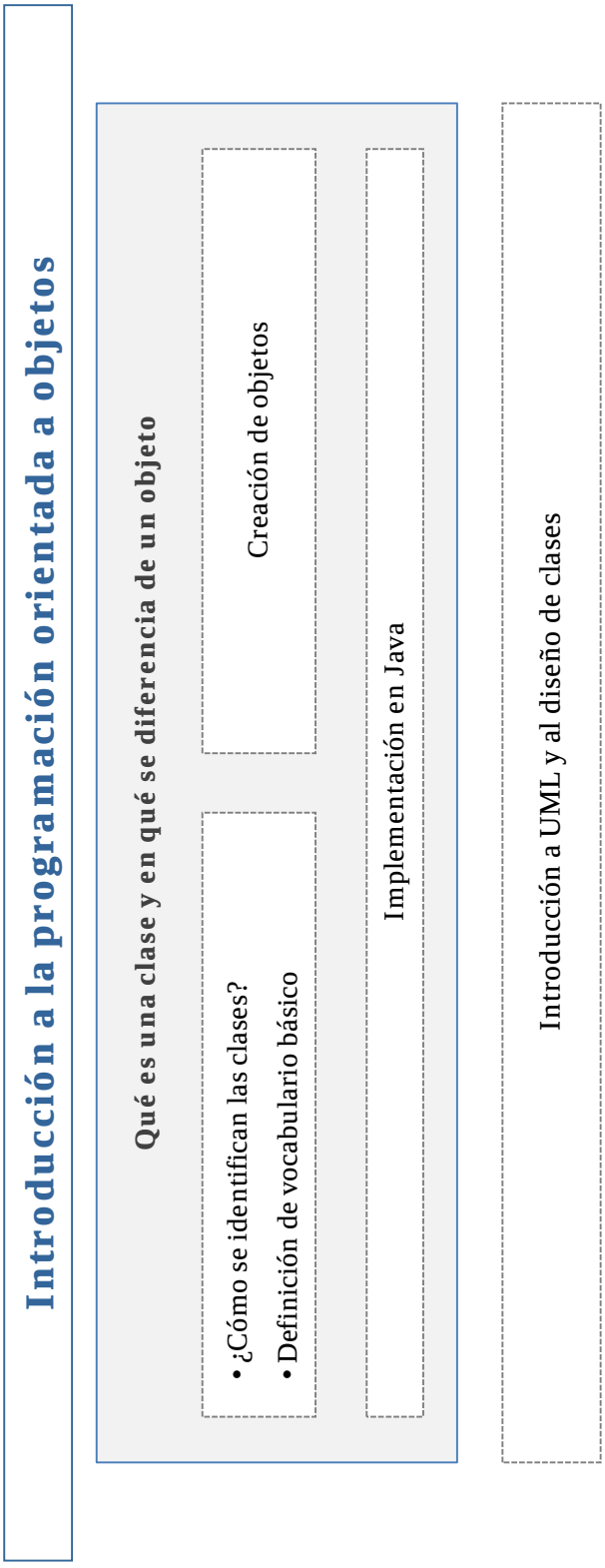
Introducción a la programación orientada a objetos

TEMA 1

Esquema



Ideas clave

1.1. ¿Cómo estudiar este tema?

Para estudiar este tema debes leer las **páginas 1-46** del siguiente libro, disponible en la Biblioteca Virtual de UNIR:

García, L. F. (2010). *Todo lo básico que debería saber sobre programación orientada a objetos en Java*. Barranquilla: Uninorte.

Para comprobar si has comprendido los conceptos realiza el test de autoevaluación del tema.

En este tema vamos a aprender los conceptos básicos de la programación orientada a objetos:

- **Qué es** una clase y **cómo se diseña**.
- La **diferencia** entre clase y objeto.
- La **representación gráfica** de las mismas a través de UML (Unified Modelling Language).

1.2. Introducción a la programación orientada a objetos

Dentro de la programación existen muchos paradigmas**, estilos de programación**, que nos llevan a la resolución de problemas. Cuando un programador se enfrenta a un problema por primera vez debe identificar el **estilo que mejor se ajusta a los requisitos que ha extraído**.

Programación

Habitualmente se define como un **proceso** que tiene una **entrada**, un **proceso o manipulación** a través de la cual se obtiene una **salida**.

Es decir, un **algoritmo** es una **sucesión de pasos** que nos llevan a la **resolución** de un problema, algo así como una receta.

