COMP 401
SYNCHRONIZED METHODS

Instructor: Prasun Dewan
Prerequisite

- Animation Threads Commands
THREARDS AND SHARING

Same Lane ~  
Same  
Object/Resource

Different Lane ~  
Different  
Objects/Resources

Sharing Road ~ Sharing Computer
THE SEPARATE LANE SCENARIO

Call Diagram

Node labels: Method and Class/Interface

Edges denote calls

AnObservable Plotted Shuttle

setShuttleX(Y)()

Shared lane?

 Thread

start() start()

Main Thread

Main Class

main() start() run() run()

AShuttleAnimator

animate Shuttle()

AShuttleAnimation Command

run()
One Shuttle & Animator, Two Threads

Two threads accessing same animator and shuttle
public static void main(String[] args) {
    PlottedShuttle shuttle1 = new AnObservablePlottedShuttle(50, 100);
    OEFrame oeFrame1 = ObjectEditor.edit(shuttle1);
    oeFrame1.hideMainPanel();
    oeFrame1.setLocation(0, 0);
    oeFrame1.setSize(400, 400);
    PlottedShuttle shuttle2 = new AnObservablePlottedShuttle(100, 50);
    OEFrame oeFrame2 = ObjectEditor.edit(shuttle2);
    oeFrame2.hideMainPanel();
    oeFrame2.setLocation(400, 0);
    oeFrame2.setSize(400, 400);
    ShuttleAnimator shuttleAnimator1 = new AShuttleAnimator();
    ShuttleAnimator shuttleAnimator2 = new AShuttleAnimator();
    concurrentDemoShuttleAnimation(shuttleAnimator1, shuttle1);
    concurrentDemoShuttleAnimation(shuttleAnimator2, shuttle2);
}

Two different threads are created which interleave their activities
public static void main(String[] args) {
    PlottedShuttle shuttle1 = new AnObservablePlottedShuttle(50, 100);
    OEFrame oeFrame1 = ObjectEditor.edit(shuttle1);
    oeFrame1.setLocation(0, 0);
    oeFrame1.setSize(500, 550);
    ShuttleAnimator aShuttleAnimator = new AShuttleAnimator();
    concurrentDemoShuttleAnimation(aShuttleAnimator, shuttle1);
    ThreadSupport.sleep(500);
    concurrentDemoShuttleAnimation(aShuttleAnimator, shuttle1);
}
INTERFERING ANIMATIONS
INTERFERENCE

```java
while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}
```

Each call of method gets its own copy of local variables such as curX and curY
while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}
while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}
while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}

while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}
INTERFERENCE
AnObservable Plotted Shuttle

AShuttleAnimator

animate Shuttle()

setShuttleX(Y)()

AShuttleAnimation Command

run()

AShuttleAnimation Command

run()

Shuttle Animation Thread 1

start()

Shuttle Animation Thread 2

start()

Main Class

main()

Main Thread

What?

How?
public class ASynchronizedShuttleAnimator {
    extends AShuttleAnimator {

    public synchronized void animateFromOrigin(
        PlottedShuttle shuttle,
        int animationStep,
        int animationPauseTime) {
        super.animateFromOrigin(
            shuttle,
            animationStep,
            animationPauseTime);
    }
}

Should use keyword synchronized in method with shared data to tell Java that only one thread should execute the method at one time

Atomic method execution – only one thread executes it at one time

When a method is locked by a thread other threads wait in a queue, and when a method is unlocked the first waiting thread executes it
SYNCHRONIZED
One Shuttle and Synchronized Animator, Two Threads

On what method should lock be put?

