Centro Asociado Palma de Mallorca

Introducción Práctica de Programación Java

Antonio Rivero Cuesta

Sesión

I

Introducción

Java es un lenguaje desarrollado por Sun Microsystems en 1991.

Los padres de Java son James Gosling y Bill Joy.

Java desciende de un lenguaje llamado Oak cuyo propósito era la creación de software para la televisión interactiva.

La principal característica de Java es la de ser un lenguaje compilado e interpretado.

Todo programa en Java ha de compilarse y el código que se genera ***bytecodes*** es interpretado por una máquina virtual.

De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma.

Java es un lenguaje orientado a objetos de propósito general.

Aunque Java comenzará a ser conocido como un lenguaje de programación de applets que se ejecutan

en el entorno de un navegador web, se puede utilizar para construir cualquier tipo de proyecto.

Su sintaxis es muy parecida a la de C y C++ pero hasta ahí llega el parecido.

Java no es una evolución ni de C++ ni un C++ mejorado.

Características

- Simple.
- Orientado a objetos.
- Distribuido.
- Interpretado.
- Robusto.
- Seguro.
- Independiente de la plataforma.
- Multihilo nativo.

Simple

Sintaxis simple.

Muchos programadores lo utilizan.

Se eliminan problemas de:

- Aritmética de punteros.
- Recolección de basura.

Orientado a objetos

El ***paradigma de programación*** se basa en modelar el sistema como ***clases*** e ***instancias de las clases***.

Es una evolución del modelo imperativo.

Es un paradigma que se acerca al mundo real y nos permite aumentar la comprensibilidad de los problemas.

Existen una serie de principios fundamentales para comprender cómo se crea un programa con el paradigma de la orientación a objetos.

Estos principios son:

- La abstracción.
- El encapsulamiento.
- La modularidad.
- La jerarquía.
- El paso de mensajes.
- El polimorfismo.

Distribuido

La computación distribuida implica que varias computadoras trabajan juntas en la red.

Java ha sido diseñado para facilitar la construcción de aplicaciones distribuidas mediante una colección de clases para uso en aplicaciones en red.

La capacidad de red está incorporada a Java.

Interpretado

Java es ***interpretado*** y se necesita un *intérprete* para ejecutar programas Java.

Los programas se compilan en una Máquina Virtual Java generándose un código intermedio denominado *bytecode.*

El *bytecode* es independiente de la máquina y se puede ejecutar en cualquier máquina que tenga un intérprete Java.

Robusto

Ningún lenguaje puede asegurar fiabilidad completa.

Java se ha escrito pensando en la verificación de posibles errores y por ello como un lenguaje fuertemente tipificado.

Java ha eliminado ciertos tipos de construcciones de programación presentes en otros lenguajes que son propensas a errores.

No soporta, por ejemplo, punteros (apuntadores) y tiene una característica de manejo de excepciones en tiempo de ejecución para proporcionar robustez en la programación.

Utiliza recolección de basura en tiempo de ejecución en vez de liberación explícita de memoria.

En lenguajes como C++ es necesario borrar o liberar memoria una vez que el programa ha terminado.

Seguro

Seguridad a nivel de código

- Se comprueba que el código es coherente antes de ejecutarse.

Seguridad en ejecución

- Gestores de seguridad limitan acceso a recursos.

Independiente de la plataforma

Java sigue la política WORE (Write Once Run Everywhere)

- Escribe (compila) el código una vez y podrás ejecutarlo donde sea
- Multiplataforma mediante el uso de un código intermedio (bytecodes)
- Java Virtual Machine
- Interpreta el código bytecode

Multihilo nativo

Es la capacidad de un programa de ejecutar varias tareas simultáneamente.

La Programación multihilo está integrada en Java.

En otros lenguajes se tiene que llamar a procedimientos específicos de sistemas operativos para permitir multihilo.

Los hilos sincronizados son muy Útiles en la creación de aplicaciones distribuidas y en red.

Java Development Kit

La herramienta básica para empezar a desarrollar aplicaciones o *applets* en Java es el JDK (*Java Developer's Kit*) o Kit de Desarrollo Java, que consiste, básicamente, en un compilador y un intérprete (JVM) para la línea de comandos.

No dispone de un ***entorno de desarrollo integrado*** (IDE), pero es suficiente para aprender el lenguaje y desarrollar pequeñas aplicaciones.

