

RESPUESTAS INVESTIGACIÓN #1

(Maza Alcalde Luisin Enrique)

Respuesta 01: PARADIGMAS DE PROGRAMACIÓN

Procedural o Imperativo:

- ✓ Este paradigma indica cómo debe solucionarse un problema especificando una secuencia de acciones a realizar a través de uno o más procesos (Algoritmos).
- ✓ Ejemplo: Se requiere preparar un refresco, hay que seguir una serie de pasos hasta llegar a la preparación final y poder beber el refresco.

Orientación a Objetos:

- ✓ Es una forma de programación imperativa.
- ✓ Expresa un programa como un conjunto de objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener, reutilizar y volver a utilizar.
- ✓ Se define como “la emulación de la realidad”. Lo primordial son los objetos, los cuales tienen características y funcionalidades, tal como en la vida real.
- ✓ Las características son: encapsulamiento, abstracción, polimorfismo y herencia.
- ✓ Ejemplo: Si se habla de una persona en la vida real, la cual tiene características como su DNI, nombre, apellidos y edad, la cual posee funciones como caminar, correr, saltar.

Programación Declarativa:

- ✓ No es necesario definir algoritmos, ya que se detalla la solución del problema en lugar de como llegar a la solución.
- ✓ La solución es alcanzada a través de mecanismos internos de control pero no se especifica exactamente como llegar a ella.
- ✓ Ejemplo: Cuando se realiza un pedido de delivery, se especifica lo que se requiere del producto, pero se desconoce cómo lo prepararan y como llegara al lugar de destino.

Programación Funcional:

- ✓ Es una forma de programación declarativa.

- ✓ Se basa en el uso de una o más funciones dentro de las cuales se pueden utilizar funciones creadas anteriormente.
- ✓ Su objetivo es dividir el programa en módulos de forma que cada uno de éstos realice una única función.
- ✓ Se basa en una entrada, proceso y salida.
- ✓ Ejemplo: Función para sumar dos números, la cual luego es utilizada en algún proceso.

Programación Lógica:

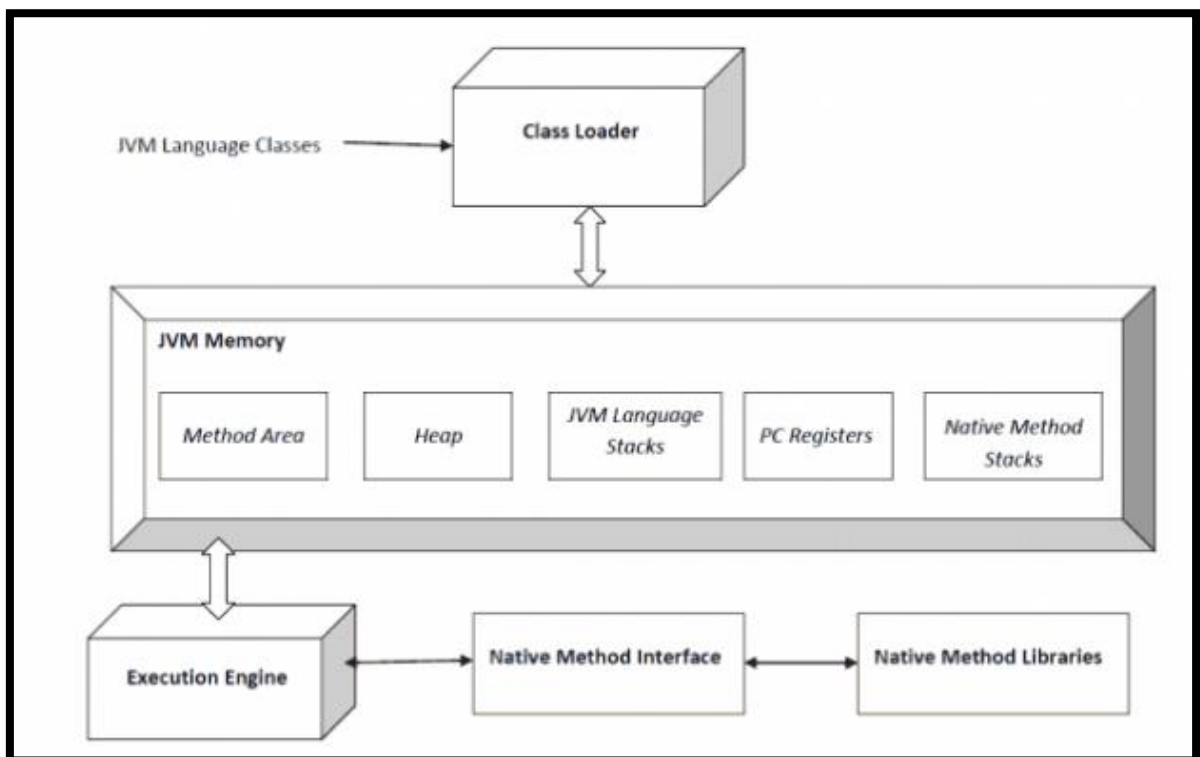
- ✓ Comprende la programación declarativa y la funcional.
- ✓ Especifica qué debe hacer el programa y no cómo hacerlo.
- ✓ Es una serie de reglas lógicas, llamadas hechos, que se aplican para deducir un resultado.
- ✓ Se emplea en aplicaciones de inteligencia artificial, sistemas donde hay interacción persona – computador y sistemas expertos.
- ✓ Ejemplo: Sistemas basados en conocimientos, los chatbots.

