Objetivos:

- Factorizar código a través de métodos template ubicados en clases abstractas.
- Buenas prácticas de programación
- Introducción al Diseño con Patrones

Ejercicio 1.
Investigue en internet o en la bibliografía en qué consiste la ***Refactorización de código***. Brinde ejemplos de la misma.

Ejercicio 2.
Una **empresa** de seguros para autos vende **pólizas** de seguros para autos de cualquier tipo. Las pólizas que vende la **aseguradora** pueden ser de dos tipos, pólizas con cobertura **parcial** y pólizas con coberturas **total**. Cada póliza contiene los siguientes datos, nombre del asegurado, fecha de alta y fecha de validez de la póliza (es decir, la fecha de cuando caduca la misma, la cual es de un año a partir de la fecha de alta) y el **auto** en cuestión asegurado.
Del auto se conoce la marca, el modelo y la valuación (es decir, el valor que le asigna la aseguradora). El valor de cada una de las pólizas (el total de lo que el asegurado debe pagar) es diferente en cada caso y se calcula de la siguiente manera:

- Para las pólizas totales el valor es de 180 pesos más el 10% del valor del auto. P.e. si tenemos un auto que vale 20.000 pesos, el valor total de la póliza es de 2180 pesos.
- En cambio, para las pólizas parciales el valor es de 100 pesos más el 7% del valor del auto.

Realice el diagrama de clases en UML e implemente en JAVA-LIKE al menos los siguientes mensajes:

a) montoAPagarDe(Poliza unaPoliza)
"Retorna el monto que debe pagar el auto según la póliza"

b) montoTotalAsegurado()
"Retorna la suma de los valores de valuación"

c) montoTotalACobrar()
"Retorna el monto que la aseguradora debe cobrar según las pólizas que tiene vendidas"

d) asegurarAutoTotal(Auto unAuto, String unNombre)
"Asegura un auto con una póliza completa"

e) cantidadDePolizas()

"retorna la cantidad de pólizas vendidas"

Ejercicio 3.

Implemente en JAVA-LIKE el ejercicio 8 (Navegador) del TP3*:" Se desea modelar un Navegador de Internet o más comúnmente llamado Browser, símil al Chrome, Edge, Firefox, etc. El Browser es sencillo, solamente…"*

Ejercicio 4.

Una **estación de servicio** está compuesta por varios surtidores de combustible. Los surtidores pueden ser de Gasoil, Euro Diesel, Nafta Súper, Nafta Común, o Nafta Premium.

Los surtidores conocen el precio básico, por litro, del combustible que se expende con el mismo. El precio puede ir variando con lo cual no es fijo.
Además del valor por litro cada combustible tiene un recargo por impuestos según la siguiente tabla:

Combustible	Recargo
Gasoil	5%
Euro Diesel	10%
Nafta Súper	7%
Nafta Común	0%
Nafta Premium	15%

Cada surtidor está conectado a un tanque el cual contiene el combustible. Cada tanque tiene una capacidad máxima y además sabe la capacidad actual del mismo. Una vez que un surtidor se conecta a ese tanque no se puede volver a cambiar. Cada tanque tiene un número de serie y dado que hay que reemplazarlos cada cierto tiempo, lo tanques tienen una fecha de puesta en funcionamiento.

a) Modele en UML el diagrama de clases
b) Implemente en JAVA-LIKE todas las clases y métodos que se piden a continuación. Como así todos los métodos necesarios que crea convenientes:

I. **capacidadDisponible()** *// El tanque retorna la capacidad disponible. Es decir, lo que falta para que esté lleno*.
II. **montoACobrar(int litros)** *// El surtidor retorna el precio a cobrar según los litros que se pasan como parámetro. Se deben tener en cuenta los recargos.*
III. **capitalActual()** *// Es el valor retornado por la estación y se trata de la suma del contenido de todos los tanques si los recargos.*
IV. **tanquesConConbustibleMenorA(int cantidad)** *// La estación retorna una colección de tanques donde la capacidad actual es menor o igual al valor pasado como parámetro*.