El Kit de desarrollo puede obtenerse:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnes

Java SE Development Kit 8

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ **Accept License Agreement** ◉ **Decline License Agreement**

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	83.51 MB	jdk-8-linux-arm-vfp-hflt.tar.gz
Linux x86	133.57 MB	jdk-8-linux-i586.rpm
Linux x86	152.47 MB	jdk-8-linux-i586.tar.gz
Linux x64	133.85 MB	jdk-8-linux-x64.rpm
Linux x64	151.61 MB	jdk-8-linux-x64.tar.gz
Mac OS X x64	207.72 MB	jdk-8-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.5 MB	jdk-8-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	95.53 MB	jdk-8-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.78 MB	jdk-8-solaris-x64.tar.Z
Solaris x64	93.15 MB	jdk-8-solaris-x64.tar.gz
Windows x86	151.68 MB	jdk-8-windows-i586.exe
Windows x64	155.14 MB	jdk-8-windows-x64.exe

El kit contiene básicamente:

- El compilador: **javac.exe**
- El depurador: **jdb.exe**
- El intérprete: **java.exe** y **javaw.exe**
- El visualizador de applets: **appletviewer.exe**
- El generador de documentación: **javadoc.exe**
- Un desensamblador de clases: **javap.exe**

El JDK no incluye IDE.

Entorno de desarrollo integrado:

- Bloc de notas.
- BlueJ
- JCreator
- Eclipse
- NetBeans

El ***entorno de desarrollo integrado*** para este curso será el Blue J.

Se puede descargar de: http://www.bluej.org/

Bibliografía

Programación orientada a objetos con Java. Una introducción práctica usando BlueJ.

David J. Barnes, Michael Kölling.

Editorial Pearson

Programación en *Java* 2 Serie *Schaum*.

Jesús Sánchez Allende y otros.

Clases y Objetos

En ***Programación Orientada a Objetos*** hay que distinguir entre dos conceptos íntimamente ligados, la ***clase*** y el ***objeto***.

Una ***clase*** es una plantilla donde vamos a definir unos ***atributos*** y unos ***métodos***.

Una clase es la ***implementación de un tipo de objeto***, considerando los objetos como ***instancias*** de las clases.

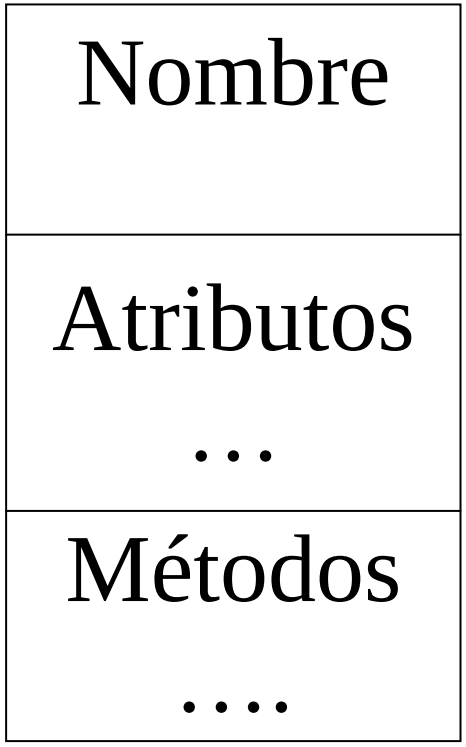
Cuando se crea un objeto, ***se instancia una clase,*** mediante el operador **new.**

```
Figura triangulo = new Figura();
```

Se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto.

Cuando se instancia un objeto el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece el objeto.

Desde el punto de vista de la programación estructurada, una ***clase*** se asemejaría a un ***módulo***, los ***atributos*** a las ***variables globales*** de dicho módulo, y los ***métodos*** a las ***funciones*** del módulo.



Estructura general de un programa en java

La estructura general de una ***clase principal*** en Java se explica a continuación.

En el caso de ser una clase distinta de la clase principal, el método **`main()`** no suele aparecer.

```java
/**
* Estructura de una clase en Java
*/
public class NombreDeClase {
   //Declaración de los atributos de la clase
   //Declaración de los métodos de la clase
   //El método main indica donde empieza la ejecución
   public static void main (String[] args){
   //Declaración de las variables del método
   //Sentencias de ejecución del método
   }
}
```

Comentario: el programa empieza con un comentario, se representa:

- Para empezar el comentario: **/****
- Para finalizar el comentario: ***/**
- Para comentarios de una línea: **//**

Definición de la clase: después del comentario se define la clase, que tiene que ser igual que el nombre del fichero que se guarda.

