Programación 2

Introducción a la Orientación a Objetos

Profesor: Eduardo Godoy
eduardo.gl@gmail.coml
Material elaborado por Rodrigo Olivares
rodrigo.olivares@uv.cl

16 de marzo de 2018

Contenido

Introducción

Antes de comenzar

¿Qué es la Orientación a Objeto?

De manera sintetizada, la orientación a objetos es un **paradigma** de programación.

¿Qué es un paradigma?

Un paradigma de programación es un modelo básico de diseño y desarrollo de soluciones, con directrices específicas, tales como: estructura, cohesión, acoploamiento, etc.

En general

El Paradigma de Orientación a Objeto es **una forma** de entender un problema, identificando las principales **entidades** que se encuentran en él.

Introducción

Antes de comenzar

¿Qué es un Lenguaje de Programación?

Es la **herramienta** seleccionada, para dar solución al problema detectado.

En relación a los Lenguaje de Programación

Se es libre de utilizar la herramienta con la cual se esté más habituado. La solución final no cambia.

Corolario

El Paradigma de Orientación a Objeto, no guarda relación con el Lenguaje de Programación seleccionado.

Introducción

Antes de comenzar

- Tipos de paradigma:
  - **Procedural o imperativo:**
    - Se decribe como una secuencia instrucciones o comandos que cambian el estado de un programa. El código máquina en general está basado en el paradigma imperativo.
  - **Declarativo:**
    - Se basa en el **qué** hacer y no en el cómo. Está basado en el desarrollo de programas especificando o **declarando** un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.
  - **Lógico:**
    - Se basa en la definición de reglas lógicas para luego, a través de un motor de inferencias lógicas, responder preguntas planteadas al sistema y así resolver problemas.

Introducción

Antes de comenzar

- Tipos de paradigma:
  - **Estructurado:**
    - La programación se divide en **bloques** (procedimientos y funciones) que pueden o no comunicarse entre sí. Además la programación se controla con secuencia, selección e iteración. Su principal ventaja es la *estructura clara* lo que entrega una mejor comprensión de la programación.
  - **Funcional:**
    - Concibe a la computación como la evaluación de funciones aritméticas y evita declarar y cambiar datos. En otras palabras, hace hincapié en la aplicación de las funciones y composición entre ellas.
  - **Orientado a objetos:**
    - Utiliza objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo abstracción, encapsulamiento, ocultamiento, herencia, polimorfismo, entre otras.

Introducción

Conceptos

Programación Orientada a Objeto

La **POO** se basa en la dividir el programa en pequeñas unidades lógicas de código. A estas pequeñas unidades lógicas de código se les denomina **objetos**. Los objetos son unidades independientes que se comunican entre ellos.

La **POO** proporciona técnicas con las cuales se modela y representa el mundo real (tan fielmente como sea posible).

Introducción

Conceptos

Qué es un Objeto?

Cualquier cosa que vemos a nuestro alrededor. Ej: auto.

Un objeto, en general, está compuesto por:

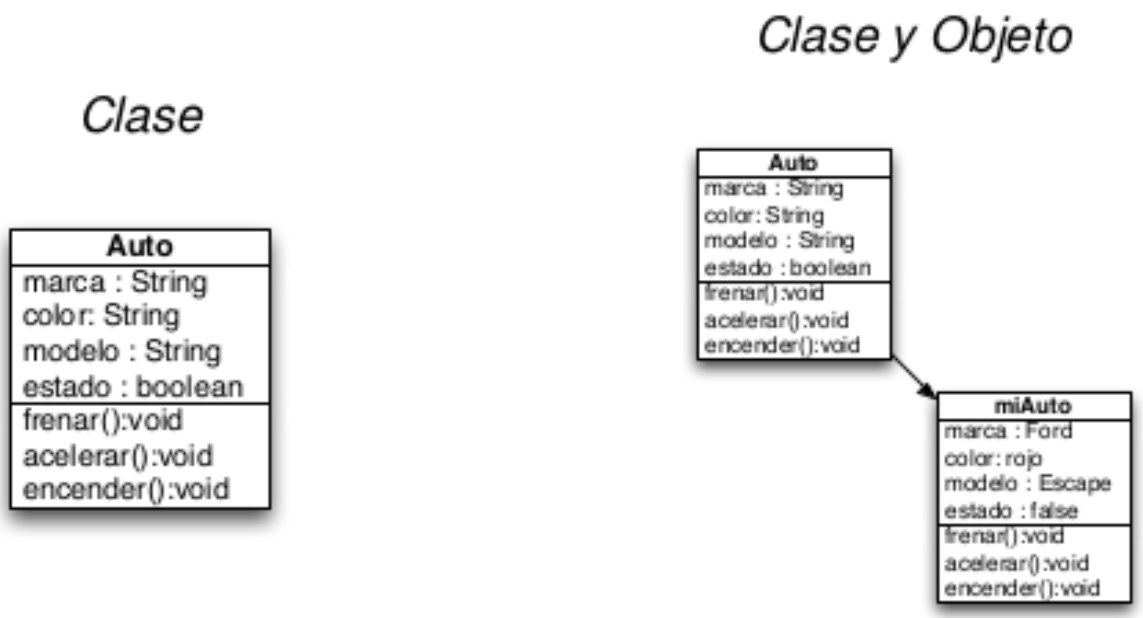
- Características / **atributos** (marca, modelo, color, etc.)
- Comportamiento / **métodos** (encender, acelerar, frenar, retroceder, etc.)

Introducción

Conceptos

Objetos y Clases

Los programas en POO están construidos en base a objetos con características (atributos) y comportamiento (métodos) específicos y que pueden comunicarse entre sí.



Introducción

Conceptos

- **Clases:**

  Modelo para múltiples objetos con características y comportamientos similares. Las clases comprenden todas las características de una serie particular de objetos. En **POO** se definen clases como un modelo abstracto de un objeto. En programación estructurada sería un tipo de dato (*en C sería struct o typedef*)

- **Instancias:**

  Representación concreta de un objeto. Al definir una clase se pueden crear muchas instancias de la misma y cada instancia puede tener diferentes características mientras se comporte y reconozca como objeto de la clase. En programación estructurada sería una *variable*.

