A). Responda:

1. **¿Qué es un TAD?**

   TAD (Tipo Abstracto de Datos) es una abstracción que define un conjunto de operaciones sobre una estructura de datos, sin preocuparse por cómo se implementan esas operaciones

2. **¿Qué es un Objeto o instancia?**

   Un objeto o instancia es una entidad que se crea a partir de una clase y que representa una ocurrencia única de esa clase. Cada objeto tiene su propio estado y comportamiento, que se definen por los atributos y métodos de la clase.

3. **¿Dónde se produce el encapsulamiento?**

   El encapsulamiento es un concepto fundamental de la programación orientada a objetos que se refiere a ocultar los detalles internos de una clase y proporcionar una interfaz pública para interactuar con ella. El encapsulamiento se produce dentro de la propia clase, mediante el uso de modificadores de acceso como public, private o protected.

   1. Public: Los miembros públicos de una clase son accesibles desde cualquier parte del programa.
   2. Private: Los miembros privados de una clase sólo son accesibles desde dentro de la propia clase.
   3. Protected: Los miembros protegidos son similares a los miembros privados, pero también son accesibles desde las subclases de la clase actual

   Por lo tanto, podemos decir que el encapsulamiento ***se produce dentro de la clase, en la definición de sus atributos y métodos y su nivel de acceso, para proteger los datos y el comportamiento de la clase de la manipulación no autorizada desde el exterior.***

4. **¿Cuáles son las semejanzas y las diferencias entre atributos y métodos?**

   Las similitudes entre ambos es que tanto los atributos como los métodos se definen dentro de una clase, y están relacionados con la forma en que se comporta el objeto. Además, tanto los atributos como los métodos pueden ser públicos o privados, lo que significa que pueden o no estar disponibles fuera de la clase.

   Las diferencias principales entre ellos es que los atributos almacenan valores mientras que los métodos realizan acciones y manipulan esos valores. También, los métodos pueden tomar argumentos y devolver valores, mientras que los atributos no pueden hacerlo por sí solos. ***En resumen, los atributos representan la información del objeto, mientras que los métodos representan las acciones que se pueden realizar con ese objeto.***

5. **¿Qué es UML? ¿Y cómo se representa una clase en ese lenguaje?**

   UML (Unified Modeling Language) es un lenguaje gráfico y de modelado utilizado para representar, visualizar y documentar sistemas de software complejos.

Una clase en UML se representa mediante un rectángulo dividido en tres partes horizontales, que incluyen el nombre de la clase, los atributos y los métodos. UML es una herramienta muy útil para modelar sistemas de software complejos y comunicar el diseño de los mismos.

B). Una con flechas los calificadores de acceso con sus correspondientes:

1. public
2. private
3. protected
4. friendly -default
5. static
6. final
7. primera letra en minúscula
8. primera letra en mayúscula

No se aplica a Clases
Sólo se aplica a atributos
Sólo se aplica a Clases
Se aplica a atributos, métodos y clases

C). Realizando abstracción, escriba atributos y sus tipos, intentando que sean diferentes para cada ámbito de aplicación.

Clases	Ámbito en los que se aplica	
Comida	Para un comercio de venta	Para el experto que lo prepara
atributos	Precio: double	Ingredientes :String
	Nombre del producto: String	Tiempo de preparación: double
	Proveedor: String	Nombre de la receta: String
	Cantidad en stock: int	Temperatura de cocción: double
	Para el cliente que compra	Para la empresa que recicla los sobrantes
atributos	Precio de compra: double	Tipo de comida: String
	Forma de pago: String	Cantidad recogida: int
	Cantidad comprada: int	Destino final: String
	Calidad del producto: String	Costo de reciclaje: double

D). Responda:

1. **¿Qué es el bytecode?**

el bytecode es el lenguaje intermedio generado por el compilador de Java a partir del código fuente, que es interpretado por la Máquina Virtual de Java para ejecutar los programas escritos en Java.
Un bytecode es un conjunto de instrucciones en código de bajo nivel que es generado por un compilador a partir de un lenguaje de programación de alto nivel.

