



COMP 401

ANIMATION, THREADS, COMMAND OBJECTS

Instructor: Prasun Dewan



PREREQUISITE

- Animation MVC

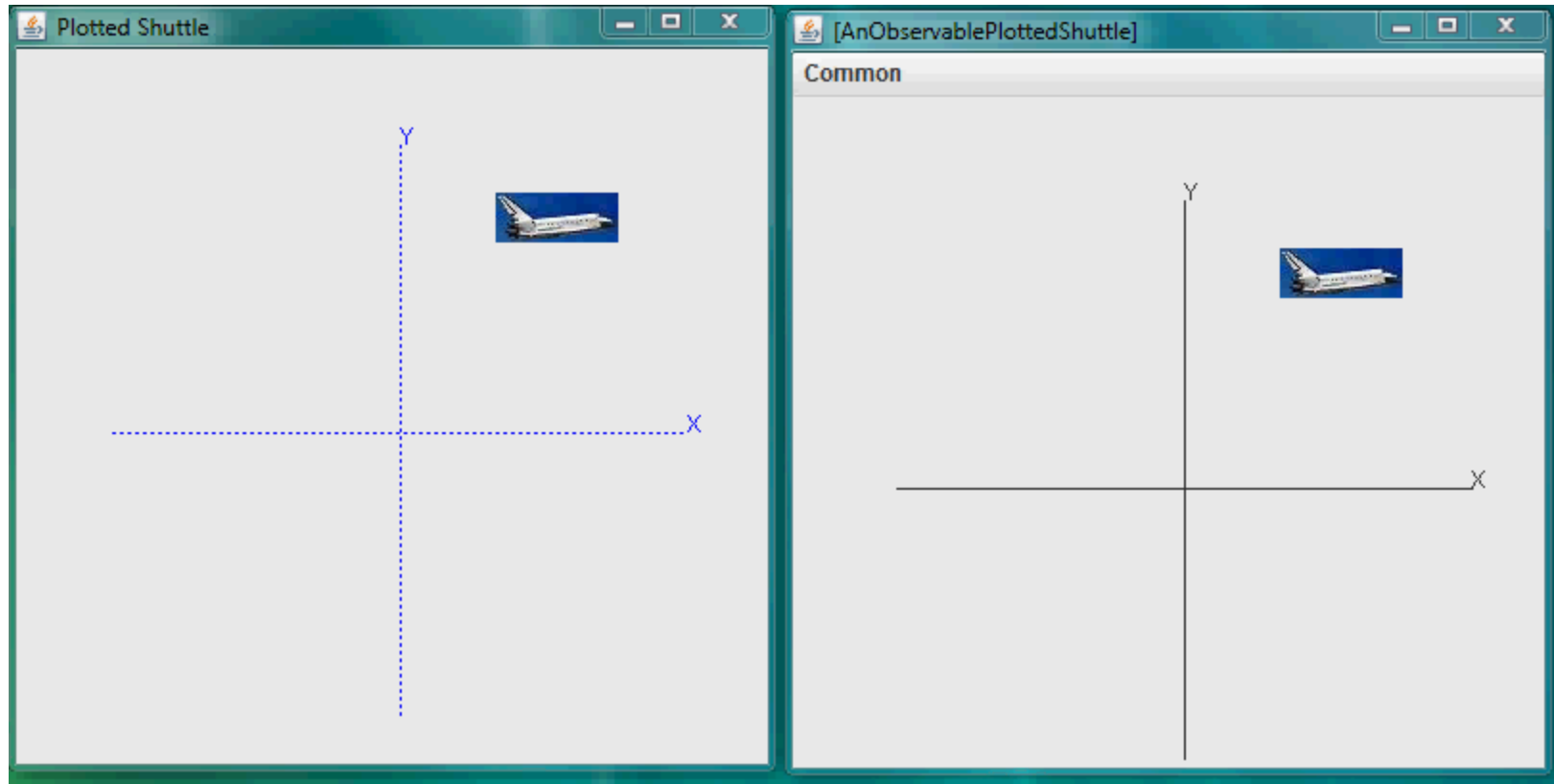


TOPICS

- Animation
- Command Object
 - Object representing an action invocation such as “Do your homework”.
- Threads
 - Support non blocking action invocation.



ANIMATION AND MVC



ANIMATEFROMORIGIN

```
public void animateFromOrigin(PlottedShuttle shuttle,  
    int animationStep, int animationPauseTime) {  
    int originalX = shuttle.getShuttleX();  
    int originalY = shuttle.getShuttleY();  
    int curX = 0;  
    int curY = 0;  
    shuttle.setShuttleX(curX);  
    shuttle.setShuttleY(curY);  
    animateYFromOrigin(shuttle, animationStep,  
        animationPauseTime, curY, originalY);  
    animateXFromOrigin(shuttle, animationStep,  
        animationPauseTime, curX, originalX);  
}
```

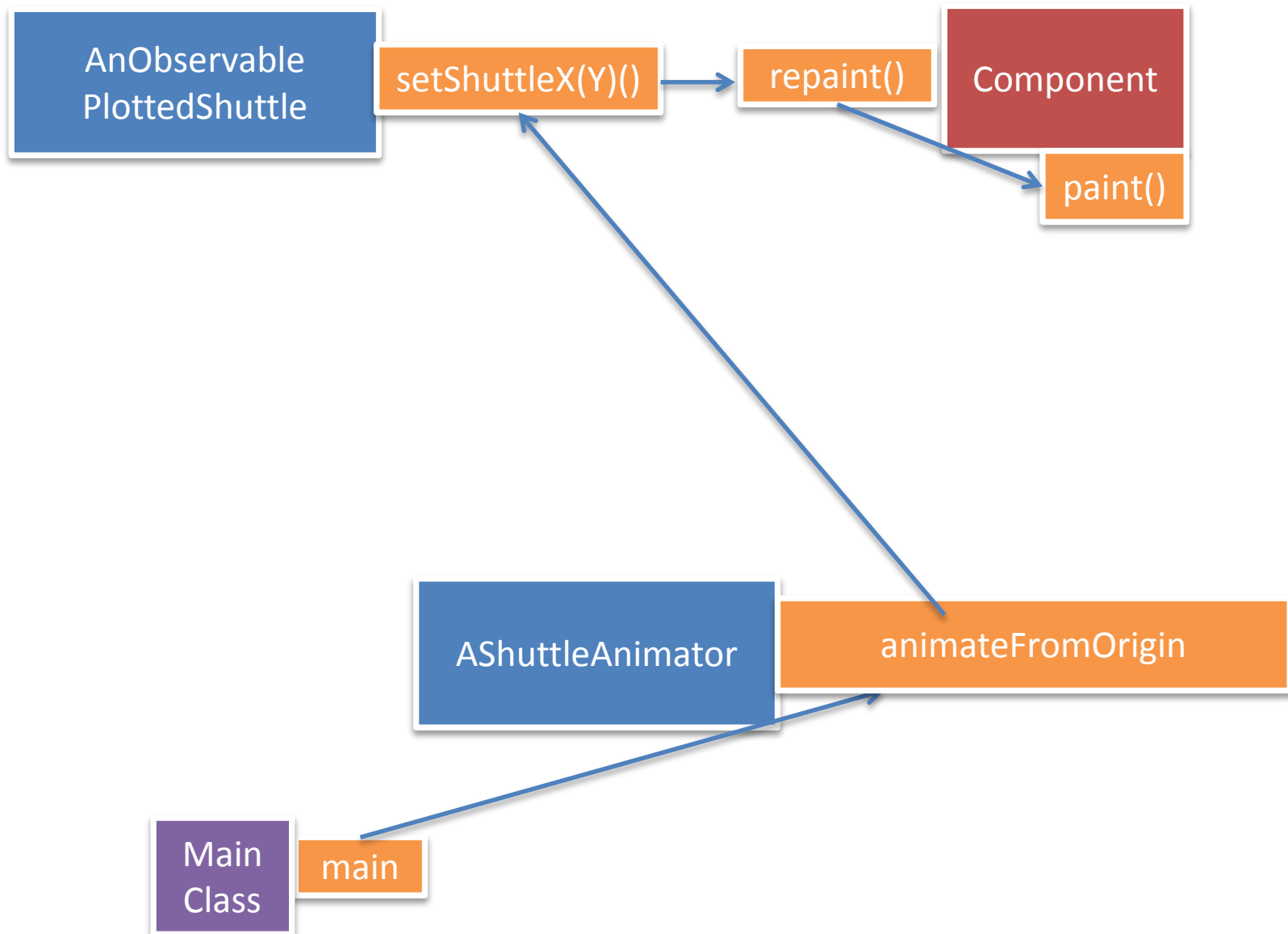


ANIMATION IN Y DIRECTION

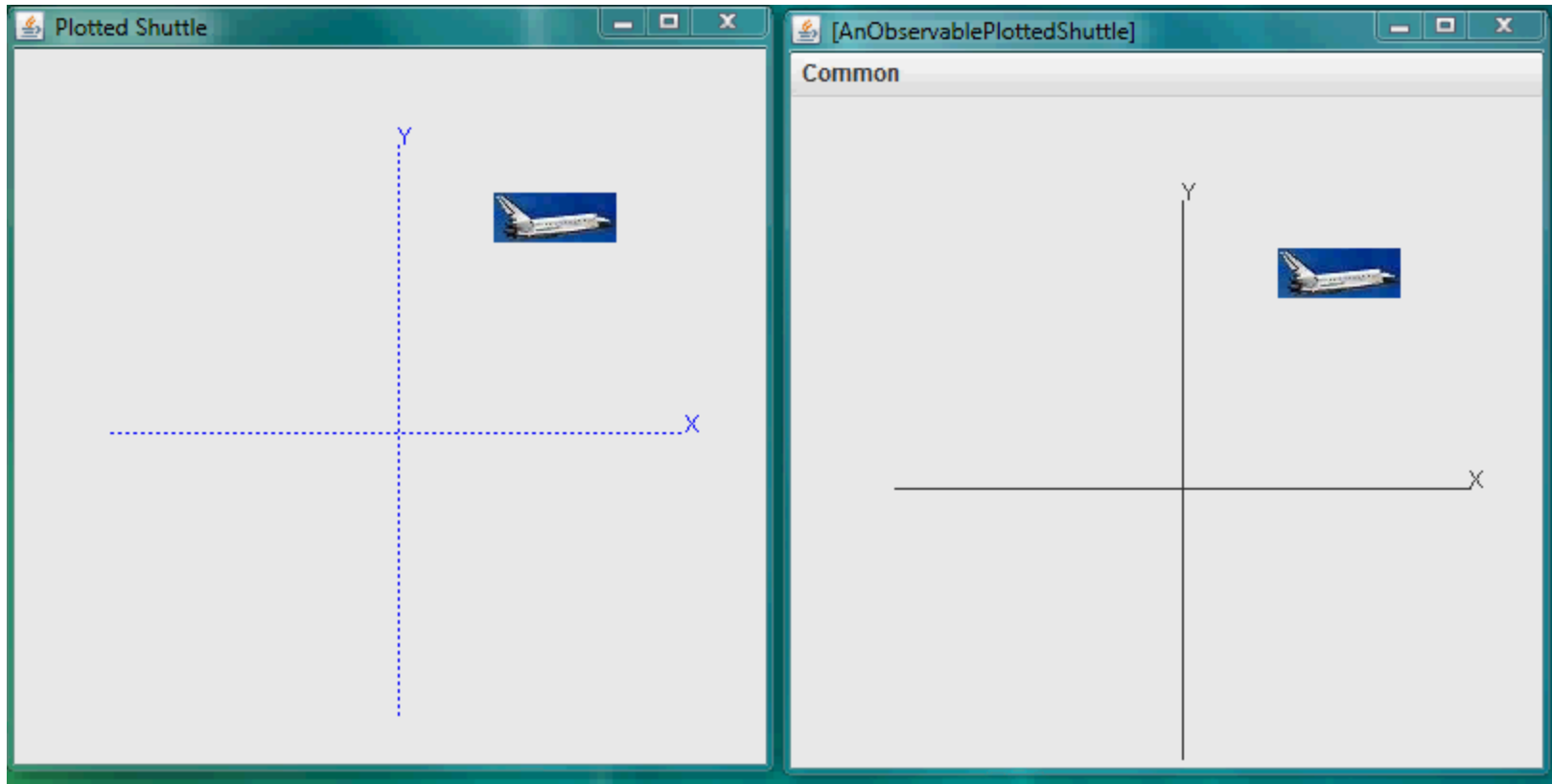
```
protected void animateYFromOrigin(PlottedShuttle shuttle,
                                   int animationStep, int animationPauseTime,
                                   int startY, int endY) {
    // make sure we don't go past final Y position
    while (startY < endY) {
        ThreadSupport.sleep(animationPauseTime);
        startY += animationStep;
        shuttle.setShuttleY(startY);
    }
    // move to destination Y position
    shuttle.setShuttleY(endY);
}
```



