

# Programación Orientada a Objetos

## Relaciones de Objetos y Clases

Jonatan Gómez Perdomo, Ph.D.

[jgomezpe@unal.edu.co](mailto:jgomezpe@unal.edu.co)

Arles Rodríguez, Ph.D.

[aerodriguezp@unal.edu.co](mailto:aerodriguezp@unal.edu.co)

Camilo Cubides, Ph.D.(c)

[eccubidesg@unal.edu.co](mailto:eccubidesg@unal.edu.co)

Grupo de investigación en vida artificial – Research Group on Artificial Life – (Alife)

Departamento de Ingeniería de Sistemas e Industrial

Facultad de Ingeniería

Universidad Nacional de Colombia



# Agenda

- 1 Relaciones
- 2 Tipos de relaciones
- 3 Diagramas UML
- 4 Problemas varios



# Relación

## Definición RAE

Conexión, correspondencia de algo con otra cosa.

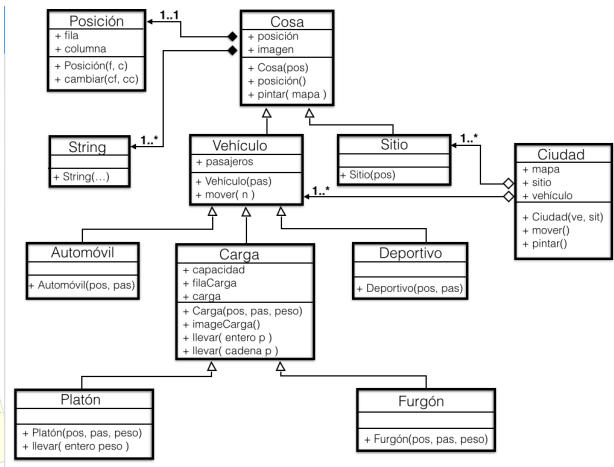
<https://dle.rae.es/relación>



# La ciudad (I)

## Versión super-simplificada

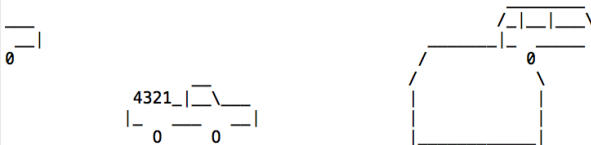
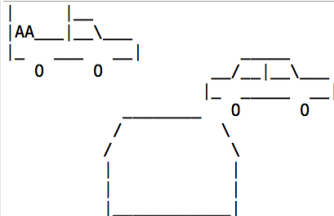
Código disponible en el subproyecto ciudad de  
<https://github.com/jgomezpe/carros>



# La ciudad (II)

Al ejecutarlo trate que la consola de su ambiente quede del alto mostrado en la imagen, verá una animación!!

<terminated> Ciudad [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_141.jdk/Contents/Home/bin/java (Oct 27, 2020 2:39:45 PM -)



# Relaciones

## Programación Orientada por Objetos

Se pueden definir diferentes tipos de relaciones entre clases, cada una con un propósito en particular y con propiedades específicas.  
Dichas relaciones indican cómo se comunican los objetos entre sí.



# Agenda

- 1 Relaciones
- 2 Tipos de relaciones**
- 3 Diagramas UML
- 4 Problemas varios



# Tipos de relaciones

Existen diversos tipos de relaciones que se pueden definir en programación orientada por objetos, entre las más conocidas se encuentran:

- Asociación.
- Inclusión
- Generalización/Especialización.