El bytecode no es código de máquina directamente ejecutable por la CPU, sino que se utiliza como una representación intermedia del programa que permite su portabilidad y su ejecución en diferentes plataformas.

El bytecode se puede ejecutar en una máquina virtual específica que se encarga de traducirlo a código de máquina en tiempo de ejecución. Algunos ejemplos de lenguajes de programación que utilizan bytecode son Java, Python y Ruby.

2. **¿Qué función cumple la JVM?**

La JVM (Java Virtual Machine) es un componente clave de la plataforma Java que se encarga de ejecutar el bytecode generado por el compilador de Java. La JVM actúa como una capa de abstracción entre el código fuente de Java y el sistema operativo subyacente, permitiendo que los programas escritos en Java se ejecuten en diferentes plataformas sin necesidad de modificar el código fuente.
.

3. ¿Cuándo se ejecuta el recolector de basura de java?

El recolector de basura de Java es un componente de la máquina virtual de Java que se encarga de liberar la memoria ocupada por objetos que ya no están en uso por la aplicación.

La ejecución del recolector de basura es una tarea automática que se realiza en segundo plano por la JVM y no puede ser controlada directamente por el programador.

El recolector de basura de Java se ejecuta cuando la JVM detecta que la memoria está empezando a escasear o que hay objetos que no se están utilizando.

F). Indique verdadero (V) o falso (F):

Un constructor…

	V/F	Afirmación	Es falso debido a…
1	F	Es el método principal para ejecutar el programa.	Su función es llenar las variables de un objeto
2	V	Crea objetos.	
3	F	Devuelve el valor de un atributo privado.	El constructor no devuelve el valor de un atributo privado. Para eso se debería hacer una función con return
4	F	Tiene sentencia return	No cuenta con sentencia return
5	V	Siempre hay uno por defecto, sin parámetros ni inicializaciones de atributos.	
6	V	Se puede sobrecargar.	
7	F	Su nombre se escribe con mayúscula.	La nomenclatura usada establece que la primera letra del nombre de la clase está en mayúscula.
8	F	Su calificador de acceso es static.	El constructor no puede ser ni static ni final.
9	V	Su tipo de devolución no se indica y corresponde a la Clase.	

```java
package primerobjeto;
public class ClaseNueva {
    public static ClaseNueva cuales[] = new ClaseNueva[100];
    public static int cuantos = 0;
    public int a;
    private int b;
    public final char letra;
    public ClaseNueva(char letra) {
        this.letra = letra;
    }
    public ClaseNueva(char letra, int a, int b) {
        this.letra = letra;
        this.a = a;
        this.b = b;
    }
    public int getB() {
        return b;
    }
    public void setB(int b) {
        this.b = b;
    }
     public void mostrarAtributos(){
        System.out.println("letra = "+letra);
        System.out.println("a = "+a+" b = "+b);
           }
    public void otro( ClaseNueva x){
        System.out.println("Capacidad de cuales: "+cuales.length);
        System.out.println("Valor de b: "+x.getB());
        System.out.println("Valor de cuantos: "+cuantos);
    }
}
```

Definición de ObjetosDeClaseNueva

```java
package primerobjeto;

public class ObjetosDeClaseNueva {

    public static void main(String[] args) {
        ClaseNueva a1 = new ClaseNueva('u',1,2);
        ClaseNueva.cuales[ClaseNueva.cuantos]=a1;
        ClaseNueva.cuantos++;

        ClaseNueva b1 = new ClaseNueva('v');
        ClaseNueva.cuales[ClaseNueva.cuantos]=b1;
        ClaseNueva.cuantos++;

        ClaseNueva c1 = new ClaseNueva('w',54,21);
        ClaseNueva.cuales[ClaseNueva.cuantos]=c1;
        ClaseNueva.cuantos++;
        for (int i=0;i<ClaseNueva.cuantos;i++)
                ClaseNueva.cuales[i].mostrarAtributos();
        System.out.println("Total objetos: "+ClaseNueva.cuantos);
    }
}
```

