Java

Sintaxis del Lenguaje

TOGETHER. FREE YOUR ENERGIES

Objetivo

- **Lograr que aprendamos a programar en Java, hasta el punto de construir aplicaciones reales en la tecnología.**

Agenda de hoy

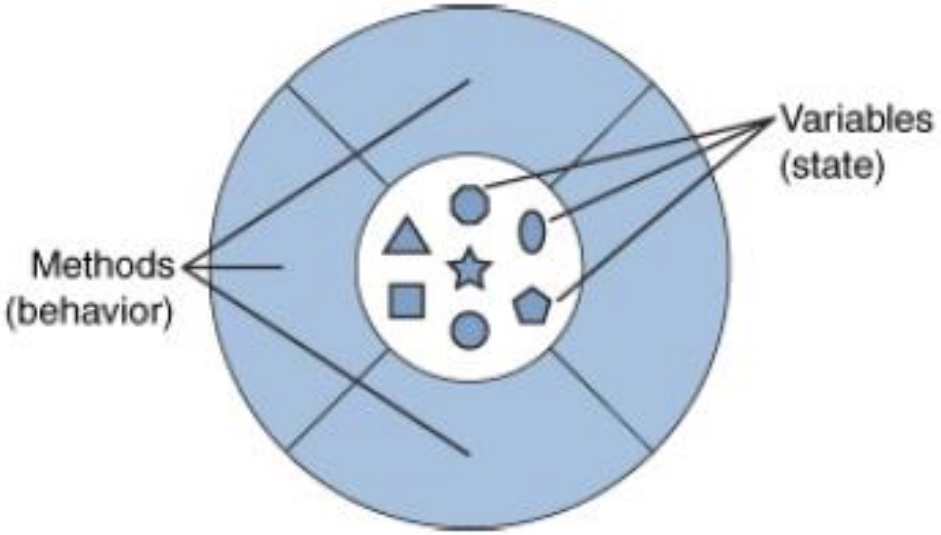
- ✓ **Parte 1: Herramientas de base y J2EE**
  - **Módulo 1:**
    - ✓ **El lenguaje Java**
      - ✓ **(Breve) Repaso de POO**
      - ✓ **POO en Java: Clases y Objetos**

Repaso de POO I

- **Características de un lenguaje orientado a objetos 'puro'**
  - **TODO es un objeto**
  - **Un programa es un conjunto de objetos indicándose entre sí qué hacer, a través del envío de mensajes.**
  - **Todo objeto tiene su propia 'memoria', hecha de otros objetos.**
  - **Todo objeto tiene un tipo**
  - **Todos los objetos de un tipo particular pueden recibir los mismos mensajes.**

