



# **COMP 401**

## **USER-INTERFACE VS. MAIN THREADS**

**Instructor: Prasun Dewan**



# PREREQUISITE

- Animation Threads Commands



# ANIMATIONS FROM MAIN

```
public static void main(String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize (450, 450);  
    ShuttleAnimator shuttleAnimator = new AShuttleAnimator();  
    shuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

No thread as single animation



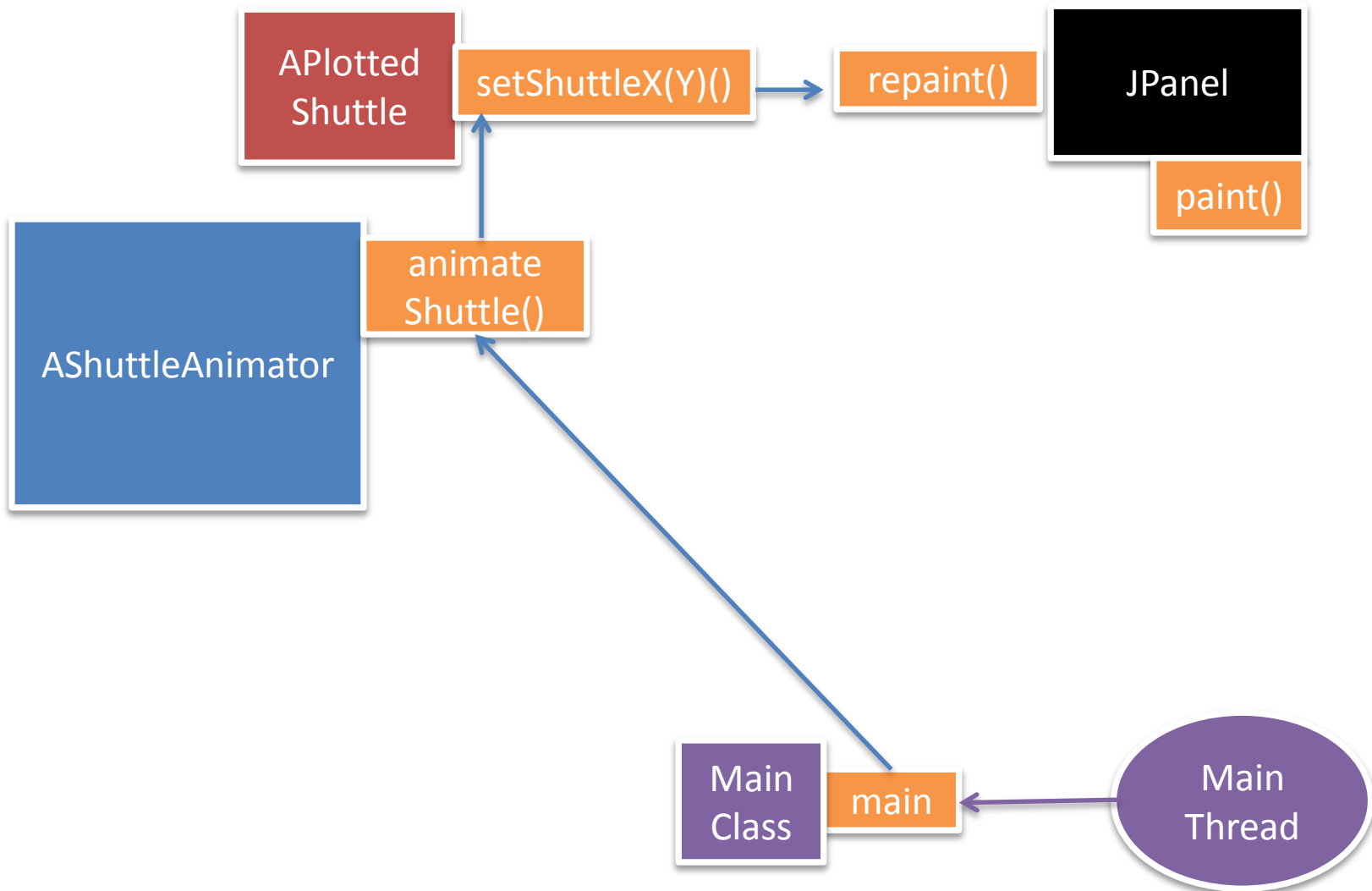
# ANIMATIONS FROM MAIN

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

Threads created, as multiple independent  
animations wanted



# SINGLE ANIMATION FROM MAIN: NO SPECIAL THREAD



# CONSIDER SINGLE ANIMATION

```
public static void main(String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize (450, 450);  
    ShuttleAnimator shuttleAnimator = new AShuttleAnimator();  
    shuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

