

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Introducción

© ADR Infor SL

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Indice

| | |
|--|-----------|
| Competencias y Resultados de Aprendizaje desarrollados en esta unidad | 4 |
| Introducción. | 5 |
| Objetivos | 5 |
| Cuestiones básicas de Java | 5 |
| ¿Qué es Java? | 5 |
| Concepto de programa | 6 |
| Lenguajes de programación | 7 |
| Lenguajes de programación de alto nivel y de bajo nivel | 7 |
| Hardware y software (Arquitectura ordenador + sistema operativo) | 9 |
| Plataforma | 10 |
| Software Java | 11 |
| JRE | 12 |
| Componentes | 12 |
| Java SE (Java Platform Standard Edition) | 13 |
| Java EE (Java Platform Enterprise Edition) | 14 |
| Java ME (Java Platform Micro Edition) | 14 |
| Descarga e instalación de JDK | 15 |
| Programación orientada a objetos | 15 |
| Introducción | 16 |
| Objetos | 17 |
| Clases | 17 |
| Modelo de objetos | 20 |
| Ejemplos de herencia | 26 |
| Programa Java | 31 |
| Métodos | 33 |
| Declaración genérica de un método | 33 |
| Nomenclatura oficial | 35 |
| Escribir un programa Java | 35 |
| Entorno de desarrollo | 38 |
| JDK | 38 |
| Eclipse versión Juno | 39 |
| Resumen | 42 |
| Ejemplos | 43 |
| Ejemplo 1 | 43 |
| Ejemplo 2 | 44 |
| Ejemplo 3 | 45 |
| Ejemplo 4 | 45 |
| Ejemplo 5 | 47 |
| Ejemplo 6 | 47 |
| Ejemplo 7 | 49 |
| Ejemplo 8 | 49 |
| Ejercicios | 52 |
| Ejercicio. Calcular el cuadrado de una suma. | 52 |
| Recomendaciones. | 52 |
| Datos a mostrar por consola. | 52 |
| Recursos | 54 |
| Enlaces de Interés | 54 |
| Glosario. | 54 |

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:

Conocer y entender qué es un programa

Resultados de Aprendizaje:

- Entender el concepto de programa
-

Competencia:

Toma de contacto con el lenguaje Java

Resultados de Aprendizaje:

- Conocer los diferentes lenguajes de programación y sus tipos
 - Conocer los componentes básicos que se deben usar para poder comenzar a desarrollar en Java
 - Escribir un programa en Java
-

Competencia:

Obtener e instalar las herramientas necesarias para crear un entorno de desarrollo

Resultados de Aprendizaje:

- Descarga e instalación de las herramientas necesarias para la realización del curso de forma óptima.
-

Competencia:

Conocer y entender la programación orientada a objetos

Resultados de Aprendizaje:

- Adquirir una idea general de lo que significa la programación orientada a objetos
- Conocer y entender que significan los objetos en Java

Introducción.

Objetivos

OBJETIVOS DEL CURSO

- Aprender las características fundamentales de Java, un lenguaje de última generación basado en la Programación Orientada a Objetos.
- Describir y conocer las construcciones y elementos de la sintaxis del lenguaje: variables, bucles, estructuras condicionales, etc.
- Conocer y utilizar las API proporcionadas por Java para la creación de estructuras y colecciones, tratamiento de excepciones, concurrencia, funcionalidades de entrada/salida, etc.
- Conocer los fundamentos de la Programación Orientada a Objetos a través del Lenguaje Java.
- Manejar los elementos principales de Java: clases, constructores, etc.
- Utilizar la herencia para la creación de aplicaciones Java.

OBJETIVOS DE LA UNIDAD

- Entender qué es un programa.
- Tomar contacto con el lenguaje Java.
- Obtener las herramientas necesarias para el seguimiento del curso.
- Instalar estas herramientas y aprender su manejo.
- Conocer y entender la programación orientada a objetos.

Cuestiones básicas de Java

¿Qué es Java?

Al hablar de Java, nos estamos refiriendo a tres cosas asociadas con la programación de software:

- un lenguaje
- una plataforma
- un fenómeno.

La eficacia y la flexibilidad del lenguaje permitieron crear una plataforma tan extensa que tiene alcance lo mismo para aplicaciones de propósito general en computadoras personales, para el funcionamiento de dispositivos móviles y aparatos electrónicos, y hasta para sitios web.

Este alcance ha creado un verdadero fenómeno tecnológico; tanto, que hoy por hoy hay más de 4 500 millones de equipos que tienen instalado Java.

Entonces, en estricto sentido, esta tecnología sirve para hacer aplicaciones, virtualmente, para cualquier componente que tenga un procesador de software.

La plataforma para el desarrollo de Java está dividida en tres ediciones: la estándar (JSE), la empresarial (JEE) y la de dispositivos móviles (JME). La primera contiene, entre muchas otras cosas, los elementos del lenguaje, los objetos para las interfaces gráficas y los mecanismos de conexión a base de datos, que son lo primero que debe saberse para desarrollar aplicaciones Java básicas.

