DESARROLLO DE APLICACIONES WEB **CALIFICACIÓN:**
EXAMEN 2ºEV. PROGRAMACIÓN
NOMBRE: **APELLIDOS:**

1. Implementar **solo 3 de** los siguientes métodos : **(4.5 Puntos)**

(1.5 Puntos)

A) `static String` **`quitarVocales`** `( String parámetro )`

Devuelve un String a partir de la cadena parámetro con el mismo contenido pero suprimiendo las vocales.

Ejemplo:

```
quitarVocales("Hola Pepe Luis") -->  "Hl Pp Ls"
```

(1.5 Puntos)

B) `public static int []` **`operarTabla`** `( int [] valores)`

Devuelve una nueva tabla de enteros de la mitad de tamaño de la tabla de valores, donde en cada posición se almacena la suma de dos posiciones consecutivas de la tabla de valores.

Nota: Para crear una tabla podemos utilizar int resu[] = new int[tamaño] y luego hacer return resu

Ejemplo `operarTabla ({ 3,2,5,1}) → {5,6}`

`operarTabla ({2,1,5,0,8,-1}) → {3,5,7}`

(1.5 Puntos)

C) `public static String` **`superEco`** `( String cadena, int veces)`

Devuelve un *String* donde la cadena pasada como parámetro se repite tantas veces con se indique el parámetro *veces*. Si veces es 0 o negativo, la cadena a devolver sera vacía ""

Ejemplos:

```
superEco("Hola",3) → "HolaHolaHola"
superEco("Oh",2) → "OhOh"
superEco("Peligro",-3) → ""
```

(1.5 Puntos)

D) `public static boolean` **`estaIncluido`** `( int Ta[ ], int Tb[ ])`

Devuelve verdadero, si todos los elementos de Ta, aparecen en la tabla Tb, aunque estén en distintas posiciones.

Ejemplos:

```
estaIncluido ({2,3,5}, { 3, 5,7,8,2,0})  → true
estaIncluido ({ 3, 5,7,8,2,0}, {2,3,5})  →  false
estaIncluido ({ 3,5,5,1} ,{ 9,5,3,1} )   →  true
estaIncluido ({2,3,5}, { 3,-3,8,10} )    →  false
```

(2 Puntos)

2. Para desarrollar una aplicación se han diseñado la clase abstracta **Bas**e y dos Interfaces **modelo1** y **modelo2** con el siguiente contenido:

```
abstract class Base {

private int atributo1;
protected int atributo2;
public int atributo3;
static int atributo4;

public void metodo1 (){
  ...
 }
public abstract void metodo2 ();

public abstract void metodo3 ();

public final void metodo4() {
  ...
 }
protected void metodo5 () {
...
 }
}
```

```
  ...
Interface  modelo1 {

void  metodoalgo1();
void  metodoalgo2();

}

Interface modelo2 {

void metodomodelo2();

}
```

Dada la definición de la clase Hija

```
class Hija extends Base implements modelo1, modelo2 {

  ...
}
```

Responder a las siguientes preguntas

1) ¿Puede estar la clase Hija en un paquete distintos de la clase Base?¿Que sería necesario utilizar?

2) ¿Que métodos tiene obligatoriamente que reescribir la clase Hija?

3) ¿Que métodos son opcionales su reescritura en la clase Hija?

4) ¿Que métodos no se pueden reescribir en la clase Hija?

5) ¿Que diferencia existe entre un atributo privado y uno protegido?

6) ¿La clase Hija puede heredar directamente de más clases además de la clase Base?

7) ¿Porqué la siguiente instrucción no sería correcta?

```
Base obj = new Base();
```

8) En la clase Base, ¿Cuáles son atributos de instancia y cuáles de clase?

(2.5 Puntos)

3.1 Desarrolla una clase *Cafetera* con atributos *capacidadMáxima* (la cantidad máxima de café que puede contener la cafetera) y *cantidadActual* (la cantidad actual de café que hay en la cafetera). Implementa, al menos, los siguientes métodos:

- Constructor común (sin parámetros): establece la capacidad máxima en 1000 (c.c.) y la actual en cero (cafetera vacía).
- Constructor de inicialización que sólo recibe la capacidad máxima, poniendo la actual a cero.
- Constructor de inicialización con los dos parámetros: con la capacidad máxima y la cantidad actual. Si la cantidad actual es mayor que la capacidad máxima de la cafetera, la ajustará al máximo.
- *llenarCafetera()*: pues eso, hace que la cantidad actual sea igual a la capacidad Máxima.
- *servirTaza(int cantidad)*: simula la acción de servir una taza con la capacidad indicada. Si la cantidad actual de café "no alcanza" para llenar la taza, se sirve lo que quede.
- *vaciarCafetera()*: pone la cantidad de café actual en cero.
- *agregarCafe(int cantidad)*: añade a la cafetera la cantidad de café indicada. Solo se puede llenar hasta la capacidad Máxima.

(1.5 Puntos)

3.2 Crear la clase *TestCafeteras* que realice la siguientes operaciones:

- Cree un *array* que almacene **cuatro** cafeteras de distinta de capacidad máxima.
- Llenas todas la cafeteras de café
- Realizar varias operaciones de *servirTaza y agregarCafe* con valores aleatorios de cantidad en todas las cafeteras de la tabla.
- Ordene el *array* por *capacidadActual*
- Mostrar un informe según este formato:

```
--------------------------------------
     Estado Actual de Cafeteras:
--------------------------------------
Nº Cantidad Actual    Capacidad Máxima
1     1000                 1500
2      850                 2000
3      600                 1000
4      200                 2000
```