Ejecución del método main de ObjetosDeClaseNueva

```
run:
letra = u
a = 1 b = 2
letra = v
a = 0 b = 0
letra = w
a = 54 b = 21
Total objetos: 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Responda:

- ¿Dentro de qué método se declaran las variables estáticas y cuáles son?

Se encuentra dentro de la clase "ClaseNueva", y declara public estatic int cuantos y public static ClaseNueva

- ¿Dónde se les asigna valor?

El valor se le asigna en ClaseNueva al ser declarado

- Observando los parámetros que reciben los constructores:
- ¿Por qué hay un atributo que recibe valor en todos los constructores?

Porque está declarado como FINAL

- ¿Podría ese valor eliminarse de esos métodos constructores?

Si

- ¿Por qué?

Ya que puede estar inicializado en su declaracion, por lo que no necesitaria estar en los constructores

- ¿Qué se debe hacer con ese valor para eliminarlo de todos los constructores?

Inicializarlo en su declaracion

- Observe el contenido del método mostrarAtributos:
- ¿Es posible definirlo como estático?

no

- ¿Por qué?

porque los métodos declarados como static solo pueden usar atributos Static

- ¿Cómo mostraría los datos de cada objeto si fuera estático?

no los podria mostrar, daria error, deberias declarar los atributos que utiliza el metodo como static primero

- Observe el contenido del método otro:
- ¿Es posible definirlo como estático?

si

- ¿Por qué?

ya que los atributos cuales y cuantos están declarados como Static también

- Si se declara estático, ¿cómo se ejecuta?

Un método estático de una clase puede ser ejecutado directamente desde la propia clase, sin necesidad de crear una instancia de la misma.

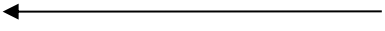
- Suponiendo que estos métodos están implementados en la clase B, y habiendo creado el objeto x de esa clase, en el método main, una con flechas lo que corresponda:

A. Realice un diagrama de clases que refleje el siguiente código JAVA (ingeniería inversa).

```
public class A  {
    String a;
    int b,c,d;

    public A(String s, int t)
   {  a = s; b = t; c,d =0;}

    public int sumar(int k) {
        int auxi = b+c+d+k;
        return auxi;
    }
}
```

```
public class B  extends A {
    private char a, b, c;

