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 *Vendedor* la cual se desea comparar por cantidad de ventas realizadas?

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()   ← Devuelve un Comparable generado
                          aleatoriamente
     crearPorTeclado()  ← 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 fabricas 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 (colecionable.cuantos())
      imprimir (colecionable.minimo())
      imprimir (colecionable.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 *Vendedor* que sea subclase de *Persona*:

```
Vendedor → Persona
      sueldoBasico          ← Es una variable que almacena un
                              número
      bonus                 ← Es una variable que almacena un
                              número
      constructor(n, d, s)  ← Es el constructor de la clase que
                              recibe un nombre "n", un DNI "d", un
                              sueldo básico "s" y los almacena en las
                              variables correspondientes. La variable
                              de instancia bonus se inicializa en 1.
      venta(monto)          ← Cada vez que un vendedor concreta una
                              venta, se invoca este método y se
                              recibe el 'monto' de la venta por
                              parámetro. Por el momento, solo debería
                              imprimir el 'monto' por consola.
      aumentaBonus()        ← Suma en 0.1 el valor del 'bonus'.
```

Ejercicio 9

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

Ejercicio 10

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

Opcional. Implemente la solución propuesta.

Ejercicio 11

Implemente la clase *Gerente*. Un *Gerente* tiene una colección con sus vendedores.

```
Gerente
      mejores                ← Es un Coleccionable con los
                             mejores vendedores

      cerrar()               ← se ejecuta al final del día.
                             Debe informar por consola el nombre
                             y apellido de los 'mejores'
                             vendedores de la jornada, junto con
                             su bonus acumulado.

      venta(monto, vendedor) ← Recibe el 'monto' facturado por
                             una venta y el 'vendedor' que la
                             realizó. Si el 'monto' es superior
                             a 5000 entonces se hacen dos cosas:
                             a) se agrega el 'vendedor' a la
                             colección 'mejores' si es que no
                             estaba en la colección; b) se
                             invoca al método aumentaBonus del
                             vendedor.
```

Ejercicio 12

Implemente el patrón Observer haciendo que los vendedores sean los observables y el gerente el observador de todos los vendedores.

Ejercicio 13

Implemente la función *jornadaDeVentas*.

```
jornadaDeVentas(coleccionable_vendedores)
     repetir 20 veces
          para cada 'vendedor' de coleccionable_vendedores
               monto = numero_al_azar_entre(1, 7000)
               vendedor.venta(monto)
```

Ejercicio 14

Implemente la siguiente función *main*.

```
main
      coleccion = new Pila()      ← ó Cola ó Conjunto ó Dicc.
      llenar(coleccion, opción_para_crear_vendedores)
      gerente = new Gerente()
      hacer que gerente sea observador de todos los vendedores
      jornadaDeVentas(colección)
      gerente.cerrar()
```

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 *VendedorPauperrimo* como subclase de *Vendedor* agregando los siguientes métodos:

```
robar()
descansar()
molestarALosCompañeros()
```

Los tres métodos deben imprimir por consola un texto acorde a la acción que está realizando el vendedor.

Implemente las clases *Seguridad*, *Cliente* y *Encargado*, las tres con un método *reaccionar* que impriman un mensaje distinto por consola.

Haga que *Seguridad*, *Cliente* y *Encargado* sean observadores del vendedor paupérrimo.

El vendedor paupérrimo después de concretar una venta hace lo siguiente:

- Si el monto de la venta fue menor a 500, roba un artículo y sólo el de seguridad debería llamarle la atención.
- Si el monto de la venta fue mayor a 4000, descansa y solo el cliente va a hacerle una consulta.
- Si el monto de la venta está entre 500 y 4000, molesta a sus compañeros y solo el encargado debería llamarle la atención.

Modifique el método *main* del ejercicio 15 para agregar un *VendedorPauperrimo* a la colección. Ejecute el método main y compruebe el correcto funcionamiento del módulo.

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.