CURSO DE PROGRAMACIÓN FULL STACK

RELACIONES ENTRE CLASES

PARADIGMA ORIENTADO A OBJETOS

GUÍA RELACIONES ENTRE CLASES

RELACIONES ENTRE CLASES

Las relaciones existentes entre las distintas clases indican como se están comunicando las clases entre sí. Dentro de las relaciones entre clases que existen vamos a definir las siguientes:

USO O ASOCIACIÓN

Representa un tipo de relación muy particular, en la que una clase es instanciada por otro objeto y clase. La asociación se podría definir como el momento en que dos objetos se unen para trabajar juntos y así, alcanzar una meta. Un punto a tomar muy en cuenta es que ambos objetos son independientes entre sí. Para validar la asociación, la frase “Usa un”, debe tener sentido, por ejemplo: El programador usa una computadora. El objeto computadora va a existir más allá de que exista o no el objeto programador.

COMPOSICIÓN

La composición es una relación más fuerte que la asociación, es una "relación de vida", es decir, que el tiempo de vida de un objeto está condicionado por el tiempo de vida del objeto que lo incluye. La composición es un tipo de relación dependiente en dónde un objeto más complejo es conformado por objetos más pequeños. En esta situación, la frase “Tiene un”, debe tener sentido, por ejemplo: el cliente tiene una cuenta bancaria. Esta relación es una composición, debido a que al eliminar el cliente la cuenta bancaria no tiene sentido, y también se debe eliminar, es decir, la cuenta existe sólo mientras exista el cliente.

HERENCIA

La herencia es una relación fuerte entre dos clases donde una clase es padre de otra. Las propiedades comunes se definen en la superclase (clase padre) y las subclases heredan estas propiedades (Clase hija). En esta relación, la frase “Un objeto es un-tipo-de una superclase” debe tener sentido, por ejemplo: un perro es un tipo de animal, o también, una heladera es un tipo de electrodoméstico.

Herencia y Constructores

Una diferencia entre los constructores y los métodos es que los constructores no se heredan, pero los métodos sí. Todos los constructores definidos en una superclase pueden ser usados desde constructores de las subclases a través de la palabra clave super. La palabra clave super es la que me permite elegir qué constructor, entre los que tiene definida la clase padre, es el que debo usar. Si la superclase tiene definido el constructor vacío y no colocamos una llamada explícita super, se llamará el constructor vacío de la superclase.

Herencia y Métodos

Todos los métodos accesibles o visibles de una superclase se heredan a sus subclases. ¿Qué ocurre si una subclase necesita que uno de sus métodos heredados funcione de manera diferente? Los métodos heredados pueden ser redefinidos en las clases hijas. Este mecanismo se lo llama sobreescritura. La sobreescritura permite que las clases hijas sumen sus particulares en torno al funcionamiento y agrega coherencia al modelo. Lo que se espera es que a medida que desciendo por la jerarquía de herencia, me encuentre con clases más específicas.

Polimorfismo

El término polimorfismo es una palabra de origen griego que significa "muchas formas". Este término se utiliza en la POO para referirse a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos, es decir, que la misma operación se realiza en las clases de diferente forma. Estas operaciones tienen el mismo significado y comportamiento, pero internamente, cada operación se realiza de diferente forma. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

MODIFICADORES DE CLASES Y METODOS

Clases Finales

El modificador final puede utilizarse también como modificador de clases. Al marcar una clase como final impedimos que se generen hijos a partir de esta clase, es decir, cortamos la jerarquía de herencia.

Métodos Finales

El modificador final puede utilizarse también como modificador de métodos. La herencia nos permite reutilizar el código existente y uno de los mecanismos es la crear una subclase y sobrescribir alguno de los métodos de la clase padre. Cuando un método es marcado como final en una clase, evitamos que sus clases hijas puedan sobrescribir estos métodos.

