



COMP 401

SYNCHRONIZED METHODS

Instructor: Prasan Dewan



PREREQUISITE

- Animation Threads Commands

More?



THREADS AND SHARING

Same Lane ~
Same
Object/Resource



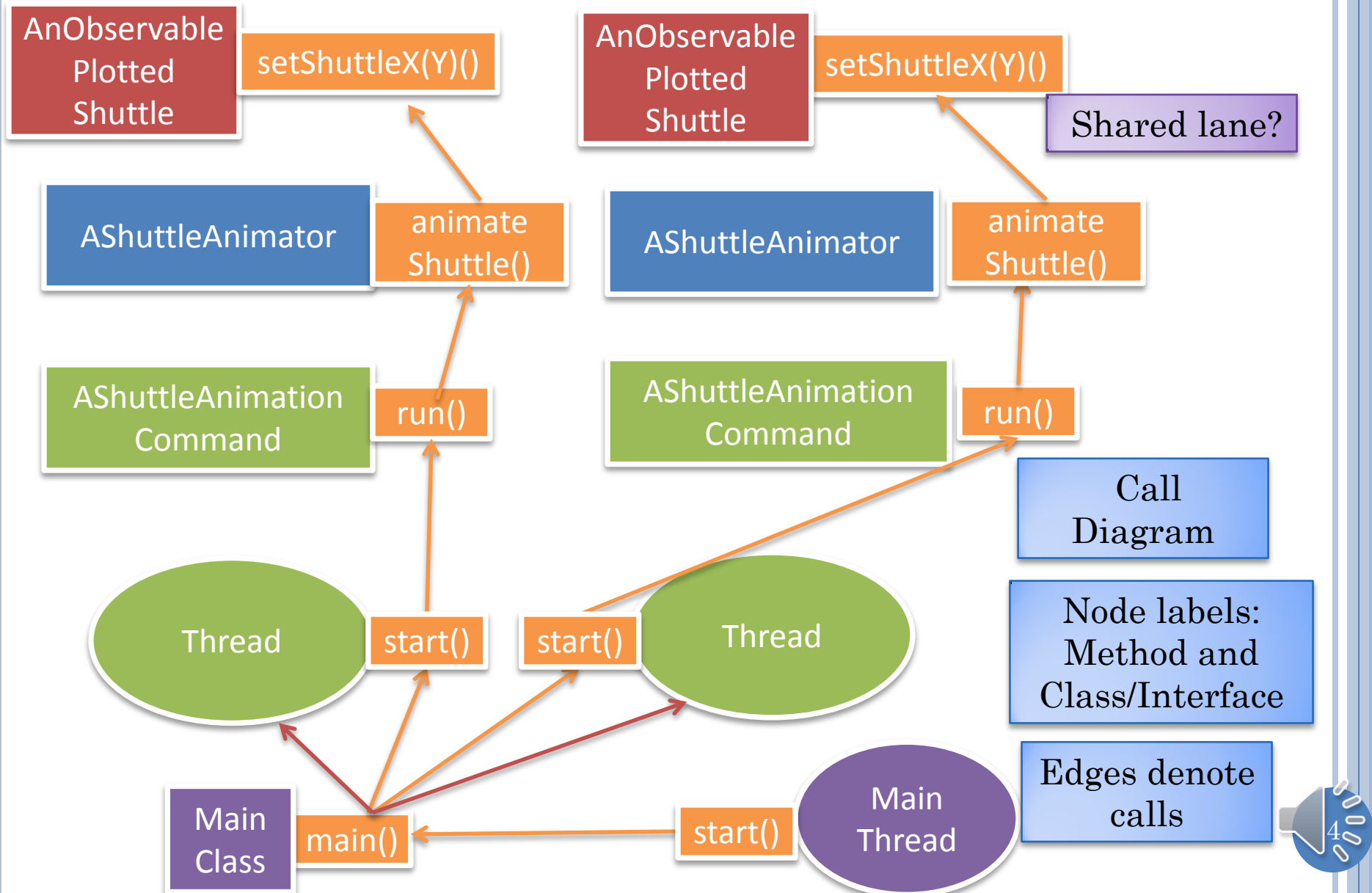
Different Lane ~
Different
Objects/Resources



