Programación II

Carla TARAMASCO

Profesora e Investigadora

DECOM & CNRS

mail : alumnos_uv@yahoo.cl

13 mars 2013

Plan de la sección Paradigmas de Programación

- Definición de paradigma
- Principales Paradigmas
  - Imperativo
  - Declarativo
  - Estructurado
  - Orientado a Objetos
  - Funcional
  - Logico

Bibliografia

- Aprendiendo Java en 21 dias. Laura Lemay and Charles Perkins. Edición : Prentice Hall Hispanoamericana SA, 1996
- Java como programar. P Deital and H Deitel. Edición : Pearson Prentice Hall, 2008
- Java como programar. P Deital and H Deitel. Edición : Pearson Prentice Hall, 2004
- Aprenda Java como si estuviera en primero. Javier Garcia et all. Universidad de Navarra, 2000
- Guia de iniciación al lenguaje Java, Universidad de Burgos, 1999
- Aprendiendo Java y programación a objetos. Gustavo Perez, 2008
- Apuntes de programación. Universidad de Cergy Pointoise, 2010
- Curso Java de Everis, 2006
- Programación con Java. Profesor : Carlos Alberto Román Zamitiz. Universidad Nacional Autonoma de Mexico : http ://profesores.fi-b.unam.mx/carlos/java/indice.html
- Documentación JAVA : http ://www.oracle.com/technetwork/java/javase/documentation/index.html

Paradigmas de Programación

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas, que permite producir programas con directrices específicas, tales como : estructura modular, fuerte cohesión, alta rentabilidad, etc.

Plan de la sección Paradigmas de Programación

Paradigma Imperativo

Describe la programación como una secuencia instrucciones o comandos que cambian el estado de un programa. El código máquina en general está basado en el paradigma imperativo. Su contrario es el paradigma declarativo. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea. Los primeros lenguajes imperativos fueron los lenguajes de máquina.

Paradigmas de Programación

Paradigma Declarativo

No se basa en el cómo se hace algo (cómo se logra un objetivo paso a paso), sino que describe (declara) cómo es algo.
Se enfoca 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. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora que es lo que se desea obtener o que es lo que se está buscando).

Paradigmas de Programación

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.

Paradigmas de Programación

Paradigma Orientado a Objetos

Está basado en el uso de objetos : estructuras de datos que contienen atributos y métodos con sus interacciones. La idea es encapsular estados o propiedades y operaciones o comportamientos en objetos que se comunican entre si. Su principal ventaja es la reutilización de códigos y su facilidad para pensar soluciones a determinados problemas.

Paradigmas de Programación

Paradigma Funcional

Este paradigma concibe a la computación como la evaluación de funciones y evita declarar y cambiar datos. En otras palabras, hace hincapié en la aplicación de las funciones y composición entre ellas, en contraste con el estilo de programación imperativa, más que en los cambios de estados y la ejecución secuencial de comandos.

Paradigmas de Programación

Paradigma 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 los problemas.

Otros Paradigmas y subparadigmas

Paradigma orientado al sujeto, paradigma reflectante, programación basada en reglas, paradigma basado en restricciones, programación basada en prototipos, etc.

Paradigmas de Programación

Paradigma 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 los problemas.

Otros Paradigmas y subparadigmas

Paradigma orientado al sujeto, paradigma reflectante, programación basada en reglas, paradigma basado en restricciones, programación basada en prototipos, etc.

Plan de la Introducción a la POO y a Java

- Introducción a la Programación Orientada a Objetos.
  - Objetos y clases
  - Comportamiento y atributos
  - Características asociadas a la POO.
    - Abstracción
    - Encapsulamiento
    - Ocultamiento
    - Herencia
    - Polimorfismo
- Introducción a la Programación en Java
  - Inicios de Java
  - Ventajas de Java
  - Programar en Java
- Ejercicios de Introducción a Java

Definiciones

POO

La **POO** se basa en la dividir el programa en pequeñas unidades lógicas de código. A estas pequeñas unidades lógicas de código se les llama objetos. Los objetos son unidades independientes que se comunican entre ellos.
La **POO** es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.

Definiciones

Que es un objeto ?