El propósito del lenguaje Java es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que implica que las aplicaciones desarrolladas con Java son **multi plataforma**.

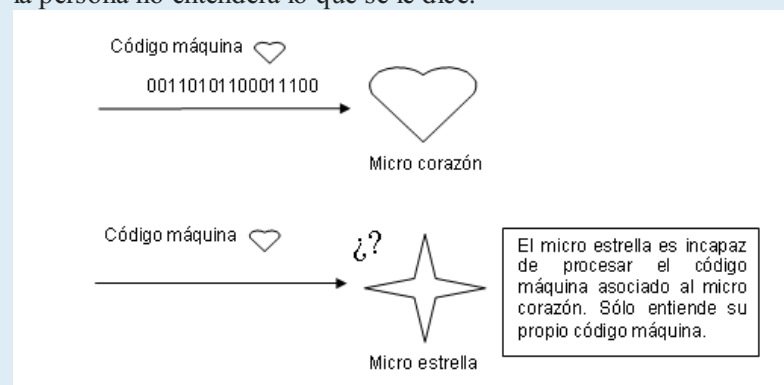
Concepto de programa

Vocabulario: Programa

Un programa es un conjunto de **instrucciones que indican a un procesador**, que puede estar o no en un ordenador, las acciones que debe ejecutar. Estas instrucciones están expresadas en lenguaje o **código máquina**, único lenguaje que entienden los procesadores.

Cada procesador tiene su **propio código máquina** de modo que si a un procesador le llegan unas instrucciones asociadas a un programa no expresadas en su código máquina, no es capaz de interpretarlas.

Una analogía es el idioma. Si alguien se dirige a una persona en un idioma que desconoce la persona no entenderá lo que se le dice.



Estas **instrucciones** de código máquina son una serie de **secuencias de bytes**. Cada byte se compone de 8 bits y un bit puede ser un 0 o un 1. Un bit es la unidad mínima de información con la que es capaz de trabajar un procesador. Un ejemplo de byte podría ser este: 10011010.

Lenguajes de programación

Vocabulario: Lenguaje de Programación

Es un conjunto de normas, instrucciones y códigos prefijados que permiten construir programas capaces de ser interpretados por procesadores.

Un lenguaje de programación es la herramienta utilizada para crear programas.

Es como el **alfabeto de una lengua**: para que una persona pueda comunicarse y entenderse con otra debe aprender las vocales, las consonantes, su pronunciación, cómo se unen letras para formar palabras inteligibles, cómo se agrupan palabras para formar frases, etc. Este alfabeto y sus normas de aprendizaje cambian de una lengua a otra. Ocurre lo mismo en los lenguajes de programación: para que un programador pueda relacionarse con un procesador a través de un programa es necesario que aprenda las reglas y características básicas del único idioma que entienden: su **código máquina**.

Los **programadores no suelen trabajar directamente con código máquina** porque hay muchos tipos (tantos como procesadores) y, además, porque se programa con un lenguaje muy distinto al que habitualmente se emplea (recordar que las instrucciones de código máquina se expresan en bytes). Para evitar estos dos problemas se emplean los lenguajes de programación de alto nivel.

Lenguajes de programación de alto nivel y de bajo nivel

Vocabulario: Lenguaje de Alto Nivel

Un lenguaje de alto nivel se compone de palabras, generalmente inglesas, entendibles por los humanos. Su principal característica es que las **instrucciones asociadas a los programas** creados con ellos **no trabajan directamente con el procesador**, sino que previamente **deben pasar a través de un filtro o compilador** que las prepara adecuadamente para que el procesador sepa interpretarlas.

Las instrucciones asociadas a programas creados con lenguajes de alto nivel deben transformarse en instrucciones de código máquina para que el procesador pueda interpretarlas; esto se hace con un compilador (es el filtro mencionado anteriormente).

Así, un **compilador** es un programa que

- **Comprueba** el cumplimiento de una serie de normas que deben cumplir las instrucciones de un programa.
- **Transforma** dichas instrucciones expresadas en lenguajes de alto nivel en instrucciones de código máquina directamente ejecutables por un procesador.

Java, Visual Basic, Visual Basic.NET, C, C++, C#, Pascal, Cobol, SmallTalk, Delphi, Perl, Python, etc.

Vocabulario: Lenguaje de Bajo Nivel

Un lenguaje de bajo nivel trabaja con **instrucciones directamente interpretables** por un procesador.

El uso de la palabra **bajo** en su denominación **no** implica que el lenguaje sea **inferior a un lenguaje de alto nivel**, si no que se refiere a la **reducida abstracción entre el lenguaje y el hardware**.

Se utiliza este tipo de lenguajes para programar tareas críticas de los Sistemas Operativos, de aplicaciones en tiempo real o controladores de dispositivos.

El **ensamblador** es un tipo de lenguaje de bajo nivel.