Introducción

Conceptos

- **Atributos:**

  Características que diferencian a un objeto de otro y determinan la "*forma*" de ese objeto.
  Los atributos se definen como **variables**. De hecho podrían, ser vistas como **variables globales del objeto**. Cada instancia de una clase puede tener diferentes valores para sus variables, por lo que a cada variable se le denomina **variable de instancia**. La clase define el tipo de atributo y cada instancia guarda su propio valor para ese atributo.

Introducción

Conceptos

- **Métodos:**

  Comúnmente denominadas **funciones** o **procedimientos** definidas dentro de una clase, y que operan en sus instancias. Los objetos se **comunican** entre sí mediante el uso **llamadas a de métodos**. Se dice que una clase puede **llamar** al método de otra clase.
  Se pueden definir métodos de instancia (que operan en una instancia de la clase) y los métodos de clase que operan sobre la clase.

Introducción

Principios importantes

- **Abstracción:**
  - Capacidad de analizar y representar las características esenciales de fenómenos complejos.
  - Una clase es una abstracción o una entidad esencial del problema o la solución.
  - Ejemplo:

    Qué características tienen de los autos?
    Qué comportamiento tienen los autos?
  - En la **POO** el concepto de clase es la representación y el mecanismo por el cual se gestionan las abstracciones.

Introducción

Principios importantes

- **Encapsulamiento:**
  - Técnica para proteger y ocultar el estado interno y el *conocimiento* de una entidad.
  - En el paradigma de la orientación a objetos, la clase/objeto es la unidad fundamental de encapsulamiento.
  - En la POO la utilidad del encapsulamiento va por la facilidad de manejar la complejidad, ya que tendremos clases (como cajas negras) donde sólo se conoce el comportamiento pero no los detalles internos

Introducción

Principios importantes

- **Ocultamiento de información:**
  - Es la capacidad de ocultar los detalles internos del comportamiento de una clase y exponer sólo los detalles que sean necesarios para el resto del sistema.
  - Cada tipo de objeto expone una **interfaz** que especifica como pueden interactuar con los objetos de la clase.
  - En la **POO** el objeto poseé (al menos) dos vistas:

    Restringir el uso de la clase, pues habrá cierto comportamiento **privado** que no podrá ser accedido por otras clases.
    Controlar el uso de la clase, pues se darán ciertos mecanismos para modificar el estado de una clase.
  - En Java, el ocultamiento se logra utilizando los modificadores de acceso: public, private y protected delante de las variables y método.

Introducción

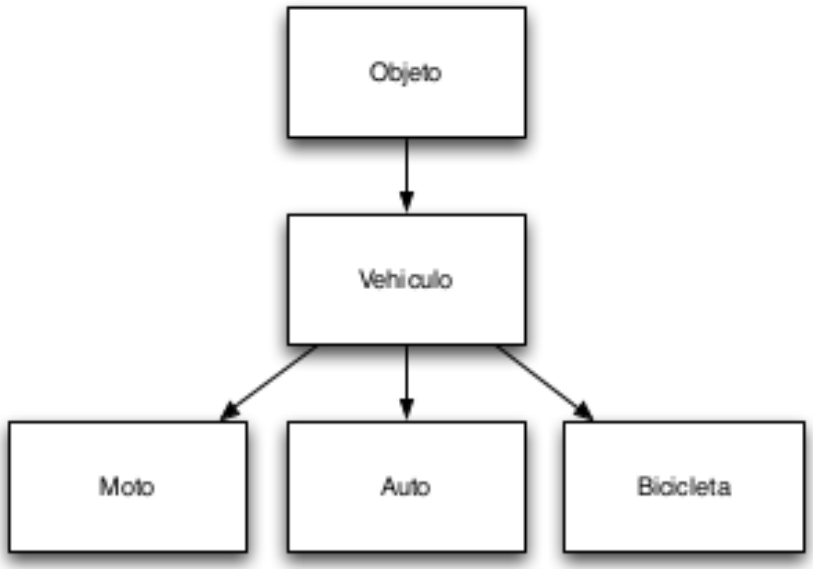
Principios importantes

- **Herencia:**
  - Organización jerárquica de clases.
  - Cada clase tiene una superclase y puede tener una o más subclases.
  - Las subclases heredan todos los atributos y métodos de la superclase, esto significa que las subclases incorporan a sus atributos y métodos propios, los atributos y métodos heredados de la superclase.
  - En Java la herencia se logra usando la palabra reservada: **extends**.
  - En la parte superior de la jerarquía de clase de Java, está la clase **Object**. Todas las clases heredan de esta superclase y cada clase hacia abajo agrega más información.

