CURSO DE PROGRAMACIÓN FULL STACK

PROGRAMACIÓN ORIENTADA A OBJETOS

PARADIGMA ORIENTADO A OBJETOS

GUÍA DE PARADIGMA ORIENTADO A OBJETOS

PARADIGMAS DE PROGRAMACION

Un paradigma de programación es una manera o estilo de programación. Existen diferentes formas de diseñar un programa y varios modos de trabajar para obtener los resultados que necesitan los programadores. Por lo que un paradigma de programación se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas.

PROGRAMACION ORIENTADA A OBJETOS

La Programación Orientada a Objetos (POO) es un paradigma de programación, es decir, un modelo o un estilo de programación que se basa en el concepto de clases y objetos. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de código (clases) para crear instancias individuales de objetos.

Con el paradigma de Programación Orientado a Objetos lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos, lo que constituye la base de este paradigma.

La programación orientada a objetos se enfoca en los objetos, sus atributos y las interacciones que se producen entre ellos para diseñar programas.
Un programa orientado a objetos es, esencialmente, un conjunto de objetos que se crean, interaccionan entre sí y dejan de existir cuando ya no son útiles durante la ejecución de un programa. Un programa puede llegar a ser muy complejo. La complejidad es más manejable cuando se descompone y se organiza en partes pequeñas y simples, los objetos.

¿POR QUÉ POO?

La Programación Orientada a objetos permite que el código sea reutilizable, organizado y fácil de mantener. Sigue el principio de desarrollo de software utilizado por muchos programadores DRY (Don't Repeat Yourself), para evitar duplicar el código y crear de esta manera programas eficientes. Además, evita el acceso no deseado a los datos o la exposición de código propietario mediante la encapsulación y la abstracción, de la que hablaremos en detalle más adelante.

CLASES Y OBJETOS

Una clase es un molde para crear múltiples objetos que encapsula datos y comportamiento. Una clase es una combinación específica de atributos y métodos y puede considerarse un tipo de dato de cualquier tipo no primitivo. Así, una clase es una especie de plantilla o prototipo de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos. En su forma más simple, una clase se define por la palabra reservada class seguida del nombre de la clase. El nombre de la clase debe empezar por mayúscula.

Si el nombre es compuesto, entonces cada palabra debe empezar por mayúscula. La definición de la clase se pone entre las llaves de apertura y cierre.

```
public class NombreClase {

	// atributos

	// constructores

	// metodos

}
```

Una vez que se ha declarada una clase, se pueden crear objetos a partir de ella. A la creación de un objeto se le denomina instanciación. Es por esto que se dice que un objeto es una instancia de una clase y el término instancia y objeto se utilizan indistintamente. Para crear objetos, basta con declarar una variable de alguno de los tipos de las clases definidas.

```
NombreClase nombreObjeto;
```

Para crear el objeto y asignar un espacio de memoria es necesario realizar la instanciación con el operador new. El operador new instancia el objeto y reserva espacio en memoria para los atributos y devuelve una referencia que se guarda en la variable.

```
nombreObjeto = new nombreClase();
```

Tanto la declaración de un objeto como la asignación del espacio de memoria se pueden realizar en un solo paso:

```
NombreClase nombreObjeto = new NombreClase();
```

A partir de este momento los objetos ya pueden ser referenciados por su nombre

ACCESO A LOS ATRIBUTOS

Desde un objeto se puede acceder a los atributos mediante la siguiente sintaxis:

```
nombreObjeto.atributo;
```

ESTADO Y COMPORTAMIENTO

En términos más generales, un objeto es una abstracción conceptual del mundo real que se puede traducir a un lenguaje de programación orientado a objetos. Los objetos del mundo real comparten dos características: Todos poseen *estado y comportamiento*. Por ejemplo, el perro tiene estado (color, nombre, raza, edad) y el comportamiento (ladrar, caminar, comer, acostarse, mover la cola). Por lo tanto, un estado permite informar cuáles son las características del objeto y lo que éste representa, y el comportamiento, consiste en decir lo que sabe hacer.

El estado de un objeto es una lista de variables conocidas como sus atributos, cuyos valores representan el estado que caracteriza al objeto.

El comportamiento es una lista de métodos, procedimientos, funciones u operaciones que un objeto puede ejecutar a solicitud de otros objetos. Los objetos también se conocen como instancias.