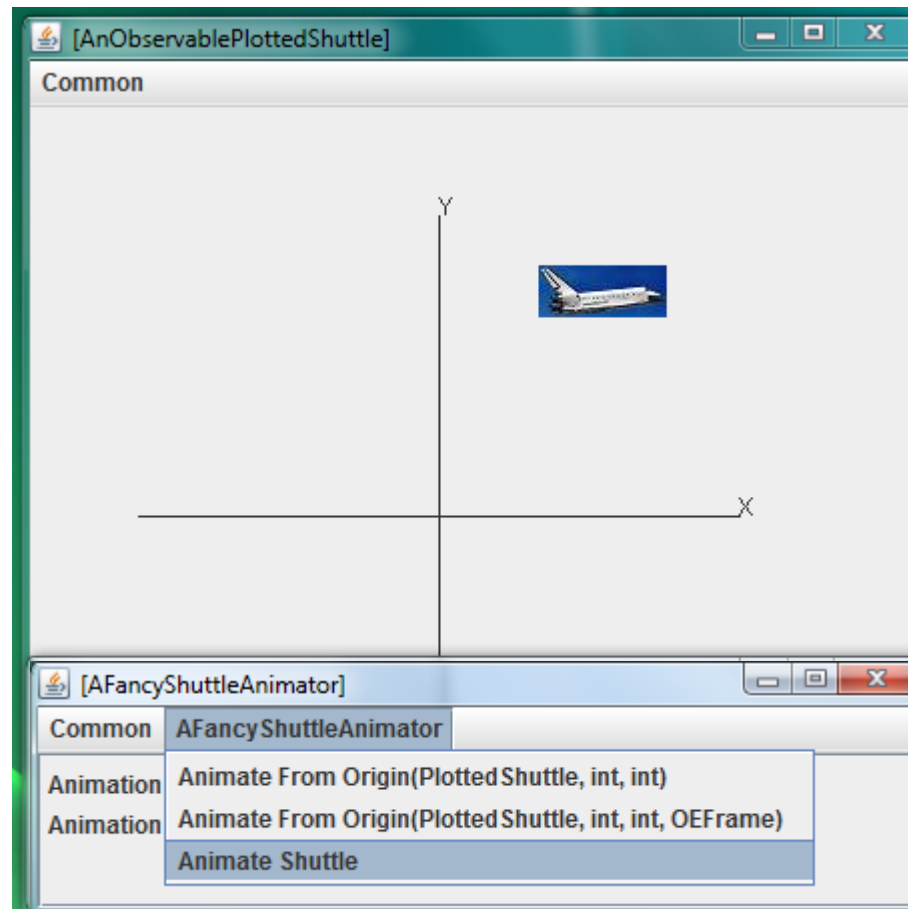
Start animation from the user interface?

Extension of ShuttleAnimator that allows  
parameters to be properties

We can edit these properties interactively and  
start animation with them as parameters



# GUI



# MAIN VS. INTERACTIVE ANIMATION

```
public static void main(String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize (450, 450);  
    ShuttleAnimator shuttleAnimator = new AShuttleAnimator();  
    shuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

```
public static void main (String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize (450, 450);  
    FancyShuttleAnimator shuttleAnimator = new AFancyShuttleAnimator();  
    ObjectEditor.edit(shuttleAnimator);  
}
```





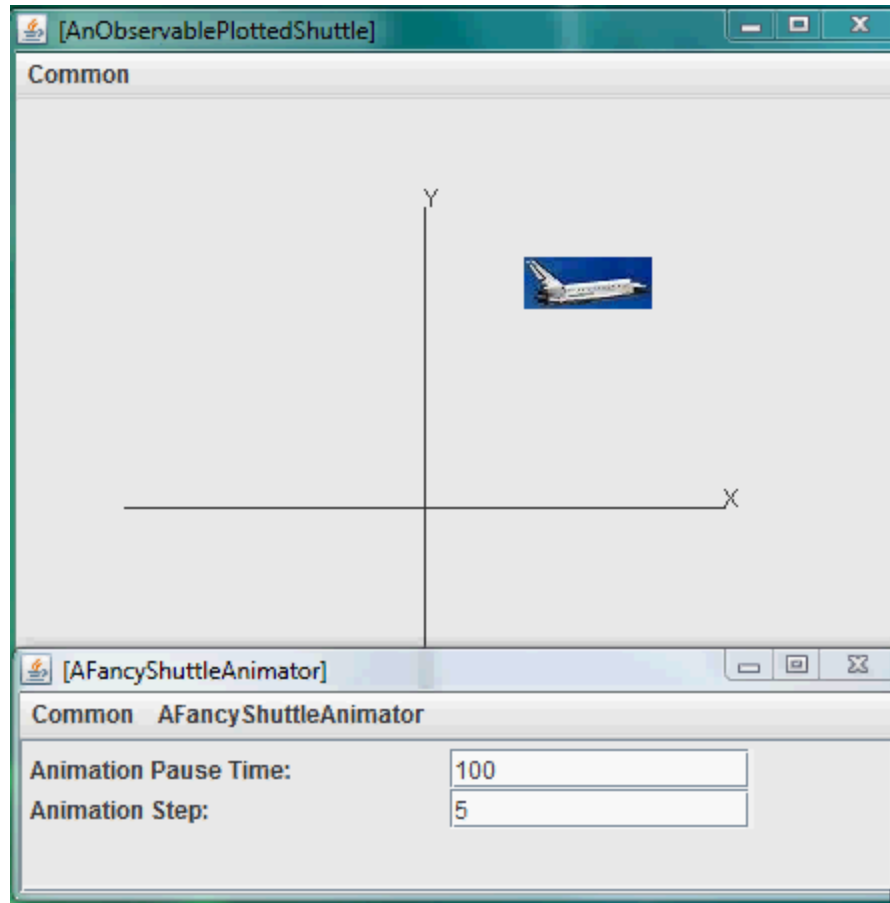
# FANCY ANIMATOR

