Animation with Timers

A single threaded approach

- Using threads has a down side.
  - Thread safety has to be ensured.
  - Deadlock has to be avoided
- Using timer objects is an alternative.
- A timer object is an object that periodically messages a command object.
- These messages are on the GUI event thread so multithreading is not an issue

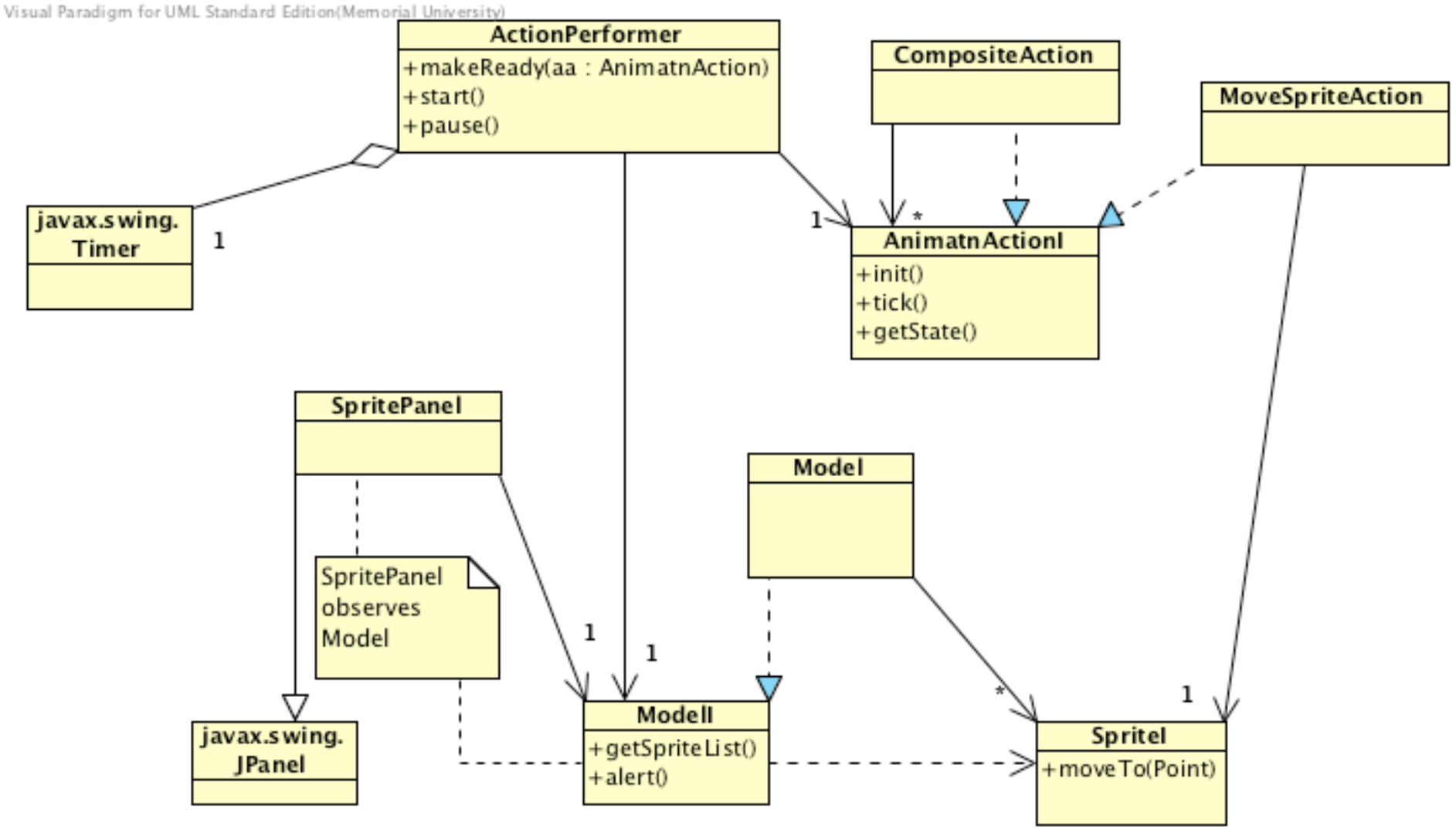
The "simple time animation" example

- The timer sends events to a "sprite" object which updates its state.
- The timer also sends repaint messages to the display system.

The action based animation example

- Here the “sprites” are passive objects that can paint them selves and be repositioned.
- The behaviour is factored out into Action objects.
- An ActionPerformer object encapsulates the Timer and sends “tick” messages to an Action object.
- Composite actions can be formed from a list of individual actions.

