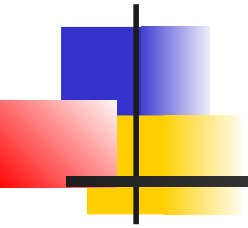


State Machine

Designing components using a state machine model.

When State Matters

Some components behave differently depending on the *state* they are in.

Examples:

- Alarm Clock - showing time, setting alarm, ringing
- Stop Watch - running or stopped
- Calculator - has value or not, has operand or not
- *parsers* - state depends on previous value

Simple Example: Stopwatch

Stopwatch *behaves differently* when it is running and stopped.

Identify States

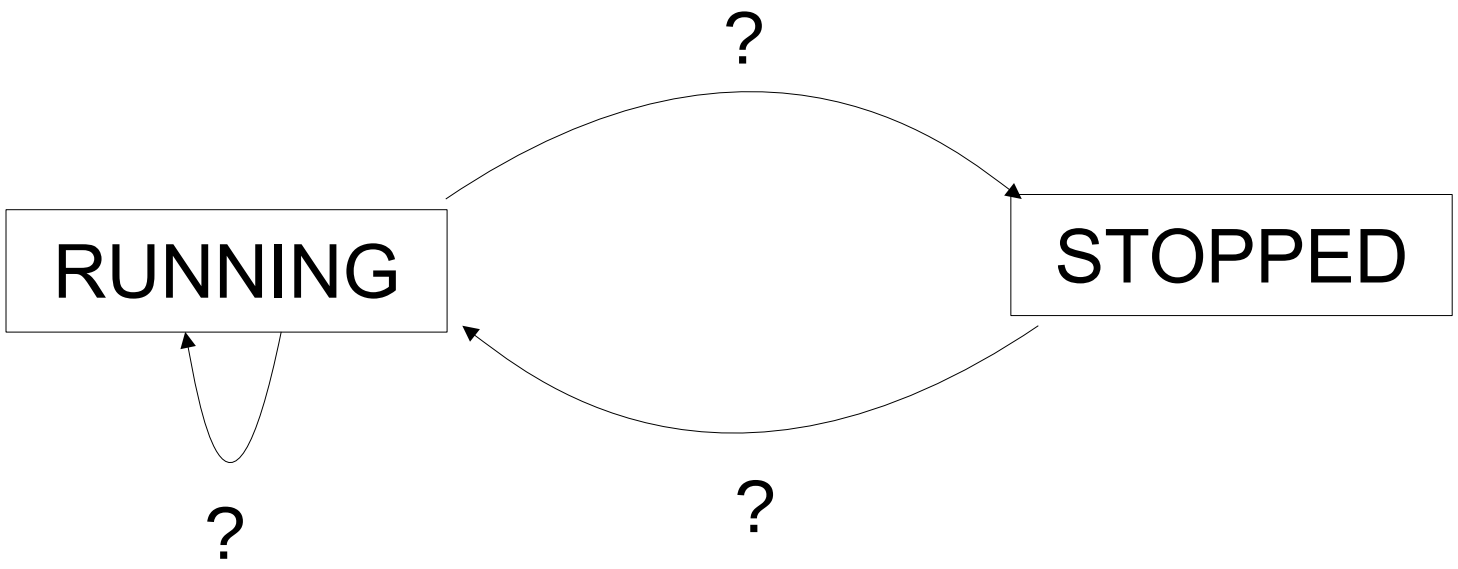
Stopwatch states: RUNNING and STOPPED

RUNNING

STOPPED

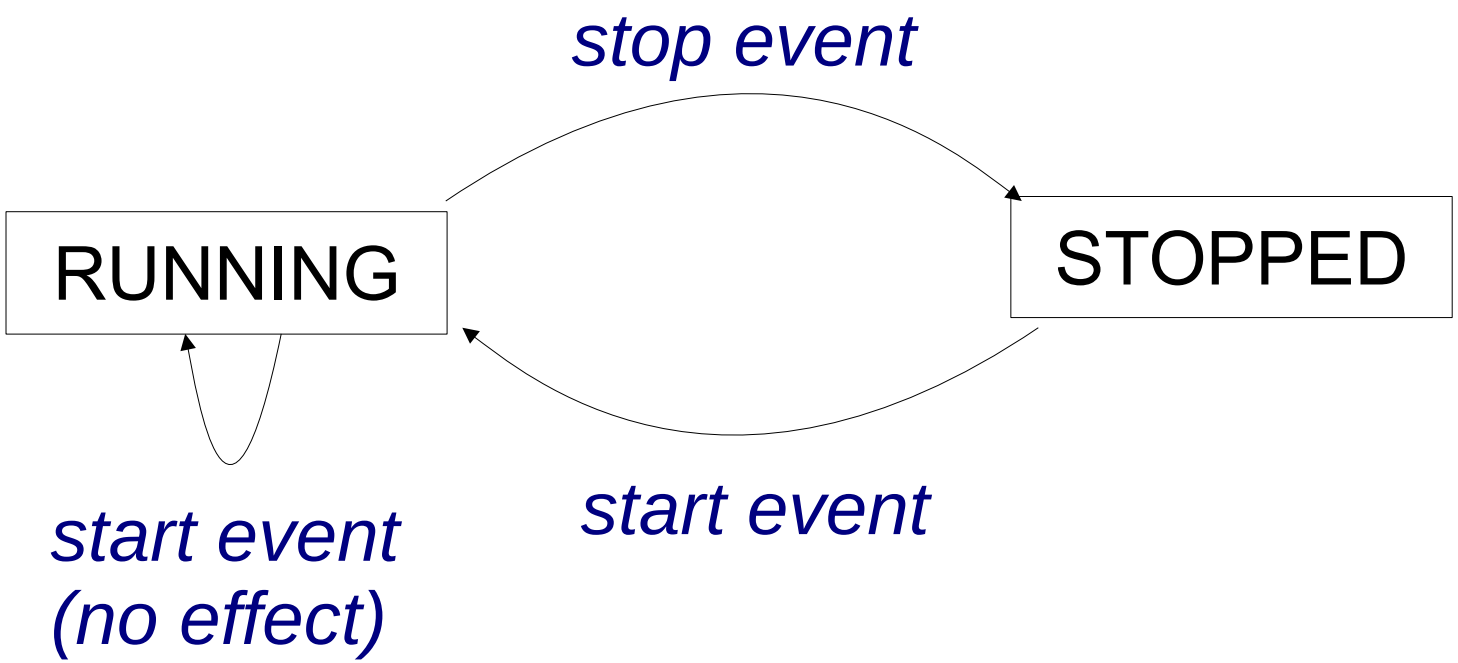
Events

Events are actions that can cause a state machine to change state.
What events cause a stopwatch to change state?



