PRUEBA TÉCNICA

Nombre: Víctor E. Fernández S.
Cédula (PPT): 7554258.
Fecha: 19-09-2023.

1. **Defina los siguientes conceptos:**

Clase:	**R:** Una clase en programación sería una especie de modelo que los programadores creamos como punto de partida para construir una aplicación. En estas clases podemos definir atributos y métodos (que sería las funciones que será capaz de ejecutar la clase).
Objeto:	**R:** Un objeto sería la instancia de una clase. Ejemplo, si tenemos una clase Suma que recibe 2 números por atributos y tiene un método llamado suma(). Para hacer uso de dicho método, tendríamos que crear un objeto de la clase Suma() y por medio de ese nuevo objeto, setear los valores necesarios para llamar el método suma().
Herencia:	**R:** Yo veo la herencia como una facilidad de la POO ya que nos permite heredar métodos y atributos de otras clases. Dicho de otro modo la herencia nos permite acceder a funciones todos los métodos, atributos y funcionalidad en general que pueda proveer su clase padre adicional a los atributos y métodos propios.
Polimorfismo:	**R:** Puedo decir que el polimorfismo es la capacidad que tienen los objetos para hacer la misma acción pero cada uno de manera diferente, adaptado a sus propias necesidades.
Sobrecarga:	**R:** Seria crear varias versiones un mismo método diferenciándolos entre sí por el tipo de parámetro y valor a retornar.
Paquete:	**R:** No son más que carpetas que nos permiten organizar el código de nuestra aplicación.
Framework:	**R:** Diría que son herramientas que nos encapsulan un poco la complejidad de escribir código desde cero. Ya que por lo general vienen con muchas clases pre construidas que nos permiten crear funcionalidades mucho más rápido sin tener que iniciar desde cero.

2. **Asumiendo que existe una clase publica llamada Apuesta con métodos public int getValorApuesta() y public void set ValorApuesta(int valor). El resultado de la línea en negrilla una vez ha sido ejecutado por completo el método encimarApuesta es: <u>1500, 0, null o lanza una excepción.</u>**

public void encimarApuesta(){
Apuesta apuesta = null;
apuesta = crearApuesta(apuesta);
System.out.println(apuesta.getValorApuesta());
}
public Apuesta crearApuesta(Apuesta apuesta){
apuesta = new Apuesta();
apuesta.setValorApuesta(1500);
return apuesta;
}
	R: La respuesta seria **1500.**
	
	Aunque el valor de entrada en el método es **crearApuesta()** es **null**, el propio método internamente está creando una nueva instancia de la clase apuesta y usa esa nueva instancia para setear el nuevo valor de **1500** y por ende es este el valor que retorna.

3. **Mencione los 7 tipos de datos primitivos en Java:**
   R: int, long, float, chart, boolean, double, short.

4. **Dado el siguiente código:**

Public static void main(String[] args){
for(int i=0; i < =10; i++){
if (i > 6) break;
}
System.out.println(i);
}
 	
El resultado de la ejecución del código es:
• 6
• 7
• 10
• Compilación fallida
• Una excepción es lanzada.
	R: Fallaría en la compilación.
	
	No sé si es intencional o no, pero en el (<=) del ciclo for hay un espacio en el medio (< =) generando allí el primer error.
	
	Luego se intenta imprimir la variable i que esta fuera del ciclo de vida de esa variable.

5. **Se tienen las siguientes entidades:**

Empresa(Id_empresa, nombre_empresa)
Producto(Id_producto, nombre_producto)
Vendedor(Id_vendedor, num_documento, nombre, fk_id_empresa)
Venta(Id_venta, valor_total, fk_id_vendedor, fecha, **fk_id_cliente**)
DetalleVenta(id_detalle_venta, valor_total, fk_id_producto, cantidad, fk_id_venta)

Realizar una consulta SQL que muestre la información de la siguiente manera:

Empresa	Producto	Cantidad	Valor Total

R:

```
SELECT t4.nombre_empresa  AS "EMPRESA",
       t5.nombre_producto AS "PRODUCTO",
       t1.cantidad        AS "CANTIDAD",
       t1.valor_total     AS "VALOR TOTAL"
FROM   detalle_venta t1
       INNER JOIN producto t5
               ON t5.id_producto = t1.fk_id_producto
       INNER JOIN venta t2
               ON t2.id_venta = t1.fk_id_venta
       INNER JOIN vendedor t3
               ON t3.id_vendedor = t2.fk_id_vendedor
       INNER JOIN empresa t4
               ON t4.id_empresa = t3.fk_id_empresa;
```

Marque el campo **fk_id_cliente** en el enunciado el problema porque aunque se nombra como fk en el enunciado no se nombró ninguna entidad cliente.

