State: Fun with Finite State Machines

1. Consider an ordinary telephone
2. What you do with it depends on the state of the phone/line
   1. If its ringing or you want to make a call, you can pick it up
   2. Phone must be off the hook to talk/make a call
   3. If you try calling someone, and its busy, you put the handset down
3. Changes in state can be explicit ot in response to event (Observer pattern)
4. State: A pattern in which the object’s behavior is determined by its state. An object transitions from one state to another (something needs to trigger a transition)
5. A formalized construct which manages state and transitions is called a state machine.

Classic Implementation

1. Strange implementation → not really used in the industry in this way
2. LightSwitch example
3. State is owned by LightSwitch but it changes the owner's state.

Handmade State Machine

1. You have pairs of states and triggers that indicate which trigger the state goes from which state to which one.
2. No example code for this one.

Spring statemachine

1. Uses spring library. Example for using third party code.
2. Example code:

package com.activemesa.behavioral.state.spring;

import org.springframework.statemachine.StateMachine;

import org.springframework.statemachine.config.StateMachineBuilder;

import org.springframework.statemachine.state.State;

import org.springframework.statemachine.transition.Transition;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.Collection;

import java.util.EnumSet;

import java.util.List;

import java.util.stream.Collectors;

// plural because SSM defines a class called State already

enum States

{

OFF\_HOOK, // starting

ON\_HOOK, // terminal

CONNECTING,

CONNECTED,

ON\_HOLD

}

enum Events

{

CALL\_DIALED,

HUNG\_UP,

CALL\_CONNECTED,

PLACED\_ON\_HOLD,

TAKEN\_OFF\_HOLD,

LEFT\_MESSAGE,

STOP\_USING\_PHONE

}

class SpringStatemachineDemo

{

public static StateMachine<States, Events> buildMachine()

throws Exception

{

StateMachineBuilder.Builder<States, Events> builder

= StateMachineBuilder.builder();

builder.configureStates()

.withStates()

.initial(States.OFF\_HOOK)

.states(EnumSet.allOf(States.class));

builder.configureTransitions()

.withExternal()

.source(States.OFF\_HOOK)

.event(Events.CALL\_DIALED)

.target(States.CONNECTING)

.and()

.withExternal()

.source(States.OFF\_HOOK)

.event(Events.STOP\_USING\_PHONE)

.target(States.ON\_HOOK)

.and()

.withExternal()

.source(States.CONNECTING)

.event(Events.HUNG\_UP)

.target(States.OFF\_HOOK)

.and()

.withExternal()

.source(States.CONNECTING)

.event(Events.CALL\_CONNECTED)

.target(States.CONNECTED)

.and()

.withExternal()

.source(States.CONNECTED)

.event(Events.LEFT\_MESSAGE)

.target(States.OFF\_HOOK)

.and()

.withExternal()

.source(States.CONNECTED)

.event(Events.HUNG\_UP)

.target(States.OFF\_HOOK)

.and()

.withExternal()

.source(States.CONNECTED)

.event(Events.PLACED\_ON\_HOLD)

.target(States.OFF\_HOOK)

.and()

.withExternal()

.source(States.ON\_HOLD)

.event(Events.TAKEN\_OFF\_HOLD)

.target(States.CONNECTED)

.and()

.withExternal()

.source(States.ON\_HOLD)

.event(Events.HUNG\_UP)

.target(States.OFF\_HOOK);

return builder.build();

}

// requires org.springframework.statemachine

public static void main(String[] args) throws Exception

{

StateMachine<States, Events> machine = buildMachine();

machine.start();

States exitState = States.ON\_HOOK;

BufferedReader console = new BufferedReader(

new InputStreamReader(System.in)

);

while (true)

{

State<States, Events> state = machine.getState();

System.out.println("The phone is currently " + state.getId());

System.out.println("Select a trigger:");

List<Transition<States, Events>> ts = machine.getTransitions()

.stream()

.filter(t -> t.getSource() == state)

.collect(Collectors.toList());

for (int i = 0; i < ts.size(); ++i)

{

System.out.println("" + i + ". " +

ts.get(i).getTrigger().getEvent());

}

boolean parseOK;

int choice = 0;

do

{

try

{

System.out.println("Please enter your choice:");

choice = Integer.parseInt(console.readLine());

parseOK = choice >= 0 && choice < ts.size();

}

catch (Exception e)

{

parseOK = false;

}

} while (!parseOK);

// perform the transition

machine.sendEvent(ts.get(choice).getTrigger().getEvent());

if (machine.getState().getId() == exitState)

break;

}

System.out.println("And we are done!");

}

}



Summary

1. Given sufficient complexity, it pays to formally define possible states and events/triggers
2. Can define
   1. State entry/exit behaviors
   2. Action when a particular event causes a transition
   3. Guard conditions enabling/disabling a transition
   4. Default action when no transitions are found for an event