

# Orientación a Objetos

**UNRN**

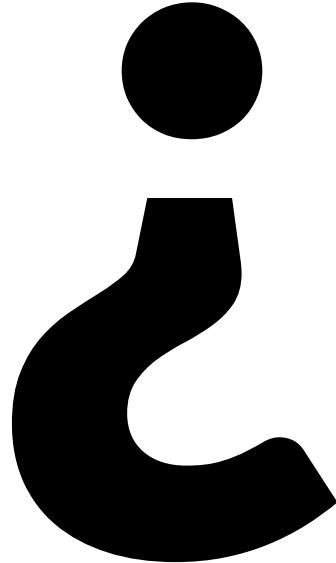
Universidad Nacional  
de Río Negro

r75





**¿Preguntas?**



# ¿Dudas del TP4?





**¿Preguntas?**

---

# Un pequeño flashback a Programación 1



# Variables globales

# Variables globales en C

```
int cuenta = 0;      ← ¡Fuera de las funciones!  
  
void la_funcion() {  
    cuenta++;  
    printf("La funcion fue llamada %d veces\n", cuenta);  
}  
int main() {  
    la_funcion();  
    la_funcion();  
    la_funcion();  
    return 0;  
}
```

Pero, ¿que  
problemas trae su  
uso?

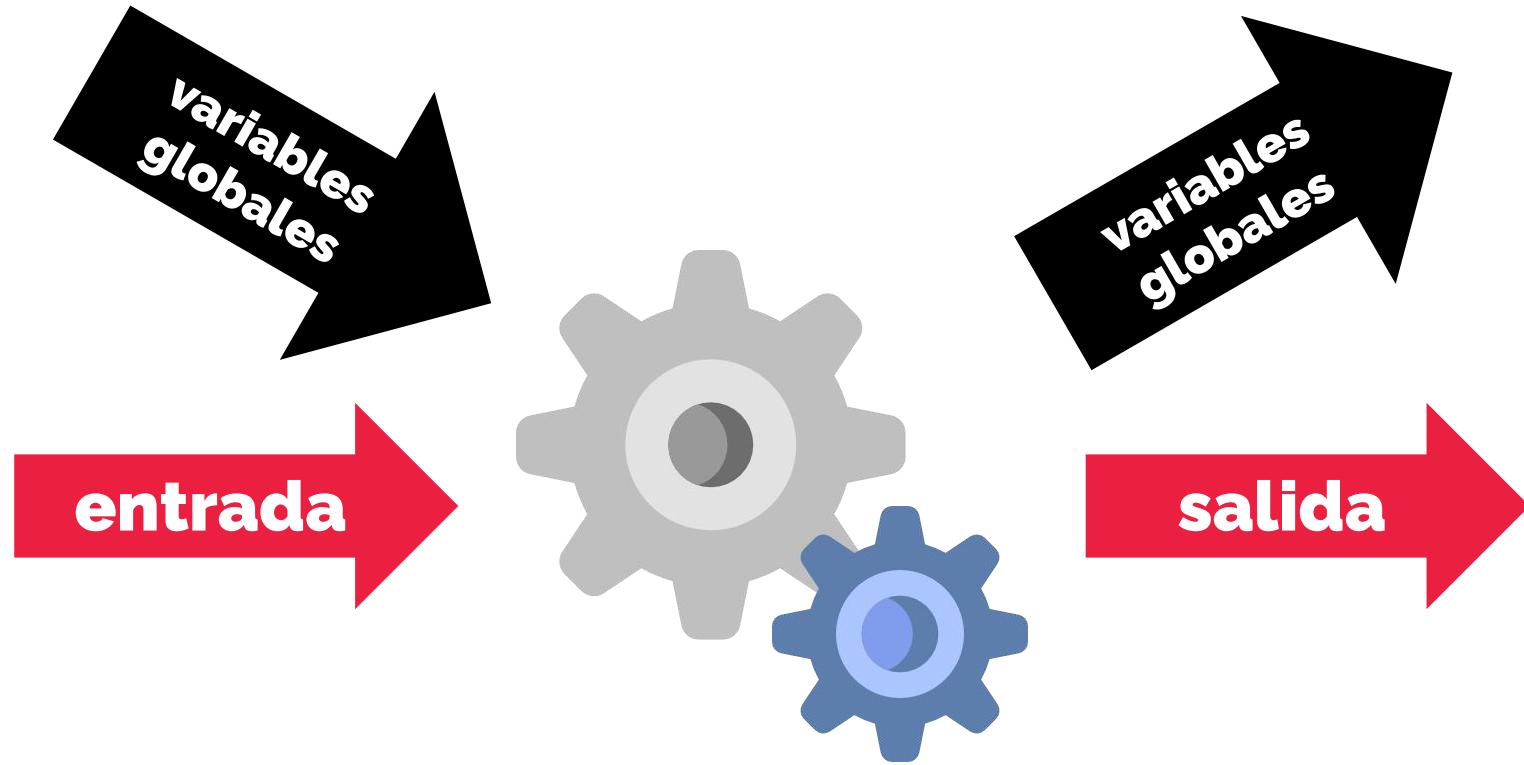


**Su valor puede  
cambiar desde  