Sin embargo hay ocasiones en las que para solucionar un programa vemos que no solo necesitamos una secuencia de pasos, sino que existen **elementos que interaccionan entre sí para lograr un objetivo**.

Por ejemplo, si queremos realizar un juego que simule un partido de fútbol, los diferentes participantes deben interaccionar entre ellos. Si tuviéramos que hacerlo a través de programación tradicional podríamos, por ejemplo, crear un procedimiento jugador que implementase su comportamiento y, a través de variables, podríamos definir todos los jugadores. Pero, ¿cuántas variables deberíamos tener? ¿Cómo implementamos la interacción entre los distintos jugadores? Para solucionar este problema aparece la **programación orientada a objetos**.

La **POO** (Programación Orientada a Objetos) se basa en el concepto de que los elementos que forman parte de un programa se pueden crear como **entidades que interaccionen entre ellas**, que se intercambian mensajes, para solucionar un problema.

En el ejemplo anterior del simulador de futbol, de manera muy simplificada, podríamos identificar a los jugadores como entidades que interaccionan enviándose mensajes (jugadas) para marcar un gol al equipo contrario.

Conceptos básicos de orientación a objetos

En este apartado vamos a introducir el vocabulario básico que utilizamos en POO y algunos conceptos que veremos en temas posteriores en profundidad.

- **Clase**: Es la base de la POO. Se puede decir que es una **forma de representar** las **características comunes de elementos** que van a formar parte del programa.

- **Objeto**: Es la **personalización de las características definidas** en una clase con datos concretos. Cuando creamos un objeto se dice que estamos instanciando una clase.

- **Herencia**: **Relación entre clases** que permite que una clase obtenga de otra los atributos y operaciones definidas en la clase de la cual deriva.

- **Abstracción**: Se trata de **extraer las características comunes** que tienen varios elementos para luego poder especializarlos.

- **Agregación**: Relación en la que un **objeto forma parte de otro mayor**.

- **Persistencia**: Es la **propiedad de un objeto de mantenerse en el tiempo**. Por ejemplo, siendo almacenado en una base de datos o en un archivo.

- **Visibilidad**: Capacidad por la cual un **objeto o método puede ser ocultado al resto** de elementos que forman parte del código.

- **Encapsulación**: Se trata de **organizar los datos** de tal manera que no sean manipulables por cualquiera, sino que para llegar a ellos tengamos que trabajar con procedimientos adecuados.

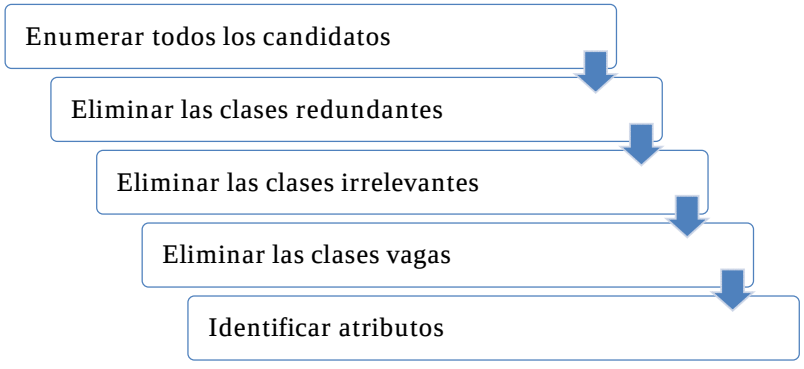
- **Polimorfismo**: Es la capacidad por la cual un método realizará **tareas diferentes** dependiendo del objeto que lo invoque.

1.3. Diseño de clases

Cuando nos enfrentamos a un problema orientado a objetos lo más complicado, sobre todo al principio, es **identificar las clases involucradas**. ¿Cómo podemos saber qué es una clase y qué no? La clave para tener éxito es abstraer el problema, es decir, obtener de los requisitos que muestran la información necesaria para identificar los elementos que forman parte del mismo.

Aunque no siempre se cumple se suele decir que se pueden identificar clases dentro de un problema porque suelen ser **nombre comunes**.

El proceso más sencillo para la identificación de clases es:



Ejemplo 1: Una biblioteca ha decidido informatizar el préstamo de libros La biblioteca tiene libros y revistas que se pueden prestar durante un máximo de una semana. Si el libro es de consulta, no se puede prestar, ya que se debe utilizar únicamente dentro de la biblioteca. Los usuarios de la biblioteca que no realicen la devolución de los préstamos en el tiempo adecuado serán sancionados sin poder alquilar libros durante una semana.