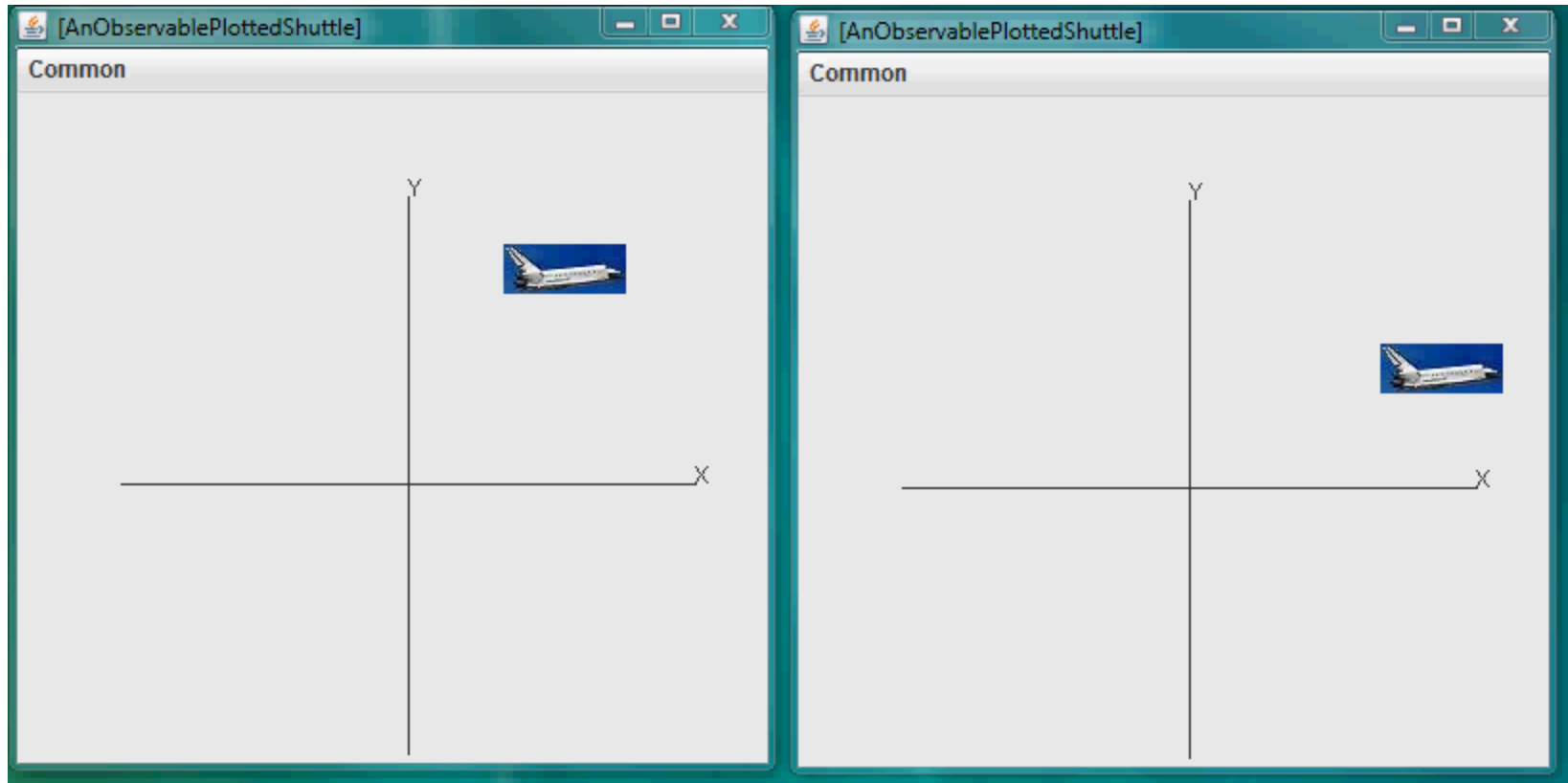
OBSERVABLE ARCHITECTURE



ANIMATION AND MVC



TWO INDEPENDENT ANIMATIONS



INDEPENDENT ANIMATIONS?

```
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();  
    shuttleAnimator1.animateFromOrigin(shuttle1, 5, 100);  
    shuttleAnimator2.animateFromOrigin(shuttle2, 5, 100);  
}
```

Second animation
waits for first one
to finish

Need multitasking




WHAT WE NEED

```
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 ShuttleAnimator();  
    ShuttleAnimator shuttleAnimator2 = new ShuttleAnimator();  
    concurrentDemoShuttleAnimation(shuttleAnimator1, shuttle1);  
    concurrentDemoShuttleAnimation(shuttleAnimator2, shuttle2);  
}
```


Somehow these two
methods should
interleave their
activities



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE




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




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



INTERLEAVING EXAMPLE



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



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



INTERLEAVING ACTIVITIES

- Each interleaved activity is associated with its own program counter, which marks the next statement to be executed for that activity.
- At any one time, a single CPU executes the statement of only one activity called the current activity.
- The CPU does not wait for the current activity to complete before switching to another activity.



INTERLEAVING ACTIVITIES IN REAL LIFE



Smile at
baby

Read
email

Feed dog

Feed baby

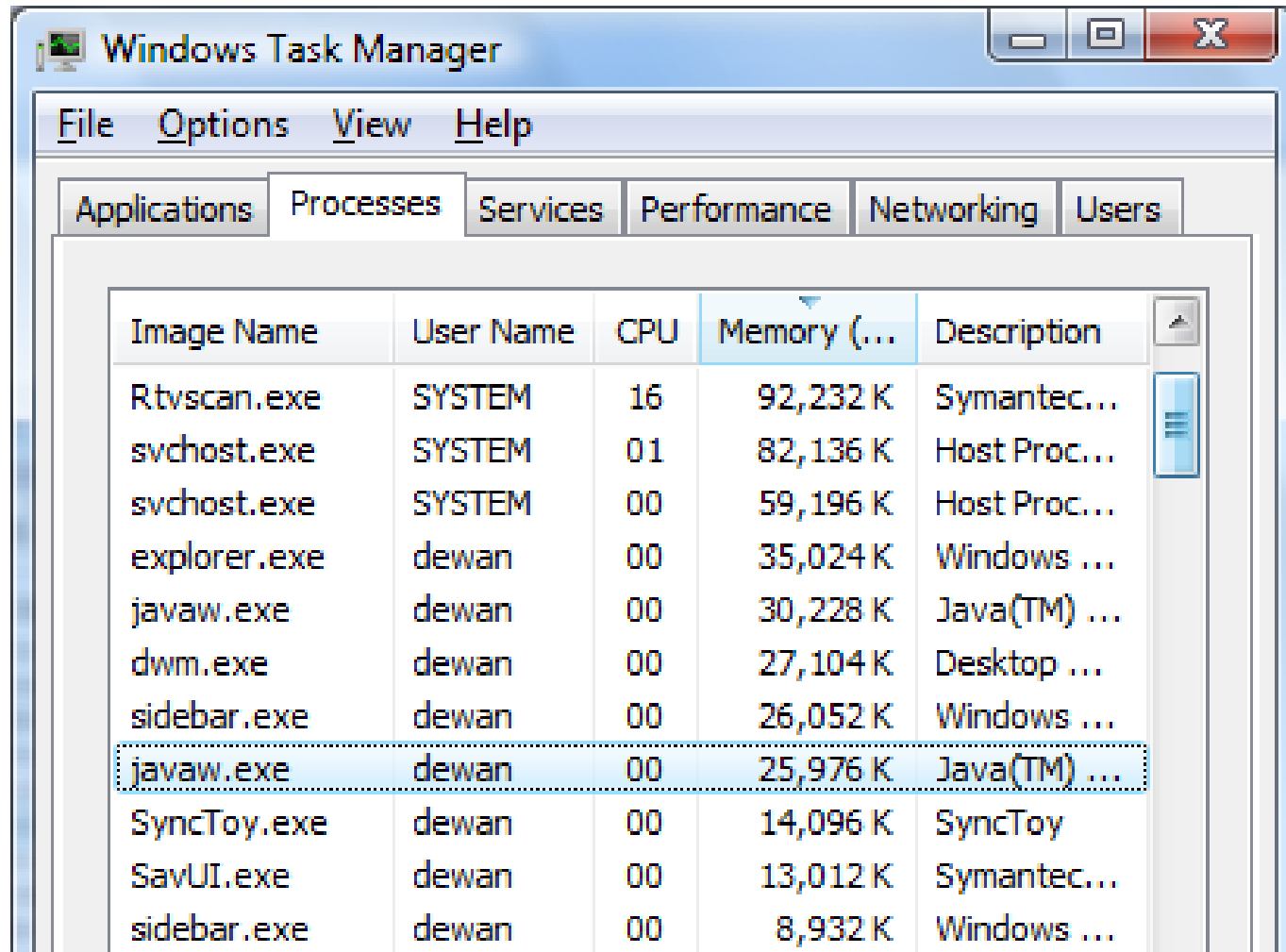
Reply to
email

Threads can
have different
priorities

Hug
baby



INTERLEAVED PROCESSES



Windows Task Manager

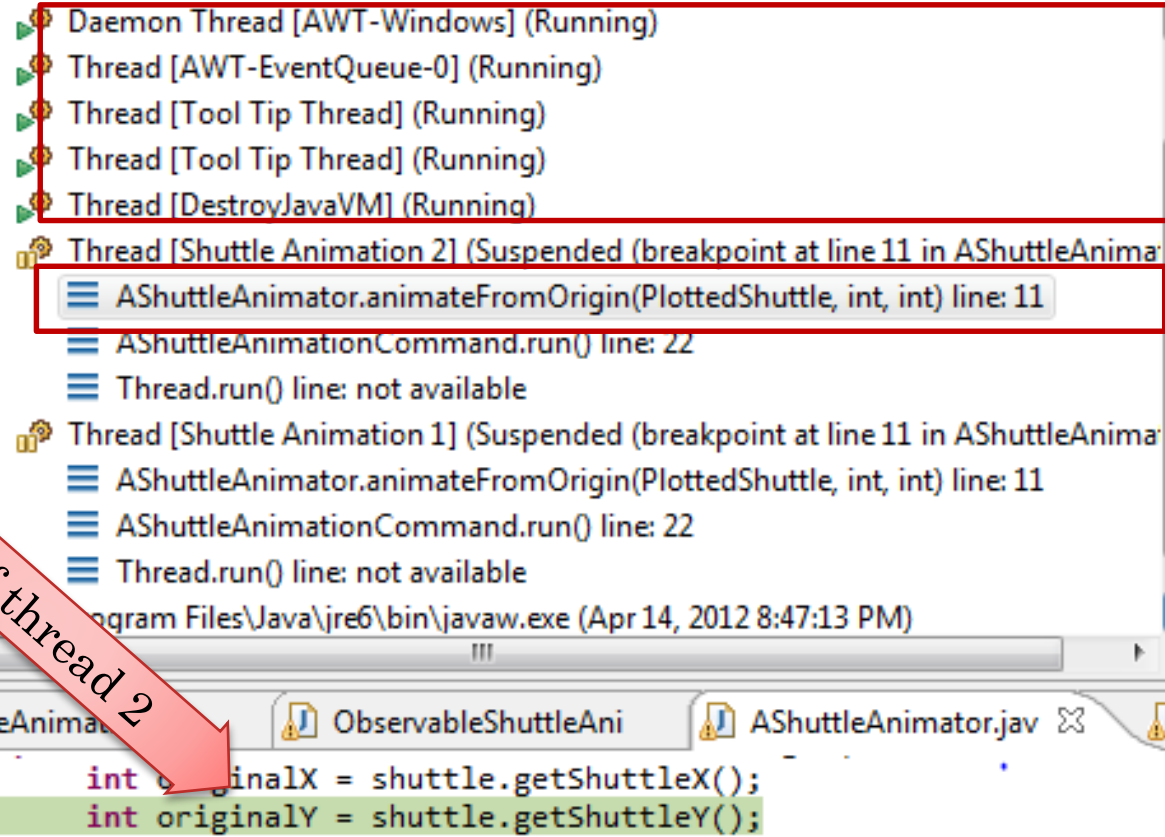
File Options View Help

Applications Processes Services Performance Networking Users

Image Name	User Name	CPU	Memory (...)	Description
Rtvsan.exe	SYSTEM	16	92,232 K	Symantec...
svchost.exe	SYSTEM	01	82,136 K	Host Proc...
svchost.exe	SYSTEM	00	59,196 K	Host Proc...
explorer.exe	dewan	00	35,024 K	Windows ...
javaw.exe	dewan	00	30,228 K	Java(TM) ...
dwm.exe	dewan	00	27,104 K	Desktop ...
sidebar.exe	dewan	00	26,052 K	Windows ...
javaw.exe	dewan	00	25,976 K	Java(TM) ...
SyncToy.exe	dewan	00	14,096 K	SyncToy
SavUI.exe	dewan	00	13,012 K	Symantec...
sidebar.exe	dewan	00	8,932 K	Windows ...

