2do. Parcial JAVA Plan 2024 - Tema A

Complete las respuestas y presione ***"Enviar"***. Solo se aceptará un sólo envío. Controle que sus datos e email sean correctos, caso contrario no recibirá devolución a vuelta de correo. Código ***fuente igual o repetido,*** calificará 0 (cero) en alumnos involucrados, reservándose la Cátedra la aplicación del Reglamento de Estudio vigente. Asegúrese que su código fuente compile en la IDE de referencia. A criterio de la Cátedra, podrán realizarse *"Coloquios"* sobre el código fuente presentado.-

yamilhidalgo@gmail.com Cambiar de cuenta

Borrador guardado

* Indica que la pregunta es obligatoria

Correo *

[redacted]

Ejercicio 4: Agregar elemento

10 puntos

En base al proyecto Concesionario:
https://github.com/facundouferer/CursoDeJava/tree/Desarrollo/src/Parciales/Parcial2025/Segundo/Concesionario

Implemente en la clase Inventario el método agregarAuto(Auto auto).

Este método debe:

- Verificar si en la colección interna de autos ya existe un auto con la misma patente que el auto recibido como parámetro.

- Si no existe, debe agregar el auto a la lista y devolver true.

- Si ya existe un auto con esa patente, no debe agregarlo y debe devolver false.

@param auto El auto a agregar al inventario

@return true si el auto fue agregado exitosamente, false si ya existe un auto con esa patente
/
public boolean agregarAuto(Auto auto) {
// Verificar si ya existe un auto con la misma patente
for (Auto autoExistente : autos) {
if (autoExistente.getPatente().equals(auto.getPatente())) {
// Ya existe un auto con esa patente, no se agrega
return false;
}
}

// No existe un auto con esa patente, se agrega a la lista
autos.add(auto);
return true;
}

Nombres y Apellido - Legajo *

27044

Dada la imagen adjunta, seleccione la afirmación INCORRECTA. 1 punto

a. Únicamente las clases que implementan la interfaz List permiten el uso de iteradores.
b. Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
c. Un iterador permite recorrer cualquier tipo de colección hacia adelante utilizando el método next() combinado con el método hasNext() para comprobar si se ha alcanzado el final de la colección.
d. Una colección puede recorrerse tanto con un iterador como con un ciclo for-each. Ambas formas son equivalentes.

- [x] a
- [ ] b
- [ ] c
- [ ] d

Responda según imagen. 2 puntos

Pregunta: Necesita crear una clase que almacene como elemento base de la misma objetos únicos. No se necesita que guarden orden alguno, pero sí que no se repitan. ¿Qué interfaz sería la más apropiada para este fin?

a. List.
b. Map.
c. Vector.
d. Set.

- [x] a
- [ ] b
- [ ] c
- [ ] d

Dado el código en imagen adjunta, cuál es el resultado por Consola? 3 puntos

```
1. public class Twine {
2.   public static void main(String[] args) {
3.     String s = "";
4.     StringBuffer sb1 = new StringBuffer("hi");
5.     StringBuffer sb2 = new StringBuffer("hi");
6.     StringBuffer sb3 = new StringBuffer(sb2);
7.     StringBuffer sb4 = sb3;
8.     if(sb1.equals(sb2)) s += "1 ";
9.     if(sb2.equals(sb3)) s += "2 ";
10.     if(sb3.equals(sb4)) s += "3 ";
11.     String s2 = "hi";
12.     String s3 = "hi";
13.     String s4 = s3;
14.     if(s2.equals(s3)) s += "4 ";
15.     if(s3.equals(s4)) s += "5 ";
16.     System.out.println(s);
17.   }
18. }
```

- [ ] 1 3
- [ ] 1 5
- [ ] 1 2 3
- [ ] 1 4 5
- [x] 3 4 5
- [ ] 1 3 4 5
- [ ] 1 2 3 4 5

Ejercicio 2: Excepciones 15 puntos

En base al proyecto Concesionario que se encuentra en la URL: https://github.com/facundouferer/CursoDeJava/tree/Desarrollo/src/Parciales/Parcial2025/Segundo/Concesionario

Cree una excepción que debe lanzarse cuando se intente crear un objeto Auto con una cantidad de puertas menor a 3.

La excepción debe ser verificada, por lo tanto el código que construye un Auto deberá manejarla con try/catch o declararla con throws.

La clase de la excepción debe incluir:

Un constructor que reciba un mensaje personalizado.

