

# Orientación a Objetos

## Estados

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

25 de noviembre de 2020





# Temas

## 1 Estados

- Definiciones
- Estados en un programa orientado a objetos
- Ejemplo

# Estado

## Definición

- El concepto de *estado* se utiliza para especificar cómo se encuentra un sistema en un tiempo dado.
- Esta descripción se da en terminos de *variables*, cada una de las cuales toma su valor de un *dominio*<sup>a</sup> que le es propio.
- Al transcurrir el tiempo los valores de estas variables se pueden modificar, para describir esta dinámica se utilizan:
  - *Máquinas de estados finitas* o *Autómatas de estados finitos*
  - *Sistemas de transiciones*<sup>b</sup>

<sup>a</sup>El dominio es el conjunto de valores posibles para una variable.

<sup>b</sup>*Transition system* 2017.

# Temas

## 1 Estados

- Definiciones
- Estados en un programa orientado a objetos
- Ejemplo

# Estados en un programa orientado a objetos

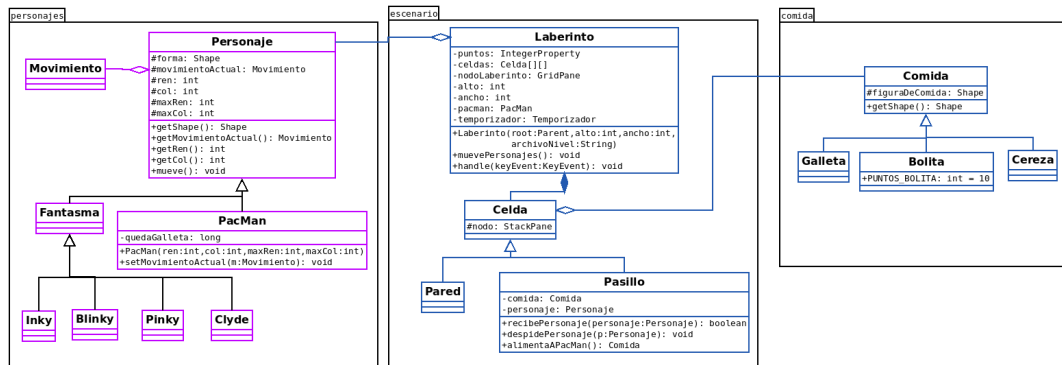
- En un programa, el estado es el conjunto de valores asignados, en un tiempo dado, a todas las **variables vivas** del programa: atributos de objetos existentes y variables locales.
- Para facilitar el análisis del comportamiento del programa, consideraremos los estados de objetos individuales y las transiciones entre ellos.
- Cuando los objetos interactúan entre sí, necesitaremos considerar al estado como dado por la unión de los atributos de los objetos en comunicación.

# Temas

- 1 Estados
  - Definiciones
  - Estados en un programa orientado a objetos
  - Ejemplo

# PacMan

El siguiente diagrama de clases nos puede ayudar a visualizar las variables que definen el estado del programa para jugar PacMan.





# Autómatas

- 1 Estados
- 2 Autómatas**
- 3 Ejemplo: Autómatas Celulares
- 4 Orientación a objetos en Java

# Temas

## 2 Autómatas

- Autómatas finitos
- Ejemplo: Pacman
- Máquina de estados

# Autómatas finitos o máquinas de estados

“Un autómata finito contiene un conjunto de estados  $Q$  y un conjunto de transiciones de estado a estado que ocurren cuando al autómata se le alimentan símbolos de un alfabeto  $\Sigma$ .” Viso G. 2008

## Definición

Un *autómata finito*  $M$  es un quintuplo  $M = (Q, \Sigma, \delta, q_0, F)$  donde:

- $\Sigma$ : Alfabeto finito de entrada
- $Q$ : Conjunto finito de estados
- $\delta$ : Función de transición que asigna un estado siguiente dados un estado y un símbolo.

$$\delta : Q \times \Sigma \rightarrow Q$$

- $q_0$ : Estado inicial  $q_0 \in Q$
- $F$ : Estados finales, que aceptan la cadena,  $F \subseteq Q$

# Temas

## 2 Autómatas

- Autómatas finitos
- Ejemplo: Pacman
- Máquina de estados

# Pacman

Podemos utilizar un autómatata finito para expresar los estados del personaje PacMan.