Repaso de POO II

- **Objetos**



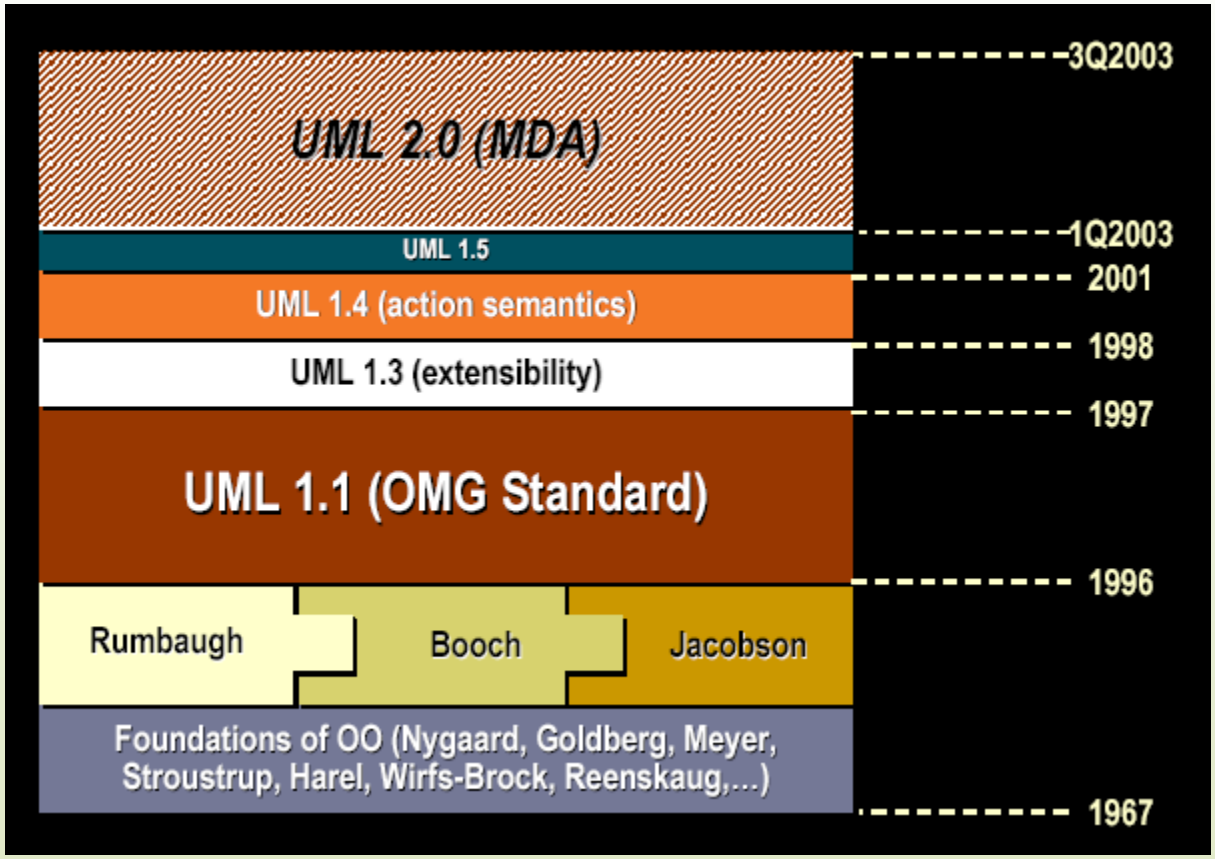
Repaso de POO III

- **Objetos**
  - **Abstracción de software que modela todos los aspectos relevantes de una realidad.**
  - **Representación de la realidad que tiene identidad, estado y comportamiento.**
  - **Entidad real o abstracta con una identidad propia y un conjunto de propiedades y comportamientos que la caracterizan.**