cualquier parte**

**No hay control en la forma  
que cambian los valores**

**El debugging es  
mucho más  
complejo**

**Esto hace los tests  
son mucho más  
difíciles**

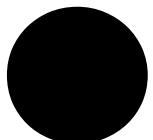


¡Se pierde la **delimitación** de los argumentos y retorno!

**UNRN**

Universidad Nacional  
de Río Negro

**Por eso se  
desaconseja su uso**  
**(no es una cuestión de que no se use)**



**Que tiene que ver  
esto con Java y  
Programación  
Orientada a  
Objetos**

# La Orientación a objetos

**Es una forma de  
asociar código con  
datos**

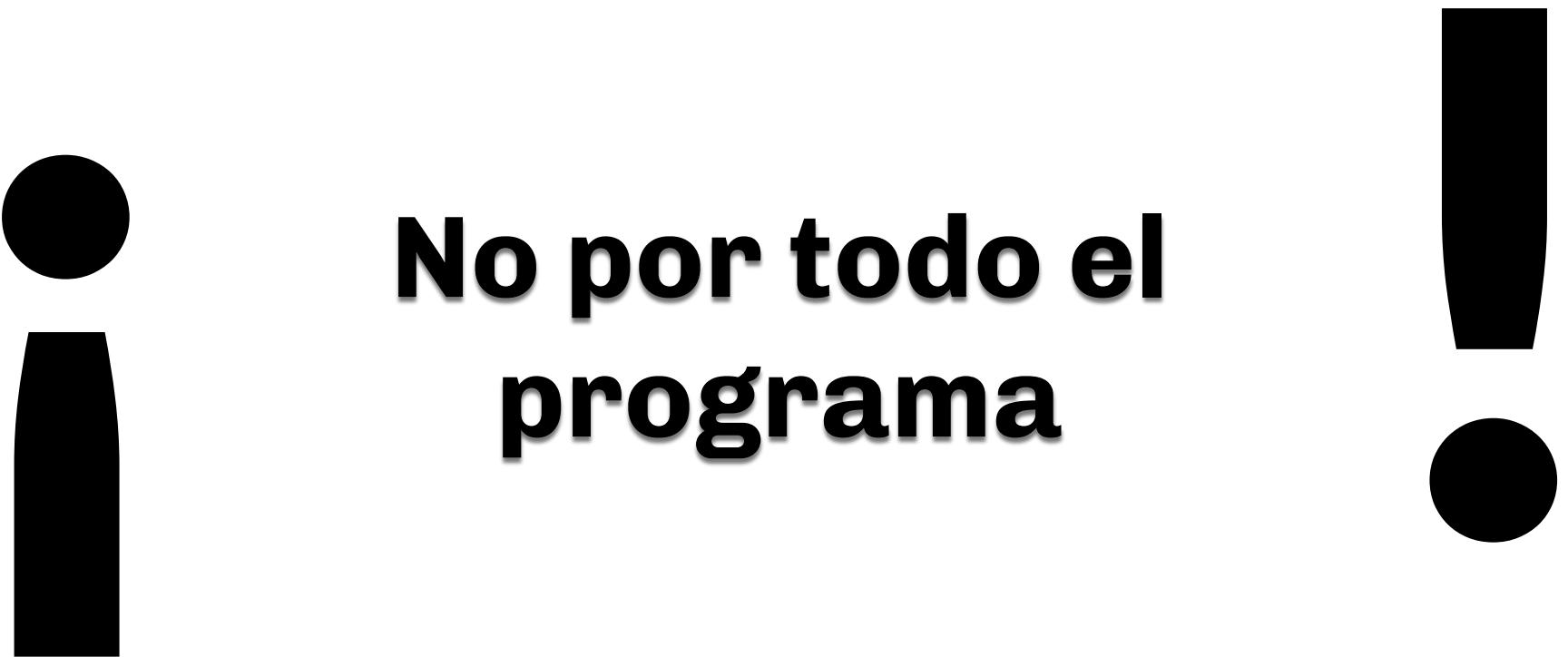
O lo podemos ver  
como usar variables  
no tan globales