Métodos Abstractos

Un método abstracto es un método declarado, pero no implementado, es decir, es un método del que solo se escribe su nombre, parámetros, y tipo devuelto, pero no su código de implementación. Estos métodos se heredan por las clases hijas y éstas son las responsables de implementar sus funcionalidades.

Clases Abstractas

En java se dice que una clase es abstracta cuando no se permiten instancias de esa clase, es decir que no se pueden crear objetos. Cuando una clase posee al menos un método abstracto esa clase necesariamente debe ser marcada como abstracta.

Esto indica que no pueden existir instancias de esa clase debido a que poseen por lo menos un método sin implementar. Los hijos de una clase abstracta deben ser los encargados de sobrescribir los métodos abstractos e implementar las funcionalidades.

Interfaces

Una interfaz en Java es una clase que contiene todos sus métodos abstractos y sus atributos son sólo constantes. En las interfaces se especifica qué se debe hacer, pero no su implementación. Las clases que implementen una interface serán las responsables de describir la lógica del comportamiento de los métodos. La principal diferencia entre interface y abstract es que una interface proporciona un mecanismo de encapsulación de los métodos sin forzar a la utilización de una herencia.

Nota: Para más teoría revisar el drive donde hay un archivo con teoría más a fondo y más extensa.

PREGUNTAS DE APRENDIZAJE

1) ¿Qué código de los siguientes tiene que ver con la herencia?

   a) public class Componente extends Producto
   b) public class Componente inherit Producto
   c) public class Componente implements Producto
   d) public class Componente belong to Producto

2) ¿Cuál tipo de clase debe tener al menos un método abstracto?

   a) final
   b) abstract
   c) public
   d) Todas las anteriores

3) ¿De cuántas clases se puede derivar Java?

   a) Tres clases
   b) Dos clases
   c) Una clase
   d) Cinco clases

4) Una clase que termina la cadena de una herencia de la puede declarar como:

   a) final
   b) abstract
   c) public
   d) Ninguna de las anteriores

5) ¿Qué código asociarías a una Interfaz en Java?

   a) public class Componente interface Product
   b) Componente cp = new Componente (interfaz)
   c) public class Componente implements Printable
   d) Componente cp = new Componente.interfaz

6) ¿Qué significa sobrecargar (overload) un método?
   a) Editarlo para modificar su comportamiento
   b) Cambiarle el nombre dejándolo con la misma funcionalidad
   c) Crear un método con el mismo nombre, pero diferentes argumentos
   d) Añadirle funcionalidades a un método

EJERCICIOS DE APRENDIZAJE

En este módulo de POO, vamos a empezar a ver cómo dos o más clases pueden relacionarse entre sí, ya sea por una relación entre clases o mediante una herencia de clases.

RELACIONES ENTRE CLASES

VER VIDEOS:

A. Identificar nuevas clases
B. Uso y Composición
C. Agregación

1. Realizar el juego de la ruleta rusa en Java. Como muchos saben, el juego se trata de un número de jugadores, que con un revolver, el cual posee una sola bala en el tambor, se dispara en la cabeza. Las clases a hacer del juego son las siguientes:

Clase Revolver: esta clase posee los siguientes atributos: posición actual (posición del tambor donde se dispara, puede que esté la bala o no) y posición bala (la posición del tambor donde se encuentra la bala). Estas dos posiciones, se generarán aleatoriamente.

Métodos:

- llenarRevolver(): le pone los valores de posición actual y de posición de la bala. Los valores deben ser aleatorios.
- disparar(): devuelve true si la bala coincide con la posición actual
- siguienteBala(): cambia a la siguiente posición del tambor
- toString(): muestra información del revolver (posición actual y donde está la bala)

Clase Jugador: esta clase posee los siguientes atributos: id (representa el número del jugador), nombre (Empezara con Jugador más su ID, "Jugador 1" por ejemplo) y vivo (indica si está vivo o no el jugador. El número de jugadores será decidido por el usuario, pero debe ser entre 1 y 6. Si no está en este rango, por defecto será 6.