ActionPerformers



Animation Actions

```
public enum AnimatnActionState { INIT, DONE }

public interface AnimatnActionI {
    public void init( ) ;
    public void tick( ) ;
    public AnimationActnState getState( ) ;
}
```

An ActionPerformers controls an AnimatnActionI

```
public class ActionPerformer extends Observable {
    public enum State {
            READY, RUNNING, PAUSED, DONE } ;
    private State state = State.DONE ;
    private ModelI model;
    private AnimationActionI action;
    private Timer timer ;
```

An ActionPerformer sends ticks to its AnimatnActionI

```
public ActionPerformer( ModelI model ) {
        this.model = model;
        timer = new Timer(20, new ActionListener() {
           @Override public void
                   actionPerformed(ActionEvent e) { tick() ; } } ) ;
}
private void tick() {
    if( state == RUNNING ) { action.tick();
        if( action.getState() == ActionState.DONE ) {
                   timer.stop() ; state = State.DONE ;
                   action = null ; setChanged() ; }
        model.alert() ; notifyObservers() ; } }
```

ActionPerformer: remaining methods

- **Remaining methods change and report the ActionPerformer's state**

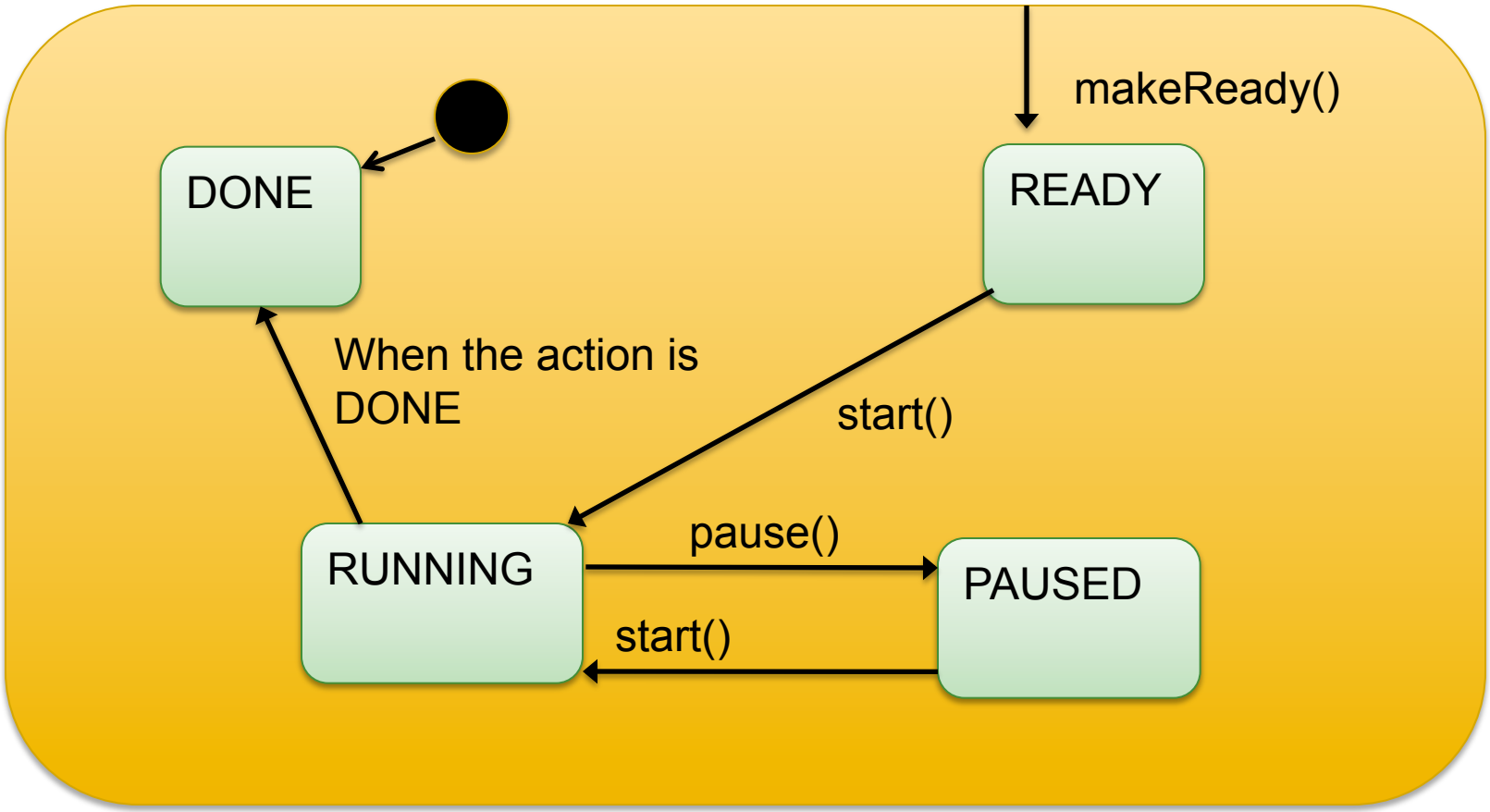
  public void makeReady(AnimatnActionI action)

  public void start()

  public void pause()

  public State getState()

ActionPerformer state chart



(makeReady can be sent to the actionPerformer regardless of state)