TA Programming of Interactive Systems

http://davidbonnet.com/is

David Bonnet
bonnet@lri.fr

Cédric Fleury
cfleury@lri.fr

Introduction to SwingStates

Exercises 5 & 6

State Machines

Finite state machines (FSM):

States represent interaction states:

Idling, dragging, drawing, …

Transitions are triggered by events:

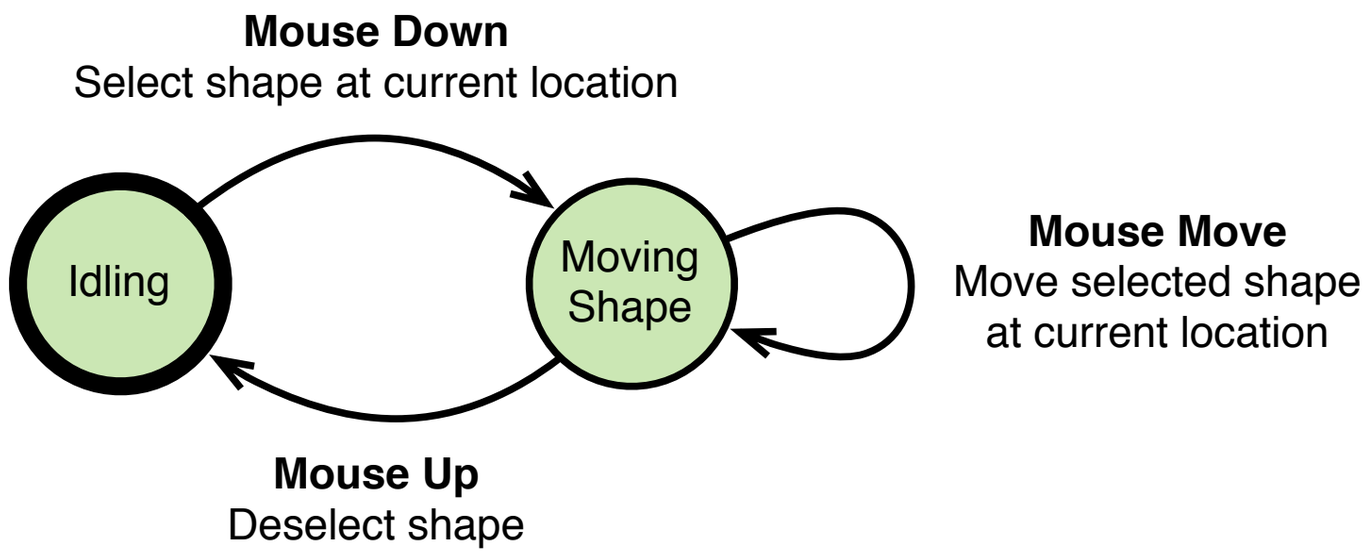
User events (mouse click, key press, …)

System events (timeout, incoming packet, …)

Custom events (gesture recognition, …)

Example

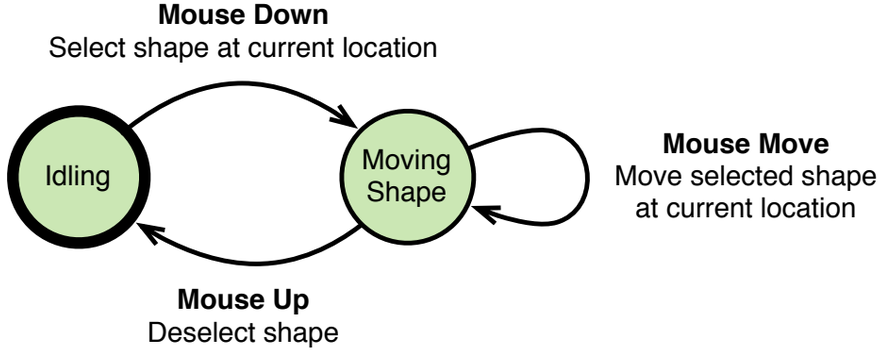
Dragging a shape:



Example

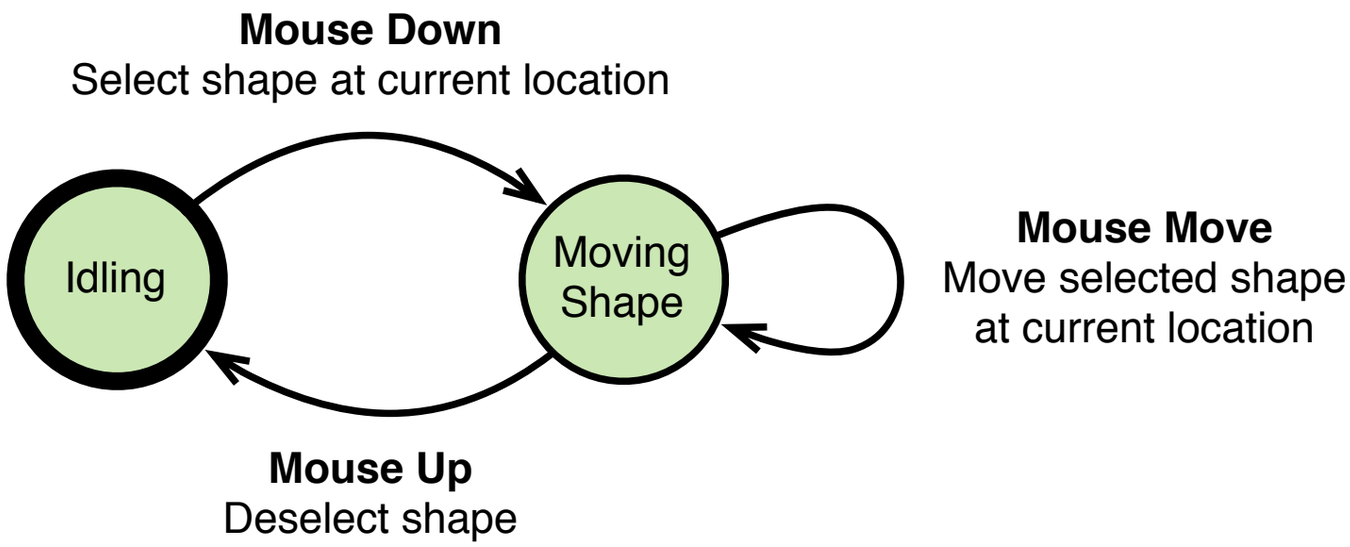
Implementing this with callbacks:

```
Shape dragged = null;
new MouseAdapter() {
     public void mousePressed(MouseEvent e) {
          // dragged is initialized
     }
     public void mouseReleased(MouseEvent e) {
          // dragged is set back to null
     }
}
new MouseMotionAdapter() {
     public void mouseDragged(MouseEvent e) {
          // dragged is translated
     }
}
```