Todo lo que contiene la clase está entre las llaves de abrir y cerrar. {.....}.

Atributos de clase: permiten guardar la información de un objeto, son las variables que definen el estado de los objetos.

Declaración de métodos: determinan el comportamiento del objeto.

Definición de método: a continuación escribimos el método main().

Todos los programas en Java deben de tener un método main().

Declaración de variables.

Sentencias: dentro del método main(), en el bloque delimitado por { y } se escriben las sentencias.

Es la parte que se ejecuta del programa.

Hola Mundo

```java
/**
* @author Antonio Rivero
* @version 1.00 2014/4/9
*/
public class HolaMundo {
  public static void main(String[] args) {
    System.out.println("Hola mundo");
  }
}
```

Identificadores

Los identificadores son los nombres que se les da a las variables, clases, interfaces, atributos y métodos de un programa.

Reglas para la creación de identificadores:

Java hace distinción entre mayúsculas y minúsculas.

Pueden estar formados por cualquiera de los caracteres del código Unicode, pero el primer carácter no puede ser un dígito numérico y no pueden utilizarse espacios en blanco ni símbolos coincidentes con operadores.

La longitud máxima de los identificadores es prácticamente ilimitada.

No puede ser una palabra reservada del lenguaje ni los valores lógicos true o false.

No pueden ser iguales a otro identificador declarado en el mismo ámbito.

Por convenio, los nombres de las variables y los métodos deberían empezar por una letra minúscula y los de las clases por mayúscula.

Además, si el identificador está formado por varias palabras la primera se escribe en minúsculas (excepto

para las clases) y el resto de palabras se hace empezar por mayúscula.

Estas reglas no son obligatorias, pero son convenientes ya que ayudan al proceso de codificación de un programa, así como a su legibilidad. Es más sencillo distinguir entre clases y métodos o variables.

Sentencias y expresiones

Las ***sentencias de asignación*** almacenan el valor representado por el lado derecho de la sentencia en una variable nombrada a la izquierda.

```
precio = precioDelBoleto;
```

Las sentencias de asignación funcionan tomando el valor de lo que aparece del lado derecho del operador y copiando dicho valor en una variable ubicada en el lado izquierdo.

Normalmente, las sentencias se ponen unas debajo de otras, aunque sentencias cortas pueden colocarse en una misma línea.

He aquí algunos ejemplos de sentencias:

```
int i=1;
import java.awt.*;
System.out.println("Hola Mundo");
rect.mover(10, 20);
```

La parte de la derecha se denomina una ***expresión:*** las expresiones son cosas que la computadora puede evaluar.

Una regla es que el ***tipo de una expresión*** debe coincidir con el ***tipo de la variable*** a la que es asignada.

Ejemplo:

// Esta expresión asigna un valor a la variable *x*.

```
x = 123;
```

// Esta expresión realiza una operación.

```
Y = (x+100)/4;
```

// Esta expresión es una llamada a una función miembro *calcularArea* desde un objeto *circulo* de una clase determinada.

```
area = circulo.calcularArea(2.5);
```

// Esta expresión reserva espacio en memoria para un objeto de la clase *Rectangulo* mediante la llamada a una función especial denominada constructor.

```
Rectangulo r = new Rectangulo(10, 10, 200, 300);
```

Atributos

Las clases presentan el estado de los objetos a los que representan mediante variables denominadas ***atributos***.

Los atributos permiten guardar la información de un objeto.

Estos datos se almacenan en ***campos*** o ***atributos*** que se declaran como variables en el ámbito de la clase.

También son conocidos como ***variables de instancia***.

Variables

Una variable es un nombre que se asocia con una porción de la memoria del ordenador, en la que se guarda el valor asignado a dicha variable.

Hay diferentes tipos de variables que requieren distintas cantidades de memoria para guardar datos.

Todas las variables han de declararse antes de usarlas.

La declaración consiste en una sentencia en la que figura el tipo de dato y el nombre que asignamos a la variable.

Una vez declarada se le podrá asignar valores.

Java tiene ***tres tipos de variables***:

- de instancia
- de clase
- locales

Variables de instancia

Se usan para guardar los atributos de un objeto particular.

Las ***variables de instancia*** también son conocidas como ***campos***.

Se definen siempre fuera de los métodos y constructores.