# **un objeto es**



# **juntos**

**Ese estado es  
compartido **solo** por  
una instancia del  
objeto**



# No por todo el programa

---

# **Esta es la clave de la Orientación a objetos**

## **Controlar el acceso a los datos**

# Tiempo sexagesimal

```
public class Tiempo{  
    private int hora;  
    private int minutos;  
    private int segundos;
```

datos

```
public Tiempo(int hora, int minutos, int segundos);  
public Tiempo(int segundos);  
public void sumar(int segundos);  
public void sumar(Tiempo otro);  
public void restar(int segundos);  
public void restar(Tiempo otro);  
public int comparar(Tiempo otro);
```

comportamiento

1

# **Encapsulamiento**

**Ocultar la forma en la que  
guardamos el estado y controlar  
su acceso**

**Mantener el estado  
del objeto es la  
misión del  
comportamiento**

**El encapsulamiento se  
logra con los  
calificadores de acceso  
public /protected/private**

# public

Calificador de acceso que indica que es accesible por todos dentro y fuera de la clase.

# **protected**

Calificador de acceso que indica que es accesible por la clase y sus descendientes\*

# **private**

Calificador de acceso que indica que solo será accesible por los miembros de la clase y objetos de la misma.

**Aplican tanto a  
atributos como a  
métodos**

2

# Construcción

**Tomar una clase y darle valores  
a los atributos.**

**La instanciación de una clase.**

# Constructores

```
public class Tiempo{  
    private int hora;  
    private int minutos;  
    private int segundos;  
  
    public Tiempo(int h, int m, int s){  
        hora = h;  
        minutos = m;  
        segundos = s;  
    }  
}
```



Con argumentos

# Instanciando

**MiClase ref = new MiClase(argumentos);**



# Llamando al constructor

```
public class TiempoApp{  
    public static void main(String[] args){  
        Tiempo uno = new Tiempo(10,2,40);  
        Tiempo dos = new Tiempo(10,30,40);  
        uno.comparar(dos);  
    }  
}
```

**Cada instancia tiene  
su propio estado**

# Ciclo de vida II

```
public class TiempoApp{  
    public static void main(String[] args){  
        Tiempo uno = new Tiempo(10,2,40);  
        Tiempo dos = new Tiempo(10,2,40);  
    }  
}
```

*Qué pasa con uno y dos aca*

3

## Destrucción

Cuando no hay más referencias  
apuntando al objeto

**Las clases, definen  
tipos (como typedef)**

# Por eso, podemos hablar de

```
public class Tiempo{  
  
    public void sumar(Tiempo otro){  
        segundos = segundos + otro.segundos;  
        minutos = minutos + otro.minutos;  
        horas = horas + otro.horas;  
    }  
}
```

Dentro de la clase Tiempo

# Pero también de atributos y variables

```
public class Tiempo{  
    private Tiempo masTiempo;  
  
    public void sumar(Tiempo otro){  
        codigo para sumar  
    }  
}
```

