

PROGRAMACIÓN ORIENTADA A OBJETOS

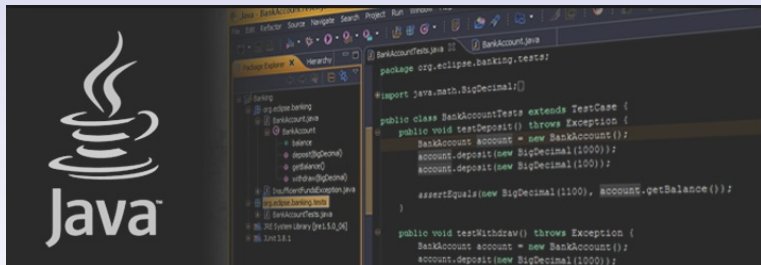
Conceptos

Jazna Meza Hidalgo
Ingeniero Civil en Informática
SCJP 5.0
Magister en Ciencias de la Computación

September 28, 2020

mailto: ymeza at ubiobio.cl

TEMA - CONCEPTOS DE ORIENTACIÓN A OBJETOS



DEFINICIÓN

La Programación Orientada a Objetos (POO) es un modelo de programación que utiliza objetos, unidos mediante mensajes, para la solución de problemas.

- El elemento principal es el objeto
- Los datos y los procesos forman una entidad única

PROBLEMAS

ENUNCIADO # 1

Se desea manejar información de los gatos. Los gatos poseen nombre y edad. Los gatos pueden dormir y comer.

ENUNCIADO # 2

Se desea manejar información de los automóviles. Los datos del automóvil son: color, marca y número de puertas. Un automóvil se puede encender.

PROBLEMAS - ANALIZADOS

ENUNCIADO # 1

Se desea manejar información de los **gatos**. Los gatos poseen *nombre* y *edad*. Los gatos pueden *dormir* y *comer*.

ENUNCIADO # 2

Se desea manejar información de los **automóviles**. Los datos del automóvil son: *color*, *marca* y *número de puertas*. Un automóvil se puede *encender*.

