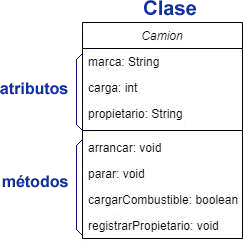
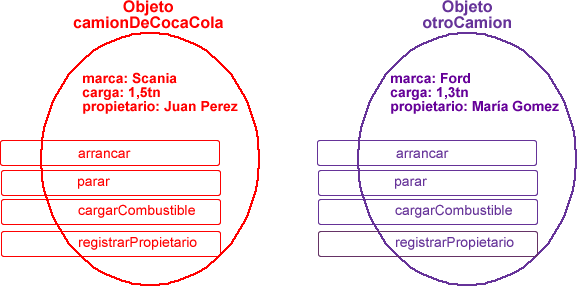
# CLASES Y OBJETOS



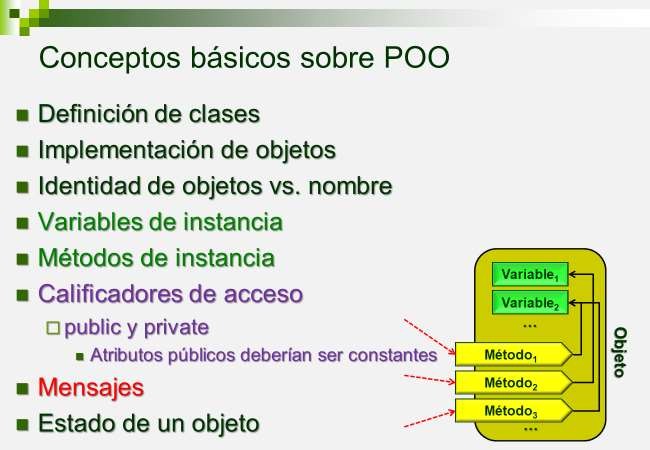
La definición de una clase describe un nuevo tipo de datos, con atributos y métodos que le son propios. Luego, deben definirse variables de este tipo o clase. Cada una de estas variables será un objeto, una instancia o un ejemplar de la clase. Los términos objeto, instancia o ejemplar de una clase son sinónimos en la jerga de la POO.

Así para nuestra clase camión, podemos definir dos objetos o ejemplares de la clase:



Debemos observar que cada objeto, ocupa un lugar en la memoria (se mapean en el heap), y tiene sus propios valores para los atributos. Así, el objeto camionDeCocaCola es un ejemplar o instancia de la clase Camion, distinta al ejemplar o instancia otroCamion.

Tanto el camionDeCocaCola como el camión otroCamion, por ser camiones, tienen los mismos comportamientos: ambos se pueden arrancar, parar, cargarCombustible() o registrarPropietario().



Para comprender mejor la idea de clase y objetos, veamos cómo definir en Java2 la clase Camión de nuestro ejemplo.

public class Camion {

/\* Atributos \*/

public final String marca; private int carga;

private String propietario;

/\* Métodos o comportamientos \*/

/\* Constructor \*/

public Camion(String marca, int carga, String propietario){ this.marca = marca;

this.carga = carga; this.prop = propietario;

}

public void setCarga(int kg){ if(kg>0)

this.carga = kg;

}

public int getCarga(){ return carga;

}

public void setProp(String prop){ this.prop=prop;

}

public String getProp(){ return this.prop;

}

public void arrancar(){

System.out.println(“arranca camión” + marca);

}

public void parar(){

System.out.println(“detiene camión”);

}

public void cargarCombustible(){

System.out.println(“carga combustible”);

}

public void registrarPropietario(String prop){ propietario = prop;

System.out.println(“registra propietario” + propietario);

}

}

Las clases son el bloque primario de construcción de programas orientados a objetos. Aquí no existen funciones o procedimientos aislados, sino que todo: atributos y métodos se engloba dentro del bloque de cada clase.

El código que declara una clase enuncia los atributos y métodos propios de esa clase o TAD, es una declaración de tipo, pero no asigna espacio en memoria para las variables estructuradas de ese tipo o de esa clase que sea necesario utilizar en el programa.

La declaración de una clase comienza por la palabra reservada **class** seguida del nombre que daremos a este nuevo tipo de datos.

## Bloque de la clase

Cualquier cosa dentro de las llaves de apertura y cierre de una clase está en el ámbito de la clase.

## Atributos, métodos y calificadores de acceso

Los atributos y métodos que son miembro de una clase deben ser identificados mediante su ***nombre*** y su ***tipo***; y deben ser ***calificados*** individualmente (si no se califican, se consideran como friendly).