Variables de clase

También se conocen como ***variables estáticas***, siempre tienen el mismo valor para todos los objetos de una determinada clase.

En realidad no son variables sino ***constantes***.

En el siguiente ejemplo, ***PI*** es una ***variable de clase*** y ***radio*** es una ***variable de instancia***.

PI guarda el mismo valor para todos los objetos de la clase *Circulo*, pero el radio de cada círculo puede ser diferente

```
class Circulo{
   static final double PI=3.1416;
   double radio;
//...
}
```

Variables locales

Se utilizan dentro de los métodos.

En el siguiente ejemplo *area* es una variable local a la función *calcularArea* en la que se guarda el valor del área de un objeto de la clase *Circulo*.

Una variable local existe desde el momento de su definición hasta el final del bloque en el que se encuentra.

```
class Circulo{
//...
   double calcularArea(){
      double area=PI*radio*radio;
      return area;
   }
}
```

En el lenguaje Java, las variables locales se declaran en el momento en el que son necesarias.

Es una buena costumbre inicializar las variables en el momento en el que son declaradas.

Delante del nombre de cada variable se ha de especificar el tipo de variable que he destacado en letra negrita.

Las variables son uno de los elementos básicos de un programa, y se deben:

- Declarar
- Inicializar
- Usar

Para declarar una variable hay que seguir una serie de normas:

1. El primer carácter del nombre debe ser una letra mayúscula o minúscula, “_” o “$”.
2. No pueden utilizarse como nombres de variables las palabras reservadas de JAVA.
3. Los nombres deben ser continuos, es decir, sin espacios en blanco.
4. Los identificadores de variables son sensibles a las mayúsculas y a las minúsculas.

Métodos

Los ***métodos*** son las funciones mediante las que las clases representan el comportamiento de los objetos.

En dichos métodos se modifican los valores de los atributos del objeto, y representan las capacidades del objeto.

Sintaxis General

```
[especificadores] tipoDevuelto nombreMetodo([lista parámetros]) [throws listaExcepciones]
{
   //instrucciones
   [return valor;]
}
```

Los elementos que aparecen entre corchetes son opcionales.

Especificadores

Son opcionales y determinan el tipo de acceso al método.

Un especificador de acceso o alcance es una palabra reservada que nos permite controlar nivel de alcance que tiene cada variable, método o clase.

Tenemos: public, private, protected, static, final.

public

Establece un nivel de acceso total, es decir una clase public puede ser usada por otras clases del mismo paquete y de otros paquetes no relacionados, lo mismo para los métodos y variables.

private

Establece un nivel restringido de acceso, en el cual los miembros declarados como tal solo son accesibles dentro de la clase, aunque la clase herede de esta última, las variables y métodos private son usados cuando los clientes de una clase no deben preocuparse como se efectúan las acciones claves de un programa.

Establecer miembros private establece el mayor nivel de seguridad.

protected

Establece un nivel de acceso intermedio entre public y private en el que los miembros solo son accesibles por clases y métodos dentro del mismo paquete.

static

El modificador static crea variables y metodos de clase (a esto es lo que nos referimos cuando hablamos de variables de clase) estos objetos pueden ser llamados desde cualquier parte de la clase modificando sus valores reales de forma definitiva.

La diferencia con una variable normal es que cuando llamamos a una variable de instancia (normal) cambiamos su valor solo en ese instancia (metodos, etc..) en cambio una variable estática cambia de valor

definitivamente cada vez que se llama, en palabras simples actualizamos su valor real desde cualquier parte del programa.

final

El modificador final, crea una constante, en realidad en java no existen las constantes, pero final hace que las variables imiten su comportamiento.

Cualquier intento por modificar una variable final creara un error en tiempo de ejecución.

Un ejemplo de final, es la variable PI, de la clase Math

tipoDevuelto

Indica el tipo del valor que devuelve el método.

En Java es imprescindible que en la declaración de un método se indique el tipo de dato que ha de devolver.

El dato se devuelve mediante la instrucción return.

Si el método no devuelve ningún valor este tipo será void.

nombreMetodo

Es el nombre que se le da al método.

Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

Lista de parámetros

Son opcionales, después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros, también llamados argumentos, separados por comas.

Estos parámetros son los datos de entrada que recibe el método para operar con ellos.

Un método puede recibir cero o más argumentos.

Se debe especificar para cada argumento su tipo separados por comas.

Los paréntesis son obligatorios aunque estén vacíos.

Los parámetros pueden ser también objetos.