Un constructor por defecto con un mensaje predeterminado indicando que un auto debe tener al menos 3 puertas.

```
package Parciales.Parcial2025.Segundo.Concesionario;

import
Parciales.Parcial2025.Segundo.Concesionario.excepciones.CantidadPuertasInvalidaExcepti
on;

/**
*/
public class Auto {

private String marca;
private String modelo;
private int anio;
private int cantidadPuertas;
private double precio;
private String color;

/**

Constructor que crea un nuevo Auto.

Valida que la cantidad de puertas sea al menos 3.

@param marca La marca del auto

@param modelo El modelo del auto

@param anio El año de fabricación

@param cantidadPuertas La cantidad de puertas del auto

@param precio El precio del auto

@param color El color del auto

@throws CantidadPuertasInvalidaException Si la cantidad de puertas es menor a 3
*/
public Auto(String marca, String modelo, int anio, int cantidadPuertas,
double precio, String color) throws CantidadPuertasInvalidaException {

// Validación de cantidad de puertas
if (cantidadPuertas < 3) {
throw new CantidadPuertasInvalidaException(
"Error: No se puede crear un auto con " + cantidadPuertas +
" puertas. Un auto debe tener al menos 3 puertas."
);
}

this.marca = marca;
this.modelo = modelo;
this.anio = anio;
this.cantidadPuertas = cantidadPuertas;
```

```
this.precio = precio;
this.color = color;
}

// Getters y Setters

public String getMarca() {
return marca;
}

public void setMarca(String marca) {
this.marca = marca;
}

public String getModelo() {
return modelo;
}

public void setModelo(String modelo) {
this.modelo = modelo;
}

public int getAnio() {
return anio;
}

public void setAnio(int anio) {
this.anio = anio;
}

public int getCantidadPuertas() {
return cantidadPuertas;
}

/**

Establece la cantidad de puertas del auto.
Valida que sea al menos 3.
@param cantidadPuertas La nueva cantidad de puertas
@throws CantidadPuertasInvalidaException Si la cantidad es menor a 3
*/
public void setCantidadPuertas(int cantidadPuertas) throws
CantidadPuertasInvalidaException {
if (cantidadPuertas < 3) {
throw new CantidadPuertasInvalidaException();
}
this.cantidadPuertas = cantidadPuertas;
}
public double getPrecio() {
return precio;
}
```

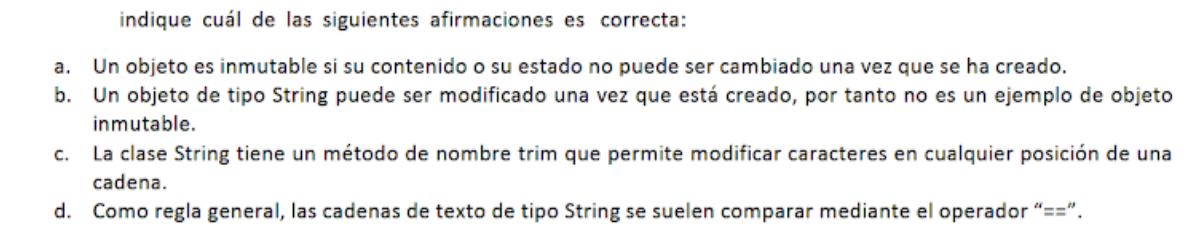
```
public void setPrecio(double precio) {
this.precio = precio;
}

public String getColor() {
return color;
}

public void setColor(String color) {
this.color = color;
}

@OverRide
public String toString() {
return String.format("Auto [Marca: %s, Modelo: %s, Año: %d, Puertas: %d, " +
"Color: %s, Precio: $%.2f]",
marca, modelo, anio, cantidadPuertas, color, precio);
}
}
```

Marque la afirmacion CORRECTA, según imagen adjunta. 2 puntos



indique cuál de las siguientes afirmaciones es correcta:

a. Un objeto es inmutable si su contenido o su estado no puede ser cambiado una vez que se ha creado.
b. Un objeto de tipo String puede ser modificado una vez que está creado, por tanto no es un ejemplo de objeto inmutable.
c. La clase String tiene un método de nombre trim que permite modificar caracteres en cualquier posición de una cadena.
d. Como regla general, las cadenas de texto de tipo String se suelen comparar mediante el operador "==".