Métodos:

- llenarJugador(): el metodo tiene un Scanner, le pide al usuario los valores del jugador y se guardan en los atributos correspondientes. Excepto vivo que se pone en true por defecto. Este jugador lleno se guarda en un ArrayList de jugadores para ser usado despues.
- disparo(Revolver r): el metodo, recibe el revolver y utilizando los metodos de disparar() y siguienteBala() de Revolver tiene que lograr la siguiente tarea. El jugador se apunta y se dispara, si la bala se dispara, el jugador muere, el atributo vivo pasa a false y el método devuelve true, sino false.

Clase Juego: esta clase posee los siguientes atributos: Jugadores (conjunto de Jugadores) y Revolver.

Métodos:

- llenarJuego(ArrayList<Jugador>jugadores, Revolver r): este metodo recibe los jugadores y el revolver para guardarlos en los atributos del juego.
- ronda(): En cada ronda se recorre la lista de jugadores y llama al método disparo(Revolver r) de la clase jugador. A este, se le pasa el revolver que tenemos de atributo. El jugador se apunta y se dispara, y luego se informara del estado de la partida, usando lo que devuelve el método disparo(Revolver r) (El jugador se dispara, no ha muerto en esa ronda, etc.). Cuando un jugador muere la ronda y el juego se terminan.

En cada turno uno de los jugadores, dispara el revólver, si este tiene la bala el jugador muere y el juego termina.

2. Nos piden hacer un programa sobre un Cine (solo de una sala) que tiene un conjunto de asientos (8 filas por 9 columnas). Del cine nos interesa conocer la película que se está reproduciendo y el precio de la entrada. Luego, de las películas nos interesa saber el título, duración, edad mínima y director. Por último, del espectador, nos interesa saber su nombre, edad y el dinero que tiene disponible.

Los asientos son etiquetados por una letra (columna) y un número (fila), la fila 1 empieza al final de la matriz como se muestra en la tabla. También deberemos saber si está ocupado o no el asiento.

8 A 8 B 8 C 8 D 8 E 8 F 8 G 8 H 8 I
7 A 7 B 7 C 7 D 7 E 7 F 7 G 7 H 7 I
6 A 6 B 6 C 6 D 6 E 6 F 6 G 6 H 6 I
5 A 5 B 5 C 5 D 5 E 5 F 5 G 5 H 5 I
4 A 4 B 4 C 4 D 4 E 4 F 4 G 4 H 4 I
3 A 3 B 3 C 3 D 3 E 3 F 3 G 3 H 3 I
2 A 2 B 2 C 2 D 2 E 2 F 2 G 2 H 2 I
1 A 1 B 1 C 1 D 1 E 1 F 1 G 1 H 1 I

Se debe realizar una pequeña simulación, en la que se generen muchos espectadores y se los ubique en los asientos aleatoriamente (no se puede ubicar un espectador donde ya este ocupado el asiento). Los espectadores serán ubicados de uno en uno.
Tener en cuenta que sólo se podrá sentar a un espectador si tiene el dinero suficiente para pagar la entrada, si hay espacio libre en la sala y si tiene la edad requerida para ver la película. En caso de que el asiento este ocupado se le debe buscar uno libre.

3. Realizar una baraja de cartas españolas orientada a objetos. Una carta tiene un número entre 1 y 12 (el 8 y el 9 no los incluimos) y un palo (espadas, bastos, oros y copas). Esta clase debe contener un método toString() que retorne el número de carta y el palo. La baraja estará compuesta por un conjunto de cartas, 40 exactamente.