ELEMENTOS DE UNA CLASE

Una clase describe un tipo de objetos con características comunes. Es necesario definir la información que almacena el objeto y su comportamiento.

ATRIBUTOS

El estado o información de un objeto se almacena en atributos. Los atributos pueden ser de tipos primitivos de Java (descriptos en la guía Intro Java) o del tipo de otros objetos. La declaración de un atributo de un objeto tiene la siguiente forma:

```
<modificador>* <tipo> <nombre> [ = <valor inicial> ];
```

- <nombre>: puede ser cualquier identificador válido y denomina el atributo que está siendo declarado.
- <modificador>: si bien hay varios valores posibles para el <modificador>, por el momento solo usaremos modificadores de visibilidad: public, protected, private.
- <tipo>: indica la clase a la que pertenece el atributo definido.
- <valor inicial>: esta sentencia es opcional y se usa para inicializar el atributo del objeto con un valor particular.

Estos atributos irán al principio de la clase.

CONSTRUCTORES

Además de definir los atributos de un objeto, es necesario definir los métodos que determinan su comportamiento. Toda clase debe definir un método especial denominado constructor para instanciar los objetos de la clase. Este método tiene el mismo nombre de la clase. La declaración básica toma la siguiente forma:

```
[<modificador>] <nombre de clase> ( <argumento>* ) {

   <sentencia>*

}
```

- <nombre de clase>: El nombre del constructor debe ser siempre el mismo que el de la clase.
- <modificador>: Actualmente, los únicos modificadores válidos para los constructores son public, protected y private.
- <argumentos>: es una lista de parámetros que tiene la misma función que en los métodos.

El método constructor se ejecuta cada vez que se instancia un objeto de la clase. Este método se utiliza para **inicializar** los atributos del objeto que se instancia.

Para diferenciar entre los atributos del objeto y los identificadores de los parámetros del método constructor, se utiliza la palabra this. De esta forma, los parámetros del método pueden tener el mismo nombre que los atributos de la clase.

La instanciación de un objeto consiste en asignar un espacio de memoria al que se hace referencia con el nombre del objeto. Los identificadores de los objetos permiten acceder a los valores almacenados en cada objeto.

El Constructor por Defecto

Cada clase tiene al menos un constructor. Si no se escribe un constructor, el lenguaje de programación Java le provee uno por defecto. Este constructor no posee argumentos y tiene un cuerpo vacío. Si se define un constructor que no sea vacío, el constructor por defecto se pierde, salvo que creemos un nuevo constructor vacio.

MÉTODOS

Los métodos son funciones que determinan el comportamiento de los objetos. Un objeto se comporta de una u otra forma dependiendo de los métodos de la clase a la que pertenece. Todos los objetos de una misma clase tienen los mismos métodos y el mismo comportamiento. Para definir los métodos, el lenguaje de programación Java toma la siguiente forma básica:

```
<modificador>* <tipo de retorno> <nombre> ( <argumento>>* ) {

    <sentencias>*

    return valorRetorno;

}
```

- <nombre>: puede ser cualquier identificador válido, con algunas restricciones basadas en los nombres que ya están en uso.
- <modificador>: el segmento es opcional y puede contener varios modificadores diferentes incluyendo a public, protected y private. Aunque no está limitado a estos.
- <tipo de retorno>: el tipo de retorno indica el tipo de valor devuelto por el método. Si el método no devuelve un valor, debe ser declarado void. La tecnología Java es rigurosa acerca de los valores de retorno. Si el tipo de retorno en la declaración del método es un int, por ejemplo, el método debe devolver un valor int desde todos los posibles caminos de retorno (y puede ser invocado solamente en contextos que esperan un int para ser devuelto). Se usa la sentencia return dentro de un método para devolver un valor.
- <argumento>: permite que los valores de los argumentos sean pasados hacia el método. Los elementos de la lista están separados por comas y cada elemento consiste en un tipo y un identificador.

Existen tres tipos de métodos: métodos de consulta, métodos de consulta y operaciones Los métodos de consulta sirven para extraer información de los objetos, los métodos modificadores sirven para modificar el valor de los atributos del objeto y las operaciones definen el comportamiento de un objeto.