Process is an interleaved activity created by the Operating System each time a main method is run

INTERLEAVED THREADS



Thread is an interleaved activity within a process.

Threads in a process work cooperatively to do the job of the process.

Java creates several threads when running our program for UI processing, garbage collection, main method,



THE PROGRAM COUNTER OF OTHER THREAD

The screenshot displays the Java IDE's Thread console and the program counter for a specific thread. The Thread console lists several threads, with a red box highlighting the running threads: Daemon Thread [AWT-Windows] (Running), Thread [AWT-EventQueue-0] (Running), Thread [Tool Tip Thread] (Running), Thread [Tool Tip Thread] (Running), and Thread [DestroyJavaVM] (Running). Below these, two threads are suspended: Thread [Shuttle Animation 2] and Thread [Shuttle Animation 1]. The stack frames for Thread [Shuttle Animation 1] are expanded, showing the current method being executed: AShuttleAnimator.animateFromOrigin(PlottedShuttle, int, int) line: 11. A red box highlights this stack frame. A red arrow points to this thread with the text "PC of thread 1". The IDE's status bar at the bottom shows the current file is AShuttleAnimator.java, and the program counter is at line 11, column 11.

Thread [Shuttle Animation 1] (Suspended (breakpoint at line 11 in AShuttleAnimator.animateFromOrigin(PlottedShuttle, int, int) line: 11))


```
AShuttleAnimator.animateFromOrigin(PlottedShuttle, int, int) line: 11
AShuttleAnimationCommand.run() line: 22
Thread.run() line: not available
```

Program Files\Java\jre6\bin\javaw.exe (Apr 14, 2012 8:47:13 PM)


ShuttleAnimat ObservableShuttleAni AShuttleAnimator.jav

```
int originalX = shuttle.getShuttleX();
int originalY = shuttle.getShuttleY();
```

CONCURRENT ACTIVITIES




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




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



CONCURRENT ACTIVITIES



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



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


Processes/threads can execute concurrently in multi-processor, multi-core computers.

On single-core, single-processor, a single processor/core interleaves their execution.


CONCURRENCY IN “REAL” LIFE




CONCURRENCY AND INTERLEAVING



```
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);  
}
```




CONCURRENCY AND INTERLEAVING



```
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);  
}
```




CONCURRENCY AND INTERLEAVING



```
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);  
}
```



CONCURRENCY AND INTERLEAVING



Two hands serving
three balls ~ two cores
serving three threads.

While a ball is in the air the other balls
can be served ~ while a thread is
waiting for user input or sleeping,
others can be served.



INTERLEAVING ACTIVITIES

- Each interleaved activity is associated with its own program counter, which marks the next statement to be executed for that activity.
- At any one time, a single CPU executes the statement of only one activity called the current activity.
- The CPU does not wait for the current activity to finish before it starts to execute another activity.
How should we create a separate thread to execute `to animateFromOrigin`?

Java creates the threads.

Requires us to tell it what method to execute.



INDEPENDENT ANIMATIONS?

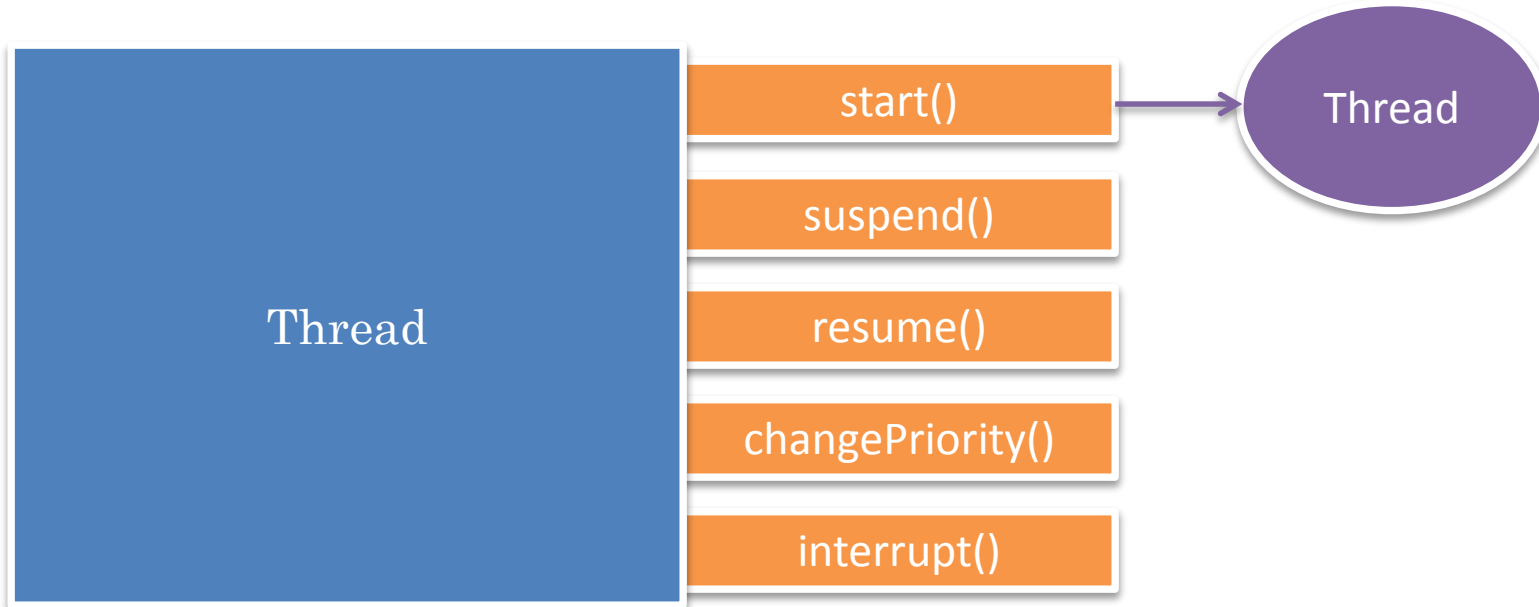
```
public static void main(String[] args) {  
    PlottedShuttle shuttle1 = new AnObservablePlottedShuttle(50, 100);  
    OEFram e oeFrame1 = ObjectEditor.edit(shuttle1);  
    oeFrame1.hideMainPanel();  
    oeFrame1.setLocation(0, 0);  
    oeFrame1.setSize(400, 400);  
    PlottedShuttle shuttle2 = 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();  
    shuttleAnimator1.animateFromOrigin(shuttle1, 5, 100);  
    shuttleAnimator2.animateFromOrigin(shuttle2, 5, 100);  
}
```

Second animation
waits for first one
to finish

How to get two
threads



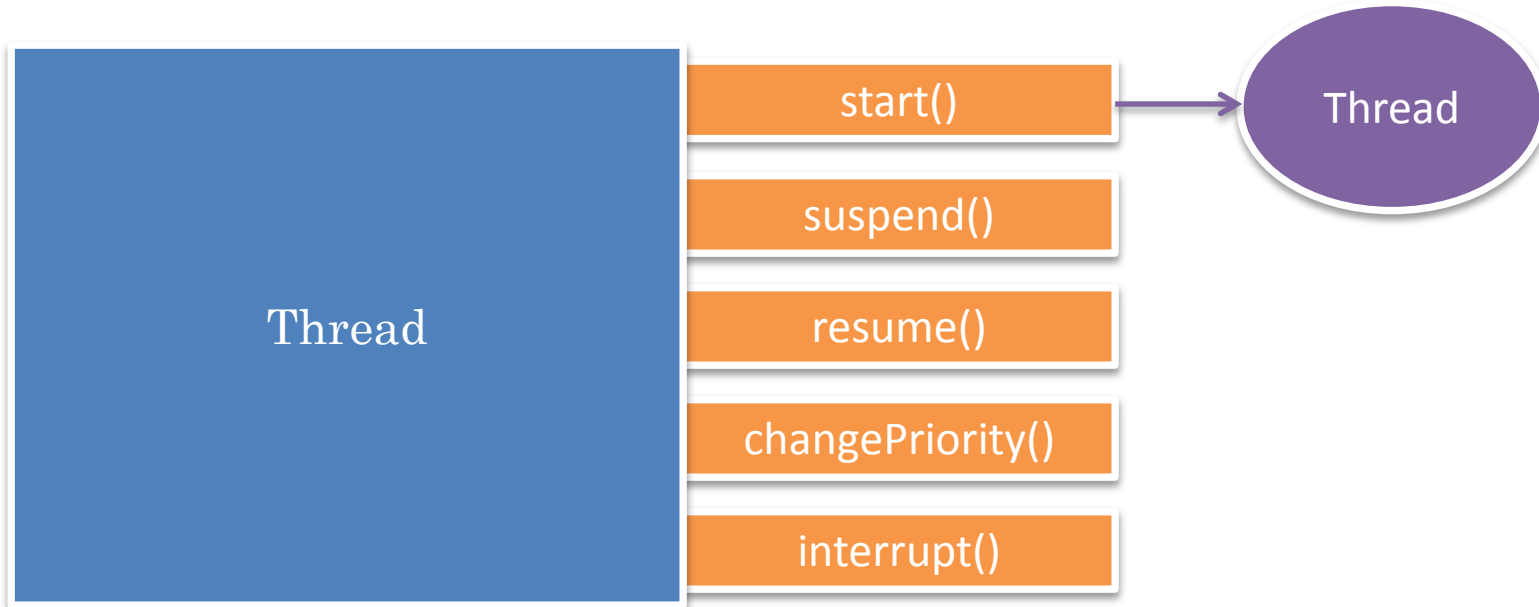
THREAD OBJECT



A thread is an object that can be started., suspended, resumed, interrupted while sleeping, given lower/higher priority ...

How to create a startable thread?

THREAD OBJECT (REVIEW)



A thread is an object that can be started., suspended, resumed, interrupted while sleeping, given lower/higher priority ...

How to create a startable thread?



INDEPENDENT ANIMATIONS?

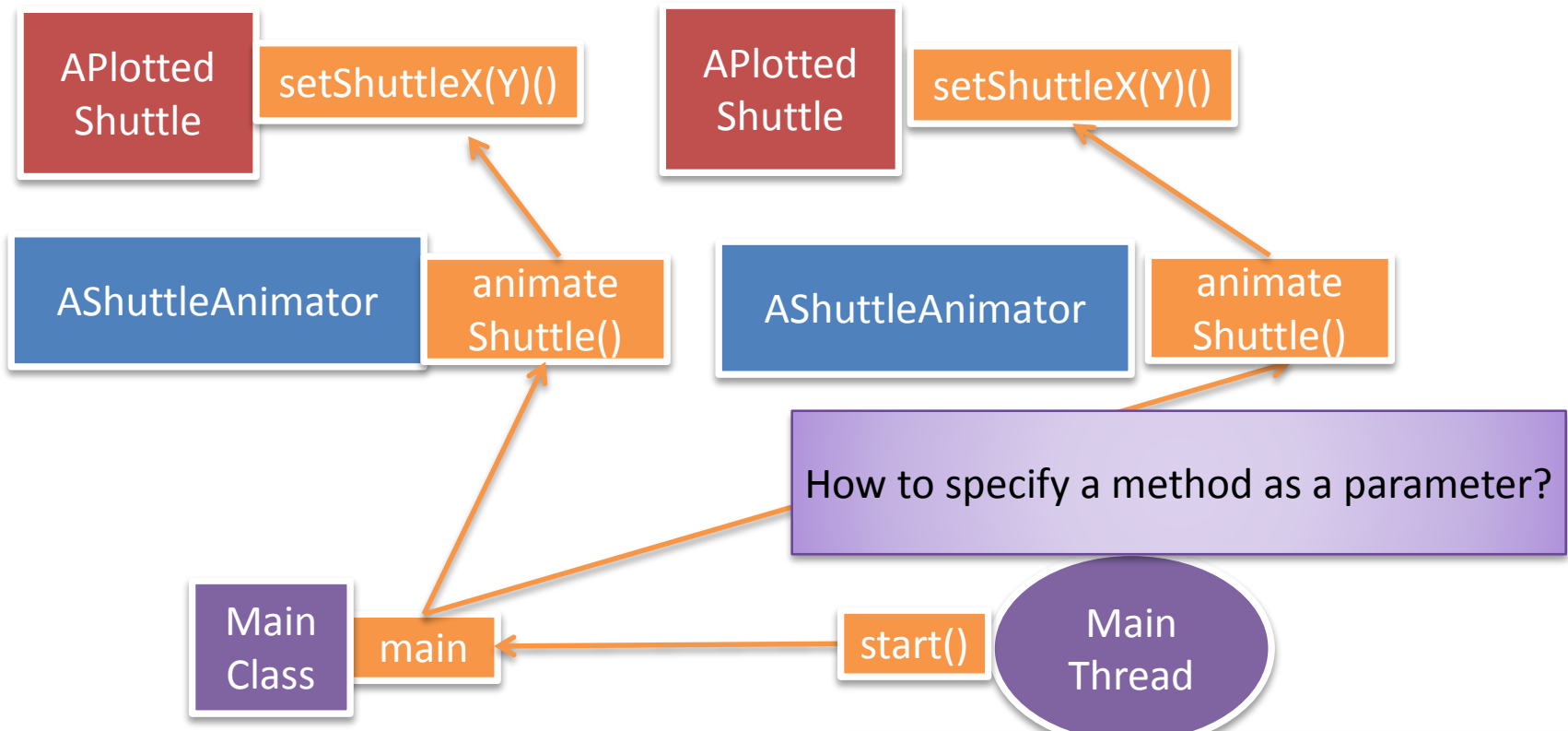
```
public static void main(String[] args) {  
    PlottedShuttle shuttle1 = new AnObservablePlottedShuttle(50, 100);  
    OEFrame oeFrame1 = ObjectEditor.edit(shuttle1);  
    oeFrame1.hideMainPanel();  
    oeFrame1.setLocation(0, 0);  
    oeFrame1.setSize(400, 400);  
    PlottedShuttle shuttle2 = 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();  
    shuttleAnimator1.animateFromOrigin(shuttle1, 5, 100);  
    shuttleAnimator2.animateFromOrigin(shuttle2, 5, 100);  
}
```