```
public class AFancyShuttleAnimator extends AShuttleAnimator
    implements FancyShuttleAnimator {

    int animationStep = 5;
    int animationPauseTime = 100;
    PlottedShuttle shuttle;
    public AFancyShuttleAnimator(PlottedShuttle theShuttle) {
        shuttle = theShuttle;
    }
    public int getAnimationStep() {
        return animationStep;
    }
    public void setAnimationStep(int animationStep) {
        this.animationStep = animationStep;
    }
    public int getAnimationPauseTime() {
        return animationPauseTime;
    }
    public void setAnimationPauseTime(int animationPauseTime) {
        this.animationPauseTime = animationPauseTime;
    }
    public void animateShuttle() {
        animateFromOrigin(shuttle, animationStep, animationPauseTime);
    }
}
```



# VIDEO



# WHEN DOES MAIN TERMINATE?

```
public static void main(String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize(450, 400);  
    ShuttleAnimator shuttleAnimator = new AShuttleAnimator(shuttle);  
    shuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

Main thread executes loop

Main thread terminates

```
public static void main (String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize(450, 400);  
    FancyShuttleAnimator shuttleAnimator = new FancyShuttleAnimator(shuttle);  
    ObjectEditor.edit(shuttleAnimator);  
}
```

Main thread starts UI

Main thread terminates

UI Thread (created by Java)  
executes loop



# CONSIDER SINGLE ANIMATION

## ShuttleAnimationDriver [Java Application]

lectures.animation.loops.ShuttleAnimationDriver at localhost:58863

Thread [main] (Suspended (breakpoint at line 16 in AS

AShuttleAnimator.animateFromOrigin(PlottedS

ShuttleAnimationDriver.main(String[]) line: 20

Thread [AWT-Shutdown] (Running)

Daemon Thread [AWT-Windows] (Running)

Thread [AWT-EventQueue-0] (Running)

Thread [Tool Tip Thread] (Running)

Main thread  
executes loop

## AFancyShuttleAnimator [Java Application]

lectures.animation.threads.AFancyShuttleAnimator at localhost:58579

Thread [AWT-Shutdown] (Running)

Daemon Thread [AWT-Windows] (Running)

Thread [AWT-EventQueue-0] (Suspended (breakpoint in

AFancyShuttleAnimator(AShuttleAnimator).animate

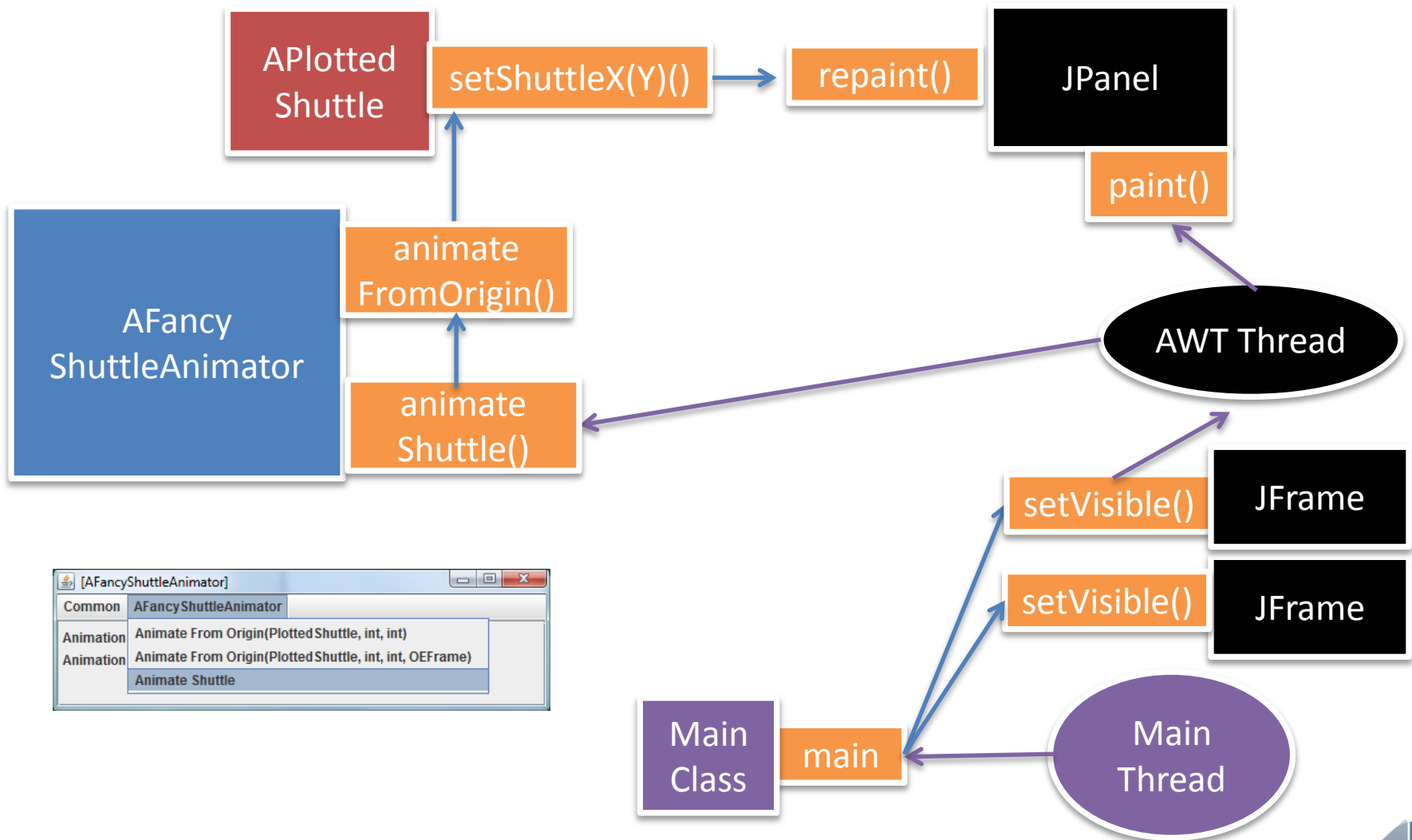
AFancyShuttleAnimator.animateShuttle() line: 32