Repaso de POO IV

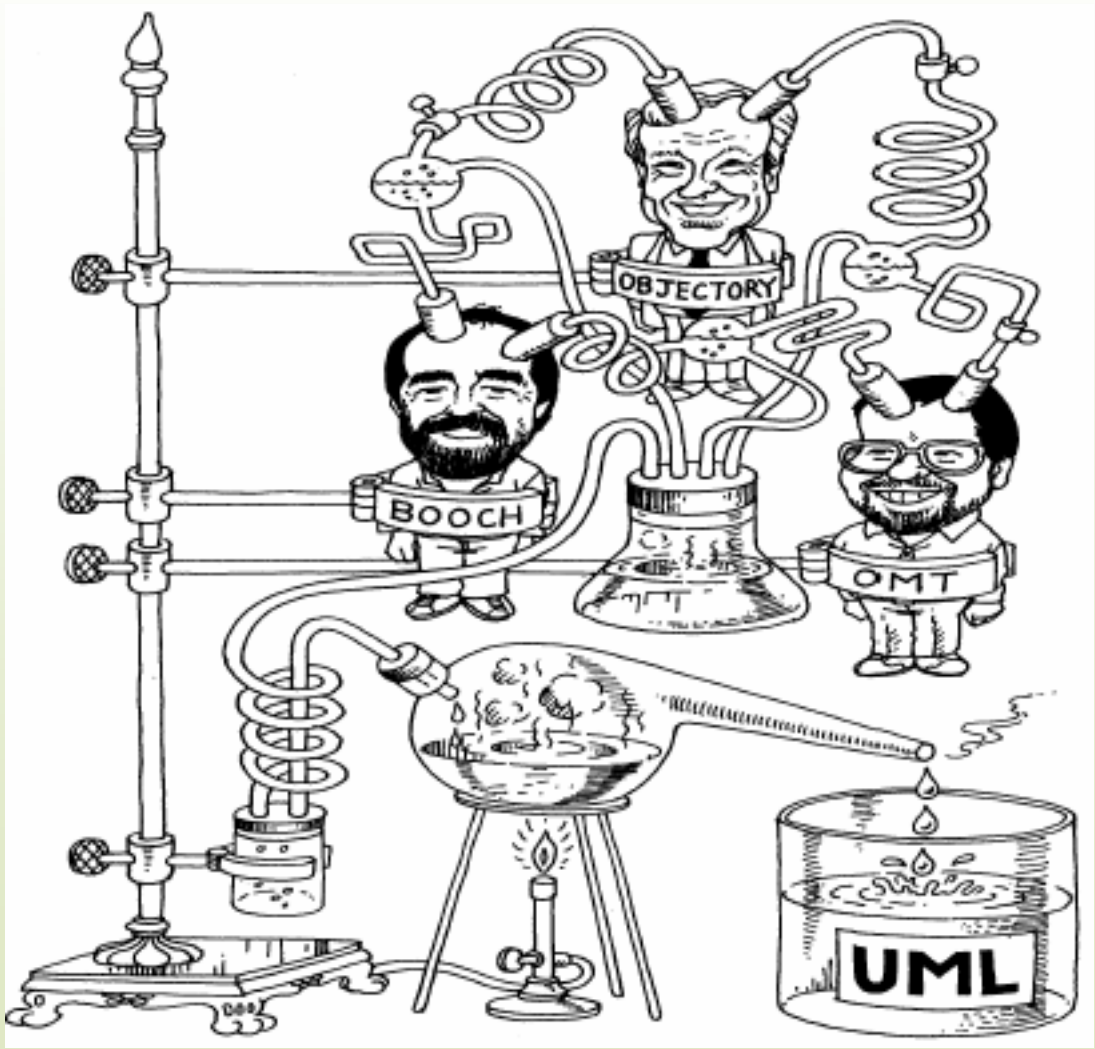
- **Elementos del modelo de objetos**
  - **Abstracción.**
    - **Consiste en la generalización de los atributos y comportamientos de un grupo de objetos.**
  - **Encapsulamiento**
    - **Un objeto se comporta como una caja negra. Sabemos qué operaciones realiza y no necesitamos conocer su implementación.**
  - **Herencia**
    - **Un objeto puede adquirir las propiedades y comportamientos de otro.**
  - **Modularidad**
    - **Es la propiedad de un sistema que fue descompuesto en un conjunto de módulos cohesivos y débilmente acoplados.**