Las operaciones que podrá realizar la baraja son:

- barajar(): cambia de posición todas las cartas aleatoriamente.
- siguienteCarta(): devuelve la siguiente carta que está en la baraja, cuando no haya más o se haya llegado al final, se indica al usuario que no hay más cartas.
- cartasDisponibles(): indica el número de cartas que aún se puede repartir.
- darCartas(): dado un número de cartas que nos pidan, le devolveremos ese número de cartas. En caso de que haya menos cartas que las pedidas, no devolveremos nada pero debemos indicárselo al usuario.
- cartasMonton(): mostramos aquellas cartas que ya han salido, si no ha salido ninguna indicárselo al usuario
- mostrarBaraja(): muestra todas las cartas hasta el final. Es decir, si se saca una carta y luego se llama al método, este no mostrara esa primera carta.

VIDEO: Herencia, Clases Abstractas y Polimorfismo

4. Tenemos una clase padre Animal junto con sus 3 clases hijas Perro, Gato, Caballo. Crear un método abstracto en la clase Animal a través del cual cada clase hija deberá mostrar luego un mensaje por pantalla informando de que se alimenta. Generar una clase Main que realice lo siguiente:

```
public class Main {

    public static void main(String[] args) {

        //-->Declaracion del objeto PERRO
            Animal perro = new Perro("Stich","Carnivoro",15,"Doberman");
        perro.Alimentarse();
        //-->Declaracion de otro objeto PERRO
        Perro perro1 = new Perro("Teddy","Croquetas",10,"Chihuahua");
        perro1.Alimentarse();


        //-->Declaracion del objeto Gato
        Animal gato = new Gato("Pelusa","Galletas",15,"Siames");
        gato.Alimentarse();
        //-->Declaracion del objeto Caballo
        Animal caballo = new Caballo("Spark","Pasto",25,"Fino");
        caballo.Alimentarse();

    }
}
```

5. Crear una superclase llamada Electrodomestico con los siguientes atributos: precio base, color, consumo energético (letras entre A y F) y peso. Por defecto, el color será blanco, el consumo energético sera F, el precioBase de $1000 y el peso de 5 kg. Los colores disponibles para los electrodomésticos son blanco, negro, rojo, azul y gris. No importa si el nombre está en mayúsculas o en minúsculas.

Los constructores que se deben implementar son los siguientes:

- Un constructor por defecto.
- Un constructor con el precio y peso. El resto de los atributos por defecto.
- Un constructor con todos los atributos pasados por parámetro.

Los métodos a implementar son:

- Métodos getters y setters de todos los atributos.

- Método comprobarConsumoEnergetico(char letra): comprueba que la letra es correcta, sino es correcta usara la letra por defecto. Este método se debe invocar al crear el objeto y no será visible.

- Método comprobarColor(String color): comprueba que el color es correcto, y si no lo es, usa el color por defecto. Este método se invocará al crear el objeto y no será visible.

- Método precioFinal(): según el consumo energético y su tamaño, aumentará el precio. Esta es la lista de precios:

LETRA	PRECIO
A	$1000
B	$800
C	$600
D	$500
E	$300
F	$100

TAMAÑO	PRECIO
Entre 0 y 19 kg	$100
Entre 20 y 49 kg	$500
Entre 50 y 79 kg	$800
Mayor que 80 kg	$1000

A continuación, se debe crear una subclase llamada Lavadora con el atributo carga, además de los atributos heredados. Por defecto, la carga es de 5 kg.

Los constructores que se implementarán serán:

- Un constructor por defecto.
- Un constructor con el precio y peso. El resto por defecto.

- Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

- Método get y set del atributo carga.

- Método precioFinal(): si tiene una carga mayor de 30 kg, aumentará el precio en $500, si la carga es menor o igual, no se incrementará el precio. Este método debe llamar al método padre y añadir el código necesario. Recuerda que las condiciones que hemos visto en la clase Electrodoméstico también deben afectar al precio.
- Se debe crear también una subclase llamada Televisor con los siguientes atributos: resolución (en pulgadas) y sintonizador TDT (booleano), además de los atributos heredados. Por defecto, la resolución será de 20 pulgadas y el sintonizador será false.