6. **Escriba un procedimiento almacenado o aplicación Java que consolide los totales de venta del punto anterior, garantizando tener solo los últimos 6 meses en dicho consolidado**

R:

```
CREATE PROCEDURE nombrePA()
begin
  SELECT t4.nombre_empresa                  AS "EMPRESA",
         t5.nombre_producto                 AS "PRODUCTO",
         t1.cantidad                        AS "CANTIDAD",
         Sum(t1.valor_total * t1.cantidad) AS "VALOR TOTAL"
  FROM   detalle_venta t1
         INNER JOIN producto t5
                 ON t5.id_producto = t1.fk_id_producto
         INNER JOIN venta t2
                 ON t2.id_venta = t1.fk_id_venta
         INNER JOIN vendedor t3
                 ON t3.id_vendedor = t2.fk_id_vendedor
         INNER JOIN empresa t4
                 ON t4.id_empresa = t3.fk_id_empresa
  WHERE  t2.fecha >= Date_sub(Curdate(), INTERVAL 6 month)
  GROUP  BY id_detalle_venta;
end;
```

Luego de tener el procedimiento ya creado en MySQL, podemos llamarlo de la siguiente manera:

CALL nombrePA()

7. **Escriba 3 formas diferentes de recorrer un arreglo usando el lenguaje Java**

R:

```
String[] a = {"Uno", "Dos", "Tres"};
for (int i = 0; i < a.length; i++) {
System.out.println(a[i]);
}
```

```
String[] a = {"Uno", "Dos", "Tres"};
int i = 0;
while (i < a.length) {
        System.out.println(a[i]);
         i++;
}
```

```
String[] a = {"Uno", "Dos", "Tres"};
int i = 0;
do {
   System.out.println(a[i]);
   i++;
} while (i < a.length);
```

8. **Qué es un patrón de diseño.**

R: Pudiera ir a internet y conseguir un concepto más elaborado, pero yo definiría los patrones de diseños como un conjunto de reglas y formas estandarizadas de diseñar software, lo que les facilita a los desarrolladores crear software de calidad y escalable en el tiempo.

9. Construya un XML que defina una estructura jerárquica para Países, Departamentos, Ciudades y Barrios.

R:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<main>
    <pais nombre="Colombia">
        <departamento nombre="Santander">
            <ciudad nombre="Bucaramanga">
                <barrio nombre="Barrio 1"/>
                <barrio nombre="Barrio 2"/>
            </ciudad>
            <ciudad nombre="Floridablanca">
                <barrio nombre="Barrio 1"/>
            </ciudad>
        </departamento>
        <departamento nombre="Antioquia">
            <ciudad nombre="Medellin">
                <barrio nombre="Barrio 1"/>
            </ciudad>
        </departamento>
    </pais>
    <pais nombre="Vanezuela">
        ...
    </pais>
</main>
```