Example

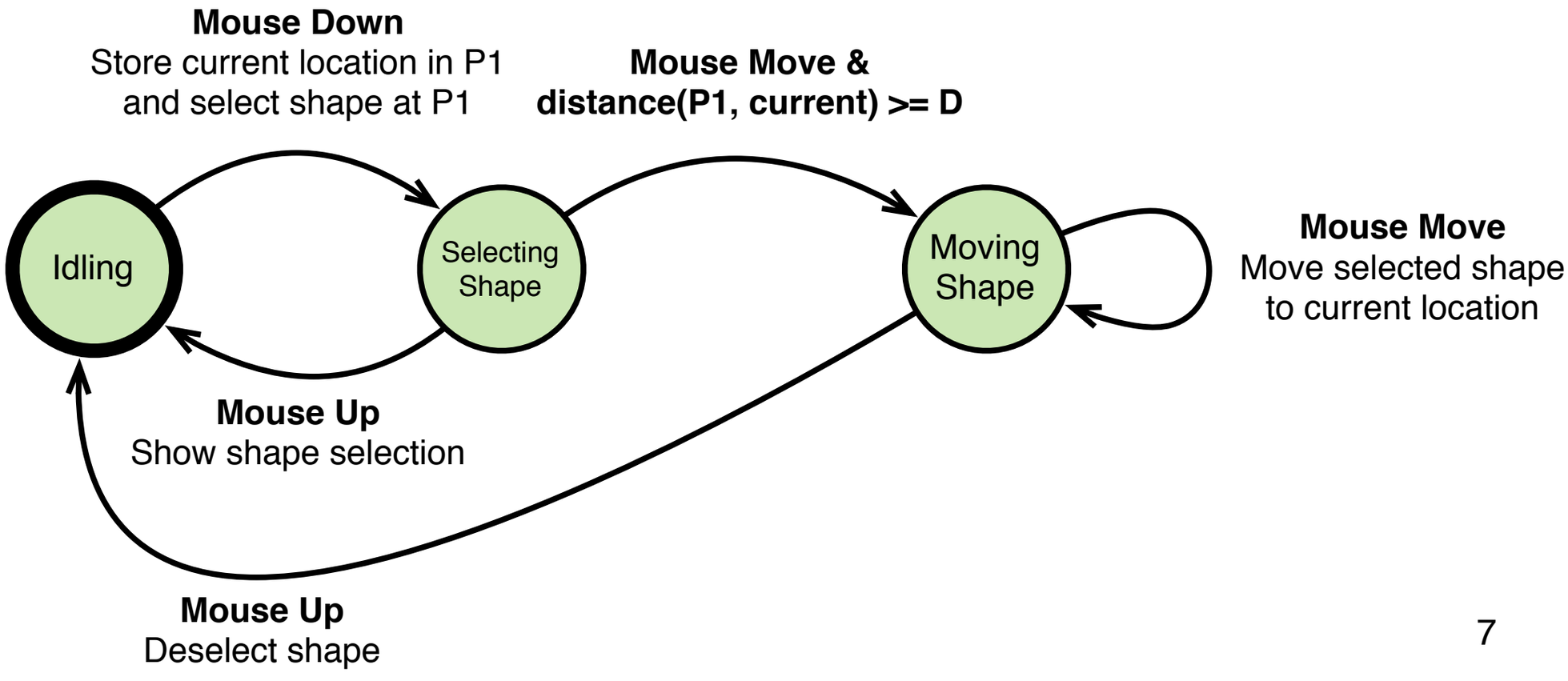
This is okay for simple state machines.



Example

This is okay for simple state machines.

