Materia: Programación II	
Nivel: 2º Cuatrimestre	
Tipo de Examen: Primer Parcial	
Apellido(1):	Fecha: 09 oct 2025
Nombre/s(1):	Docentes a cargo(2): Facundo Rocha - Leandro Ramirez
División(1): 124	Nota(2):
DNI(1):	Firma(2):

(1) Campos a completar solo por el estudiante en caso de imprimir este enunciado en papel.
(2) Campos a completar solo por el docente en caso de imprimir este enunciado en papel.

Enunciado:

De acuerdo a las descripciones de las siguientes clases, se pide:
Modelar en UML (adjuntando la imagen en la entrega) y posteriormente crear el código en JAVA.
Generar un proyecto de tipo consola en JAVA nombrado como: PP.Apellido.Nombre. Crear un package "Entidades", el cual tendrá las siguientes clases:

Fabricante:

Posee todos sus atributos privados.

- *nombre: String.*
- *pais: String.*

Un único constructor que inicializa dichos atributos.

Métodos:

- *(+) sonIguales(Fabricante , Fabricante):boolean* : Retorna true si los nombres y los países son iguales, false, caso contrario. Este método será público y estático.
- *(+)getFabricante(): String* : Este método público y de instancia, retornará una cadena de caracteres que contenga el nombre y el país del fabricante, separados por guiones medios. Ej.: "Snow Cream - Italia".

ProductoHeladeria:

Esta clase **no podrá instanciarse** y todos sus atributos son protegidos:
- *fabricante: Fabricante*.
- *nombre: String.*
- *precio: double.*
- *puntaje: int.*
- *generadorPuntaje: Random (De clase).*

Constructores:
- *ProductoHeladeria (nombre: String, precio: double, fabricante: Fabricante).*
- *ProductoHeladeria (nombre: String, precio: double, nombreFabricante: String, paisFabricante: String).*

Bloque estático:
- Inicializa el atributo *generadorPuntaje*.

Métodos:
- *(+) getPuntaje(): int* : retornará el valor correspondiente del atributo *puntaje*, que se inicializará en dicha propiedad, si y sólo si, su valor es cero. Para inicializar dicho atributo, se utilizará el atributo de clase *generadorPuntaje* con valores aleatorios entre 1 y 10.
- *(-) mostrar(ProductoHeladeria): string:* Método de clase que retorna una cadena detallando todos los atributos del parámetro de tipo *ProductoHeladeria* que recibe.
- *(+) sonIguales(ProductoHeladeria*, *ProductoHeladeria): boolean* : Método de clase que retorna true, si al comparar dos objetos de tipo *ProductoHeladeria*, los nombres y fabricantes son iguales, false, caso contrario.
- Implementar interface **Comparable<T>** tal que la clase implemente su método compareTo(T): int que permitirá definir como orden natural de comparación el nombre de los productos.
- *(+) equals(Object):boolean* : Retorna true, si el objeto recibido por parámetro no es nulo, es de tipo *ProductoHeladeria* y además los nombres y fabricantes sean iguales. False, caso contrario.
- *(+) toString(): String* : Retornará el detalle del nombre, fabricante y puntaje del producto.

Interface IVendible:
- *getPrecioTotal(): double*

Enumerados:

● *SaborHelado* :

[CHOCOLATE, VAINILLA, FRUTILLA, CREMA_AMERICANA, DULCE_DE_LECHE]

● *CategoriaHelado* :

[CLASICO, GOURMET, VEGANO]

● *TipoPostre* :

[TORTA_HELADA, COPA_BROWNIE, SUNDAE]

● *TipoProducto* :

[HELADOS, POSTRES, TODOS]

Helado (deriva de ProductoHeladeria e implementa IVendible):

Posee dos atributos:

● *sabor: SaborHelado*.
● *categoriaHelado: CategoriaHelado*.

Serán inicializados por su único constructor.

Métodos:

● *(+) getPrecioTotal() : double :* Método público y de instancia, devuelve el precio total estimado del producto. Este valor será calculado a partir del precio del producto, incrementado según la categoría del helado, de acuerdo a las siguientes reglas:

- Si la categoría es *CLASICO*, se retorna el valor más un 5% adicional.
- Si la categoría es *GOURMET*, se retorna el valor más un 20% adicional.
- Si la categoría es *VEGANO*, se retorna el valor más un 10% adicional.

● *(+) toString() : String* : Retorna una cadena de caracteres conteniendo la información completa del helado incluyendo su precio total.
● *(+) equals(Object) : boolean* : Retorna true, si ambos productos son iguales y si el sabor del helado es el mismo. False, caso contrario.

Postre (deriva de ProductoHeladeria e implementa IVendible):

Posee un único atributo:

● *tipoPostre: TipoPostre*.

Será inicializado por su único constructor.

Métodos:
● *(+) getPrecioTotal() : double :* Método público y de instancia, devuelve el precio total estimado del producto. Este valor será calculado a partir del precio del producto, incrementado según el tipo de postre, de acuerdo a las siguientes reglas:
- Si el tipo es *TORTA_HELADA*, se retorna el valor más un 30% adicional.
- Si el tipo es *COPA_BROWNIE*, se retorna el valor más un 20% adicional.
- Si el tipo es *SUNDAE*, se retorna el valor más un 10% adicional.

● *(+) toString() : String* : Retorna una cadena de caracteres conteniendo la información completa del postre incluyendo su precio total.
● *(+) equals(Object) : boolean* : Retorna true, si ambos productos son iguales y si el tipo de postre es el mismo. False, caso contrario.

Heladeria:

Dicha clase posee dos atributos, ambos privados:

- *capacitad: int*.
- *productos: Collection<ProductoHeladeria>* : (Elegir la colección que crean correspondiente).