Dentro de la clase Tiempo



**¿Preguntas?**

**¿Como nos podemos  
referir a la  
instancia?**

# Superposición de identificadores

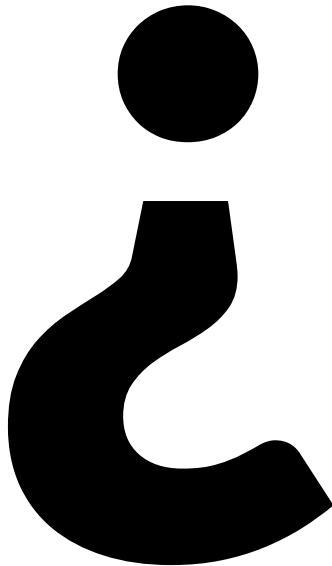
```
public class Tiempo{  
    private int hora;  
    private int minutos;  
    private int segundos;  
  
    public Tiempo(int hora, int minutos, int segundos){  
        hora = hora;  
        minutos = minutos;  
        segundos = segundos;  
    }  
}
```

¿como desempatamos?

¿a qué nos estamos refiriendo?

# Desempatando con this

```
public class Tiempo{  
    private int hora;  
    private int minutos;  
    private int segundos;  
  
    public Tiempo(int hora, int minutos, int segundos){  
        this.hora = hora;  
        this.minutos = minutos;  
        this.segundos = segundos;  
    }  
}
```



¿this?  
What's this



4

**this**

**La referencia al objeto**

**Todos los objetos tienen una  
referencia a si mismos.**



**¿Preguntas?**

---

# ¿Como se crea una clase?

```
public class NombreClase {  
    espacio para atributos  
    public NombreClase(){  
        constructor  
    }  
    espacio para métodos  
}
```

2

0

# Las clases van en CamelCase

2

# Los atributos van en dromedarioCase

1

2

# **Los atributos como mínimo, protected**

2



**¿Preguntas?**

5

## método

**El código de una clase, llamar a uno se denomina *invocar* o pasar un mensaje.**

# 6

## Interfaz

**El conjunto de métodos de una clase, esta determina el comportamiento del objeto**

7

## atributo

**La información definida en una clase y que recibe valores para transformarse en el estado de un objeto**



**¿Preguntas?**

# **Primer enfoque a Diseño Orientado a Objetos**

# Pensemos algunas clases

**Indiquen:**  
**Nombre**  
**Atributos**  
**Métodos**

Un ejemplo

Auto

atributos

cilindrada motor  
cantidad de cambios  
potencia de frenos  
conductor

métodos

acelerar  
frenar

[10 minutos]

# **Las primeras abstracciones de la realidad**

# ¿Qué es un Auto?

Auto

atributos

cilindrada motor

cantidad de cambios

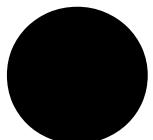
potencia de frenos

conductor

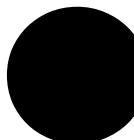
métodos

acelerar

frenar



# Cómo sería una Bicicleta



# Y una Moto?

# Vehículo

Vehículo  
atributos  
    conductor  
métodos  
    acelerar  
    frenar

# **El paso de Auto a Vehículo es una generalización**

8

## Generalización

Pasar de una clase más cercana  
a la realidad a un concepto más  
abstracto

9

# Especialización

Pasar de una clase más abstracta a una cercana a la realidad

1

## Herencia

**Es la relación entre clases de un mismo tipo.**

- La clase que hereda especializa a la madre.
- La clase madre es la generalización de la hija.

0

# Herencia

Los métodos definidos por el padre, son  
utilizables por los hijos

# Herencia II

[19:11] Santiago Tosoni

Los autos son vehículos, pero no todos los  
vehículos son autos

---

# Auto [Versión 1]

atributos

cilindrada motor

cantidad de cambios

potencia de frenos

conductor

métodos

acelerar

frenar

¡Queda mucho afuera!