Para acceder a los atributos de un objeto se definen los métodos get y set. Los métodos get se utilizan para consultar el estado de un objeto y los métodos set para modificar su estado. Un método get se declara public y a continuación se indica el tipo de dato que devuelve. Es un método de consulta. La lista de parámetros de un método get queda vacía. En el cuerpo del método se utiliza return para devolver el valor correspondiente al atributo que se quiere devolver, y al cual se hace referencia como this.nombreAtributo.

Por otra parte, un método set se declara public y devuelve void. La lista de parámetros de un método set incluye el tipo y el valor a modificar. Es un método modificador. El cuerpo de un método set asigna al atributo del objeto el parámetro de la declaración.

Por último, un método de tipo operación es aquel que realiza un cálculo o modifica el estado de un objeto. Este tipo de métodos pueden incluir una lista de parámetros y puede devolver un valor o no. Si el método no devuelve un valor, se declara void.

Invocación de métodos

Un método se puede invocar dentro o fuera de la clase donde se ha declarado. Si el método se invoca dentro de la clase, basta con indicar su nombre. Si el método se invoca fuera de la clase entonces se debe indicar el nombre del objeto y el nombre del método. Cuando se invoca a un método ocurre lo siguiente:

- En la línea de código del programa donde se invoca al método se calculan los valores de los argumentos.
- Los parámetros se inicializan con los valores de los argumentos.
- Se ejecuta el bloque código del método hasta que se alcanza return o se llega al final del bloque.
- Si el método devuelve un valor, se sustituye la invocación por el valor devuelto.
- La ejecución del programa continúa en la siguiente instrucción donde se invocó el método.

Parámetros y argumentos

Los parámetros de un método definen la cantidad y el tipo de dato de los valores que recibe un método para su ejecución. Los argumentos son los valores que se pasan a un método durante su invocación. El método recibe los argumentos correspondientes a los parámetros con los que ha sido declarado. Un método puede tener tantos parámetros como sea necesario. La lista de parámetros de la cabecera de un método se define con la siguiente sintaxis:

```
tipo nombre [tipo nombre,]
```

Durante la invocación de un método es necesario que el número y el tipo de argumentos coincidan con el número y el tipo de parámetros declarados en la cabecera del método. Durante el proceso de compilación se comprueba que durante la invocación de un método se pasan tantos argumentos como parámetros tiene declarados y que además coinciden los tipos. Esta es una característica de los lenguajes que se denominan “strongly typed” o “fuertemente tipados”.

Paso de parámetros

Cuando se invoca un método se hace una copia de los valores de los argumentos en los parámetros. Esto quiere decir que, si el método modifica el valor de un parámetro, nunca se modifica el valor original del argumento.

Cuando se pasa una referencia a un objeto se crea un nuevo alias sobre el objeto, de manera que esta nueva referencia utiliza el mismo espacio de memoria del objeto original y esto permite acceder al objeto original.

El valor de retorno

Un método puede devolver un valor. Los métodos que no devuelven un valor se declaran void, mientras que los métodos que devuelven un valor indican el tipo que devuelven: int, double, char, String o un tipo de objeto.

Sobrecarga de métodos

La sobrecarga de métodos es útil para que el mismo método opere con parámetros de distinto tipo o que un mismo método reciba una lista de parámetros diferente. Esto quiere decir que puede haber dos métodos con el mismo nombre que realicen dos funciones distintas. La diferencia entre los métodos sobrecargados está en su declaración, y más específicamente, en la cantidad y tipos de datos que reciben.

ABSTRACCIÓN Y ENCAPSULAMIENTO

La abstracción es la habilidad de ignorar los detalles de las partes para enfocar la atención en un nivel más alto de un problema. El encapsulamiento sucede cuando algo es envuelto en una capa protectora. Cuando el encapsulamiento se aplica a los objetos, significa que los datos del objeto están protegidos, "ocultos" dentro del objeto. Con los datos ocultos, ¿cómo puede el resto del programa acceder a ellos? (El acceso a los datos de un objeto se refiere a leerlos o modificarlos.) El resto del programa no puede acceder de manera directa a los datos de un objeto; lo tiene que hacer con ayuda de los métodos del objeto. Al hecho de proteger los datos o atributos con los métodos se denomina encapsulamiento.