```xml
<?xml version="1.0" encoding="UTF-8"?>
<main>
  <pais nombre="Colombia">
    <departamento nombre="Santander">
      <ciudad nombre="Bucaramanga">
        <barrio nombre="Barrio 1"/>
        <barrio nombre="Barrio 2"/>
      </ciudad>
      <ciudad nombre="Floridablanca">
        <barrio nombre="Barrio 1"/>
      </ciudad>
    </departamento>
    <departamento nombre="Antioquia">
      <ciudad nombre="Medellin">
        <barrio nombre="Barrio 1"/>
      </ciudad>
    </departamento>
  </pais>
  <pais nombre="Vanezuela">
    ...
  </pais>
</main>
```


10. Defina y de un escenario de aplicación de los siguientes patrones de diseño: Singletón, Fachada, Fábrica Abstracta.

R:

Singleton	Fachada	Fabrica Abstracta
El patrón singleton se usa en cuando es necesario que solo haya una instancia de una determinada clase en toda la aplicación. Ejemplo: Archivos de conexión a bases de datos, cuando es necesario que solo haya una instancia de conexión en toda la aplicación. También es común verlo cuando hacemos uso de clases que contienen algún tipo de conjuración global como información de servidores o cuando queremos estandarizar las respuestas de nuestras APIs podría ser otro ejemplo.	Es la que más conozco y la que uso cuando programo en Spring Boot. No es más que crear una interfaz que donde definiremos solo los métodos que nos interesan ver de una determinada clase para una funcionalidad especifica. Ejemplo: Supongamos que tenemos una clase Producto que es usada por diferentes módulos del sistema como inventario, producción, ventas y contiene una infinidad de métodos que son usados por esos módulos. Pero se nos solicita crear una funcionalidad para que una entidad llamada ClienteWeb pueda consultar los precios de un determinado producto pasando si ID. Creamos una fachada de la clase Producto donde solo llamaremos el método consultarPrecio() y así la entidad ClienteWeb no tiene que saber cuan compleja es la case Producto.	*Siendo honesto, nunca he usado dicho patrón de diseño. Pero Puedo aprenderlo de ser necesario.*

11. **Para qué sirve el entityManager y cuáles son los métodos básicos que este provee.**
    R: Nos permite gestionar los procesos que deseemos realizar con nuestras entidades en la base de datos de una manera mucho más simplificada. El EntityManager ya nos provee métodos para agregar, buscar, eliminar, actualizar nuestros registros. Entre los métodos más usados están: persist, find, merge, remove.

12. **En J2EE que es un EJB. Mencione los principales tipos.**
