PRUEBA TÉCNICA

Nombre: Yuver Santiago Hurtado Peña **Fecha**: 22/02/2024

Cedula: 1.007.571.769

Objetivo: Evaluar los conocimientos técnicos en cuanto a Desarrollo de Software usando la plataforma JEE.

1) Defina los siguientes conceptos:

- **Clase:** Es un plano, plantilla o prototipo que define las características y comportamientos comunes que tendrán los objetos que se creen a partir de ella. Es decir, una clase describe las propiedades y métodos que los objetos de esa clase tendrán.

- **Objeto:** Un objeto es una instancia concreta de una clase. Es decir, es una entidad que posee características y comportamientos específicos definidos por su clase. Los objetos son las unidades fundamentales de la programación orientada a objetos y se utilizan para representar entidades del mundo real o conceptos abstractos en un programa de computadora.

- **Herencia:** La herencia es un concepto fundamental en la programación orientada a objetos que permite que una clase (llamada clase derivada o subclase) herede propiedades y comportamientos de otra clase (llamada clase base o superclase). Esto significa que la clase derivada puede aprovechar y extender la funcionalidad de la clase base, lo que promueve la reutilización del código y la organización jerárquica de las clases.

- **Polimorfismo:** El polimorfismo es un principio de la programación orientada a objetos que permite que un objeto pueda comportarse de diferentes maneras según el contexto en el que se encuentre. Esto se logra mediante la capacidad de una clase base para ser utilizada de manera genérica y que sus métodos puedan ser sobrescritos por clases derivadas para adaptarse a situaciones específicas.

- **Sobrecarga:** La sobrecarga es un concepto que permite definir múltiples versiones de un mismo método o función en una clase, pero con diferentes parámetros de entrada. Esto significa que se pueden crear métodos con el mismo nombre pero con diferentes listas de argumentos, lo que facilita la llamada a funciones con diferentes tipos de datos.

- **Paquete:** En programación, un paquete (o módulo) es una colección de clases, funciones y otros elementos relacionados que se agrupan juntos para facilitar su organización y reutilización. Los paquetes ayudan a estructurar y modularizar el código, lo que mejora su mantenibilidad y escalabilidad.

- **Framework:** Un framework es una estructura conceptual y/o software que proporciona un conjunto de herramientas, bibliotecas y convenciones predefinidas para facilitar el desarrollo de aplicaciones. Los frameworks suelen ofrecer soluciones a problemas comunes de desarrollo, proporcionando una base sobre la cual los desarrolladores pueden construir y personalizar sus aplicaciones de manera más rápida y eficiente.

2. Asumiendo que existe una clase pública llamada Apuesta con métodos public int getValorApuesta() y public void set ValorApuesta(int valor)

```
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;
}
```

El resultado de la línea en negrilla una vez ha sido ejecutado por completo el método encimarApuesta es:

a. 1500
b. 0
c. Null
d. Es Lanzada una excepción.

3. Para qué sirve el archivo MANIFEST.MF en una aplicación y donde está ubicado.

R./ El archivo MANIFEST.MF es un archivo de metadatos utilizado en aplicaciones Java, especialmente en el contexto de archivos JAR (Java ARchive). Este archivo proporciona información sobre la estructura y configuración de la aplicación, así como también detalles sobre las dependencias y atributos específicos de la aplicación.

El archivo MANIFEST.MF se encuentra dentro del directorio META-INF (META-INF/MANIFEST.MF) en un archivo JAR. En un archivo JAR estándar, la estructura del directorio META-INF es utilizada para almacenar metadatos relacionados con el archivo JAR y su contenido.

4. En una aplicación JSF donde se definen las reglas de navegación y Backing Bean.

R/

- **Reglas de Navegación**: En JSF, las reglas de navegación se definen en el archivo faces-config.xml. Este archivo de configuración es utilizado por el contenedor de servlets (como Apache Tomcat) para configurar los componentes de la aplicación JSF. Dentro de este archivo, se especifican las transiciones entre las vistas y las acciones que deben ejecutarse al navegar entre las páginas
- **Backing Beans**: Los backing beans son clases de Java que actúan como controladores en una aplicación JSF. Estas clases contienen métodos y propiedades que interactúan con la interfaz de usuario y manejan la lógica de presentación. Los backing beans se pueden definir en cualquier paquete o directorio dentro de la estructura de la aplicación web

5. 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/:

- **Crear un Filtro de Autenticación**: Debes crear una clase Java que implemente la interfaz javax.servlet.Filter. Esta clase será responsable de verificar si el usuario está autenticado antes de permitir el acceso a las páginas protegidas.
- **Configurar el Filtro en el archivo web.xml:** Debes configurar el filtro en el archivo web.xml para que se aplique a las solicitudes entrantes
- **Modificar Páginas para Autenticación:** En las páginas que deseas proteger, como las páginas de contenido restringido o las páginas a las que solo pueden acceder usuarios autenticados, debes agregar la lógica de autenticación. Esto puede incluir un formulario de inicio de sesión y la validación de credenciales en un backing bean.