1

# Abstracción

**es el proceso de identificar las características esenciales de un objeto y representarlas de manera simplificada. Dejando de lado detalles que no tengan que ver con el objetivo.**

1

Identificar lo  
relevante a la tarea  
es crítico

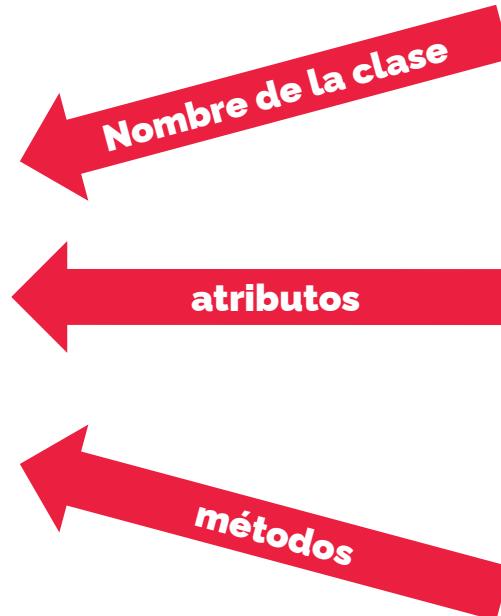
**Para comprender  
mejor estas  
relaciones**

# **Diagrama de clases**

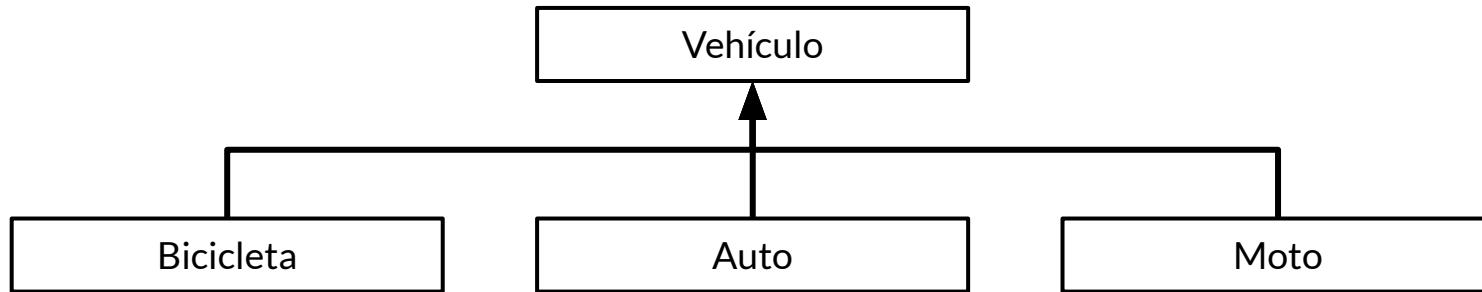
# **UML**

# Detalle de clase

NombreClase
-atributo: TipoAtributo -tributo: Tributo
+cambiaAtributo(TipoAtributo) +obtieneAtributo(): TipoAtributo +saluda()



# Herencia en UML



# Concepto clave

1

**Composición  
es una relación de tipo  
"todo/parte", en donde la parte  
no puede existir por su cuenta.**

2

**auto-motor  
edificio-ambiente**

1

## Agregación

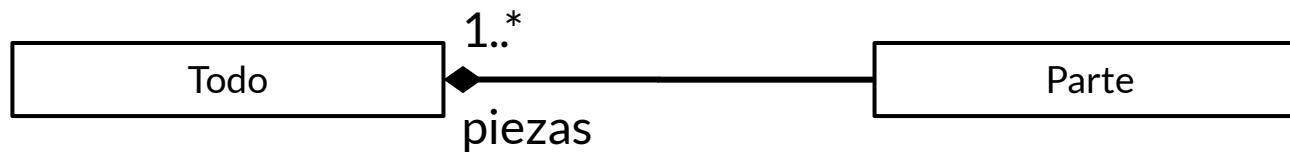
es una relación en la que la clase contenida en otra, puede existir de manera separada.