Second animation
waits for first one
to finish

Need multitasking



PREDEFINED MAIN THREAD



Each thread has a stack of method calls

A call to a method M^2 by M^1 gets pushed on top of the call to M^1 which blocks

The call to M^2 is popped when it returns, allowing M^1 to continue

Creating a new thread means asking a Thread object to create a new stack so no blocking occurs

→ Must specify to (the constructor of) the Thread object the first method call in the stack (method + parameters + object)



ACTION OBJECT (METHOD OBJECT)

Action Object
(Method Object)

`execute (targetObject, params)`

Provides an execute operation to perform some action.

The execute operation takes as an argument the object on which the target operation is to be invoked and an array of parameters of the target action.

Can pass to a constructor a method object.

A method that does not take method objects as arguments is a first-order function

A method that takes as an argument a method object is a second (higher) order function.

A method that takes as an argument an N-order method object is an N+1 order method

Java reflection supports higher order functions



USING REFLECTION TO CREATE THREAD OBJECT

```
public static void demoShuttleAnimation(  
    ShuttleAnimator aShuttleAnimator, PlottedShuttle aShuttle) {  
    aShuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

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

Reflection allows method
parameters

Exceptions can be thrown because a
method can be invoked on an arbitrary
object with arbitrary parameters

A la grammar/vocabulary vs. phrase book



ACTION VS. COMMAND

```
public static void demoShuttleAnimation(  
    ShuttleAnimator aShuttleAnimator, PlottedShuttle aShuttle) {  
    aShuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

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

Bundle the target object and the three arguments into a single command object associated with some action

The constructor of the command object checks that the target and parameters are compatible with its action



ACTIONOBJECT (METHOD OBJECT)

Action Object
(Method Object)

execute
(targetObject,
params)

Use an object in which the
action, target and params are
bundled together



COMMAND OBJECT (METHOD OBJECT)

Action Object
(Method Object)

execute
(targetObject,
params)

Commmand
(action with target
+ parameters)

execute ()

Constructor
(targetObject, params)

Provides a execute operation to
perform some action.

The execute operation takes no
arguments.

Action vs. command object \leftrightarrow “do”
vs “Bob, do your homework”

Constructor takes target object and
parameters of operation as
arguments.

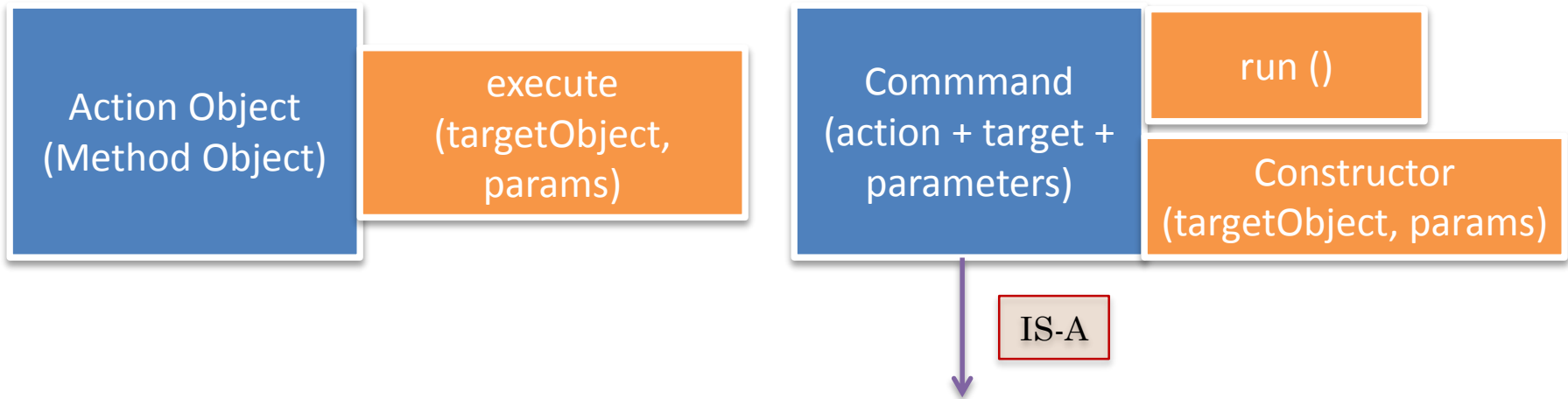
Action is an operation that can be
invoked on many different
arguments

A command is a specific action
invocation.

Action vs. command object \leftrightarrow
“move” vs “move Arthur 50 30”



JAVA THREAD COMMAND OBJECT



```
package java.lang;
public interface Runnable {
    public void run();
}
```

The interface is called Runnable instead of Command and the method is called run() instead of execute() because Thread designers probably did not realize they were using a command object

The command object like an observable, model, view is a general design object that occurs in several contexts and was invented in the context of undo



RUNNABLE IMPLEMENTATION

```
public class AShuttleAnimationCommand implements Runnable {
    ShuttleAnimator shuttleAnimator;
    PlottedShuttle shuttle;
    int animationStep;
    int animationPauseTime;
    public AShuttleAnimationCommand (
        ShuttleAnimator aShuttleAnimator,
        PlottedShuttle aShuttle,
        int anAnimationStep,
        int anAnimationPauseTime) {
        shuttleAnimator = aShuttleAnimator;
        shuttle = aShuttle;
        animationStep = anAnimationStep;
        animationPauseTime = anAnimationPauseTime;
    }
    public void run() {
        shuttleAnimator.animateFromOrigin(shuttle,
            animationStep, animationPauseTime);
    }
}
```

Action is hardwired:
Separate implementation of
Command Object Interface
for each action

Action

Parameters

Target Object

TRACING COMMAND OBJECT

```
public static void demoShuttleAnimation(  
    ShuttleAnimator aShuttleAnimator, PlottedShuttle aShuttle) {  
    aShuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
}
```

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

Creates a new
Command

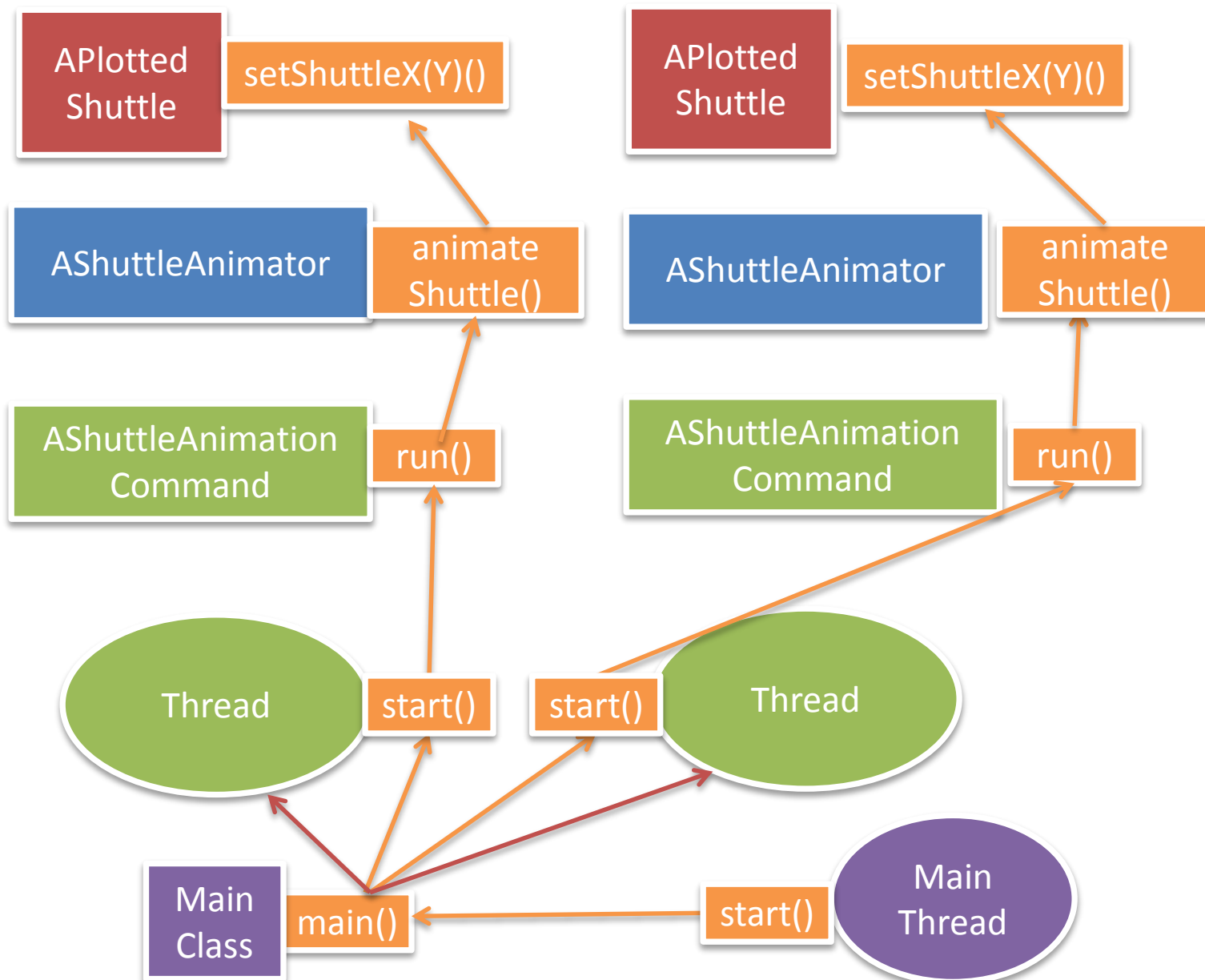
Starts the thread, which invokes the run
method on the command object passed to
constructor.

Creates a new
Java thread

The run method calls animateFromOrigin() on
target object



TWO SHUTTLES AND ANIMATORS, TWO THREADS



SYNCHRONOUS/ASYNCHRONOUS METHOD CALL

```
public static void demoShuttleAnimation(  
    ShuttleAnimator aShuttleAnimator, PlottedShuttle aShuttle) {  
    aShuttleAnimator.animateFromOrigin(aShuttle, 5, 100);  
    System.out.println("animation finished");  
}
```

```
public static void concurrentDemoShuttleAnimation(  
    ShuttleAnimator aShuttleAnimator, PlottedShuttle aShuttle) {  
    Thread thread = new Thread(new AShuttleAnimationCommand  
        (aShuttleAnimator, aShuttle, 5, 100));  
    threadNumber++;  
    thread.setName(SHUTTLE_THREAD_NAME + " " + threadNumber);  
    thread.start();  
    System.out.println("animation finished");  
}
```

Method caller waits for
called method to finish.

Thread creator does not wait
for created thread to finish.