V. **incrementarValorCombustible(int porcentaje)** *// La estación incrementa el valor de cada combustible según el porcentaje que se pasa como parámetro*

Ejercicio 5. (SUBIR a https://plataformaed.unnoba.edu.ar)
Se pide modelar una empresa de alquiler de vehículos. Los vehículos que alquila esta empresa pueden ser autos o camiones de transporte.
De la empresa se conoce su nombre, su dirección y sus clientes. De cada cliente se sabe su nombre, dirección y CUIT.

La empresa registra cada alquiler que realiza a sus clientes. Los alquileres pueden ser alquileres simples o de flota, es decir, se alquilan más de un vehículo al mismo cliente.

De cada alquiler se conoce el vehículo alquilado en el caso de ser un alquiler simple o la flota de vehículos en el caso de ser un alquiler de una flota, además se conoce el cliente, y la fecha.

De cada vehículo se conoce su marca, modelo, el precio del vehículo y el precio de alquiler del mismo. Además, en el caso de los autos se tiene un monto extra que corresponde al monto del seguro.

Además de tener un valor diferente por cada vehículo, el valor de alquiler depende de la siguiente tabla:

Tipo Alquiler	Valor de Alquiler	Detalle
Autos	Costo + Seguro	
Camiones	Costo + Seguro del 5%	El 5% se calcula sobre el valor del camión.
Flota	Suma de todos los vehículos – 5% de descuento por alquilar una flota	Es decir que se deben tener en cuenta los valores de alquiler y luego aplicar el descuento

Se pide que:
1. Realice el diagrama de clases en UML
2. Desarrolle en JAVA-like todos las clases y métodos necesarios teniendo en cuenta que el sistema debe tener como mínimo la siguiente funcionalidad:

a) public void registrarAlquiler(Cliente cliente, Vehículo vehiculo); //Registra en el sistema el nuevo alquiler de dicho vehiculo.
b) public void registrarAlquiler(Cliente cliente, Vector<Vehículo> vehiculos); //Registra en el sistema el nuevo alquiler de dichos vehiculos.

c) public float montoAlquilerDe(Alquiler alquiler) //Retorna el monto del alquiler que se pasa como parámetro.
d) public float montoTotalAlquileres() //Retorna la suma de todos los alquileres del sistema.
e) public float montoTotalVehiculos() //Retorna la suma de todos los valores de los vehículos alquilados.
f) public int totalVehiculosAlquilados() //Retorna el total de vehículos alquilados

Ejercicio 6.

Elegir un buen nombre para un objeto, un método o una variable es un problema común en todos los lenguajes de programación.
Un buen nombre debe ser subjetivo y depende del proyecto y del lenguaje de programación. En todo lenguaje, debemos respetar ciertas reglas de convivencia ya establecidas.
Supongamos que tenemos que elegir correctos nombres para algunos métodos, variables y clases, teniendo en cuenta las recomendaciones vistas en la teoría y en el libro *Smalltalk with Style*[1] que aplican a JAVA-LIKE.

a) Seleccione el nombre que le daría a un método que retorna el número de teléfono de una agenda de teléfonos.

- □ phoneNumber()
- □ number
- □ PhoneNumber()
- □ Number

b) Indique cual sería el mensaje para recuperar una palabra que se encuentra en una colección.

- □ peekWord()
- □ word()
- □ getWord()

c) Supongamos que tenemos un objeto que representa expresiones faciales. Como le indicaría que se muestre sorprendido

- □ Surprised()
- □ lookSurpriced ()

d) Cuales serias los métodos correctos para testear si:

i. Un objeto es un string
- □ isString()
- □ string ()

ii. Una persona esa enojada

[1] **Suzanne Skublics, Edward Klimas** *and* **David A. Thomas.** ISBN: 0-13-165549-3, Prentice Hall

- □ Hungry()
- □ isHungry()

iii. Un auto tiene cuatro ruedas.
- □ hasFourWheels()
- □ fourWheels()

e) Indique cuales de los siguientes getters y setters son correctos si tengo un objeto que contiene una colección de libros

- □ books()
- □ getBooks()
- □ books(Vector<Book> aBooksCollection)
- □ setBooks(Vector<Book> aBooksCollection)

f) ¿Cuáles de las siguientes abreviaciones son correctas?