Ejemplo 2: Se quiere informatizar el proceso de gestión de los alumnos con sus matrículas. Todos los alumnos deben estar matriculados como mínimo en una asignatura. Las asignaturas pueden ser anuales o cuatrimestrales. Una misma asignatura podrá ser impartida por uno o varios profesores. Un curso se compone de varias asignaturas que podrán ser impartidas durante un cuatrimestre o un año. Los alumnos podrán consultar las notas de todas las asignaturas en las que estén matriculados.

La solución a ambos ejemplos se verá en las clases presenciales virtuales.

Implementación de clases en Java

En Java las clases se definen como:

```
class Miclase {
    // Definición de la clase atributos y métodos
}
```

Una vez tenemos identificadas las clases debemos pasar a **dotarlas de contenido**. Para ello debemos comenzar **identificando los atributos** de las mismas. Los **atributos** nos ofrecen características de las clases y permiten a través de sus valores componer el estado de un objeto. En Java se definen:

```
[visibilidad] [tipo] nombre;
```

`Private`: únicamente se puede ver este atributo dentro de la clase.
`Public`: Se puede acceder a él desde cualquier clase del programa.
`Protected`: Accesible desde la propia clase y sus subclases.

Ejemplos:

```
Private int edad;
Public int velocidad;
Protected String nombre;
```

Por último, para dotar a las clases de comportamiento debemos definir los métodos que forman parte de la misma.

Las clases pueden tener tantos métodos como queramos, pero existen una serie de ellos que son especiales por su comportamiento.

- ***Getters*** y ***Setters***. Los *getters* nos permiten **obtener el valor** de los atributos privados y los *setters* **establecerlos**.

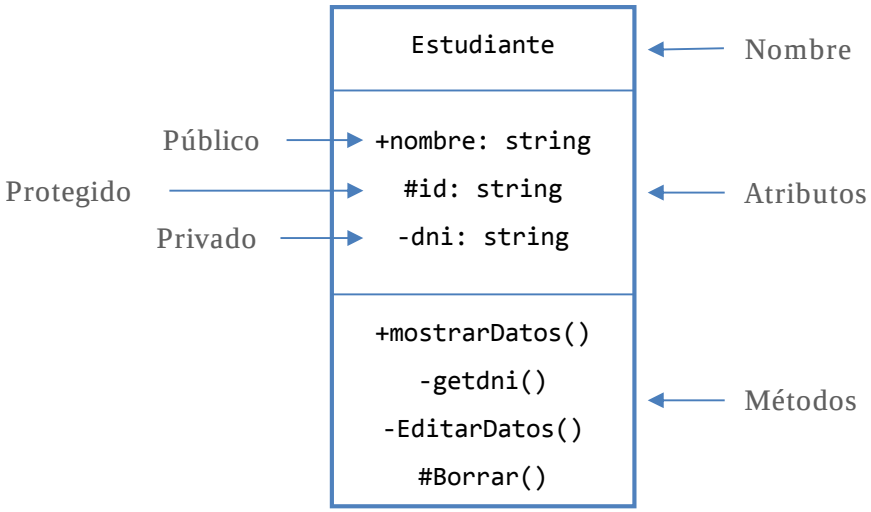
- Los **constructores** son **métodos** que tienen el mismo nombre que la clase y son invocados al instanciar la clase. El constructor puede, por ejemplo, establecer los valores por defecto que deben tener los atributos. Una clase puede tener varios constructores que permitirán que se cree de manera diferente dependiendo del que se invoque.

La creación de objetos se realiza a través del operador `new` que invoca el constructor que se corresponda con los parámetros actuales indicados.

1.4. Introducción a UML para el modelado de los problemas

En UML una clase se identifica como un **cuadrado con tres partes**, aunque en diseños iniciales se puede definir como un cuadrado en el que se indica el nombre únicamente.

En la **parte superior** se indica el **nombre de la clase**, en la **parte central** se indican los **atributos** que tiene esa clase y en la **parte inferior** se especifican los **métodos** con los que se puede operar sobre esa clase.



De manera adicional puede existir un rectángulo inferior en el que se indicarían las **responsabilidades de la clase**.

