

# Metodologías de Programación I

---

## Práctica 1.

### Repaso de POO. Interfaces

#### Ejercicio 1

Implemente la siguiente interface que representa cualquier objeto que sabe compararse con otro objeto de su “mismo tipo”:

Comparable

```
    sosIgual(Comparable) ← Devuelve verdadero si el objeto
                           que recibe el mensaje es el mismo que
                           el “comparable” recibido por parámetro,
                           devuelve falso en caso contrario
    sosMenor(Comparable) ← Devuelve verdadero si el objeto
                           que recibe el mensaje es más chico que
                           el “comparable” recibido por parámetro,
                           devuelve falso en caso contrario
    sosMayor(Comparable) ← Devuelve verdadero si el objeto
                           que recibe el mensaje es más grande que
                           el “comparable” recibido por parámetro,
                           devuelve falso en caso contrario
```

Nota: asumiremos que el elemento que recibe el mensaje y el objeto recibido por parámetro son del “mismo tipo”.

#### Ejercicio 2

Implemente la clase *Numero*:

Numero

```
    valor                ← Es una variable que almacena un
                           número entero
    constructor(v)        ← Es el constructor de la clase que
                           recibe un valor “v” y lo almacena en la
                           variable “valor”
    getValor              ← Devuelve la variable “valor”
```

Haga que la clase *Numero* implemente la interface *Comparable*.

**Nota:** Si se implementa en C#, en los tres métodos a implementar, se deberá castear el parámetro recibido a *Numero*.

### Ejercicio 3

Implemente la siguiente interface que representa objetos que almacenan comparables:

```
Coleccionable
    cuantos          ← Devuelve la cantidad de elementos
                     comparables que tiene el coleccionable
    minimo           ← Devuelve el elemento de menor valor
                     de la colección
    maximo           ← Devuelve el elemento de mayor valor
                     de la colección
    agregar(Comparable) ← Agrega el comparable recibido por
                           parámetro a la colección que recibe el
                           mensaje
    contiene(Comparable) ← Devuelve verdadero si el
                           comparable recibido por parámetro está
                           incluido en la colección y falso en
                           caso contrario
```

### Ejercicio 4

Implemente las clases *Pila* y *Cola* (vistas en Algoritmos y Programación) y haga que las dos clases implementen la interface *Coleccionable*.

### Ejercicio 5

Implemente una función *llenar* que reciba un *Coleccionable* y que le agregue 20 comparables elegidos al azar.

```
llenar(Coleccionable)
    repetir 20 veces
        comparable = new Numero(valor_elegido_al_azar)
        coleccionable.agregar(comparable)
```

### Ejercicio 6

Implemente una función *informar* que reciba un *Coleccionable* e imprima por consola la cantidad de elementos que tiene el coleccionable recibido por parámetro, el elemento mínimo, el máximo y si contiene, o no, como elemento un valor leído por teclado.

```
informar(Coleccionable)
    imprimir (coleccionable.cuantos())
    imprimir (coleccionable.minimo())
    imprimir (coleccionable.maximo())
    comparable = new Numero(leer_por_teclado)
    si (coleccionable.contiene(comparable))
        imprimir("El elemento leído está en la colección")
    sino
        imprimir("El elemento leído no está en la colección")
```

## Ejercicio 7

Implemente un módulo *main* que cree una *Pila* y una *Cola*, que las llene y que invoque a la función *informar* con la pila y cola creadas.

```
main
    pila = new Pila()
    cola = new Cola()
    llenar(pila)
    llenar(cola)
    informar(pila)
    informar(cola)
```

## Ejercicio 8

Cree la clase *ColeccionMultiple*:

```
ColeccionMultiple
    pila                ← Es una variable que almacena una pila
    cola                ← Es una variable que almacena una cola
    constructor(p, c)  ← Es el constructor de la clase que
                        recibe un pila "p" y una cola "c" y las
                        almacena en las variables
                        correspondientes
```

Haga que la clase *ColeccionMultiple* implemente la interface *Coleccionable*, donde:

- `cuantos`      ← Devuelve la cantidad de elementos de ambos coleccionables
- `minimo`       ← Devuelve el elemento de menor valor entre ambos coleccionables
- `maximo`       ← Devuelve el elemento de mayor valor entre ambos coleccionables
- `agregar`      ← no hace nada
- `contiene`     ← Devuelve verdadero si el comparable recibido por parámetro está incluido en alguna de las dos colecciones o en ambas y falso en caso contrario