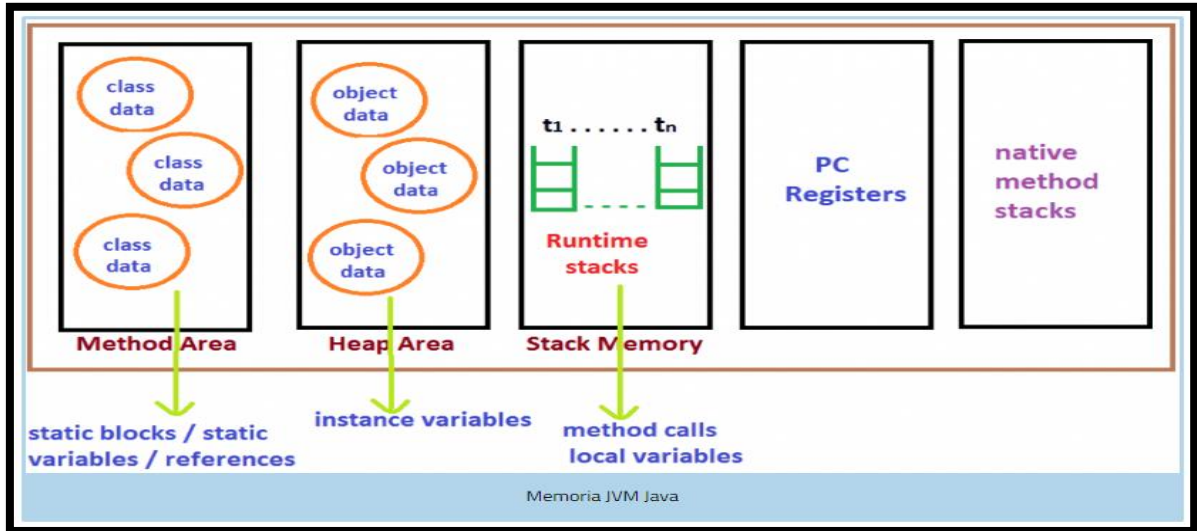
Comentario: El paradigma que llamo mi atención es la orientación a objetos (Ya que nos permite organizarnos, mantener y escribir el programa) y lógica (Creo que es el futuro, la cual hoy en día se está aplicando mucho, con los sistema basados en inteligencia artificial).