Thread creation is typically the last step in a method call chain, so thread creator will probably be gone by the time the started thread executes

SYNCHRONOUS VS. ASYNCHRONOUS

```
operation(<parms>)
```

```
animateFromOrigin();
```

```
System.out.println("hello")
```

Synchronous: Operation invoker **blocks** or waits until the operation finishes

Asynchronous: Operation invoker does not block until completion

Some other operation (e.g. observer notification) needed to wait for result or completion status



SYNCHRONOUS/ASYNCHRONOUS ANALOGY

- Synchronous

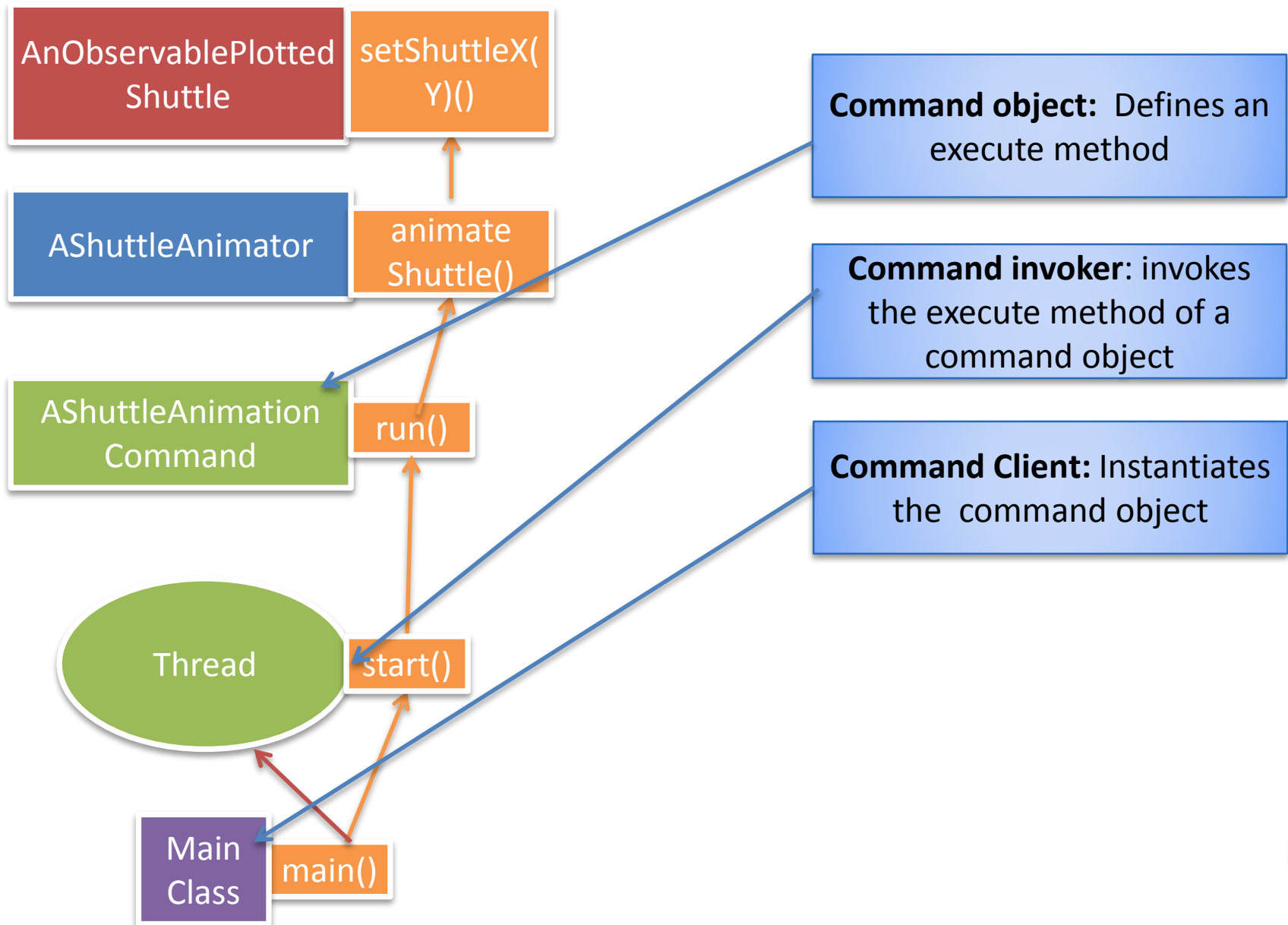
- I ask you a question in class and wait for answer.
- I order food at subway and wait till my sandwich is made

- Asynchronous

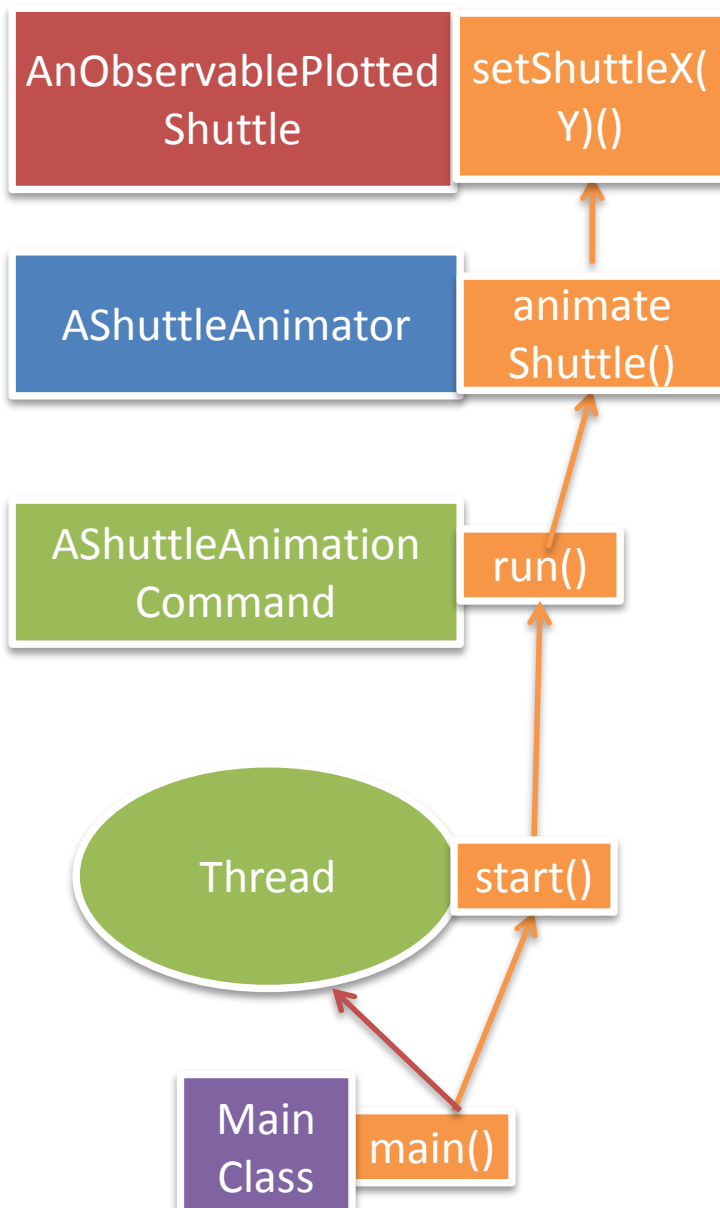
- I give you the question as home work.
- I order food at a fancier restaurant with waiters.



COMMAND DESIGN PATTERN



THREAD-BASED ANIMATION DESIGN PATTERN



Animated object: performs each animation step, unaware of the animation

Animating object: Implements looping animation methods that take as parameters the animated object and optional animation controls

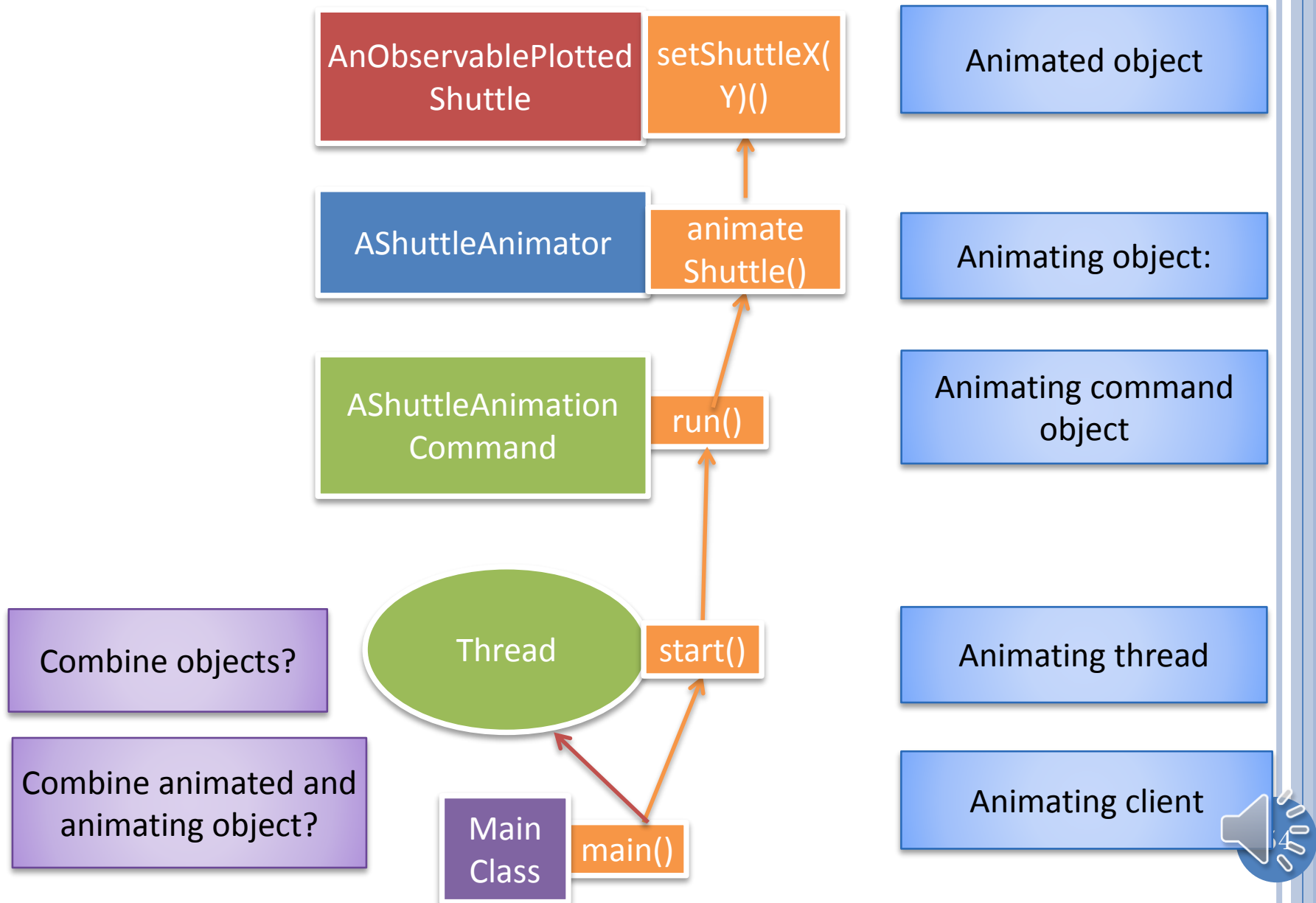
Animating command object: Its execute method calls an animating method in an animating object with parameters of the method. Constructor takes as parameters the animating object, animated object and animation controls.