Recuerda:

- **Atributo público**: Accesible desde cualquier parte del código:
- **Atributo protegido**: Accesible desde la propia clase y sus subclases.
- **Atributo privado**: Solo accesible desde la misma clase.
- **Atributo sin modificador**: Se denomina de «paquete».

En la siguiente tabla, extraída de la página de Oracle (http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html) se muestra un resumen.

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

1.5. Introducción a la eficiencia y complejidad de un algoritmo

La eficiencia de un algoritmo o éxito que un algoritmo debe de realizar, no tiene que depender del potencial del hardware o del tipo de sistema operativo en el cual se ejecuta. Es decir, si comparamos dos algoritmos, uno será más eficiente que otro si los cálculos realizados o funciones de coste son menores al obtener el mismo resultado.

Esto dará lugar a que un algoritmo más eficiente pueda resolver su función con respecto a otro. Para poder comparar dos algoritmos es necesario analizar la eficiencia en función del tiempo que consume en ejecutarse y en función del grado de complejidad que tenga.

Come ejemplo ilustrativo, examinaremos dos de los algoritmos más populares que organizan una serie de datos, como son: ordenación por burbuja (**BubbleSort**) y **QuickSort**. Se puede comprobar que el tiempo utilizado por QuickSort será menor a la de BubbleSort aun cuando el sistema donde se ejecuta el BubbleSort sea más rápido que el sistema donde se ejecuta el QuickSort.

A continuación, se muestra el algoritmo BlubbleSort y QuickSort implementado en una clase de Java:

```
package Metodosdebusqueda;
public class Burbujas {
public void OrdenarBurbujas(int[] arreglo)
{
int aux;
boolean cambios=false;
while(true)
{
cambios=false;
for(int k=1;k<arreglo.length;k++){
if(arreglo[k]<arreglo[k-1]){
                        auxiliar= arreglo[k];
                        arreglo[k]=arreglo[k-1];
                        arreglo[k-1]=auxiliar;
                        cambios=true;
                }
            }
                if (cambios==false)
                        break;
                }
        }
}
```

Método burbuja (BubbleSort)


```
package Metodosdebusqueda;
public class QuickSort {
	public void OrdenarQuickSort(int[] arreglo)
	{
		arreglo = quicksort1(arreglo);
	}
	public int [] quicksort1(int numeros[])
	{
		return quicksort2(numeros,0,numeros.length-1);
	}
	public int[] quicksort2(int numeros[], int izq, int der)
	{
		if(izq>=der)
			return numeros;
		int i=izq, d=der;
		if(izq!=der)
		{
			int pivote;
			int aux;
			pivote = izq;
			while(izq!=der){
				while(numeros[der]>=numeros[pivote] && izq<der)
					der--;
				while(numeros[izq]<numeros[pivote] && izq<der)
					izq++;
				if(der!=izq)
				{
					aux = numeros[der];
					numeros[der]=numeros[izq];
					numeros[izq]=aux;}

				}
			if(izq==der){
				quicksort2(numeros,i,izq-1);
				quicksort2(numeros,izq+1,d);
			}

			}
		else
			return numeros;
		return numeros;
		}
}
```

Método QuickSort

Como se puede observar, el método QuickSort tiene más líneas de código que el método burbuja. **Esto no quiere decir que el método burbuja sea más eficiente**. Haciendo un análisis de coste computacional, comprobaremos que el último método es más eficiente y no dependerá de las características del sistema operativo o hardware.

Tanto el método Burbuja como el método Quicksort pueden ser comparados de tal manera que, a un determinado número de entradas, un método consuma más tiempo que otro. En el siguiente tema se verá como calcular la complejidad de un algoritmo y como en función de la entrada los rendimientos pueden variar.

Lo + recomendado

No dejes de leer...

Conceptos de programación orientada a objetos

Consulta la página de Oracle, en la que podrás encontrar información detallada de los conceptos básicos en cuanto a programación orientada a objetos.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://docs.oracle.com/javase/tutorial/java/concepts/

No dejes de ver...

Ventajas e inconvenientes de la POO

En este videotutorial se introducen las principales ventajas e inconvenientes de la programación orientada a objetos (el vídeo está en inglés).