## ¿Para qué se califican los atributos y métodos? - Encapsulamiento

Tanto los atributos como los métodos que son miembro de una clase pueden ser calificados . Los calificadores, que son palabras reservadas del lenguaje Java, pero que existen como concepto en cualquier lenguaje OO, se denominan Calificadores o Modificadores de Acceso a los Miembros.

* ***public***: cualquier variable de instancia o método declarado con el modificador de acceso public estará accesible en cualquier punto del programa que tenga acceso a un objeto de la clase.
* ***private***: cualquier variable de instancia o método declarado como private sólo será accesible desde los métodos de la clase.

Luego veremos, cuando incorporemos los conceptos de herencia y de administración de paquetes, las implicancias de calificar a una variable de instancia o método como protected.

Calificar atributos y métodos permite encapsular u ocultar los detalles de la implementación de la clase hacia fuera. Ciertamente una persona puede conducir un vehículo y utilizar sus servicios sin necesidad de conocer en detalle cómo trabaja el motor. Esto tiene que ver con el concepto de **encapsulamiento**.

## Objetos – Instancias de una clase

Con la definición de la clase Camión que dimos previamente, no hemos todavía construido ningún objeto o ejemplar de este tipo. Para hacerlo, debemos agregar el código en el cual se construyen (se instancian o mapean en la memoria) objetos de esta clase.

En nuestro ejemplo vamos a construir las dos instancias camionDeCocaCola y

otroCamion que propusimos esquemáticamente antes.

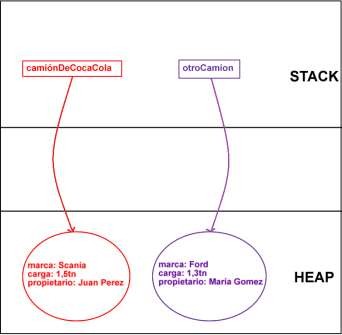
Camion camionDeCocaCola = new Camion(“Scania”, 1.5f,”Juan Perez”)

Camion otroCamion = new Camion(“Ford”,1.3f,“Maria Gomez”)

La palabra new es una palabra reservada en Java que invoca al constructor de la clase. El resultado de activar el constructor es efectivamente construir un ***objeto*** o una ***instancia*** de la clase. Es decir, mapear el objeto en la memoria.

En Java, cualquier tipo primitivo (int, char, float, double, etc.) se maneja por valor y cualquier tipo no primitivo (los TAD o clases que definimos) se maneja por referencia. Es decir, se mapea en la memoria dinámica asignada en el Heap.

Por lo tanto, como resultado de la activación del constructor, new devuelve una referencia, un puntero, al objeto que el constructor mapeó en el Heap. Las variables camionDeCocaCola y otroCamion son variables locales en el Stack que apuntan al objeto que está en zona de Heap.



## Variables y Métodos de Instancia

Como podemos observar en la figura anterior, los atributos de la clase Camión pueden tomar distintos valores para cada objeto que efectivamente se ha construido en memoria. La marca del camionDeCocaCola es “Scania”, mientras que la marca de otroCamion es “Ford” por citar un ejemplo. Por eso se llama a esos atributos **Variables de instancia** o **Variables de Ejemplar**, dado que existe una instancia de ellos para cada ejemplar de la clase***.***

Del mismo modo, no es lo mismo arrancar o registrar el propietario del camionDeCocaCola, que arrancar o registrar el propietario del otroCamion. Por ello, los métodos que se activan desde un objeto particular (una instancia o ejemplar particular de la clase) se llaman **Métodos de instancia**.

### Ejemplo

1. camionDeCocaCola.registrarPropietario(“Juan Perez”);
2. otroCamion.registrarPropietario(“Juan Perez”);

*En 1. se activa el método de instancia registrarPropietario del objeto camionDeCocaCola que asigna un valor a la variable de instancia propietario de ese objeto.*

*En 2. se activa el método de instancia registrarPropietario del objeto otroCamion que asigna un valor a la variable de instancia propietario del otro camion.*

## Constructores

Un constructor es un método que tiene el mismo nombre de la clase y aunque puede recibir parámetros, no debe ser calificado con ningún tipo de devolución, ni siquiera void.

En la definición de nuestra clase Camión, el método constructor es:

/\* Constructor \*/

public Camion(String marca, int carga, String propietario){ this.marca = marca;

this.carga = carga; this.prop = propietario;

}

La llamada o activación del constructor de una clase, mapea un objeto de dicha clase en la memoria dinámica y devuelve una referencia a dicho objeto. En general los constructores se utilizan también para inicializar los valores de los atributos de la clase.

Así en nuestro ejemplo, el constructor de Camión recibe argumentos marca, carga y propietario e inicializa con estos valores los atributos definidos en la clase.