UML-Historia y evolución

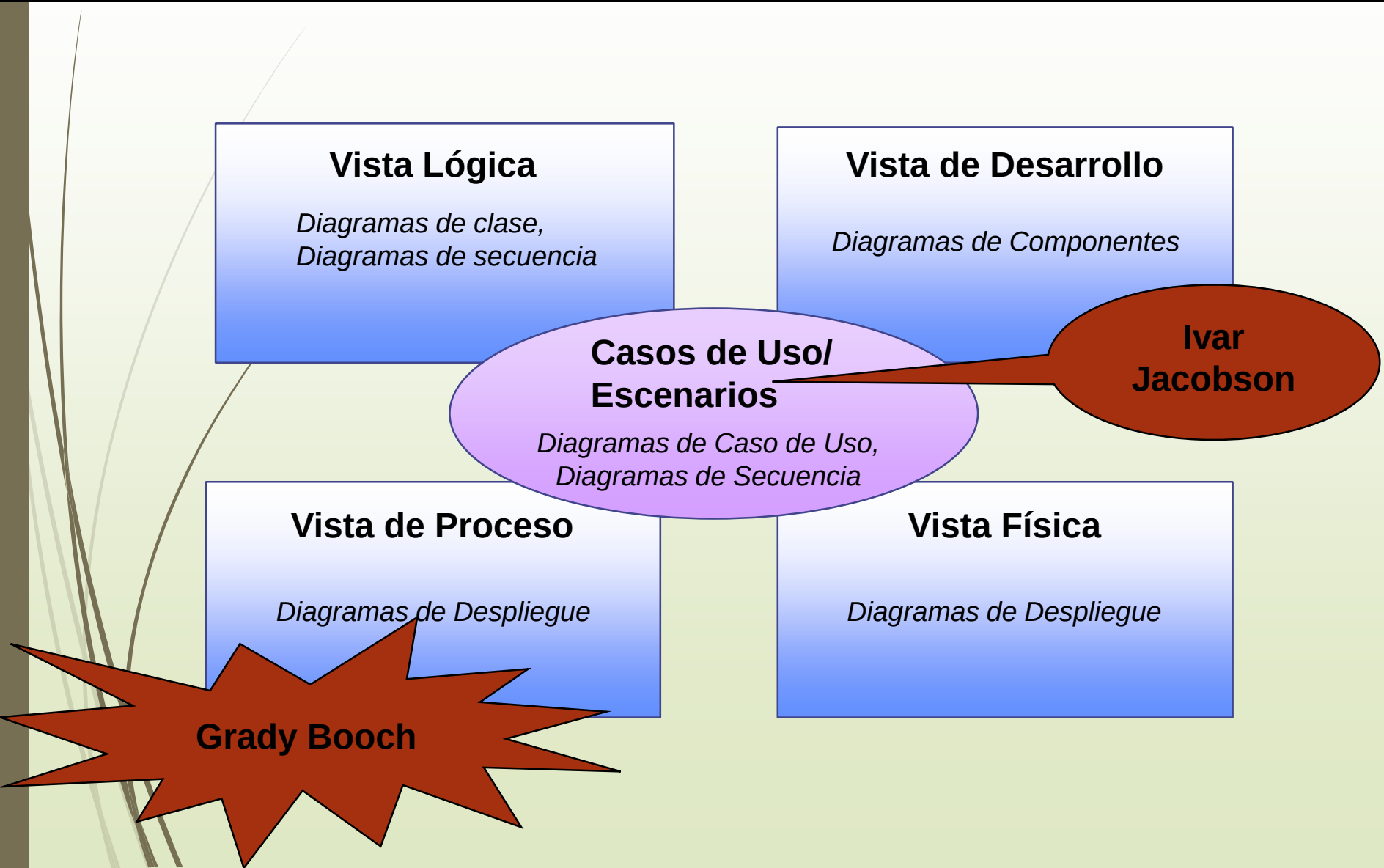


MDA: Model Driven Architecture

Autores de UML



La vista “4+1”


Vista Lógica
Diagramas de clase,
Diagramas de secuencia
Vista de Desarrollo
Diagramas de Componentes
Casos de Uso/
Escenarios
Diagramas de Caso de Uso,
Diagramas de Secuencia
Ivar
Jacobson
Vista de Proceso
Diagramas de Despliegue
Vista Física
Diagramas de Despliegue
Grady Booch

Autores de UML

Modelo de Requerimientos
Diagrama de casos de uso
Modelado del contexto
Modelado de requerimientos

Modelo Lógico-Estático
Diagrama de Estados
Diagrama de Clases
Relaciones entre Clases
Dependencias
Generalización
Asociación

Modelo Lógico-Dinámico
Diagramas de interacción
- Actividad
- Colaboración
- Secuencia