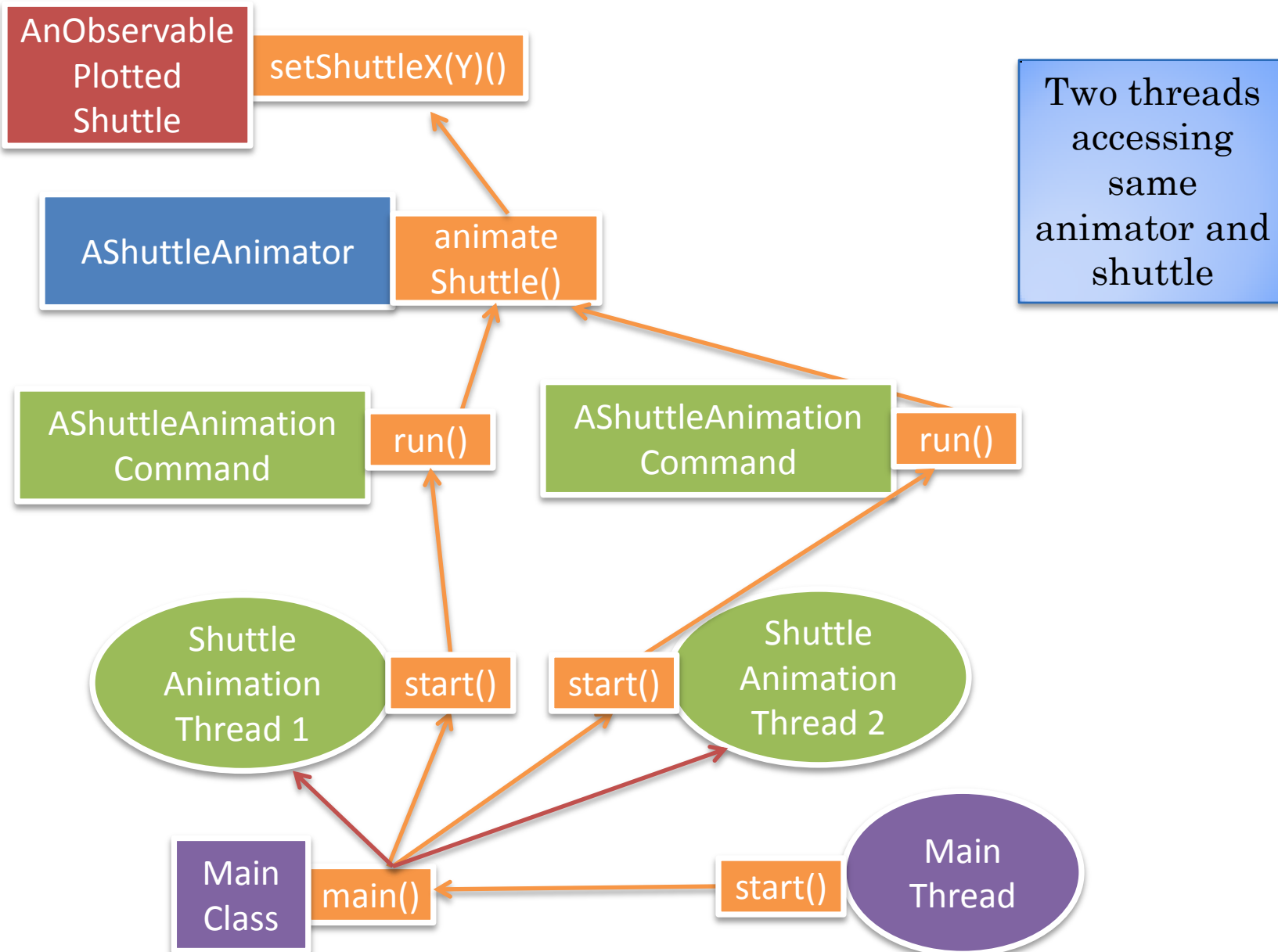
Sharing Road ~ Sharing Computer



THE SEPARATE LANE SCENARIO



ONE SHUTTLE & ANIMATOR, TWO THREADS



TWO SHUTTLES AND ANIMATORS, TWO THREADS

```
public static void main(String[] args) {  
    PlottedShuttle shuttle1 = new AnObservablePlottedShuttle(50, 100);  
    OEFFrame oeFrame1 = ObjectEditor.edit(shuttle1);  
    oeFrame1.hideMainPanel();  
    oeFrame1.setLocation(0, 0);  
    oeFrame1.setSize(400, 400);  
    PlottedShuttle shuttle2 = new AnObservablePlottedShuttle(100, 50);  
    OEFFrame 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



ONE SHUTTLES & ANIMATOR, TWO THREADS

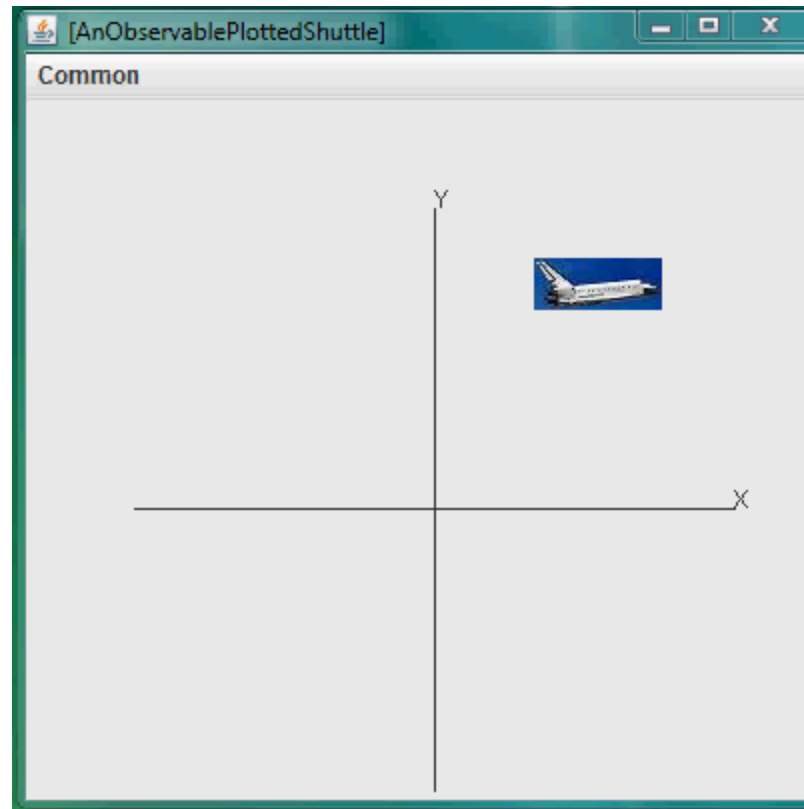
```
public static void main(String[] args) {  
    PlottedShuttle shuttle1 = new AnObservablePlottedShuttle(50, 100);  
    OEFrame oeFrame1 = ObjectEditor.edit(shuttle1);  
    oeFrame1.setLocation(0, 0);  
    oeFrame1.setSize(500, 550);  
    ShuttleAnimator aShuttleAnimator = new AShuttleAnimator();  
    concurrentDemoShuttleAnimation(aShuttleAnimator, shuttle1);  
    ThreadSupport.sleep(500);  
    concurrentDemoShuttleAnimation(aShuttleAnimator, shuttle1);  
}
```

Each thread
manipulates the
same shuttle


Second thread starts with
a 500 ms delay




INTERFERING ANIMATIONS



INTERFERENCE



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