Los ***tipos simples*** de datos se pasan siempre por ***valor*** y ***los objetos y vectores*** por ***referencia***.

throws listaExcepciones

Es opcional, indica las excepciones que puede generar y manipular el método.

return

Se utiliza para devolver un valor.

La palabra clave return va seguida de una expresión que será evaluada para saber el valor de retorno.

Esta expresión puede ser compleja o puede ser simplemente el nombre de un objeto, una variable de tipo primitivo o una constante.

El tipo del valor de retorno debe coincidir con el tipoDevuelto que se ha indicado en la declaración del método.

Si el método no devuelve nada (tipoDevuelto = void) la instrucción return es opcional.

Un método puede devolver un tipo primitivo, un array, un String o un objeto.

Un método tiene un único punto de inicio, representado por la llave de inicio {.

La ejecución de un método termina cuando se llega a la llave final } o cuando se ejecuta la instrucción return.

La instrucción return puede aparecer en cualquier lugar dentro del método, no tiene que estar necesariamente al final.

Implementación de Métodos

Describir lo que el método debe hacer.

Determinar las entradas del método.

Determinar los tipos de las entradas.

Determinar el tipo del valor retornado.

Escribir las instrucciones que forman el cuerpo del método.

Prueba del método: diseñar distintos casos de prueba.

Métodos get y set

Son la forma de acceder a atributos de una clase.

Generalmente, se usan con atributos privados, ya que a los públicos se puede acceder directamente sin tener que acudir a ellos.

Método get

Lo usamos en las clases para mostrar u obtener el valor de un atributo.

Sintaxis:

```
public tipo_dato_atributo getAtributo(){
     return atributo;
}
```

Métodos set

Lo usamos en las clases para inicializar o modificar el valor de un atributo.

Sintaxis:

```
public void setAtributo(tipo_dato_atributo variable){
    this.atributo = variable;
}
```

Constructores

Tiene las siguientes características:

- Tiene el mismo nombre de la clase.
- Es el primer método que se ejecuta.
- Se ejecuta de forma automática.
- No puede devolver datos.
- Se ejecuta una única vez.
- El objetivo es inicializar los atributos.

Los constructores son métodos para crear nuevas instancias de una clase.

Sirve para inicializar el objeto.

Si no se especifica ningún constructor Java crea el constructor vacío.

Se pueden sobrecargar los constructores para aceptar diferentes tipos de argumentos.

Se puede invocar desde el constructor a otro constructor de la clase.

`this(argumentos)`

Se puede invocar al constructor de la superclase para configurar las variables heredades.

`super(argumentos)`

Deben ser la primera instrucción.

this

Dentro de un método, hace referencia al objeto al que se aplica el método.

Esta palabra del lenguaje se utiliza, cuando existe ambigüedad entre nombres de parámetros de un método y atributos de la clase.

Sólo aparece en métodos de instancia, si no hay instancia, no hay **`this`**.

Puede acceder a variables aunque estén ocultas.

En el siguiente método:

```
public void ponGrupo(String grupo. Horario horario){
   this.grupo = grupo;
   this.horario= horario;
```

`this` funciona como una referencia especial, de forma que **`this.grupo`** se refiere al atributo **`grupo`** declarado en la clase, para diferenciarlo de la variable grupo declarado como parámetro del método.

super

Hace referencia a la superclase del objeto

Muy útil al redefinir métodos

En los constructores, para seleccionar el constructor de la superclase

Sobrecarga

La única limitación en la elección del nombre de un método es que, en una clase, todos los métodos deben tener diferente signatura, básicamente distinto nombre y parámetros.

Esto permite que existan varios métodos con el mismo nombre pero con diferentes parámetros.

Por ejemplo, se podían tener dos métodos de nombre **`Arboles()`**.

```
public class Arboles  {
    public Arboles() {
    }
    public Arboles(String tipo) {
    }
    public Arboles(int altura) {
    }
    public Arboles(int altura,String tipo) {
    }
    public Arboles(int altura,String tipo,int lado) {
    }
}
```