Events

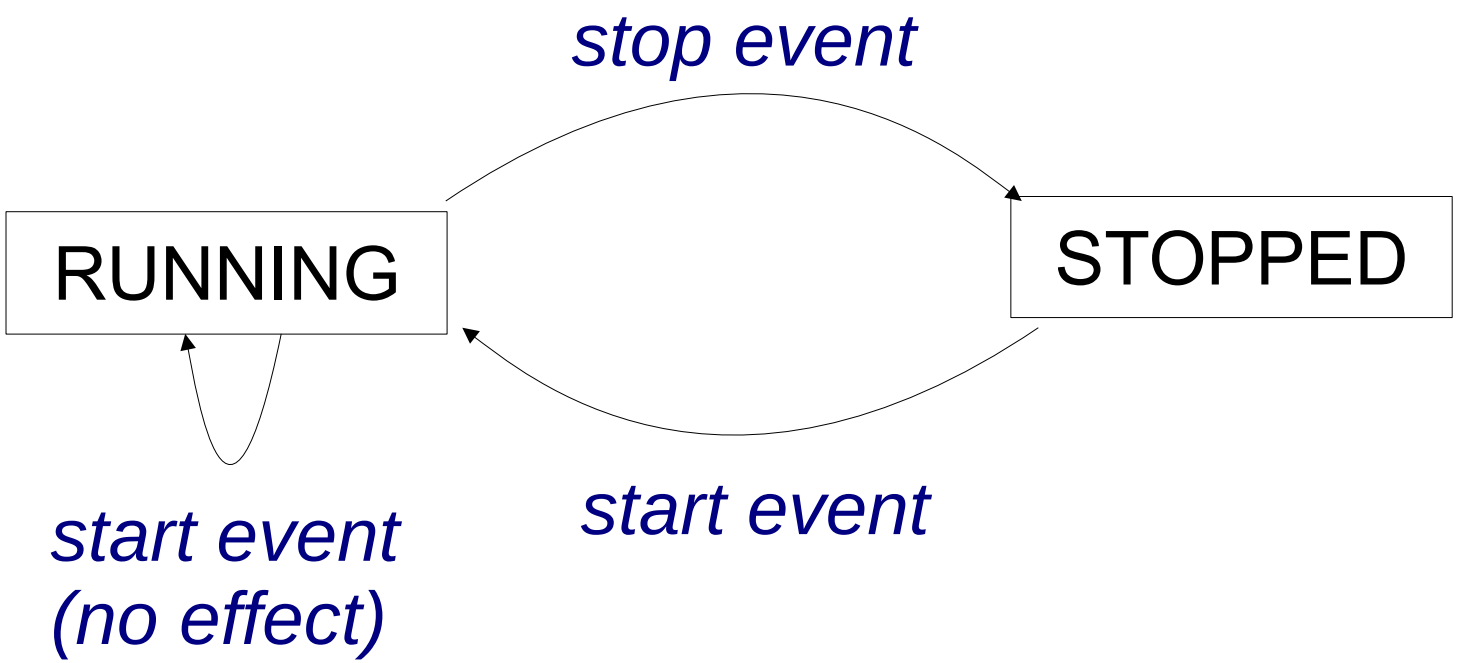
Events can be *user actions* or other external events.



Actions

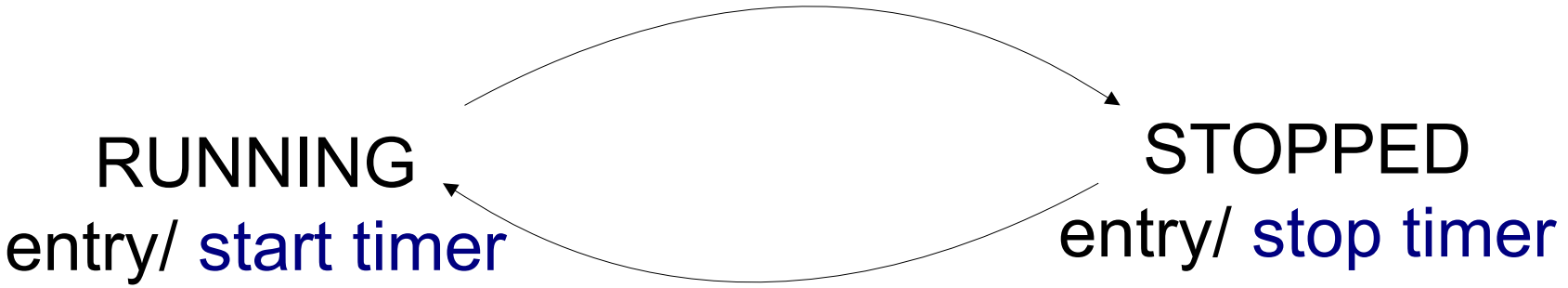
The stopwatch performs an *action* in response to an event.

As shown here, some events cause it to change state.



More Actions

The stopwatch performs an *action* whenever it enters a new state.



Activities

An activity is something that lasts for some time.
An action is (nearly) instantaneous.

In the StopWatch UI, "*update display*" is an activity.

Programming a State Machine

Design the state machine first – step by step.

1. Identify the **states**
2. Identify **events***:* external and internally generated
3. Identify **actions** & **activities** the state machine performs
4. Draw a **state machine** diagram.

Finally,

5. Code *state-dependent behavior* using state machine.

What behavior depends on state?

We use the attribute `running` to keep track of state.

```
class StopWatch {
    private boolean running;

    public void start( ) {
        if (running) return;
        startTime = System.nanoTime();
        running = true; // change state
    }
```

What behavior depends on state?