Main thread starts  
UI and terminates

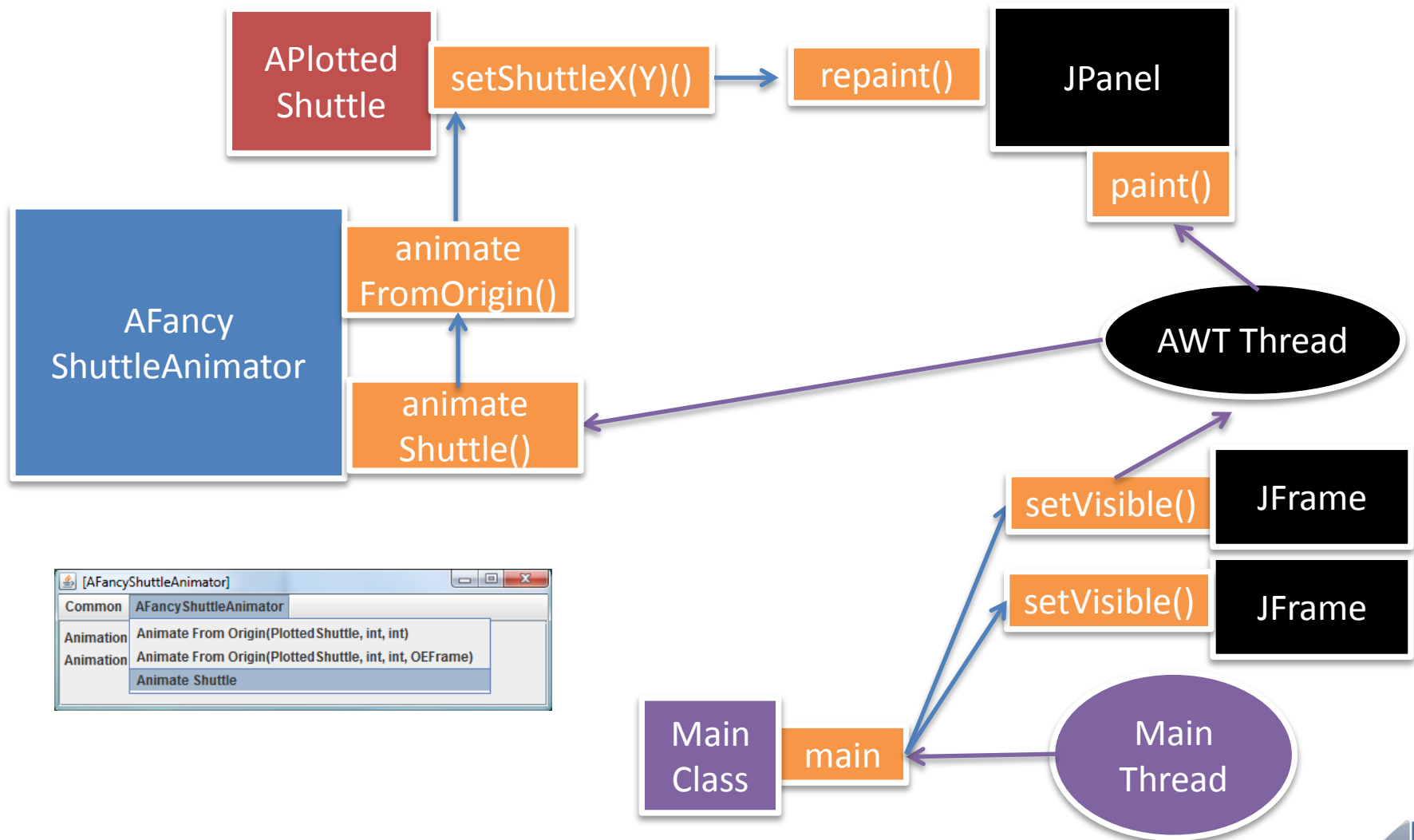
UI Thread (created by  
Java) executes loop



