



UNRN

Universidad Nacional
de Río Negro



| unrionegro

Programación 2

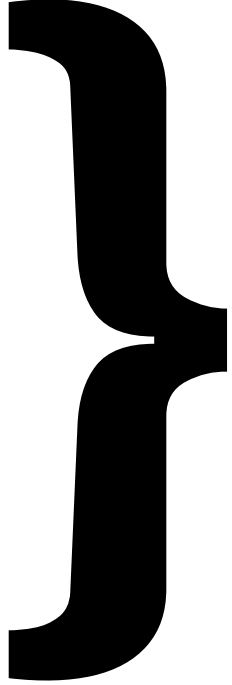
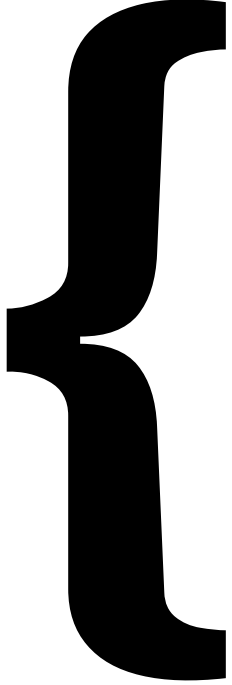
IX

UNRN

Universidad Nacional
de **Río Negro**

Martín René Vilugrón
mrvilugron@unrn.edu.ar

Revisemos los casos analizados



**Necesito que
alguien tome nota
(en Discord si es
posible)**

Rent a car

Una pequeña arrendadora de vehículos sin chofer que opera en la región de San Carlos de Bariloche.

Casa de Té

Una casa de té de los kilómetros está estudiando la posibilidad de desarrollar un sistema para las comandas (pedidos de los clientes)

Agencia de Turismo

Una agencia turística de la localidad, vende paquetes turísticos a destinos del país, desea comenzar a desarrollar un sistema para la gestión de las ventas de los paquetes.

Biblioteca

La biblioteca de la universidad está buscando informatizar su catálogo de libros y revistas.

Taller mecánico

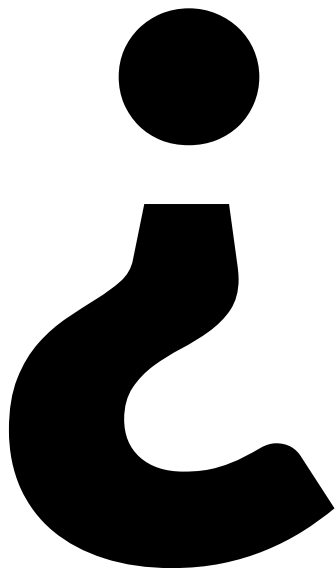
Un taller mecánico está buscando desarrollar un sistema para el control de los vehículos que son reparados.



¿Preguntas?

Paquetes

1: Evitan conflictos de nombres



**Auto
para el taller
para el rentacar
¿debieran tener
nombres
diferentes?**



2: Mantienen juntas partes relacionadas

**Es razonable pensar
que las piezas del
auto van a estar
“cerca” y juntas**

Creación de paquetes

package un.directorio.por.punto;

es

un/directorio/por/punto

y en un proyecto gradle es

src/main/java/un/directorio/por/punto

aunque
canonicamente

Se utiliza el nombre de dominio al revés

ar.edu.unrn.andina.programacion.practica.seis

Mantengámoslo simple

`programacion.practica.seis`

(la práctica vendrá con esta estructura)



¿Preguntas?

Uso de paquetes

Algo que venimos usando

```
import java.util.Scanner; // del "sistema"
```

```
import programacion.practica.cuatro.Arreglo; // de algo propio
```

```
import java.util.*; //aunque correcto, importen lo que se utiliza  
import programacion.practica.cuatro.*;
```




¿Preguntas?

Objetos en Java

Definición

Estructura base ¡Pero esto es una clase!