```
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



TOP THREAD SETS ITS CURY




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




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



TOP THREAD SLEEPS




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




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



BOTTOM THREAD SETS ITS CURY



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



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



INTERFERENCE

The screenshot displays an IDE interface with a thread list at the top and a code editor at the bottom. The thread list on the left shows several threads, including 'Thread [Shuttle Animation 1]' and 'Thread [Shuttle Animation 2]', both suspended at breakpoints in 'AShuttleAnimator.animateFromOrigin'. The main pane shows the code for 'ShuttleAnimationDriver', with the line 'shuttle.setShuttleY(curY);' highlighted in green. The right pane shows the code for 'AConcurrentShuttleAnimator', with the line 'shuttle.setShuttleY(curY);' also highlighted in green. The code in the right pane includes a while loop that updates the shuttle's Y position and prints an animation message.

Thread [Shuttle Animation 1] (Suspended (breakpoint at line 22 in AShuttleAnimator))

- AShuttleAnimator.animateFromOrigin(PlottedShuttle, int, int) line: 22
- AShuttleAnimationCommand.run() line: 22
- Thread.run() line: not available

Thread [DestroyJavaVM] (Running)

Thread [Shuttle Animation 2] (Suspended (breakpoint at line 16 in AShuttleAnimator))

- AShuttleAnimator.animateFromOrigin(PlottedShuttle, int, int) line: 16
- AShuttleAnimationCommand.run() line: 22

ShuttleAnimationDriver

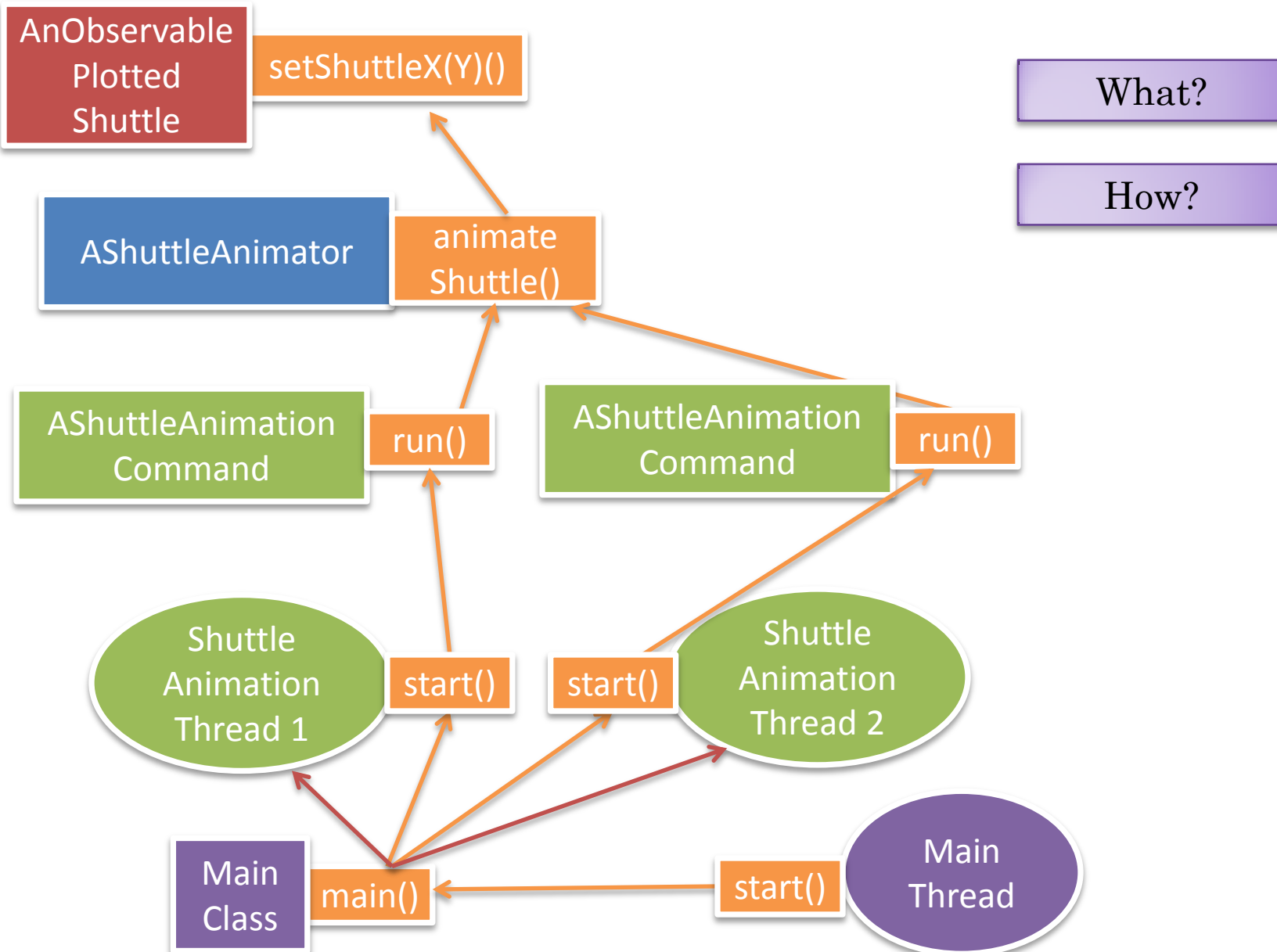
```
shuttle.setShuttleX(curX);
shuttle.setShuttleY(curY);
// make sure we don't go past final Y position
System.out.println("Animation in Y Direction");
while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}
```