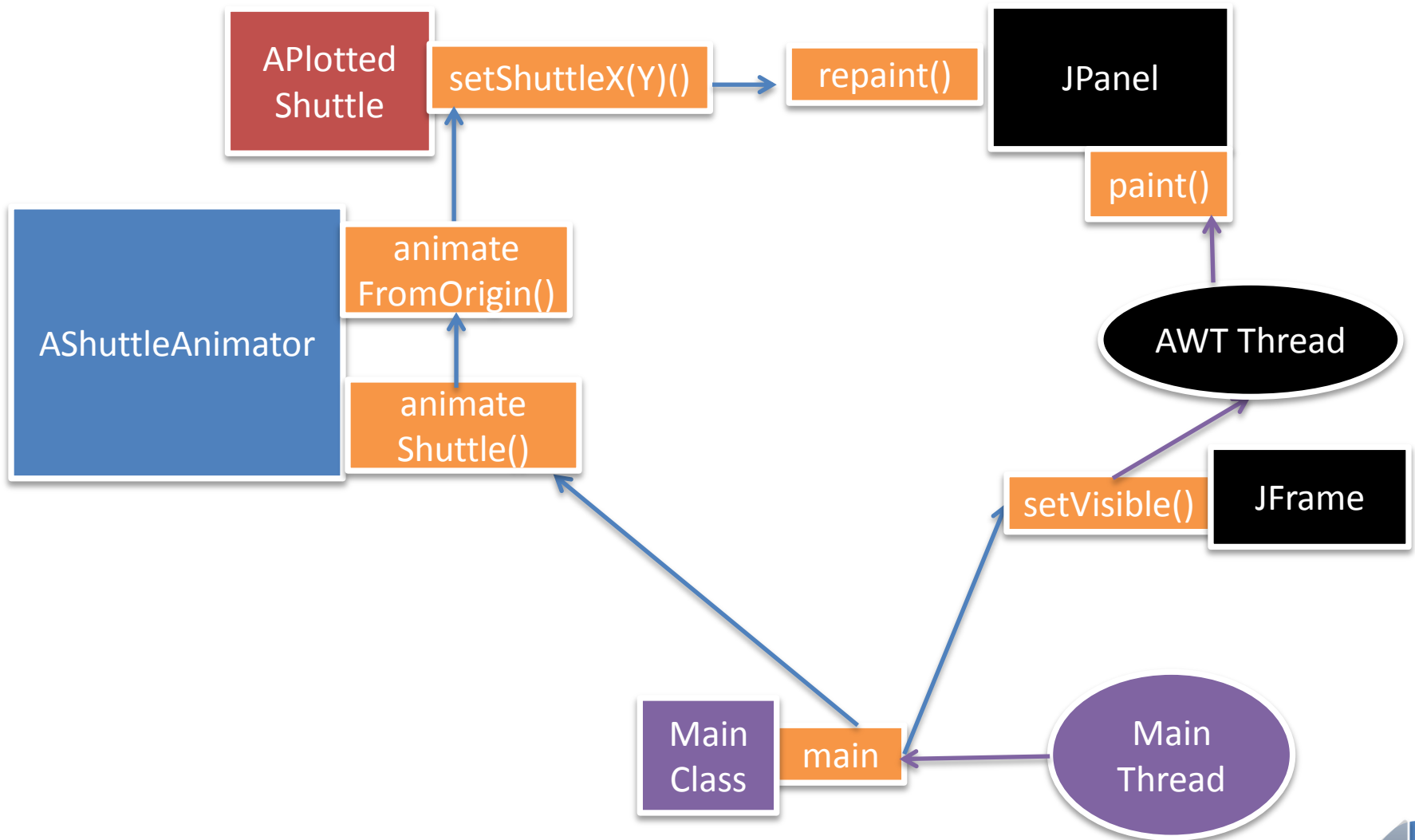
# INTERACTIVE ANIMATION: NO SPECIAL THREAD



# INTERACTIVE ANIMATION: NO SPECIAL THREAD (REVIEW)




# SINGLE ANIMATION FROM MAIN: NO SPECIAL THREAD



# INTERLEAVING WITH UI THREAD


```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitAndProcessNextQueuedUIEvent();  
}
```