Cualquier cosa que vemos a nuestro alrededor. Ej : auto.
Componentes de los objetos :

- características (marca, modelo, color, etc)
- comportamiento (frenar, acelerar, retroceder, encender, etc).

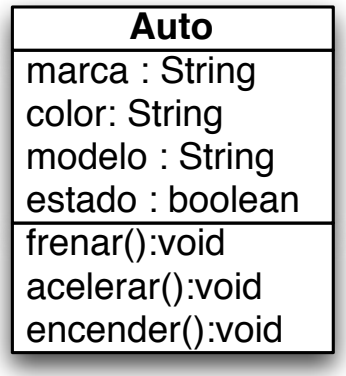
Definiciones

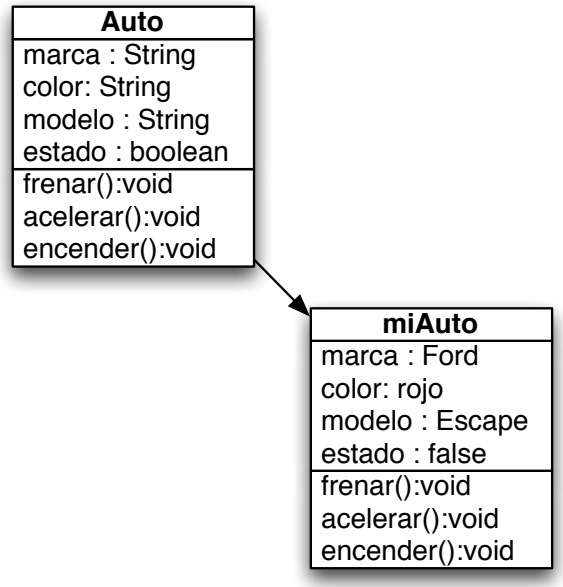
Objetos y clases

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

Clase y Objeto

Clase





Definiciones

Objetos y clases

- **Clases :** Modelo para múltiples objetos con características 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 seria un tipo de dato (en C seria 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 diversas características mientras se comporte y reconozca como objeto de la clase. En programación estructurada seria una variable
- *Ejemplo* : Clase Button.

Definiciones

Atributos y Comportamiento

Atributos : Características que diferencian a un objeto de otro y determinan la apariencia, estado u otras cualidades de ese objeto. Los atributos se definen como variables de hecho podrían ser las variables globales del objeto completo. Cada instancia de una clase puede tener diferentes valores para sus variables, a cada variable se le llama una variable de instancia. La clase define el tipo de atributo y cada instancia guarda su propio valor para ese atributo.

Definiciones

Atributos y Comportamiento

Los **métodos** son funciones definidas dentro de una clase que operan en las instancias de esas clases. Los objetos se comunican entre si mediante el uso de métodos, 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.

Características asociadas a la POO

Abstracción

La abstracción consiste en captar las características esenciales y el comportamiento de un objeto.
Ej : ¿Qué características podemos abstraer de los autos ?¿Qué características semejantes tienen todos los autos ?. Todos tendrán una marca, un modelo, puertas, ventanas, etc. Y en cuanto a su comportamiento todos los automóviles podrán acelerar, frenar, retroceder, etc.
En POO el concepto de clase es la representación y el mecanismo por el cual se gestionan las abstracciones.
En Java las clase se definen :

```
public class Auto{
        // variables
        // metodos
}
```

Características asociadas a la POO

Encapsulamiento

El encapsulamiento consiste en unir en la clase las características y comportamientos, esto es, las variables y métodos. Es tener todo esto es una sola entidad. En los lenguajes estructurados esto era imposible.
La utilidad del encapsulamiento va por la facilidad para manejar la complejidad, ya que tendremos clases como cajas negras donde sólo se conoce el comportamiento pero no los detalles internos.

Características asociadas a la POO

Ocultamiento

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 a otros objetos que especifica cómo pueden interactuar con los objetos de la clase.

- Restringir el uso de la clase porque habrá cierto comportamiento privado de la clase que no podrá ser accedido por otras clases.
- Controlar el uso de la clase porque se darán ciertos mecanismos para modificar el estado de una clase

En Java el ocultamiento se logra usando las palabras reservadas : *public, private* y *protected* delante de las variables y métodos.