# La ciudad (III) - Clase Ciudad

```
public class Ciudad {  
    protected Vehiculo[] vehiculo;  
    protected Sitio[] sitio;  
    protected char[][] mapa;  
    public Ciudad( Vehiculo[] vehiculo,  
                  Sitio[] sitio )  
    { ... }  
    public void mover() {  
        for( int i=0; i<vehiculo.length; i++ ) {  
            int f = (int)(Math.random()*3) - 1;  
            vehiculo[i].mover(f, 1);  
        }  
    }  
}
```



# La ciudad (III) - Clase Vehículo

```
public class Vehiculo extends Cosa{
    protected int pasajeros;
    public Vehiculo(Posicion posicion, int pasajeros) {
        super( posicion );
        this.pasajeros = pasajeros;
    }
    public void mover( int cambio_fila,
                      int cambio_columna ) {
        this.posicion.cambiar(cambio_fila,
                              cambio_columna);
    }
}
```



# Asociación

Describe cuando un objeto (instancia) de una clase puede comunicarse con un objeto de otra clase (recordemos que comunicarse es acceder tanto a atributos como a métodos de otro objeto).

Esta relación se representa con una flecha de apuntando de la clase del objeto que accede a la clase accedida. Si se acceden mutuamente se elimina la flecha.

¿Qué relaciones de asociación en los métodos mover de las clases Ciudad y Vehiculo existen?



# Dependencia

La dependencia de objetos declara que un objeto *depende* o *necesita* de otro objeto para su correcto funcionamiento.

Se conoce esta relación como la más *débil*, puesto que uno **usa** al otro sin necesidad de conocer su implementación.

## Ejemplo

Se tiene una clase `Ecuacion` que usa la clase `Math` para resolver ciertas operaciones matemáticas. A `Ecuacion` no le interesa cómo `Math` resuelve dichas operaciones, sólo espera el resultado para continuar con su funcionamiento.



# Inclusión

Describe cuando un objeto (instancia) es un atributo de otro objeto (Si, los objetos pueden ser atributos, genial!).

Existen dos tipos de inclusión:

- Agregación.
- Composición

Esta relación se representa con una línea desde la clase del objeto incluido hasta la clase que incluye, línea que termina en un rombo (◊) relleno o no dependiendo del tipo de inclusión. Encima de la línea se indica la cantidad de objetos incluidos.

¿Qué relaciones de inclusión existen en el proyecto la ciudad?



# Agregación

La agregación de objetos ocurre cuando un objeto es parte de otro objeto sin ser completamente dependiente de su existencia.

Esta relación de inclusión se representa con una línea desde la clase del objeto agregado hasta la clase que agrega, línea que termina en un rombo no relleno (◊).

## Ejemplo

Un objeto de tipo `Cliente` puede tener un objeto de tipo `TarjetaCredito`. También puede existir el caso en que un `Cliente` no tenga una `TarjetaCredito`. Así se define una agregación entre `Cliente` y `TarjetaCredito`.

¿Qué relaciones de agregación existen en el proyecto la ciudad?, ¿porqué?



# Composición

La composición de objetos ocurre cuando un objeto está **compuesto** por otros objetos. Es la agregación disjunta y estricta, donde se requiere la existencia de sus partes para la existencia del objeto.

Esta relación de inclusión se representa con una línea desde la clase del objeto agregado hasta la clase que agrega, línea que termina en un rombo relleno (◆).

## Ejemplo

Se puede pensar en un objeto Computador. Este estará compuesto por sus partes, tales como Procesador, Pantalla, Memoria, etc.

¿Qué relaciones de agregación existen en el proyecto la ciudad?, ¿porqué?



# Objetos recursivos (I)

La recursividad (o relación reflexiva) de clases se refiere al caso en que una clase tenga relaciones consigo misma.

De esta manera, algunos atributos de un objeto de dicha clase pueden ser objetos de la misma clase a la que pertenece el objeto. Dicha recursividad se puede romper en el momento en el que se asigna al atributo del objeto un valor nulo (`null`) o de otra clase.

## Ejemplo

La clase `Directorio` tiene como atributo una lista de objetos de la clase `Archivo` o de la clase `Directorio` (los que conocemos como sub-directorios). Estos últimos, pueden tener a su vez archivos y/o sub-directorios, y así sucesivamente. La recursividad se producirá hasta el último directorio dentro del árbol de directorios.





