Instructor: Prasun Dewan
INTERACTIVE TRAFFIC CONTROL

How does a thread wait for an event?
WAIT FOR ANOTHER THREAD TO RELEASE METHOD

Waiting on previous event rather than some future event to occur

Shuttle Animation Thread 1

synchronized animateShuttle() -> AShuttle Animator

Shuttle Animation Thread 2
WAIT FOR TIME TO PASS

Could continuously “poll” for event → would not react immediately and would waste CPU time

Need to wait for signal produced by some other thread
ANALOGIES

proceed()  waitFor Proceed()
WAIT AND NOTIFY

Shuttle Animation Thread

waitForProceed()

AClearance Manager

proceed()

ObjectEditor.edit(clearanceManager);
public class AClearanceManager implements ClearanceManager {
    @Row(0)
    @ComponentWidth(100)
    public synchronized void proceed() {
        notify();
    }
    public synchronized void waitForProceed() {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
wait and notify must be executed in synchronized methods as they access shared Queue

Wait releases synchronized lock on object, makes calling thread wait, and lets some other thread get object synchronized lock
**Wait and Notify Methods**

- `wait()`: makes the calling thread wait (in a queue) until the next `o.notify()` call (by some other thread), allows other synchronized methods to execute.

- `notify()`: if some threads are waiting as a result of `o.wait()`, then unblocks the first one who waited.

To wait changes some internal variable (a Queue) in class `Object` which is accessed by `notify`.

Since shared data structure accessed by multiple threads, `wait()` and `notify()` must execute in synchronized methods.
public class AShuttleAnimatorWaitingForClearance
   extends AConcurrentShuttleAnimator {
   ClearanceManager clearanceManager;
   public AShuttleAnimatorWaitingForClearance(PlottedShuttle theShuttle,
      ClearanceManager aClearanceManager) {
      super(theShuttle);
      clearanceManager = aClearanceManager;
   }
   protected void animateYFromOrigin(PlottedShuttle shuttle,
      int animationStep, int animationPauseTime,
      int startY, int endY) {
      clearanceManager.waitForProceed();
      super.animateYFromOrigin(shuttle, animationStep,
         animationPauseTime, startY, endY);
   }
}
public class ManualShuttleTrafficControl
    extends ConcurrentShuttleLaunchAnimation {

    public static int ANIMATOR_FRAME_HEIGHT = 125;
    public static int CLEARANCE_FRAME_WIDTH = 200;
    public static int CLEARANCE_FRAME_HEIGHT = 150;
    static int animatorNumber;

    public static void displayShuttleAnimator(
            ShuttleAnimator shuttleAnimator1) {
        OEFrame frame = ObjectEditor.edit(shuttleAnimator1);
        frame.setLocation(START_FRAME_X +
                (animatorNumber * SHUTTLE_FRAME_WIDTH),
                START_FRAME_Y + SHUTTLE_FRAME_HEIGHT);
        frame.setSize(SHUTTLE_FRAME_WIDTH, ANIMATOR_FRAME_HEIGHT);
        animatorNumber++;
    }

    public static void displayClearanceManager(
            ClearanceManager aClearanceManager) {
        OEFrame frame = ObjectEditor.edit(aClearanceManager);
        frame.setLocation(START_FRAME_X,
                START_FRAME_Y + SHUTTLE_FRAME_HEIGHT/2);
        frame.setSize(CLEARANCE_FRAME_WIDTH, CLEARANCE_FRAME_HEIGHT);
    }
}
public static void main(String[] args) {

    ClearanceManager clearanceManager =
    new AClearanceManager();
    PlottedShuttle shuttle1 =
    new AnObservablePlottedShuttle(SHUTTLE1_X, SHUTTLE1_Y);
    displayShuttleFrame(shuttle1);
    ShuttleAnimator shuttleAnimator1 =
    new AShuttleAnimatorWaitingForClearance(
            shuttle1, clearanceManager);
    displayShuttleAnimator(shuttleAnimator1);
    PlottedShuttle shuttle2 =
    new AnObservablePlottedShuttle(SHUTTLE2_X, SHUTTLE2_Y);
    displayShuttleFrame(shuttle2);
    ShuttleAnimator shuttleAnimator2 =
    new AShuttleAnimatorWaitingForClearance(
            shuttle2, clearanceManager);
    displayShuttleAnimator(shuttleAnimator2);
    displayClearanceManager(clearanceManager);
}
SIMULTANEOUS LAUNCH
**ANALOGIES**

- `proceedAll()`

- `proceed()`

- `waitFor Proceed()`

- Every one waiting proceeds

- One person proceeds
public class ABroadcastingClearanceManager extends AClearanceManager implements BroadcastingClearanceManager {
    @Row(1)
    @ComponentWidth(100)
    public synchronized void proceedAll() {
        notifyAll();
    }
}
**NOTIFYALL**

<table>
<thead>
<tr>
<th>wait()</th>
<th>toString()</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify()</td>
<td>equals()</td>
</tr>
<tr>
<td>notifyAll()</td>
<td>clone()</td>
</tr>
</tbody>
</table>

---

- **o.wait()**: makes the calling thread wait (in a queue) until the next `o.notify()` call (by some other thread).

- **o.notify()**: if some threads are waiting as a result of `o.wait()`, then unblocks the first one who waited.