6. Mencione los 7 tipos de datos primitivos en Java:

R/

- **byte**: Es un tipo de dato entero de 8 bits con signo que puede almacenar valores en el rango de -128 a 127.
- **short:** Es un tipo de dato entero de 16 bits con signo que puede almacenar valores en el rango de -32,768 a 32,767.
- **int:** Es un tipo de dato entero de 32 bits con signo que puede almacenar valores en el rango de -2^31 a 2^31 - 1.
- **long:** Es un tipo de dato entero de 64 bits con signo que puede almacenar valores en el rango de -2^63 a 2^63 - 1.
- **float:** Es un tipo de dato de punto flotante de 32 bits que puede representar números decimales de precisión simple.
- **double:** Es un tipo de dato de punto flotante de 64 bits que puede representar números decimales de precisión doble.
- **boolean:** Es un tipo de dato que solo puede tener dos valores: true o false, utilizado para representar valores lógicos.

7. 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 anterior es:

a- 6
b- 7
c- 10
d- Compilación Fallida.
e- Una excepción es lanzada.

8. En J2EE que es un EJB. Mencione los principales tipos.

R./

- **Session Beans (EJB de sesión):** Son EJBs que encapsulan la lógica de negocio de una aplicación. Pueden ser de dos tipos:
    1. **Stateless Session Beans (EJB de sesión sin estado):** No mantienen un estado de conversación con un cliente específico entre invocaciones de método. Son útiles para implementar lógica de negocio que no depende del estado del cliente.
    2. **Stateful Session Beans (EJB de sesión con estado):** Mantienen un estado de conversación con un cliente específico entre invocaciones de método. Son útiles para implementar lógica de negocio que necesita mantener un estado específico para un cliente durante su sesión.
- **Entity Beans (EJB de entidad):** Representan objetos persistentes en una base de datos relacional. Los Entity Beans gestionan el mapeo entre objetos Java y registros en la base de datos. En versiones anteriores de J2EE, los Entity Beans eran una parte central de la especificación de EJB, pero en versiones más recientes, como Java EE 6 y superior, se prefieren enfoques como JPA (Java Persistence API) para el acceso a datos.
- **Message-Driven Beans (EJB impulsados por mensajes):** Son EJBs utilizados para procesar mensajes de forma asíncrona. Estos beans se activan por la llegada de un mensaje a una cola JMS (Java Message Service). Son útiles para tareas de procesamiento de mensajes que requieren un comportamiento asíncrono y escalabilidad.

9. En qué tipo de EJB se puede mapear una tabla de base de datos.

R./

- **Entity Beans CMP (Container Managed Persistence):** En versiones anteriores de J2EE, la persistencia de los Entity Beans se gestiona completamente por el contenedor de EJB. El

contenedor era responsable de administrar el ciclo de vida de los Entity Beans y de gestionar automáticamente la sincronización entre los objetos Java y los datos de la base de datos. Sin embargo, este enfoque tenía limitaciones y se volvió menos popular con la introducción de JPA (Java Persistence API).

- **Entity Beans BMP (Bean Managed Persistence):** Este enfoque permite que el desarrollador controle manualmente la lógica de persistencia de los Entity Beans. El desarrollador es responsable de escribir el código para administrar el almacenamiento y la recuperación de los datos de la base de datos en los métodos del bean. Aunque ofrece más control sobre la persistencia, también puede ser más complejo de implementar.

10. Defina y de un escenario de aplicación de los siguientes patrones de diseño:

Singletón, Fachada, Fábrica Abstracta.