Normalmente, cada procesador tiene distinto código máquina, y un programa compilado para un determinado procesador no vale para otro (una novela escrita en castellano no se entiende en China a no ser que se traduzca). Con los programas que se quieren ejecutar en máquinas que no tienen el mismo procesador ocurre lo mismo.

Se deben tener tantos compiladores como procesadores distintos se tengan y volver a compilar el programa para que funcione en el procesador concreto.

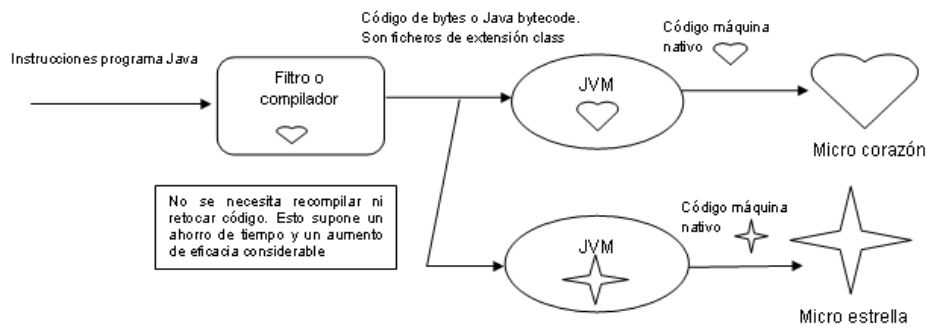
Esto es una limitación que Java ha resuelto mediante la JVM (Java Virtual Machine) o **Máquina Virtual de Java**.

Vocabulario: Máquina Virtual de Java

La JVM es un programa capaz de crear, a partir de código Java compilado que recibe el nombre de código de bytes o Java bytecode, el código máquina nativo asociado al procesador con el que se está trabajando.

En la analogía de la novela se podría decir que la JVM sería un **traductor mágico** capaz de traducir una novela escrita en cualquier lengua, a nuestra lengua materna.

Introducción



La inclusión de la JVM entre el compilador y el micro va en detrimento de la velocidad de ejecución y rapidez de interacción con el usuario, por lo que Java es un lenguaje lento en relación a otros como C++, por ejemplo

En función de lo que se considere prioritario para una aplicación se optará por un lenguaje u otro

Hardware y software (Arquitectura ordenador + sistema operativo)

Hablando a nivel de ordenador, **además del procesador debe considerarse el sistema operativo** (s.o. a partir de ahora) que trabaja en él.

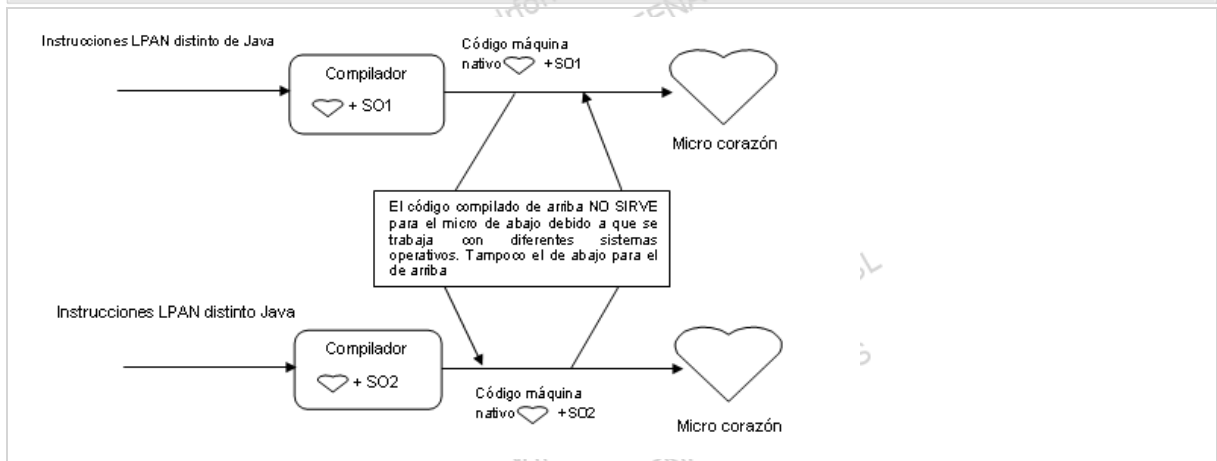
El código máquina asociado a un procesador de un ordenador depende del s.o. que tenga instalado;

Si se dispone de dos ordenadores con el mismo procesador pero con dos s.o. distintos, uno con Windows y otro con Linux, y se realiza un programa que se compila usando el compilador asociado al primer ordenador (monta Windows), este programa, ya compilado, no podría ejecutarse en el segundo ordenador (monta Linux). Si se desea ejecutar debería compilarse con el compilador asociado al segundo ordenador.