Introducción

Principios importantes

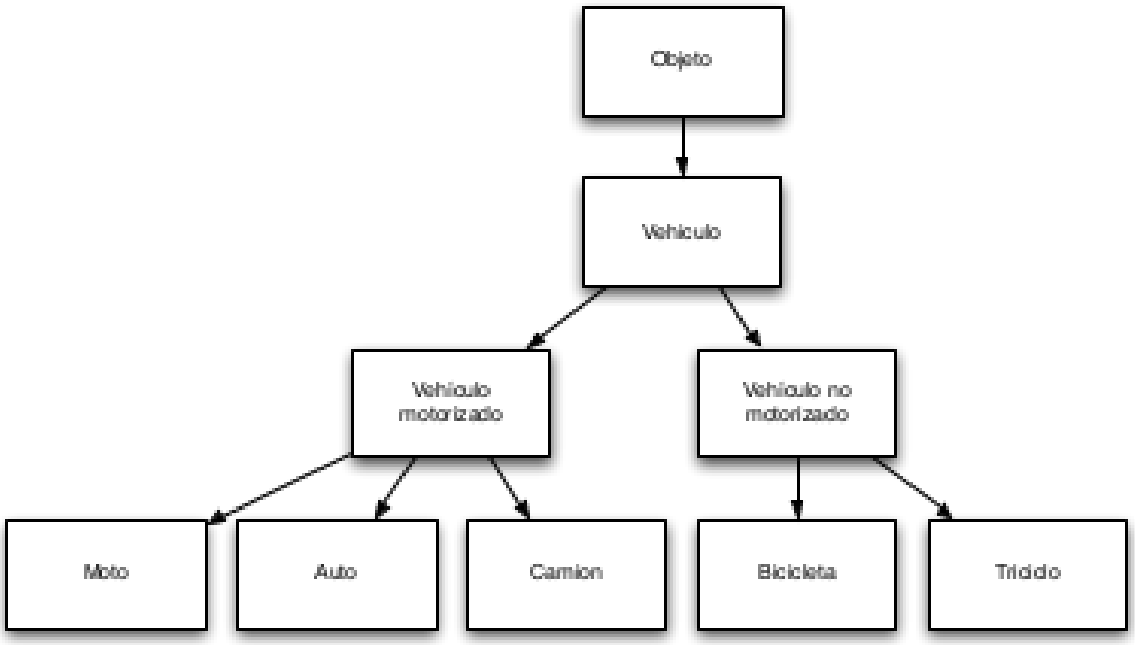
Jerarquía de herencia



Introducción

Principios importantes

Jerarquía de herencia



Introducción

Principios importantes

- **Polimorfismo:**
  - Literalmente, **Poli (muchos/as) - Morphus(formas)**.
  - En la **POO** es la propiedad que le permite a métodos con el mismo nombre implementar distintas funcionalidades, según las clases donde se apliquen. Es decir, métodos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre. Al llamarlos se utilizará el comportamiento correspondiente al objeto que se esté usando.
  - **Beneficios:** Código más genérico. Maximiza la calidad de reuso y extensibilidad.
  - **Ejemplo:**
    - Operador + para sumar enteros, reales, etc y concatenar cadenas de caracteres.

Introducción

Principios importantes

- **Envío de mensajes:**
  - Un objeto es inútil si está aislado.
  - Los objetos de un programa interactúan y se comunican entre ellos por medio de **mensajes**.
  - Cuando un **objeto A** quiere que un **objeto B** ejecute una de sus funciones (métodos del objeto B), el objeto A **envía un mensaje** al objeto B. Los mensajes son invocaciones a los métodos de los objetos.
  - Ejemplo: si objeto *miAuto* debe acelerar.

    El objeto al cual se manda el mensaje (miAuto).
    El método que debe ejecutar (acelerar()).
    Los parámetros que necesita ese método (10 km/h).
  - Estas tres partes del mensaje (objeto destinatario, método y parámetros) son suficiente información para que el objeto que recibe el mensaje, ejecute el método.

Modelamiento

Ejemplos

Modelar las siguientes situaciones

- → Estacionamiento de automóviles
- → Terminal de punto de ventas
- → RentaCar (arriendo de vehículos)
- → Sistema telefónico (Smart-Phone)
- → Banca Persona
- → Portal de alumnos (UV)

Lenguajes de programación

Inicio de Java

Inicio de Java

- **Desarrollado por:** Sun Microsystems en 1991.
- **Propietario actual:** Oracle Corporation desde 2010.
- **Objetivo inicial y actual:**
  - Desarrollar un lenguaje de programación para crear software pequeños, rápidos, eficientes y portátiles para diversos dispositivos de hardware (teléfonos celulares, radiolocalizadores y asistentes digitales personales).
  - Ser el nexo universal que conecte a los usuarios con la información que esté situada en el computador local, en un servidor Web o en una base de datos.
- **Principio:** *Write Once, Run Everywhere*

Lenguajes de programación

Características

Características

- **Independencia de plataforma**, tanto a nivel del código fuente como del binario.
- *Diferencia entre código fuente y binario?*
  - *Independencia en código fuente*: los tipos primitivos de datos de Java tienen tamaño consistentes en todas las plataformas de desarrollo. Las bibliotecas de Java facilitan la escritura del código, que puede desplazar se plataforma a plataforma.
  - *Independencia en binario*: los archivos binarios (bytecodes) pueden ejecutarse en distintas plataformas sin necesidad de volver a compilar la fuente.

Lenguajes de programación

Ventajas

Ventajas

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Elimina redundancia a través de la herencia y polimorfismo
- Agiliza el desarrollo de software.

Lenguajes de programación

Ventajas

Ventajas

- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software.
- Recolección de basura.
- En Java no hay punteros (simplicidad).
- Las cadenas y los arreglos son objetos reales
- La administración de la memoria es automática

Lenguajes de programación

Ambiente de desarrollo Java

Compilador

Genera los ficheros compilados en bytecode (en vez de generar código de máquina) (extensión *.class) a partir del código fuente (extensioón *.java). El bytecode será el conjunto de instrucciones independientes de la plataforma.