- $\Sigma$ : Las acciones y eventos que pueden ocurrir: las teclas que presiona el usuario y si PacMan se come una galleta, en cuyo caso podrá comer fantasmas durante un intervalo finito de tiempo.

$$\Sigma = \{\rightarrow, \leftarrow, \uparrow, \downarrow, \text{comeGalleta}, \text{finTiempo}\}$$

- $Q$ : Estados de movimiento de PacMan. Mientras el usuario no presione una dirección, PacMan se sigue desplazando en la última dirección que se le dejó. Entonces, cada estado de PacMan está dado por las combinaciones de valores posibles para sus dos variables:

$$\text{estadoDeMovimiento} \in \{\text{Izquierda}, \text{Derecha}, \text{Abajo}, \text{Arriba}\}$$

$$\text{poder} \in \{\text{Normal}, \text{con Galleta}\}$$

- $\delta$ : Función que indica a qué dirección de movimiento cambia y si transita entre comer o no fantasmas.
- $q_0$ :  $q_0 = \text{Derecha Normal}$
- $F$ : En este caso PacMan solo podría tomar cualquier estado como final. Pero, si consideráramos al estado del laberinto completo, los estados finales serían todos aquellos donde ya no queden bolitas por comer.





# Temas

## 2 Autómatas

- Autómatas finitos
- Ejemplo: Pacman
- Máquina de estados

# Máquina de estados

- Es más común escuchar este término en el campo de la ingeniería.
- Es básicamente un autómata finito conocido como *Máquina de Mealy* y sólo agrega el hecho de que puede devolver un símbolo por cada transición que realiza.

# Máquina de estados

## Definición (Máquina de estados)

Una *máquina de estados* es un concepto utilizado para describir el comportamiento posible de un sistema discreto, que evoluciona en el tiempo. Está constituida por:

**Estados** Un conjunto  $S = \{s_1, s_2, \dots\}$  de estados, que representan la configuración actual del sistema. El sistema sólo se puede encontrar en un estado a la vez.

**Entradas** Un conjunto  $I = \{i_1, i_2, \dots\}$  de posibles valores con lo cuales se alimentará a la máquina en cada paso temporal.

**Salidas** Se tiene un conjunto  $O = \{o_1, o_2, \dots\}$  de símbolos, que puede emitir la máquina en cada paso temporal. Qué símbolo emita puede depender del estado en el que se encuentre actualmete y del valor de entrada que acabe de recibir.

## Máquina de estados

**Función de transición** Una función  $\gamma$  que describe bajo qué condiciones puede el sistema transitar de un estado hacia otro.

**Función de salida** Una función  $\beta$  que indica de qué depende el símbolo que emitirá la máquina en cada paso temporal.

Fuentes:

- 1 [https://en.wikipedia.org/wiki/Transition%5C\\_system](https://en.wikipedia.org/wiki/Transition%5C_system)
- 2 <https://www.techopedia.com/definition/16447/state-machine>
- 3 [https://github.com/pluginaweek/state\\_machine](https://github.com/pluginaweek/state_machine)

## Ejemplo: Autómatas Celulares

- 1 Estados
- 2 Autómatas
- 3 Ejemplo: Autómatas Celulares**
- 4 Orientación a objetos en Java

# Temas

## 3 Ejemplo: Autómatas Celulares

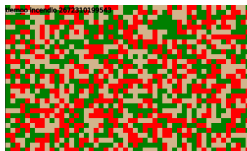
- Definiciones
- Simulación

# Autómata celular

## Definición (Autómata celular)

Un *autómata celular* es un sistema dinámico discreto en el cual el tiempo transcurre por pasos y el espacio se encuentra dividido en *celdas* regulares de igual tamaño y forma acomodadas en una *mallá*.

- Cada celda puede tomar un valor de un dominio  $\mathbb{D}$ , común a todas las celdas del autómata.
- El *estado* del autómata al tiempo  $t$  es el conjunto de valores  $x^t$  de todas sus celdas.



## Definición

- El estado del autómata se actualiza del tiempo  $t$  al tiempo  $t + 1$  calculando los nuevos valores para todas sus celdas.

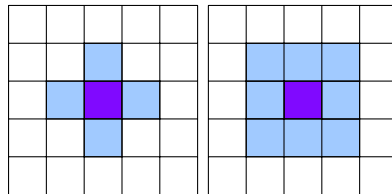
$$x^{t+1} = f(x^t)$$

- El valor de cada celda se calcula con reglas sencillas locales, donde el nuevo valor  $x_{ij}^{t+1}$  depende de los valores de sus celdas vecinas, de modo que:

$$x_{ij}^{t+1} = f(x_{ij}^t)$$



# Vecindad



**Figura:** Izquierda: vecindad de Von Neumann. Derecha: vecindad de Moore. Los dos tipos de vecindades más usuales en autómatas celulares bidimensionales. El estado de la celda central es determinado, en el siguiente paso temporal, por su valor actual y el de las celdas coloreadas a su alrededor.