AConcurrentShuttleAnimator

```
shuttle.setShuttleX(curX);
shuttle.setShuttleY(curY);
// make sure we don't go past final Y position
System.out.println("Animation in Y Direction");
while (curY < originalY) {
    ThreadSupport.sleep(animationPauseTime);
    curY += animationStep;
    shuttle.setShuttleY(curY);
}
```

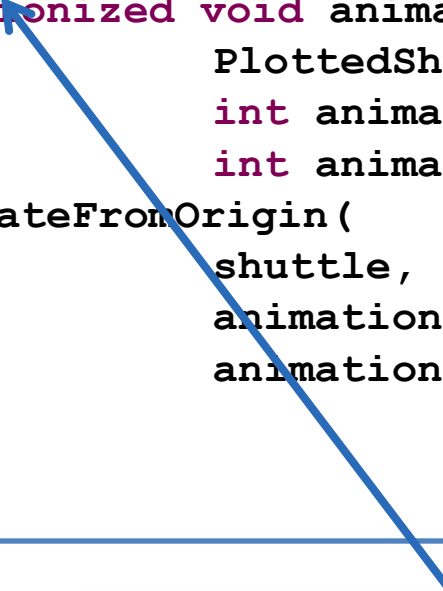


SOLUTION?



SYNCHRONIZED VERSION

```
public class ASynchronizedShuttleAnimator
    extends AShuttleAnimator {
    public synchronized void animateFromOrigin(
        PlottedShuttle shuttle,
        int animationStep,
        int animationPauseTime) {
        super.animateFromOrigin(
            shuttle,
            animationStep,
            animationPauseTime);
    }
}
```



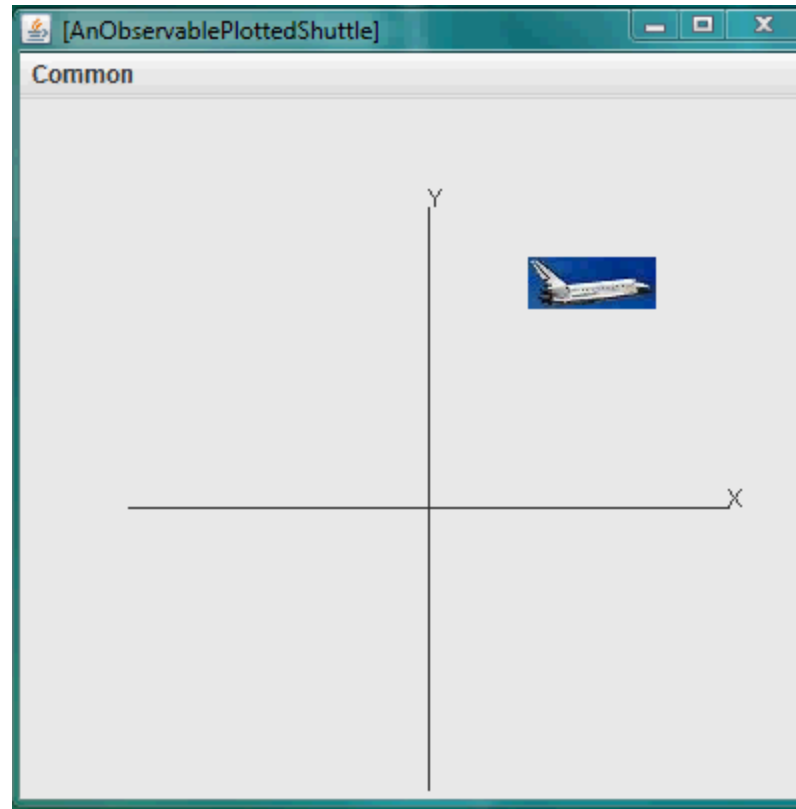
Should use keyword synchronized in method with shared data to tell Java that only one thread should execute the method at one time