Lenguajes de programación

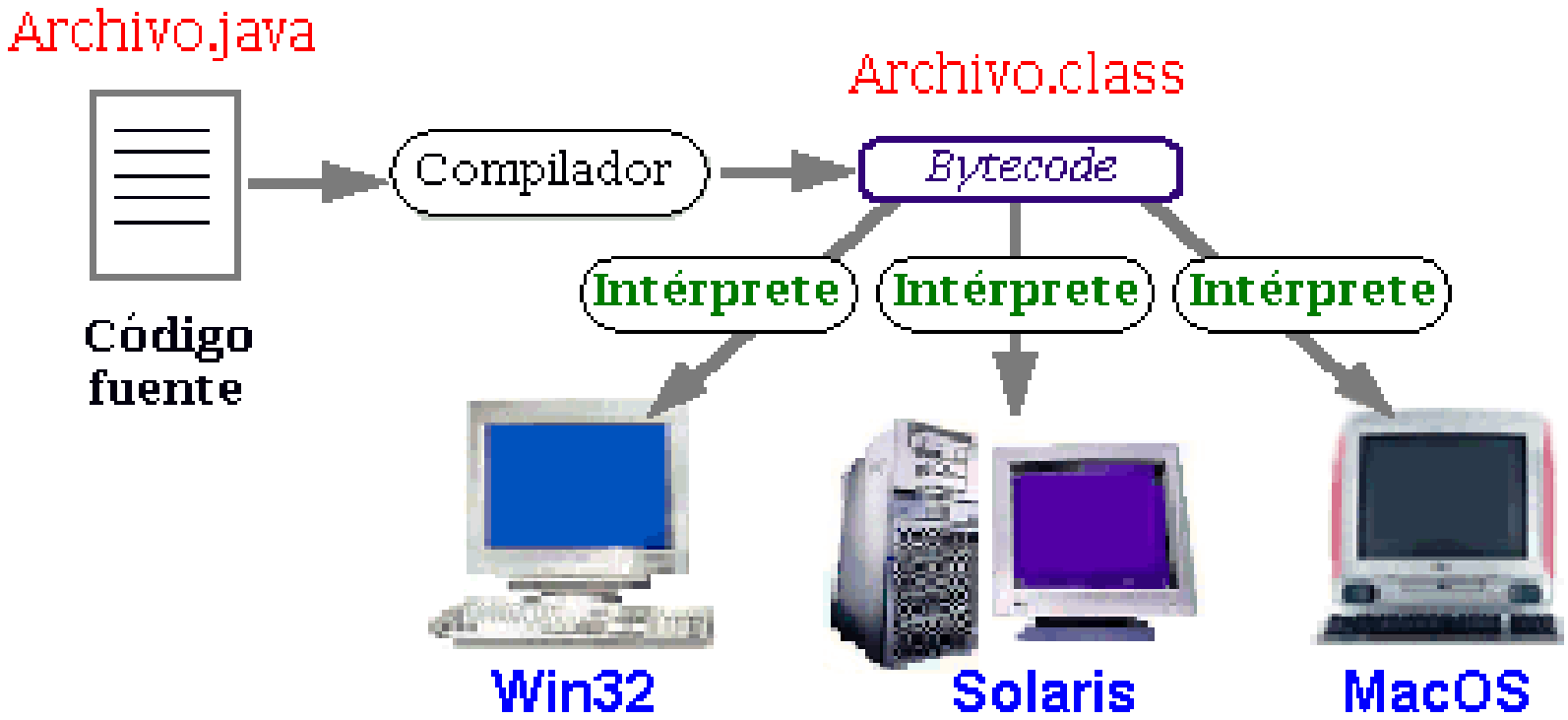
Ambiente de desarrollo Java

Interprete

Denominado Java Virtual Machine (JVM). Interpreta el bytecode (código neutro) convirtiendo a código particular de la CPU utilizada, lo que permite la ejecución el programa. JVM es una aplicación que simula una computadora, pero oculta el sistema operativo y el hardware subyacente.

Lenguajes de programación

Compilación, interpretación y ejecución



Lenguajes de programación

Kit de desarrollo Java (JDK)

Kit de desarrollo Java (JDK)

Contiene las herramientas, el conjunto programas y bibliotecas (interfaces de programación de aplicaciones: APIs) necesarias para desarrollar, compilar y ejecutar programas en Java.

1. **javac**: Es el compilador de Java. Se encarga de convertir el código fuente escrito en Java a bytecode. Recibe como argumento todos los archivos de código fuente (con extension .java). Este comando no es parte de Java Runtine Environment (JRE) dado que JRE esta destinado únicamente a ejecutar código binario, no permite compilar. Ej: *javac Auto.java*
2. **java**: Es el intérprete de Java. Ejecuta el bytecode a partir de los archivos con extension .class. Recibe como argumento el nombre del binario ejecutable en formato bytecode sin la extension de archivo .class que identifica de manera visual un binario java. Este comando es parte de JRE y JDK. Ej: *java Auto*

Lenguajes de programación

Kit de desarrollo Java (JDK)

Kit de desarrollo Java (JDK)

Contiene las herramientas, el conjunto programas y bibliotecas (interfaces de programación de aplicaciones: APIs) necesarias para desarrollar, compilar y ejecutar programas en Java.

1. **jar**: Herramienta para trabajar con archivos JAR. Permite empaquetar las clases y archivos de Java para fabricar un único archivo contenedor de las aplicaciones, multimedia y gráficos. Este comando es parte solo de JDK. Ej: para crear un JAR *jar -cf Auto.jar Auto**, para extraer un JAR *jar -xf jar_file*
2. **javadoc**: Crea documentación en formato HTML a partir de el código fuente y los comentarios. Ej: *javadoc Auto.java*
3. **jdb**: Debugger permite detener la ejecución del programa en un punto deseado, lo que ayuda la detección y corrección de errores. Ej: se debe compilar *javac Auto.java* y luego *jdb Auto 10* (debug con punto de interrupción en la línea 10).

Lenguajes de programación

Fases para el desarrollo de un programa

Fases para el desarrollo de un programa

1. **Creación:** usar un editor para escribir el código fuente. Guardarlo con extensión **.java**. Pueden ser muy útil utilizar entornos de desarrollo (IDEs).
2. **Compilación:** se usa el comando **javac**. Ej compilación: *javac HelloWorld.java*. El resultado de esta fase es un archivo **.class**
3. **Cargar en memoria:** El cargador de memoria toma los archivos **.class** y los lleva a memoria principal.
4. **Verificación del bytecode:** Verifica que los bytecode sean válidos y no violen restricciones de seguridad.
5. **Ejecución:** La JVM ejecuta los bytecode usando el comando **java**. Las JVM usan una combinación de interpretación y de *compilación justo a tiempo (JIT).* La JVM analiza el código buscando las partes que se ejecutan con frecuencia, para traducirlas al lenguaje de máquina del computador, así cuando encuentra nuevamente el código lo ejecuta en lenguaje de máquina que es más rápido. Así lo programas en java pasan por dos fases de compilación una para traducir el código fuente a bytecode y la otra para traducir el bytecode a lenguaje de máquina. Ej ejecución: *java HelloWorld*

Lenguajes de programación

Variables

Definición

- Una **variable** es un **nombre** que contiene un valor que puede cambiar a lo largo del programa.

Tipos principales de variables

- De acuerdo con el tipo de información que contienen, en JAVA hay dos tipos principales de variables:
  - Variables de **tipos primitivos**. Están definidas mediante un único valor que puede ser **entero**, de **punto flotante**, **carácter** o **booleano**. JAVA permite distinta **precición** y distintos rangos de valores para estos tipos de variables **char, byte, short, int, long, float, double, boolean**.
  - Variables **referencia**. Las variables referencia son referencias o nombres de una información más compleja: **arrays** u **objetos** de una determinada clase.