## Ejercicio 9

Modifique el módulo *main* para crear una *ColeccionMultiple* e informe con esta colección.

```
main
    pila = new Pila()
    cola = new Cola()
    multiple = new ColeccionMultiple(pila, cola)
    llenar(pila)
    llenar(cola)
    informar(pila)
    informar(cola)
    informar(multiple)
```

## Ejercicio 10

*Para reflexionar.* Además de la creación de la nueva clase *ColeccionMultiple* y la adaptación del módulo *main*, responda ¿qué tuvo que modificar de lo realizado en los primeros seis ejercicios?

## Ejercicio 11

Implemente la clase *Persona*:

|                   |   |
|-------------------|---|
| Persona           |   |
| nombre            | ← Es una variable que almacena un string  |
| dni               | ← Es una variable que almacena un numero  |
| constructor(n, d) | ← Es el constructor de la clase que recibe un nombre "n" y un DNI "d" y los almacena en las variables correspondientes. |
| getNombre         | ← Devuelve la variable nombre   |
| getDNI            | ← Devuelve la variable dni  |

Haga que la clase *Persona* implemente la interface *Comparable*. Compare las personas por *dni* o por *nombre*, según prefiera.

## Ejercicio 12

Implemente una función *llenarPersonas* que reciba un *Coleccionable* y que le agregue 20 personas elegidos al azar.

```
llenarPersonas(coleccionable)
    repetir 20 veces
        comparable = new Persona(nombre_al_azar, dni_al_azar)
        coleccionable.agregar(comparable)
```

### Ejercicio 13

Modifique el módulo *main* para crear una *ColeccionMultiple* de personas e informe con esta colección.

```
main
    pila = new Pila()
    cola = new Cola()
    multiple = new ColeccionMultiple(pila, cola)
    llenarPersonas(pila)
    llenarPersonas(cola)
    informar(multiple)
```

### Ejercicio 14

*Para reflexionar.* Además de la creación de la nueva clase *Persona*, la creación de la función *llenarPersonas* y la adaptación del módulo *main*, responda ¿qué tuvo que modificar de lo realizado en los ejercicios 1 a 6 y el 8?

### Ejercicio 15

Implemente la clase *Alumno* que sea subclase de *Persona*:

|                         |   |
|-------------------------|---|
| Alumno → Persona        |   |
| legajo                  | ← Es una variable que almacena un número  |
| promedio                | ← Es una variable que almacena un número  |
| constructor(n, d, l, p) | ← Es el constructor de la clase que recibe un nombre "n", un DNI "d", un legajo "l" y un promedio "p" y los almacena en las variables correspondientes. |
| getLegajo               | ← Devuelve la variable legajo   |
| getPromedio             | ← Devuelve la variable promedio   |

### Ejercicio 16

Implemente una función *llenaAlumnos* que reciba un *Coleccionable* y que le agregue 20 alumnos elegidos al azar.

```
llenarAlumnos(Coleccionable)
    repetir 20 veces
        comparable = new Alumno(nombre_al_azar, dni_al_azar,
                                legajo_al_azar, promedio_al_azar)
        coleccionable.agregar(comparable)
```

## Ejercicio 17

Modifique el módulo *main*:

```
main
    pila = new Pila()
    cola = new Cola()
    multiple = new ColeccionMultiple(pila, cola)
    llenarAlumnos(pila)
    llenarAlumnos(cola)
    informar(multiple)
```

Responda ¿Funcionó? ¿Fue necesario decir explícitamente que *Alumno* tiene que implementar la interface *Comparable*? ¿Cuál fue el criterio por el cual se comparó a los alumnos? ¿Por qué?

## Ejercicio 18

Reimplemente los métodos de comparable en *Alumno* para que se compare por legajo o promedio, a elección.

Ejecute el módulo *main* del ejercicio anterior para comprobar su correcto funcionamiento.

*Este ejercicio, y todos los anteriores que dependen de éste, debe ser entregado en el aula virtual del campus.*

## Ejercicio 19

*Para reflexionar.* Responda ¿Podría haber hecho esto mismo sin interfaces? ¿A qué costo?