# Temas

## 3 Ejemplo: Autómatas Celulares

- Definiciones
- Simulación

# Comportamiento emergente

- El resultado que se obtiene al realizar simulaciones con estas reglas sencillas es que el sistema completo manifiesta propiedades que sólo se explican como consecuencia de las interacciones entre sus partes.
- A estos comportamientos complejos, donde se observa que *el todo es más que la suma de las partes* se le conoce como *comportamientos emergentes*.
- Los autómatas son un tipo de ecuaciones en diferencias y son utilizados frecuentemente para modelar fenómenos naturales.

# Ejemplo: simulaciones

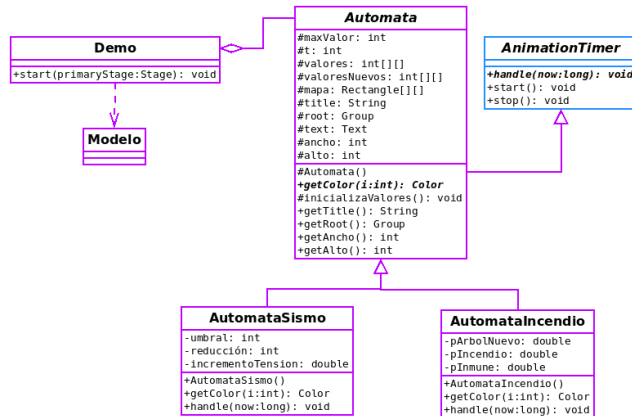


Figura: El arreglo con los estados de las celdas contiene el estado del autómatas.

# Orientación a objetos en Java

- 1 Estados
- 2 Autómatas
- 3 Ejemplo: Autómatas Celulares
- 4 Orientación a objetos en Java

# Temas

## 4 Orientación a objetos en Java

- Clases
- Atributos
- Métodos

# Declaración de clases

## Sintaxis (Firma)

```
<declaración de clase> ::= <acceso> class <identificador> {  
    <declaraciones>  
    <método main>  
}  
  
<acceso> ::= public | private | protected |  $\emptyset$  (i.e. package)  
<identificador> ::= (<letra>|_)(<letra>|<dígito>|_)*
```

Por convención, los nombres de clases inician con mayúscula.

```
1  public class Persona{  
2  
3  }
```

# Temas

## 4 Orientación a objetos en Java

- Clases
- Atributos
- Métodos



# Atributos

## Sintaxis (Atributos)

```
<declaración de atributo> ::= <acceso> <modificador> <tipo>
    <identificador>, ..., <identificador>;
<modificador> ::= final | static | ∅
<tipo> ::= <tipo primitivo> | <identificador de clase>
```

```
1  public class Persona{
2      public static final String SLOGAN = "¡Que viva la gente!";
3      private String _nombre;
4      private short _edad;
5      private Color _colorOjos;
6      private Color _colorCabello;
7      private Sexo _sexo;
8  }
```

# Temas

## 4 Orientación a objetos en Java

- Clases
- Atributos
- Métodos

# Métodos

- Funcionalmente los métodos de una clase pueden ser clasificados en:
  - Constructores:** Crean a los objetos e inicializan sus atributos.
  - De acceso y modificación:** Devuelven el valor de un atributo o permiten asignarle un valor, según las restricciones impuestas por la clase, respectivamente.
  - De actualización y manipulación:** Cambian el estado del objeto<sup>[1]</sup>.
  - De implementación:** Ofrecen los servicios especiales para los que fue diseñado el objeto.
  - Auxiliares:** Permiten al objeto organizar mejor sus tareas y no proveen servicios al exterior, es decir, se declaran como privados (o protegidos).
- Método **main**. Permite ejecutar a la clase como un programa, es estático, por lo que no aplica a ningún objeto en particular.

---

<sup>[1]</sup> Los valores de sus atributos

# Constructor

- El constructor permite crear a los objetos que las clases describen.
- Java asigna valores por defecto a los atributos del objeto.
  - Para los atributos de tipo numérico, es **0**.
  - Para los booleanos es **false**.
  - Para objetos el valor es **null**, es decir “no hay objeto”.

Aunque las versiones de Java más recientes marcan error si el programador no asigna un valor explícitamente.

- Estos valores se pueden cambiar en el constructor.
- Si el programador no agrega explícitamente ningún constructor, Java pone un **constructor por defecto**, que no recibe parámetros y asigna los valores por defecto a todas las variables.
- En cuanto el programador agrega un constructor, desaparece el constructor por defecto.