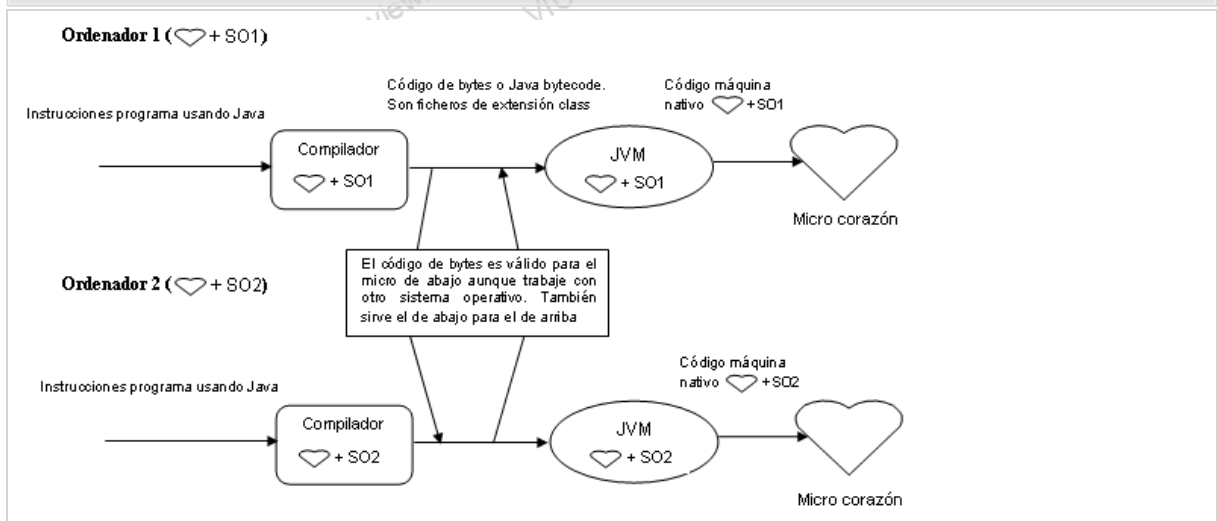
Esto significa que el programa no es portable o multi plataforma.

En Java, mediante la JVM, se ha conseguido que todos los programas sean portables o multi plataforma ya que la ejecución de las instrucciones Java no dependen del s.o. sino de la JVM.

Si no se utiliza lenguaje Java:



Si se utiliza Java:



Plataforma

Vocabulario: Plataforma

Todo **entorno físico o lógico sobre el que se puede ejecutar un programa o aplicación** se denomina **plataforma**. La mayoría de plataformas actuales combinan un ordenador y un s.o.

Actualmente se cuenta con dos tipos de plataformas a utilizar:

- **Entorno o plataforma PC:** ordenadores con una arquitectura interna que sigue unos parámetros de diseño creados por IBM hace 20 años y a un sistema operativo (Windows 95, 98, 2000, NT, XP, Linux, Unix, Solaris, AIX de IBM, HP-UX de Hewlett-Packard, etc.).
- **Entorno o plataforma MAC:** ordenadores con unas características de diseño internas definidas por Apple y a un sistema operativo llamado Mac O.S.

No es preciso disponer de un ordenador para ejecutar un programa Java, basta con diseñar máquinas virtuales adecuadas para esos dispositivos y un mecanismo para introducir el código de bytes (habitualmente Internet).

En la actualidad, la programación en entornos no PC ni Mac, es una parte muy importante de Java, y además en continua evolución y desarrollo. El nombre que recibe dicha parte es Jini.

[Página oficial de Jini en oracle](#) En este enlace podemos ver las novedades de Java respecto al desarrollo para diferentes entornos.

Existen **televisores** interactivos, **teléfonos** celulares, **relojes**, **cadenas de música**, **electrodomésticos**, etc. en las que puede ejecutarse un programa Java.

Una cadena de música con este procesador y conectada a Internet permitiría escuchar directamente archivos mp3 almacenados en páginas web o escuchar conciertos en directo independientemente del lugar donde se desarrolle el mismo, una lavadora podría ejecutar una secuencia de lavado indicada por el usuario estando éste en otro país, un microondas podría ponerse en marcha de forma remota, un aparato de aire acondicionado podría controlarse también remotamente etc.

Software Java

Vocabulario: Java

Es un **lenguaje de programación** de propósito general, **concurrente**, **orientado a objetos** que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

La base del lenguaje Java es permitir que los desarrolladores de aplicaciones **escriban el programa una vez y lo ejecuten en cualquier dispositivo** (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Sun Microsystems es la empresa americana que creó el lenguaje de programación Java allá por el año 1995. Se promocionan diciendo que son el punto en las empresas puntocom y abogan por escribir código que cumpla la premisa "Write Once, Run Anywhere" que se podría traducir como "**Escribe código una vez donde tú quieras y puedes ejecutarlo cuantas veces quieras donde tú quieras, sin ningún retoque ni recompilación**".

Actualmente ha sido comprada por la empresa Oracle que es la que gestiona hoy en día las descargas y actualizaciones del producto.

Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados

JRE

Vocabulario: JRE

El **JRE** (Java Runtime Environment) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java.

El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web.

Oracle ofrece también el **SDK** de Java 2, o **JDK** (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el **compilador de Java**, **Javadoc** para generar documentación o el **depurador**.

Puede también obtenerse como un paquete independiente, y considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

Componentes

JRE se compone de Bibliotecas de Java que ofrecen apoyo para el desarrollo en Java. Algunos ejemplos de estas bibliotecas son:

| | |
|---|---|
| Las bibliotecas centrales | <ul style="list-style-type: none"> • Una colección de bibliotecas para implementar estructuras de datos como listas, arrays, árboles y conjuntos. • Bibliotecas para análisis de XML. • Seguridad. |
| Bibliotecas de internacionalización y localización. | <ul style="list-style-type: none"> • La API para acceso a bases de datos JDBC (Java DataBase Connectivity). • La interfaz JNDI (Java Naming and Directory Interface) para servicios de directorio. • RMI (Remote Method Invocation) y CORBA para el desarrollo de aplicaciones distribuidas. |
| Bibliotecas para la interfaz de usuario, que incluyen: | <ul style="list-style-type: none"> • El conjunto de herramientas nativas AWT (Abstract Window Toolkit), que ofrece componentes GUI (Graphical User Interface), mecanismos para usarlos y manejar sus eventos asociados. • Las Bibliotecas de Swing, construidas sobre AWT pero ofrecen implementaciones no nativas de los componentes de AWT. • APIs para la captura, procesamiento y reproducción de audio. |
| Implementación dependiente | Una implementación dependiente de la plataforma en que se ejecuta de la máquina virtual de Java (JVM), que es la encargada de la ejecución del código de las bibliotecas y las aplicaciones externas. |
| Plugins | Plugins o conectores que permiten ejecutar applets en los navegadores Web. |
| Java Web Start | Java Web Start, para la distribución de aplicaciones Java a través de Internet. |
| Documentación | Documentación y licencia. |

Java SE (Java Platform Standard Edition)

Vocabulario: Java SE

Java SE es el software necesario para ejecutar cualquier aplicación desarrollada para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio de un PC de escritorio.

Java SE constituye la plataforma base para programar cualquier aplicación en Java.

Se compone de un conjunto de librerías de clases que forman el corazón del lenguaje Java.

El software que incluye estas clases y un conjunto de herramientas que permiten desarrollar aplicaciones con el lenguaje Java se llama **SDK** (Software Development Kit) o **JDK** (Java Development Kit), ambas expresiones se refieren a lo mismo.

El JDK se puede descargar desde la página oficial de Oracle relacionada con Java <http://www.oracle.com/technetwork/java/index.html>.

Dentro del JDK se incluyen el compilador y la JVM (Java Virtual Machine) o Máquina virtual de Java.

Cada plataforma tiene su propia versión. En la página de Java dentro de Oracle puede descargarse el JDK para Windows, Linux, Solaris, etc.

Existen varias versiones, la más moderna es la 1.8.X aunque todavía se sigue utilizando de forma masiva la versión 1.7.

En el curso se va a trabajar con la versión 1.7, aunque se puede realizar con cualquier versión 1.4 o superior. Se recomienda un mínimo de 512 Mb de RAM para poder trabajar sin problemas.

Java EE (Java Platform Enterprise Edition)

Vocabulario: Java EE

Java EE es el software necesario para ejecutar cualquier aplicación desarrollada orientada a **entornos distribuidos empresariales o de Internet**.

Java EE es una plataforma del lenguaje Java destinada al desarrollo de aplicaciones empresariales estructuradas típicamente en tres capas:

- capa de presentación de datos
- capa de lógica de negocio
- capa de datos persistentes (bases de datos).

Se compone de un conjunto de estándares y bibliotecas Java que permiten la creación de las aplicaciones empresariales anteriormente citadas.

Esta plataforma, junto con Java SE, se utiliza para desarrollar aplicaciones que se almacenan en un servidor Java EE y se ejecutan en clientes web (típicamente navegadores web).

Sin conocer lo fundamental de Java SE, resulta inviable introducirse en Java EE.

Java ME (Java Platform Micro Edition)

Vocabulario: Java ME

Java ME es el software necesario para ejecutar cualquier aplicación desarrollada orientada a **entornos de limitados recursos**, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.

Plataforma del lenguaje Java destinada al desarrollo de aplicaciones para pequeños dispositivos móviles de memoria limitada, poca capacidad de procesamiento y con interfaces gráficas limitadas.

Típicamente teléfonos móviles, PDAs (Personal Assistant Digital), Pockets PCs, televisiones, relojes, sistemas de ayuda para automóviles, tarjetas, etc. Como en J2EE, la base para programar mediante J2ME, es J2SE.