Let's consider the ability to select and drag an object:



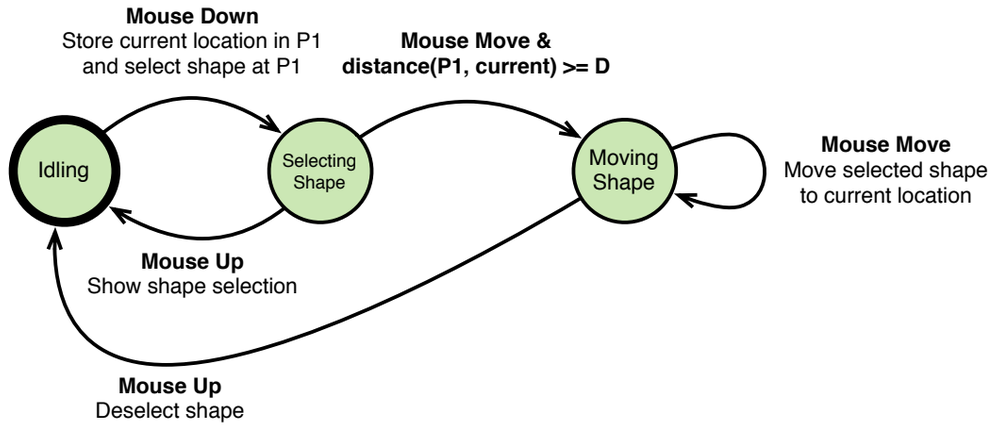
Example

Implementing the FSM with callbacks:

```
Shape dragged = null;
boolean dragging = false;
new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
            // dragged is initialized
      }
      public void mouseReleased(MouseEvent e) {
            // dragged is set back to null
            // shape is selected if not dragged

      }
}
new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent e) {
            if (!dragging) {
                  // check if dragging occurs

            } else {

                  // drag shape

            }

      }
}
```

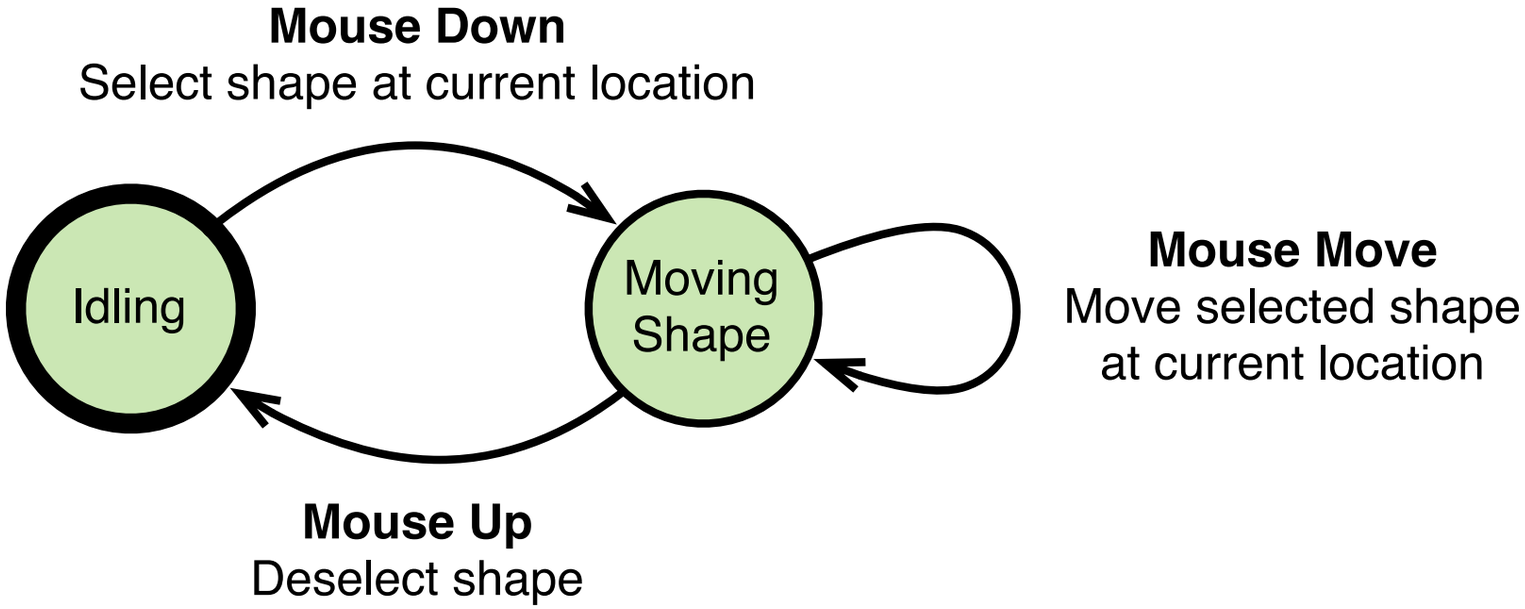
State variables

Separate listeners



SwingStates

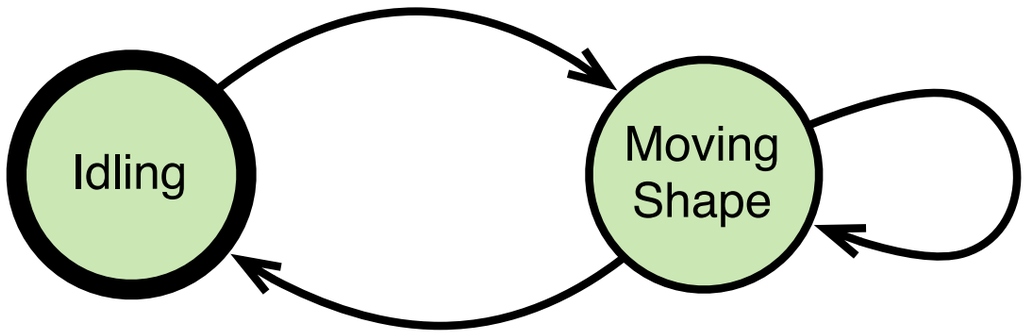
Let's implement the state machine with SwingStates. Here is the first example:



SwingStates

We change the name of the transitions to match the terminology of SwingStates:

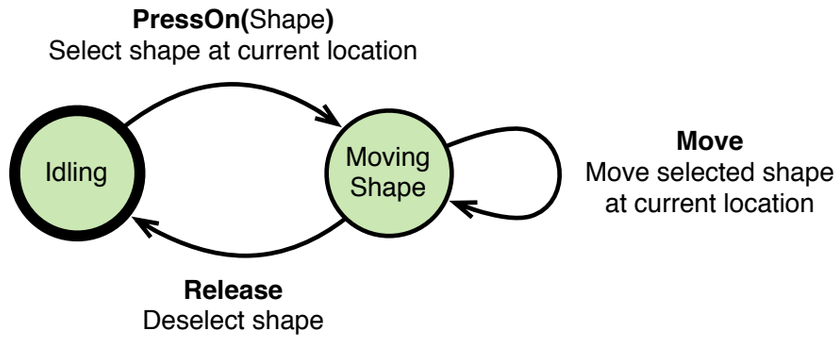
PressOn(Shape**)**
Select shape at current location



Move
Move selected shape at current location

Release
Deselect shape

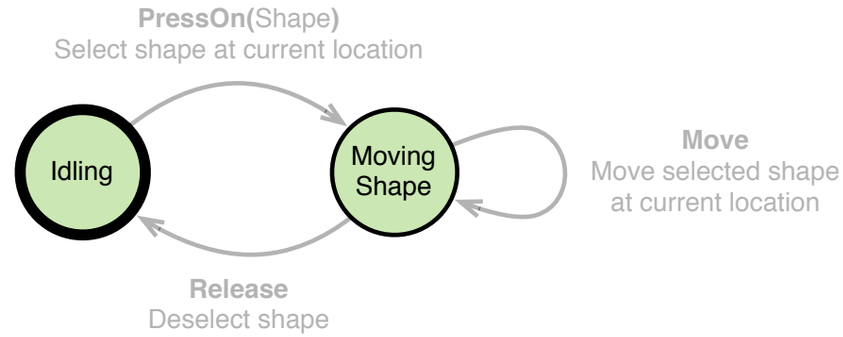
SwingStates



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
      // put states here
};
```

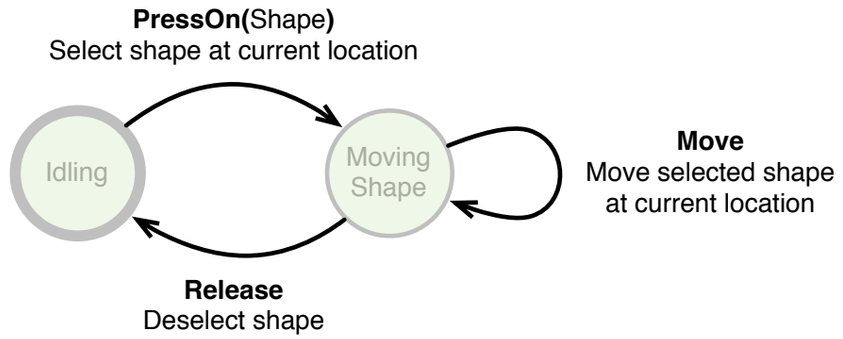
SwingStates



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
      State idling = new State() {
            // put transitions here
      };
      State movingShape = new State() {
            // put transitions here
      };
};
```