Lenguajes de programación

Variables

Rol de las variables

- Desde el punto de vista del rol o misión que cumplan en el programa, las variables pueden ser:
  - Variables **miembros** de una clase. Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias. Comúnmente son llamados **atributos** de la clase.
  - Variables **locales**. Se definen dentro de un método o más en general dentro de cualquier bloque entre llaves $\{\}$. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque. Pueden ser también tipos primitivos o referencias.

Lenguajes de programación

Variables - Nombres de variables

Características de las variables

- Los nombres de variables en JAVA se pueden crear con mucha libertad.
- Pueden ser cualquier conjunto de caracteres numéricos y alfanuméricos, sin algunos caracteres especiales utilizados por JAVA como operadores o separadores (,.+-*/ etc).
- Existe una serie de palabras reservadas las cuales tienen un significado especial para JAVA y por lo tanto no se pueden utilizar como nombres de variables.

Lenguajes de programación

Variables - Palabras reservadas

Palabras reservadas

abstract	boolean	break	byte	case	catch
char	class	const*	continue	default	do
double	else	extends	final	finally	float
for	goto*	if	implements	import	instanceof
int	interface	long	native	new	null
package	private	protected	public	return	short
static	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while

(*) son palabras reservadas, pero no se utilizan en la actual implementación del lenguaje JAVA

Lenguajes de programación

Variables -Tipos de datos primitivos

Características de las variables

Se llaman **tipos primitivos** de variables de JAVA a aquellas variables sencillas que contienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto flotante.

Características de las variables

- Java dispone de **ocho** tipos primitivos de variables:
  - Un tipo para almacenar valores **true** y **false (boolean)**.
  - Un tipo para almacenar **caracteres (char)**.
  - Seis tipos para guardar valores **numéricos**.
    - Cuatro tipos para **enteros (byte, short, int y long)**
    - Dos para valores **reales de punto flotante (float y double)**.

Lenguajes de programación

Variables -Tipos de datos primitivos

Tipo de variable	Descripción
boolean	1 byte. Valores true y false.
char	2 bytes. Unicode. Comprende el código ASCII.
byte	1 byte. Valor entero entre -128 y 127.
Short	2 bytes. Valor entero entre -32768 y 32767.
int	4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647.
long	8 bytes. Valor entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.
float	4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38.
double	8 bytes (unas 15 cifras decimales equivalentes). De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308.

Lenguajes de programación

Variables -Tipos de datos primitivos

Los tipos primitivos de JAVA tienen algunas características importantes:

- El tipo **boolean** no es un valor numérico.
  - Sólo admite los valores **true** o **false**.
  - El tipo boolean no se identifica con el igual o distinto de cero, como en otros lenguajes.
  - El resultado de la expresión lógica que aparece como condición en un bucle o en una bifurcación debe ser boolean.
- El tipo **char** contiene caracteres en código UNICODE (que incluye el código ASCII), y ocupan 16 bits (2 bytes) por carácter. Comprende los caracteres de prácticamente todos los idiomas.

Lenguajes de programación

Variables -Tipos de datos primitivos

Los tipos primitivos de JAVA tienen algunas características importantes:

- Los tipos **byte, short, int y long** son números enteros que pueden ser positivos o negativos, con distintos valores máximos y mínimos. A diferencia de otros lenguajes, en JAVA no hay enteros **unsigned**.
- Los tipos **float** y **double** son valores de punto flotante (números reales) con 6-7 y 15 cifras decimales exactas (significativos), respectivamente.
- Se utiliza la palabra **void** para indicar la ausencia de un tipo de variable determinado.

Lenguajes de programación

Variables -Tipos de datos primitivos

Los tipos primitivos de JAVA tienen algunas características importantes:

- A diferencia de otros lenguajes, los tipos de variables en JAVA están perfectamente definidos en todas y cada una de las posibles plataformas. Por ejemplo, un int ocupa siempre la misma memoria y tiene el mismo rango de valores, en cualquier tipo de ordenador.
- Existen extensiones de JAVA para aprovechar la arquitectura de los procesadores Intel, que permiten realizar operaciones de punto flotente con una precisión extendida de 80 bits (10 bytes).

Lenguajes de programación

Variables - Definición de variables

Definición de variables en JAVA.

- Una variable se define especificando el tipo y el nombre de dicha variable.
- Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase.
- Si no se especifica un valor en su declaración, las variables primitivas deberán ser inicializadas antes de su uso, sino, JAVA no reconocerá la variable en el contexto actual.

Lenguajes de programación

Variables - Definición de variables

Definición de variables en JAVA.

- Es importante distinguir entre la **referencia** a un objeto y el **objeto** mismo.
- Una referencia es una variable que indica dónde está guardado un objeto en la memoria del ordenador.
  - A diferencia de otros lenguajes, JAVA no permite acceder al valor de la dirección, pues en este lenguaje se han eliminado los **punteros**.
- Al declarar una referencia, ésta aún no se encuentra "apuntando" a ningún objeto en particular y por eso se le asigna el valor **null**.
- Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el objeto utilizando el operador **new**. Este operador reserva en la memoria del ordenador espacio para ese objeto.

Lenguajes de programación

Variables - Visibilidad y vida de las variables

Definición

- Se entiende por **visibilidad, ámbito, alcance o scope** de una variable, el fragmento o parte de la aplicación donde dicha variable es accesible y por lo tanto puede ser utilizada en una expresión.
- En JAVA todas las variables deben estar incluidas en una clase.
- En general las variables declaradas dentro de unas llaves $\{\}$, es decir dentro de un bloque, son visibles y existen dentro de estas llaves.

Lenguajes de programación

Operadores - Aritméticos

Operadores aritméticos

→ Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: **suma** (+), **resta** (-), **multiplicación** (*), **división** (/) y **resto de la división** (%).

Lenguajes de programación

Operadores - Asignación

Operadores de asignación

- Los operadores de asignación permiten asignar un valor a una variable.
- El operador de asignación por excelencia es el operador **igual** (=).
- La forma general de las sentencias de asignación con este operador es:
  - **variable** = **expresion**;
- JAVA dispone de otros operadores de asignación. Se trata de las versiones abreviadas del operador (=) que realizan operaciones "acumulativas" sobre una variable.