Descarga e instalación de JDK**Descarga**

1. Acceder a <http://www.oracle.com/technetwork/java/index.html>, sección **Download/Java SE**.
2. Pulsar el botón Download en la sección Java SE Development Kit (JDK), seleccionar plataforma y lenguaje, y pulsar el enlace correspondiente.
3. Se descarga un fichero del tipo **jdk-xxx-windows-i586.exe** si nuestro procesador es de **32 bits** y **jdk-xxx-windows-x64.exe** si es de **64 bits**.
4. Descargado el exe, se ejecutará y se abrirá un asistente de instalación..

Instalación

1. Descargado el exe, se ejecutará y se abrirá un asistente de instalación.
2. El proceso es muy sencillo: hay que decir a todo que sí y seleccionar las opciones por defecto.



Descargar la JDK de Oracle

Programación orientada a objetos

Introducción

La **programación orientada a objetos** facilita la creación de software de calidad ya que **fomenta su fácil** mantenibilidad, alta escalabilidad y la reutilización de componentes software.

La programación orientada a objetos trata de amoldarse al modo de pensar del hombre y no al de la máquina. Esto es posible gracias a la forma racional con la que se manejan las abstracciones que representan las entidades del dominio del problema, y a propiedades como la herencia o el encapsulamiento.

El elemento básico de la POO es el **objeto**.

Vocabulario: Objeto

Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones o métodos que pueden modificar dicho estado.

Es complicado empezar a entender la programación orientada a objetos, pero para empezar basta saber que en la vida real cualquier concepto (sustantivo) sería un objeto: un ser humano, una ciudad, una cuenta bancaria, etc.

Otro elemento importante de la POO es el atributo.

Vocabulario: Atributo

un atributo es una especificación que **define** una **propiedad** de un Objeto, elemento o archivo. También puede referirse o establecer el valor específico para una instancia determinada de los mismos.

Se intuye que una Cuenta Bancaria podría definir una serie de atributos como

- número
- titular
- saldo

que asumirán distintos valores para cada una y que servirían para identificarla de forma única. Lo importante es pensar que todas las cuentas bancarias deberían contar con esos atributos.

Lo mismo ocurriría con Ser Humano. Pensando un poco podemos concluir que para nosotros un ser humano se define mediante los atributos

- color de ojos
- edad

- nombre

entre otros.

Objetos

Se puede definir objeto como el **encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios**. [Piattini et al., 1996].

Un objeto, además de un estado interno, presenta una interfaz para poder interactuar con el exterior. Es por esto por lo que se dice que **en la programación orientada a objetos "se unen datos y procesos"**, y no como en su predecesora, la programación estructurada, en la que estaban separados en forma de variables y funciones.

Un objeto consta de:

Tiempo de vida

la *duración de un objeto en un programa* siempre está limitada en el tiempo. La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Los objetos son creados mediante un mecanismo denominado instanciación y cuando dejan de existir se dice que son destruidos.

Estado

todo objeto posee un estado, definido por sus atributos o campos. Con él se definen las propiedades del objeto, y el estado en que se encuentra en un momento determinado de su existencia. Por ejemplo, en el caso de **CuentaBancaria** podría ser:

- **numero:** 1234567890123456
- **titular:** Jesús Fernández
- **saldo:** 10000

Comportamiento

todo objeto ha de presentar una interfaz, definida por sus métodos, para que el resto de objetos que componen los programas puedan interactuar con él.

Clases

Vocabulario: Clase

Las clases son abstracciones que representan a un conjunto de objetos con unas características y comportamiento comunes.

En este sentido, una clase puede considerarse como una plantilla o molde que sirve para la creación o instanciación de objetos.

Podemos volver a pensar en una cuenta bancaria como ejemplo, pero en este caso una clase sería una cuenta bancaria concreta.

¿Qué caracteriza a las cuentas bancarias? Todas tienen un número de veinte dígitos, están asociadas a un titular y presentan un saldo aunque, por supuesto, dichas características para cada cuenta bancaria pueden ser distintas.

- **Características comunes:** todas las cuentas tienen **número identificativo, titular y saldo**.
- **Comportamiento común:** en todas las cuentas se pueden realizar dos operaciones básicas: **depositar y retirar dinero**.

No es que las citadas anteriormente sean todas las características que definen a una cuenta bancaria, pero sí en las que se está interesado como responsable de modelar un objeto de negocio del sistema software que se está diseñando.

Los métodos permiten a las clases modificar el comportamiento de los objetos, cambiando los valores de los atributos e interactuando con otros métodos de la clase que expongan una determinada funcionalidad precisada.

Características comunes

Las características comunes deben representarse de alguna forma a nivel de código. Para ello utilizan los atributos de clase.

Las clases representan el estado de los objetos mediante los valores que dichos atributos asumen para cada uno de ellos. En la mayoría de lenguajes orientados a objetos, estos atributos son implementados mediante variables de instancia (declaradas justo después de la clase y accesibles a todos los métodos de la misma).