Respuesta 02: JVM

- ✓ Máquina virtual de proceso nativo, se ejecuta sobre una plataforma.
- ✓ Capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (bytecode), el cual es generado por el compilador del lenguaje Java (.class).
- ✓ La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje.
- ✓ Permite ejecutar código Java en cualquier plataforma para la que exista una JVM.
- ✓ Tareas importantes: Reservar espacio en memoria para los objetos creados, liberar la memoria no usada (Garbage Collection), asignar variables a registros y pilas.

MEMORIA JVM	
Method area:	Área de método, se almacena toda la información del nivel de clase, como nombre de la clase, nombre inmediato de la clase principal, información de métodos y variables, etc., incluidas las variables estáticas. Solo hay un área de método por JVM, y es un recurso compartido.
Heap area:	Información de todos los objetos se almacena en el área heap. También hay un área heap por JVM, es un recurso compartido.
Stack area:	Para cada subproceso, JVM crea una pila en tiempo de ejecución que se almacena aquí. Cada bloque de esta pila se llama registro de activación/marco de pila que almacena los métodos de llamadas. Todas las variables locales de ese método se almacenan en su marco correspondiente. Una vez que finaliza un hilo, JVM destruirá la pila en tiempo de ejecución. No es un recurso compartido.
PC Registers:	Almacena la dirección de la instrucción de ejecución actual de un hilo. Obviamente, cada hilo tiene registros de PC separados.
Native method stacks:	Para cada hilo, se crea una pila nativa separada. Almacena información del método nativo.





```
Persona.java  Alumno.java  Deportista.java  Test.java  TestJVM.java
Source  History  [Icons]

1  package ejemplo01;
2  import ...2 lines
4  /**
5   * Código Java para demostrar el uso del objeto Class creado por JVM
6   * @author Luisin Maza
7   */
8  public class TestJVM {
9
10     public static void main(String[] args) {
11
12         //Declarar e instanciar objeto de nombre persona, de la clase
13         //Persona
14         Persona persona = new Persona();
15
16         //Obtener el objeto Class creado por la JVM
17         Class clase = persona.getClass();
18
19         //Imprimir el tipo de objeto usando clase
20         System.out.println(clase.getName());
21
22         //Obtener todos los métodos en una matriz
23         Method m[] = clase.getDeclaredMethods();
24         for (Method method : m)
25             System.out.println(method.getName());
26
27         //Obtener todos los campos en una matriz
28         Field f[] = clase.getDeclaredFields();
29         for (Field field : f)
30             System.out.println(field.getName());
31     }
32 }
```

