
UNRN
Universidad Nacional
de Río Negro
| unrionegro

Orientación a objetos

Martín René Vilugron
martinvilu@unrn.edu.ar

PROGRAMACIÓN 2

2023

XIII

UNRN

Universidad Nacional
de **Río Negro**

UNRN

Universidad Nacional
de **Río Negro**

UNRN

Universidad Nacional
de **Río Negro**

UNRN

Universidad Nacional
de **Río Negro**

Revisemos

El Supermercado

Producto

```
public Producto{
     protected float precio;
     protected int codigo;
     protected String nombre;
     protected int cantidad;

     public float calcularPrecio() { return precio * cantidad; }

     public Producto fraccionar(int fraccion) throws NoHaySuficiente{
          if (fraccion > this.cantidad){
               throw new NoHaySuficiente();
          }
          this.cantidad = this.cantidad - fraccion;
          return new Producto(this, cantidad);
     }
     public void fusionarCantidades(Producto otro) throws ProductoIncorrecto{
          this.cantidad = this.cantiadad + otro.cantidad;
     }
     public boolean esMismoProducto(Producto otro){ return this.codigo == otro.codigo; }
}
```

ProductoGranel

```
public ProductoGranel extends Producto{
     protected float descuento;
     protected int unidadesPorBulto;
     protected Producto delPaquete;

     public ProductoGranel(... lo de Producto+, float descuento, int unidadesPorBulto){
          super(... lo de Producto+);
          this.descuento = descuento;
          this.unidadesPorBulto = unidadesPorBulto;
     }

     public float calcularPrecio() {
          return (precio * cantidad) * descuento;
     }

}
```

ProductoSuelto

```
public ProductoSuelto extends Producto{
     protected String unidadMedida;

     public ProductoSuelto(... lo de Producto+, String unidad){
          super(... lo de Producto+);
          this.unidadMedida = unidad;
     }
     public float calcularPrecio() {
          return (delPaquete.precio * cantidad) * descuento;
     }

}
```

Supermercado

```
public class ConjuntoProductos{
     protected ArrayList<Producto> conjunto;

     public void agregar(Producto producto);
     public Producto tomarProducto(String nombre, int cantidad);
     public Producto tomarProducto(int codigo, int cantidad);
}

public class Supermercado extends ConjuntoProductos{
     protected Carrito[] carritos;
     protected Estanteria[] gondolas;
}

public Carrito extends ConjuntoProductos{
     public Producto sacar(String nombre);
     public float calcularTotal();
}
public Estanteria extends ConjuntoProducto {}
```


¿Preguntas?

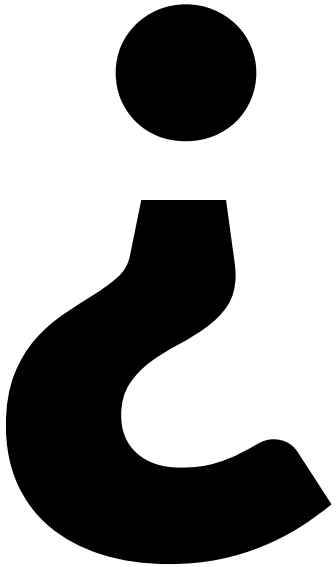
ArregloDinamico III

el regreso genérico

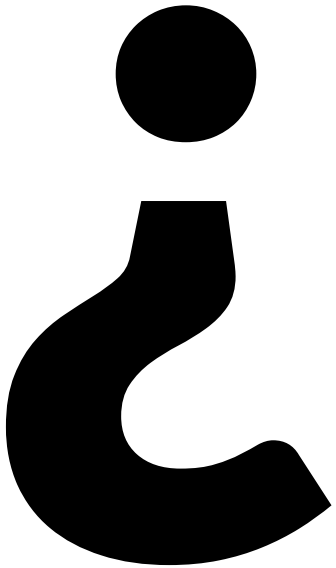
Algunos métodos más importantes

```
public class ArregloDinamico {
    private int[] arreglo;
    private int largo;

    public void modificar(int posicion, int valor);
    public void insertar(int posicion, int valor);
    public int extraer(int posicion);
    public int obtener(int posicion);
}
```



Pero si queremos guardar `float`'s

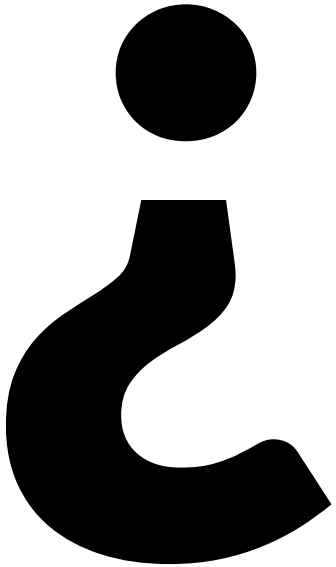


¿Pero qué pasa con el código?

```
public class ArregloDinamico {
    private float[] arreglo;
    private int largo;

    public void modificar(int posicion, float valor);
    public void insertar(int posicion, float valor);
    public float extraer(int posicion);
    public float obtener(int posicion);
}
```

Porque mucho no cambia...