# LOOP EXECUTES

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitForAndProcessNextQueuedUIEvent();  
}
```



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



# LOOPING THREAD UPDATES SHUTTLE AND ENQUEUES REPAINT EVENT

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitForAndProcessNextQueuedUIEvent();  
}
```


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

```
//in Java component showing  
//the shuttle  
public void repaint() {  
    enqueueRepaintEvent(this );  
}
```



# LOOP RE-EXECUTES

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitForAndProcessNextQueuedUIEvent();  
}
```



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



# LOOPING THREAD UPDATES SHUTTLE AND ENQUEUES ANOTHER REPAINT EVENT

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitForAndProcessNextQueuedUIEvent();  
}
```


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

```
//in Java compoent showing  
//the shuttle  
public void repaint() {  
    enqueueRepaintEvent(this );  
}
```



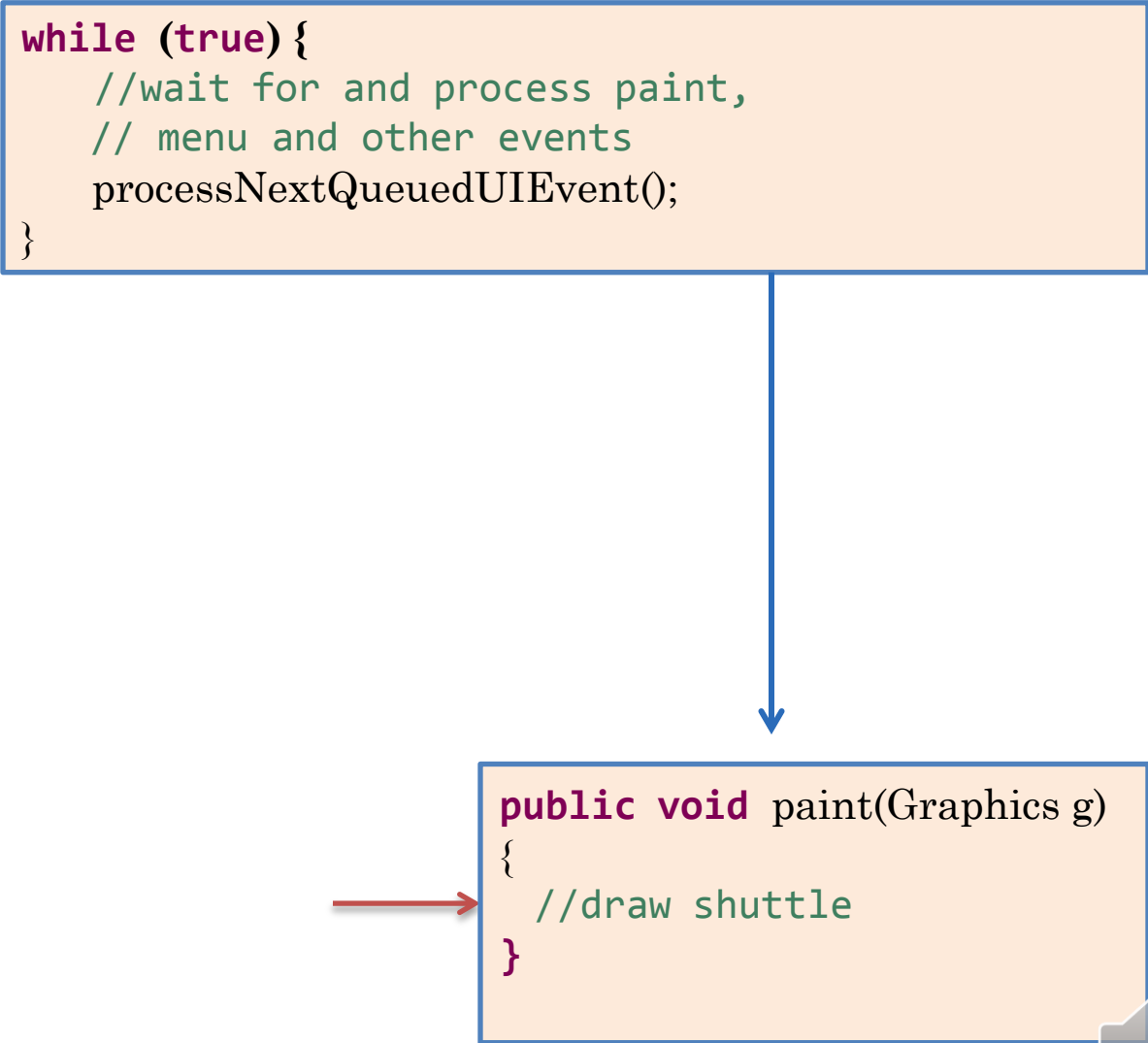
# LOOP FINISHES

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    processNextQueuedUIEvent();  
}
```



# UI THREAD PROCESSES ENQUEUED PAINTS

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    processNextQueuedUIEvent();  
}
```




```
public void paint(Graphics g)  
{  
    //draw shuttle  
}
```

The diagram illustrates the flow of execution in a UI thread. A blue arrow points from the `processNextQueuedUIEvent()` call in the `while (true)` loop to the `paint` method. A red arrow points to the `paint` method from the left.



