USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Multithreading

Synchronization

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read value of **x**	read value of **x**
2	calculate **x + 1**	calculate **x – 1**
3	assign **x** to calculated result	assign **x** to calculated result
4		
5		
6		
7		

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read **`x = 1`**	
2	calculate **`1 + 1 = 2`**	
3	assign **`x = 2`**	
4		read **`x = 2`**
5		calculate **`2 - 1 = 1`**
6		assign **`x = 1`**
7		

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read **`x = 1`**	
2	calculate **`1 + 1 = 2`**	
3	assign **`x = 2`**	
4		read **`x = 2`**
5		calculate **`2 - 1 = 1`**
6		assign **`x = 1`**
7	final value **`x = 1`**	

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read `x = 1`	
2		read `x = 1`
3	calculate `1 + 1 = 2`	
4		calculate `1 - 1 = 0`
5	assign `x = 2`	
6		assign `x = 0`
7		

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read `x = 1`	
2		read `x = 1`
3	calculate `1 + 1 = 2`	
4		calculate `1 - 1 = 0`
5	assign `x = 2`	
6		assign `x = 0`
7	final value `x = 0`	

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read `x = 1`	
2		read `x = 1`
3	calculate `1 + 1 = 2`	
4		calculate `1 - 1 = 0`
5		assign `x = 0`
6	assign `x = 2`	
7	final value `x = 2`	

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Problems

- Operators x++ and x-- are not **atomic** operations
  - Unable to **divide** operation(s)
  - Unable to **interrupt** when multithreading
  - All operations succeed or all fail (no partial results)
- Shared data is modified between read and use
  - Shared variable x is not **thread safe**

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Thread Safety

- An object is **thread safe** if it maintains a valid or consistent state even when accessed concurrently
- Includes all constants and **immutable** objects
  - e.g. `String` or primitive types that are `final`
- Includes some **mutable** objects
  - e.g. `StringBuffer` (***NOT*** `StringBuilder`), `java.util.concurrent.*`

*You are not allowed to use the `java.util.concurrent` package in this class!

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Synchronization

- Use **synchronization** is coordinate threads
  - Use to protect objects that are not thread safe
  - Use to provide atomic blocks of code
- Synchronization in Java
  - Use **`synchronized`** functions or blocks of code
  - Use **`volatile`** variables
  - Use specialized **`lock`** objects

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Synchronized Blocks

- Must specify an object to use as a lock
  - Any calls to `wait()` or `notify()` within block must be called on lock object
- Exact behavior depends on type of object used
  - A class member versus an instance member versus an inner instance member all behave differently

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Synchronization Example

```
1  private Object lock;
2  private int a;
3
4  public void increment {
5      synchronized (lock) {
6          a++;
7      }
8  }
9
```

```
10 public void decrement {
11     synchronized (lock) {
12         a--;
13     }
14 }
```


Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science

Synchronized Blocks

- A thread entering block must attempt to **acquire** lock
  - Only one thread may hold lock object at once
  - Multiple blocks 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 block

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Synchronization Example

```
1  private Object lock;
2  private int a;
3
4  public void increment {
5      synchronized (lock) {
6          a++;
7      }
8  }
9
```

```
10 public void decrement {
11     synchronized (lock) {
12         a--;
13     }
14 }
```


Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science

Synchronization Example

```
1  // private Object lock;
2  private int a;
3
4  public void increment {
5      synchronized (this) {
6          a++;
7      }
8  }
9
```

```
10 public void decrement {
11     synchronized (this) {
12         a--;
13     }
14 }
```


Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science

Synchronization Example

```
1  private int a;
2
3  public synchronized void increment {
4      a++;
5  }
6
7  public synchronized void decrement {
8      a--;
9  }
```


Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

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

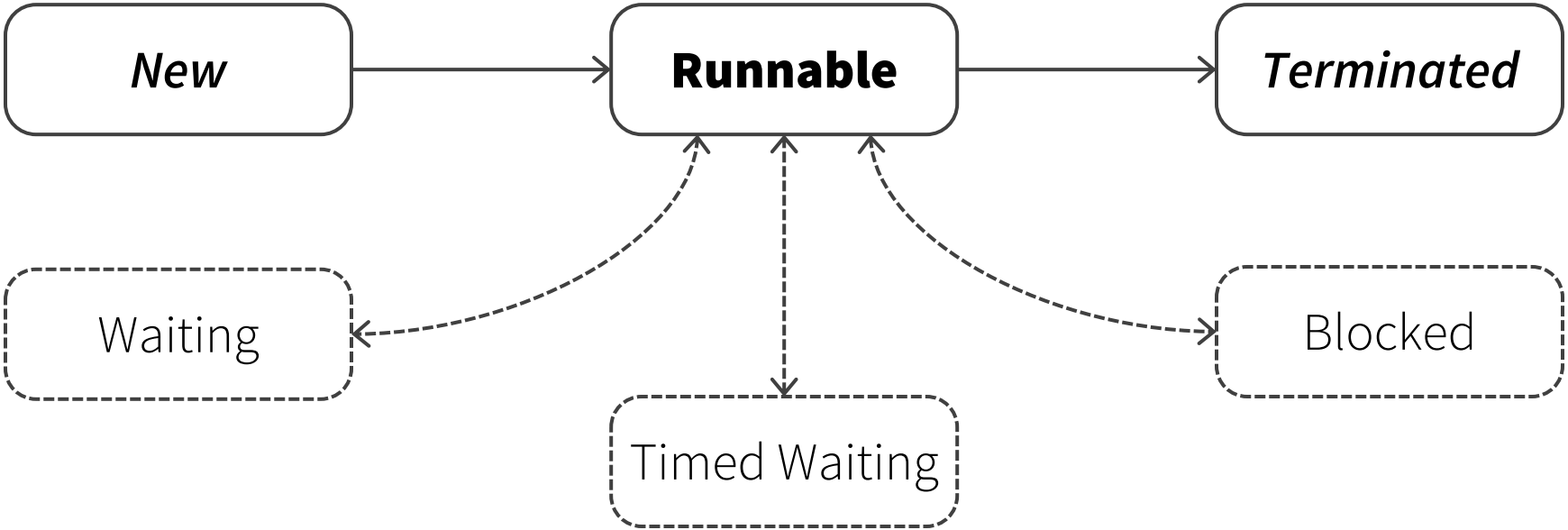
Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Synchronization Issues

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

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

Thread States



http://www.ibm.com/developerworks/java/tutorials/j-threads/section3.html

Sophie Engle ◆ sjengle@cs.usfca.edu
Department of Computer Science