Main Class

main

AnObservable Plotted Shuttle

setShuttleX(Y)()

AShuttleAnimator

animate Shuttle()

AShuttleAnimation Command

run()

run()

Shuttle Animation Thread 1

Shuttle Animation Thread 2

Main Thread

Main

Class

AShuttleAnimation Command

setShuttleX(Y)()
**SYNCHRONIZE SET METHODS OF SHUTTLE?**

Set method executes without interference, not animation loop

- AnObservable Plotted Shuttle
  - tShuttleX(Y)()
  - AShuttleAnimator
    - animate Shuttle()
    - run()
    - run()
    - AShuttleAnimation Command
      - Shuttle Animation Thread 1
      - main
      - Main Class
      - Main Thread
    - Shuttle Animation Thread 2
      - run()
      - AShuttleAnimation Command

---

**Main Thread**

- Main
- Shuttle Animation Thread 1
- Shuttle Animation Thread 2
- AShuttleAnimation Command
- AnObservable Plotted Shuttle
  - tShuttleX(Y)()
SYNCHRONIZE RUN METHODS OF COMMAND?

Each command has its own queue

Main Class: main

AShuttleAnimator
- animate Shuttle()

AShuttleAnimation Command
- run()

Shuttle Animation Thread 1
- setShuttleX(Y)()

Shuttle Animation Thread 2
- run()

AShuttleAnimation Command

Main Thread

AnObservable Plotted Shuttle

SetShuttleX(Y)()
Synchronize Thread Creation?

Threads created atomically, but then they can interfere.
Synchronize makes caller wait till interfering activity finishes

Method executed by new thread should be synchronized, not the method that created the thread.
**Two Synchronized Methods**

```java
public class ASynchronizedShuttleAnimator
    extends AShuttleAnimator {

    public synchronized void animateFromOrigin(
            PlottedShuttle shuttle,
            int animationStep,
            int animationPauseTime) {
        super.animateFromOrigin(
            shuttle,
            animationStep,
            animationPauseTime);
    }

    public synchronized void animateFromOrigin(
            PlottedShuttle shuttle,
            int animationStep,
            int animationPauseTime, OFrame frame) {
        super.animateFromOrigin(
            shuttle,
            animationStep,
            animationPauseTime,
            frame);
    }
}
```

Only one synchronized method can be executed at one time in a class

Lock and queue is with the object, not method
**Synchronized Methods**

- Start call to synchronized method on object O
- if some synchronized method is executing in O, then makes calling thread wait in O’s queue
- Finish call to synchronized method on object O
- Unblocks first waiting thread in O’s queue
SYNCHRONIZED METHOD ANALOGY?
One Shuttle, Two Synchronized Animators

```java
public class OneShuttleTwoSynchronizedShuttleAnimators
    extends ConcurrentShuttleAnimationDriver {
    public static void main(String[] args) {
        PlottedShuttle shuttle1 =
            new AnObservablePlottedShuttle(50, 100);
        OEFrame oeFrame = ObjectEditor.edit(shuttle1);
        oeFrame.hideMainPanel();
        oeFrame.setSize(400, 400);
        ShuttleAnimator aShuttleAnimator1 =
            new ASynchronizedShuttleAnimator();
        ShuttleAnimator aShuttleAnimator2 =
            new ASynchronizedShuttleAnimator();
        concurrentDemoShuttleAnimation(aShuttleAnimator1, shuttle1);
        ThreadSupport.sleep(500);
        concurrentDemoShuttleAnimation(aShuttleAnimator2, shuttle1);
    }
}
```

Lock, queue is with each object, not the class

Will again get interference
**One Shuttle, Two Threads and Synchronized Animators**

```
ASynchronizedShuttleAnimator.animateShuttle()

APlottedShuttle.setShuttleX(Y)()

ASynchronizedShuttleAnimator.animateShuttle()

APlottedShuttle.setShuttleX(Y)()

AShuttleAnimationCommand.run()

AShuttleAnimationCommand.run()

ShuttleAnimationThread1

ShuttleAnimationThread2

MainThread

MainClass.main

Need to define our own synchronization protocol
```
Synchronization

- Methods that access global state and can be executed by multiple threads should be made synchronized.
- Only one synchronized method in an object will be executed at one time.