Características asociadas a la POO

Herencia

Organización jerárquica de las clases.
Cada clase tiene una superclase y puede tener una o mas subclases. Las subclases heredan todos los métodos y variables de las superclases, esto significa que las subclases, aparte de los atributos y métodos propios, tienen incorporados los atributos y métodos heredados de la superclases. De esta manera se crea una jerarquía de clases.
En Java la herencia se logra usando la palabra reservada : *extends*. En la parte superior de la jerarquía de clase Java esta la clase *Object* todas las clases heredan de esta superclase, cada clase hacia abajo agrega mas información.

Características asociadas a la POO

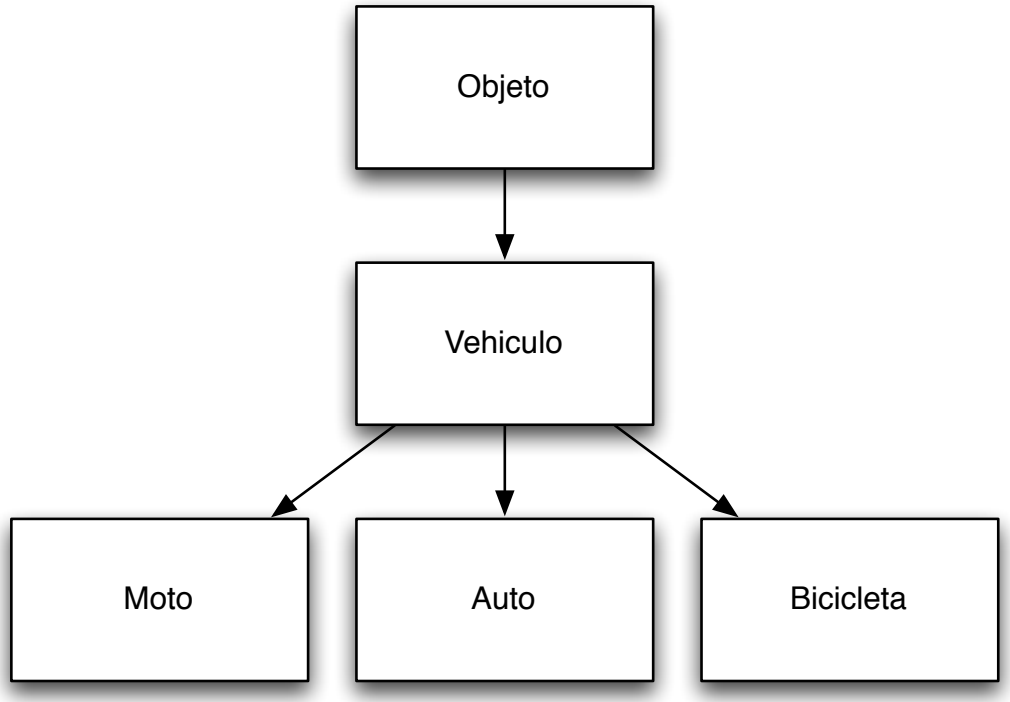
Crear una jerarquía entre clases

Subclasificación : Creación de una nueva clase que heredara de otra clase en la jerarquía. Al usar subclasificación, solo se necesita definir las diferencias entre la clase creada y las super clases de esta. Si se crea una clase sin indicar la super clase en la primera linea, java automáticamente supone que esta heredando de la clase *Object*