CONCEPTOS

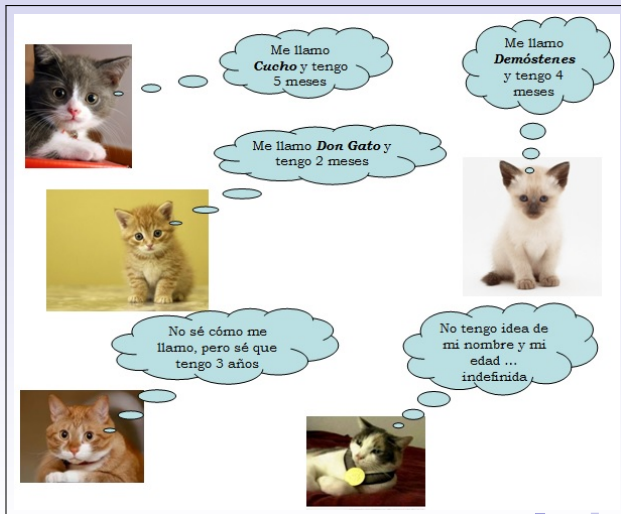
NUEVOS CONCEPTOS



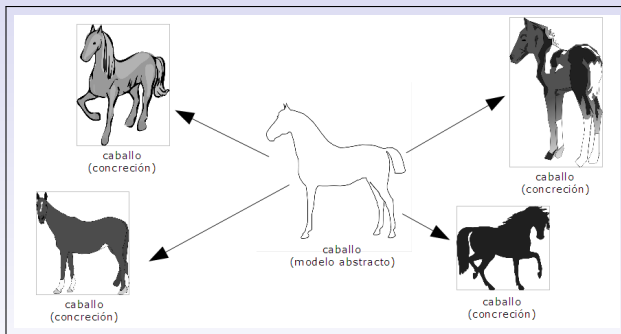
TÉRMINOS A UTILIZAR

- Clase
- Objeto
- Atributos
- Métodos

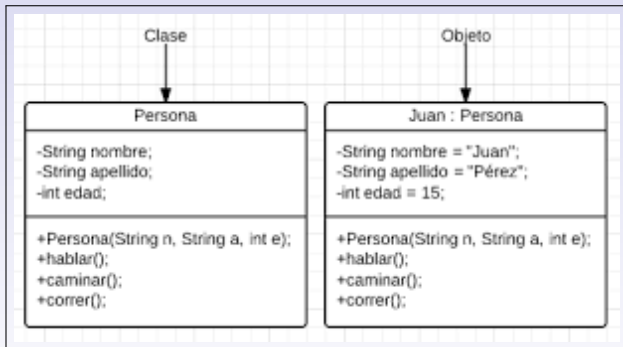
CONCEPTO DE OBJETO



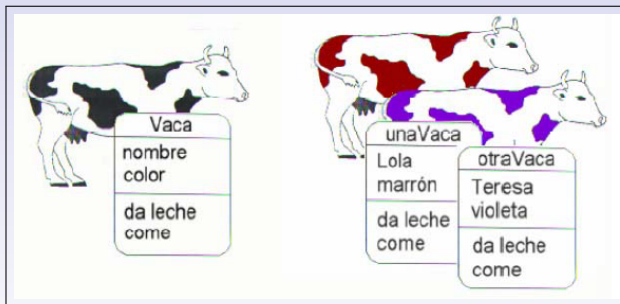
CONCEPTO DE OBJETO



CONCEPTO DE OBJETO



CONCEPTO DE OBJETO



RELACIÓN ENTRE CLASE Y OBJETO



RELACIÓN ENTRE LOS CONCEPTOS



CONCEPTO DE DATO

QUÉ VEMOS

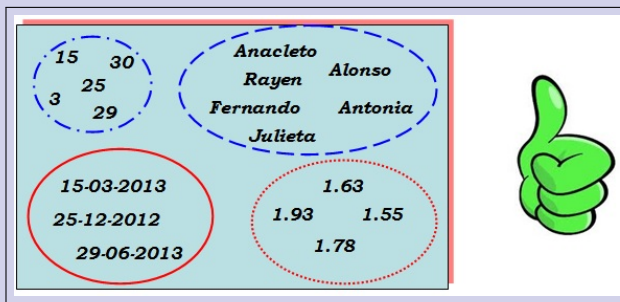
15	30	Anacleto	
	29		Alonso
3	25	Rayen	
		Fernando	
			1.55
Julieta	Antonia	1.63	
15-03-2013			1.93
	25-12-2012		
29-06-2013		1.78	

¿Datos?



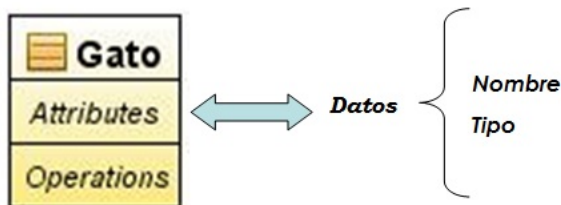
CONCEPTO DE DATO

TIPOS DE DATOS



CONCEPTO DE DATO

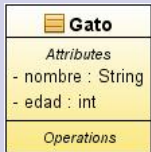
DATO Y TIPO DE DATO



¿Con qué
tipos de
datos vamos
a trabajar?

CONCEPTO DE ATRIBUTO

EJEMPLO 1



EJEMPLO 2



ESTAMOS OK



CONCEPTO DE MÉTODO

DEFINICIÓN

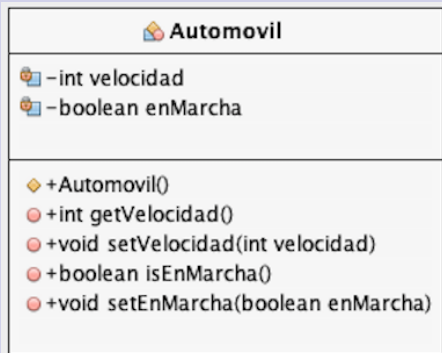
Definen la funcionalidad asociada a los objetos, las acciones que ellos pueden realizar.

TIPOS DE MÉTODOS

- Métodos que modifican atributos, cambiar estado del objeto.
- Retornan valores dependiendo del estado.