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

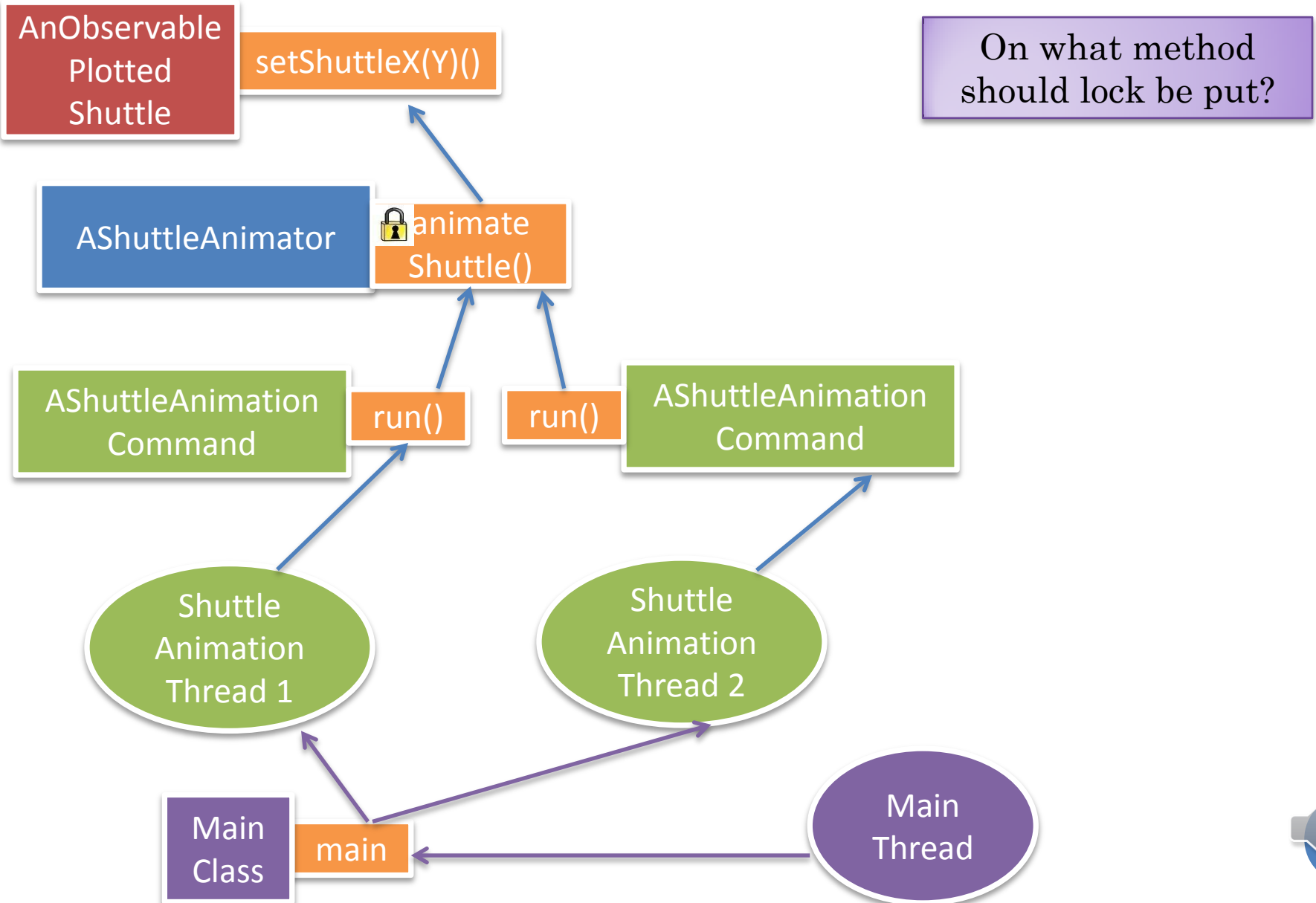
When a method is locked by a thread other threads wait in a queue, and when a method is unlocked the first waiting thread executes it