- [x] a
- [ ] b
- [ ] c
- [ ] d

Dado el código fuente en imagen adjunta, cual sería la salida por consola? 3 puntos

```
1. public class Incognita {
2. public static int metodoIncognita(String input) {
3.    int count = 0;
4.    int length = input.length();
5.    int i = 0;
6.
7.    String lowercase = input.toLowerCase();
8.    while(i < length) {
9.          switch(lowercase.charAt(i)) {
10.               case 'a':
11.               case 'e':
12.               case 'i':
13.               case 'o':
14.               case 'u':
15.                    count++;
16.         }
17.         i++;
18.   }
19. return count;
20. }
21.
22. public static void main(String [] args) {
23.   int x = metodoIncognita("Otorrinolaringologo");
24.   System.out.print(x);

25. }
26. }
```

- [ ] 0
- [x] 9
- [ ] 19
- [ ] 20

Dado el código en imagen, cual sería la salida por Consola? 5 puntos

```
2. public class Checkout2 implements Runnable {
3.   void doStuff() { }
4.   synchronized void doSynch() {
5.      try { Thread.sleep(1000); }
6.      catch (Exception e) { System.out.print("e "); }
7.   }
8.   public static void main(String[] args) {
9.     long start = System.currentTimeMillis();
10.    new Thread(new Checkout2()).start();
11.    Thread t1 = new Thread(new Checkout2());
12.    t1.start();
13.    try { t1.join(); }
14.    catch (Exception e) { System.out.print("e "); }
15.    System.out.println("elapsed: "
                          + (System.currentTimeMillis() - start));
16.  }
17.  public void run() {
18.    for(int j = 0; j < 4; j++) {
19.      doStuff();
20.      try { Thread.sleep(1000); }
21.      catch (Exception e) { System.out.print("e "); }
22.      doSynch();
23. } } }
```

- [ ] Compilación falla.
- [x] El tiempo transcurrido sería de alrededor de 8 segundos.
- [ ] El tiempo transcurrido sería de alrededor de 9 segundos.
- [ ] El tiempo transcurrido sería de alrededor de 12 segundos.
- [ ] Ninguna de las anteriores.

Seleccione la opción CORRECTA, según la afirmación en la imagen : 1 punto

Pregunta: Un **Set** es una estructura:

a. Que almacena cada elemento individual una sola vez como mínimo. No mantiene un orden específico.
b. Que almacena cada elemento individual una sola vez como mínimo. Mantiene un orden específico.
c. Que almacena cada elemento individual una sola vez como máximo. No mantiene un orden específico.
d. Que almacena cada elemento individual una sola vez como máximo. Mantiene un orden específico.

- [ ] a
- [ ] b
- [x] c
- [ ] d

Dado el código en imagen adjunta, cual sería su salida por consola? 5 puntos

```
public class test {
    public static void main(String args[]) {
        int i=1, j=1;
        try {
            i++;
            j--;
            if(i == j)
                i++;
        }
        catch(ArithmeticException e) {
            System.out.print(0);
        }
        catch(ArrayIndexOutOfBoundsException e) {
            System.out.print(1);
        }
        catch(Exception e) {
            System.out.print(2);
        }
        finally {
            System.out.print(3);
        }
        System.out.print(“,4”);
    }
}
```

a. 0,4
b. 1,4
c. 2,4
d. 3,4

- [ ] a
- [ ] b
- [ ] c
- [x] d

Seleccione opción CORRECTA, según imagen. 2 puntos

Pregunta: Dado el siguiente fragmento de código, indique cuál es la salida de su compilación/ejecución:

```
1. String nombre = null;
2. File file = new File("/folder", nombre);
3. System.out.print(file.exists());
```

a. true
b. false
c. NullPointerException en línea 2.
d. NullPointerException en línea 3.

- [ ] a
- [x] b
- [ ] c
- [ ] d

Dado el código en imagen adjunta, cual sería el resultado por Consola? 5 puntos

```
2. class Noodle {
3.   String name;
4.   Noodle(String n) { name = n; }
5. }
6. class AsianNoodle extends Noodle {
7.   public boolean equals(Object o) {
8.     AsianNoodle n = (AsianNoodle)o;
9.     if(name.equals(n.name))  return true;
10.    return false;
11.  }
12.  public int hashCode() { return name.length(); }
13.  AsianNoodle(String s) { super(s); }
14. }
15. public class Soba extends AsianNoodle {
16.   public static void main(String[] args) {
17.     Noodle n1 = new Noodle("bob");  Noodle n2 = new Noodle("bob");
18.     AsianNoodle a1 = new AsianNoodle("fred");
19.     AsianNoodle a2 = new AsianNoodle("fred");
20.     Soba s1 = new Soba("jill");  Soba s2 = new Soba("jill");
21.     System.out.print(n1.equals(n2) + " " + (n1 == n2) + " | ");
22.     System.out.print(a1.equals(a2) + " " + (a1 == a2) + " | ");
23.     System.out.println(s1.equals(s2) + " " + (s1 == s2));
24.   }
25.   Soba(String s) { super(s); }
26. }
```