    R: Los EJB le proporcionan el programador un conjunto de componentes del lado del server y que ayudan mas que todo en tema de seguridad, transacciones, persistencias. Permitiendo así que el programador se centre en lo que es realmente necesario a nivel de lógica de negocio para resolver la necesidad.

    Los tipos de EJB:
    - De Entidad
    - De Sesión
        - Con estado
        - Sin Estados
    - Dirigido por Mensajes

13. **En qué tipo de EJB se puede mapear una tabla de base de datos.**
    R: En el EJB Entity. Pero es importante saber que desde la versión 6 de Java se usa JPA.

14. **Para qué sirve el archivo MANIFEST.MF en una aplicación y donde está ubicado.**
    R: Es un archivo de metadatos usado en los JAR, y sirve para proporcionar información de la aplicación como por ejemplo:
    - Versión
    - Rutas de las clases y paquetes
    - Dependencias
    - Tareas que puede ejecutar la aplicación antes o después de correr.

    Y se encuentra dentro de los JAR. En: META-INF/MANIFEST.MF

15. **En una aplicación JSF donde se definen las reglas de navegación y Backing Bean.**
    R: Marco en rojo la siguiente pregunta porque es importante mi mencionar que yo manejo el backend de las aplicaciones con Spring Boot y el Font con Angular.
    En Spring las rutas se definen en el controlador de la aplicación haciendo uso de la notación **@GetMapping**, **@PostMapping** dependiendo el verbo de la petición que se necesite.

16. **Si se requiere un filtro en la aplicación para que cada vez que se solicite una página se verifique que el usuario esté autenticado que archivos debería construir y cuales debería modificar.**
    R: Tomando como referencia la respuesta de la anterior pregunta. En caso que la aplicación se haga en angular y el backend en Spring Boot el control de sesiones se manejaría a través de un JWT. Y cada servicio de la aplicación hace una verificación de ese token para saber si le brinda el acceso a la información o no.
    Y se haría haciendo uso de la librería **SpringSecurity**.

17. Se requiere de una aplicación web (Angular, JSF) por medio de la cual se puedan vender recargas en línea. Se debe poder identificar en cualquier momento la cantidad y valor de recargas discriminada por operador (Tigo, Movistar, Comcel, Uff) y persona que realiza la venta.

a. **Implementar APIs necesarias en Spring boot (Opcional)**
b. **Crear pantalla para la venta de recargas (no se requiere diseño)**
c. **Realice un diagrama relacional, diagrama de casos de uso, diagrama de secuencia y de clases que sirva como solución para dicha implementación.**
d. **Subir la solución a un repositorio GIT**

Link del repositorio:

https://github.com/victor1693/GelsaAPP.git

Diagrama de casos de uso:

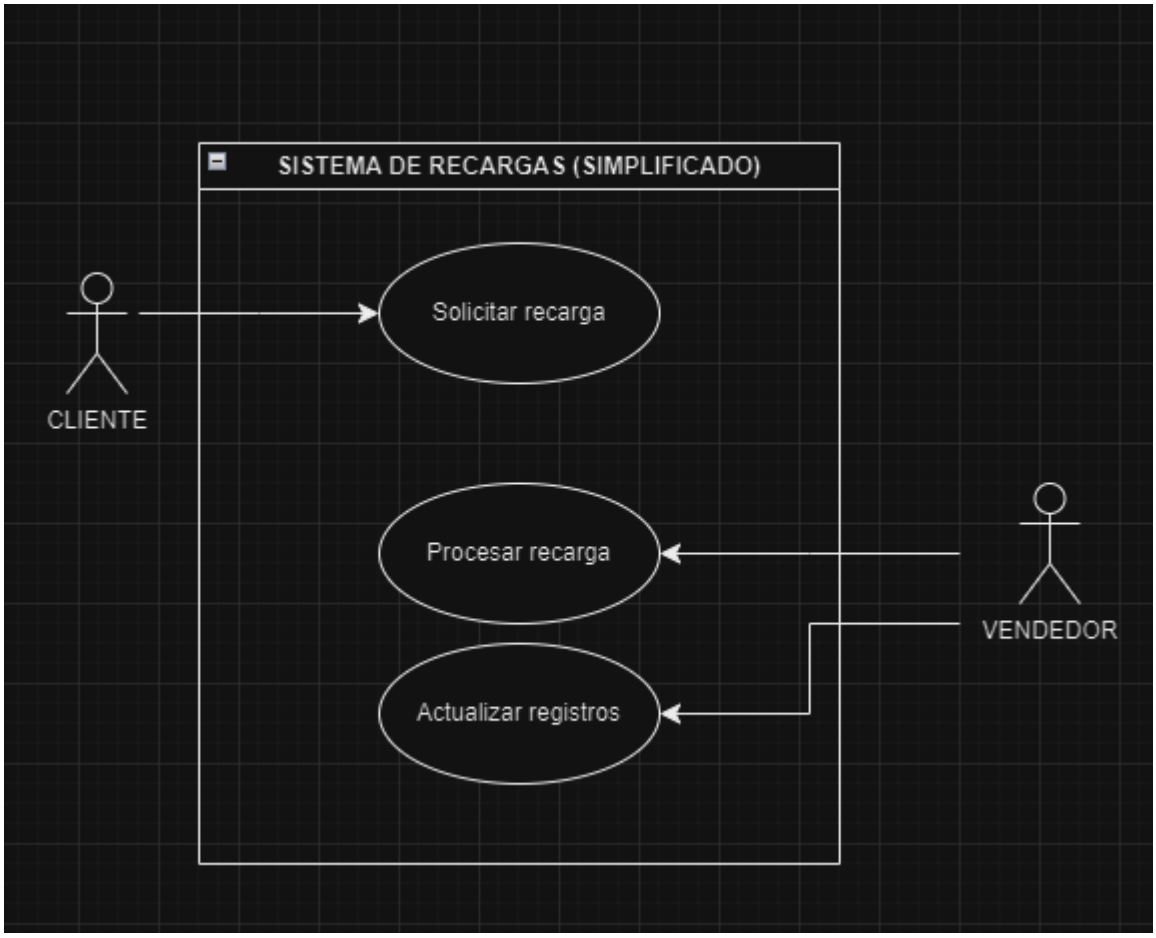

Diagrama relacional

Diagrama de clases