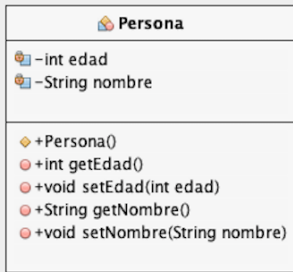
VAMOS A IDENTIFICAR

EJEMPLO



PRIMERA IMPLEMENTACIÓN

DIAGRAMA

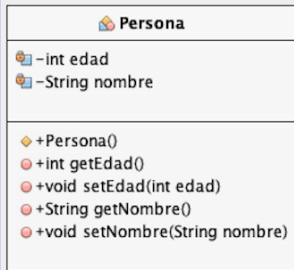


CÓDIGO

```
1 public class Persona
2 {
3     /* Definición de atributos */
4     private int edad;
5     private String nombre;
6 }
```

PRIMERA IMPLEMENTACIÓN

DIAGRAMA

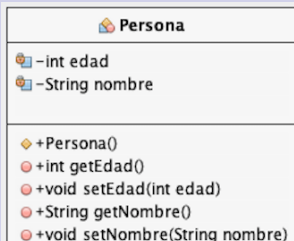


CÓDIGO

```
1 public class Persona
2 {
3     /* Definición de atributos */
4     private int edad;
5     private String nombre;
6
7     /* Retorna la edad */
8     public int getEdad()
9     {
10         return this.edad;
11     }
12 }
```

PRIMERA IMPLEMENTACIÓN

DIAGRAMA



CÓDIGO

```
1 public class Persona
2 {
3     /* Definición de atributos */
4     private int edad;
5     private String nombre;
6
7     /* Retorna la edad */
8     public int getEdad()
9     {
10         return this.edad;
11     }
12
13     /* Cambia la edad */
14     public void setEdad(int edad)
15     {
16         this.edad = edad;
17     }
18 }
```

CÓDIGO FUENTE Y BYTECODE

CÓDIGO FUENTE

```
MBP-de-JAZNA:Code GhostGirl$ ls
Persona.java
MBP-de-JAZNA:Code GhostGirl$ _
```

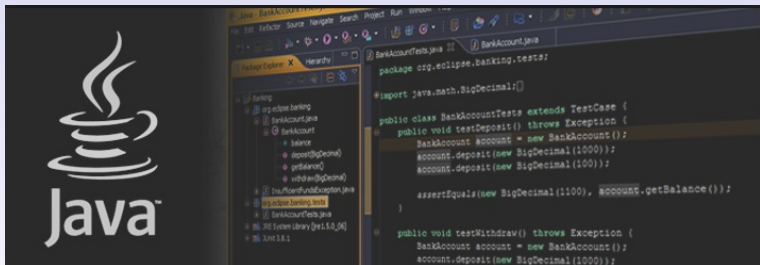
GENERACIÓN DE *bytecode*

```
Code — bash — 80x24
MBP-de-JAZNA:Code GhostGirl$ ls
Persona.java
MBP-de-JAZNA:Code GhostGirl$ javac Persona.java
MBP-de-JAZNA:Code GhostGirl$ ls
Persona.class  Persona.java
MBP-de-JAZNA:Code GhostGirl$ _
```

LET'S GO

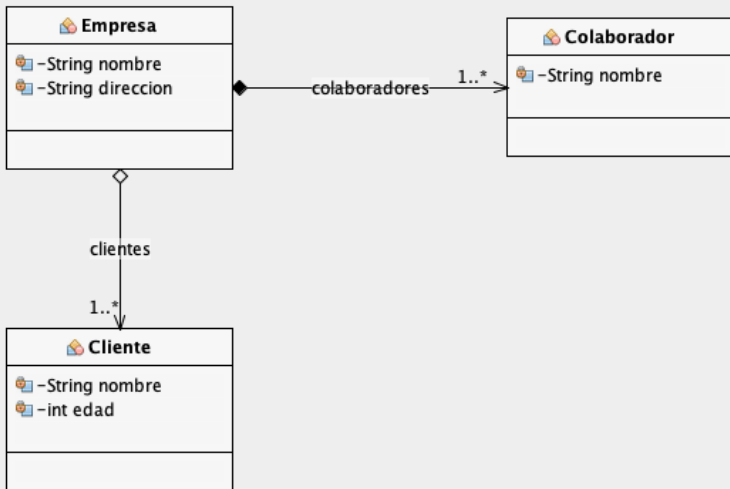


TEMA - RELACIÓN ENTRE CLASES



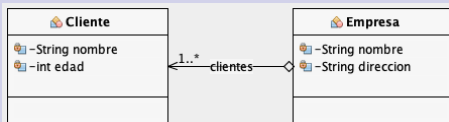
AGREGACIÓN Y COMPOSICIÓN

EJEMPLO



AGREGACIÓN Y COMPOSICIÓN

EJEMPLO



AGREGACIÓN

- Es un tipo de asociación que una clase es parte de otra clase, pero es **débil**.
- Los componentes **SI** pueden ser compartidos por varios objetos.
- La destrucción del objeto *Empresa* **NO** conlleva a la destrucción de los objetos *Cliente*.