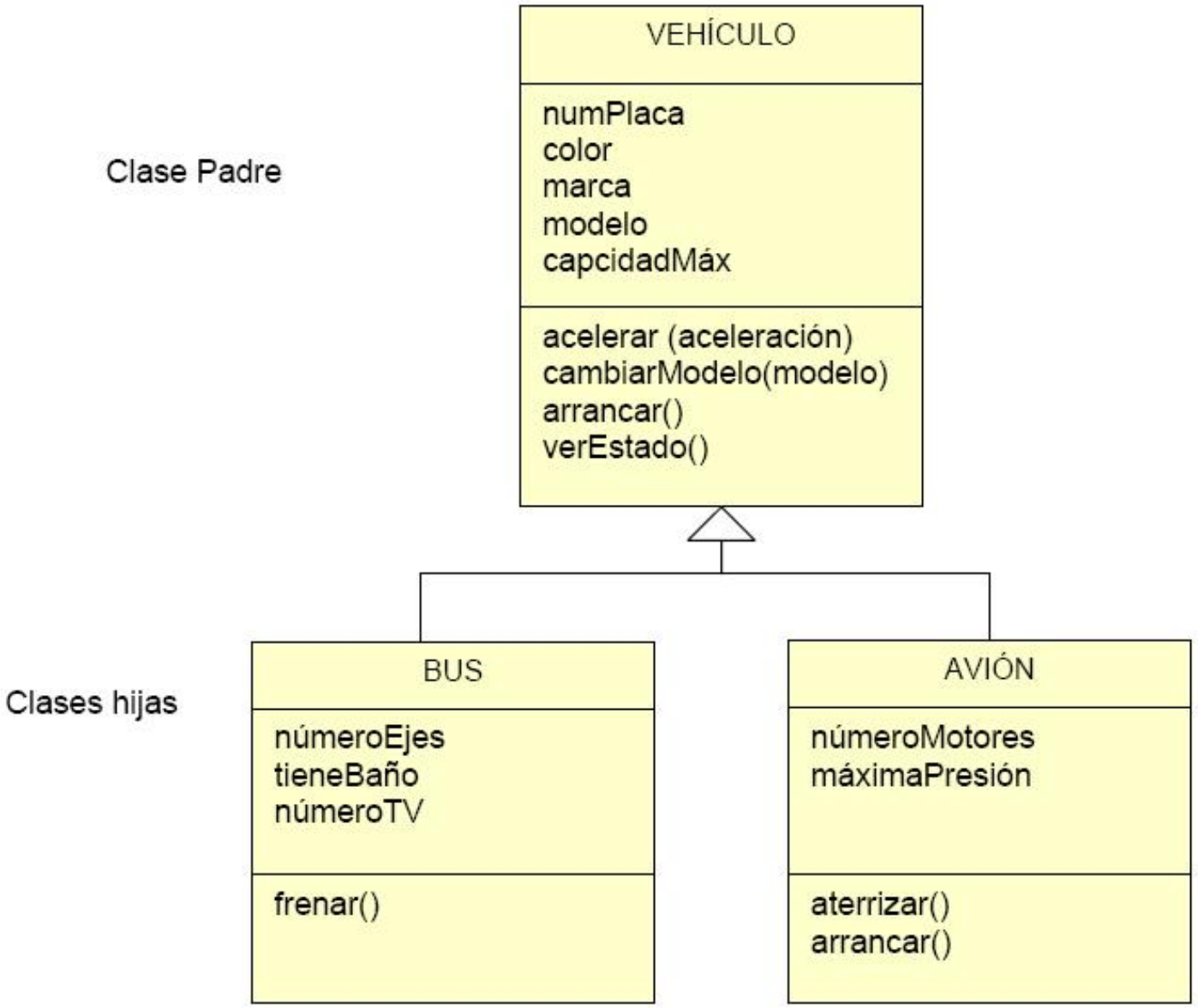
Dependiendo de los valores de los argumentos con que se llame al método **`Arboles`**, se ejecutaría uno u otro de los definidos.

Reutilización y Herencia

La herencia es el mecanismo fundamental de relación entre clases en la programación orientada a objetos.

Relaciona las clases de manera jerárquica, una clase ***padre*** sobre otras clases ***hijos***.

Los descendientes de una clase heredan todos los ***atributos*** y ***métodos*** que sus ascendientes hayan especificado como ***heredables***, además de crear los suyos propios.


Clase Padre
VEHÍCULO
numPlaca
color
marca
modelo
capcidadMáx
acelerar (aceleración)
cambiarModelo(modelo)
arrancar()
verEstado()
Clases hijas
BUS
númeroEjes
tieneBaño
númeroTV
frenar()
AVIÓN
númeroMotores
máximaPresión
aterrizar()
arrancar()

Ventajas de la herencia

Evita la duplicación de código. El uso de la herencia evita la necesidad de escribir copias de código idénticas o muy similares dos veces o con frecuencia, aún más veces.