Los constructores que se implementarán serán:

- Un constructor por defecto.

- Un constructor con el precio y peso. El resto por defecto.

- Un constructor con la resolución, sintonizador TDT y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

- Método get y set de los atributos resolución y sintonizador TDT.

- Método precioFinal(): si el televisor tiene una resolución mayor de 40 pulgadas, se incrementará el precio un 30% y si tiene un sintonizador TDT incorporado, aumentará $500. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Finalmente, se debe crear una clase ejecutable que realice lo siguiente:

Crear un array de Electrodomesticos de 10 posiciones. Asignar a cada posición un objeto de las clases anteriores con los valores que desees.

Luego, recorrer este array y ejecutar el método precioFinal(). Se deberá también mostrar el precio de cada tipo de objeto, es decir, el precio de todos los televisores, por un lado, el de las lavadoras por otro, y la suma de todos los Electrodomesticos. Por ejemplo, si tenemos una lavadora con un precio de 2000 y un televisor de 5000, el resultado final será de 7000 (2000+5000) para electrodomésticos, 2000 para lavadora y 5000 para televisor.

6. En un puerto se alquilan amarres para barcos de distinto tipo. Para cada Alquiler se guarda: el nombre y DNI del cliente, las fechas inicial y final de alquiler, la posición del amarre y el barco que lo ocupará. Un Barco se caracteriza por: su matrícula, su eslora en metros y año de fabricación. Un alquiler se calcula multiplicando el número de días de ocupación (incluyendo los días inicial y final) por un valor módulo de cada barco (obtenido simplemente multiplicando por 10 los metros de eslora) y por un valor fijo (2 es en la actualidad).
Sin embargo, se pretende diferenciar la información de algunos tipos de barcos especiales:

- Número de mástiles para veleros
- Potencia en CV para embarcaciones deportivas a motor
- Potencia en CV y número de camarotes para yates de lujo.

El módulo de los barcos de un tipo especial (ej. Velero) se obtiene como el módulo normal más la suma de su atributo particular (ej. Número de mástiles)

Utilizando la herencia de forma apropiada, diseñe el diagrama de clases y sus relaciones, con detalle de atributos y métodos necesarios. Luego, programe en Java los métodos que permitan calcular el alquiler de cualquier tipo de barco.

7. Se plantea desarrollar un programa que permita representar la siguiente situación.
Un polideportivo es al mismo tiempo una instalación deportiva y un edificio; y lo que interesa conocer es la superficie que tiene y los tipos de polideportivos. Un edificio de oficinas es un edificio; interesa conocer la superficie que tiene.
Para lograr esto vamos a tener que crear una interfaz InstalacionDeportiva con un método int tipoDeInstalacion() y una interfaz Edificio con un metodo double calcularSuperficie(). Luego crearemos una clase Polideportivo con sus atributos largo, ancho, nombre y tipo de instalación que puede ser Techado o Abierto, esta clase implementará las dos interfaces. Además, vamos a crear una clase EdificioDeOficinas con sus atributos número de oficinas, ancho, largo y numero de pisos. Esta clase solo implementará la interfaz Edificio.
Por ultimo, crear una clase Test con el método main y dos ArrayList. El primer ArrayList debe contener tres polideportivos, y el segundo, dos edificios de oficinas. Utilizar un iterator para recorrer las colecciones y utilizar el método tipoDeInstalación() para saber cuantos polideportivos son techados y cuantos abiertos. Además usaremos el método calcularSuperficie(), para calcular la superficie de todos los polideportivos como de todos los edificios de oficina.
Una vez realizado el ejercicio responda: ¿Entre qué clases existe una relación que se asemeja a la herencia múltiple?