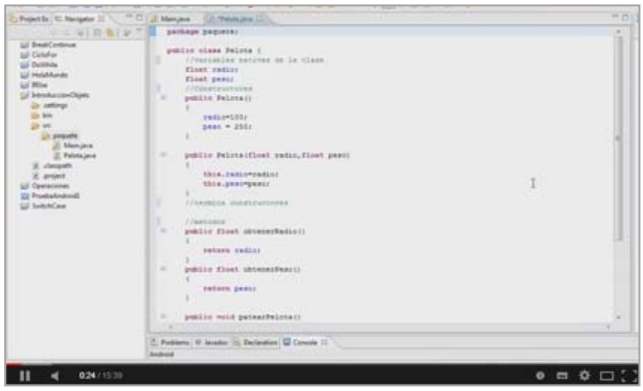
Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=SS-9y0H3Si8

Introducción a la POO

Vídeo que explica de manera clara y sencilla los conceptos más básicos de la programación orientada a objetos (el vídeo está en inglés).

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=P1WcKEgvRFE

Clases, objetos y métodos en Java

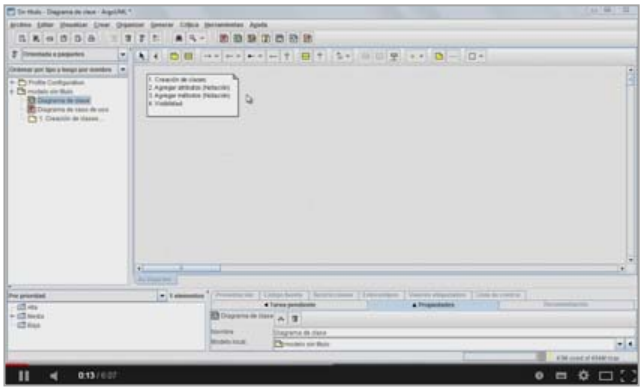


Este vídeo recoge los conceptos básicos de la programación orientada a objetos en Java.

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=AEXLtATMkZM

Diagrama de clases UML

En este vídeo podemos ver el diagrama de clases UML utilizando la herramienta ArgoUML.



Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

https://www.youtube.com/watch?v=rvDhp6agNRc

+ Información

A fondo

Conceptos básicos de programación orientada a objetos

En la documentación oficial de Oracle para Java se explican los conceptos básicos de programación orientada a objetos.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
https://docs.oracle.com/javase/tutorial/java/concepts/index.html

Ejemplos de diseño de clases y objetos

En la documentación oficial de Oracle para Java puedes encontrar diferentes tutoriales y ejemplos de diseño de clases y objetos.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
https://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Diseñar y programar, todo es empezar

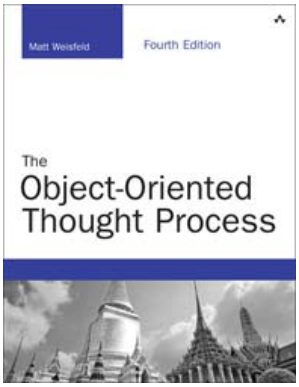
Vélez, J. (2011). *Diseñar y programar, todo es empezar: Una introducción a la programación orientada a objetos usando UML y Java*. Madrid: Dykinson.

Dentro de este libro podrás encontrar una descripción de cómo desarrollar *software* orientado a objetos desde el diseño hasta la implementación aplicando técnicas de ingeniería del *software*.

El libro está disponible en la Biblioteca Virtual de UNIR.

Aproximación a la programación orientada a objetos moderna

Weisfeld, M. (2009). *The object oriented Thought process*. Estados Unidos: Addison Wesley.

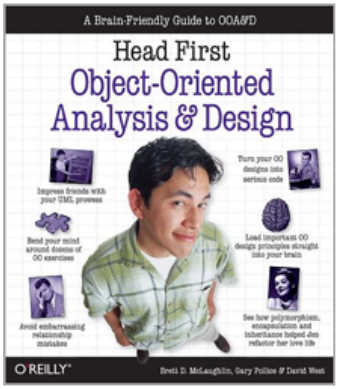


Este libro muestra una aproximación a la programación orientada a objetos moderna. Se tratan diferentes lenguajes de programación y muestra cómo se deben enfrentar los problemas dependiendo de la aplicación a la que nos enfrentemos.

Accede a una parte del libro desde el aula virtual o a través de la siguiente dirección web:
http://books.google.es/books?id=pxK0hSuDyX0C&printsec=frontcover

Aproximación al pensamiento orientado a objetos

Laughtlin, B. D. (2006). *Head First, object-oriented analysis and design*. O'Reilly.



Este libro muestra una aproximación al pensamiento orientado a objetos de una forma original y entretenida pero no por ello dejando de mostrar conceptos complejos.