Se reutiliza código. El código que ya existe puede ser reutilizado. Si ya existe una clase similar a la que necesitamos, a veces podemos crear una subclase a partir de esa clase existente y reutilizar un poco de su código en lugar de implementar todo nuevamente.

Facilita el mantenimiento. El mantenimiento de la aplicación se facilita pues la relación entre las clases está claramente expresada. Un cambio en un campo o en un método compartido entre diferentes tipos de subclases se realiza una sola vez.

Facilita la extendibilidad. En cierta manera, el uso de la herencia hace mucho más fácil la extensión de una aplicación.

Esta propiedad permite la reutilización del código.

Resulta muy fácil aprovechar el código de clases existentes, modificándolas mínimamente para adaptarlas a las nuevas especificaciones.

Polimorfismo

El polimorfismo es muy parecido o más bien tiene sus bases en la capacidad de **herencia** que presentan los lenguajes orientados a objetos.

En la herencia, las clases padres comparten métodos con las clases hijas.

Con el polimorfismo se hace prácticamente lo mismo pero en vez de clases padres se tiene ***clases abstractas***.

Las clases abstractas tienen métodos abstractos es decir métodos que solo están declarados sus nombres pero su forma de actuar difiere de una clase hija a otra.

Los métodos actúan dependiendo de la clase que haga mención del método declarado en la clase abstracta.

Clases Abstractas

La abstracción permite resaltar lo más representativo de algo sin importar los detalles.

La estructura es prácticamente igual, poseen nombre, atributos y métodos pero para que una clase sea abstracta la condición es que ***al menos*** uno de sus métodos sea abstracto

Una clase Abstracta es similar a una clase normal.

Se agrega la palabra reservada **abstract** y no se especifica el cuerpo del método.

Características de una Clase Abstracta.

Una clase Abstracta no puede ser instanciada.

No se pueden crear objetos directamente, solo puede ser heredada.

Si al menos un método de la clase es abstract, esto obliga a que la clase completa sea definida abstract, sin embargo la clase puede tener el resto de métodos no abstractos.

Los ***métodos abstract*** no llevan cuerpo, no llevan los caracteres {}.

Todas las clases hijas que hereden de una clase ***abstract*** debe implementar obligatoriamente todos los métodos abstractos de la superclase.

La etiqueta "***@Override***" sirve para indicar en el código que estamos reescribiendo o especializando un método que se encuentra en la clase padre y que queremos redefinir en la clase hija

¿Cuándo Utilizarlas?