`start(), stop(), getElapsed(), isRunning()`

```
public double getElapsed() {
    if (running)
        return (System.nanoTime()-startTime)
                * NANOSECONDS;
    else
        return (stopTime-startTime)
                * NANOSECONDS;
}
public void stop() {
    if (! running ) return;
    stopTime = System.nanoTime();
    running = false;
}
```

Two Implementations of State Machine

1. State variable

   - use a variable to represent state

   - use a "switch" statement inside state-dependent methods. States are "case" in the switch.

2. Object-Oriented Approach

   - an Interface for State.

   - one class for each concrete state.

State Variable

We used a `boolean` variable (`running`) to record the state.

This works when there are only 2 states.

For more states we need another type of state variable.

Example: a StopWatch with Start, Stop, and Hold states.

State variable to remember state

```
// use "int" or "char"
class StopWatch {

    final int STOPPED = 0;
    final int RUNNING = 1;
    final int HOLDING = 2;

    int state;
    public void stop( ) {
       switch(state) {
       case RUNNING:
         // handle "stop" when
         // stopwatch is in
         // RUNNING state
```

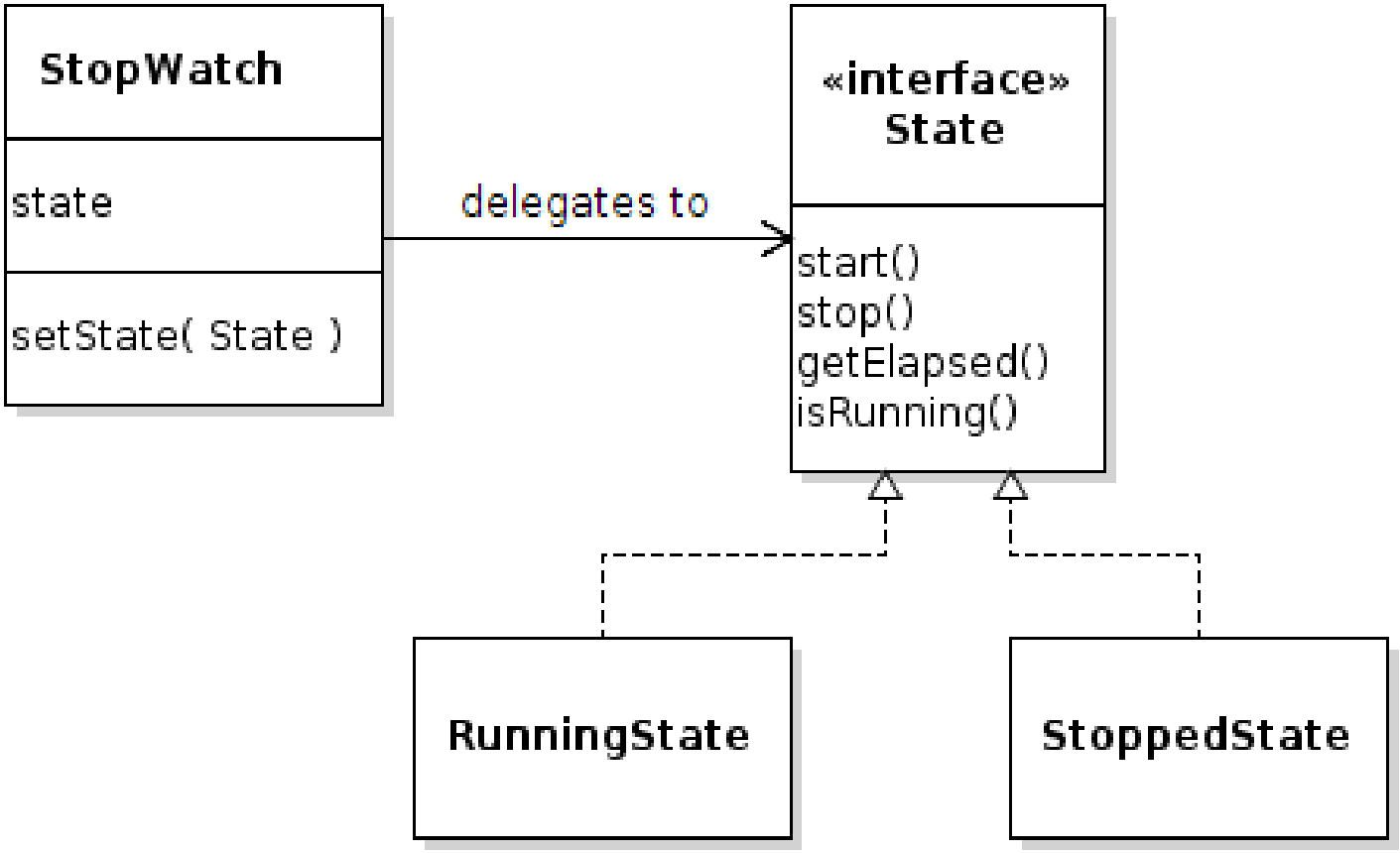
```
// use an enum
class StopWatch {
   enum State {
      STOPPED,
      RUNNING,
      HOLDING;
   }
   State state;
   public void stop() {
      switch(state) {
       case RUNNING:
         // handle "stop" when
         // stopwatch is in
         // RUNNING state
```