# Objetos recursivos (II)

Gráficamente, se puede trazar un árbol de directorios, que estarán enlazados por las relaciones entre los objetos.

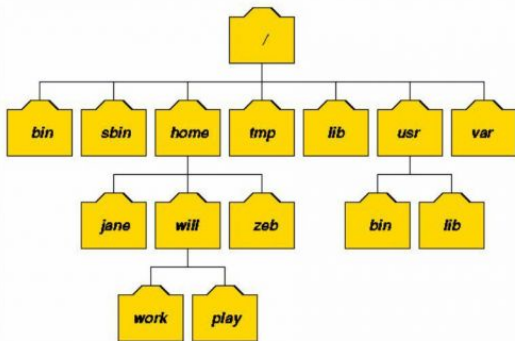


Imagen tomada desde

[https://www.ticarte.com/sites/su/users/7/image/estructura\\_directorios.jpg](https://www.ticarte.com/sites/su/users/7/image/estructura_directorios.jpg)

# Objetos recursivos (III)

```
import java.util.ArrayList;
import java.util.List;

public class Directorio{
    public String nombre;
    List<Directorio> subdirs;
    List<String> archivos;
    Directorio(String nombre) {
        this.nombre = nombre;
        this.subdirs = new ArrayList<Directorio>();
        this.archivos = new ArrayList<String>();
    }
}
```

...



# Objetos recursivos (IV)

...

```
public String espacios(int nivel ){
    String s = "";
    for( int i=0 ; i<nivel; i++ ){ s += ' '; }
    return s;
}
```

```
String toString(int nivel) {
    String s = espacios(nivel)+nombre+'\n';
    for( Directorio d:subdirs )
        s += d.toString(nivel+1) + '\n';
    for( String a:archivos )
        s += espacios(nivel+1) + a + '\n';
    return s;
}
```

...



# Objetos recursivos (V)

...

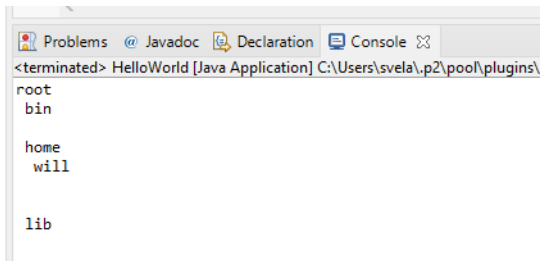
```
public String toString(){ return this.toString(0); }  
public static void main(String[] args) {  
    Directorio root = new Directorio("root");  
    // Primer nivel  
    root.subdirs.add(new Directorio("bin"));  
    root.subdirs.add(new Directorio("home"));  
    root.subdirs.add(new Directorio("lib"));  
    // Segundo nivel  
    root.subdirs.get(0)  
        .subdirs.add(new Directorio("will"));  
    System.out.println(root);  
}
```

}



# Objetos recursivos (VI)

El resultado del código anterior es el siguiente.



```
<terminated> HelloWorld [Java Application] C:\Users\svela\.p2\pool\plugins\  
root  
  bin  
  
  home  
    will  
  
  lib
```



# Generalización/Especialización

La generalización de clases se refiere a la relación de una superclase y sus subclases. Se le conoce también como **Herencia**.

Objetos de distintas clases pueden tener los mismos atributos y exhibir comportamientos similares.

## Ejemplo

La clase Animal tiene relación de generalización con las clases Mamifero, Anfibios, Reptiles, etc.



# Agenda

- 1 Relaciones
- 2 Tipos de relaciones
- 3 Diagramas UML**
- 4 Problemas varios



# Diagramas UML

Los diagramas UML (Unified Modeling Language) son representaciones gráficas que definen un estándar para la interpretación visual de objetos, estados y procesos dentro de un sistema.

Se trata de una serie de normas y estándares que se pueden (algunos dicen deben) seguir a la hora de modelar la estructura de componentes en un diagrama.