Diagrama de componentes
Diagramas de despliegue

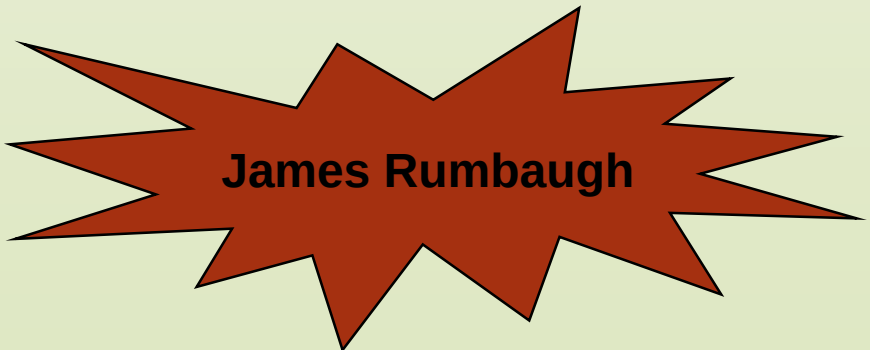
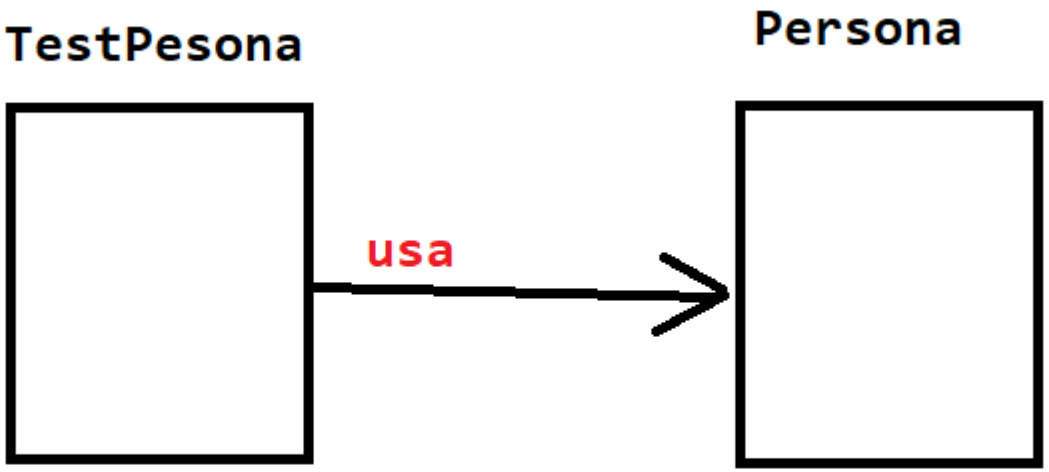


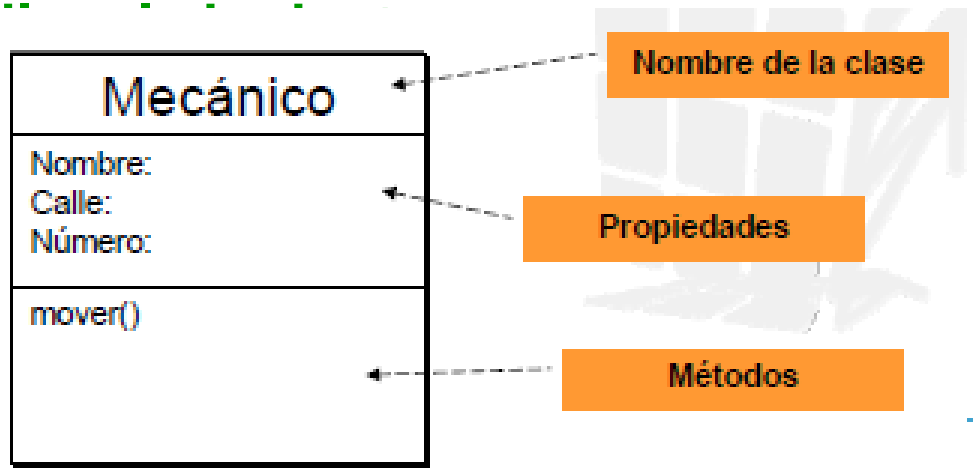
Diagrama de Clases (uml) unified modeling language




```
new Persona();
```

Repaso de POO V

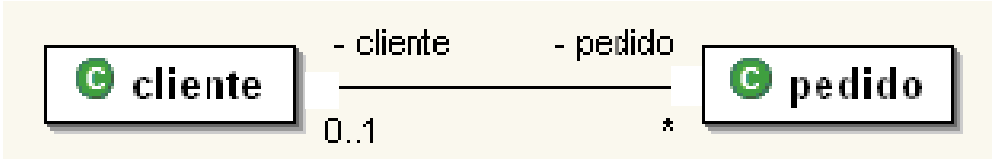
- **Clase**
  - **Una clase es un template o blueprint a partir del cual se construyen objetos.**
  - **“Grupo de objetos marcado por características comunes que incluyen estructura y comportamiento” (Booch)**
  - **Los objetos son instancias de una clase.**
  - **Se suelen representar gráficamente mediante un esquema sim**



Repaso de POO VI

- **Relación de Asociación.**

  Permite asociar objetos que colaboran entre si.