ABSTRACCIÓN

La abstracción es la propiedad que considera los aspectos más significativos o notables de un problema y expresa una solución en esos términos. La abstracción posee diversos grados o niveles de abstracción, los cuales ayudan a estructurar la complejidad intrínseca que poseen los sistemas del mundo real. La abstracción encarada desde el punto de vista de la programación orientada a objetos es el mecanismo por el cual se proveen los límites conceptuales de los objetos y se expresan sus características esenciales, dejando de lado sus características no esenciales. Si un objeto tiene más características de las necesarias los mismos resultan difíciles de usar, modificar, construir y comprender. En el análisis hay que concentrarse en ¿Qué hace? y no en ¿Cómo lo hace?

ENCAPSULAMIENTO

La encapsulación o encapsulamiento significa reunir en una cierta estructura a todos los elementos que a un cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad, y es el proceso de agrupamiento de datos y operaciones relacionadas bajo una misma unidad de programación, lo que permite aumentar la cohesión de los componentes del sistema.

El encapsulamiento oculta lo que hace un objeto de lo que hacen otros objetos y del mundo exterior por lo que se denomina también ocultación de datos. Un objeto tiene que presentar "una cara" al mundo exterior de modo que se puedan iniciar sus operaciones.
Los métodos operan sobre el estado interno de un objeto y sirven como el mecanismo primario de comunicación entre objetos. Ocultar el estado interno y hacer que toda interacción sea a través de los métodos del objeto es un mecanismo conocido como encapsulación de datos.

MODIFICADORES DE ACCESO

Para lograr el uso correcto del encapsulamiento vamos utilizar los modificadores de acceso, estos, van a dejarnos elegir como se accede a los datos y a través de que se accede a dichos datos. Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación, se detallan los niveles de acceso con sus símbolos correspondientes:

- Public: Este modificador permite a acceder a los elementos desde cualquier clase, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.

- Private: Es el modificador más restrictivo y especifica que los elementos que lo utilizan sólo pueden ser accedidos desde la clase en la que se encuentran. Este modificador sólo puede utilizarse sobre los atributos de una clase y sobre interfaces y clases internas, no sobre clases o interfaces de primer nivel, dado que esto no tendría sentido. Es importante destacar también que el modificador private convierte los elementos en privados para otras clases, no para otras instancias de la clase. Es decir, un objeto de una determinada clase puede acceder a los atributos privados de otro objeto de la misma clase.

- Protected: Este modificador indica que los elementos sólo pueden ser accedidos desde su mismo paquete y desde cualquier clase que extienda la clase en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como private, no tiene sentido a nivel de clases o interfaces no internas.

Si no especificamos ningún modificador de acceso se utiliza el nivel de acceso por defecto (Default), que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete. Los distintos modificadores de acceso quedan resumidos en la siguiente tabla

Visibilidad	Public	Private	Protected	Default
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	SI	SI	SI
Desde una Subclase del mismo Paquete	SI	SI	SI	SI
Desde una Subclase fuera del mismo Paquete	SI	NO	SI, a través de la herencia	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

ATRIBUTOS Y METODOS ESTÁTICOS

Un atributo o un método de una clase se puede modificar con la palabra reservada static para indicar que este atributo o método no pertenece a las instancias de la clase si no a la propia clase.

Se dice que son atributos de clase si se usa la palabra clave static: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria), es decir que, si se poseen múltiples instancias de una clase, cada una de ellas no tendrán una copia propia de este atributo, si no que todas estas instancias compartirán una misma copia del atributo. A veces a las variables de clase se les llama variables estáticas. Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (la variable es diferente para cada objeto).

En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree. Por ello es adecuado el uso de la palabra clave static. Cuando usamos *"static final"* se dice que creamos una constante de clase, un atributo común a todos los objetos de esa clase.

Ejemplo:

```
Public class Cuenta {

private static String saldo;

}
```

ATRIBUTOS FINALES

En este contexto indica que una variable es de tipo constante: no admitirá cambios después de su declaración y asignación de valor. La palabra reservada *final* determina que un atributo no puede ser sobrescrito o redefinido, es decir, no funcionará como una variable "tradicional", sino como una constante. Toda constante declarada con *final* ha de ser inicializada en el mismo momento de declararla. El modificador *final* también se usa como palabra clave en otro contexto: una clase *final* es aquella que no puede tener clases que la hereden. Lo veremos más adelante cuando hablemos sobre herencia.

Cuando se declaran constantes es muy frecuente que los programadores usen letras mayúsculas (como práctica habitual que permite una mayor claridad en el código), aunque no es obligatorio.