- [ ] Compilación falla.
- [ ] tru e true | true true | true true
- [x] true false | true false | true false
- [ ] false false | true false | true false
- [ ] false false | true false | false false
- [ ] false false | false false | false false

Marque la opción CORRECTA, según la imagen adjunta. 3 puntos

a. Las colecciones de objetos son objetos que pueden almacenar un número predeterminado e invariable de otros objetos.
b. Un iterador es un objeto que proporciona funcionalidad para recorrer todos los elementos de una colección.
c. Un ciclo consiste en la escritura repetida de un bloque de sentencias.
d. Un arreglo (array) es un tipo especial de colección que puede almacenar un número variable de elementos.

- [ ] a
- [ ] b
- [x] c
- [ ] d

Dado el código en imagen adjunta, cual sería el resultado? 3 puntos

```
2. public class Tshirt extends Thread {
3.   public static void main(String[] args) {
4.     System.out.print(Thread.currentThread().getId() + " ");
5.     Thread t1 = new Thread(new Tshirt());
6.     Thread t2 = new Thread(new Tshirt());
7.     t1.start();
8.     t2.run();
9.   }
10.   public void run() {
11.     for(int i = 0; i < 2; i++)
12.       System.out.print(Thread.currentThread().getId() + " ");
13. } }
```

- [ ] No se produce ninguna salida.
- [ ] 1 1 9 9 1
- [ ] 1 2 9 9 2
- [ ] 1 9 9 9 9
- [ ] Se lanza una Excepción durante la ejecución.
- [x] Compilación falla debido a un error en línea 4.
- [ ] Compilación falla debido a un error en línea 8.

Responda según imagen. 2 puntos

Pregunta: Dado el siguiente código

```
String c1=new String("Hola");
String c2=new String("Mundo");
if (……)
      System.out.println("Ambas cadenas son iguales");
else
      System.out.println("Ambas cadenas no son iguales");
```

¿Cuál de las siguientes opciones debería ponerse en la línea de puntos para llevar a cabo la comparación de las cadenas c1 y c2 en función de la salida proporcionada por el programa?

a. c1==c2
b. c1.equals(c2)
c. c1.compareTo(c2)>=0
d. c1=c2

- [ ] a
- [x] b
- [x] c
- [ ] d

Responda según imagen. 2 puntos

Pregunta: Dada la siguiente declaración:

```
Map < String, Double > map = new HashMap < String, Double > ();
```

¿Cuál de las siguientes opciones es correcta?

a. `map.add( " pi " , 3.14159);`
b. `map.add( " e " , 2.71828D);`
c. `map.add( " log(1) " , new Double(0.0));`
d. Ninguna de las anteriores

- [ ] a
- [ ] b
- [ ] c
- [x] d

Responda según imagen.

4 puntos

Pregunta: Dado el siguiente código, ¿Cuál de las siguientes afirmaciones es correcta?

```
Set < Object > objetos = new HashSet<Object>();
String obj1 = "JAVA";
int obj2 = 5;
Boolean obj3 = new Boolean(true);
objetos.add(obj3);
objetos.add(obj1);
objetos.add(obj2);
objetos.add(obj3);
for(Object object : objetos) {
System.out.print(object);
```

a. Error en tiempo de ejecución.
b. Se muestran por pantalla *JAVA 5* y *true* en un orden no determinado.
c. Se muestran por pantalla *JAVA 5* y *true* en el orden exacto en el que fueron insertadas en la colección.
d. Se muestran por pantalla *JAVA 5* y *true* en un orden no determinado y, además, "true" se muestra dos veces.