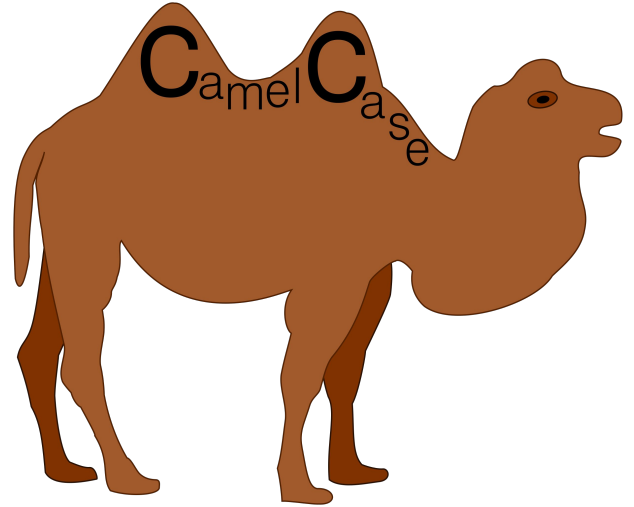


```
class MiClase{
    String unaCadena;

    MiClase(String argumento){
        unaCadena = "Hola objetos" + argumento;
        // El Constructor le da un valor a los atributos
        // las instrucciones sobre la inicialización
    }

    int unMetodo(int argumento){
        this.unaCadena = String(argumento);
        // código
    }
}
```

Las clases, en CamelloCase



Los atributos y métodos en dromedarioCase



¿Construcción?



**El constructor es
encargado de
instanciar la clase**
(cuando sucede al hacer new)



Estructura base

```
class MiClase{
    String unaCadena;

    MiClase(String argumento){
        unaCadena = "Hola objetos " + argumento;
        // El Constructor le da un valor a los atributos
        // las instrucciones sobre la inicialización
    }
}
```


**Es un método
especial *sin retorno*
con el mismo
nombre que la clase**

opcionalmente
con argumentos

¡Múltiples constructores!

```
class MiClase{
    String unaCadena;

    MiClase(){
        unaCadena = "Hola Objetos";
        // Sin argumentos, y si no dice nada...
    }

    MiClase(String cadena){
        unaCadena = cadena; // Con argumentos
    }
}
```

Dos instanciaciones válidas

**¡Acá la clase se
vuelve objeto!**



```
MiClase uno = new MiClase();
```

```
MiClase dos = new MiClase("Roberto");
```

***Que vamos a ver más adelante**

Esto es posible por

sobrecarga de métodos

***Que vamos a ver más adelante**

**Que busca el método
cuyos tipos de
argumento coincidan**

OJO

¡Solo se ejecuta uno!


(aunque es posible encadenarlos)

¡Pero se pueden encadenar!

```
class MiClase{
    String unaCadena;

    MiClase(){
        this("el argumento");
    }

    MiClase(String argumento){
        unaCadena = argumento; // Con argumentos
    }
}
```



—

y Sobre la Sobrecarga

UNRN

Universidad Nacional
de Río Negro

Sobrecarga de métodos

```
class Sobrecargado{  
  
    int metodoSobrecargado(int a, int b);    //(1)  
    int metodoSobrecargado(int a, float b); //(2) } OK  
  
    int metodoSobrecargado(int uno, int dos); // ¡es igual a (1)!  
  
}
```

solo 've' la combinación de tipos

El nombre que tenga el argumento no es tenido en cuenta

El orden de los tipos importa

```
class Sobre cargado Dos{
```

```
void metodoSobre cargado(int a, float b);
```

```
void metodoSobre cargado(float a, int b);
```

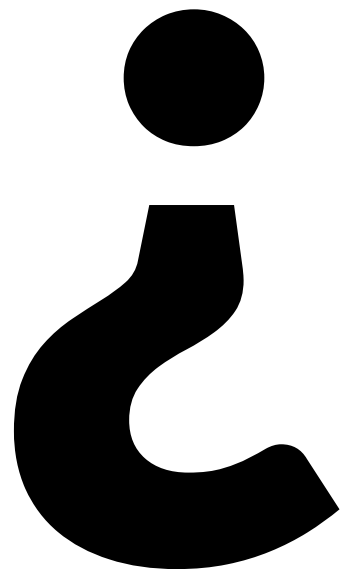
```
}
```

} OK también



¿Preguntas?

this
¿what's this?



**this es la referencia a
la instancia del objeto
creado**

Estructura base

```
class MiClase{  
    int nombreAtributo;  
  
    void metodo(int argumento){  
        nombreAtributo = argumento;  
        // acá no hay conflicto  
    }  
    void otroMetodo(int nombreAtributo){  
        this.nombreAtributo = nombreAtributo;  
        // pero acá sí  
    }  
}
```



para resolver la
superposición de nombres

Ejemplo

El arreglo dinámico

Arreglo Dinámico

Se desea crear un arreglo que pueda recibir cualquier cantidad de números enteros, incluso si esta cantidad no es conocida de antemano.