¿Querés saber más de interfaces? Ejemplo1 – Ejemplo2

8. Una compañía de promociones turísticas desea mantener información sobre la infraestructura de alojamiento para turistas, de forma tal que los clientes puedan planear sus vacaciones de acuerdo a sus posibilidades. Los alojamientos se identifican por un nombre, una dirección, una localidad y un gerente encargado del lugar. La compañía trabaja con dos tipos de alojamientos: Hoteles y Alojamientos Extrahoteleros.

Los Hoteles pueden ser de tres, cuatro o cinco estrellas. Las características de las distintas categorías son las siguientes:

- Hotel *** Cantidad de Habitaciones, Número de Camas, Cantidad de Pisos, Precio de Habitaciones.

- Hotel **** Cantidad de Habitaciones, Número de camas, Cantidad de Pisos, Gimnasio, Nombre del Restaurante, Capacidad del Restaurante, Precio de las Habitaciones.

- Hotel ***** Cantidad de Habitaciones, Número de camas, Cantidad de Pisos, Gimnasio, Nombre del Restaurante, Capacidad del Restaurante, Cantidad Salones de Conferencia, Cantidad de Suites, Cantidad de Limosinas, Precio de las Habitaciones.

Los gimnasios pueden ser clasificados por la empresa como de tipo "A" o de tipo "B", de acuerdo a las prestaciones observadas. Las limosinas están disponibles para cualquier cliente, pero sujeto a disponibilidad, por lo que cuanto más limosinas tenga el hotel, más caro será.

El precio de una habitación debe calcularse de acuerdo a la siguiente fórmula: PrecioHabitación = $50 + ($1 x capacidad del hotel) + (valor agregado por restaurante) + + (valor agregado por gimnasio) + (valor agregado por limosinas).

Donde:

Valor agregado por el restaurante:

- $10 si la capacidad del restaurante es de menos de 30 personas.
- $30 si está entre 30 y 50 personas.
- $50 si es mayor de 50.

Valor agregado por el gimnasio:

- $50 si el tipo del gimnasio es A.
- $30 si el tipo del gimnasio es B.

Valor agregado por las limosinas:

- $15 por la cantidad de limosinas del hotel.

En contraste, los Alojamientos Extra hoteleros proveen servicios diferentes a los de los hoteles, estando más orientados a la vida al aire libre y al turista de bajos recursos. Por cada Alojamiento Extrahotelero se indica si es privado o no, así como la cantidad de metros cuadrados que ocupa. Existen dos tipos de alojamientos extrahoteleros: los Camping y las Residencias. Para los Camping se indica la capacidad máxima de carpas, la cantidad de baños disponibles y si posee o no un restaurante dentro de las instalaciones. Para las residencias se indica la cantidad de habitaciones, si se hacen o no descuentos a los gremios y si posee o no campo deportivo. Realizar un programa en el que se representen todas las relaciones descriptas. Realizar un sistema de consulta que le permite al usuario consultar por diferentes criterios:

- todos los alojamientos

- todos los hoteles de una determinada localidad
- todos los campings de una determinada localidad

9. Sistema Gestión Facultad. Se pretende realizar una aplicación para una facultad que gestione la información sobre las personas vinculadas con la misma y que se pueden clasificar en tres tipos: estudiantes, profesores y personal de servicio. A continuación, se detalla qué tipo de información debe gestionar esta aplicación:

- Por cada persona, se debe conocer, al menos, su nombre y apellidos, su número de identificación y su estado civil.
- Con respecto a los empleados, sean del tipo que sean, hay que saber su año de incorporación a la facultad y qué número de despacho tienen asignado.
- En cuanto a los estudiantes, se requiere almacenar el curso en el que están matriculados.
- Por lo que se refiere a los profesores, es necesario gestionar a qué departamento pertenecen (lenguajes, matemáticas, arquitectura, ...).
- Sobre el personal de servicio, hay que conocer a qué sección están asignados (biblioteca, decanato, secretaría, ...).