- □ receivedTime()
- □ rcvdTime()
- □ rTime()

- □ animationsState()
- □ animSt()

Patrones de Diseño

Ejercicio 7.
Supongamos que tenemos que diseñar un sistema en donde se simula el **FileSystem** de un Sistema Operativo. En dicho filesystem podemos tener archivos de cualquier tipo y carpetas.
Los archivos tienen un nombre y un peso. Las carpetas tienen un nombre y dentro de las carpetas podemos tener o archivos u otras carpetas, es decir que se puede tener una estructura recursiva.
El peso de las carpetas está dado por la suma del contenido de las mismas.

1. Modele en UML.
2. ¿Qué patrón usaría?
3. Describa brevemente el patrón seleccionado
4. Implemente en JAVA-LIKE al menos la siguiente funcionalidad:
    a. add(Folder f)
    b. add(File f)
    c. remove(Folder f)
    d. remove(File f)
    e. size()

Ejercicio 8.
Se desea desarrollar un cliente de mensajería instantánea similar al gTalk, ICQ, MSN, ¡Y! Messenger.

- El cliente envía mensajes solamente de texto.
- Un cliente puede estar online/offline.
    - En el caso de estar online puede enviar y recibir mensajes.
    - En el caso de estar offline no puede ni enviar ni recibir mensajes.
- Se desea guardar un historial de los mensajes enviados y recibidos.

1. Modele en UML el Mensajero.
2. Cómo modela la condición de online/offline

Se desea ampliar la funcionalidad del mensajero anterior.
Ahora un cliente puede estar **online/offline/away.**

- En el caso de estar **online** puede enviar y recibir mensajes.
- En el caso de estar **offline** no puede ni enviar ni recibir mensajes.
- En el caso de estar **away** cada vez que recibe un mensaje responde al mismo con un mensaje automático diciendo: *"Este es un mensaje automático: el cliente no está disponible momentáneamente"*

3. ¿Cómo extiende el Mensajero anterior?

4. ¿Qué patrón seleccionó?
5. Diseñe la solución final en UML

Ejercicio 9. (**SUBIR a https://plataformaed.unnoba.edu.ar**)
Un sistema de alarmas está compuesto por una central y varios sensores. Los sensores pueden ser sensores de movimiento o de contacto (comúnmente los que se instalan en las puertas y ventanas).
Los sensores pueden estar en tres estados posibles:

- **Apagado**: el sensor está fuera de uso y no registra cambios
- **Esperando**: el sensor está esperando algún cambio
- **Alerta**: se produjo una alerta. (p.e. un sensor de movimiento detectó un movimiento)

Cada vez que los sensores registran, o pasan al estado de alerta, deben informar dicho suceso a la central. Los mismos quedan en dicho estado hasta que la central indique lo contrario.

Cuando la central se activa, ésta debe activar cada uno de los sensores y ponerlos en estado *Esperando.* Cuando la central se desactiva, esta debe activar cada uno de los sensores y ponerlos en estado *Apagado.* Asimismo, la central puede pasar un sensor (o varios) del estado de *Alerta* al estado de *Esperando.*

Cada vez que se produce una alerta por medio sensor (no importa cual), la central registra ese cambio en un historial. Tenga en cuenta cuáles serían los datos necesarios para determinar cuál sensor fue activado, en qué fecha, etc.

1. Realice en UML un diagrama de clases de la solución.
2. Indique qué patrón o patrones utilizo y donde.
3. Implemente en JAVA-Like.

Nota: tenga en cuenta que se pueden agregar nuevos sensores de diferentes tipos a la central y esto no debería afectar el funcionamiento de la misma.

Nota2: se debe **implementar el patrón completo**, si bien Java ya soporta parte del mecanismo de la implementación del Observer, en esta materia aún no se poseen los conocimientos para aplicar dicha implementación.