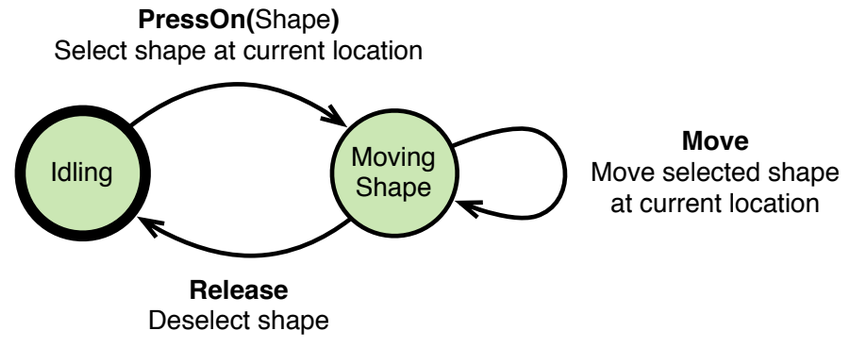
SwingStates



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
    State idling = new State() {
        Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
            public void action() { /* select current shape */ }
        };
    };
    State movingShape = new State() {
        Transition move = new Move() { // transition to the current state
            public void action() { /* move selected shape */ }
        };
        Transition release = new Release(BUTTON1, "=> idling") {
            public void action() { /* deselect shape */ }
        };
    };
};
```

SwingStates



Implementing the FSM with SwingStates:

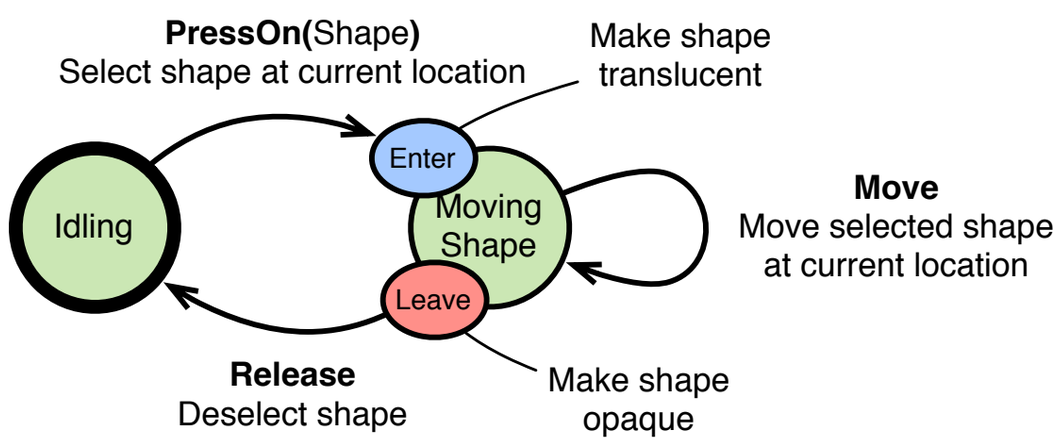
```
StateMachine sm = new StateMachine() {
    State idling = new State() {
        Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
            public void action() { /* select current shape */ }
        };
    };
    State movingShape = new State() {
        Transition move = new Move() { // transition to the current state
            public void action() { /* move selected shape */ }
        };
        Transition release = new Release(BUTTON1, "=> idling") {
            public void action() { /* deselect shape */ }
        };
    };
};
```

Enter / Leave

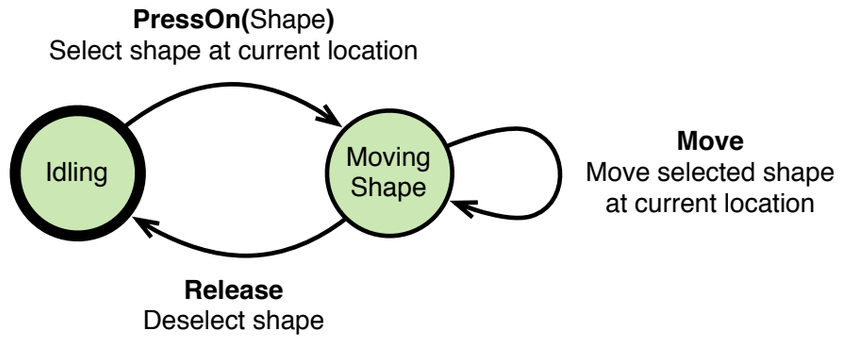
In general, the user should know in which state the system is. To that end, **actions** can be triggered when **entering** or **leaving** a state to express this change.

Example:

When being dragged, the shape becomes translucent:



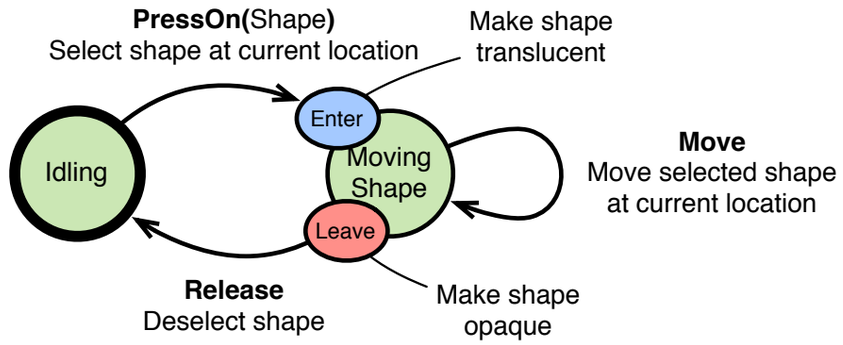
Enter / Leave



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
      State idling = new State() {
            Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
                  public void action() { /* select current shape */ }
            };
      };
      State movingShape = new State() {
            Transition move = new Move() { // transition to the current state
                  public void action() { /* move selected shape */ }
            };
            Transition release = new Release(BUTTON1, "=> idling") {
                  public void action() { /* deselect shape */ }
            };
      };
};
```