Dichos diagramas indican qué debe hacer un sistema, pero no cómo hacerlo. Así, se define un modelo que está libre a la implementación.

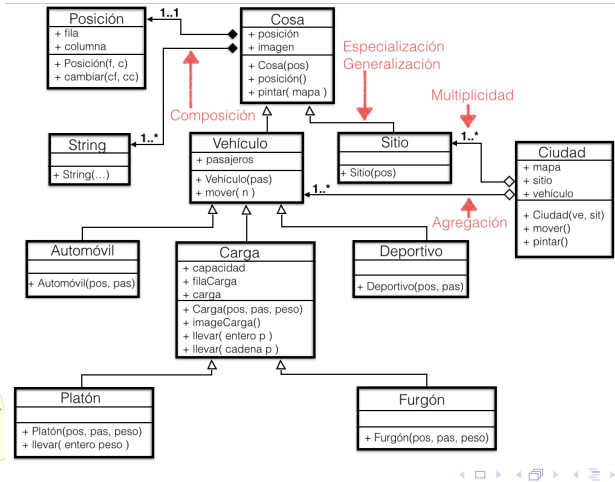




# La ciudad (IV)

## Relaciones

Modelo UML del código disponible en el subproyecto ciudad de <https://github.com/jgomezpe/carros>



# Agenda

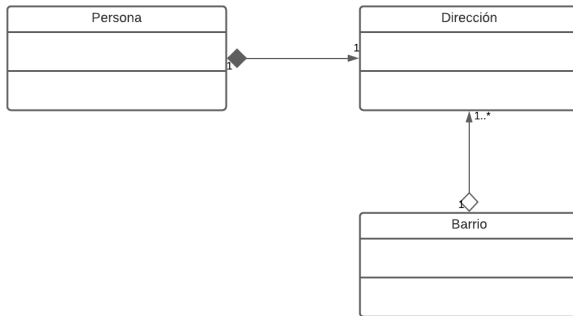
- 1 Relaciones
- 2 Tipos de relaciones
- 3 Diagramas UML
- 4 Problemas varios**



# Problemas

## Problema I

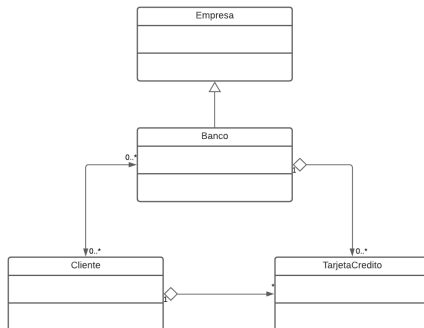
Implementar el siguiente diagrama de clases. Los atributos y métodos se dejan abiertos a la implementación. Nótese que la relación de Persona y Dirección es una **relación fuerte**, mientras que la relación de Dirección y Barrio es una **relación débil**.



# Problemas

## Problema II

Implementar el siguiente diagrama de clases. Los atributos y métodos se dejan abiertos a la implementación. Tenga en cuenta las relaciones y las multiplicidades entre las clases.



# Problemas

## Problema III

LinkedIn es una red social orientada al uso empresarial. Es reconocida por su sistema de *contactos*.

- Puedes agregar contactos de **primer grado**, que serán las personas con las que estés conectado directamente.
- Los contactos de **segundo grado** serán las personas que estén conectadas con tus contactos de primer grado.
- Los contactos de **tercer grado** serán las personas que estén conectadas con tus contactos de segundo grado.



# Problemas

## Problema III (Continuación)

Desarrollar un diagrama de clases recursivo, donde una persona pueda tener una lista de contactos, quienes a su vez serán personas que tienen contactos, etc.

Realizar la implementación del diagrama de clases en Java (véase el ejemplo de los directorios).



*Imagen tomada desde <https://cartasdepresentacion.net/wp-content/uploads/2020/02/reglas-red-contactos-1024x745.jpg>*