Para crear o instanciar un objeto de la clase CuentaBancaria deberemos escribir la siguiente línea de código. En este ejemplo instanciamos dos objetos de tipo CuentaBancaria:

```
CuentaBancaria cuenta1 = new CuentaBancaria("12345678901234567890", "Juan Sanz", 1000);
```

```
CuentaBancaria cuenta2 = new CuentaBancaria("98765432109876543210", "Sara Gil", 3000);
```

Comportamiento común

El comportamiento común se representa en el código en forma de métodos.

Los métodos permiten a las clases modificar el comportamiento de los objetos, cambiando los valores de los atributos e interactuando con otros métodos de la clase que expongan una determinada funcionalidad precisada.

Ejemplo de llamada a un método:

```
cuenta1.depositar(300);
```

Esta línea de código invoca al método depositar y provoca un cambio de valor en el atributo saldo del objeto de la clase CuentaBancaria cuya referencia es cuenta1: antes de la llamada, el saldo era de 1000, pero después será de 1300.

Ejemplos:

Características comunes

```
// Declaracion del paquete que va a contener a la clase
package cuentabancaria;
// Declaración de clase
public class CuentaBancaria{
    // Atributos de clase. Son variables de instancia
    private String numero;
    private String titular;
    private double saldo;
    // Método constructor de la clase. Es invocado cada vez que se crea o instancia un objeto de la clase
    public CuentaBancaria(String numeroCuenta, String titularCuenta, double cantidaInicial)
    {
        numero = numeroCuenta;
        titular = titularCuenta;
        saldo = cantidaInicial;
    }
    ...
}
```

Comportamiento común

```

...
// Método que ejemplifica la acción de retirar dinero
public void retirar(double cantidad) throws NoFondosDisponiblesException {
    if (saldo >= cantidad) {
        saldo = saldo - cantidad;
    }
    else
        throw new NoFondosDisponiblesException("No hay suficientes fondos");
}
// Método que ejemplifica la acción de depositar dinero
public void depositar(double cantidad){
    if(cantidad > 0){
        saldo = saldo + cantidad;
    }
}
...

```

Modelo de objetos

Existen una serie de principios fundamentales para comprender cómo se modeliza la realidad al crear un programa bajo el paradigma de la orientación a objetos. Estos principios son:

- Abstracción
- Encapsulación
- Modularidad
- Herencia
- Paso de mensajes
- Polimorfismo
- Relaciones entre objetos
- Visibilidad

No importa si ahora no se entienden bien estos conceptos. A medida que se vaya avanzando en el curso y conforme se vayan realizando ejercicios se comprenderán mejor.

Abstracción

Mediante la abstracción, la mente humana **modela la realidad en forma de objetos**. Para ello busca parecidos entre la realidad y la posible implementación de objetos del programa que simulen el funcionamiento de los objetos reales.

Los seres humanos no pensamos en las cosas como un conjunto de cosas menores; por ejemplo, no vemos un cuerpo humano como un conjunto de células, entendemos la realidad como objetos con comportamientos bien definidos. No necesitamos conocer los detalles de porqué ni cómo funcionan las cosas; simplemente solicitamos determinadas acciones en espera de una respuesta; cuando una persona desea desplazarse, su cuerpo le responde comenzando a caminar.

Pero **la abstracción humana se gestiona de una manera jerárquica, dividiendo sucesivamente sistemas complejos en conjuntos de subsistemas**, para así entender más fácilmente la realidad.

Esta es la forma de pensar que la orientación a objeto intenta cubrir.

Encapsulación

Permite a los objetos **elegir qué información es publicada** y qué información es ocultada al resto de los objetos. Para ello los objetos suelen presentar sus métodos como públicos y sus atributos como privados e inaccesibles desde otros objetos. Esto se hace para dotar al programador de mecanismos que permitan aplicar lógica de control y validación cuando otros objetos consulten o modifiquen los atributos.

Con el encapsulado de los datos se consigue que las personas que utilicen un objeto sólo tengan que comprender su interfaz, olvidándose de cómo está implementada, y en definitiva, reduciendo la complejidad de utilización.

En la clase CuentaBancaria los métodos setters(setSaldo, setNumero, setTitular) permiten actuar del modo comentado anteriormente, mientras que los getters(getSaldo, getNumero, getTitular) acceden al valor de los atributos.

```
...
public double getSaldo() {
    return saldo;
}
public void setSaldo(double saldo) {
    if(saldo < 0){
        // Mostrar mensaje al usuario indicándole que no es posible
    }else{
        this.saldo = saldo;
    }
}
public String getNumero() {
    return numero;
}
public void setNumero(String numero) {
    this.numero = numero;
}
public String getTitular() {
    return titular;
}
public void setTitular(String titular) {
    this.titular = titular;
}
...
```

Modularidad

Mediante la modularidad, se propone al programador **dividir su aplicación en varios módulos diferentes** (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos **con un sentido propio**.