Enter / Leave



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
    State idling = new State() {
        Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
            public void action() { /* select current shape */ }
        };
    };
    State movingShape = new State() {
        public void enter() { /* make shape translucent */ }
        Transition move = new Move() { // transition to the current state
            public void action() { /* move selected shape */ }
        };
        Transition release = new Release(BUTTON1, "=> idling") {
            public void action() { /* deselect shape */ }
        };
        public void leave() { /* make shape opaque */ }
    };
};
```

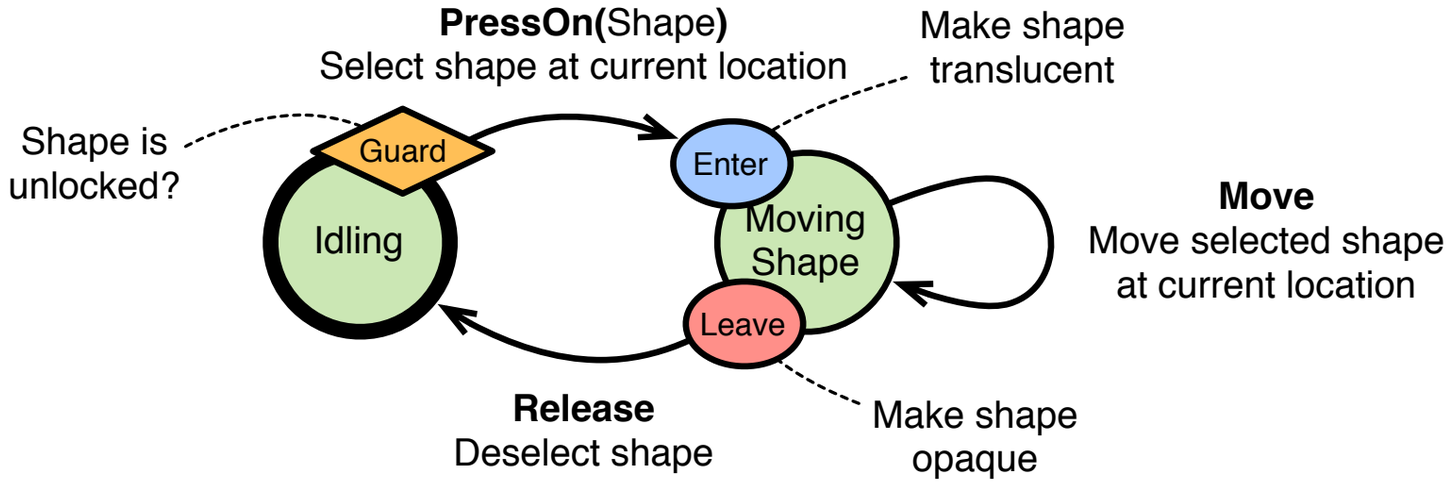
Guard

Transitions can be moderated by a **guard**.

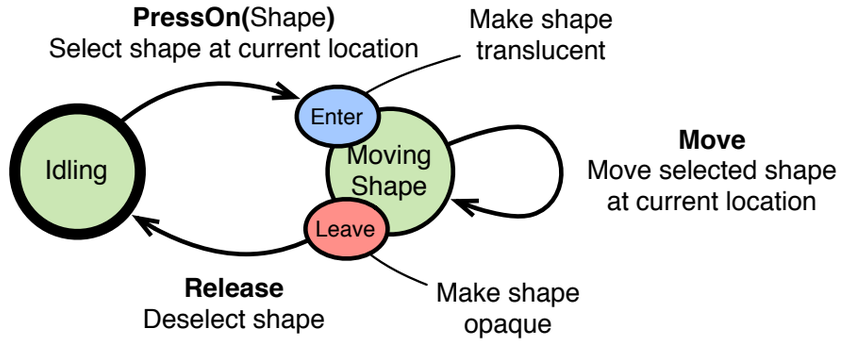
If the **boolean** it returns is true, the transition will happen.

Example:

Only **unlocked** shapes can be moved:



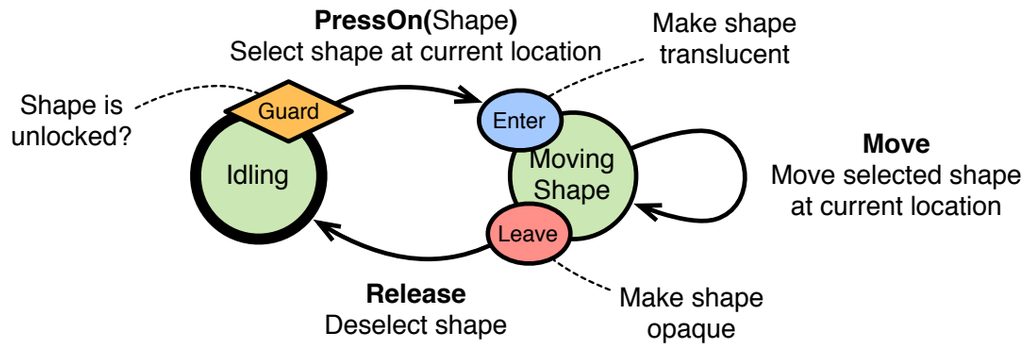
Guard



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
      State idling = new State() {
            Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
                  public void action() { /* select current shape */ }
            };
      };
      State movingShape = new State() {
            public void enter() { /* make shape translucent */ }
            Transition move = new Move() { // transition to the current state
                  public void action() { /* move selected shape */ }
            };
            Transition release = new Release(BUTTON1, "=> idling") {
                  public void action() { /* deselect shape */ }
            };
            public void leave() { /* make shape opaque */ }
      };
};
```

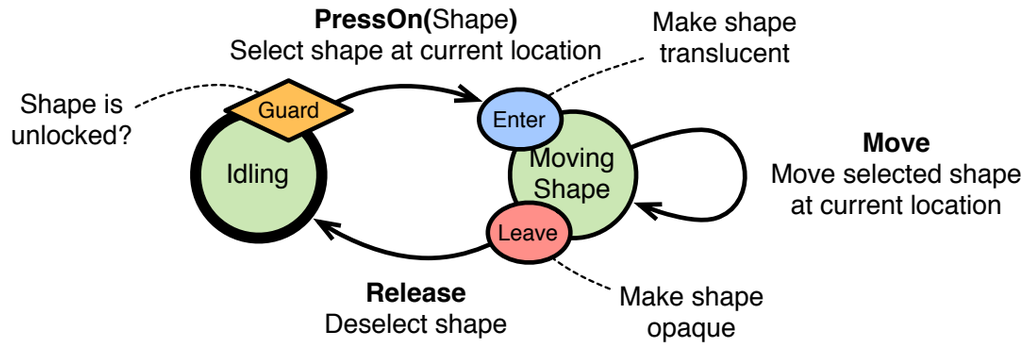
Guard



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
    State idling = new State() {
        Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
            public boolean guard() { /* return current shape is unlocked */ }
            public void action() { /* select current shape */ }
        };
    };
    State movingShape = new State() {
        public void enter() { /* make shape translucent */ }
        Transition move = new Move() { // transition to the current state
            public void action() { /* move selected shape */ }
        };
        Transition release = new Release(BUTTON1, "=> idling") {
            public void action() { /* deselect shape */ }
        };
        public void leave() { /* make shape opaque */ }
    };
};
```

Summary



Implementing the FSM with SwingStates:

```
StateMachine sm = new StateMachine() {
    State idling = new State() {
        Transition press = new PressOnShape (BUTTON1, "=> movingShape") {
            public boolean guard() { /* return current shape is unlocked */ }
            public void action() { /* select current shape */ }
        };
    };
    State movingShape = new State() {
        public void enter() { /* make shape translucent */ }
        Transition move = new Move() { // transition to the current state
            public void action() { /* move selected shape */ }
        };
        Transition release = new Release(BUTTON1, "=> idling") {
            public void action() { /* deselect shape */ }
        };
        public void leave() { /* make shape opaque */ }
    };
};
```

Calling Order

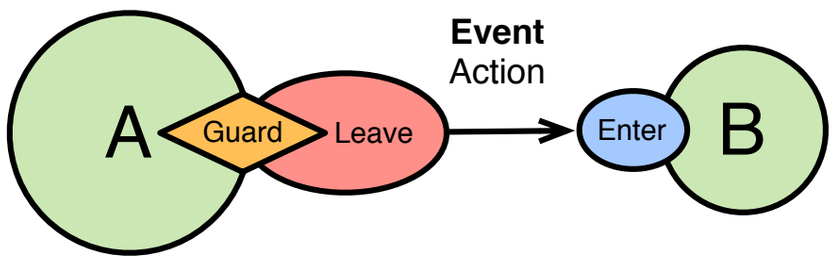
When a transition from a state A to a state B occurs, the following methods are called:

Transition.guard()

StateA.leave()

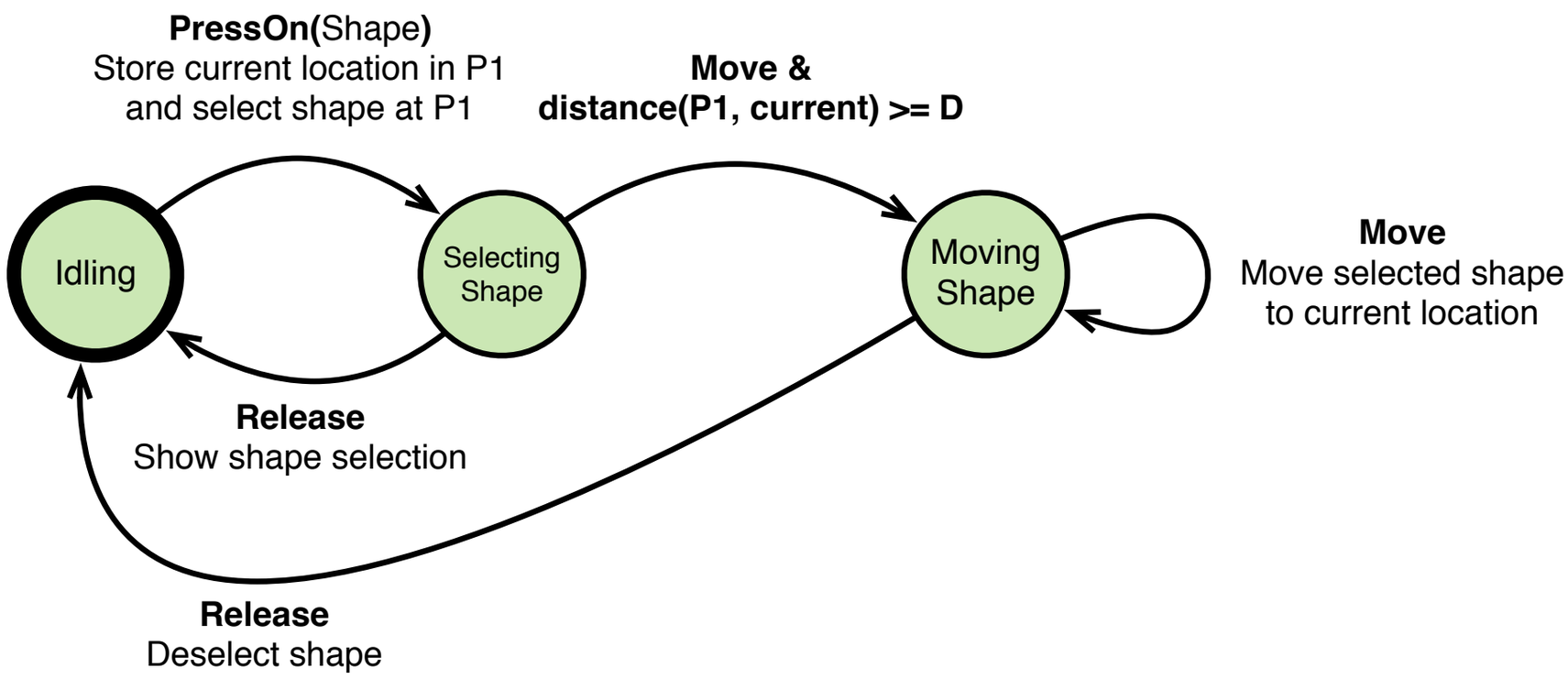
Transition.action()

StateB.enter()

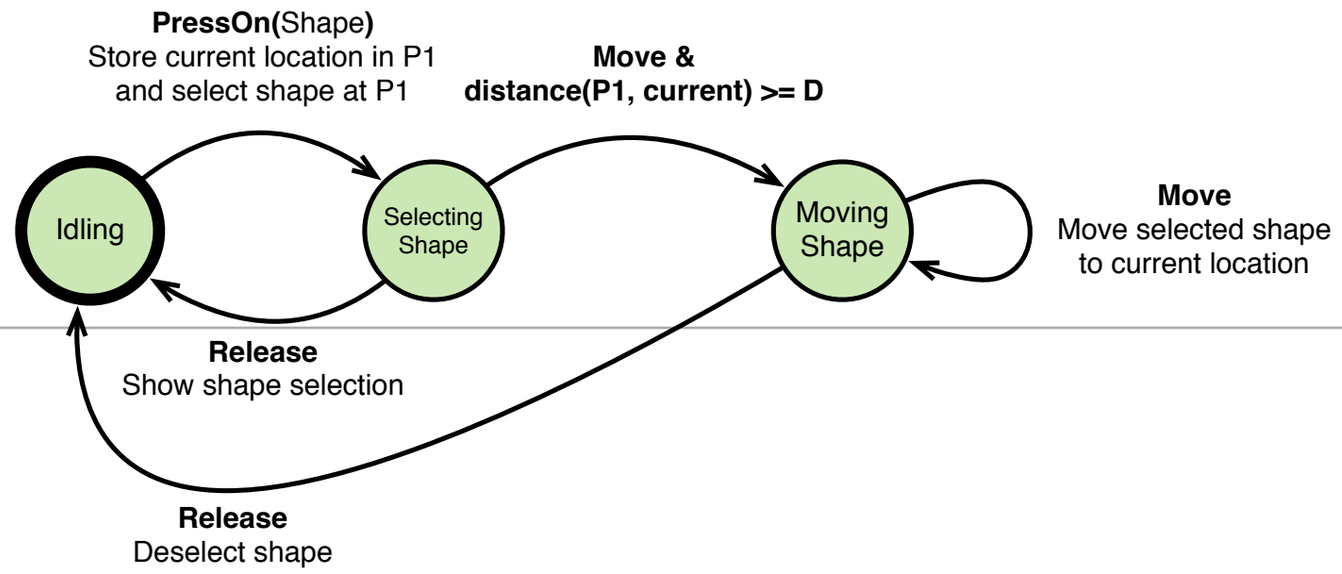


Example

Now let’s implement with SwingStates the other state machine that combines selection and dragging:



Example