Ejemplo:

```
Public class Perro {

private final int edad;

}
```

EN RESUMEN

Antes de POO, la técnica estándar de programación era la programación procedural. Se denomina programación procedural porque en ella se destacan los procedimientos o tareas que resuelven un problema. Se piensa primero en lo que se quiere hacer: los procedimientos.

En contraste, el paradigma POO invita a pensar en lo que se desea que represente el programa. Normalmente se responde esta invitación identificando algunas cosas en el mundo que se desea que el programa modele. Estas cosas podrían ser entidades físicas o conceptuales, por ejemplo, un libro. Una vez identificadas las cosas que se quiere modelar, se identifican sus propiedades/atributos básicos. Estos se pueden agrupar todos juntos en una estructura coherente llamada objeto que creamos a través de las clases.

Proyecto con ejemplos de POO

El ejemplo está pensado para una pagina que va a administrar perros en su pagina web, muestra las distintas maneras de llenar y de mostrar un objeto.

Link: Programación Orientada a Objetos

CLASES DE UTILIDAD PARTE 2

Recordemos que las clases de utilidad son clases dentro del API de Java que son muy utilizadas en el desarrollo de aplicaciones. Las clases de utilidad son clases que definen un conjunto de métodos que realizan funciones, normalmente muy reutilizadas. Estas nos van a ayudar junto con las estructuras de control, a lograr resolver problemas de manera más sencilla.

Entre las clases de utilidad de Java más utilizadas y conocidas están las siguientes: Arrays, String, Integer, Math, Date, Calendar y GregorianCalendar. En la guía anterior vimos solo las clases Math y String. Ahora vamos a ver el resto de las clases.

CLASE ARRAYS

La clase Arrays es una clase de utilidad que posee una gran cantidad de métodos para manipular arreglos.

Método	**Descripción.**
Arrays.equals(arreglo1, arreglo2)	Retorna true o false, si dos arreglos del mismo tipo de dato son iguales.
Arrays.fill(arreglo, variable) Arrays.fill(arreglo, int desde, int hasta, variable)	Este método lo que hace es inicializar todo el arreglo con la variable o valor que pasamos como argumento. Este método se puede usar con cualquier tipo de dato y le podemos decir desde y hasta que índice queremos que llene con ese valor.

Arrays.sort(arreglo) Arrays.sort(arreglo, int desde, int hasta)	Este método sirve para ordenar un arreglo de manera ascendente. A este método también le podemos decir desde y hasta que índice queremos que ordene.
Arrays.binarySearch(arreglo,valor)	Este método sirve para buscar un elemento determinado en un arreglo. El método devuelve la posición en la cual se encuentra el elemento. La implementación del algoritmo de búsqueda utilizado es el de búsqueda binaria, por lo tanto, antes de utilizar este método debemos asegurarnos que el arreglo se encuentre ordenado.
Arrays.toString(arreglo)	Este método imprime el arreglo como una cadena, la cadena consiste en una lista de los elementos del arreglo encerrados entre corchetes ("[]"). Los elementos adyacentes están separados por comas (",").

CLASE INTEGER

La clase Integer permite convertir un tipo primitivo de dato int a objeto Integer. La clase Integer pertenece al paquete java.lang del API de Java y hereda de la clase java.lang.Number.

Método	**Descripción.**
Integer(String s)	Constructor que inicializa un objeto con una cadena de caracteres. Esta cadena debe contener un número entero.
compareTo(entero, otroEntero)	Compara dos objetos Integer numéricamente. Retorna 0 si son iguales, entero negativo si el primer numero es menor o entero positivo si el primer numero es mayor.
doubleValue()	Retorna el valor del Integer en tipo primitivo double
equals(Object obj)	Compara el Integer con el objeto del parámetro. Devuelve true si son iguales y false si no.

parseInt(String s)	Convierte la cadena de caracteres numérica del parámetro en tipo primitivo int.
toString()	Retorna el valor del Integer en una cadena de caracteres.

CLASE DATE

La clase Date modela objetos o variables de tipo fecha. La clase Date representa un instante de tiempo específico con una precisión en milisegundos y permite el uso del formato Universal Coordinated Time (UTC). Por otro lado, muchas computadoras están definidas en términos de Greenwich Mean Time (GMT) que es equivalente a Universal Time (UT). GMT es el nombre estándar y UT es el nombre científico del estándar. La diferencia entre UT y UTC es que UTC está basado en un reloj atómico y UT está basado en un reloj astronómico.