Animating thread: Invoker of the execute method of an animating command object

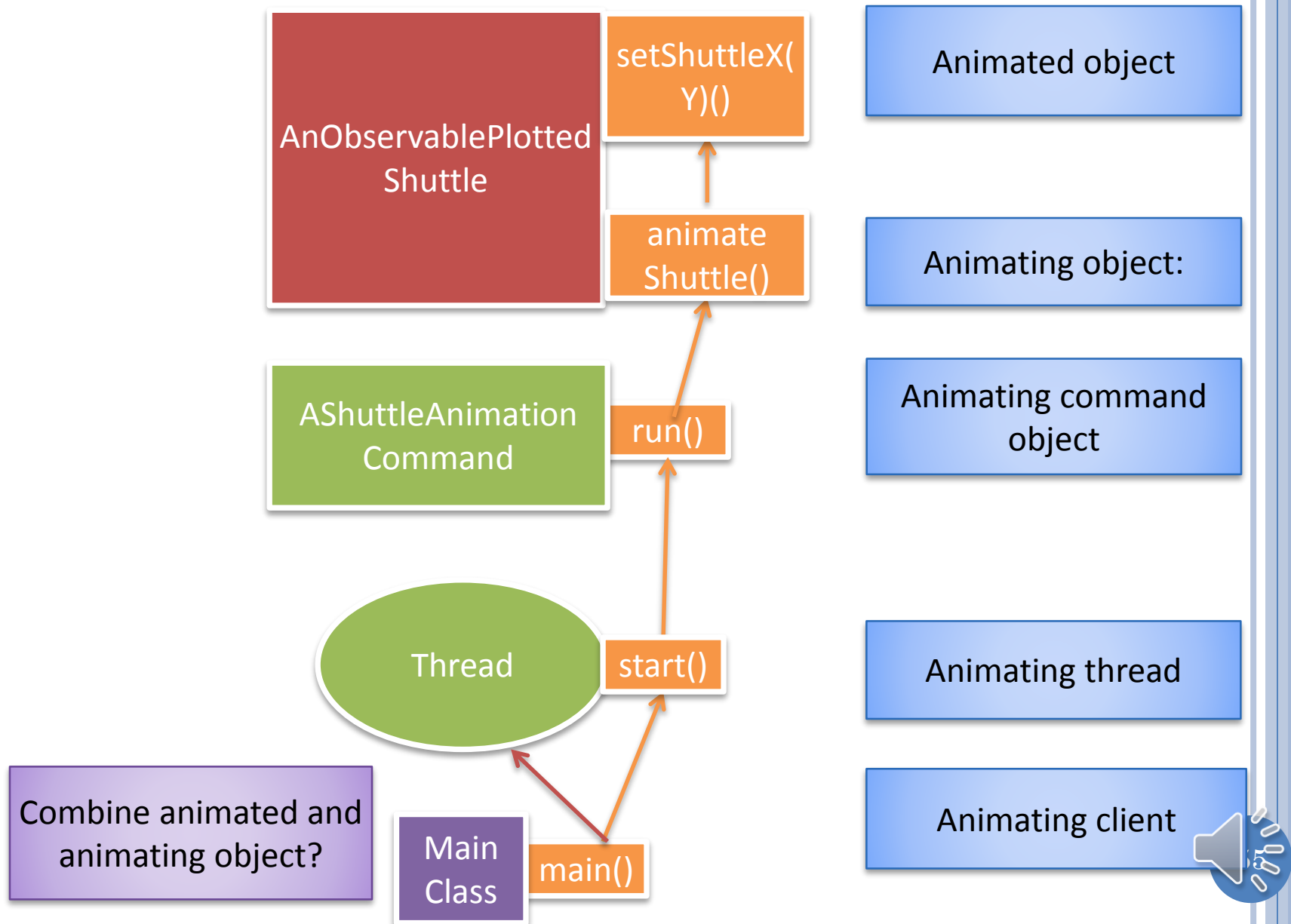
Animating client: Client of an animating command object and creator and starter of an animating thread



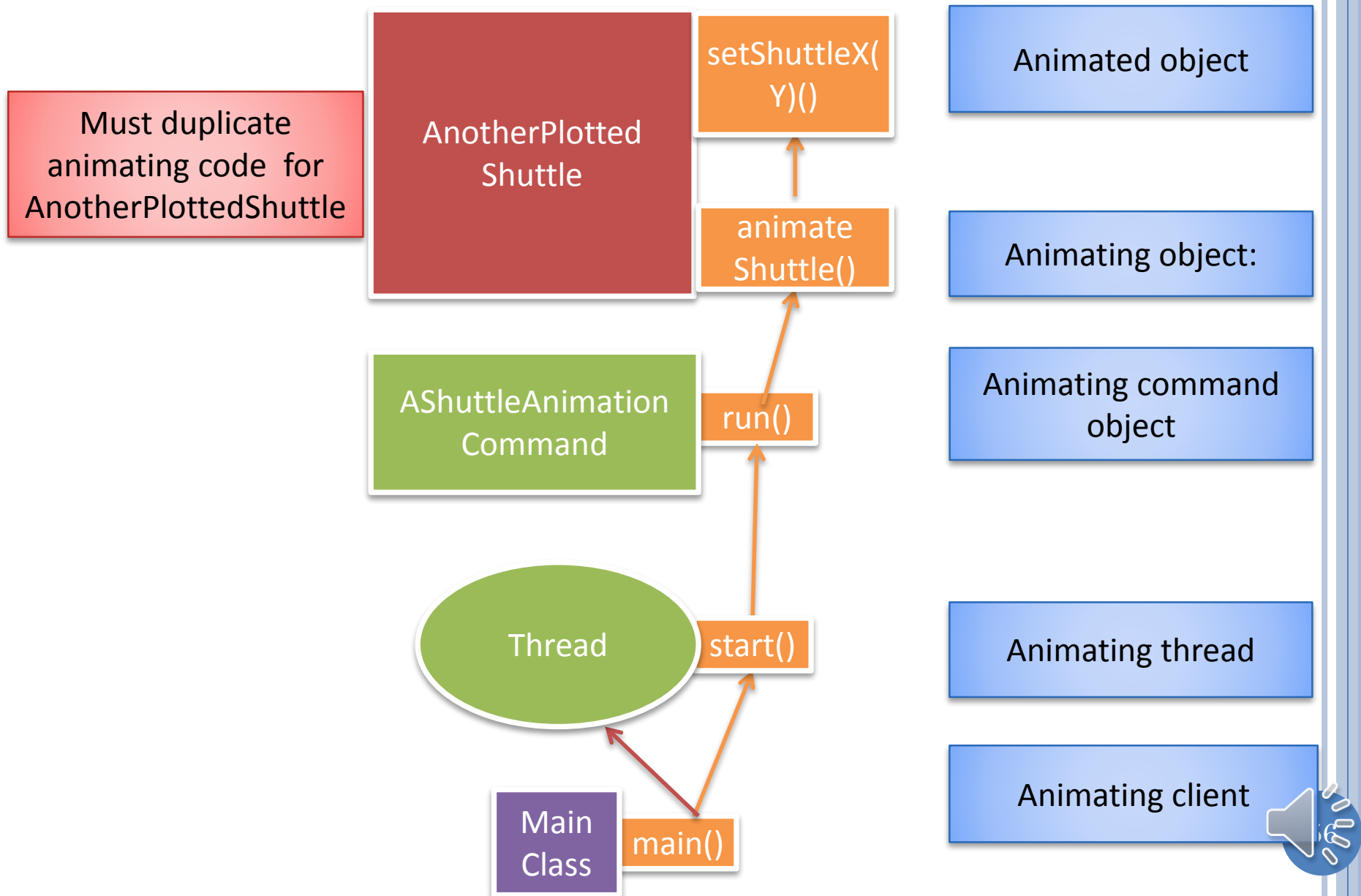
THREAD-BASED ANIMATION DESIGN PATTERN



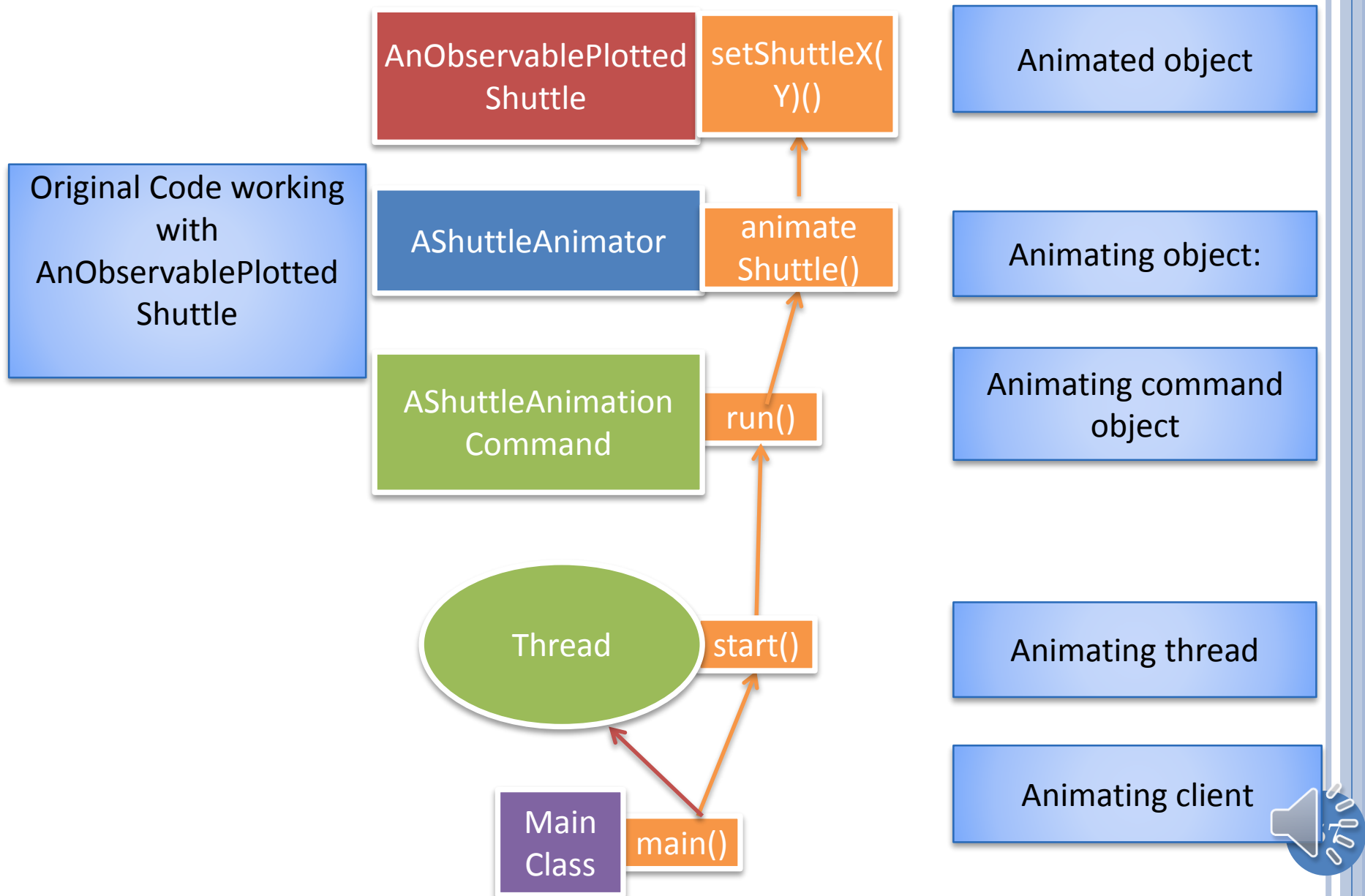
COMBINE ANIMATED AND ANIMATING?