Respuesta 03: COLECCIÓN Y ARRAYS

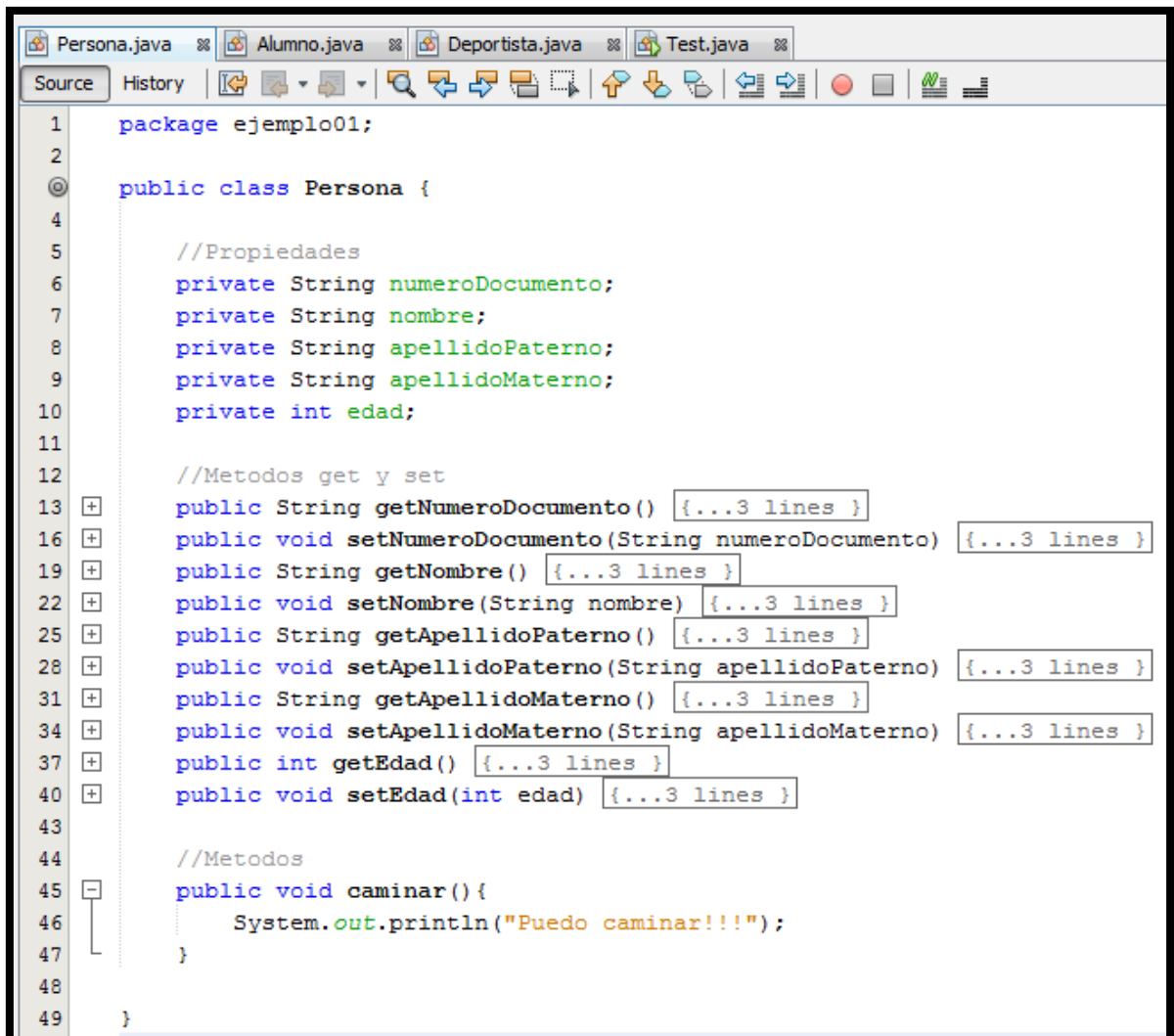
- ✓ Una colección, es un objeto que agrupa múltiples elementos en una sola unidad.
- ✓ Una colección, es todo aquello que se puede recorrer (o “iterar”) y de lo que se puede saber el tamaño.
- ✓ Las colecciones se utilizan para almacenar, recuperar, manipular y comunicar datos agregados.
- ✓ Por ejemplo: Un directorio de correo (Una lista de email), guía telefónica (Asignación de nombres a números de teléfonos).
- ✓ Diferencias:

Colección	Arrays
Tamaño dinámico.	Tamaño definido en la creación.
Se pueden insertar, modificar y eliminar elementos dinámicamente.	Se le asigna valor según tipo definido en una posición.
Cuenta con una serie de funciones.	Limitado de funciones.
Puede contener varios tipos de datos.	Solo un tipo de dato.

Respuesta 04: HERENCIA SIMPLE Y MÚLTIPLE

- ✓ Herencia es un mecanismo propio de la programación orientada a objetos.
- ✓ Nos permite crear clases a partir de otra, las cuales las unen vínculos sumamente estrechos, casi de familia.
- ✓ Ventajas: Modelado de la realidad, evitar redundancias en el código, facilitar la reutilización del código y reducir las líneas de código.
- ✓ La palabra reservada para utilizar la herencia es “extends”.
- ✓ **Herencia simple:** Una clase sólo puede tener una clase padre o superclase directa de la cual hereda todas sus propiedades y métodos.
- ✓ **Herencia múltiple:** Una clase puede heredar de dos o más clases padres o superclases.
- ✓ La herencia múltiple en java no es soportada nativamente. Sin embargo muchos desarrolladores la simulan utilizando la palabra reservada “implements” e interfaces, que sirve para implementar o cubrir una clase con respecto a otra.
- ✓ Ejemplo: Herencia simple, Clase principal Persona (Cuenta con sus propiedades y métodos, entre ellos el método “caminar”), clase hija Alumno (Cuenta con sus propiedades y métodos, entre ellos el método “saltar”, además va a heredar sus

