



# **Representación de maquinas de estados finitos (FSM) en componentes y en el lenguaje de programación Java**





# Agenda

- Introducción
- Descripción del problema
- Trabajo previo
- Solución propuesta





# Introducción





# Introducción

- Trabajo exploratorio
- Ventajas e implicaciones de agregar la noción de estado tanto a un componente como a un lenguaje de programación orientado a objetos





# Descripción del problema





# Descripción del problema

- Existen varias maneras de representar FSM en lenguajes de programación orientados a objetos:
  - Operador switch
  - Redes de Petri
  - Métodos virtuales





# Descripción del problema



- ¿Qué pasaría si se conociera el estado de un componente u objeto de manera externa?
- ¿Qué ventajas e implicaciones tendrían el usar este enfoque?





# Descripción del problema



Dos perspectivas:

- Implementar FSM en un lenguaje de programación actual
- Proponer un mecanismo a un lenguaje de programación para que sea posible implementar FSM de manera mas transparente





# Trabajo previo



# Trabajo previo

- Un FSM  $M$  es una quintupla  $M = (S, I, \delta, S_0, F)$  donde:
  - $S$  es el conjunto finito de estados.
  - $I$  es un alfabeto finito de símbolos de entrada.
  - $\delta$  es la transición de un estado a otro  $S \times I \rightarrow S$ .
  - $S_0$  es el estado inicial.
  - $F$  son los estados de aceptación.





# Trabajo previo

- FSM usadas en hardware ej. PLC
- Modelar problemas teóricos de programación





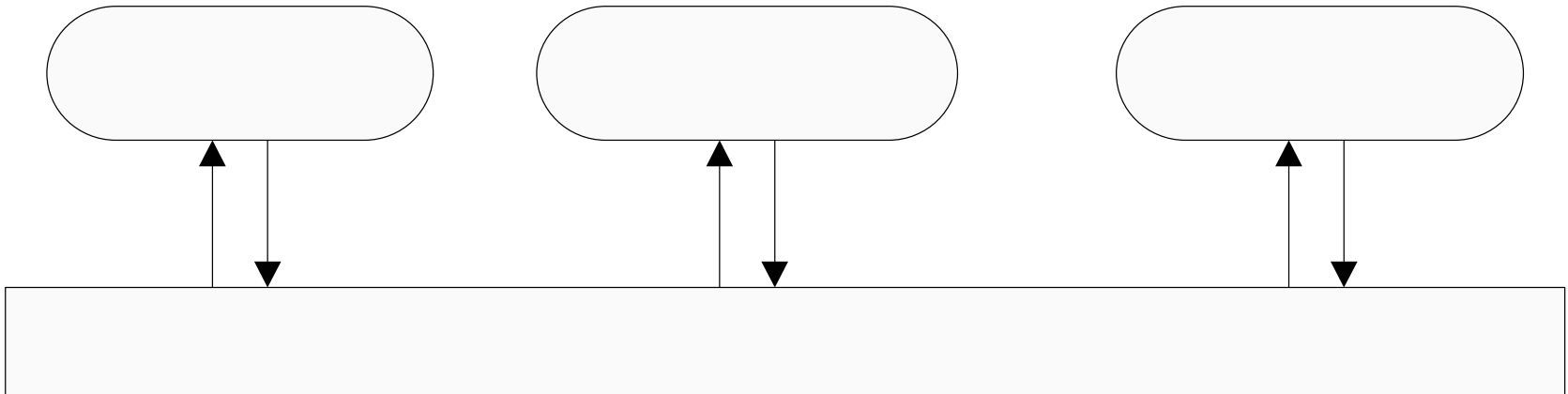
# Trabajo previo

- Tendencia actual con respecto a FSM:
  - Representar estados en lenguajes de programación orientados a objetos, tiene sus inicios en el uso de sistemas basados en eventos, también conocido como sistemas reactivos



# Trabajo previo

- Sistema basado en eventos





# Trabajo previo

- El programar este tipo de sistemas bajo un enfoque orientado a objetos además del uso de eventos y con ayuda de un enfoque de maquina de estado se le conoce como **programación orientada a objetos basada en estados.**