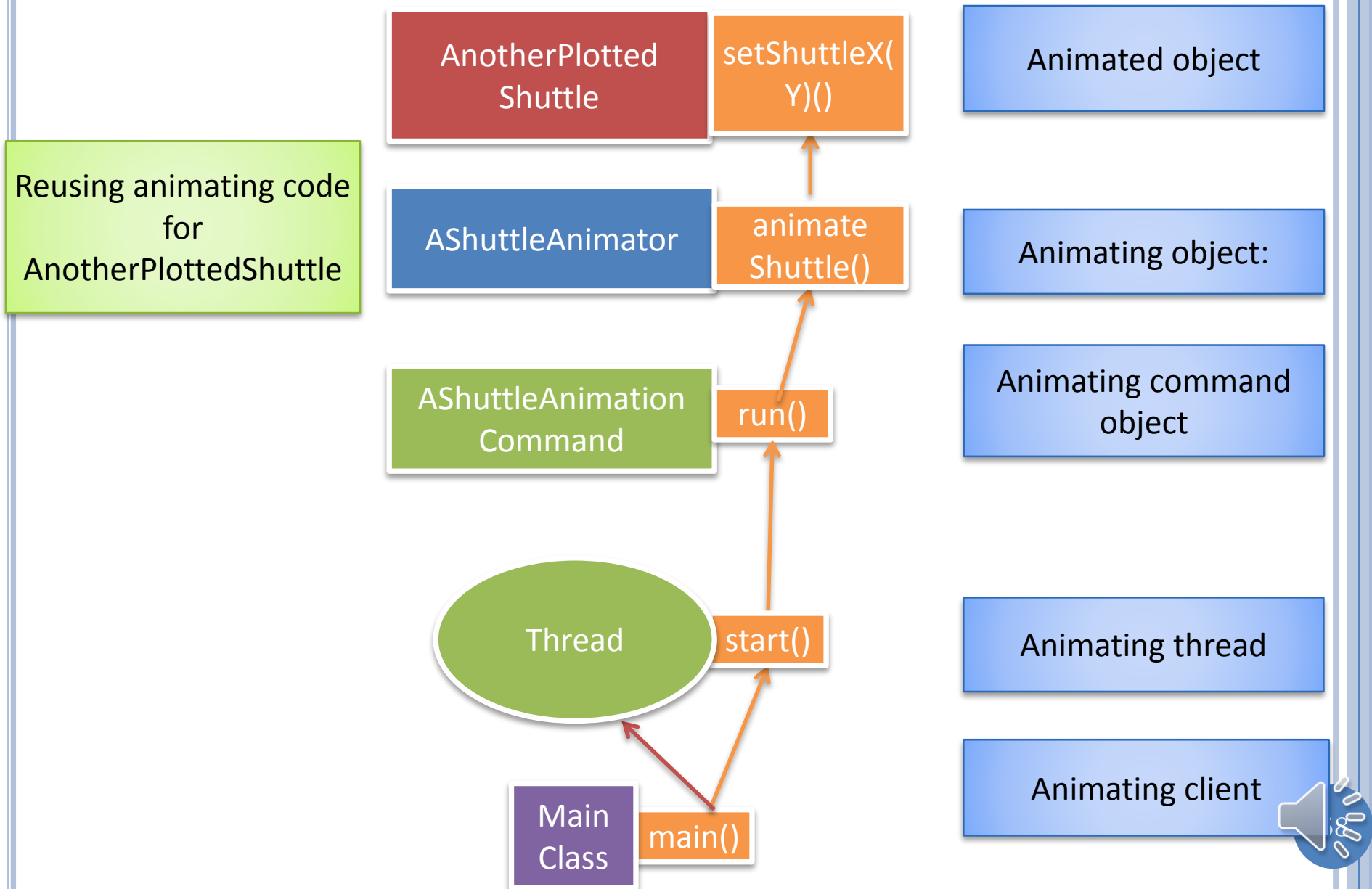
COMBINING ANIMATED AND ANIMATING



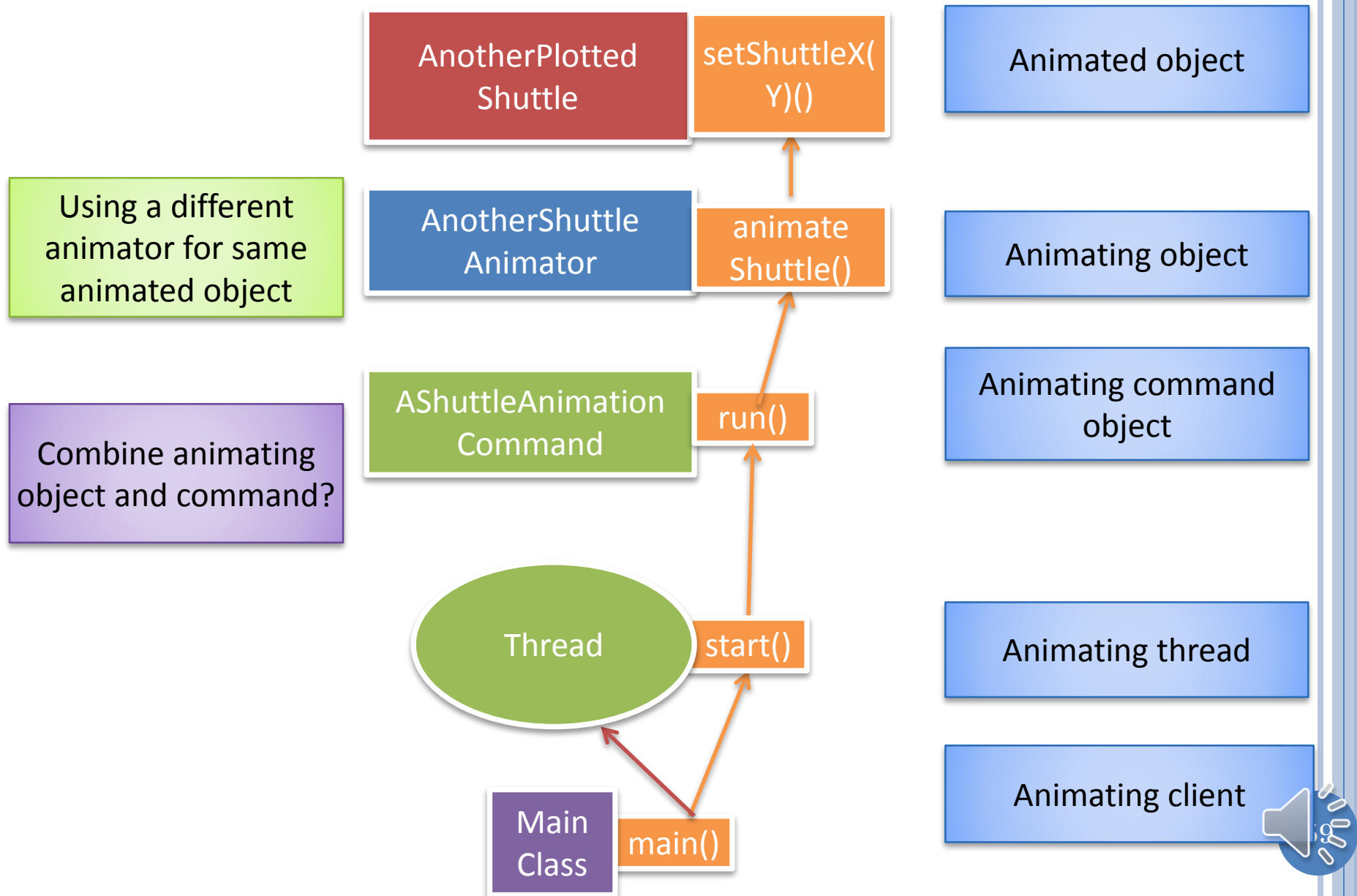
THREAD-BASED ANIMATION DESIGN PATTERN



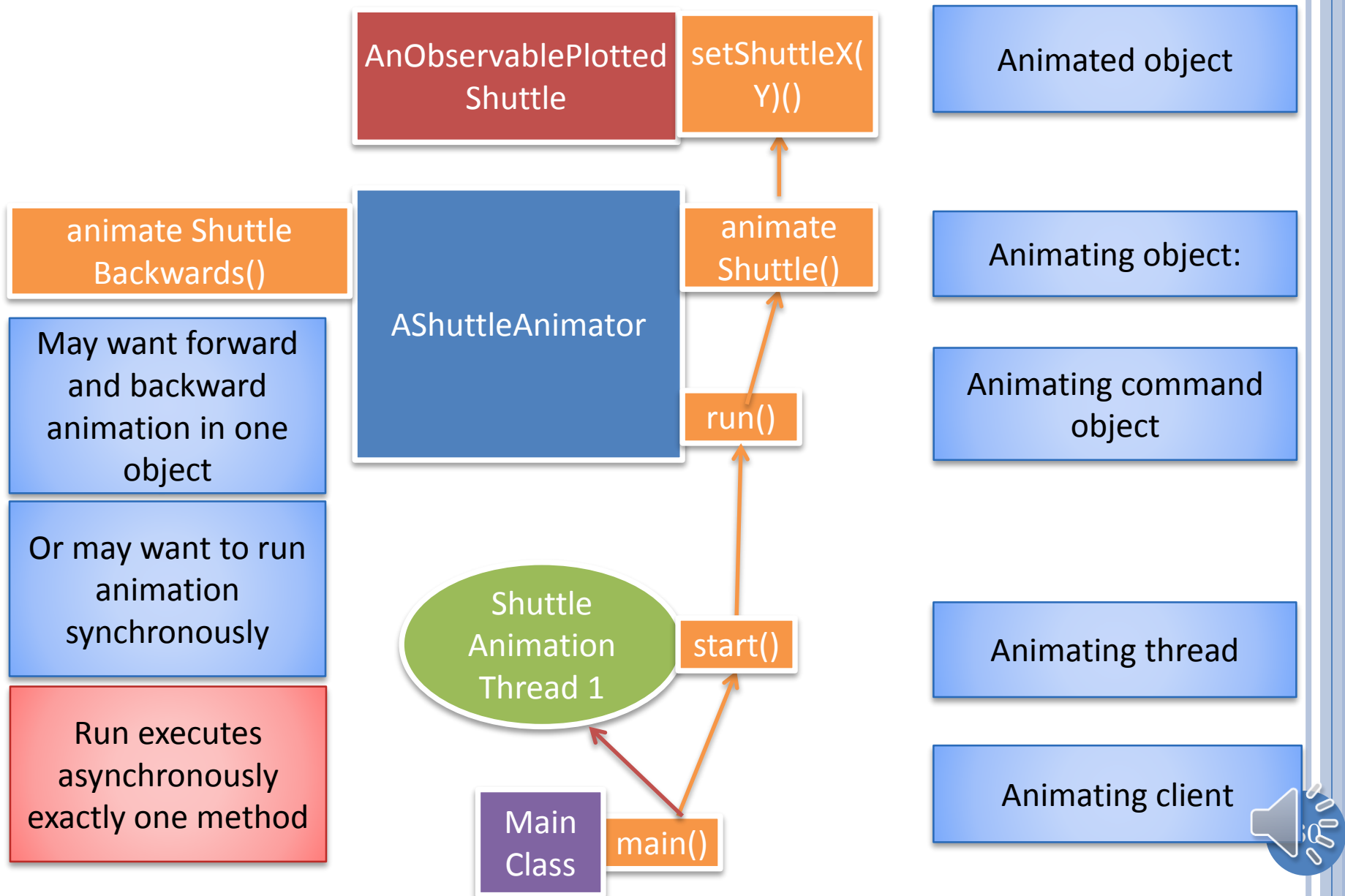
THREAD-BASED ANIMATION DESIGN PATTERN



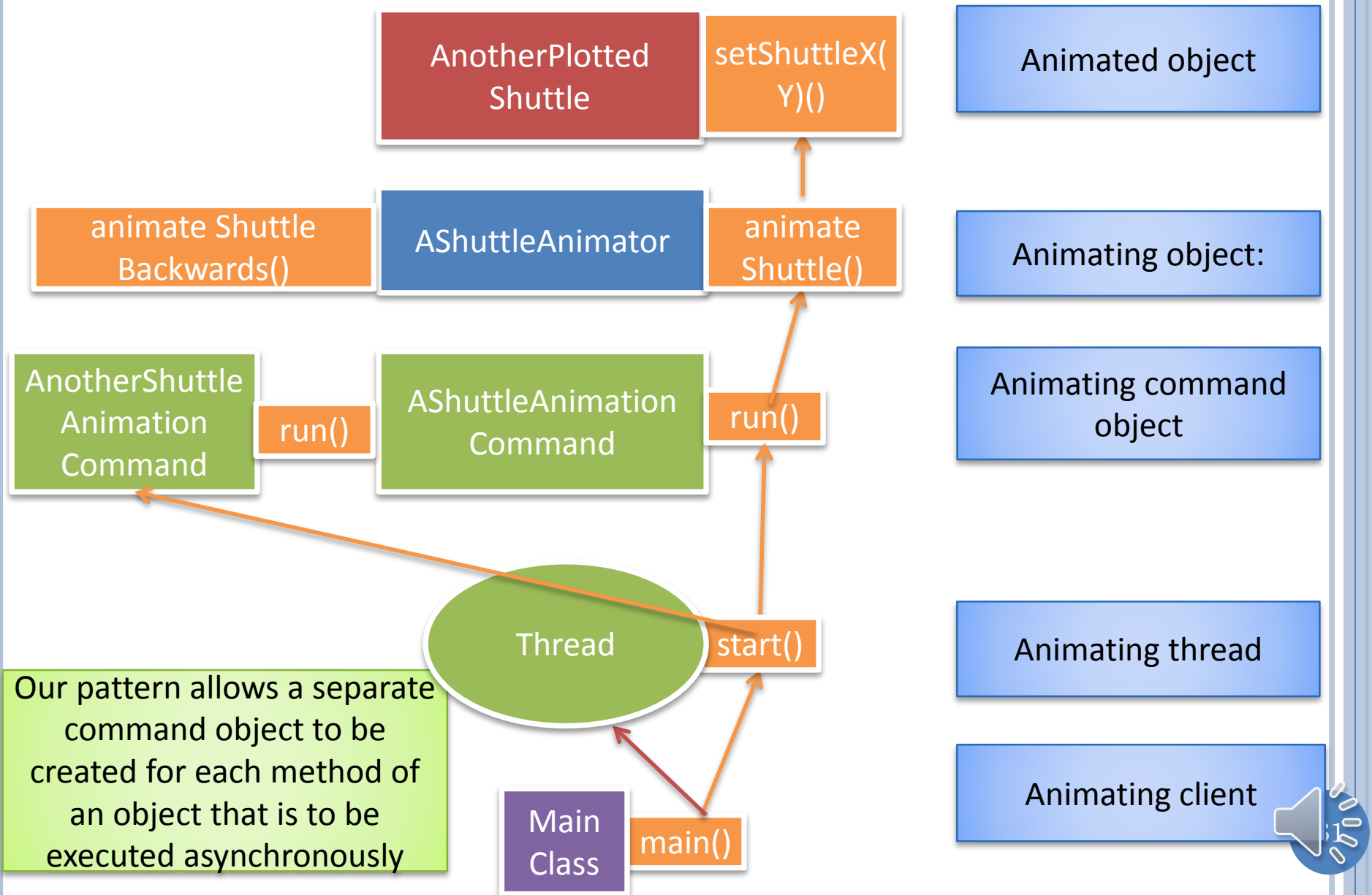
THREAD-BASED ANIMATION DESIGN PATTERN



COMBINE ANIMATOR AND COMMAND?



THREAD-BASED ANIMATION DESIGN PATTERN



SUBCLASS THREAD?