     public B(String h, int j, char m )
{
       super(h,j);
       a = m;
}
     public void inicializaCarB(char b)
    {
        this.b = b;
    }
     public void muestraCars() {
        System.out.println( a );
        System.out.println( b );
        System.out.println( c );
    }
    public int sumar(int k) {
        int auxi = ((int)b+(int)c+ super.d+k);
        return auxi;
    }
}
```

A
+ String a + int b + int c + int d



1. Respecto al diagrama anterior: ¿cómo se denomina el mecanismo utilizado con el método sumar? Compruebe si funcionan ambos correctamente.

El mecanismo utilizado con el método sumar se denomina "sobreescritura" o "redefinición de método". Ambas implementaciones del método sumar tienen el mismo nombre y parámetros, pero la implementación en la clase B anula la implementación en la clase A.

El método sumar no funcionará correctamente ya que la variable "c" no se inicializa en ningún lugar de la clase B, por lo tanto, tendrá un valor predeterminado de carácter nulo.

2. ¿Cómo se denominaría el mecanismo, si se implementa el método sumar en la clase B, con otros tipos o cantidad de parámetros?

Si se implementa el método sumar en la clase B con otros tipos o cantidad de parámetros, se denominaría "sobrecarga de método". La sobrecarga de método implica tener varios métodos con el mismo nombre, pero con diferentes parámetros, lo que permite llamar al método con diferentes tipos o cantidades de argumentos.

3. ¿Qué significado tienen las palabras reservadas this y super?

"this" se refiere al objeto actual en el que se está trabajando, mientras que "super" se refiere al objeto de la superclase. "this" se utiliza para evitar confusiones entre variables locales y de instancia, mientras que "super" se utiliza para acceder a los miembros de la superclase.

4. Según el funcionamiento del método inicializaCarB¿Cuál sería el nombre adecuado para el mismo?

El nombre adecuado para el método inicializaCarB podría ser "setB" o "asignarB", ya que su función es asignar un valor al atributo "b" de la clase B.

5. Falta definir algunos métodos, ¿Cuáles son?

Faltan los siguientes métodos:

- Un constructor sin parámetros para la clase A
- Métodos de acceso (getters y setters) para los atributos "a", "b" y "c" de la clase B

6. ¿Cuál es el estado de un objeto?

El estado de un objeto se refiere a los valores actuales de sus atributos en un momento dado. El estado de un objeto puede cambiar durante la ejecución de un programa a medida que se llaman métodos y se actualizan los valores de sus atributos.

