Herencia

Se pide resolver cada uno de los ejercicios utilizando diagramas de clases UML y el lenguaje de programación Java.

[G.01] - El viajar es un placer.

Crear un proyecto de biblioteca de clases con las clases Automovil, Moto, Camion.

1. Crear un enumerado Colores { ROJO, BLANCO, AZUL, GRIS, NEGRO}
2. Camion tendrá los atributos: cantidadRuedas : short, cantidadPuertas : short, color : Colores, cantidadMarchas : short, pesoCarga : int.
3. Automovil tendrá los atributos: cantidadRuedas : short, cantidadPuertas : short, color : Colores, cantidadMarchas : short, cantidadPasajeros : int.
4. Moto tendrá los atributos: cantidadRuedas : short, cantidadPuertas : short, color : Colores, cilindrada : short.
5. Crear, en cada clase, un constructor que reciba todos sus atributos.
6. Crear la clase VehiculoTerrestre y abstraer la información necesaria de las clases anteriores. Luego generar una relación de herencia entre ellas, según corresponda.
7. VehiculoTerrestre tendrá un constructor que recibirá todos sus atributos. Modificar las clases que heredan de ésta para que lo reutilicen.

Una vez diseñado el diagrama de clases UML, crear, implementar y probar la jerarquía de clases en un proyecto de consola.

[G.02] - ¡Cuántos archivos!

En toda computadora existen archivos, que constan de nombre, peso, localización y si están o no abiertos. Las operaciones comunes a todos los archivos tienen que ver con poder abrirse, cerrarse y moverse de ubicación.
En particular, los archivos multimedia cuentan con una duración en segundos y un indicador sobre si están o no en reproducción. Estos cuentan con capacidades para reproducirse y pararse.
Existen tipos especiales de archivos multimedia: Los archivos de audio y los archivos de video.
Un archivo de audio tiene formato (mp3, wav, entre otros), artista y álbum mientras que un archivo de video tiene formato (mp4, mkv, entre otros), alto y ancho (en pixeles).
Por último, se cuenta con archivos de texto, de los que se conoce su codificación y tienen la capacidad de cifrar su contenido.
Basado en el enunciado descrito, se pide realizar el diagrama de clases que lo modelice, con sus relaciones, atributos y métodos.
Una vez diseñado el diagrama de clases UML, crear, implementar (en un proyecto de tipo biblioteca de clases) y probar la jerarquía de clases en un proyecto de consola.

[G.03] - ¡Cuántos archivos! (2)

Aprovechando el modelo generado en el ejercicio anterior, se pide agregar los siguientes métodos a la clase Computadora, que operen sobre su colección de archivos:

- El método **cerrarTodos** que cierre todos los archivos que estuviesen abiertos.
- El método **cantArchivosDeTexto** que retorne la cantidad de archivos de texto existentes.
- El método **cifrarArchivos** que cifre todos los archivos cuya codificación sea "UTF-8".
- El método **duracionPromedio** que debe devolver la duración en segundos promedio de todos los archivos de audio y video.
- El método **videosFullHD** que debe devolver una lista de videos cuya resolución sea Full HD (1920x1080).

Una vez actualizado el diagrama de clases UML, implementar y probar la jerarquía de clases.

[G.04] - Bicicletería

Un taller de bicicletas desea un pequeño prototipo para evaluar cuántos servicios podrá realizar.
Se sabe que cada bicicleta tendrá una marca, modelo y cantidad de kilómetros. A su vez, puede tomar también bicicletas eléctricas, de las cuales se sabe además su potencia en watts.
El taller sólo presta servicios a aquellas bicicletas con menos de 2000 kms. En el caso de las bicicletas eléctricas, solo se tomarán aquellas de 250w de potencia o menos (sin importar los km).