# Sintaxis

¡El constructor siempre se llama igual que la clase y no tiene tipo de regreso!

## Sintaxis (Constructor)

```
<constructor> ::= <acceso> <identificador de Clase> (<Parámetros>){  
    <implementación>  
}  
<Parámetros> ::= <parámetro> (,<Parámetros>)* | ∅  
<parámetro> ::= <tipo> <identificador>
```

# Ejemplo

```
1  public class Persona{
2      ...
3      //Firma: Persona()
4      public Persona(){
5          ...
6      }
7      //Firma: Persona(String, short, Color, Color, Sexo)
8      public Persona(String nombre, short edad,
9                      Color colorOjos, Color colorCabello,
10                     Sexo sexo){
11          ...
12      }
13 }
```

# Métodos en general

## Sintaxis (Método)

```
<método> ::= <acceso> (<tipo>|void) <identificador>(<Parámetros>){  
    <implementación>  
}
```

- Los nombres de los métodos inician con minúsculas.

# Métodos

```
1  public class Persona{
2      ...
3      public String getNombre(){ //Acceso
4          ...
5      }
6      private void daPaso(Dirección dirección){//Auxiliar
7          ...
8      }
9      public void camina(){ //Manipulación
10         ...
11     }
12     public Reporte hazReporte(){ //Implementación
13         ..
14     }
15 }
```



# Advertencia

**OJO:** no repetir firmas de métodos dentro de una misma clase.

```
1  public class Matemático{
2      ...
3      public double calculaÁngulo(double cateto,
4                                  double hipotenusa){
5          ...
6      } //Firma: calculaÁngulo(double, double)
7      public double calculaÁngulo(double cOpuesto,
8                                  double cAdyacente){
9          ...
10     } //Firma: calculaÁngulo(double, double)
11 }
```

# Método main

```
1  public class Argumentos{
2      public static void main(String[] args){
3          for(int i = 0; i < args.length; i++){
4              System.out.println("Argumento_" + i + ":_ "
5                                  + args[i]);
6          }
7      }
8  }
```

Al llamar al programa en consola:

```
user@compu:~/$ java Argumentos
user@compu:~/$ java Argumentos algo otro tercero
Argumento 1: algo
Argumento 2: otro
Argumento 3: tercero
```

# Construcción de objetos

## Sintaxis (Constructor)

`<construcción de objeto> ::= new <invocación de método constructor>`

```
1 Persona m = new Persona("Kike", 15, colorOjos, colorCabello,
2                      Sexo.MASCULINO);
3 Persona f = new Persona("Lilí", 15, colorOjos, colorCabello,
4                      Sexo.FEMENINO);
```

- Los objetos se crean en el montículo.
- Lo que devuelven `new` y la función constructora es la dirección del objeto que fue creado.

# Accediendo a los miembros de un objeto

## Sintaxis (Derreferenciar)

`<Referencia a miembro> ::= <identificador de objeto>.<id de miembro>`

Solamente es posible acceder a los atributos para los que se tiene permiso.

```
1 // Dentro de la clase Persona
2 Persona otra = new Persona();
3 otra._nombre;
```

# this

- Todos los métodos de una clase reciben como parámetro al objeto al que se envía el mensaje. En Java se le llama `this`.

```
1 public class Persona{
2     /** Constructor. */
3     public Persona(String nombre, short edad,
4                     Color colorOjos, Color colorCabello,
5                     Sexo sexo){
6         this._nombre = nombre;
7         this._edad = edad;
8         this._colorOjos = colorOjos;
9         this._colorCabello = colorCabello;
10        this._sexo = sexo;
11    }
12 }
```



# Invocación de método

## Sintaxis (Invocar método)

```
<invocación de método> ::= this.<nombre del método> (Argumentos)
    | <nombre del método> (Argumentos)
<Argumentos> ::= <argumento>,<Argumentos> | ∅
<argumento> ::= <constante> | <variable>
```

```
1  persona.camina();
2  this.daPaso();
```

# Bibliografía I

-  *Transition system* (jul. de 2017). Wikimedia Foundation, Inc. URL: [https://en.wikipedia.org/wiki/Transition%5C\\_system](https://en.wikipedia.org/wiki/Transition%5C_system).
-  Viso G., Elisa (2008). *Introducción a la teoría de la computación*. Temas de computación. Las prensas de ciencias. 304 págs. ISBN: 978-970-32-54-15-6.

# Licencia

Creative Commons  
Atribución-No Comercial-Compartir Igual