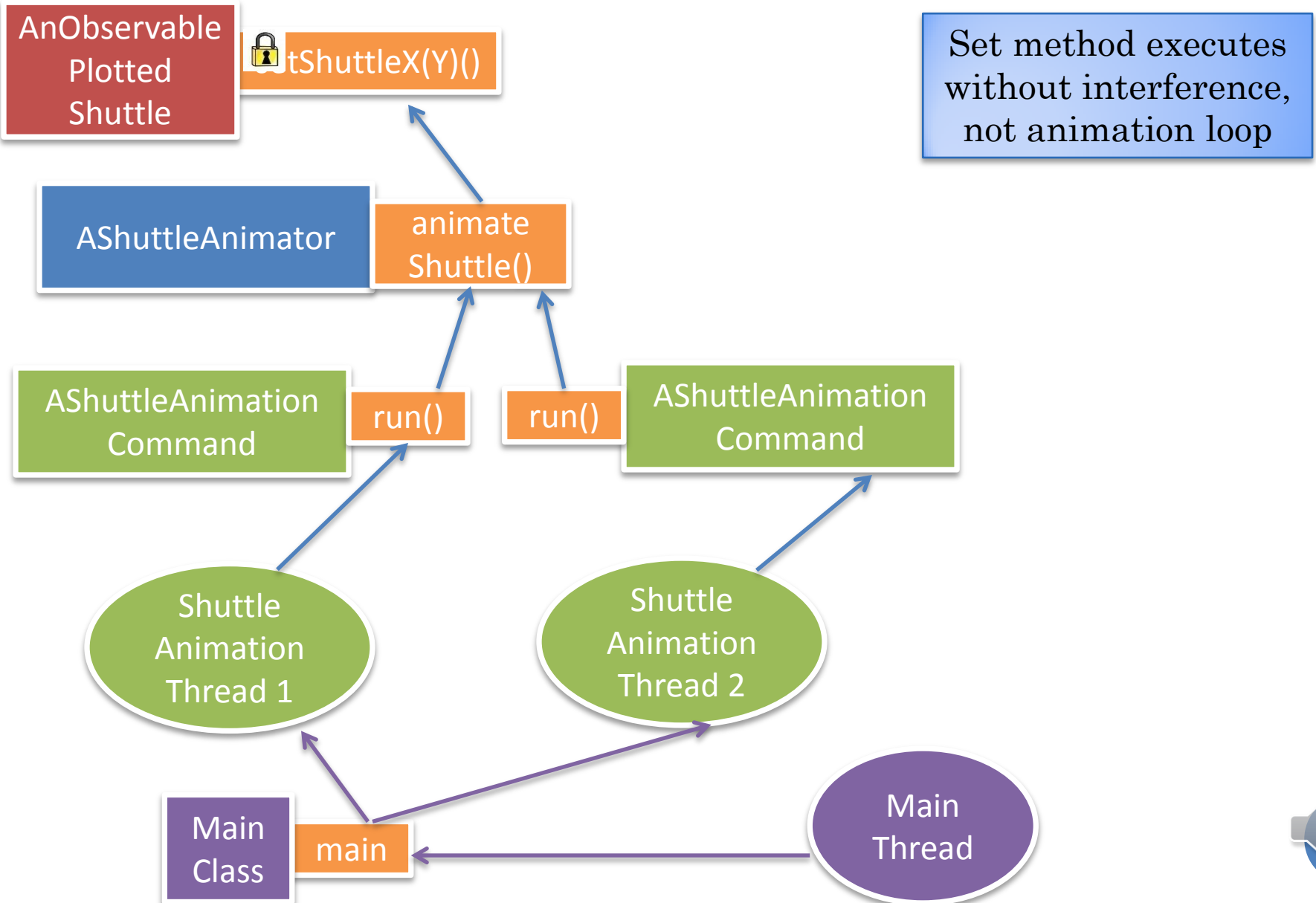
SYNCHRONIZED



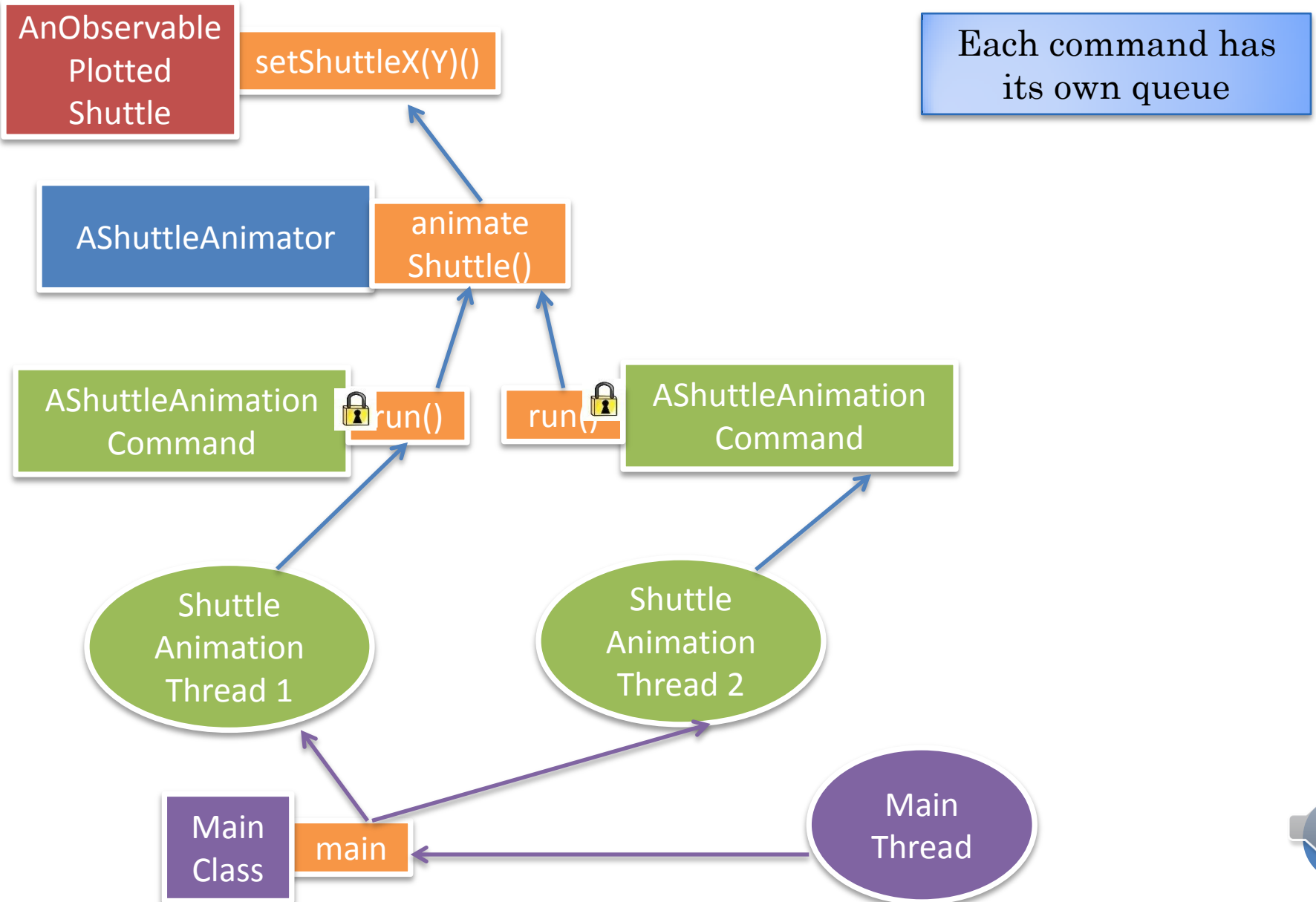
ONE SHUTTLE AND SYNCHRONIZED ANIMATOR, TWO THREADS