Lenguajes de programación

Operadores - Asignación acumulativas

Operador	Uso	Expesión equivalente
$+=$	op1 $+=$ op2	op1 $=$ op1 $+$ op2
$-=$	op1 $-=$ op2	op1 $=$ op1 $-$ op2
$*=$	op1 $*=$ op2	op1 $=$ op1 $*$ op2
$/=$	op1 $/=$ op2	op1 $=$ op1 $/$ op2
$\%=$	op1 $\%=$ op2	op1 $=$ op1 $\%$ op2

Lenguajes de programación

Operadores - Unarios e Instanceof

Operadores unarios

Los operadores **más** (+) y **menos** (-) unarios sirven para mantener o cambiar el signo de una variable o expresión numérica.

Operador **instanceof**

- El operador **instanceof** permite saber si un objeto pertenece o no a una determinada clase.
- Es un operador binario cuya forma general es:
  - objectName **instanceof** ClassName
- Devuelve **true o false** según el objeto pertenezca o no a la clase.

Lenguajes de programación

Operadores - Condicional ?

Operador condicional ?

- Este operador permite realizar bifurcaciones condicionales sencillas.
- Su forma general es la siguiente:
  - expresionBooleana **?** respExpresionVerdero : respExpresionFalso
- Se evalúa **expresionBooleana** y devuelve **respExpresionVerdero** si es verdadera o **respExpresionFalso** si es falsa.
- Es el único operador ternario en JAVA (requiere de tres argumentos).

Lenguajes de programación

Operadores - Incrementales

Operadores incrementales

- JAVA dispone del operador **incremento** $(++)$ y **decremento** $(--)$.
- El operador $(++)$ incrementa en una unidad la variable a la que se aplica, mientras que $(--)$ la reduce en una unidad.
- Estos operadores se pueden utilizar de dos formas:
  - **Precediendo** a la variable **$(++x)$**. En este caso primero se utiliza la variable en la expresión (con el valor actual) y luego se incrementa.
  - **Siguiendo** a la variable **$(x++)$**. En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.

Lenguajes de programación

Operadores - Relacionales

Operadores relacionales

- Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor.
- El resultado de estos operadores es siempre un valor **boolean (true o false)** según se cumpla o no la relación considerada.
- Estos operadores se utilizan con mucha frecuencia en las **bifurcaciones** y en los **bucles**.

Lenguajes de programación

Operadores - Relacionales

Operador	Uso	El resultado es true si:
$>$	op1 $>$ op2	op1 es mayor que op2
$>=$	op1 $>=$ op2	op1 es mayor o igual que op2
$<$	op1 $<$ op2	op1 es menor que op2
$<=$	op1 $<=$ op2	op1 es mayor o igual que op2
$==$	op1 $==$ op2	op1 y op2 son iguales
$!=$	op1 $!=$ op2	op1 y op2 son diferentes

Lenguajes de programación

Operadores - Lógicos

Operadores lógicos

- Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos **(true y/o false)** o los resultados de los operadores relacionales.
- Debe notarse que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser true y el primero es false, ya se sabe que la condición de que ambos sean true no se va a cumplir).
- Esto puede traer resultados no deseados y por eso se han añadido los operadores (**&**) y (|) que garantizan que los dos operandos se evalúan siempre.

Lenguajes de programación

Operadores - Lógicos

Operador	Nombre	Uso	Resultado es true si:
&&	AND	op1 **&&** op2	op1 y op2 son true. Si op1 es false ya no se evalúa op2.
||	OR	op1 || op2	op1 y op2 son true. Si op1 es verdadero ya no se evalúa op2
! =	Negación	! op	op es falso y es falso si op es verdadero ya no se evalúa op2
&	AND	op1 **&** op2	op1 y op2 son true. Siempre se evalúa op2.
|	OR	op1 | op2	op1 u op2 son true. Siempre se evalúa op2.

Lenguajes de programación

Operadores - Concatenación de caracteres

Operador de concatenación de caráteres

- El operador **más (+)** se utiliza también para concatenar cadenas de caracteres.
- Por ejemplo, para escribir una cantidad con un rótulo y unas unidades puede utilizarse la sentencia:
  - System.out.println("El total asciende a " + totalUnidades);
- En el ejemplo anterior, el operador de concatenación se utiliza para construir la cadena de caracteres que se desea imprimir por medio del mótodo **println()**.
- La variable numérica **totalUnidades** es convertida automáticamente por JAVA en cadena de caracteres para poderla concatenar. En otras ocasiones se deberá llamar explícitamente a un método para que realice esta conversión.

Lenguajes de programación

Estructuras de control

Definición

- Facilitan la realización de determinadas acciones, mientras que una condición se cumpla.
- Permiten tomar decisiones de **qué** hacer, en función de las condiciones que se den en el programa en un momento dado de su ejecución.

En JAVA

El lenguaje JAVA soporte cuatro tipos de estructuras de control:

- **Toma de decisión:** if / if-else / switch-case.
- **Bucle o ciclo:** for / while / do-while.
- **Saltos:** break / continue / return / goto.
- **Excepciones.**

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- En JAVA la sentencia **if / if-else** dota a los programas de la habilidad de ejecutar conjuntos de sentencias según la condición.