- **La cardinalidad se puede utilizar para indicar el nivel de dependencia.**
  - **1**
  - **0..1**
  - **1..***
  - ***

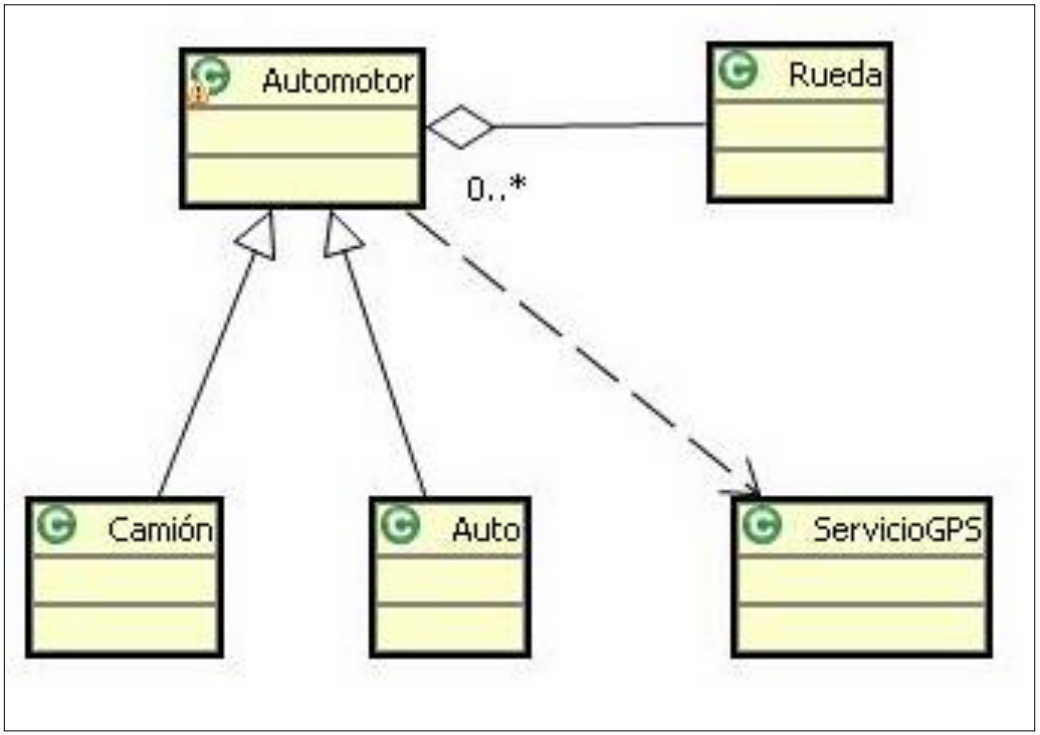
Repaso de POO VII

- **Otras relaciones principales entre clases**

- **Dependencia** 'usa'

- **Agregación** 'tiene'

- **Herencia** 'es un'



Ejercicio

Ejercicio III de la guía

Clases y Objetos en Java I

- **Cómo definir clases en Java**

```
[modificador] class NombreClase {
<Conjunto de propiedades>
<Conjunto de métodos>
}
```

[modificador] puede ser:

- public
- final
- abstract
- **sin modificador: Es** protected**.**

```
public class Producto {
     String descripcion;
     double precio;
     static double iva;

     double calculaPVP(){
          return precio*iva;
     }

     String getDescripcion(){
          return descripcion;
     }

     static double getIva(){
          return iva;
     }
}
```