ASOCIACIÓN Y COMPOSICIÓN

EJEMPLO



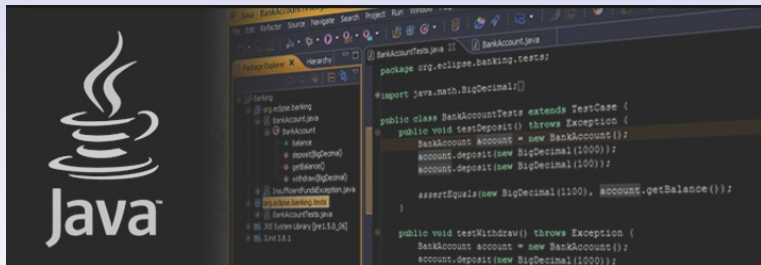
COMPOSICIÓN

- Es un tipo de asociación que una clase es parte de otra clase, pero es **fuerte**.
- Los componentes **NO** pueden ser compartidos por varios objetos.
- La destrucción del objeto *Empresa* **SI** conlleva a la destrucción de los objetos *Empleado*.

LET'S GO







TEMA - HERENCIA













CONCEPTO DE HERENCIA





EJEMPLO











Entrenador

 -int nivel
 -String nombre
 -String rol
 -int federacion

 +Entrenador()
 +Entrenador(int nivel, String nombre, String rol, int federacion)
 +int getNivel()
 +void setNivel(int nivel)
 +String getNombre()
 +void setNombre(String nombre)
 +String getRol()
 +void setRol(String rol)
 +int getFederacion()
 +void setFederacion(int federacion)

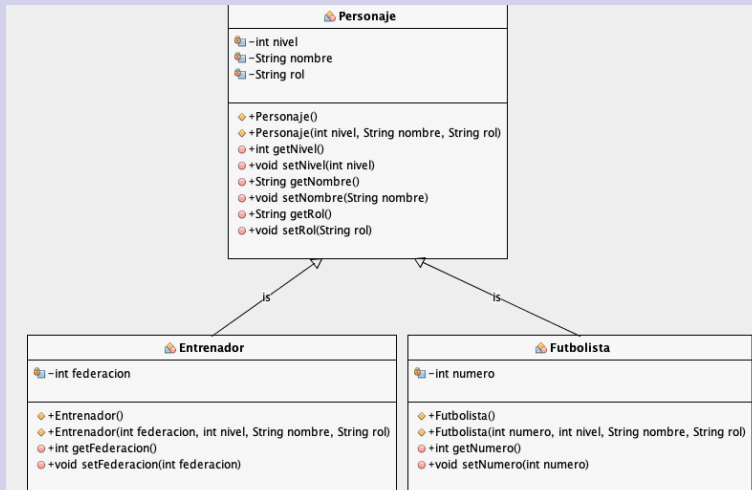
Futbolista

 -int nivel
 -String nombre
 -String rol
 -int numero

 +Futbolista()
 +Futbolista(int nivel, String nombre, String rol, int numero)
 +int getNivel()
 +void setNivel(int nivel)
 +String getNombre()
 +void setNombre(String nombre)
 +String getRol()
 +void setRol(String rol)
 +int getNumero()
 +void setNumero(int numero)

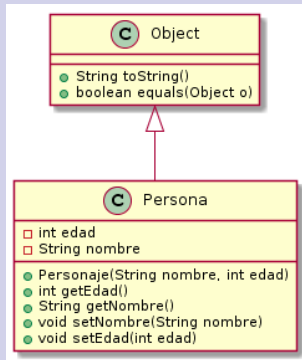
CONCEPTO DE HERENCIA

EJEMPLO



CONCEPTO DE HERENCIA

CLASE *Object*



LET'S GO