- [x] a
- [x] b
- [x] c
- [ ] d

Seleccione la opción CORRECTA, según imagen. 3 puntos

Pregunta: ¿Cual es el resultado del siguiente programa?

```
1. public class ComparadorRaro {
2. private Integer x;
3.
4. public boolean compare(int y) {
5.     return x == y;
6. }
7.
8. public static void main(String [] args) {
9.     ComparadorRaro u = new ComparadorRaro();
10.    if(u.compare(21)) {
11.            System.out.println(”true”);
12.    } else {
13.            System.out.println(”false”);
14.    }
15. }
16. }
```

a. true
b. false
c. Error de compilacion en la línea 5.
d. La línea 5 lanza una excepción NullPointerException

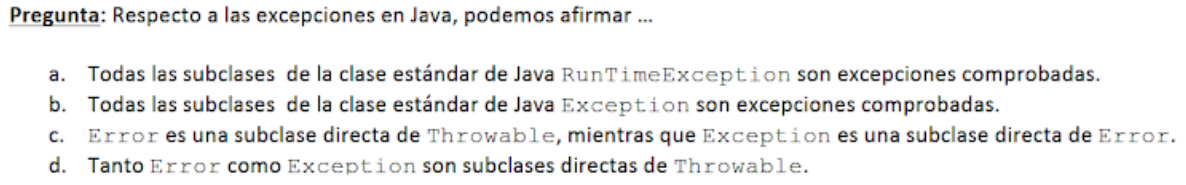
- [ ] a
- [ ] b
- [ ] c
- [x] d

Dado el código en imagen, cual sería el resultado? 3 puntos

```
2. public class Maize {
3.   public static void main(String[] args) {
4.     String s = "12";
5.     s.concat("ab");
6.     s = go(s);
7.     System.out.println(s);
8.   }
9.   static String go(String s) {
10.    s.concat("56");
11.    return s;
12. } }
```

- [ ] ab
- [x] 12
- [ ] ab56
- [ ] 12ab
- [ ] 1256
- [ ] 12ab56
- [ ] Compilación falla.

Según imagen adjunta, seleccione la opción CORRECTA. 1 punto



- [ ] a
- [ ] b
- [ ] c
- [x] d

Ejercicio 3: Analizar la Igualdad

10 puntos

En base al proyecto Concesionario: https://github.com/facundouferer/CursoDeJava/tree/Desarrollo/src/Parciales/Parcial2025/Segundo/Concesionario

Implementar los métodos para determinar si dos vehículos se consideran iguales únicamente si tienen la misma patente.

/*Implementar los métodos para determinar si dos vehículos se consideran iguales únicamente si tienen la misma patente

Implementar los métodos para determinar si dos vehículos se consideran iguales únicamente si tienen la misma patente

Implementación de métodos equals() y hashCode() para la clase Vehiculo
Dos vehículos se consideran iguales si tienen la misma patente
*/
// En la clase Vehiculo, agregar estos métodos:

/**

Compara este vehículo con otro objeto para determinar si son iguales.

Dos vehículos son iguales si y solo si tienen la misma patente.

@param obj El objeto a comparar

@return true si los vehículos tienen la misma patente, false en caso contrario
*/
@OverRide
public boolean equals(Object obj) {
// Verificar si es la misma referencia
if (this == obj) {
return true;
}

// Verificar si el objeto es null
if (obj == null) {
return false;
}

// Verificar si son de la misma clase
if (getClass() != obj.getClass()) {
return false;
}

// Castear a Vehiculo
Vehiculo other = (Vehiculo) obj;

// Comparar las patentes
if (patente == null) {
return other.patente == null;
}

return patente.equals(other.patente);
}

/**

Genera un código hash para el vehículo basado en su patente.

Este método debe ser sobrescrito junto con equals() para mantener

el contrato general de hashCode.

@return El código hash del vehículo
*/
@OverRide
public int hashCode() {
// Usar un número primo como base
final int prime = 31;
int result = 1;

// Calcular el hash basado en la patente
result = prime * result + ((patente == null) ? 0 : patente.hashCode());

return result;
}

/**

Alternativa más moderna usando Objects.equals() y Objects.hash()
(Disponible desde Java 7)
*/
// Importar al inicio de la clase:
// import java.util.Objects;

/*
@OverRide
public boolean equals(Object obj) {
if (this == obj) return true;
if (obj == null) return false;
if (getClass() != obj.getClass()) return false;

Vehiculo other = (Vehiculo) obj;
return Objects.equals(patente, other.patente);
}