# UI THREAD WAITS FOR NEXT EVENT



```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitForAndProcessNextQueuedUIEvent();  
}
```



# UI EVENT LOOP AND ANIMATIONS

AWT Thread

```
while (true) {  
    //wait for and process paint,  
    // menu and other events  
    waitAndProcessNextQueuedUIEvent(),  
}
```

Event Loop

Mouse Click

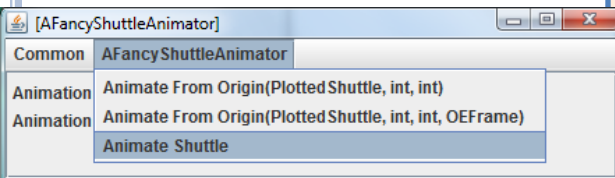
Mouse Click

Repaint

Repaint

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

Listener Code



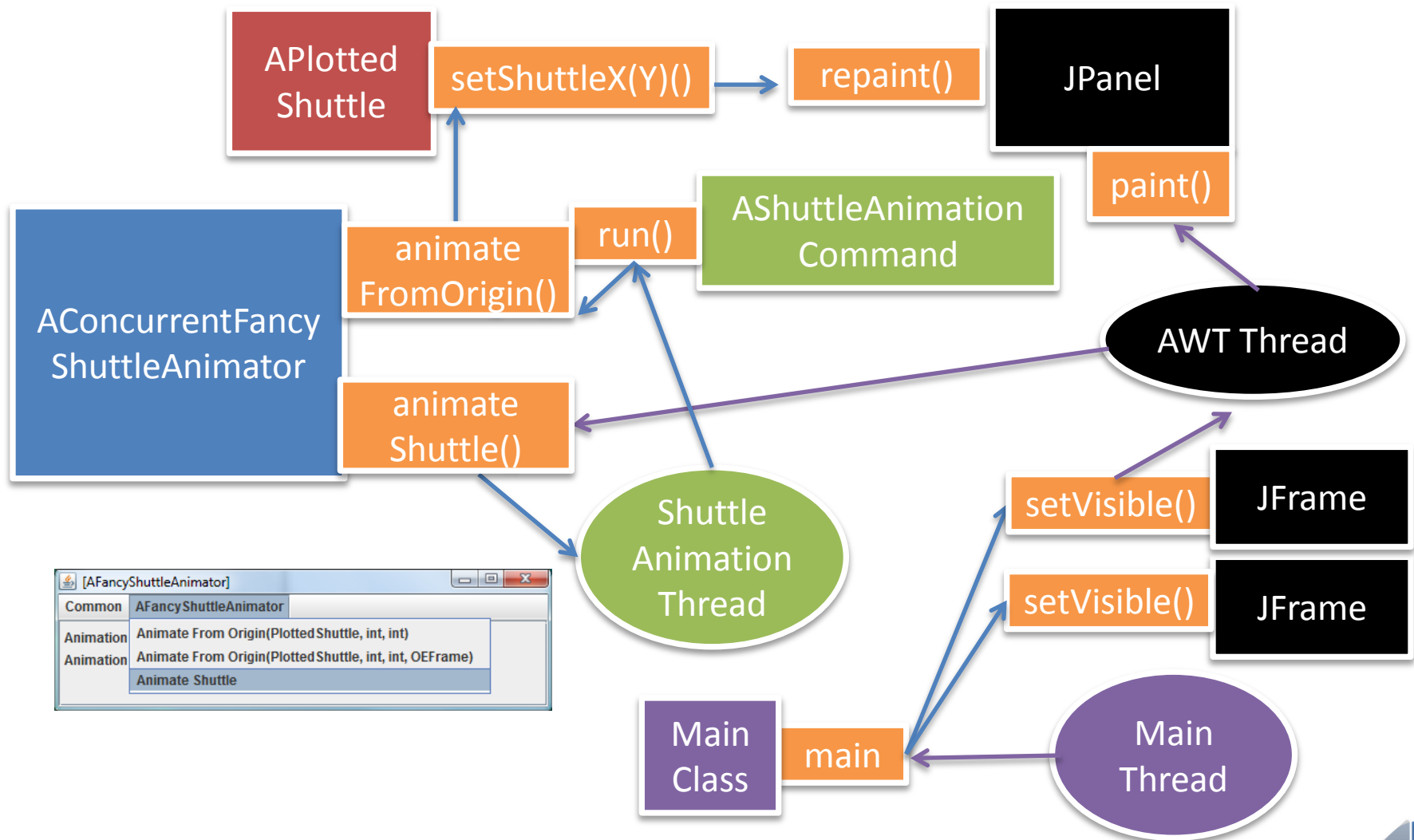
New UI event not processed until  
listeners for previous event finish