R/

- **Singleton:** El patrón Singleton es útil cuando solo se necesita una instancia de una clase en toda la aplicación. Por ejemplo, en un sistema de registro, donde se necesita un único registro de actividad para toda la aplicación. Si múltiples instancias del registro estuvieran disponibles, podría haber problemas de consistencia y sincronización.
- **Fachada:** Imagina un sistema de reserva de vuelos. En lugar de que el cliente tenga que interactuar directamente con cada componente del sistema (como la reserva de asientos, la gestión de pagos, etc.), se proporciona una fachada que encapsula todas estas operaciones complejas detrás de una interfaz simplificada. El cliente solo necesita interactuar con la fachada para realizar una reserva de vuelo.
- **Fábrica Abstracta:** Supongamos que estás desarrollando un videojuego y necesitas crear diferentes tipos de personajes, como guerreros, magos y arqueros. Cada tipo de personaje tiene su propio conjunto de armas y armaduras. En lugar de crear las armas y armaduras directamente en el código del juego, puedes utilizar una fábrica abstracta que genere una familia de objetos relacionados (por ejemplo, una fábrica de personajes que produzca guerreros con espadas y escudos, magos con varitas y túnicas, etc.). Esto permite que el código del juego sea más flexible y fácil de mantener, ya que puedes cambiar la fábrica abstracta para generar diferentes familias de objetos sin afectar al resto del código

11. 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
   E.nombre_empresa AS Empresa,
   P.nombre_producto AS Producto,
   SUM(DV.cantidad) AS Cantidad,
   SUM(DV.valor_total) AS 'Valor Total'
FROM
   Empresa E
   JOIN Vendedor V ON E.Id_empresa = V.fk_id_empresa
   JOIN Venta Vn ON V.Id_vendedor = Vn.fk_id_vendedor
   JOIN DetalleVenta DV ON Vn.Id_venta = DV.fk_id_venta
   JOIN Producto P ON DV.fk_id_producto = P.Id_producto
GROUP BY
   E.nombre_empresa,
   P.nombre_producto;
```

12. 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./

```
DELIMITER //

CREATE PROCEDURE ConsolidarVentasUltimos6Meses()
BEGIN
   DECLARE fecha_limite DATE;
   -- Calcular la fecha límite hace 6 meses desde la fecha actual
   SET fecha_limite = DATE_SUB(CURRENT_DATE(), INTERVAL 6 MONTH);
   -- Eliminar datos antiguos del consolidado
   DELETE FROM ConsolidadoVentas WHERE fecha_venta < fecha_limite;

   -- Insertar datos actualizados del consolidado
```

```
    INSERT INTO ConsolidadoVentas (Empresa, Producto, Cantidad, ValorTotal)
    SELECT
        E.nombre_empresa AS Empresa,
        P.nombre_producto AS Producto,
        SUM(DV.cantidad) AS Cantidad,
        SUM(DV.valor_total) AS ValorTotal
    FROM
        Empresa E
        JOIN Vendedor V ON E.Id_empresa = V.fk_id_empresa
        JOIN Venta Vn ON V.Id_vendedor = Vn.fk_id_vendedor
        JOIN DetalleVenta DV ON Vn.Id_venta = DV.fk_id_venta
        JOIN Producto P ON DV.fk_id_producto = P.Id_producto
    WHERE
        Vn.fecha >= fecha_limite
    GROUP BY
        E.nombre_empresa,
        P.nombre_producto;
END//


DELIMITER ;
```

13. Para qué sirve el entityManager y cuáles son los métodos básicos que este provee.

R./

El entityManager es una interfaz proporcionada por JPA (Java Persistence API) que se utiliza para realizar operaciones de persistencia en entidades gestionadas por un contexto de persistencia.

Sus métodos principales son:

- **persist(Object entity):** Este método se utiliza para agregar una nueva entidad al contexto de persistencia y guardarla en la base de datos.
- **find(Class<T> entityClass, Object primaryKey):** Permite recuperar una entidad por su clave primaria.
- **getReference(Class<T> entityClass, Object primaryKey)**: Similar a find(), pero devuelve una referencia proxy a la entidad en lugar de cargarla inmediatamente desde la base de datos
- **remove(Object entity):** Elimina una entidad del contexto de persistencia y de la base de datos.
- **merge(T entity):** Actualiza una entidad persistente con los valores de una entidad proporcionada.
- **flush()**: Sincroniza el contexto de persistencia con la base de datos, forzando cualquier cambio pendiente a ser aplicado a la base de datos.
- **getTransaction():** Devuelve una instancia de EntityTransaction que permite controlar las transacciones.
- **joinTransaction():** Asocia el entityManager con la transacción actual si ya está en curso.

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

R./

- for:

```
for (int i = 0; i < arrayList.length; i++) {
   System.out.println(arrayList[i]);
}
```

- while:

```
int[] arrayList= {1, 2, 3, 4, 5};

int i = 0;
while (i < arrayList.length) {
   System.out.println(arrayList[i]);
   i++;
}
```

- for-each:

```
int[] arrayList= {1, 2, 3, 4, 5};


for (int elemento : arrayList) {

   System.out.println(elemento);