7. ¿Qué son la ingeniería directa y la inversa?

La ingeniería directa se refiere al proceso de crear un diseño de software a partir de un conjunto de requisitos o especificaciones. La ingeniería inversa se refiere al proceso de crear un modelo o diseño de software a partir del código existente.

8. ¿Cuál es la diferencia entre sobrecarga y redefinición?

La sobrecarga de método implica tener varios métodos con el mismo nombre, pero con diferentes parámetros, lo que permite llamar al método con diferentes tipos o cantidades de argumentos. La redefinición de método implica anular un método en la subclase con una implementación diferente, pero con el mismo nombre y parámetros que el método en la superclase.

B. Observe las siguientes sentencias y explique por qué no son posibles:

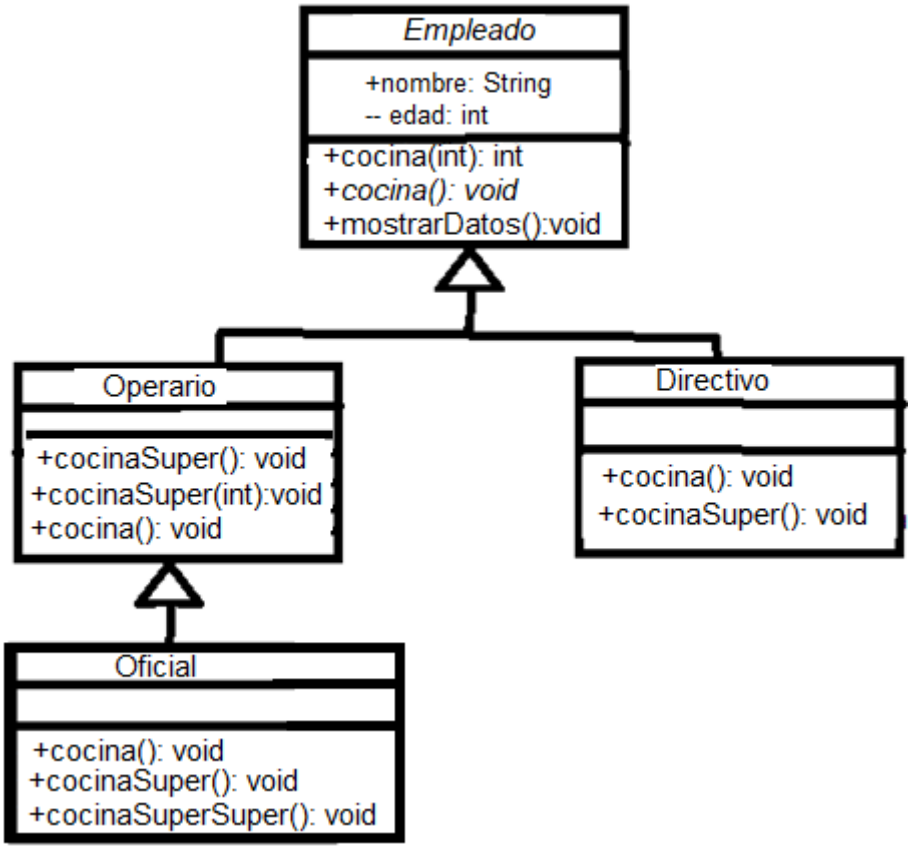
1. * private static final int VALOR = 9;

2. * public final double PI = 6;

3. * public static final moneda;

La sentencia no es posible porque no se ha proporcionado un tipo de datos para la variable "moneda". Además, la palabra clave "final" indica que la variable solo puede ser asignada una vez y no se puede cambiar su valor posteriormente. Pero no se ha asignado ningún valor a la variable "moneda", por lo que es imposible declararla como final.

C. Teniendo el modelo:



Observe responda las siguientes preguntas:

1. ¿En qué clases está redefinido el método cocina()?

El método esta redefinido en la clase Operario, Directivo, Oficial.

2. ¿En qué clases está sobrecargado?

El método esta sobrecargado en la clase Empleado.

3. ¿Qué otro método está redefinido y en qué clases?

El otro método redefinido es cocinaSuper() en la clase Operario y se redefine en la clase Oficial.

4. ¿En qué métodos se utiliza “super()”?

Se utiliza en los métodos cocinaSuper() y cocinaSuperSuper()

5. ¿Para qué se usa “ super.”?

“super.” Se usa para hacer referencia a atributos o métodos de la clase Padre, desde una subclase.

6. Desde la clase Directivo, ¿a qué atributos de la clase Oficial no se puede acceder directamente invocándolo con su nombre?

A los atributos private o protected (con su get) propios de la clase (aunque en este caso no cuenta con ninguno).

O si está hablando de atributos heredados solo podría acceder a nombre ya que es público.

7. Indique la sentencia a utilizar para cada uno de los siguientes casos
   a. Desde un objeto de clase Directivo, activar el cocina() de Empleado.