Lo duplicamos

1. Cambiamos a algo mas genérico como base

```
public class ArregloDinamicoObjetos {
    private Object[] arreglo;
    private int largo;

    public void modificar(int posicion, Object valor);
    public void insertar(int posicion, Object valor);
    public Object extraer(int posicion);
    public Object obtener(int posicion);
}
```

Porque mucho no cambia...

La entrada es directa

```
ArregloDinamicoObjetos arreglo = new ArregloDinamicoObjetos(10);
Integer numero = 10;

arreglo.insertar(numero, 0);

Integer salida = (Integer)arreglo.obtener(0);
```

La salida requiere una conversión

Para bien y para mal

```
ArregloDinamicoObjetos arreglo = new ArregloDinamicoObjetos(10);
Integer numero = 10;

Auto movil = new Auto(“Nissan”);

arreglo.insertar(movil, 0);

Integer salida = (Integer)arreglo.obtener(0);
```

¡Esto puede fallar!

Podemos insertar de todo

La herencia puede *ayudar*

Pero, ¿esto funciona?

```
public class ArregloDeInteger{
    public void modificar(int posicion, Integer valor);
    public void insertar(int posicion, Integer valor);
    public Integer extraer(int posicion);
    public Integer obtener(int posicion);
}
```

¿La sobrecarga se puede aplicar a todos los métodos?

No con todos

```
public class ArregloDeInteger extends ArregloDinamico{
    public void modificar(int posicion, Integer valor);
    public void insertar(int posicion, Integer valor);
    public Integer extraer(int posicion);
    public Integer obtener(int posicion);
}
```

El tipo de retorno, no es tenido en cuenta para la sobrecarga...

No nos resuelve *el problema...*

Genéricos

Polimorfismo paramétrico

¡Versión generica!

```
public class ArregloDinamico<T> {
    private T[] arreglo;
    private int largo;

    public void insertar(int posicion, T valor);
    public T extraer(int posicion);
    public T obtener(int posicion);
}
```

¡Ahora somos libres!

```
ArregloDinamico<Integer> enteros = new ArregloDinamico<>();
ArregloDinamico<Auto> enteros = new ArregloDinamico<>();
ArregloDinamico<Producto> enteros = new ArregloDinamico<>();
```


¿Preguntas?

Interfaces

Nos indica comportamiento a implementar

```
public interface Implementable {
    public int unMetodoAImplementar(int argumento);
    public int otroMetodoAImplementar(String coso);
}
```

Pero también le da un ‘tipo’ a quien lo implementa.

{ Su nombre es un
‘able’
En CamelCase }

Podemos usar interfaces para comparar cosas

```
public interface Comparable{
    int compararCon(Object otro);
}
```

De manera genérica

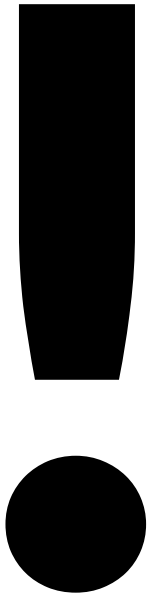
Pero con cuidado porque el cast puede fallar

```
public Auto implements Comparable{
    …
    public int compararCon(Object otro){
        Auto aComparar = (Auto)otro;
        return this.numeroSerie > otro.numeroSerie;
    }
}
```

Falta un `if` para evitar el lanzamiento de la excepción


¿Preguntas?

pero



Combinado con tipos genéricos

```
public interface Comparable<T>{
    int compararCon(T otro);
}
```

¡Es mucho mas facil!

No hay cast que pueda fallar

```
public Auto implements Comparable<Auto>{
    …
    public int compararCon(Auto otro){
        return this.numeroSerie > otro.numeroSerie;
    }
}
```

¡Porque ya es un Auto!


¿Preguntas?

¿y si queremos implementar?

```
public void ordenarMayorMenor();
```

Podemos pedir que `ArregloDinamico` solo acepte `Comparables`

Podemos agregarlo al arreglo, con un extra.

```
public class ArregloDeInteger<T extends Comparable>{
    public void modificar(int posicion, T valor);
    public void insertar(int posicion, T valor);
    public T extraer(int posicion);
    public T obtener(int posicion);
    public void ordenarMayorMenor(){
        Comparable uno = arreglo[0];
        Comparable dos = arreglo[1];
        int orden = dos.compararCon(uno);
    }
}
```

Indicando que sea lo que sea que guardemos, tiene que ser `Comparable`

Podemos agregarlo al arreglo, con un extra.

```
public class ArregloDeInteger<T extends Comparable<T>>{
    public void modificar(int posicion, T valor);
    public void insertar(int posicion, T valor);
    public T extraer(int posicion);
    public T obtener(int posicion);
    public void ordenarMayorMenor(){
        T uno = arreglo[0];
        T dos = arreglo[1];
        int orden = dos.compararCon(uno);
    }
}
```

Indicando que sea lo que sea que guardemos, tiene que ser `Comparable`

{ Pero también,
que sea del mismo
Tipo }


¿Preguntas?

Una cosa más, esto existe en el JDK

```
package java.lang;

public interface Comparable<T>{
    int compareTo(T otro);
}
```

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Comparable.html

TP10

Transformar ArregloDinamico en ArregloGenerico

https://classroom.github.com/a/tipiFTSQ


¿Preguntas?

unrn.edu.ar

UNRN Universidad Nacional de Río Negro

| unrionegro