La sentencia para **if** es:

if (condición) {
    Bloque de sentencias si la condición es **verdadera**.
}

La sentencia para **if-else** es:

if (condición) {
    Bloque de sentencias si la condición es **verdadera**.
}
else {
    Bloque de sentencias si la condición es **falsa**.
}

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Supongamos que un programa debe realizar diferentes acciones dependiendo de si el usuario oprime el botón **aceptar** o el botón **cancelar** en una ventana de diálogo. Nuestro programa puede realizar esta bifurcación usando la sentencia **if-else**:

Bifurcación con la sentencia **if-else**.

```
if (respuesta.equals("Aceptar")) {
    System.out.println("Su petición esta siendo atendida");
}
else {
    System.out.println("Cancelando acción" );
}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Se pueden anidar expresiones **if-else**, para poder implementar aquellos casos con múltiples acciones. Esto es lo que se suele denominar como sentencias **else-if**.

Ejemplo de Bifurcaciones múltiples:

- Supongamos que se desea escribir un programa que clasifique según el contenido de una variable denominada **valor**, asigne una letra a otra variable denominada **clasificacion**.
- A, para un valor entre 100 y 91 (inclusive).
- B, para un valor entre 90 y 81 (inclusive).
- C, para un valor entre 80 y 71 (inclusive).
- F, si no es ninguno de los anteriores.

Lenguajes de programación

Estructuras de control - Sentencias condicionales

Ejemplo de Bifurcaciones múltiples - Solución

```
int valor;
char clasificacion;

if (valor > 90   &&   valor <= 100) {
    clasificacion = 'A';
}
else if (valor > 80   &&   valor <= 90) {
    clasificacion = 'B';
}
else if (valor > 70   &&   valor <= 80) {
    clasificacion = 'C';
}
else {
    clasificacion = 'F';

}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Este sistema de programación (else-if) no es recomendable, por la pérdida de claridad en la lectura del programa. Por ello el lenguaje JAVA incluye la sentencia **switch-case**, para dirigir el flujo de control de variables con múltiples valores.
- La sentencia **switch** permite seleccionar entre varias opciones, según el valor de cierta expresión.
- Cada sentencia case debe ser única y el valor que evalúa debe ser del mismo tipo que el devuelto por la expresión de la sentencia **switch**.
- Las sentencias **break** permiten salir del **switch** y continar con la siguiente instrucción. Las sentencias **break** son necesarias porque sin ellas se ejecutarían secuencialmente las sentencias **case** restantes.

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Finalmente, se puede usar la sentencia **default** para manejar los valores que no son explícitamente contemplados por alguna de las sentencias **case**. Su uso es altamente recomendado.

La forma general de la sentencia **switch** es la siguiente:

```
switch (expresionMultivalor) {
    case valor1:
        bloqueDeSentenciasParaValor1;
        break;
    case valor2:
        bloqueDeSentenciasParaValor2;
        break;
   ...
    default:
        bloqueDeSentenciasParaValorNoDeterminado;
        break;
}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

Ejemplo de sentencia **switch**

- Supongamos un programa con una variable entera **meses** cuyo valor indica el mes actual y se desea imprimir el nombre del mes en que estemos. Se puede utilizar la sentencia **switch** para realizar esta operación.
- Si el valor a comparar no existe o no es un valor válido, el programa debe indicar el error.
- Luego de implementar la solución con la sentencia **switch**, implementarla con la sentencia **else-if** y comparar ambas soluciones.

Lenguajes de programación

Estructuras de control - Sentencias condicionales

Ejemplo de sentencia **switch** - Solución

```
int mes;

switch (mes) {
    case 1:
        System.out.println("Enero");
        break;
    case 2:
        System.out.println("Febrero");
        break;
    //... Otros meses
    default:
        System.out.println("Mes no válido");
        break;
}
```

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles

- En Java se reconocen tres tipos de sentencias de iteración:
  - **while**.
  - **do-while**.
  - **for**.

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **while**.

- Sentencia **while**:
  - El bucle **while** es el bucle básico de iteración.
  - Sirve para realizar una acción sucesivamente mientras se cumpla una determinada condición.

La forma general de la sentencia **while** es la siguiente:

```
while (expresionBooleana) {
    bloqueDeSentencias;
}
```

- Las sentencias se ejecutan mientras la *expresionBooleana* tenga un valor de **verdadero**.

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **while**.

- Multiplicar un número por 2, mientras sea menor que 100.

Ejemplo de sentencia **while**

```
int num = 1;

while (num < 100) {
    num = num * 2;
}
```

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **do-while**.

- Setencia **do-while**.
  - El bucle **do-while** es similar al bucle **while**.
  - En el bucle **while** la expresión se evalúa al principio del ciclo, a diferencia del bucle **do-while** donde la evaluación se realiza al final.
  - La sentencia **do-while** es el constructor de bucles menos utilizado en la programación, pero tiene sus usos, cuando el bucle deba ser ejecutado por lo menos una vez.

La forma general del bucle **do-while** es la siguiente:

```
do {
    bloqueDeSentencias;
} while (expresionBooleana);
```

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **do-while**.

Ejemplo bucle **do-while**

- Se desea leer cierta información de entrada.
- Si la información de entrada es la acción 'SALIR', imprimir por pantalla que se ha salido del programa.
- Luego de implementar el ejemplo con el bucle **do-while**, impleméntelo utilizando la sentencia **while**.

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **do-while**.

Ejemplo bucle **do-while** - Solución

```
do {
    System.out.print("Ingrese opción: ");
    br = new BufferedReader(new InputStreamReader(System.in));
} while (!br.readLine().equals("SALIR"));

System.out.println("Ha salido del programa.");
```

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **for**.

- Setencia **for**.
  - La sentencia **for** se resume un bucle **do-while** con una iniciación previa.
  - Es muy común que en los bucles **while** y **do-while** se inicien las variables de control de número de pasadas por el bucle, inmediatamente antes de comenzar los bucles.

La forma general del bucle **for** es la siguiente:

```
for ( iniciación ; condición de término ; incremento ) {
    bloqueDeSentencias;
}
```

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **for**.

- La **iniciación** es una sentencia que se ejecuta una vez antes de entrar en el bucle.
- El **criterio de término** es una expresión que determina cuándo se debe terminar el bucle.
  - Esta expresión se evalúa al final de cada iteración del bucle.
  - Cuando la expresión se evalúa en falso, el bucle termina.
- El **incremento** es una expresión invocada en cada iteración del bucle.
  - En estricto rigor puede ser una acción cualquiera, aunque se suele utilizar para incrementar una variable contador (no indice).
- Algunos (o todos) estos componentes pueden omitirse, pero los puntos y coma (;) siempre deben estar (aunque sea sin nada entre sí).

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **for**.

- Se debe utilizar el bucle **for** cuando se conozcan las restricciones (su instrucción de iniciación, criterio de término e instrucción de incremento).

Ejemplo bucle **for**

- Se desea llenar un arreglo con los 100 primeros valores pares enteros.
- Luego de implementar el ejemplo con el bucle **for**, impleméntelo utilizando la sentencia **while**.

Lenguajes de programación

Estructuras de control - Sentencias de iteración o bucles **for**.

Ejemplo bucle **for** - Solución

