  **PRUEBA TÉCNICA**

**Nombre**: **Fecha**:

**Cedula**:

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

* **Defina los siguientes conceptos:**

**Clase**: Es una plantilla o blueprint que define las propiedades y comportamientos (métodos) que tendrán los objetos creados a partir de ella.

**Objeto**: Es una instancia de una clase. Representa un elemento concreto que tiene estado (atributos) y comportamiento (métodos).

**Herencia**: Es un mecanismo que permite crear una nueva clase a partir de una clase existente, heredando sus atributos y métodos. Esto promueve la reutilización de código.

Polimorfismo: Es la capacidad de una clase de tener múltiples formas. Se refiere a la posibilidad de invocar métodos que pueden comportarse de diferentes maneras según el objeto que los llama.

**Sobrecarga**: Es la capacidad de definir múltiples métodos en la misma clase con el mismo nombre, pero con diferentes parámetros (tipos o cantidad).

**Paquete**: Es un contenedor que agrupa clases relacionadas para organizar mejor el código y evitar conflictos de nombres entre clases.

**Framework**: Es un conjunto de herramientas y bibliotecas que proporcionan una estructura estándar para desarrollar aplicaciones, facilitando la implementación de ciertas funcionalidades.

* **Asumiendo que existe una clase publica 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:**

* 1500
* **Para qué sirve el archivo MANIFEST.MF en una aplicación y donde está ubicado.**

R./

El archivo **MANIFEST.MF** en una aplicación es un archivo que contiene metadatos sobre el JAR (Java ARchive) que lo contiene. Se encuentra ubicado en el directorio META-INF dentro del archivo JAR. Sirve para definir información como la versión del JAR, las dependencias, la clase principal para ejecutar y otros atributos relevantes.

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

**R./ En una aplicación JSF, las reglas de navegación se definen en el archivo faces-config.xml o utilizando anotaciones en los Managed Beans (Backing Beans). Los Backing Beans son clases que actúan como intermediarias entre la vista (páginas XHTML) y la lógica de negocio.**

* **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./**

**Para implementar un filtro que verifique que el usuario esté autenticado cada vez que se solicite una página, se deben construir:**

**Un filtro (una clase que implemente javax.servlet.Filter).**

**Modificar el archivo web.xml para registrar el filtro y especificar las URL que debe interceptar.**

* **Mencione los 7 tipos de datos primitivos en Java:**

**R./**

**byte,short,int,long,float,double,boolean.**

* 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:**

* **6**

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

**R./**

**Un EJB (Enterprise Java Bean) es un componente de software que encapsula la lógica de negocio en una aplicación J2EE. Los principales tipos son:**

**Session Beans: Representan la lógica de negocio.**

**Entity Beans: Representan los datos persistentes.**

**Message-driven Beans: Procesan mensajes de manera asíncrona.**

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

**R./**

**El tipo de EJB que se puede mapear a una tabla de base de datos es el Entity Bean.**

* **Defina y de un escenario de aplicación de los siguientes patrones de diseño:**

**Singletón, Fachada, Fábrica Abstracta.**

**Singleton**: Asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella. Ejemplo: conexión a la base de datos.

**Fachada**: Proporciona una interfaz simplificada a un conjunto de interfaces en un subsistema, ocultando su complejidad. Ejemplo: un sistema de gestión de pedidos.

**Fábrica Abstracta**: Define una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas.

* **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,

dv.cantidad AS Cantidad,

dv.valor\_total AS Valor\_Total

FROM

DetalleVenta dv

JOIN

Venta v ON dv.fk\_id\_venta = v.id\_venta

JOIN

Producto p ON dv.fk\_id\_producto = p.id\_producto

JOIN

Vendedor ven ON v.fk\_id\_vendedor = ven.id\_vendedor

JOIN

Empresa e ON ven.fk\_id\_empresa = e.id\_empresa;

* **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 ConsolidarVentasUltimos6Meses()

BEGIN

SELECT

v.fk\_id\_vendedor,

SUM(v.valor\_total) AS TotalVentas

FROM

Venta v

WHERE

v.fecha >= DATE\_SUB(CURDATE(), INTERVAL 6 MONTH)

GROUP BY

v.fk\_id\_vendedor;

END;

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

**R./**

**EntityManager** es una interfaz que permite interactuar con el contexto de persistencia en JPA. Provee métodos básicos como:

**persist():** Para insertar una nueva entidad.

**merge():** Para actualizar una entidad existente.

**remove():** Para eliminar una entidad.

**find():** Para buscar una entidad por su clave primaria.

**createQuery():** Para crear y ejecutar consultas.

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

R./

for (int i = 0; i < arreglo.length; i++) {

System.out.println(arreglo[i]);

}

for (Tipo elemento : arreglo) {

System.out.println(elemento);

}

int i = 0;

while (i < arreglo.length) {

System.out.println(arreglo[i]);

i++;

}

* **Qué es un patrón de diseño.**

**R./**

Un patrón de diseño es una solución general y reusable a un problema recurrente en el diseño de software. Proporciona un enfoque estandarizado y puede ser aplicado a problemas específicos en el contexto del desarrollo.

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

**R./**

**<Paises>**

**<Pais nombre="Colombia">**

**<Departamentos>**

**<Departamento nombre="Antioquia">**

**<Ciudades>**

**<Ciudad nombre="Medellín">**

**<Barrios>**

**<Barrio nombre="El Poblado"/>**

**<Barrio nombre="Laureles"/>**

**</Barrios>**

**</Ciudad>**

**<Ciudad nombre="Bello">**

**<Barrios>**

**<Barrio nombre="Niquía"/>**

**<Barrio nombre="Zaragoza"/>**

**</Barrios>**

**</Ciudad>**

**</Ciudades>**

**</Departamento>**

**</Departamentos>**

**</Pais>**

**</Paises>**

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