Atributos un arreglo

Operaciones

(de interés particular)

construcción

cambiar Tamaño

...y lo del miércoles

El arreglo dinámico

```
class ArregloDinamico{

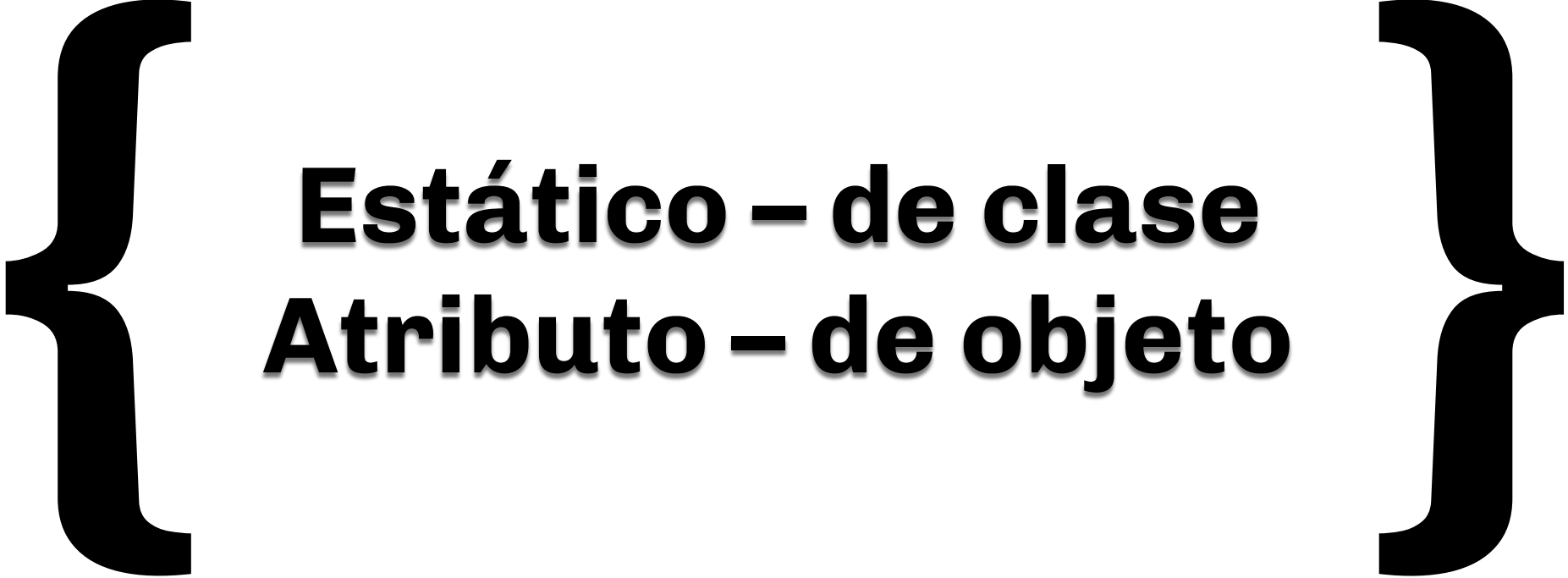
    int[] arreglo;

    ArregloDinamico(int largo){
        this.arreglo = new int[largo];
    }
    void cambiarTamano(int tamano){
        int[] nuevo = new int[tamano];
        copiar(nuevo, arreglo);
        arreglo = nuevo;
    }
    static void copiar(int[] destino, int[] origen){ ... }
}
```

static

Es un método de clase, no tiene asociada una instancia de la misma.

(no hay `this`)



Estático – de clase
Atributo – de objeto

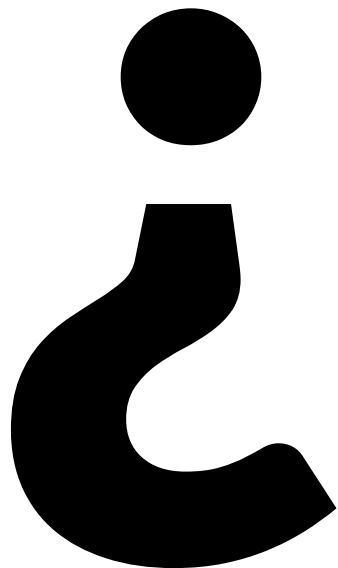
¿Que diferencias hay?

```
void copiar(int[] origen){ ... }
```

```
static void copiar(int[] destino, int[] origen){ ... }
```




¿Preguntas?