El ejercicio consiste, en primer lugar, en definir la jerarquía de clases de esta aplicación. A continuación, debe programar las clases definidas en las que, además de los constructores, hay que desarrollar los métodos correspondientes a las siguientes operaciones:

- Cambio del estado civil de una persona.
- Reasignación de despacho a un empleado.
- Matriculación de un estudiante en un nuevo curso.
- Cambio de departamento de un profesor.
- Traslado de sección de un empleado del personal de servicio.
- Imprimir toda la información de cada tipo de individuo. Incluya un programa de prueba que instancie objetos de los distintos tipos y pruebe los métodos desarrollados.

Material Extra:

A. Desarrollar un simulador del sistema de votación de facilitadores de EGG.

El sistema de votación de Egg tiene una clase llamada Alumno con los siguientes atributos: nombre, apellido, DNI y cantidad de votos.

- La clase Simulador debe tener un método que retorna un listado de alumnos generados de manera aleatoria. Las combinaciones de nombre, apellido deben ser generadas de manera aleatoria. Nota: usar listas de tipo String para generar los nombres y los apellidos.

- Ahora hacer un generador de combinaciones de DNI posibles, deben estar dentro de un rango real de números de documentos. Y agregar a los alumnos su DNI.

- Se debe imprimir por pantalla el listado de alumnos.

- Una vez hecho esto debemos generar una clase Voto, esta clase tendrá como atributos, el alumno que vota y una lista de los alumnos a los que votó.

- El método votación de la clase Simulador que recibe el listado de alumnos y para cada alumno genera tres votos de manera aleatoria. Tener en cuenta que un alumno no puede votarse a sí mismo o votar más de una vez al mismo alumno. Utilizar un hashset para resolver esto. Todos los alumnos deben guardarse en una lista de alumnos. Nota: En este método debemos guardar en la clase Voto a el alumno que vota, a los alumnos a los que votó y sumarle uno a la cantidad de votos a cada alumno votado, que es un atributo de la clase Alumno.

- El recuento de votos recibe la lista de Alumnos y comienza a hacer el recuento de votos utilizando la clase Alumno.

- Se deben crear 5 facilitadores con los 5 primeros alumnos votados y se deben crear 5 facilitadores suplentes con los 5 segundos alumnos más votados.

B. Ahora se debe realizar unas mejoras a la clase Baraja. Lo primero que haremos es que nuestra clase Baraja será la clase padre y será abstracta. Le añadiremos el número de cartas en total y el número de cartas por palo. El método crearBaraja() será abstracto.

La clase Carta tendrá un atributo genérico que será el palo de nuestra versión anterior.

Creamos dos Enumeraciones : https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/):

PalosBarEspañola:

OROS
COPAS
ESPADAS
BASTOS

PalosBarFrancesa:

DIAMANTES
PICAS
CORAZONES
TREBOLES

Crear dos clases hijas:

BarajaEspañola: tendrá un atributo boolean para indicar si queremos jugar con las cartas 8 y 9 (total 48 cartas) o no (total 40 cartas).

BarajaFrancesa: no tendrá atributos, el total de cartas es 52 y el número de cartas por palo es de 13. Esta clase tendrá dos métodos llamados:

- cartaRoja(Carta<PalosBarFrancesa> c): este método recibe una carta y dice si es diamantes o de corazones.
- cartaNegra(Carta<PalosBarFrancesa> c): este método recibe una carta y dice si es trebol o de picas.

De la carta modificaremos el método toString().

Si el palo es de tipo PalosBarFrancesa:

La carta número 11 será Jota

La carta numero 12 será Reina

La carta numero 13 será Rey

La carta numero 1 será As

Si el palo es de tipo PalosBarEspañola:

La carta numero 10 será Sota

La carta numero 12 será Caballo

La carta numero 13 será Rey

La carta numero 1 será As