Accede a una parte del libro desde el aula virtual o a través de la siguiente dirección web:
http://books.google.es/books?id=-QpmamSKl_EC&printsec=frontcover

Lenguajes de programación orientada a objetos

Pérez, M. (2014). *Lenguajes de programación orientada a objetos*. Estados Unidos: CreateSpace.

En este libro se profundiza en los lenguajes de programación orientados a objetos, en la gestión de excepciones, en el trabajo con librerías de clases, herencia, polimorfismos, etc. Un bloque amplio de contenido se ocupa de la implementación del paradigma orientado a objetos utilizando un lenguaje de programación concreto (en nuestro caso Java). Posteriormente se aborda el desarrollo de aplicaciones en el modelo de programación web. También se dedica una parcela importante al acceso a bases de datos relacionales. En un último bloque se abordan temas relativos a la calidad en el desarrollo del *software*, documentación y pruebas, finalizando el contenido profundizando en el proceso de ingeniería del software, sus fases, modelos del proceso de ingeniería, requisitos, metodologías de desarrollo orientado a objetos y herramientas CASE.

Recursos externos

Eclipse

Eclipse es una plataforma de desarrollo. Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene en mente un lenguaje específico, sino que es un IDE genérico.

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
https://www.eclipse.org/downloads/

En el siguiente vídeo puedes ver cómo crear un proyecto en Eclipse desde cero:
https://www.youtube.com/watch?v=J9lkAK0L16I

Test

1. ¿Cuál crees que es la mejor descripción de clase?

A. Conjunto de variables que se unen para solucionar un problema.

B. Conjunto de métodos y variables que se unen para resolver un problema.

C. Generalización de las características de elementos que forman parte de un problema.

D. Especialización de los elementos que forman parte del problema de tal forma que son los elementos más pequeños que se pueden identificar.

2. ¿Qué significa instanciar una clase?

A. Eliminar una clase.

B. Copiar una clase

C. Implementar una clase

D. Crear un objeto.

3. Cuando hablamos de objetos, qué sentencia es verdad:

A. Un objeto se crea siempre a partir de una clase.

B. Un objeto se puede crear a partir de otro objeto.

C. Un objeto se puede crear sin valores.

D. Un objeto se crea a partir de las variables que luego lo van a formar.

4. Si estamos definiendo un problema para gestionar un taller mecánico y decimos que de un coche queremos guardar su matrícula, kilometraje y cilindrada. ¿A qué nos estamos refiriendo?

A. A los atributos de la clase coche.

B. A lo métodos de la clase coche.

C. Estamos identificando clases que formarán parte del problema.

D. A los objetos que luego utilizaremos en la resolución del problema.

5. Los métodos *getters* se utilizan para:

A. Asignar el valor de los atributos a clases que no están completamente implementadas.

B. Asignar el valor a los atributos de una clase.

C. Obtener el valor de los atributos de una clase.

D. No existen dentro de la definición de una clase.

6. Los métodos *setters* se utilizan para:

A. Obtener el valor de los atributos a clases que no están completamente implementadas.

B. Asignar el valor a los atributos de una clase.

C. Obtener el valor de los atributos de una clase.

D. No existen dentro de la definición de una clase.

7. Respecto a los métodos de una clase, ¿qué sentencia es verdad?

A. Los métodos de una clase identifican el comportamiento de la clase.

B. Los métodos de una clase siempre tienen que tener parámetros.

C. Los métodos de una clase solo pueden ser utilizados por otras clases.

D. Los métodos de una clase se definen cuando se crea el objeto.

8. Los constructores

A. Son invocados cuando se crea una clase.

B. Las clases no tienen constructores son los objetos los que los tienen.

C. Los constructores son métodos públicos porque deben ser invocados por otros objetos.

D. Los constructores no pueden tener parámetros.

9. En Java:

A. La visibilidad indica desde donde es accesible un atributo.

B. Las clases deben estar en archivos separados.

C. En un programa puede existir más de una clase `main`.

D. Los atributos son siempre tipos definidos por el usuario.

10. En UML las clases gráficamente:

A. Se identifican con un rectángulo que puede estar divido o no en tres partes.

B. Solo se pone el nombre y los métodos dentro de un rectángulo.

C. Solo se ponen los atributos dentro de un rectángulo.

D. Se pone el nombre y los atributos dentro de un rectángulo.