State variable with "switch"

```
// Stopwatch with enum for states
class StopWatch {
    private State state;
    public void start() {
        switch (state) {
        case RUNNING:
            return;  // do nothing
        case STOPPED:
            // start the stopwatch
            startTime = System.nanoTime();
            state = State.RUNNING;
            return;
        case HOLD:
            ...
```

The O-O Approach

Use *Objects* to encapsulate state and the behavior that depends on state.
The *context* (StopWatch) delegates behavior to state objects.



Interface for State-dependent Behavior

Actual states must each provide this behavior.

```
public class State {
    /** Handle start event */
    public void start();

    /** Handle stop event */
    public void stop();

    /** Get the elapsed time */
    public double getElapsed();

    /** Inquire if stopwatch is running */
    public boolean isRunning();
```

States without "if"

Each state knows exactly what to do.

```
public class RunningState {
    /** Handle start event */
    public void start() {
        // do nothing -- already running
    }

    /** Inquire if stopwatch is running */
    public boolean isRunning() {
        return true;
    }
```

Delegating Behavior

Delegate means "let someone else do it".

Stopwatch delegates behavior to the **`state`**.

```
public class StopWatch {
    private State state;

    public void start( ) {
       state.start();
    }
    public void stop() { state.stop(); }
    public double getElapsed() {
        return state.getElapsed();
    }
```

State Objects and Changing State

The *context* (StopWatch) needs a setState method as a way of changing the state.

```
// Create states with a reference to stopwatch
final State RUNNING = new RunningState(this);
final State STOPPED = new StoppedState(this);
// initially the Stopwatch is stopped
private State state = STOPPED;

// a method for changing the state
public void setState(State newstate) {
    this.state = newstate;
}
```

Example of Changing State

If the stopwatch is running and the Stop button is pressed, then change to stopped state...