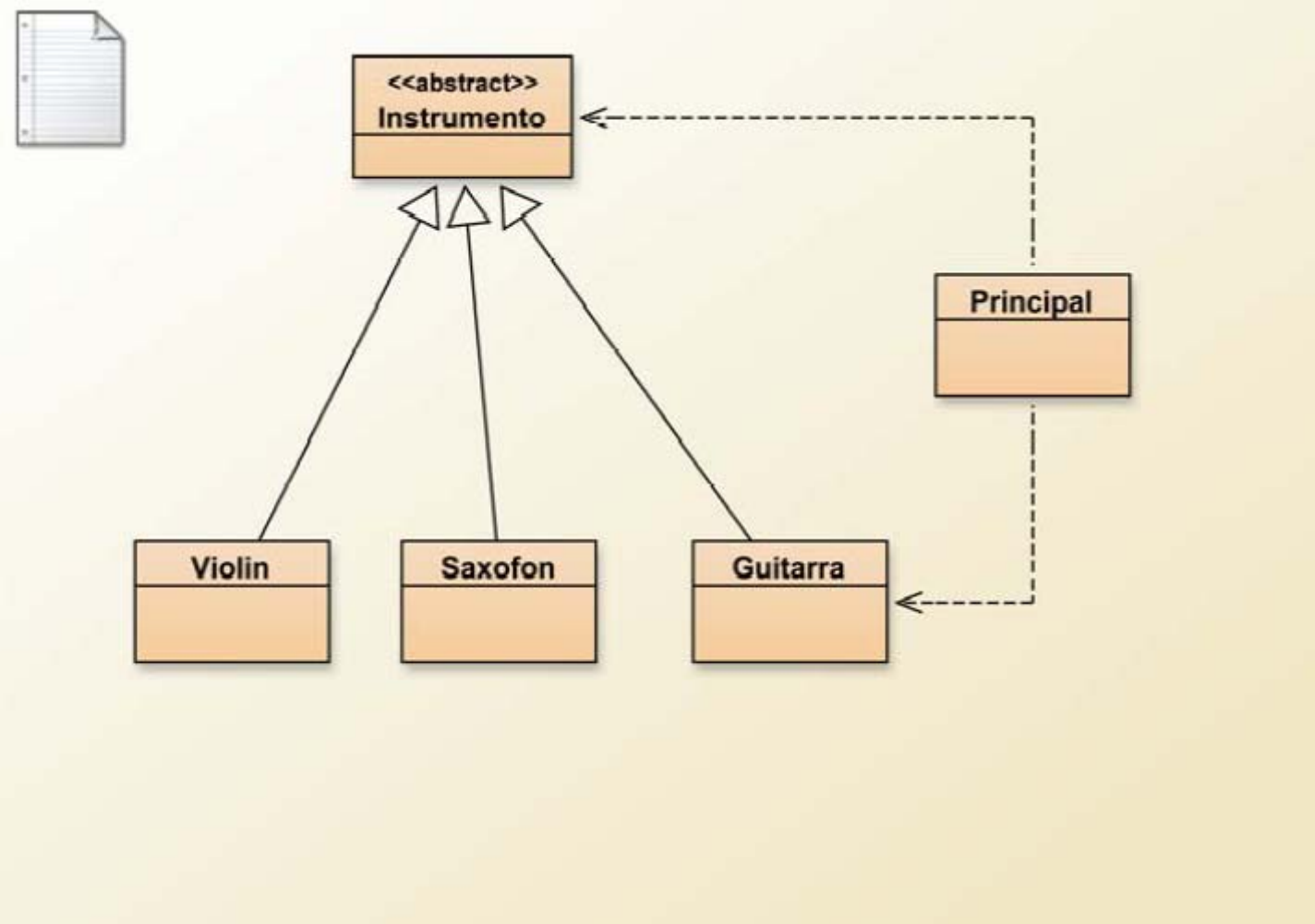
Al trabajar clases y métodos abstractos, no solo mantenemos nuestra aplicación más organizada y fácil de entender sino que también al no poder instanciar una clase abstracta nos aseguramos de que las propiedades específicas de esta, solo estén disponibles para sus clases hijas.

Con las ***Clases Abstractas*** lo que hacemos es definir un proceso general que luego será implementado por

las clases concretas que hereden dichas funcionalidades.

Es decir, si tengo una clase que hereda de otra Abstracta, estoy obligado a poner en el código, todos los métodos abstractos de la clase padre, pero esta vez serán métodos concretos y su funcionalidad o cuerpo será definido dependiendo de para que la necesite, de esa manera si tengo otra clase que también hereda del mismo padre, implementaré el mismo método pero con un comportamiento distinto.

Ejemplo Instrumentos



En el diagrama vemos una clase Abstracta **Instrumento**, posee una propiedad tipo String y un método abstracto **tocar**(), vemos también las clases hijas **Guitarra**, **Saxofon** y **Violin** que para este ejemplo solo utilizaremos (mediante la herencia) las propiedades de la clase Padre.

Todos los instrumentos musicales se pueden tocar, por ello creamos este método abstracto, ya que es un proceso común en todos los instrumentos sin importar el detalle de como se tocan, pues sabemos que una guitarra no se toca de la misma manera que el

saxofón, así al heredar de la clase Instrumento, todas sus clases hijas están obligadas a implementar este método y darle la funcionalidad que le corresponda.

Como vemos cada una de las clases concretas implementan el método **tocar()** y le dan la funcionalidad dependiendo de cómo se toque el instrumento, también en cada constructor de las clases definimos el **tipo**, pero si nos fijamos bien en las clases concretas no tenemos la variable tipo declarada, pues estamos usando la variable heredada de la clase **Instrumento**.

Hay que tener en cuenta que cuando trabajamos con clases Abstractas, estas solo pueden ser heredadas pero no instanciadas, esto quiere decir que no podemos crear objetos directamente de estas clases, con ***new***.

Como vemos en la clase Principal tenemos la lógica para ejecutar nuestra aplicación y usamos el concepto de Polimorfismo para crear los objetos de tipo Instrumento por medio de sus clases Hijas, pero en ningún momento creamos un objeto como instancia directa de la clase abstracta.