```
StateMachine sm = new StateMachine() {
      State idling = new State() {
            Transition press = new PressOnShape (BUTTON1, "=> selecting") {
                  public void action() { /* select shape and store current location to P1 */ }
            };
      };
      State selecting = new State() {
            Transition move = new Move("=> movingShape") {
                  public boolean guard() { /* check if distance > D */ }
            };
            Transition release = new Release(BUTTON1, "=> idling") {
                  public void action() { /* change object appearance */ }
            };
      }
      State movingShape = new State() {
            Transition move = new Move() {
                  public void action() { /* move selected shape */ }
            };
            Transition release = new Release(BUTTON1, "=> idling") {}
      };
};
```

Attaching FSMs

A FSM needs to be **attached** to a specific element where it can listen for events:

```
JStateMachine.attachTo(java.awt.Component)

CStateMachine.attachTo(fr.lri.swingstates.canvas.CElement)

CStateMachine.attachTo(fr.lri.swingstates.canvas.Canvas)
```

A typical error is to forget attaching FSMs.

Attaching FSMs

```
StateMachine sm = new StateMachine() {
      State idling = new State() {
            Transition press = new PressOnShape (BUTTON1, "=> selecting") {
                  public void action() { /* select shape and store current location to P1 */ }
            };
      };
      State selecting = new State() {
            Transition move = new Move("=> movingShape") {
                  public boolean guard() { /* check if distance > D */ }
            };
            Transition release = new Release(BUTTON1, "=> idling") {
                  public void action() { /* change object appearance */ }
            };
      }
      State movingShape = new State() {
            Transition move = new Move() {
                  public void action() { /* move selected shape */ }
            };
            Transition release = new Release(BUTTON1, "=> idling") {}
      };
};

sm.attachTo(shape);
```

Do not forget that!

Transitions

Transitions define two event **properties**:

- the **type** (press, release, move, etc.)
- the optional **target** (element type, group, tag, etc.).

A transition can have no specific target, meaning it occurs solely based on the nature of the event.

- **Key events** and **custom events** are often target-less.
- **Move** events should be target-less.

Transitions

Target-less	Shape	Tag
Click	ClickOnShape	ClickOnTag
Press	PressOnShape	PressOnTag
Release	ReleaseOnShape	ReleaseOnTag
Drag	DragOnShape	DragOnTag
Move	MoveOnShape	MoveOnTag
Enter	EnterOnShape	EnterOnTag
Leave	LeaveOnShape	LeaveOnTag
KeyPress		
KeyRelease		
KeyType		
TimeOut		

Shape events relate to the shape
to which the state machine is attached to.

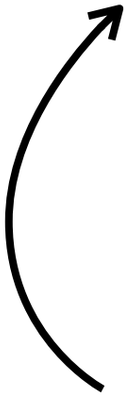
Two StateMachine classes

CStateMachine is dedicated to SwingStates components that inherit from:

```
fr.lri.swingstates.canvas.Canvas
fr.lri.swingstates.canvas.CElement
```

JStateMachine is dedicated to Java Swing components that inherit from:

```
java.awt.Component
```



A JStateMachine replaces standard Swing listeners such as ActionListener.