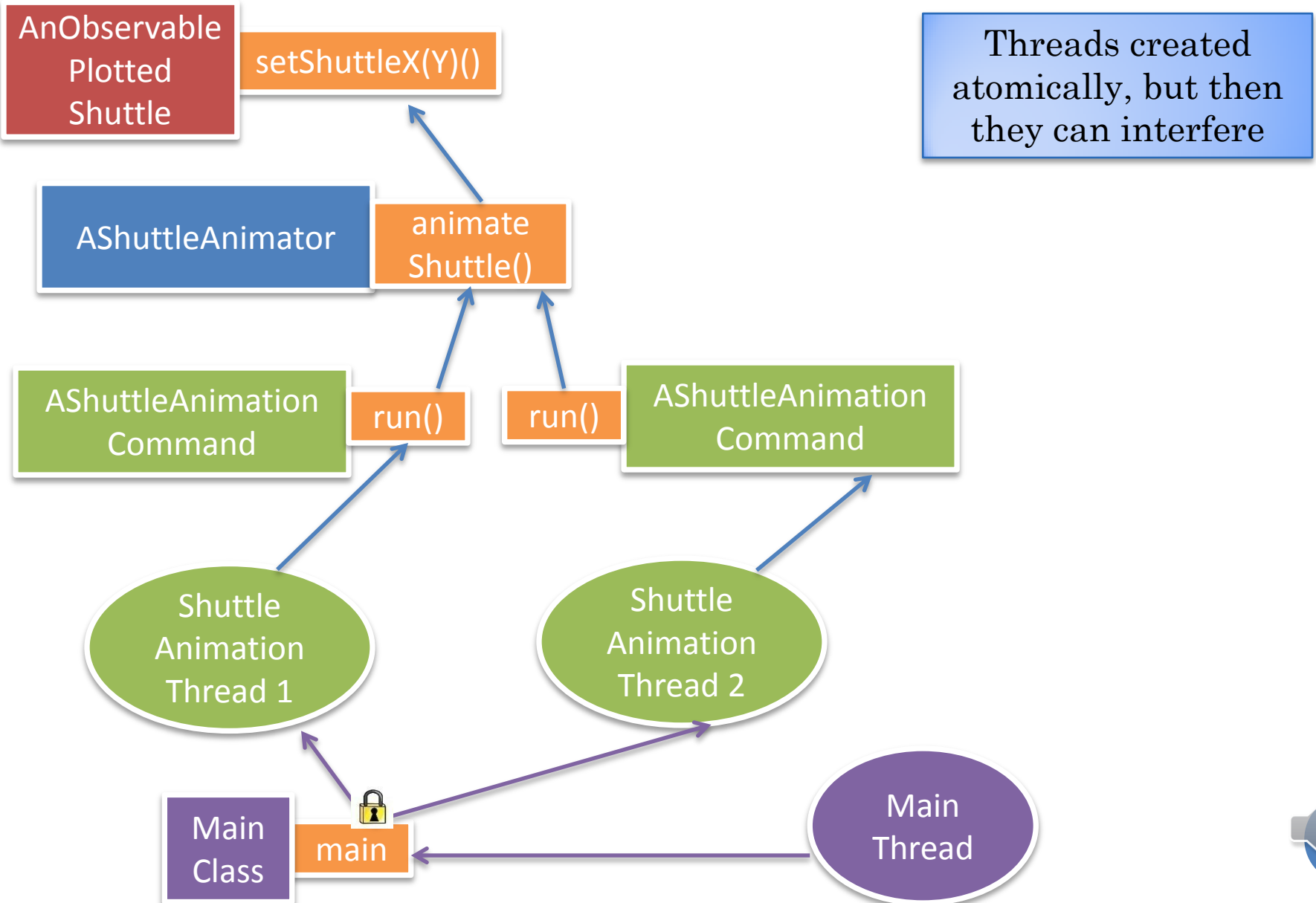
SYNCHRONIZE SET METHODS OF SHUTTLE?



SYNCHRONIZE RUN METHODS OF COMMAND?



SYNCHRONIZE THREAD CREATION?



SYNCHRONIZE THREAD CREATOR?

```
public synchronized static void concurrentDemoShuttleAnimation(  
    ShuttleAnimator aShuttleAnimator, PlottedShuttle aShuttle) {  
    Thread thread = new Thread(this, animateFromOriginMethod,  
                                new Object[] {aShuttle, 5, 100});  
  
    threadNumber++;  
    thread.setName(SHUTTLE_THREAD_NAME + " " + threadNumber);  
    thread.start();  
}
```

