Keyword

- Used to create **atomic** (uninterruptible) code
- Can be applied to blocks of code or an entire method
- If applied consistently everywhere shared data is accessed by multiple threads, provides **thread safety**
- Requires an **intrinsic lock** or monitor lock object to determine which threads to block



Synchronized Keyword

```
private Object lock;  
private int a;
```

```
public void increment {  
    synchronized (lock) {  
        a++;  
    }  
}
```

```
public void decrement {  
    synchronized (lock) {  
        a--;  
    }  
}
```

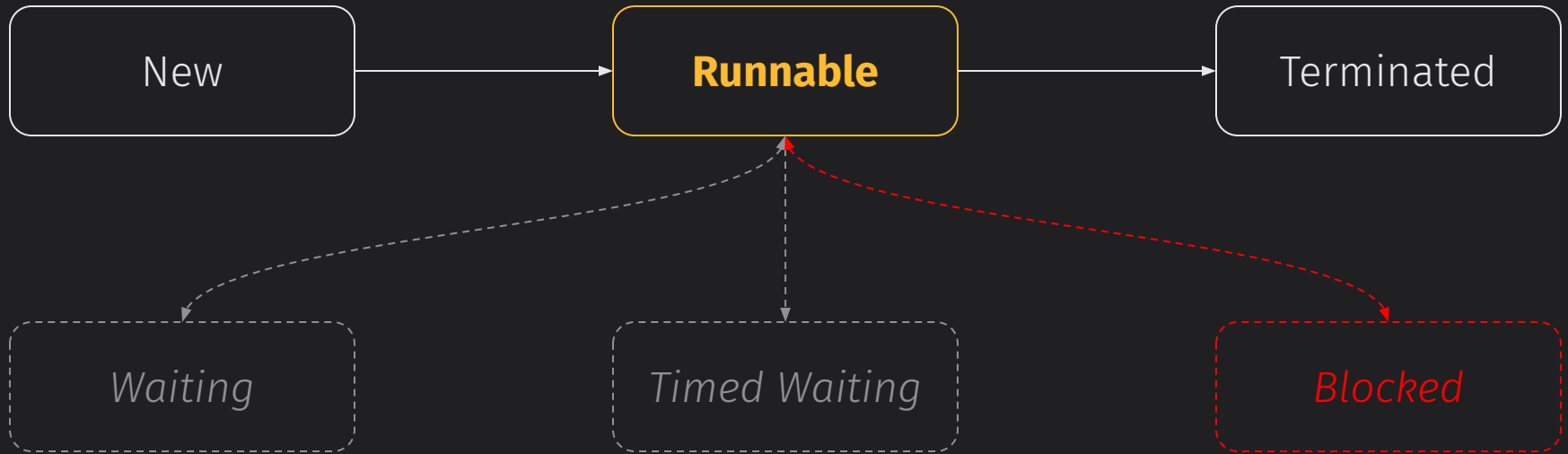


Synchronized Keyword

- An entering thread must attempt to **acquire** lock
 - Only one thread may hold lock object at once
 - Other code may use the same lock object
- The thread is **blocked** until able to obtain lock object
- The lock object is automatically **released** when a thread exits the synchronized code



Thread States



<https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Thread.State.html>

Synchronized Keyword

```
private Object lock;  
private int a;
```

```
public void increment {  
    synchronized (lock) {  
        a++;  
    }  
}
```

```
public void decrement {  
    synchronized (lock) {  
        a--;  
    }  
}
```



Intrinsic Locks



Intrinsic Locks

- Must specify an object to use as the intrinsic lock
- Exact behavior depends on type of object used
 - e.g. class member versus an instance member
- Controls which threads are blocked and how many threads may access a synchronized block



Synchronization Example

```
private Object lock;  
private int a;
```

```
public void increment {  
    synchronized (lock) {  
        a++;  
    }  
}
```

```
public void decrement {  
    synchronized (lock) {  
        a--;  
    }  
}
```



Synchronization Example

```
private Object lock1;           private Object lock2;
private int a;

public void increment {        public void decrement {
    synchronized (lock1) {   synchronized (lock2) {
        a++;                    a--;
    }
}                               }
```

**Assume lock1 and lock2 are different instances...*



Synchronization Example

```
// private Object lock;  
private int a;
```

```
public void increment {  
    synchronized (this) {  
        a++;  
    }  
}
```

```
public void decrement {  
    synchronized (this) {  
        a--;  
    }  
}
```



Synchronization Example

```
private int a;
```

```
public synchronized void increment {  
    a++;  
}
```

```
public synchronized void decrement {  
    a--;  
}
```



Synchronized Methods

- Any method may be declared **synchronized**
 - `public synchronized void method()`
- Equivalent to placing all code within method in a **synchronized (this)** block
- All **synchronized** methods within a class use the same lock and may not run concurrently

** [Using “this” to handle synchronization can cause security issues...](#) ***



Synchronization Issues

- Protects code, **NOT** objects
 - Does not protect the lock or any objects within
- Must be used consistently to provide **thread safety**
 - Objects accessed within may still be accessed concurrently elsewhere in code
- Causes **blocking**, which slows down code





CHANGE THE WORLD FROM HERE