propiedades y métodos de la clase Persona). Se tiene una clase Test de prueba donde se demuestra que el objeto de nombre alumno puede acceder a las propiedades y métodos de la clase Persona.



```
1 package ejemplo01;
2
3
4 public class Persona {
5     //Propiedades
6     private String numeroDocumento;
7     private String nombre;
8     private String apellidoPaterno;
9     private String apellidoMaterno;
10    private int edad;
11
12    //Metodos get y set
13    public String getNumeroDocumento() {...3 lines }
14
15
16    public void setNumeroDocumento(String numeroDocumento) {...3 lines }
17
18
19    public String getNombre() {...3 lines }
20
21
22    public void setNombre(String nombre) {...3 lines }
23
24
25    public String getApellidoPaterno() {...3 lines }
26
27
28    public void setApellidoPaterno(String apellidoPaterno) {...3 lines }
29
30
31    public String getApellidoMaterno() {...3 lines }
32
33
34    public void setApellidoMaterno(String apellidoMaterno) {...3 lines }
35
36
37    public int getEdad() {...3 lines }
38
39
40    public void setEdad(int edad) {...3 lines }
41
42
43
44    //Metodos
45    public void caminar(){
46        System.out.println("Puedo caminar!!!");
47    }
48
49 }
```

```
Persona.java Alumno.java Deportista.java Test.java
Source History
1 package ejemplo01;
2
3 public class Alumno extends Persona {
4
5     //Propiedades
6     private String numeroCarneEscolar;
7
8     //Metodos get y set
9     public String getNumeroCarneEscolar() {...3 lines }
10
11     public void setNumeroCarneEscolar(String numeroCarneEscolar) {...3 lines }
12
13
14
15
16     //Metodos
17     public void saltar() {
18         System.out.println("Hola puedo saltar!!!");
19     }
20
21 }
```

```
Persona.java Alumno.java Deportista.java Test.java
Source History
1 package ejemplo01;
2
3 public class Test {
4
5     public static void main(String[] args) {
6
7         //Declarar e instanciar objeto de nombre alumno, de la clase
8         //Alumno
9         Alumno alumno = new Alumno();
10
11         //Asignar valores a propiedades
12         alumno.setNumeroDocumento("12345678");
13         alumno.setNumeroCarneEscolar("9999");
14         alumno.setNombre("Luisin");
15         alumno.setApellidoPaterno("Maza");
16         alumno.setApellidoMaterno("Alcalde");
17         alumno.setEdad(15);
18
19         //Imprimir propiedades
20         System.out.println("Documento: " + alumno.getNumeroDocumento());
21         System.out.println("Carne: " + alumno.getNumeroCarneEscolar());
22         System.out.println("Nombre: " + alumno.getNombre());
23
24         //Imprimir metodos
25         alumno.caminar();
26         alumno.saltar();
27
28     }
29
30 }
```

Respuesta 05: EXPRESIONES LAMBDA

- ✓ Permite programación funcional, en donde las funciones juegan un papel fundamental.
- ✓ Para entender la expresión lambda son dos conceptos. El primero es la expresión lambda, en sí misma. El segundo es la interfaz funcional.
- ✓ Una expresión lambda es, esencialmente, un método anónimo (es decir, sin nombre).
- ✓ Una interfaz funcional es una interfaz que contiene uno y solo un método abstracto. Normalmente, este método especifica el propósito previsto de la interfaz.
- ✓ La idea de las expresiones lambda es tener un código más limpio y legible.
- ✓ Nos ayudará a construir programas más claros y más flexibles.
- ✓ Una expresión lambda se compone de dos elementos. En primer lugar de un conjunto de parámetros y en segundo lugar de una expresión que opera con los parámetros indicados.
- ✓ Sintaxis: (arg1, arg2...) -> { cuerpo}