}
```

15. Qué es un patrón de diseño.

R/

Son soluciones habituales a problemas comunes en el diseño de software. Cada patrón es como un plano que se puede personalizar para resolver un problema de diseño particular de tu código.

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

R/

```
<estructura>
    <pais nombre="País 1">
        <departamento nombre="Departamento 1">
            <ciudad nombre="Ciudad 1">
                <barrio nombre="Barrio 1"/>
                <barrio nombre="Barrio 2"/>
            </ciudad>
            <ciudad nombre="Ciudad 2">
                <barrio nombre="Barrio 3"/>
                <barrio nombre="Barrio 4"/>
            </ciudad>
        </departamento>
        <departamento nombre="Departamento 2">
            <ciudad nombre="Ciudad 3">
                <barrio nombre="Barrio 5"/>
                <barrio nombre="Barrio 6"/>
            </ciudad>
            <ciudad nombre="Ciudad 4">
                <barrio nombre="Barrio 7"/>
                <barrio nombre="Barrio 8"/>
            </ciudad>
        </departamento>
    </pais>
    <pais nombre="País 2">
        <departamento nombre="Departamento 3">
            <ciudad nombre="Ciudad 5">
                <barrio nombre="Barrio 9"/>
```

```
                <barrio nombre="Barrio 10"/>
            </ciudad>
            <ciudad nombre="Ciudad 6">
                <barrio nombre="Barrio 11"/>
                <barrio nombre="Barrio 12"/>
            </ciudad>
        </departamento>
        <departamento nombre="Departamento 4">
            <ciudad nombre="Ciudad 7">
                <barrio nombre="Barrio 13"/>
                <barrio nombre="Barrio 14"/>
            </ciudad>
            <ciudad nombre="Ciudad 8">
                <barrio nombre="Barrio 15"/>
                <barrio nombre="Barrio 16"/>
            </ciudad>
        </departamento>
    </pais>
</estructura>
```

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**

R/
Aclaración: la tabla persona hace referencia al vendedor es decir quien hace la recarga.

C → R/ a continuación diagramas
Diagrama de clase.

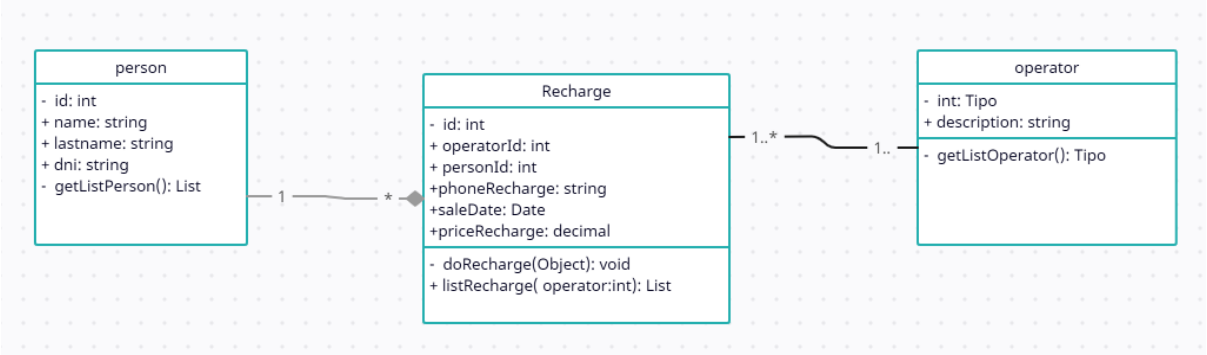


Diagrama de secuencia

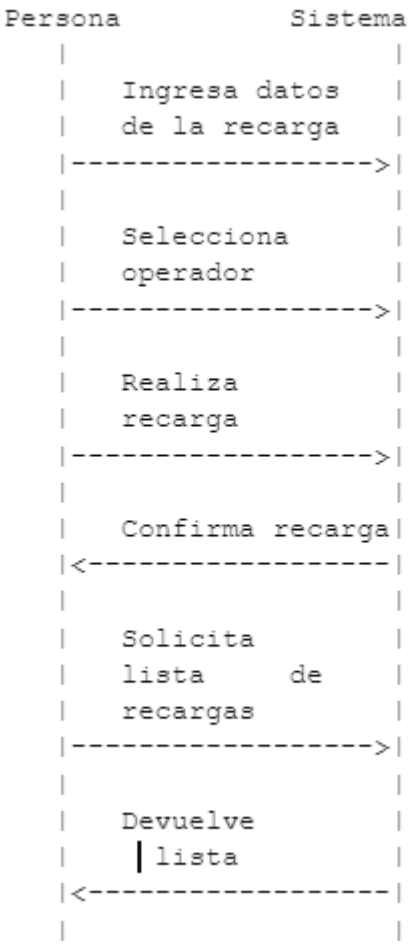


Diagrama relacional

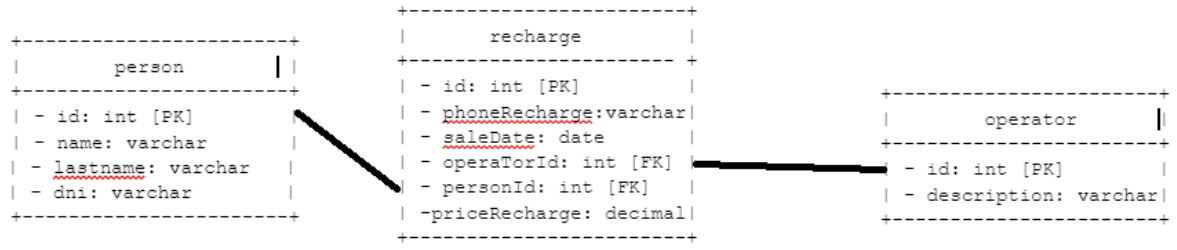
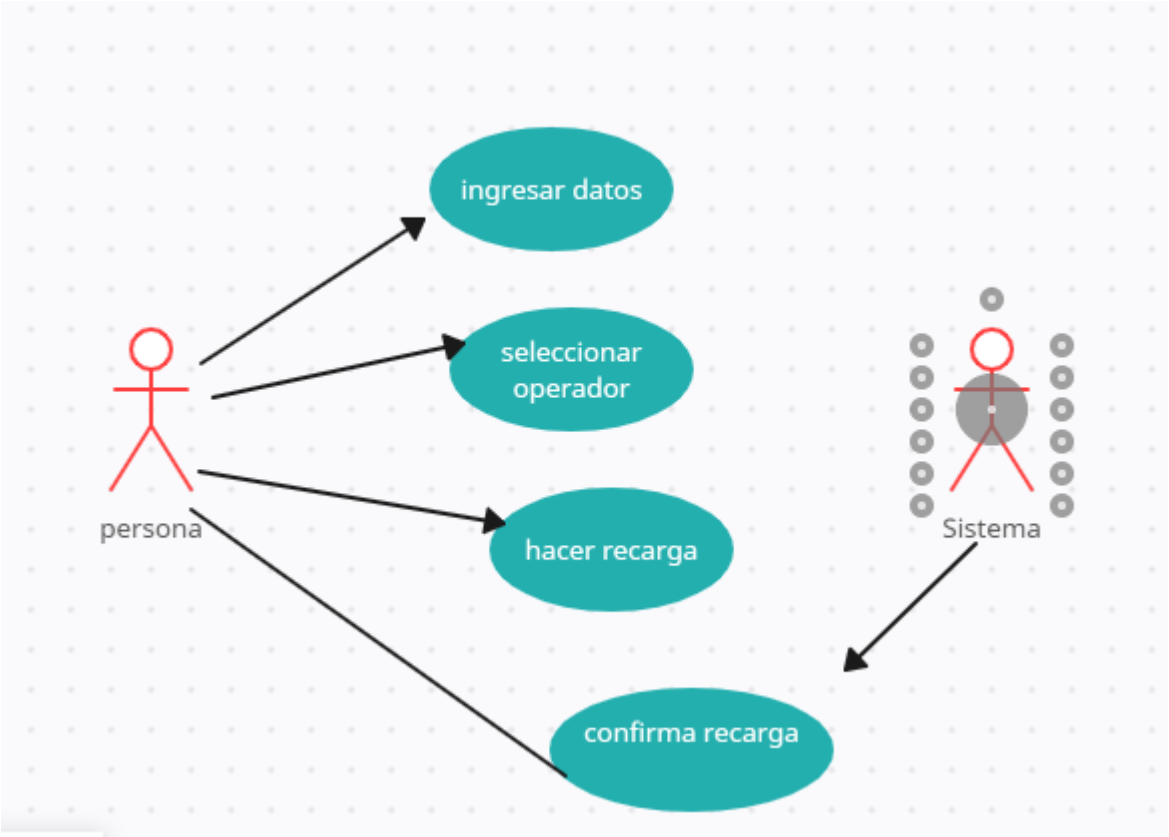


Diagrama de caso de uso



d → R/ A continuación link del repositorio de git.
https://github.com/YuverHurtadoP/recargas_spring_boot_angular/tree/master