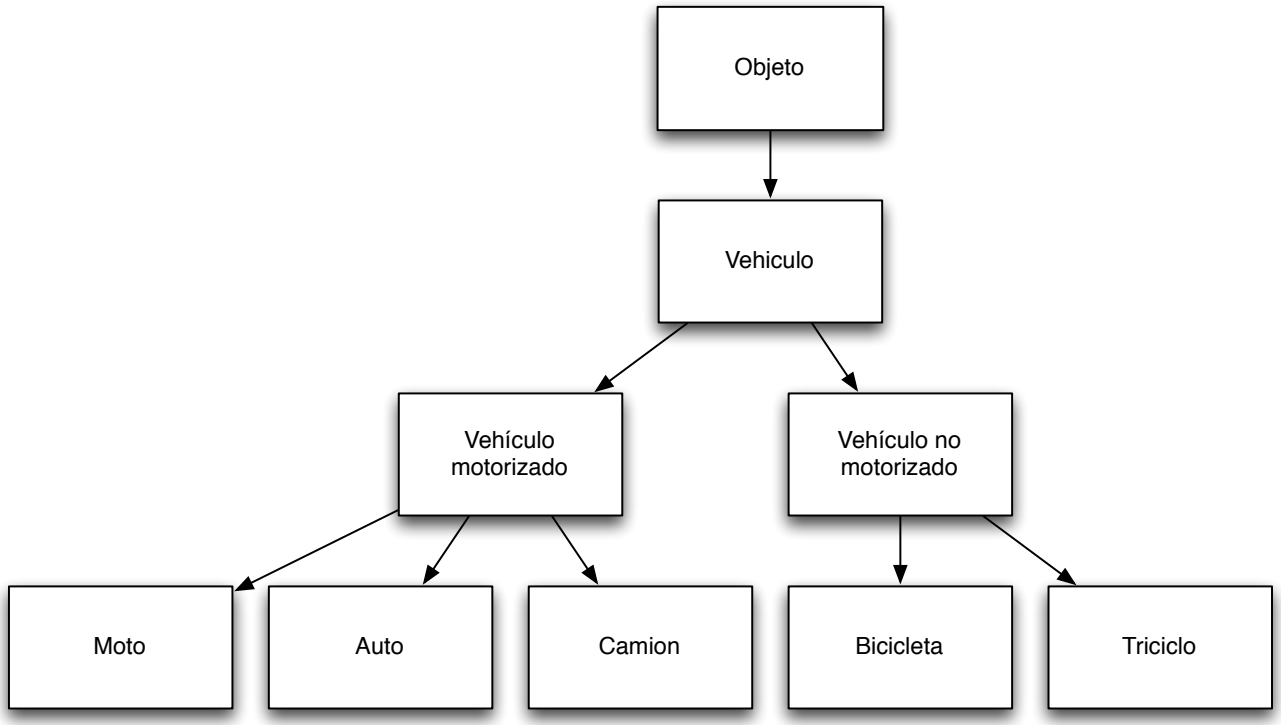
Jerarquía de herencia : se deben desarrollar las clases en una jerarquía en la cual se separe la información común para múltiples clases de la información particular.

Ejemplo : Crear una jerarquía para una clase Auto, una clase Moto y una clase Bicicleta.

Jerarquia de herencia



Jerarquia


Objeto
Vehiculo
Vehículo motorizado
Vehículo no motorizado
Moto
Auto
Camion
Bicicleta
Triciclo

Características asociadas a la POO

Polimorfismo

Literalmente significa ”cualidad de tener mas de una forma”. En 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.

Características asociadas a la POO

Envio 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 de B), el objeto A manda un mensaje al objeto B. Los mensajes son invocaciones a los métodos de los objetos. Por 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).

Estas tres partes del mensaje (objeto destinatario, método y parámetros) son suficiente información para que el objeto que recibe el mensaje ejecute el método.

Inicios de Java

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

Ventajas de Java : independencia de plataformas

- **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.
    bytecodes : conjunto de instrucciones más abstracto que el código máquina (código intermedio), pero no son especificas a un procesador.

Ventajas de Java : Orientado a Objetos

- 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 traves de la herencia y polimorfismo
- Agiliza el desarrollo de software.
- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software.
- Recolección de basura

Ventajas de Java : Simplicidad

- En Java no hay punteros
- Las cadenas y los arreglos son objetos reales
- La administración de la memoria es automatica

Programar en Java

El ambiente de desarrollo Java

- Un compilador : genera los ficheros compilados en bytecode (en vez de generar código de maquina) (extensión ***.class**) a partir del código fuente (extensión ***.java**). El bytecode son instrucciones independientes de la plataforma.
- Un interprete Java denominado *Java Virtual Machine (JVM)* : que interpreta el bytecode (código neutro) convirtiendo a código particular de la CPU utilizada, lo que permite ejecutar el programa. JVM es una aplicación que simula una computadora pero oculta el sistema operativo y el hardware subyacente. JVM es un programa nativo ejecutable en una plataforma especifica.
- Java tiene la característica de ser al mismo tiempo compilado e interpretado.