Basado en el enunciado descrito anteriormente, se pide realizar:

- El diagrama de clases que lo modelice, con sus relaciones, atributos y métodos.
- La explotación del método **cantServiciosPosibles** que reciba como parámetro una lista de bicicletas y retorne a cuántas de ellas se le podrá prestar servicio según las condiciones planteadas por el taller.
- Refactorear lo que consideres necesario para cumplir con las nuevas reglas impuestas por el taller para las bicicletas eléctricas:

“Solo se tomarán aquellas de 250w de potencia o menos y que tengan menos de 2000 km, al igual que el resto de las bicicletas que no son eléctricas”.

Una vez diseñado el diagrama de clases UML, crear, implementar (en un proyecto de tipo biblioteca de clases) y probar la jerarquía de clases en un proyecto de consola.

[G.05] - Administrame los empleados

Cierta empresa requiere una aplicación informática para administrar los datos de su personal, del cual se conoce: número de DNI, nombre, apellido y año de ingreso. Existen dos categorías de personal: el personal con salario fijo y el personal a comisión.

- Los empleados a comisión tienen un salario mínimo de $200.000 (para todos igual), un número de clientes captados y un monto a cobrar por cada cliente captado. El salario se obtiene multiplicando los clientes captados por el monto por cliente. Si el salario obtenido por los clientes captados no llega a cubrir el salario mínimo, cobrará el salario mínimo.
- Los empleados con salario fijo tienen un sueldo básico y un porcentaje adicional en función del número de años que llevan la empresa:

  - Menos de 2 años: sueldo básico.
  - De 2 a 5 años: 5% más.
  - De 6 a 10 años: 10% más.
  - Más de 10 años: 15% más.

Basado en el enunciado descrito anteriormente, se pide realizar:

- El diagrama de clases que lo modelice, con sus relaciones, atributos y métodos.
- La explotación del método **mostrarSalarios** que imprime por consola el nombre completo de cada empleado junto a su salario.
- La explotación del método **empleadoConMasClientes** que devuelva al empleado con mayor cantidad de clientes captados (se supone único).

Una vez diseñado el diagrama de clases UML, crear, implementar (en un proyecto de tipo biblioteca de clases) y probar la jerarquía de clases en un proyecto de consola.

[G.06] - UTN-Code

La empresa UTN-Code está construyendo un software para análisis de calidad de código llamado ***Marian***.

Este recibe programas para ser analizados y generar un informe de calidad de código. Los programas se van procesando por orden de llegada y se identifican con un id, un nombre, el nombre del responsable, un *flag* que indica si pasó las pruebas unitarias o no y un conjunto de fuentes.

Para la primera versión, se decidió soportar dos categorías de fuentes: fuentes de lenguajes de programación (*ej. Java, C#, JavaScript*) y fuentes de lenguajes de marcado (*ej. HTML, XML*).

Todos los archivos fuentes tienen un nombre y ubicación (path).

Los fuentes de programación tienen un conjunto de métodos. Cada método tiene un nombre, cantidad de parámetros, la cantidad de instrucciones y la cantidad de ramificaciones de flujo.

Por otro lado, los fuentes de marcado tienen el tamaño del archivo en KB.

El índice de calidad de los programas, fuentes y métodos es calificable mediante un mecanismo común definido según las siguientes reglas:

- Programas: 0 (cero) si no pasó las pruebas unitarias o el promedio del índice de calidad de sus fuentes.
- Fuentes de programación: promedio de índice de calidad de sus métodos.
- Métodos:
  5 / (cantidad de parámetros + 1) + 20 / cantidad de instrucciones + 3 / (cantidad de ramificaciones + 1)
- Fuentes de marcado:
  450 / tamaño en KB

Cada cierto período se ejecuta un proceso que recorre la estructura donde se encuentran los programas pendientes para validar la calidad de acuerdo a la lógica descrita.