3

auto-conductor

# Notación UML para relaciones

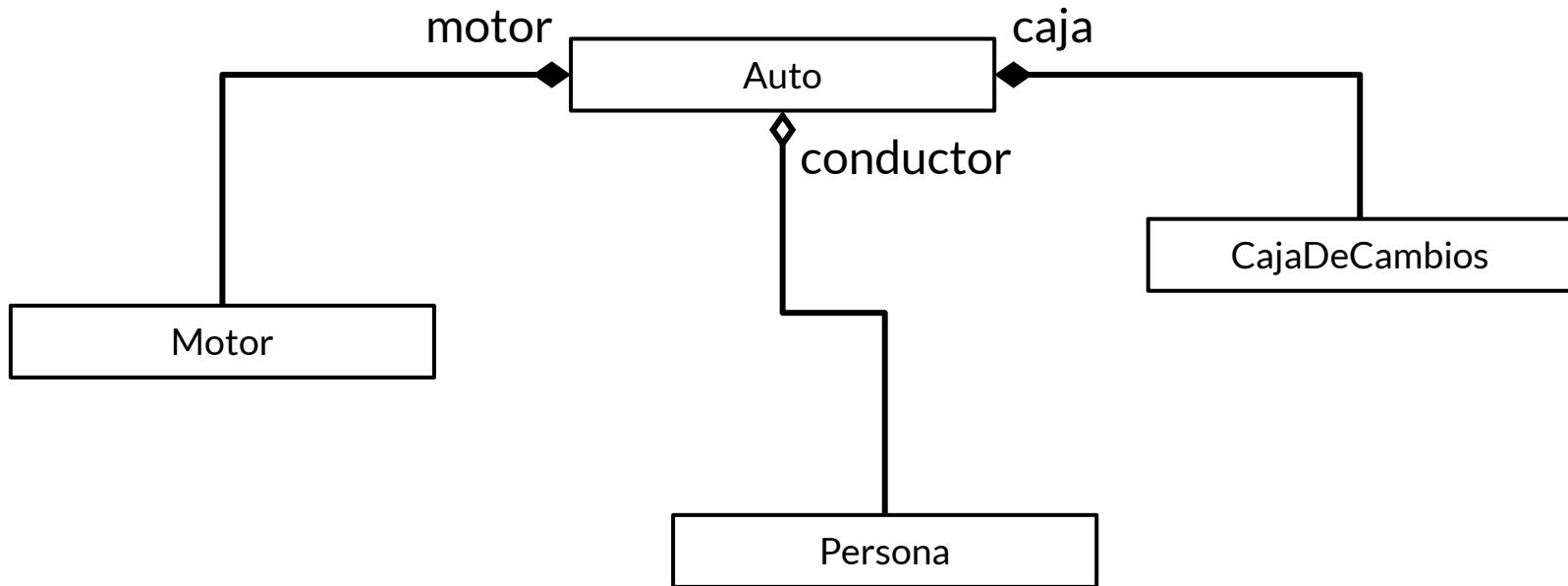
Composición



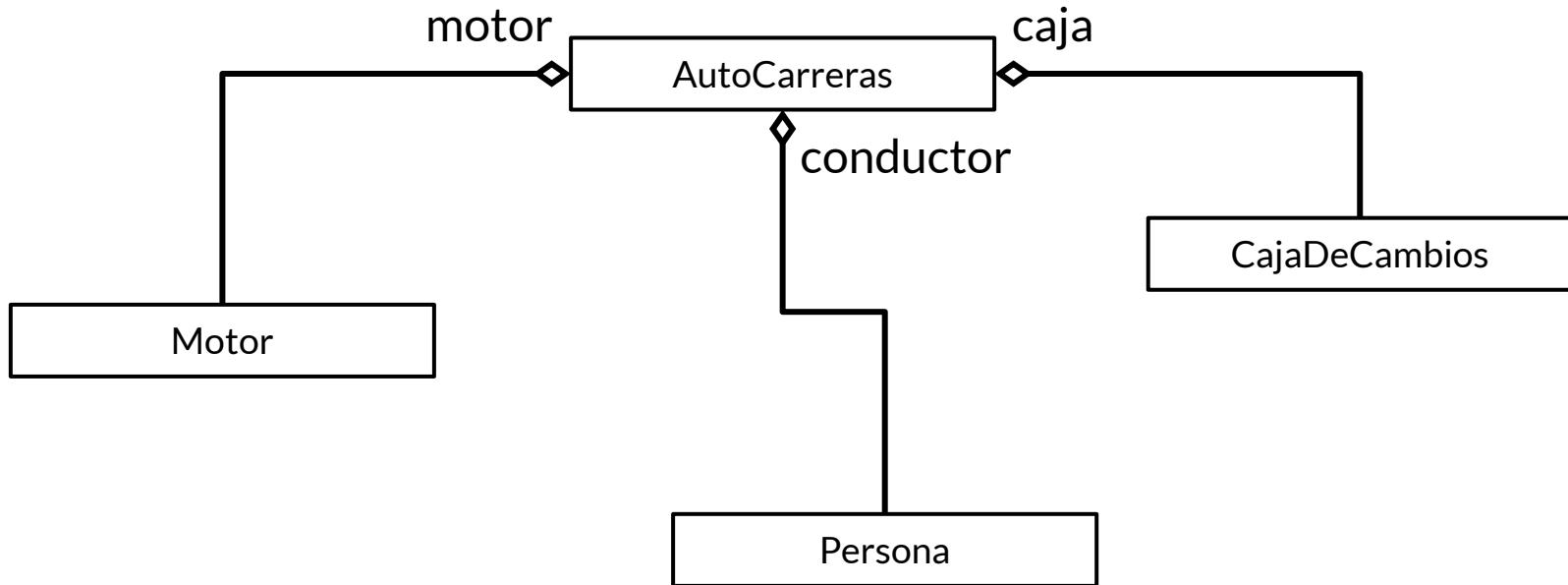
Agregación



# UML para Auto



# UML para Auto de carreras





**¿Preguntas?**

—

# Para seguir ejercitando

# array

¿que es? ¿que representa?  
¿que lo describe?  
¿qué podemos hacer con él?

---

# Tiempo

¿que es? ¿que representa?  
¿que lo describe?  
¿qué podemos hacer con él?

---

# dinero

¿que es? ¿que representa?  
¿que lo describe?  
¿qué podemos hacer con él?

---

# color

¿que es? ¿que representa?  
¿que lo describe?  
¿qué podemos hacer con él?

---

# fecha

¿que es? ¿que representa?  
¿que lo describe?  
¿qué podemos hacer con él?

**¿Y cosas menos  
“técnicas”?**

---

# bote

¿que es? ¿que representa?  
¿que lo describe?  
¿qué podemos hacer con él?

---

# Cuenta bancaria

¿que es? ¿que representa?  
¿que lo describe?  
¿que podemos hacer con él?



**¿Preguntas?**

**La POO es una  
forma de  
representar cosas**

**Es muy importante  
tener en cuenta el  
contexto**

# No es lo mismo

# **Una moto para un taller**

# **Que una moto para su usuario**

—

**La próxima clase  
seguimos con este  