Compilación y Ejecución

Figura extraída de :
http : //profesores.fib.unam.mx/carlos/java/java_basico1_1.html

Programar en Java

Kit de desarrollo Java (JDK) 1-2

Contiene las herramientas y 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 codigo fuente (con extension .java). Este comando no es parte de Java Runtine Environment (JRE) dado que JRE esta destinado unicamente 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*

Programar en Java

Kit de desarrollo Java (JDK) 2-2

3. **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. Es decir, comprimir el proyecto en un solo archivo de tipo JAR. Este comando es parte solo de JDK.Ex : para crear un JAR *jar cf Auto.jar Auto**, para extraer un JAR *jar xf jar-file*
4. **javadoc** : Crea documentación en formato HTML a partir de el código fuente y los comentarios.Ej : *javadoc Auto.java*
5. **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 *javadoc Auto.java* y luego *jdb Auto*

Programar en Java

Fases para el desarrollo de un programa

1. **Creación :** Usar un editor para escribir el código fuente. Guardarlo con extensión **.java**. Se pueden también usar 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**
4. **Verificación del bytecode :** Verifica que los bytecode sean validos 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 *c*ompilación justo a tiempo (JIT). La JVM analiza el código buscando las partes que se ejecutan con frecuencia, para traducirlas al lenguaje de maquina del computador, así cuando encuentra nuevamente el código lo ejecuta en lenguaje de maquina que es mas 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 maquina. Ej ejecución : *java HelloWorld*

Plan de la sección Fundamentos de Java

- Variables y tipos de datos
- Definiciones basicas
- Comentarios
- Literales
- Expresiones y Operadores
- Ejercicios Fundamentos

Mi primer programa

```
public class Hello{
        public static void main(String[]args){
                System.out.println("Hola mundo");
        }
}
```

Variables y tipos de datos

Identificadores

Los nombres de las clases, métodos y variables deben :

- Empezar con una letra, subrayado (_) o dólar ($), no puede tener espacios ni comenzar por números. Ex : *_var1*, *varx*, *MAX_NUM*, *$var*
- Después del primer carácter pueden usarse números.
- No puede coincidir con las palabras claves o reservadas.
- No hay un límite en el número de caracteres que pueden tener los identificadores.

Variables y tipos de datos

Palabras Claves de Java

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

* Son palabras claves de Java que no son usadas actualmente.
** Palabra clave agregada en Java 2

true, false, null son palabras reservadas, por lo cual, tampoco pueden usarse como identificadores.

Tabla de : http : //profesores.fi − b.unam.mx/carlos/java/java_basico2_3.html

Variables y tipos de datos

Las variables deben tener :

- Un tipo : que describa el tipo de dato que contiene
- Un identificador : para referir al dato que contiene
- Un valor

Variables y tipos de datos

Tipos de Variables

Java tiene 3 tipos de variables :

- Variables de instancia : se usan para definir atributos o estados de un objeto en particular.
- Variables de clase : similares a las variables de instancia, la unica diferencia es que sus valores se aplican a todas las instancias de la clase.
- Variables locales : se utilizan y se declaran dentro de las definiciones de metodo o bloques, Ej : contadores de indice de ciclos, temporales, etc. Al terminar la ejecución del bloque o metodo la variable deja de existir.

Java no tiene variables globales.

Clases de Variables

Declaración de Variables

Para usar una variable primero hay que declararla con el tipo y el identificador (o nombre) de la variable.

```
String color = "rojo";
String marca = "Toyota";
boolean estado = false;
ruedas = 4;
```

- *String :* es una clase que esta dentro de las bibliotecas de clases de Java.
- *boolean :* es un tipo de dato real que puede ser true o false, los booleanos no son números en Java.

Definiciones básicas Java

Definir una clase

```
public class Auto{
        // variables
        // metodos
}
```

public : la clase es accesible desde métodos de cualquier clase. Atributo que permite acceder a la clase desde clases e interfaces que estén en otros paquetes.

Definiciones básicas Java

Definir 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 metodo o clase.