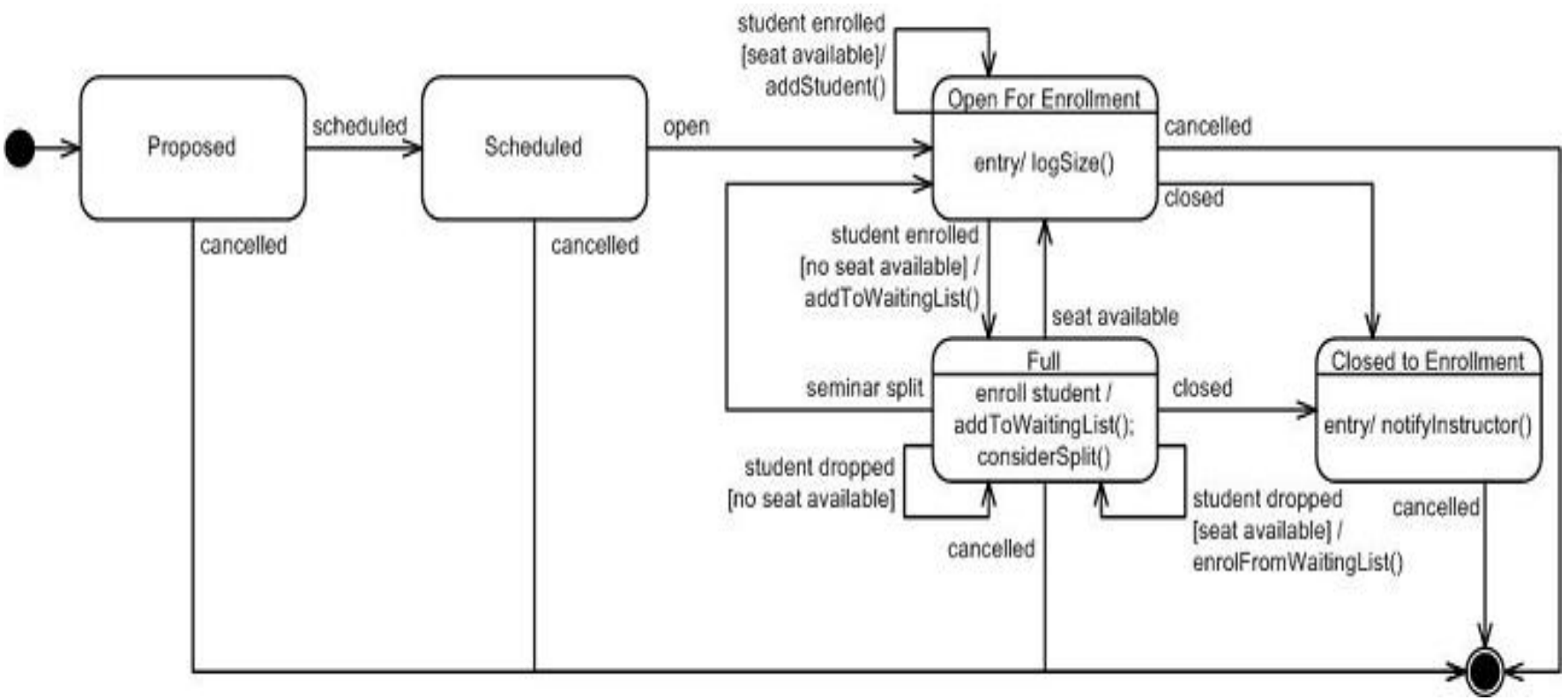
```
class RunningState implements State {
   private StopWatch context;

   public RunningState(StopWatch sw) {
       this.context = sw;
   }

   public void stop() {
      context.stopTime = System.nanoTime();
      // CHANGE STATE
      context.setState( context.STOPPED );
   }
```

UML State Machine Diagram

States for course enrollment.



UML State Machine Diagram

Read *UML Distilled*, chapter 10.

Also good: *UML for Java Programmers*, chapter 10.

Exercise: Syllable Counter

Count the syllables in a word.

As a heuristic, count ***vowel sequences*.**

Examples:

object = (o)bj(e)ct = 2 vowel sequences

beauty = b(eau)t(y) = 2 vowel sequences

Special cases:

l(a)y(ou)t = *treat "y" as consonant if after other vowel*

l(a)the = don't count final "e" if it is a single vowel

m(o)v(ie) = 2 vowel seq. "final e" rule doesn't apply here.

th(e) = exception: count final "e" if it is the *only* vowel

anti-oxidant = (a)nt(i)-(o)x(i)d(a)nt "-" is non-vowel

Example Words

How many vowel sequences in these words?

```
remarkable

selfie

county

coincidentally

she

mate

isn't
```

States for Syllable Counter

Using the above vowel counting houristic, what are the states?

consonant - last seen letter is a consonant

single_vowel - recent letter was first vowel in sequence

multi_vowel - recent letter was vowel after another vowel

hyphen - most recent char is a hyphen

nonword - the character sequence is not a word (violates some rule for word such as containing an invalid character, .e.g. a digit)

States for Syllable Counter

What should be the *starting state?*

See Syllable Counter Lab on for details.

Exercise: Skytrain Ticket Machine

Skytrain Ticket Machine

1. What are the states?
2. What are the events?
3. What actions/activities does ticket machine perform?
4. Draw a UML State Machine Diagram.