tema.**

—

# Más ejemplos

# **Vehículo**

## **atributos**

**modelo**

**velocidad**

**peso**

## **métodos**

**acelerar**

**frenar**

**Vehiculo**  
**atributos**  
    **modelo**  
    **motor**  
    **frenos**  
**métodos**  
    **acelerar**  
    **frenar**  
    **calcularPeso**

# **Motor** **atributos**

**modelo**

**peso**

**métodos**

**acelerar**

**Frenos**  
**atributos**  
    **modelo**  
    **conABS**  
**métodos**  
    **aplicar**

# **Auto extiende Vehiculo**

# **Camion extiende Vehiculo**

**atributos**

**cargamento**

**peso**

**métodos**

**cargar**

**descargar**

—

# Más ejemplos

# **Empleado**

## **atributos**

**nombre**

**apellido**

**legajo**

**teléfono**

**métodos**

**recibirTarea**

# **Cliente**

## **atributos**

**nombre**

**apellido**

**documento**

**teléfono**

## **métodos**

**comprar**

**devolver**

# **Persona**

**nombre**

**apellido**

**documento**

**teléfono**

# **Cliente extiende Persona**

# **Empleado extiende Persona**

# **Empleado atiende Cliente**



**¿Preguntas?**

# TP5

## Orientación a Objetos 1

**unrn.edu.ar**

**UNRN**

Universidad Nacional  
de Río Negro



| unrnionegro