**¿Qué pasa
cuando
reemplazamos el
arreglo?**



¿Es posible hacer esto?

```
public static void main(String[] args){  
    ArregloDinamico dyna = new ArregloDinamico(10);  
    // le damos algo de uso..  
    dyna.arreglo = new int[0];  
}
```

yeap

¿Y que pasó con el arreglo original?

Lo perdimos
¡fue reemplazado!



Pero lo podemos proteger

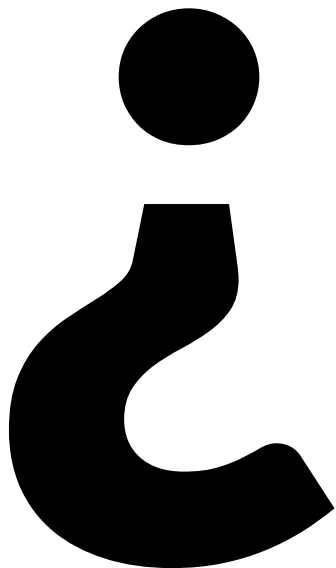


El arreglo dinámico

```
public class ArregloDinamico{  
  
    private int[] arreglo;  
  
    public ArregloDinamico(int largo){  
        this.arreglo = new int[largo];  
    }  
    public void cambiarTamano(int tamano){  
        int[] nuevo = new int[tamano];  
        copiar(nuevo, arreglo);  
        arreglo = nuevo;  
    }  
    public static void copiar(int[] destino, int[] origen){ ... }  
}
```

Con los calificadores de acceso

public y private*



**La regla
general es**



Los atributos

private

Los métodos

public

Esto encapsula la implementación



***Esconde los
detalles de la
implementación***

Arreglo Dinámico II

Se desea crear un arreglo que pueda recibir cualquier cantidad de números enteros, incluso si esta cantidad no es conocida de antemano.

Se desea que la complejidad al momento de agregar elementos sea $O(1)$ hasta 2 veces la capacidad inicial.

El arreglo dinámico versión 2

```
public class ArregloDinamico{

    private int[] arreglo; //este arreglo está pensado para tener lugar extra
    private int largo; // este es el largo visible desde el exterior

    ArregloDinamico(int largo){
        this.largo = largo; // el "largo" del arreglo
        this.arreglo = new int[largo*2]; //capacidad extra
    }
    void cambiarTamano(int tamano){
        if tamano < arreglo.lenght {
            int[] nuevo = new int[tamano];
            copiar(nuevo, arreglo);
            arreglo = nuevo;
        }else{
            this.largo = tamano;
        }
    }
}
```

Necesita que agreguemos

```
public int obtener(int posicion)
    throws IndexOutOfBoundsException {
    int valor;
    if posicion < largo{
        valor = arreglo[posicion];
    }else{
        throw new IndexOutOfBoundsException();
    }
    return valor;
}
```


Otras operaciones interesantes

```
public void modificar(int posicion, int valor)
```

```
public void insertarAlFinal(int valor)
```

```
public void insertarAlPrincipio(int valor)
```

```
public void insertar(int posicion, int valor)
```

```
public int removerDelFinal()
```

```
public int removerDelPrincipio()
```

```
public int remover(int posicion)
```



¿Preguntas?

Un ejemplo de métodos privados

Metodos privados

```
private boolean esPosicionValida(int posicion){  
    return posicion < largo && posicion >= 0;  
}
```

Necesita que agreguemos

```
public int obtener(int posicion)
    throws IndexOutOfBoundsException {
    int valor;
    if (esPosicionValida(posicion){
        valor = arreglo[posicion];
    }else{
        throw new IndexOutOfBoundsException();
    }
    return valor;
}
```

**¿Pero qué es lo que
hacemos con el
valor false?**

Necesita que agreguemos

```
private void esPosicionValida(int posicion)
    throws IndexOutOfBoundsException {
    boolean esValido = (posicion < largo) && (posicion >= 0)
    if (!esValido){
        throw new IndexOutOfBoundsException();
    }
}
```

Y después, solo tenemos que...

```
public int obtener(int posicion)
    throws IndexOutOfBoundsException {
    esPosicionValida(posicion);
    return arreglo[posicion];
}
```


Ayuda a simplificar las funciones

```
public void modificar(int posicion, int valor)
    throws IndexOutOfBoundsException {
    esPosicionValida(posicion);
    arreglo[posicion] = valor;
}
```

TP6

Es necesario que funcione con gradle como está en el repositorio

unrn.edu.ar