Basado en el enunciado descrito anteriormente, se pide realizar:

- El diagrama de clases que lo modelice, con sus relaciones, atributos y métodos.
- Los métodos **indiceCalidad** de las clases calificables.
- El método **programasPorDebajoDe** que recibe un umbral de calidad mínima y retorna una lista de programas cuyo índice de calidad esté por debajo de tal valor.
- El método **mostrarDetalleDeMetodo** de la clase Programa que recibe el nombre de un método y muestra por consola su cantidad de parámetros, instrucciones y ramificaciones, además del índice de calidad. Si no existe, se informa tal situación.
- El método **listadoFuentesMayoresAlPromedio** de la clase Programa que retorne una lista de fuentes cuyo índice de calidad supere al promedio.

Una vez diseñado el diagrama de clases UML, crear, implementar (en un proyecto de tipo biblioteca de clases) y probar la jerarquía de clases en un proyecto de consola.

[G.07] - Lavadero - vehículos

Crear en un proyecto de tipo Class Library la siguiente jerarquía de clases:
Clase Vehiculo que posea como atributos protegidos:

- patente : String (con un getter)
- cantidadRuedas : Byte
- marca : Marcas (con los siguientes enumerados: HONDA, FORD, ZANELLA, SCANIA, IVECO y FIAT). Crear getter.

Y los siguientes métodos:

- (protegido) **mostrar()** : String (retorna en formato de cadena todos los detalles del vehículo)
- (público) **Vehiculo (String, Byte, Marcas)** (sin sobrecargas)

Sobrecarga de métodos:

- (público y estático) **sonIguales(Vehiculo, Vehiculo)** : boolean. Si las patentes y marcas son iguales, retorna TRUE.

Además se pide:

Crear tres clases (Auto, Camion y Moto) que hereden de Vehiculo y que posean: cantidadAsientos (int), para auto, tara (float), para camión y cilindrada (float), para moto. Todos atributos protegidos.

Cada una de estas clases deberá implementar el método **mostrarAuto()**, **mostrarCamion()** y **mostrarMoto()** (reutilizando código de la clase base) para poder retornar un String con todos sus atributos. Generar un constructor en cada clase para inicializar cada uno de los atributos.

Por último se desea construir la clase Lavadero que tendrá como atributos:

- (privado) vehiculos : List<Vehiculo>
- (privado) precioAuto : float
- (privado) precioCamion :float
- (privado) precioMoto :float

Todos los atributos se inicializan desde su constructor con parámetros. El constructor por default, que será privado, será el único encargado de inicializar la lista genérica.

Los métodos que tendrá Lavadero son:

(público) **getDetalle()** : string (que retornará la información completa del lavadero: los precios vigentes y el listado completo de los vehículos que contiene. Reutilizar código).

(público) **getVehiculos()**: List<Vehiculo>, asociado a la lista genérica.

(público) **MostrarTotalFacturado**: devolverá la ganancia total del lavadero (Double), dicho método tendrá una sobrecarga que reciba como parámetro la enumeración Vehiculos (con AUTO, CAMION y MOTO como enumerados) y retornará la ganancia del Lavadero por tipo de vehículo.

(público) **sonIguales** entre un lavadero y un vehículo, retorna TRUE, si el vehículo se encuentra en el lavadero.

(público) **agregar**, que agrega un vehículo siempre y cuando el vehículo no se encuentre en el lavadero. Retorna un booleano indicando lo acontecido.
Ej. miLavadero.agregar(unAuto);

(público) **remover**, que quitara al vehículo del lavadero, siempre y cuando este dicho vehículo. Retorna un booleano indicando lo acontecido.
Ej. miLavadero.remover(unaMoto);

La aplicación debe poder ingresar vehículos de distintos tipos y marcas al lavadero, quitarlos, obtener las ganancias totales o por tipo de vehículo y mostrar el detalle completo de los vehículos ingresados al lavadero.

Una vez diseñado el diagrama de clases UML, crear, implementar y probar la jerarquía de clases en un proyecto de consola.