@OverRide
public int hashCode() {
return Objects.hash(patente);
}
*/

// ==
// EJEMPLO DE USO
// ==

public static void main(String[] args) {
// Crear vehículos con la misma patente
Vehiculo v1 = new Vehiculo();
v1.setPatente("ABC123");

```
v1.setMarca("Ford");
v1.setModelo("Focus");

Vehiculo v2 = new Vehiculo();
v2.setPatente("ABC123");
v2.setMarca("Chevrolet");
v2.setModelo("Cruze");

Vehiculo v3 = new Vehiculo();
v3.setPatente("XYZ789");
v3.setMarca("Ford");
v3.setModelo("Focus");

// Comparaciones
System.out.println("v1.equals(v2): " + v1.equals(v2)); // true (misma patente)
System.out.println("v1.equals(v3): " + v1.equals(v3)); // false (patente diferente)
System.out.println("v2.equals(v3): " + v2.equals(v3)); // false (patente diferente)

// HashCode
System.out.println("\nHashCodes:");
System.out.println("v1.hashCode(): " + v1.hashCode());
System.out.println("v2.hashCode(): " + v2.hashCode()); // Igual a v1
System.out.println("v3.hashCode(): " + v3.hashCode()); // Diferente

// Uso en colecciones
Set<Vehiculo> vehiculos = new HashSet<>();
vehiculos.add(v1);
vehiculos.add(v2); // No se agregará (mismo hashCode y equals)
vehiculos.add(v3);

System.out.println("\nCantidad de vehículos únicos en Set: " + vehiculos.size()); // 2
}
```

Ejercicio 1: Implementar la interface 15 puntos

En base al proyecto Concesionario:
https://github.com/facundouferer/CursoDeJava/tree/Desarrollo/src/Parciales/Parcial2025/Segundo/Concesionario

Implemente en la clase Auto el método requerido por la interface correspondiente.

El método debe calcular el precio final de venta considerando:

- Una depreciación del 5% por cada año de uso, calculada a partir del año actual y el modelo del auto.

- Un porcentaje adicional según la cantidad de puertas:

3 puertas → 30%

4 puertas → 40%

Cualquier otra cantidad → 35%

El método debe devolver el precio final aplicando primero la depreciación y luego el porcentaje adicional según las puertas.

```
import java.time.Year;

class Auto extends Vehiculo implements Ventas {
private int cantPuertas;

public Auto(String marca, int modelo, String patente, int kilometraje, int cantPuertas) throws
PuertasInsuficientesException {
    super(marca, modelo, patente, kilometraje);
    // Validar que el auto tenga al menos 3 puertas
    if (cantPuertas < 3) {
        throw new PuertasInsuficientesException(
            "Error: Un auto debe tener al menos 3 puertas. Puertas recibidas: " + cantPuertas);
    }
    this.cantPuertas = cantPuertas;
}

// Getter
public int getCantPuertas() {
    return cantPuertas;
}

// Setter
public void setCantPuertas(int cantPuertas) throws PuertasInsuficientesException {
    // Validar que el auto tenga al menos 3 puertas
    if (cantPuertas < 3) {
        throw new PuertasInsuficientesException(
            "Error: Un auto debe tener al menos 3 puertas. Puertas recibidas: " + cantPuertas);
    }
    this.cantPuertas = cantPuertas;
}

// Implementación del método abstracto
@Override
public String verTipoDeVehiculo() {
    return "🚗";
}

// Implementación del método de la interfaz Ventas
@Override
public double calcularPrecioFinal(double precioBase) {
    // 1. Calcular años de uso
    int anioActual = Year.now().getValue();
    int aniosDeUso = anioActual - modelo;

    // 2. Aplicar depreciación del 5% por cada año de uso
    double depreciacionTotal = aniosDeUso * 0.05;
    double precioConDepreciacion = precioBase * (1 - depreciacionTotal);

    // 3. Determinar porcentaje adicional según cantidad de puertas
    double porcentajePuertas;
    if (cantPuertas == 3) {
        porcentajePuertas = 0.30;
```

```
    } else if (cantPuertas == 4) {
        porcentajePuertas = 0.40;
    } else {
        porcentajePuertas = 0.35;
    }

    // 4. Aplicar porcentaje adicional
    double precioFinal = precioConDepreciacion * (1 + porcentajePuertas);

    return precioFinal;
}

// Método toString
@Override
public String toString() {
    return marca + "\t" + modelo + "\t" + patente + "\t" + kilometraje + "Km\t" + cantPuertas + "
puertas";
}
```

Enviar

Página 1 de 1

Borrar formulario