```
Directivo D1 = new Directivo();
D1.cocinaSuper();
```

   b. Desde un objeto de clase Oficial, activar el cocina(int) de Empleado.

Creemos que no es posible acceder ya que el método cocinaSuperSuper(): void de la clase oficial llama a cocina(): void en lugar de cocina(int): int.

   c. Desde el método main, creado en otra clase:
      i. Cambiar el valor del atributo edad, de un objeto de Oficial.

```
Oficial f1 =  new Oficial();
f1.setEdad(int);
```

      ii. Obtener el valor del atributo edad, de un objeto de la clase Operario.

```
Operario p1 =  new Operario();
p1.getEdad();
```

   d. Observando las cabeceras de los métodos, indique:

Clase a la que pertenece	Método	Utilidad que sugiere su cabecera	Métodos que activaría según su utilidad
Empleado	mostrarDatos()	Muestra los datos	ninguno

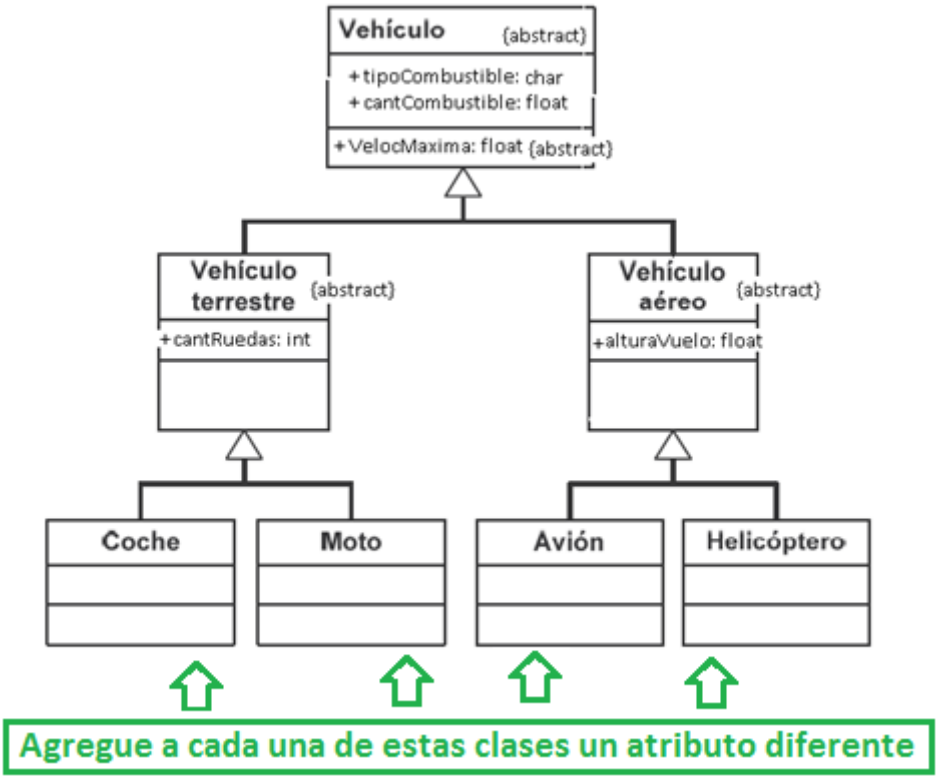
Oficial	cocinaSuper()	Llamar a cocina() de la clase Operario	cocina() (Operario)
Directivo	cocinaSuper()	Llamar a cocina() de la clase Empleado	cocina() (Empleado)
Oficial	cocinaSuperSuper ()	Llamar a cocina() de la clase Empleado	cocinaSuper() (Operario) cocina() (Empleado)

D. Si se agregara en la clase Empleado, **la variable public static int recetas = 0**, responda V o F según corresponda. Para cada respuesta compruebe el funcionamiento, creando el código en el entorno:

1. Se accedería a ella desde objetos de otras clases (de la misma jerarquía o de otras) nombrando a la clase Empleado.	V
2. Se podrá acceder desde objetos de cualquier clase, de esta jerarquía de clases.	V
3. Aunque no se pueden crear objetos de la clase Empleado, se la puede nombrar para acceder a la variable.	V
4. Un método definido en la clase Operario, podrá usar su valor sin nombrar a clases ni a objetos.	F
5. Todas las clases de esta jerarquía utilizan como propia a la variable.	F
6. Es correcto utilizar en el main: int a = Oficial.recetas * 2;	V

Herencia - Interfaces

B. Codifique el siguiente modelo



Pruebe el funcionamiento de la siguiente manera:

a. Implemente todo el modelo.
b. Si al método alturaVuelo() en la clase Avión, se le califica final. Indique verdadero o falso en las siguientes afirmaciones:
    1. El método se puede sobrecargar. V
    2. El método se puede sobreescribir o re-definir. F
    3. El método se puede ejecutar sólo desde un objeto de la clase Avión. V
    4. El método se puede ejecutar sólo desde un objeto de la clase VehículoAereo. F
    5. El método se puede ejecutar invocando desde la clase. F

1. Indique cuáles de las siguientes afirmaciones son verdaderas comprobando en el código:

Una clase abstracta…

a. no puede ser calificada como final. V
b. no puede tener métodos abstractos. F
c. puede heredar de otra clase no abstracta. V
d. no puede heredar de otra clase abstracta.F
e. necesita en su cabecera la palabra abstract. V
f. no necesita en su cabecera la palabra abstract. F
g. tiene al menos un método abstracto. V
h. tiene todos sus métodos abstractos.F

Una interface …

i. tiene al menos una variable no final. F

j. tiene todos sus métodos abstractos. V
k. todos sus atributos static y final. V
l. tiene al menos uno de sus atributos private, static y final. F

2. Responda investigando.
    a. Dé ejemplos de interfaces existentes en java.
        - ***java.util.List:*** Define la interfaz para una lista ordenada y permite operaciones de inserción, eliminación y recuperación de elementos.
        - ***java.util.Set:*** Define la interfaz para un conjunto de elementos únicos y no ordenados.
        - ***java.io.Serializable:*** Define la interfaz para marcar clases que se pueden serializar y deserializar.
    b. ¿Existe la herencia de interfaces?
    Sí, la herencia de interfaces es posible en Java. Una interfaz puede extender otra interfaz, lo que significa que hereda todos los métodos y constantes de la interfaz extendida.
    c. ¿Puede una subclase implementar a una interface?
    Sí, una subclase puede implementar una interfaz. La subclase debe proporcionar una implementación para todos los métodos declarados en la interfaz o debe ser una clase abstracta.
    d. ¿Es posible que una clase implemente más de una interface?
    Sí, es posible que una clase implemente múltiples interfaces en Java. Simplemente debe separar los nombres de las interfaces con comas en la declaración de implementación.