# Solución propuesta





# Soluciones propuestas

- Se proponen dos soluciones:
  - Representar FSM en el actual lenguaje de programación Java
  - Proponer un mecanismo al lenguaje Java el cuál permita de manera transparente representar FSM





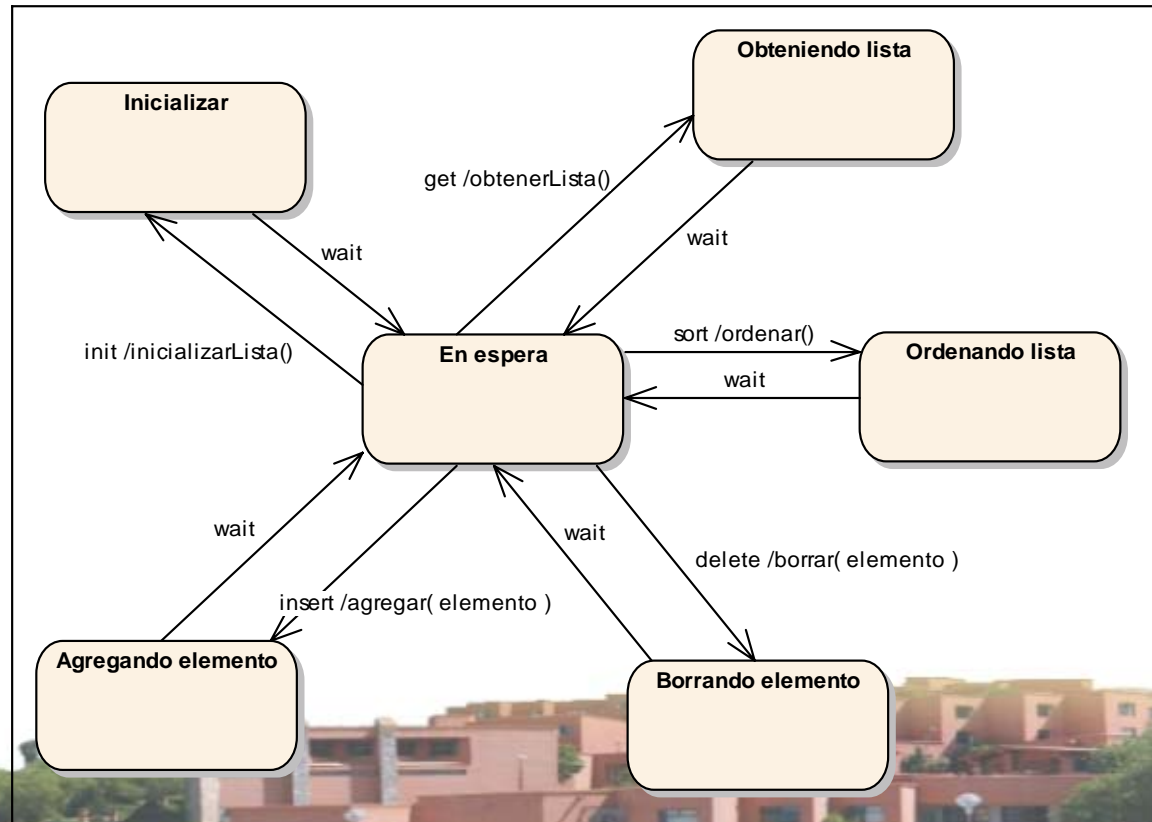
# Propuesta 1

- Utilizar el enfoque de programación orientada a objetos basada en estados en el actual lenguaje de programación Java
- Ejemplo:
  - Implementar un componente que represente una lista, el cual realiza una serie de acciones que son:
    - Inicializar la lista
    - Insertar un elemento en la lista
    - Borrar un elemento de la lista
    - Ordenar los elementos que se encuentren en la lista



# Propuesta 1

- Representación del componente Lista en una FSM





# Propuesta 1

- Diagrama de clases propuesto para representar FSM





# Propuesta 1



## Fragmento de código en lenguaje Java del componente Lista

```
1 public class Lista implements Runnable{
2     Lista listaOrdenada;
3     public void run(){
4         while( true ){
5             switch( eventType ){
6                 case eventInit:  inicializarLista();
7                 break;
8                 case eventSort:  listaOrdenada = ordenar();
9                 break;
10            ...
11        }
12    }
13 }
```