Constructores:

- *Heladeria(): Inicializa la colección y le da valor por defecto 3 a la capacidad.*
- *Heladeria(int): Recibe un entero con la capacidad.*

Métodos:

- *(-) sonIguales(ProductoHeladeria p) : boolean* : Retorna true, si es que el producto ya se encuentra en la heladeria, false, caso contrario.
- *(+) agregar(ProductoHeladeria): void* : Si la heladeria posee capacidad de almacenar al menos un producto más y dicho producto no se encuentra en la heladeria, la agrega a la colección, caso contrario, informará lo acontecido.
- *(-) getPrecioProductos(TipoProducto) : double* : Retorna el precio total de los productos de la heladeria de acuerdo con el enumerado que recibe como parámetro.
- *(-) getPrecioDeHelados() : double :* Retorna el precio total de todos los helados.
- *(-) getPrecioDePostres() : double :* Retorna el precio total de todos los postres.
- *(-) getPrecioTotal() : double :* Retorna el precio total de todos los productos.
- *(-)ordenarPorNombre(): void :* Internamente ordenará su colección de productos por nombre.
- *(+) toString() : String* : Retorna un String con toda la información de la heladeria que recibe cómo parámetro, incluyendo:
    - Nombre de la heladeria.
    - La cantidad de los productos.
    - Los productos ordenados por nombre.
    - El detalle de cada uno de los productos.
    - Los precios totales por tipo de producto.

Copiar las siguientes líneas de código en el método main (de la clase Principal), sin modificar nada.

Main:

```
 Heladeria miHeladeria = new Heladeria(5);

      Fabricante f1 = new Fabricante("FríoMix", "Argentina");
      Fabricante f2 = new Fabricante("DolceLatte", "Italia");
      Fabricante f3 = new Fabricante("SweetArt", "Uruguay");

      Helado h1 = new Helado("Chocolate Belga", 1500, f1,
CategoriaHelado.GOURMET,SaborHelado.CHOCOLATE);
      Helado h2 = new Helado("Vainilla Clásica", 1200,
f2,CategoriaHelado.CLASICO,SaborHelado.VAINILLA);
      Helado h3 = new Helado("Helado Vegano de Almendra", 1800,
f3,CategoriaHelado.VEGANO,SaborHelado.DULCE_DE_LECHE);

      Postre p1 = new Postre("Copa Brownie", 2000, "Daniel","Argentina", TipoPostre.COPA_BROWNIE);
      Postre p2 = new Postre("Torta Helada Oreo", 2500, "Freddo","Argentina",
TipoPostre.TORTA_HELADA);
      Postre p3 = new Postre("Sundae Tropical", 1800, "Lucciano","Argentina", TipoPostre.SUNDAE);

      // Agregados
      miHeladeria.agregar(h1);
      miHeladeria.agregar(p1);
      miHeladeria.agregar(h2);

      // Intento de agregar repetido
      System.out.println("\n Intentando agregar producto repetido...");
      miHeladeria.agregar(h1);

      // Agregados
      miHeladeria.agregar(p2);
      miHeladeria.agregar(h3);

      // Intento de agregar sin lugar
      System.out.println("\n Intentando agregar producto sin lugar...");
```

```
        miHeladeria.agregar(p3);

        // Pruebas de equals

        Helado h4 = new Helado("Chocolate Belga", 1500, f1,
CategoriaHelado.GOURMET,SaborHelado.CHOCOLATE);
        Postre p4 = new Postre("Copa Brownie", 2000, "Daniel","Argentina",
TipoPostre.COPA_BROWNIE);

        System.out.println("\n Pruebas equals:");
        System.out.println(h1.equals(h1));       // TRUE
        System.out.println(h1.equals(h4));       // TRUE
        System.out.println(h1.equals(h2));       // FALSE
        System.out.println(p1.equals(p3));       // FALSE
        System.out.println(p1.equals(p2));       // FALSE
        System.out.println(p1.equals(p4));       // TRUE

        // Mostrar heladería completa
        System.out.println("\n Información de la Heladería:\n");
        System.out.println(miHeladeria);
```

Objetivos de Aprobación No Directa (Calificación de 4 a 5 puntos):

1) El estudiante demuestra comprensión básica de la estructura de clases y relaciones orientadas a objetos.

2) El proyecto cumple con la mayoría de los requerimientos mínimos solicitados y permite su ejecución sin errores críticos.

3) Se identifican intentos de aplicar buenas prácticas de programación, aunque pueden presentarse fallas parciales en lógica o reutilización de código.

4) Entrega del diagrama UML con sus clases y relaciones.

Objetivos de Aprobación Directa (Calificación de 6 a 10 puntos):

5) El estudiante resuelve correctamente la totalidad del problema propuesto, reflejando dominio de los conceptos centrales de POO (encapsulamiento, herencia, sobrecarga/sobrescritura, uso de interfaces y enumeraciones).

6) El diseño es claro, coherente y se evidencia la reutilización de código donde corresponde.

7) El código es legible, estructurado y demuestra una correcta interpretación del enunciado y sus objetivos.

IMPORTANTE:

- NO se corregirán exámenes que NO compilen.
- NO se corregirán exámenes que NO contengan la imagen del modelado en UML.
- No sé corregirán proyectos que no sea identificable su autor.
- Reutilizar tanto código como sea posible.
- Colocar nombre de la clase (en estáticos), this o super en todos los casos que corresponda.
- Subir el proyecto y la imagen UML en el readme a un repositorio de github. Nombrarlo como “Primer parcial programacion 2 - apellido nombre”.

Duración: 150 minutos.