Definiciones básicas Java

Definir variables

Es posible **encadenar** nombres de variables del mismo tipo

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

Definiciones básicas Java

Definir variables

También se le puede dar un **valor inicial** a las variables

```
String color = "rojo";
String marca = "Toyota";
boolean estado = false;
int x =5,y=7;
```

Sintaxis básicas de Java

- **Comentario :** Hay tres tipos de comentarios en Java
  - // comentario de una linea
  - /*comentario de mas de una linea */
  - /**comentarios para javadoc */
- **Sentencia :** Es una linea de código terminada con punto y coma ” ;”.
- **Bloque :** Es un conjunto de sentencias agrupadas entre llaves ”{}”. Los bloques pueden ser anidados

Definiciones básicas Java

Definir un método : Arrancar el auto

```
public class Auto{

        String color;
        String marca;
        boolean estado;

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

Rappels !

- Los archivos deben llamarse igual a la clase definida public con extensión .java
- La indentación no es importante para el compilador Java. Sin embargo la indentación facilita la lectura y comprensión.
- Java es sensible a las mayúsculas y minúsculas por lo cual, a1 es distinto de A1.

Rappels !

Por Convención los nombre de :

- Las clases comienzan con una letra mayúscula : *ClasePrueba*
- Los métodos comienzan con minúscula : *abrirPuerta*
- Las variables empiezan por una letra minúscula : *nombreDato*
- Si el nombres de la clase, metodo o varible esta compuesto por mas de una palabra, las palabras se ponen juntas y la inicial de la segunda, tercera....palabra se escribe con mayúsculas
- Las constantes se escriben en mayúsculas. Si tiene varias palabras se separan con "_" : *MUN_MAX*

Definiciones básicas Java

Definir un nuevo método : Mostrar atributos

```
public class Auto{

        String color;
        String marca;
        boolean estado;

        void startMotor(){
                if (estado== true)
                        System.out.println("Auto encendido");
                else {
                        estado=true;
                        System.out.println("Ok se encendio");
                }
        }
        void showAtr(){
                System.out.println("Este auto es un" +
                                        marca+ "de color"+color);
        }
}
```

Definiciones básicas Java

Para hacer algo con la clase *Auto* se deberá crear una aplicación que la utilice o agregarle un método *main*. Todas las aplicaciones deben tener el método main()

```
public static void main (String args[]){
        Auto a = new Auto();
        a.marca = "Toyota Yaris";
        a.color="rojo";
        System.out.println("llamando a showAtri");
        a.showAtri();
        System.out.println("llamando a startMotor");
        a.startMotor();
}
```

static : atributo de la clase no del objeto. Almacena el mismo valor para todos los objetos de la clase. No es necesario crear un objeto para acceder al atributo. No se puede llamar a métodos no estáticos desde la misma clase (se necesita un objeto)

Salida del programa

```
MacBook-Pro-de-Carla-Taramasco-T:Desktop taramasco$ java Auto
llamando a showAtri
Este auto es un Toyota Yaris de color rojo
llamando a startMotor
Ok se encendio
MacBook-Pro-de-Carla-Taramasco-T:Desktop taramasco$
```

Explicación del código

- La linea : *Auto a = new Auto() ;* crea una nueva instancia de la clase Auto y guarda una referencia de ella en a. Por lo general, no se opera directamente con las clases sino que se crean objetos y luego se llama a los metodos en esos objetos.
- Las lineas : *a.marca = "Toyota Yaris" ;* y *a.color="rojo" ;* son las variables de instancia para el objeto autot la crea una nueva instancia de la clase Autot y guarda una referencia de ella en a.
- Las lineas : *a.showAtr() ;* y *a.startMotor() ;* llama a los metodos *showAtri* y *statrMotot* desde los objetos.

Explicación del código

- La linea : *Auto a = new Auto() ;* crea una nueva instancia de la clase Auto y guarda una referencia de ella en a. Por lo general, no se opera directamente con las clases sino que se crean objetos y luego se llama a los metodos en esos objetos.
- Las lineas : *a.marca = "Toyota Yaris" ;* y *a.color="rojo" ;* son las variables de instancia para el objeto autot la crea una nueva instancia de la clase Autot y guarda una referencia de ella en a.
- Las lineas : *a.showAtr() ;* y *a.startMotor() ;* llama a los metodos *showAtri* y *statrMotot* desde los objetos.

Tipos de variables

Tipos enteros

- byte : 8 bits : $-128a127$
- short : 8 bits : $-32,768a32,767$
- int : 32 bits : $-2,147,483,648a2,147,483,647$ (en valores decimales : float)
- long : 64 bits : $-9223372036854775808a9223372036854775808$ (en valores decimales : double)

Ademas de los tipo enteros y decimales, Java tiene *char* para caracteres individuales (16 bits) y boolean para true y false.

Variables

Asignación de variables