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

```
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, OEFram 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

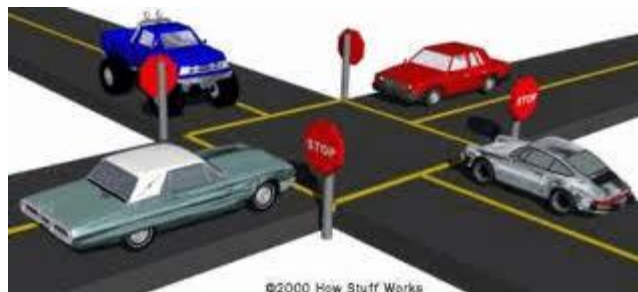
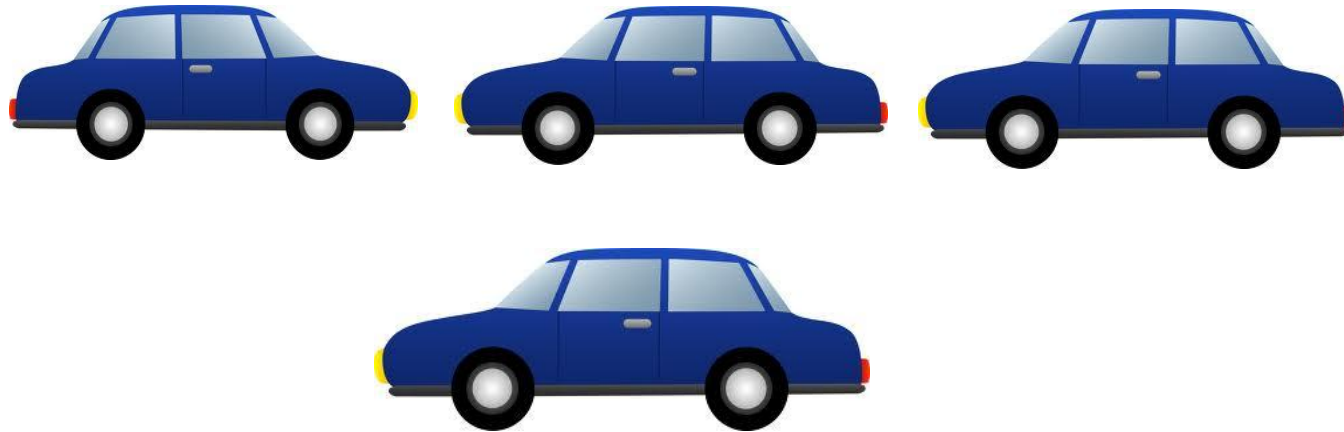
if some synchronized method is
executing in O, then makes calling
thread wait in O's queue

Finish call to synchronized
method on object O

Unblocks first waiting thread in O's
queue



SYNCHRONIZED METHOD ANALOGY?



ONE SHUTTLE, TWO SYNCHRONIZED ANIMATORS

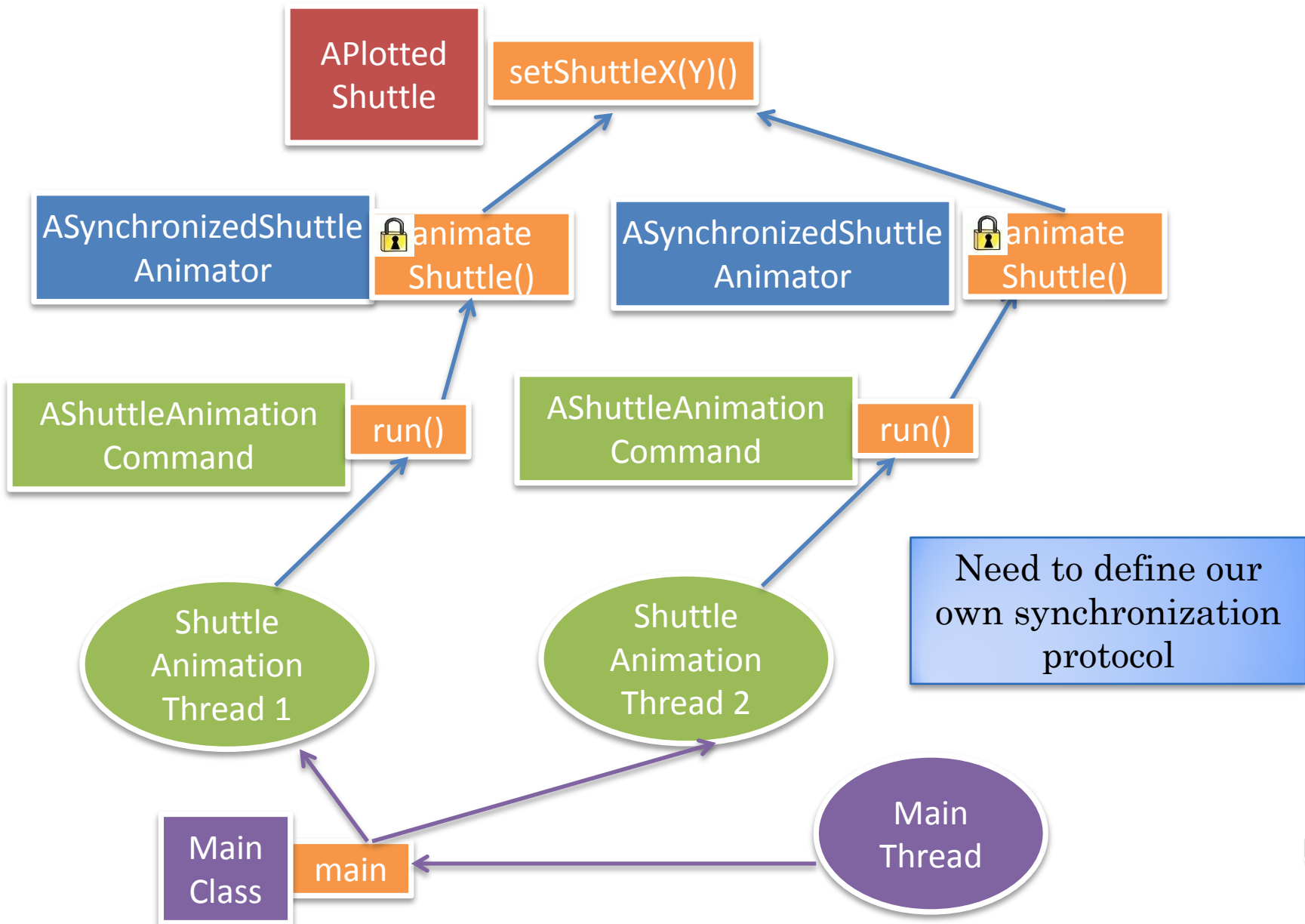
```
public class OneShuttleTwoSynchronizedShuttleAnimators
    extends ConcurrentShuttleAnimationDriver {
    public static void main(String[] args) {
        PlottedShuttle shuttle1 =
            new AnObservablePlottedShuttle(50, 100);
        OEFrmae oeFrame = ObjectEditor.edit(shuttle1);
        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



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.