```
int[] array = new int[100];

for ( int count = 0 ; count < array.length ; count++ ) {
    array[count] = (count *= 2);
}
```

Lenguajes de programación

Estructuras de control - Sentencias de salto.

- En Java se reconocen, básicamente, cuatro tipos de sentencias de salto:
  - **break**.
  - **continue**.
  - **return**.
  - **goto** - A pesar de ser una palabra reservada, no es soportada por el lenguaje JAVA.

Lenguajes de programación

Estructuras de control - Sentencias de salto **break**.

- La sentencia **break** provoca que el flujo de control salte a la sentencia inmediatamente posterior al bloque en curso (ya visto dentro de la sentencia switch).
- El uso de la sentencia **break** con sentencias etiquetadas es una alternativa al uso de la sentencia **goto**.
- Se puede etiquetar una sentencia con una identificador JAVA válido, seguido por dos puntos (:) antes de la sentencia.
  - nombreSentencia: sentenciaEtiquetada
- La sentencia **break** se utiliza para salir de una sentencia etiquetada, llevando el flujo del programa al final de la sentencia de programa que indique:
  - **break** nombreSentencia;

Lenguajes de programación

Estructuras de control - Sentencias de salto **continue**.

- Del mismo modo que en un bucle se puede desear romper la iteración, también se puede desear continuar con él, pero dejando pasar una determinada iteración.
- Se puede usar la sentencia **continue** dentro de los bucles para saltar a otra sentencia, pero **no** puede ser llamada fuera de un bucle.
- Tras la invocación a una sentencia **continue** se transfiere el control a la condición de término del bucle que vuelve a ser evaluada en ese momento.
- El bucle continúa o no, dependiendo del resultado de la evaluación.

Lenguajes de programación

Estructuras de control - Sentencias de salto **continue**.

- En los bucles **for**, en ese momento, se ejecuta la cláusula de incremento (antes de la evaluación).

Ejemplo de salto de instrucción **continue**

- Imprimir en pantalla los 100 primeros números que no son divisibles por 3.
- Luego de implementar el ejemplo con el bucle **for** y la sentencia **continue**, impleméntelo utilizando la sentencia **while**.

Lenguajes de programación

Estructuras de control - Sentencias de salto **continue**.

Ejemplo de salto de instrucción **continue** - Solución

```
for ( int count = 0 ; count < 100 ; count++ ) {
    if (count % 3 == 0) {
        continue;
    }

    System.out.println("count = " + count);
}
```

Lenguajes de programación

Estructuras de control - Sentencias de salto **return**.

- Es factible de utilizar para salir del método en curso y retornar (en general un valor), a la sentencia dentro de la cual se realizó la llamada.
- Para devolver un valor, simplemente se debe poner el valor (o una expresión que calcule el valor) a continuación de la palabra **return**.
- El valor devuelto por **return** debe coincidir con el tipo de dato declarado como valor de retorno del método.
- Cuando un método se declara como **void** se debe usar la forma de **return** (como buena práctica), sin indicarle ningún valor.

Lenguajes de programación

Estructura de clases en JAVA

```
public class Auto {
        ...
        variables
        ...
        métodos
        ...
}
```

Lenguajes de programación

Estructura de clases en JAVA - Primer programa

```
public class HelloWorld {

        public static void main(String[ ] args) {
                System.out.println(" Hello, World! ");
        }

}
```

Lenguajes de programación

Estructura de clases en JAVA - Definición de variables

```
public class Auto {

        private String color;
        String marca;
        boolean estado;
        ...
}
```

Si bien, la declaración de variables puede ir en cualquier parte, habitualmente se declaran al inicio método o clase.

Lenguajes de programación

Estructura de clases en JAVA - Definición de variables

Es posible encadenar nombres de variables del mismo tipo

```
public class Auto {

        private String color, marca;
        boolean estado;
        int x, y, z;
        ...
}
```

Lenguajes de programación

Estructura de clases en JAVA - Declaración de variables

```
public class Auto {

        private String color = "rojo", marca = "chevrolet";
        boolean estado = false;
        int x = 1, y = 0, z = -1;
        ...
}
```

Lenguajes de programación

Estructura de clases en JAVA - Comentarios

```
// Comentario en una línea
/* Comentario en más
de una línea */
/** Comentario para
javadoc */
public class Auto {
        ...
}
```

Lenguajes de programación

Estructura de clases en JAVA - Método

Método: encenderMotor

```
public class Auto {

        private String color;
        private String marca;
        private boolean estado;

        public void encenderMotor() {
                if (estado == true) {
                        System.out.println("Auto encendido");
                } else {
                        estado = true;
                        System.out.println("OK, se encendio ");
                }
        }
}
```

Lenguajes de programación

Estructura de clases en JAVA - Método

Método: mostrarInfo

```
public class Auto {

        public String color;
        public String marca;
        private boolean estado;

        public void encenderMotor() { ... }

        public void mostrarInfo() {
                System.out.println("Este auto es un " + marca + " de color " + color);
        }
}
```

Lenguajes de programación

Estructura de clases en JAVA - Método

Método: main

```
public class Auto {

        private String color;
        private String marca;
        private boolean estado;

        public void encenderMotor() { ... }

        public void mostrarInfo() { ... }

        public static void main(String[ ] args) {
                Auto a = new Auto ( );
                a.marca = "Chevrolet Sail";
                a.color = "Gris";
                System.out.println("Llamando a showAtr() ");
                a.mostrarInfo();
                System.out.println("Llamando a encenderMotor() ");
                a.encenderMotor();
        }


}
```

Lenguajes de programación

Ejercicios

Método: Mostrar atributos

Desarrollo un programa en JAVA que permita saber los últimos 20 años biciestos (iniciando desde el año actual). Desarrollo el programa basado en el paradigma de la orientación a objeto.

Preguntas

Preguntas ?