ABSTRACT

clase abstracta: no permite instancias directas
método abstracto: obliga a clases hijas a implementar

situación similar con los métodos definidos en interfaces

FINAL

clase final: no permite tener hijas/subclases, no deja hacer extends

atributo final: es convertirlo a constante

método final: no permite hacer overriding en las clases hijas

Clases y Objetos en Java

Packages and Class import

Package

```
package com.capgemini.training.java;

class Employee {
....
```

```
java.util.Date today = new java.util.Date();
```

```
package ....
import java.util.*;
class Employee {
...
```

```
package ....
import java.util.Date;
class Employee {
...
```

Import

Clases y Objetos en Java II

- **Cómo definir propiedades en Java**

[modificador] tipo unaPropiedad [= valorInicial];

```
private static int cantidad = 0;
private double precio;
```

Las propiedades son variables de tipo primitivo o pertenecientes a otras clases.

[modificador] puede ser:
- public
- protected
- private
- sin modificador

Junto con uno de los anteriores se puede indicar también:
- static
- final

```
public class Producto {
    String descripcion;
    double precio;
    static double iva;

    double calculaPVP(){
        return precio*iva;
    }

    String getDescripcion(){
        return descripcion;
    }

    static double getIva(){
        return iva;
    }
}
```

Clases y Objetos en Java III

- **Cómo definir métodos en Java**

[modificador] tipoDevuelto nombreMetodo ([tipo1 argumento1, …]) {
<bloque de código>
}

Para devolver un valor se utiliza la instrucción
return <<expresión>>

```
public int sumar (int a, int b){
    return a+b;
}
```

```
public class Producto {
    String descripcion;
    double precio;
    static double iva;

    double calculaPVP(){
        return precio*iva;
    }

    String getDescripcion(){
        return descripcion;
    }

    static double getIva(){
        return iva;
    }
}
```

Clases y Objetos en Java IV

- **Cómo definir métodos en Java**

```
[modificador] tipoDevuelto nombreMetodo ([tipo1 argumento1, …]) {
<bloque de código>
}
```

[modificador] puede ser:
- public
- protected
- private
- sin modificador

Si el método no devuelve ningún valor, el tipo devuelto será void

Junto con uno de los anteriores se puede indicar también:

- static
- final

Clases y Objetos en Java V

- **Cómo definir métodos en Java**

Constructores

```
NombreClase objeto = new NombreClase([param1, …]);
```

```
Alumno unAlumno = new Alumno(“Juan”);
```

Cuando utilizamos el operador new para crear un objeto, en realidad estamos llamando a un método especial denominado constructor

Clases y Objetos en Java VI

Constructores

```
public class Producto {
    String descripcion;
    double precio;
    static double iva;

    public Producto(String d, double p, double i){
        descripcion=d;
        precio=p;
        iva=i;
    }

    double calculaPVP(){
        return precio*iva;
    }

    String getDescripcion(){
        return descripcion;
    }

    static double getIva(){
        return iva;
    }
}
```

```
Producto p = new Producto
("Mouse", 25.0, 21.0);
```

Clases y Objetos en Java VII

Constructores

```
public class TestProducto {

    public static void main(String[] args) {
        Producto p = new Producto("Queso",6.0,16.0);
        System.out.println(p.calculaPVP());
    }
}
```

Clases y Objetos en Java VIII

Acceder a métodos y propiedades de una clase

```
NombreClase objeto = new NombreClase();

Producto p = new Producto();
```

Acceder a una propiedad

objeto.propiedad	p.precio

Acceder a un método

objeto.metodo(parámetros);	p.calculaPVP()

Si un método es static

NombreClase.metodo(parametros);	Producto.getIva()

Clases y Objetos en Java IX

Se dice que un método está sobrecargado cuando existen varios métodos con el mismo nombre y tipo de retorno, pero con distintos parámetros.

Es muy frecuente la sobrecarga de los constructores.

```
public class Alumno {
     String nombre;
     int edad;
     Alumno (String n){
          this.nombre=n;
          this.edad=0;
     }
     Alumno (String n, int e){
          this.nombre = n;
          this.edad = e;
     }
}
```

```
unAlumno = new Alumno("Jaime", 20);

otroAlumno = new Alumno("Jaime");
```

Ejercicio

Ejercicio IV de la guía

Muchas gracias!