CMSC 131
Object-Oriented Programming I
Dynamic Systems and State-Transition Diagrams
Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane
Overview

- Dynamic Systems
- State-Transition Diagrams
Dynamic Systems

- **Dynamic Systems**: Systems that change dynamically over time. Such systems arise naturally when writing programs involving **graphical user interfaces** (video games, interactive graphics). Some issues:
  - How does the system respond to external events or stimuli? Called **reactive** or **event-driven** systems
  - **State transition**: Most dynamic systems are defined in terms of information called its **state**
    - What are the **possible states** the system can be in?
    - What sorts of **state transitions** are possible, and under what circumstances do transitions occur?
    - What **actions** are performed in each state?
**Dynamic Systems**

- **Examples:**
  - **DVD Player/Recorder:** Behavior to remote control commands varies depending on the operating state: recording, playback, idle.
  - **Figure drawing program:** (e.g. Paint) The meaning of mouse actions depends on the drawing state: line, curve, ellipse, rectangle, polygon.
  - **Video game:** The meaning of user inputs depends on the current context in which the game is operating.
  - **Digital watch:** Has various modes (clock, stop watch, timer) and the meaning of buttons varies with the mode.

- How do we **design programs** for such event-driven systems?
**State Transition Systems**

- These systems have a number of elements in common:
  - **Events**: Inputs/Stimuli come in the form of events (rather than traditional text prompt + text input)
  - **State**: The behavior depends on *internal information* (which the user cannot see) called the system’s **state** or **context**
  - **Transitions**: Events can cause changes in the context and other state information
  - **Actions**: Actions (which the user may or may not see) are performed in response to each event/transition
  - **(Spontaneous actions)**: Some actions take place without any user input. (Example: animation in a video game.) These can be modeled as responses to system-generated events, like timer events
Let us consider the case of a simple interactive calculator

Events: occur when user hits the keys
State: Operands, memory, internal state of the computation (more about this later)
Actions: Perform calculations, update the display

What internal state information is needed?
Example: “3 4 + 5 6 = ”
When the “=” is processed, the calculator has saved the following information internally:
First operand: “34” (call this v1)
Operator: “+” (call this op)
Second operand: “56” (call this v2)
It must also know which operand it is reading, first or second
Calculator

- **Calculator**: Has three **states**, or **contexts**:
  - **Reading-First-Operand (RFO)**: reading digits for the first operand
  - **Reading-Second-Operand (RSO)**: reading digits for the second operand
  - **Error (ERR)**: An error occurs (e.g., invalid operand or divide by 0).

**Example:**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Context:</th>
<th>Action:</th>
<th>Display:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(init)</td>
<td>RFO</td>
<td>reset(v1)</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>RFO</td>
<td>v1 += &quot;3&quot;</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>RFO</td>
<td>v1 += &quot;4&quot;</td>
<td>34</td>
</tr>
<tr>
<td>+/-</td>
<td>RFO</td>
<td>v1 ← procUnary: &quot;34&quot;, &quot;+/-&quot;</td>
<td>-34</td>
</tr>
<tr>
<td>+</td>
<td>RSO</td>
<td>op ← &quot;+&quot;; reset(v2)</td>
<td>-34</td>
</tr>
<tr>
<td>5</td>
<td>RSO</td>
<td>v2 += &quot;5&quot;</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>RSO</td>
<td>v2 += &quot;6&quot;</td>
<td>56</td>
</tr>
<tr>
<td>*</td>
<td>RSO</td>
<td>v1 ← procBinary: &quot;-34&quot;, &quot;+&quot;, &quot;56&quot;</td>
<td>22</td>
</tr>
<tr>
<td>reset(v2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RSO</td>
<td>v2 += &quot;2&quot;</td>
<td>2</td>
</tr>
<tr>
<td>1/x</td>
<td>RSO</td>
<td>v2 ← procUnary: &quot;2&quot;, &quot;/x&quot;</td>
<td>0.5</td>
</tr>
<tr>
<td>=</td>
<td>RFO</td>
<td>v1 ← procBinary: &quot;22&quot;, &quot;*&quot;, &quot;0.5&quot;</td>
<td>11</td>
</tr>
</tbody>
</table>
How does the calculator know what operation to perform with each event? This is based on its state, or context (RFO, RSO, ERR).

We can describe the behavior using a state-transition diagram:

- **Nodes**: represent possible states the system can be in. A black circle is the initial or starting state.
- **Arcs or Edges**: represent possible transitions. Each is labeled with a pair “Event/Action” where:
  - **Event**: event that triggers the transition
  - **Action**: action/computation performed as a result of the event
(Simplified) State-Transition Diagram

Initial state

Digit(x) / \{v1 += x\}

UnaryOp(x) / \{v1 \leftarrow x v1\}

Clear: \{reset(v1)\}

BinaryOp(x) / \{op \leftarrow x; reset(v2)\}

Assign / \{v1 \leftarrow v1 op v2\}

UnaryOp(x) / \{v2 \leftarrow x v2\}

Digit(x) / \{v2 += x\}

If there is no transition for a particular event from some state, then the event is ignored.

To keep the diagram simple, these two transitions are the same for all states.
Programming State-Transition Diagrams

- To programm state-transition diagrams you can use **if-the-else** and/or **switch** statements to control the processing. You can also use the state design pattern (advanced topic).

  **Example:**

  ```java
  if ( event == X ) {
      // some event X encountered
      switch ( state ) {
          case STATE1:
              // processing for event X in state 1
              break;
          case STATE2:
              // processing for event X in state 2
              break;
      }
  } else if ( event == Y ) {
      // event Y encountered
      // same thing
  } // etc...
  ```