Animating listener should create new  
thread for animation code





# INTERACTIVE ANIMATION: SPECIAL THREAD



# CONCURRENT FANCY ANIMATOR

```
public class AConcurrentShuttleAnimator extends
AFancyShuttleAnimator {
    public AConcurrentShuttleAnimator(PlottedShuttle theShuttle) {
        super(theShuttle);
    }
    public void animateShuttle() {
        Thread thread =
            new Thread(
                (new AShuttleAnimationCommand(
                    this, shuttle, animationStep,
                    animationPauseTime)));
        thread.start();
    }
}
```

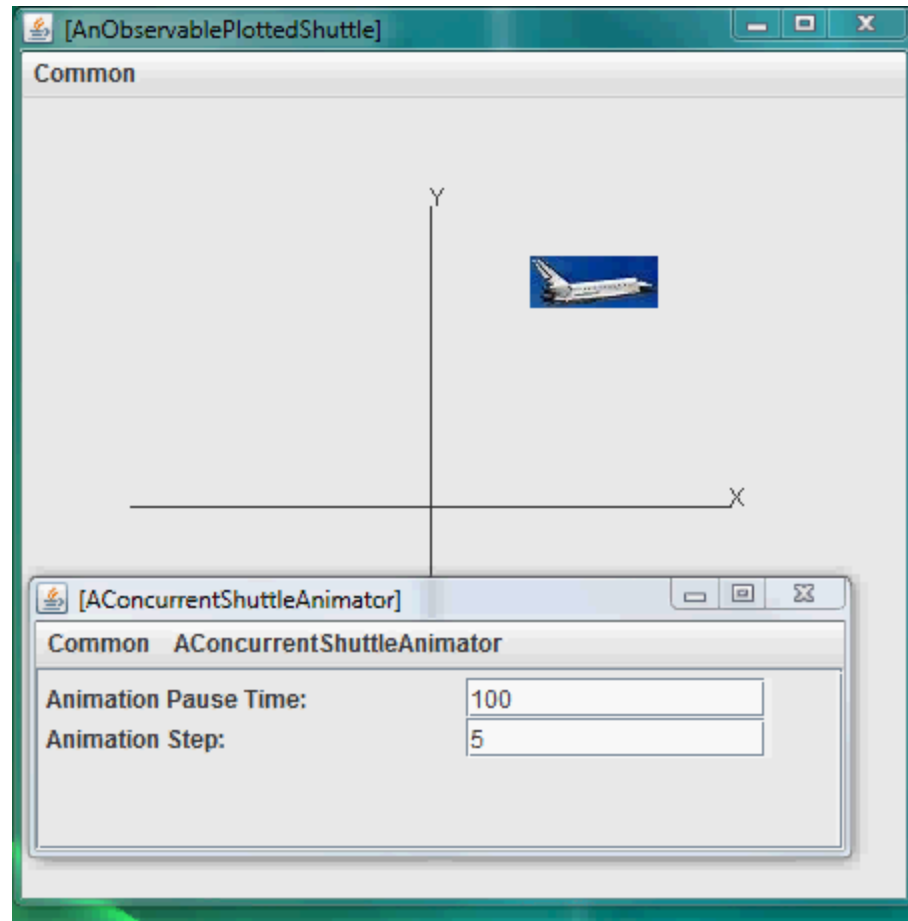


# STARTING CONCURRENT FANCY ANIMATOR

```
public static void main (String[] args) {  
    PlottedShuttle shuttle = new APlottedShuttle(50, 100);  
    OEFrame oeFrame = ObjectEditor.edit(shuttle);  
    oeFrame.hideMainPanel();  
    oeFrame.setSize (450, 450);  
    FancyShuttleAnimator shuttleAnimator =  
        new AConcurrentShuttleAnimator();  
    ObjectEditor.edit(shuttleAnimator);  
}
```



# VIDEO



The animation  
method is  
synchronized



# GUI PROCESSING

- Even if main thread terminates, the application continues to run as long as a GUI has been created, which creates the GUI thread.
- A single GUI thread is created for processing the controller (menu/button/... processing) and view (repaint) actions of all models.
- View updates cannot occur until controller returns.
- Controller action should result in a new thread if it starts an animation .
- If a single animation is started from main then no thread needs to be created a main thread executes loop and separate GUI thread updates view.

# REPAINT AND UI THREAD SEMANTICS

- When a thread (GUI or some other) calls repaint on a component C
  - It puts a repaint event for C in the GUI event queue if such an event is not already in the queue.
  - This queue contains all UI events such as mouse and key clicks.
- The GUI thread performs the following loop
  1. Waits for the GUI event queue to be non empty
  2. Removes and services the next event from the queue
    - If the next event is a repaint event for component C, it calls the update() method in C, passing it a graphics object. The update() method clears the component and calls paint() method.
    - If the next event is not a repaint() event, then it does some event-specific processing such as calling a listener for the event.
  3. Goes back to 1.