## Constructor por defecto

En Java no se exige al programador definir explícitamente en su código un constructor para cada clase. Si el programador no implementa un constructor, el compilador proveerá uno por defecto cuando compile el código de la clase.

El constructor por defecto es un constructor sin parámetros que inicializa a cero los atributos numéricos, a false los booleanos y con null los que sean referencia a otros objetos.

Para activar el constructor por defecto, se deberá utilizar new NombreDeLaClase();

Si el programador implementa un constructor para la clase, entonces el compilador no provee el constructor por defecto y el mismo, por tanto, no puede ser utilizado para instanciar objetos de esa clase. Es decir, al agregar un constructor explícito o cualquier otro ya no podré acceder al constructor por defecto, es como si dejase de existir.

## Sobrecarga de Constructores

Una clase puede tener más de un constructor, el cual debe diferenciarse en cantidad, orden y/o tipo de parámetros de los otros constructores definidos.

En nuestro ejemplo podríamos agregar otros constructores a la clase Camión public class Camion {

…

/\* Constructores \*/

public Camion(String marca, int carga, String propietario){ this.marca = marca;

this.carga = carga; this.prop = propietario;

}

/\* Constructor que solo inicializa el atributo propietario \*/ public Camion(String propietario){

this.prop = propietario;

}

}

Los constructores, que se diferencian por su signatura (cantidad, tipo u orden de los parámetros formales), se llaman **constructores sobrecargados**.

Un constructor, que recibe como argumento un objeto del mismo tipo que se está construyendo, y lo utiliza para copiar los valores de sus atributos se llama **Constructor de Copia**.

Veamos cómo utilizar el constructor de copia para construir otro objeto del tipo Camión.

1. Camion camionDeCocaCola = new Camion(“Scania”, 1.5f,”Juan Perez”);
2. Camion otroCamion = new Camion(“Ford”,1.3f,“Maria Gomez”);

## Referencia this

En Java, cada objeto o ejemplar de una clase tiene acceso a una referencia a sí mismo, llamada referencia this. De este modo, this es un apuntador al **objeto activo**, es decir, el objeto desde el que se activa un método de instancia o se accede a un atributo.

Así, en nuestro ejemplo, cuando activamos el constructor de Camión en la línea

Camion camionDeCocaCola = new Camion(“Scania”, 1.5f,”Juan Perez”);

La referencia “this.marca” vincula al atributo o variable de instancia “marca” del objeto activo; mientras que “marca” referencia al parámetro que contiene el valor “Scania” recibido como argumento.

## Métodos set/get

Es una práctica común en la POO que las clases ofrezcan métodos public que permitan a los clientes de la clase (los que usan sus servicios) asignar (o setear - set) valores a los atributos privados de la clase; u obtener (get) valores de dichos atributos.

Estos métodos se llaman, por convención (aunque no es obligatorio llamarlos así), métodos set/get respectivamente seguidos del nombre del atributo al que se quiere acceder.

En nuestro caso, los métodos setProp y getProp, permiten acceder al atributo propietario para asignar u obtener su valor. De igual forma lo hacen los métodos setCarga y getCarga sobre el atributo privado carga.

En este punto, el alumno puede preguntarse ¿Qué sentido tiene declarar un atributo privado, y después proveer métodos públicos que permitan modificar u obtener su valor desde afuera de la clase? o ¿Por qué no directamente, hacer público al atributo propietario o al atributo carga?

La respuesta a estas preguntas se ilustra en la trivial validación que hemos introducido en el método setProp, en la cual se controla que el propietario asignado no sea una cadena vacía.

public void setProp(String prop){

if (prop != “”){

this.prop=prop;

}

}

O la que introdujimos en el método setCarga, en la cual validamos que el valor a setear sea mayor a cero.

public void setCarga(int kg){ if(kg>0)

this.carga = kg;

}

Si los atributos “propietario” y “carga” fueran públicos en la clase Camión, desde cualquier otra clase se podrían modificar sus valores, sin controles que aseguren consistencia en la manipulación de los datos. Además, al crear un método especial setProp y setCarga para acceder a dichos atributos, se establecen estos controles en un único lugar. De esta forma, se favorece el criterio de **reutilización de código** que es uno de los objetivos perseguidos por la POO.

Reutilizar código no es copiar y pegar código prescrito en diversas partes de un programa, sino referenciar siempre al mismo bloque de código, en un único lugar, en todas aquellas oportunidades que se necesite implementar la función que el mismo provee. Esto favorece luego la facilidad de **mantenimiento y escalabilidad** de las aplicaciones. Estas dos características: mantenimiento y escalabilidad (capacidad de las aplicaciones para crecer e incorporar nuevas funcionalidades) son también metas que la POO intenta mejorar con relación a modelos de programación previos como la programación estructurada.