Las fechas en Java, comienzan en el valor standar based time llamado “epoch” que hace referencia al 1 de Enero de 1970, 0 horas 0 minutos 0 segundos GMT.

La clase Date posee métodos que permiten la manipulación de fechas. La clase Date pertenece al paquete java.util del API de Java.

Método	**Descripción.**
Date()	Constructor que inicializa la fecha en el milisegundo más cercano a la fecha del sistema.
Date(int dia, int mes, int año)	Constructor que inicializa la fecha sumándole 1900 al año.
after(Date fecha2)	Retorna verdadero si la fecha esta después de la fecha del parámetro
before(Date fecha2)	Retorna verdadero si la fecha esta antes de la fecha del parámetro
compareTo(Date fecha)	Compara la fecha con la del parámetro. Retorna 0 si son iguales, entero negativo si el primer numero es menor o entero positivo si el primer numero es mayor.

equals(Object obj)	Compara el Date con el objeto del parámetro. Devuelve true si son iguales y false si no.
getDay()	Retorna el valor del día de la semana de la fecha. Ejemplo: si es lunes devuelve 0, martes 1, miércoles 2, jueves 3, viernes 4, sábado 5 y domingo 6.
getDate()	Retorna el numero del día de la fecha.
getMonth()	Retorna el mes de la fecha.
getYear()	Retorna el año de la fecha.
getTime()	Retorna la fecha en milisegundos a partir del “epoch”.
setDate(int dia)	Asigna un día a la fecha.
setMonth(int mes)	Asigna un mes a la fecha.
setYear(int anio)	Asigna un año a la fecha.
setTime(long time)	Asigna la fecha en milisegundos a partir del “epoch”.
toString()	Retorna la fecha en una cadena de caracteres.

Proyecto con ejemplos de Clase Utilidad

En este proyecto hay un Main con todas las clases de utilidad previamente vistas.

Link: Clases de Utilidad

PREGUNTAS DE APRENDIZAJE

1) Responda Verdadero (V) o Falso (F)

Cuando se coloca la palabra final precediendo la declaración de una variable la misma se transforma en una constante	( )	( )
Una variable local no puede ser declarada en cualquier lugar del cuerpo de una clase o método	( )	( )
El método constructor de una clase puede tener cualquier nombre	( )	( )
Cuando un método no devuelve ningún valor se utiliza la palabra reservada void para indicar que no devuelve nada	( )	( )
La palabra reservada final determina que un atributo no puede ser redefinido	( )	( )
La devolución de un valor a través de un método se hace con la sentencia restore	( )	( )

2) ¿Cuál es la descripción que crees que define mejor el concepto clase en la programación orientada a objetos?

a) Es un concepto similar al de array
b) Es un tipo particular de variable
c) Es un modelo o plantilla a partir de la cual creamos objetos
d) Es una categoría de datos ordenada secuencialmente

3) ¿Qué elementos crees que definen a un objeto?

a) Su cardinalidad y su tipo
b) Sus atributos y sus métodos
c) La forma en que establece comunicación e intercambia mensajes
d) Su interfaz y los eventos asociados

4) Una clase es:

a) Un molde para crear múltiples objetos
b) Un tipo de variable
c) Un tipo de modificador de acceso
d) Ninguna de las anteriores

5) El modificador de acceso private, hace que los datos puedan ser accedidos por

a) Cualquier clase
b) La clase donde se encuentran
c) El método main
d) Ninguna de las anteriores

6) ¿Qué significa instanciar una clase?

a) Duplicar una clase
b) Eliminar una clase
c) Crear un objeto a partir de la clase
d) Conectar dos clases entre sí

7) Queremos crear una clase Java con atributos que puedan ser accedidos, ¿qué opción elegirías como la mejor?

a) Atributos públicos
b) Atributos static
c) Atributos privados con getters y setters
d) Ninguna de las anteriores

8) Se crean anteponiendo la palabra static a su declaración:

a) Atributos de objeto
b) Atributos de clase estáticos
c) Variables finales
d) Ninguna de las anteriores

9) No puede cambiar su valor durante la ejecución del programa:

a) Atributos de objeto
b) Atributos de clase
c) Variables finales
d) Todas las anteriores