# Propuesta 1

## Fragmento de código de un cliente que esta usando el componente Lista

```
1 public class Cliente{
2     Lista componenteLista = new Lista();
3     if ( componenteLista.getActualState().equals( "inicializar" ) )
4         System.out.println( "Espere un momento la lista se
esta inicializando" );
5     if ( componenteLista.getActualState().equals( "En espera" ) ){
6         EventType eventSort = new EventSort("sort");
7         componenteLista.sendEvent( eventSort );
8     }
9 }
```





# Propuesta 1

- Ventajas:
  - Mayor control sobre las acciones a realizar
  - Conocimiento de los estados del componente en cualquier momento
- Implicaciones:
  - El componente debe de ejecutarse como subproceso del sistema





# Propuesta 2

- Proponer un mecanismo a un lenguaje de programación para que sea posible implementar FSM de manera mas transparente



# Propuesta 2

- Agregar la clase FSM a la librería base de Java

<b>«Runnable»</b> <b><i>FSM</i></b>
+ getActualState() : String + getStates() : String + getEventNames() : List + getTransitions() : List + getActionNames() : List + run() : void





# Propuesta 2

```
01 definitionFSM {
02     states = { Inicializar, En espera, Obteniendo lista, Ordenando lista,
        Borrando elemento, Agregando elemento }
04     actions {
05         private List obtenerLista();
06         private List ordenar();
07         private void borrar( Object elemto );
08         private void agregar( Object elemento );
09         private void inicializar();
10     }
11     events = { wait, get, sort, delete, insert, init }
12     transitions = { Inicializar>>wait>>En espera,
        En espera>>get>>Obteniendo lista,
        En espera>>sort>>Ordenando lista,
        En espera>>delete>>Borrando elemento,
        En espera>>insert>>Agregando elemento,
        En espera>>init>>inicializar,
        Obteniendo lista>>wait>>En espera,
        Ordenando lista>>wait>>En espera,
        Borrando elemento>>wait>>En espera,
        Agregando elemento>>wait>>En espera }
22 }
```



# Propuesta 2

*continuación...*

```
23 public class Lista is FSM{
24     private List ordenar(){
25         ...
26         return listaOrdenada;
27         ...
28     }
29     switch( eventType ){
30         case eventInit:  inicializarLista();
31         break;
32         case eventSort:  listaOrdenada = ordenar();
33         break;
34         ...
35     }
36 }
```





# Propuesta 2

## Fragmento de código del un cliente que usa el componente Lista

```
1 public class Cliente{
2     Lista componenteLista = new Lista();
3     if ( componenteLista.getActualState().equals( "inicializar" ) )
4         System.out.println( "Espere un momento la lista se esta
inicializando" );
5     if ( componenteLista.getActualState().equals( "En espera" ) ){
6         EventType eventSort = new EventType("sort");
7         componenteLista.sendEvent( eventSort );
8     }
9 }
```





# Propuesta 2

- **Ventajas:**
  - Fácil de definir e implementar un FSM
  - Opción a implementar o no FSM
  
- **Implicaciones:**
  - El componente debe de ejecutarse como subproceso del sistema





# Referencias

- D. Shopyrin, “Object oriented implementation of state based logic”,  
<http://www.codeproject.com/cpp/statebased.asp>
- A.A. Shalyto, “Technology of automata based programming”, 2003  
<http://www.codeproject.com/gen/design/abp.asp>
- A.A. Shalyto, “Logic Control and Reactive Systems: Algorithmization and Programming”, Automation and Remote Control, Vol. 62, No. 1, 2001, pp. 1-29
- I. Sommerville, “Software Engineering”, Addison Wesley, 2001, pp. 227-229
- A. Newman, S. M. Shatz, X. Xie, “An approach to object system modeling by state-based object Petri nets”
- D. Harel, “StateCharts: a visual formalism for complex systems”, science of computer programming, 1987, #8
- O. Lehrmann Madsen, “Towards integration of state machines and object-oriented languages”, 1999,  
<http://www.cit.dk/COT/reports/reports/Case2/26/cot-2-26.pdf>
- T. Lindholm, Y. Frank, “The Java Virtual Machine Specification”, Addison-Wesley, 1999





Gracias !!!