- **o.notifyAll()**: if some threads are waiting as a result of `o.wait()`, then unblocks all of them.
public class ManualSimultaneousShuttleLaunchAnimation extends ManualShuttleTrafficControl {
    public static void main(String[] args) {
        ClearanceManager clearanceManager =
            new ABroadcastingClearanceManager();
        PlottedShuttle shuttle1 =
            new AnObservablePlottedShuttle(SHUTTLE1_X, SHUTTLE1_Y);
        displayShuttleFrame(shuttle1);
        ShuttleAnimator shuttleAnimator1 =
            new AShuttleAnimatorWaitingForClearance(shuttle1,
                                                        clearanceManager);
        displayShuttleAnimator(shuttleAnimator1);
        PlottedShuttle shuttle2 =
            new AnObservablePlottedShuttle(SHUTTLE2_X, SHUTTLE2_Y);
        displayShuttleFrame(shuttle2);
        ShuttleAnimator shuttleAnimator2 =
            new AShuttleAnimatorWaitingForClearance(shuttle2,
                                                        clearanceManager);
        displayShuttleAnimator(shuttleAnimator2);
        displayClearanceManager(clearanceManager);
    }
}
Separate Y axes so cannot synchronize on it.

Even if axes shared, do not really invoke any methods on Y axes.

Need to build our own synchronization lock that allows operations on multiple objects to be synchronized.

A la reserving a bunch of rooms for a conference.
public class ALock implements Lock {
    boolean locked;
    public synchronized void getLock() {
        if (locked) {
            try {
                wait();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        locked = true;
    }
    public synchronized void releaseLock() {
        locked = false;
        notify();
    }
}
public class AControlledShuttleAnimator extends AConcurrentShuttleAnimator {
    Lock lock;
    public AControlledShuttleAnimator(PlottedShuttle theShuttle,
                                        Lock aLock) {
        super(theShuttle);
        lock = aLock;
    }
    protected void animateYFromOrigin(PlottedShuttle shuttle,
                                       int animationStep, int animationPauseTime,
                                       int startY, int endY) {
        lock.getLock();
        super.animateYFromOrigin(shuttle, animationStep,
                                   animationPauseTime, startY, endY);
        lock.releaseLock();
    }
}
public class AutomaticAirTrafficControl {
    static final int SHUTTLE_FRAME_X = 50;
    static final int START_FRAME_Y = 50;
    static final int FRAME_WIDTH = 400;
    static final int FRAME_HEIGHT = 275;
    static final int ANIMATION_FRAME_X =
        SHUTTLE_FRAME_X + FRAME_WIDTH;
    static int shuttleNumber = 0;
    static void createAndDisplayShuttleAndAnimator(int shuttleX, int shuttleY, Lock aLock) {
        int frameY = START_FRAME_Y + shuttleNumber*FRAME_HEIGHT;
        PlottedShuttle shuttle =
            new AnObservablePlottedShuttle(shuttleX, shuttleY);
        OEFrame shuttleFrame = ObjectEditor.edit(shuttle);
        shuttleFrame.hideMainPanel();
        shuttleFrame.setLocation(SHUTTLE_FRAME_X, frameY);
        shuttleFrame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        ShuttleAnimator shuttleAnimator =
            new AControlledShuttleAnimator(shuttle, aLock);
        OEFrame animatorFrame = ObjectEditor.edit(shuttleAnimator);
        animatorFrame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        animatorFrame.setLocation(ANIMATION_FRAME_X, frameY);
        shuttleNumber++;
    }
}
public static void main(String[] args) {
    Lock lock = new ALock();
    createAndDisplayShuttleAndAnimator(100, 100, lock);
    createAndDisplayShuttleAndAnimator(100, 50, lock);
    createAndDisplayShuttleAndAnimator(50, 50, lock);
}
}
Wait, Notify, NotifyAll

- Methods provided by Object
- Can be used to make threads wait until some condition decided by our program is met.
- Wait on an object makes a thread block until a notify or notifyAll() is executed on that object
- All three methods must be executed in synchronized methods of the object.
- Waiting thread releases synchronized Object lock so some other object can execute notify() or notifyAll() on that object.
- Notify() and notifyAll() can be executed when condition to proceed occurs
- There are complicated ways to use them you will see in OS courses
public class ABMISpreadsheetInteractiveDemoer {
    public static void main (String[] args) {
        ClearanceManager clearanceManager = new AClearanceManager();
        ObjectEditor.edit(clearanceManager);
        BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        OEFrame editor = ObjectEditor.edit(bmiSpreadsheet);
        ThreadSupport.sleep (2000);
        editor.select(bmiSpreadsheet, "Weight");
        bmiSpreadsheet.setWeight(70);
        editor.refresh();
        ThreadSupport.sleep (2000);
        editor.select(bmiSpreadsheet, "Height");
        bmiSpreadsheet.setHeight(0);
        editor.refresh();
        ThreadSupport.sleep (2000);
        editor.select(bmiSpreadsheet, "Weight");
        bmiSpreadsheet.setWeight(0);
        editor.refresh();
    }
}
public class ABMISpreadsheetInteractiveDemoer {
    public static void main (String[] args) {
        ClearanceManager clearanceManager = new AClearanceManager();
        ObjectEditor.edit(clearanceManager);
        BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        OEFrame editor = ObjectEditor.edit(bmiSpreadsheet);
        clearanceManager.waitForProceed();
        editor.select(bmiSpreadsheet, "Weight");
        bmiSpreadsheet.setWeight(70);
        editor.refresh();
        clearanceManager.waitForProceed();
        editor.select(bmiSpreadsheet, "Height");
        bmiSpreadsheet.setHeight(0);
        editor.refresh();
        clearanceManager.waitForProceed();
        editor.select(bmiSpreadsheet, "Weight");
        bmiSpreadsheet.setWeight(0);
        editor.refresh();
    }
}
INTERACTIVE DEMOER