

# Metodologías de Programación I

---

## Práctica 3.

### Patrones Factory Method y Observer

#### Ejercicio 1

Recordamos: ¿Cuál es la única diferencia entre los métodos *llenar*, *llenarPersonas* y *llenarAlumnos* implementados en la práctica 1 (ejercicios 5, 12 y 16)? ¿Qué tuvo que hacer con el método *informar* (ejercicio 6) a medida que ejecutaba los métodos *main* (ejercicios 13 y 17)? ¿Qué sucedería con todos estos métodos si apareciera una nueva clase *Profesor*, la cual se desea comparar por antigüedad?

#### Ejercicio 2

Implemente la clase *GeneradorDeDatosAleatorios*.

```
GeneradorDeDatosAleatorios
    numeroAleatorio(max) ← Devuelve un número aleatorio entre
                           0 y 'max'
    stringAleatorio(cant) ← Devuelve un string aleatorio de
                           'cant' caracteres
```

#### Ejercicio 3

Implemente la clase *LectorDeDatos*.

```
LectorDeDatos
    numeroPorTeclado() ← devuelve un número leído por teclado
    stringPorTeclado() ← devuelve un string leído por teclado
```

#### Ejercicio 4

Implemente la interface *FabricaDeComparables*.

```
FabricaDeComparables
    crearAleatorio(opcion) ← Devuelve un Comparable
                           generado aleatoriamente
    crearPorTeclado(opcion) ← Devuelve un comparable donde
                           los datos se ingresan por teclado
```

#### Ejercicio 5

Implemente con el patrón Factory Method la capacidad de crear instancias de comparables (sólo las clases *Numero* y *Alumno*). Implemente las fábricas concretas *FabricaDeNumeros* y *FabricaDeAlumnos*.

## Ejercicio 6

Implemente una única función *llenar* (práctica 1, ejercicios 5 y 16) y una única función *informar* (práctica 1, ejercicios 6 y 17) que reciban una opción por parámetro que indique que comparable instanciar.

```
llenar(coleccionable, opcion)
    repetir 20 veces
        comparable = fabrica.crearAleatorio(opcion)
        coleccionable.agregar(comparable)

informar(coleccionable, opcion)
    imprimir (coleccionable.cuantos())
    imprimir (coleccionable.minimo())
    imprimir (coleccionable.maximo())
    comparable = fabrica.crearPorTeclado(opcion)
    si (coleccionable.contiene(comparable))
        imprimir("El elemento leído está en la colección")
    sino
        imprimir("El elemento leído no está en la colección")
```

Adapte, modifique y compruebe el correcto funcionamiento de los métodos *main* de los ejercicios 9 y 17 de la práctica 1. Unifique ambos métodos en un único *main*.

## Ejercicio 7

*Para reflexionar.* ¿Qué debería hacer si se quiere tener en el método *main* la opción de almacenar los comparables en una pila, en una cola, en una colección múltiple, en un conjunto o en un diccionario?

Opcional. Implemente la solución propuesta.

## Ejercicio 8

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

Profesor → Persona	
antigüedad	← Es una variable que almacena un número (años de antigüedad)
constructor(n, d, a)	← Es el constructor de la clase que recibe un nombre "n", un DNI "d", una antigüedad "a" y los almacena en las variables correspondientes.
hablarALaClase()	← Simula el dictado de una clase. Este método debería imprimir por consola la frase "Hablando de algún tema".
escribirEnElPizarron()	← Simula al profesor escribiendo algo en el pizarrón. Este método debería imprimir por consola la frase "Escribiendo en el pizarrón".

## Ejercicio 9

Implemente una fábrica concreta para la clase *Profesor* y compruebe el correcto funcionamiento del método *main* del ejercicio 6. Compare a los profesores por el campo *antigüedad*.

## Ejercicio 10

*Para reflexionar.* ¿Qué tienen en común las fábricas de la clase *Profesor* y de la clase *Alumno*? ¿Podría ampliarse la jerarquía de clases de las fábricas? ¿Cómo?

Opcional. Implemente la solución propuesta.

## Ejercicio 11

Agregue a la clase *Alumno* los siguientes métodos:

Alumno

<code>prestarAtencion()</code>	← Simula al alumno escuchando lo que dice el profesor. Este método debería imprimir por consola la frase "Prestando atención".
<code>distraerse()</code>	← Simula al alumno distrayéndose de la clase. Este método debería imprimir por consola alguna de estas frases elegidas al azar: "Mirando el celular", "Dibujando en el margen de la carpeta", "Tirando aviones de papel".

## Ejercicio 12

Implemente el patrón Observer haciendo que el profesor sea el observable y los alumnos los observadores del profesor. Los alumnos deberían prestar atención cada vez que el profesor les habla y distraerse cada vez que éste escribe en el pizarrón.

## Ejercicio 13

Implemente la función *dictadoDeClases*.

```
dictadoDeClases(profesor)
    repetir 5 veces
        profesor.hablarALaClase()
        profesor.escribirEnElPizarron()
```

## Ejercicio 14

Implemente la siguiente función *main*.

```
main
    profesor = new Profesor()
    crear 20 alumnos y hacer que todos sean observadores del
        profesor
    dictadoDeClases(profesor)
```

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

## Ejercicio 15

Opcional. Cree la clase *AlumnoFavorito* como subclase de *Alumno* sobre escribiendo el método:

<code>distraerse()</code>	← Este método debería imprimir por consola la frase: "Yo nunca me distraigo, siempre presto atención".
---------------------------	--

Agregue a la clase *Profesor* el método

<code>hacerSilencio()</code>	← Este método debería imprimir por consola la frase: "Silencio, no se distraigan".
------------------------------	--

Haga que *AlumnoFavorito* sea observador de todos sus compañeros y que al momento de que alguno de ellos tire un avión de papel le avise al profesor que sus compañeros no están prestando atención. Este aviso también debería ser implementado con *Observer*, es decir el profesor debe ser observador del *AlumnoFavorito*.

Modifique el ejercicio 14 para crear, además de los 20 alumnos, un *AlumnoFavorito* y comprobar el correcto funcionamiento del sistema.

## Ejercicio 16

Opcional. Intercambie las clases implementadas en esta práctica con otro compañero para probar si funcionan clases "externas" en el sistema desarrollado por uno mismo.