```
String color = "rojo";
String marca = "Toyota";
boolean estado = false;
ruedas = 4;
```

Expresiones y operadores

Variables

Asignación de variables

```
String color = "rojo";
String marca = "Toyota";
boolean estado = false;
ruedas = 4;
```

Expresiones y operadores

- Expresiones : enunciados que regresan un valor.
- Operadores : símbolos que se utilizan en expresiones.

Operadores

Aritméticos

- $+$: Suma
- $-$: Resta
- $*$: Multiplicación
- $/$: División
- % : Modulo

Preguntas y ejercicios del curso

- Java es un lenguaje compilado o interpretado ?
- Explique las etapas de producción de un programa, precisando los comandos necesarios
- Escriba un programa que utilice 3 variables x,y,z de tipo int y 3 variables a,b,c de tipo float. Efectue los siguientes cálculos :

  1. $z = x + y$
  2. $z = x - y$
  3. $z = x * y$
  4. $z = x/y$
  5. $c = a - b$
  6. $c = a/b$

- Prueba el programa con diferentes valores de a,b,d,e.
- Prueba con valores $a < b$, que observa en la division ?

Variables

Asignación de variables

```
public class IncreTest{
        public static void main (String args[]){

        short x=6;
        int y=4;
        float a=12.5f;
        float b=7.0f;

        System.out.println("x es : "+x+", y es : " +y);
        System.out.println("x+y = " +(x+y));
        System.out.println("x-y = " +(x-y));
        System.out.println("x*y = " +(x*y));
        System.out.println("x/y = " +(x/y));
        System.out.println("x%y = " +(x%y));
        System.out.println("a es : "+a+" y b es : " +b);
        System.out.println("a/b = " +(a/b));
        }}
```

+ *concatena la cadena de caracteres en el System.out.println*

Operadores

Asignación

- $x+ = y : x = x + y$
- $x- = y : x = x - y$
- $x* = y : x = x * y$
- $x/ = y : x = x/y$
- $x\% : x = x\%y$

Incrementos y decrementos

Asignación

- $y = x++$
- $y = ++x$

Cual es la diferencia ?

```
public class IncreTest{
        public static void main (String args[]){

        int x=0;
        int y=0;

        System.out.println("x e y son : "+x+" y : " +y +"respectivamente");
        x++;
        System.out.println("x++ resulta = "+x);
        ++x;
        System.out.println("++x resulta "+ x);
        System.out.println("Volvemos a x=0 ");
        x=0;
        y=x++;
        System.out.println("y=x++ ");
        System.out.println("x es : "+x);
        System.out.println("y es : "+y);
        y=++x;
        System.out.println("y=++x ");
        System.out.println("x es : "+x);
        System.out.println("y es : "+y) ;
        }}
```

Operadores

Comparación

- $==$: igual
- $!=$: diferente
- $<$: menor que
- $<=$: menor o igual que
- $>$: mayor que
- $>=$: mayor o igual que

Operadores

Lógicos

- $|$ o $||$: OR. Usando solo un operador $|$ se evaluarán siempre ambas expresiones, en cambio, usando los dos $||$ si el lado izquierdo de la expresion es verdadero la expresion será verdadera, nunca evaluará el lado derecho.
- $\&$ o $\&\&$: AND. Usando solo un operador $\&$ se evaluarán siempre ambas expresiones, en cambio, usando los dos $\&\&$ si el lado izquierdo de la expresion es falso la expresion será falsa, nunca evaluará el lado derecho.
- $\wedge$: XOR
- ! : NOT