Subclass Thread rather than give it a constructor argument

AnObservablePlotted Shuttle

setShuttleX(Y())

Animated object

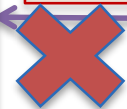
AShuttleAnimator

animate Shuttle()

Animating object:

AnAnimating Command

IS-A



AShuttleAnimation Command

run()

Animating command object

A command is executed by a thread and is not a thread!

Subclassed thread cannot inherit from another class

IS-A

run()

Thread

start()

Animating thread

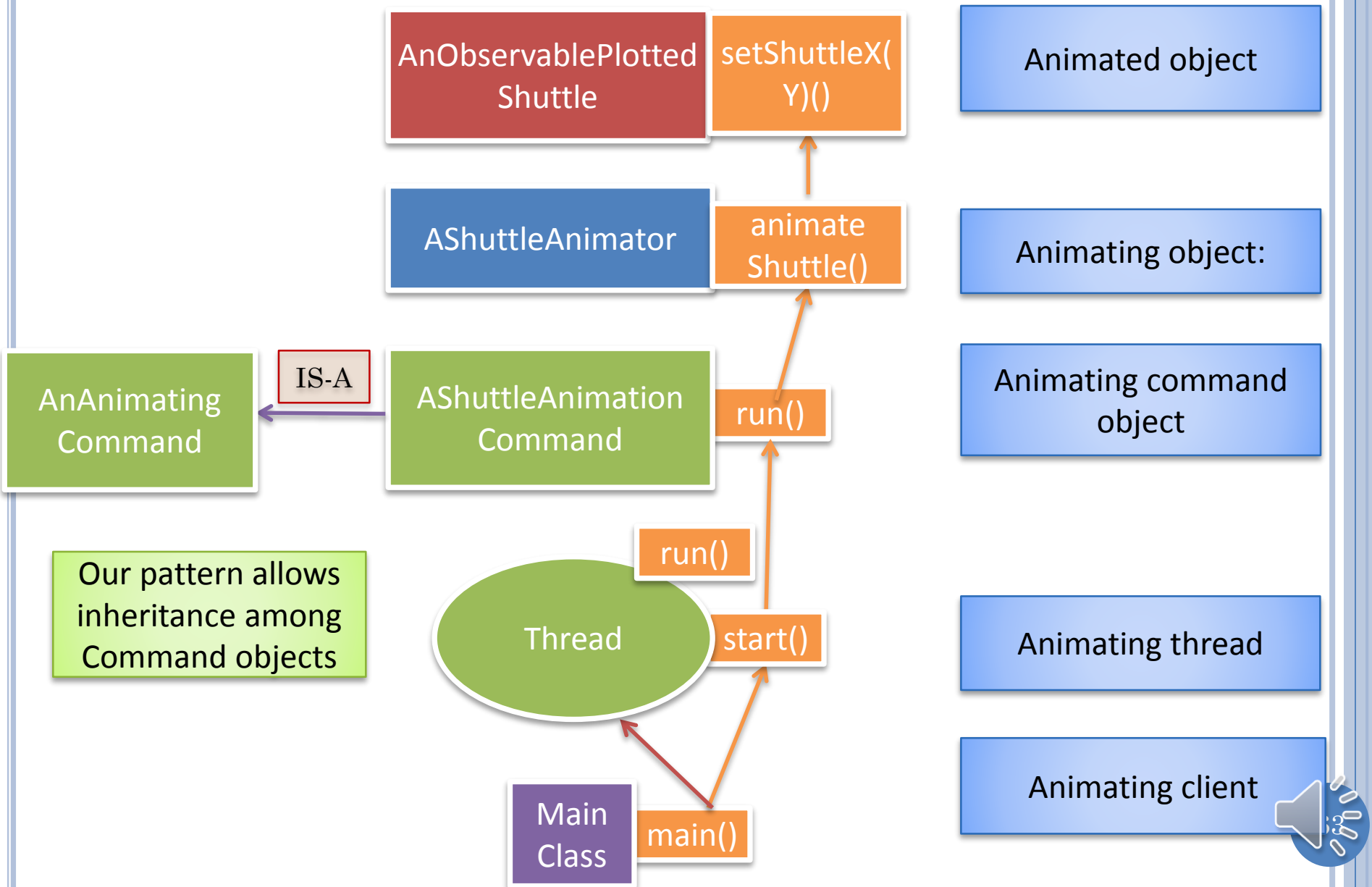
Main Class

main()

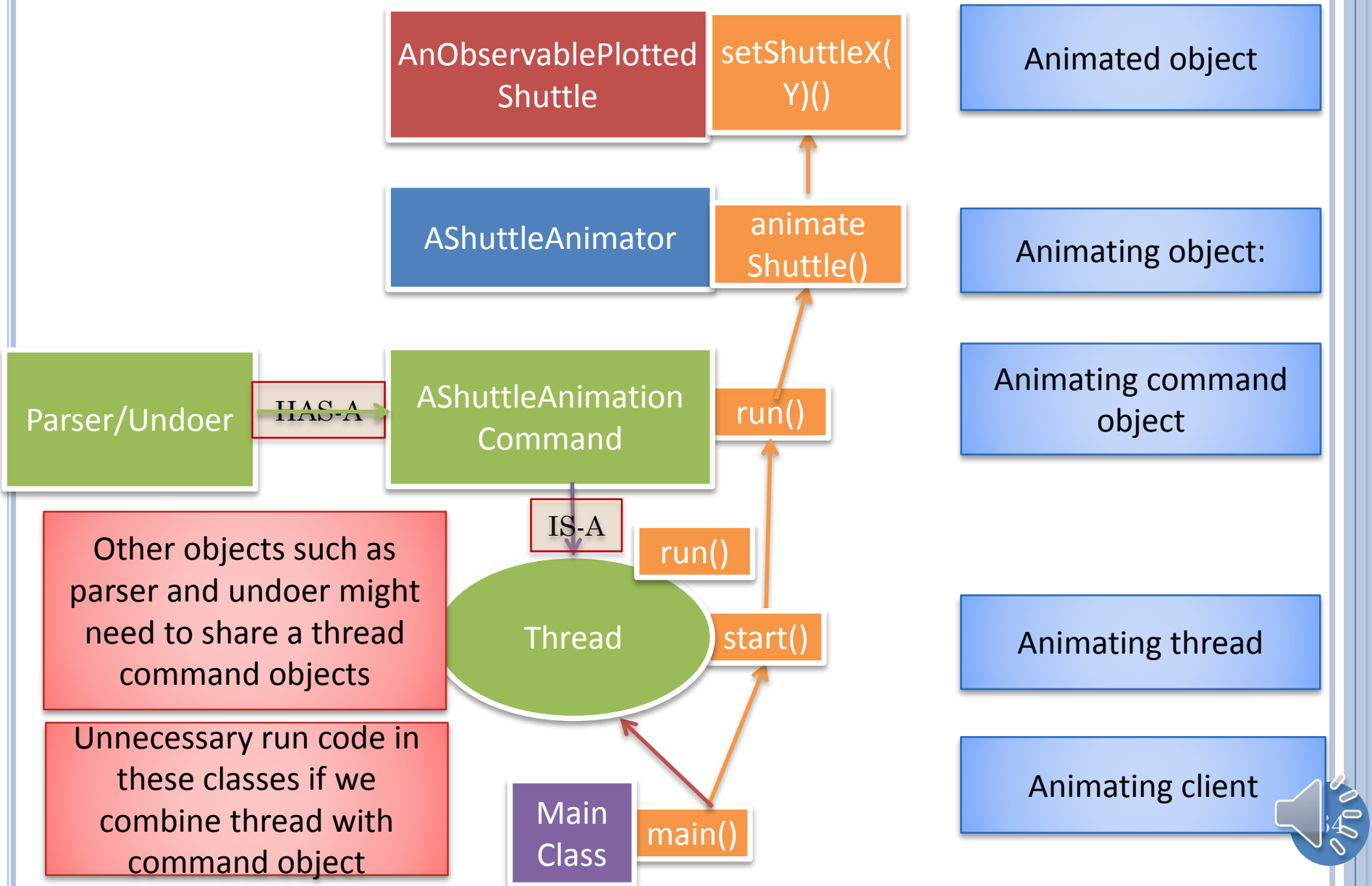
Animating client



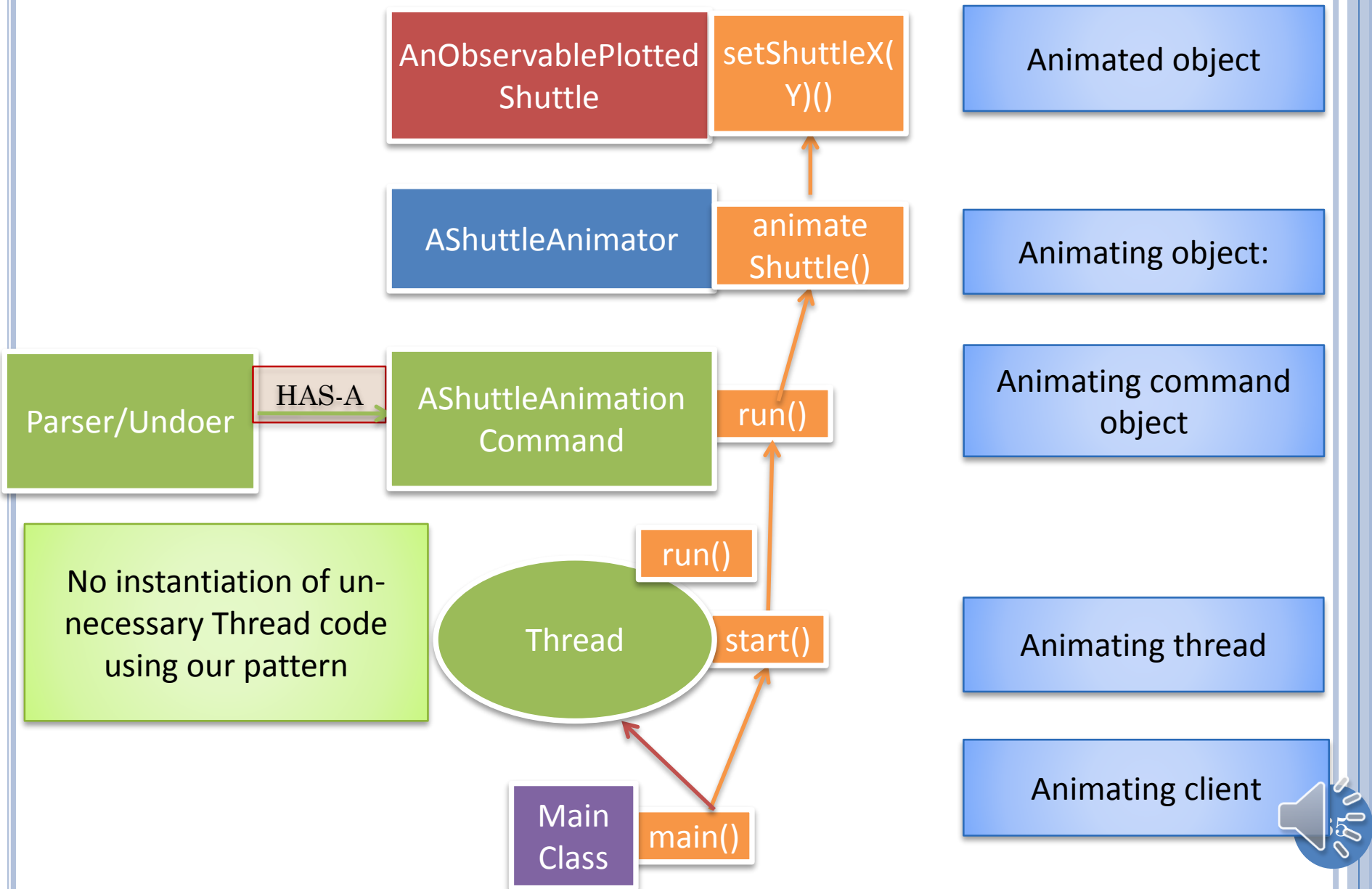
INHERITANCE AMONG COMMAND OBJECTS



ABUSING IS-A AND INSTANTIATING UNNECESSARY CODE



NOT INSTANTIATING UNNECESSARY CODE



COMMAND OBJECTS VS. THREADS

- Threads can use command objects
- Command objects can be used in non thread contexts
 - Undo/redo
 - Processing command interpreter commands



THREADS

- If we want multiple animations then we must create our own threads
- Threads use command objects.
- Command objects represent method calls



ANIMATION PATTERN

- Begin with the object to be animated
- Write one or more animating object with one or more looping methods to animate the animated object that take animation controls as arguments
- Write one or more implementations of Runnable, each of which takes the animation controls and the above two objects as parameters and calls a looping method of the animating objects
- In the main program or some model method, create one or more Thread instance, passing to the Thread constructor an instance of the Runnable command object.
- Execute the start() method on each Thread instance.