Esta fragmentación **disminuye el grado de dificultad del problema** al que da respuesta el programa, pues se afronta el problema como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

Herencia

La mayoría de nosotros ve de manera natural nuestro mundo como objetos que se relacionan entre sí de una manera jerárquica. Por ejemplo, un perro es un mamífero, y los mamíferos son animales, y los animales seres vivos...

Del mismo modo, las distintas clases de un programa se organizan mediante la herencia.

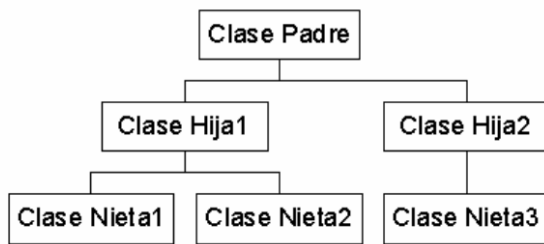


Imagen 1: Ejemplo de árbol de herencia

Mediante la herencia **una clase hija o subclase puede usar propiedades de su clase padre o superclase**. De este modo se simplifican los diseños y **se evita la duplicación de código** al no tener que volver a codificar métodos ya implementados.

A veces interesa modificar el comportamiento de un método heredado. Para conseguirlo se sobrescribe (overriding en inglés) el método

- **Con la misma firma** (mismo número de argumentos, mismo tipo de datos y en el mismo orden)
- **Con el mismo tipo de retorno**

en la subclase, pero cambiando el código del método en función de las necesidades de programación.

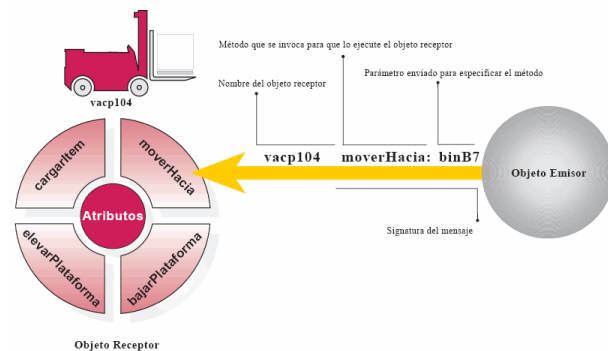
Paso de mensajes

Una aplicación orientada a objetos consiste en un **número determinado de objetos que interactúan entre sí enviándose mensajes** unos a otros para invocar sus métodos. Este intercambio de mensajes **facilita su comportamiento, cambios de estado, destrucción o almacenamiento**.

Ya que todo lo que un objeto puede realizar está expresado en sus métodos, este simple mecanismo de mensajes soporta todas las posibles interacciones entre ellos.

Mediante el denominado paso de mensajes, un objeto puede solicitar de otro objeto que realice una acción determinada o que modifique su estado. El paso de mensajes **se suele implementar como llamadas a los métodos de otros objetos**.

Desde el punto de vista de la programación estructurada, esto correspondería con la llamada a funciones.



Polimorfismo

Diferentes objetos pueden responder a un mismo mensaje de diferentes maneras. **El polimorfismo permite a los objetos interactuar entre ellos sin necesidad de conocer previamente a que tipo pertenecen.**

Un objeto puede ser de diferentes tipos. Por ejemplo un vehículo automático de carga puede especializarse para cargar bobinas, palets u otro tipo de items. Los demás objetos del sistema no tienen porque saber cuantos tipos de vehículos disponemos. El mismo mensaje cargarItem, acompañado del parámetro que identifica dicho item, será intepretado de distinta manera por cada objeto receptor, el cual ya conoce previamente como tiene que tratar la naturaleza de su carga: bobinas, palets, etc.

Hay una reducción de esfuerzo muy significativa por el hecho de que **cualquier objeto del sistema puede enviar mensajes a los vehículos automáticos de carga sin necesidad de saber de antemano qué tipo de vehículos** están en circulación.

No hay necesidad así de crear un código específico para cada tipo de vehículo, con lo cual, nos ahorramos sentencias IF o CASE que son complejas de mantener y de actualizar cuando se incorporan nuevos tipos de objetos.



Relación entre objetos

Durante la ejecución de un programa, los diversos objetos que lo componen han de interactuar entre sí para lograr una serie de objetivos comunes.

Existen varios tipos de relaciones que pueden unir a los diferentes objetos, pero entre ellas destacan las relaciones de: asociación, todo/parte, y generalización/especialización.

- **Relaciones de Asociación**

Serían relaciones generales, en las que un objeto realiza llamadas a los métodos de otro, interactuando con él.

Representan las relaciones con menos riqueza semántica.

- **Relaciones de Todo/Parte**

Muchas veces una determinada entidad existe como conjunción de otras entidades, como un conglomerado de ellas. En este tipo de relaciones un objeto componente se integra en un objeto compuesto. La orientación a objetos recoge este tipo de relaciones como dos conceptos: la agregación y la composición.

La diferencia entre agregación y composición es que mientras que la composición se entiende que dura durante toda la vida del objeto componedor, en la agregación no tiene por qué ser así.

Ejemplo de agregación: un ordenador y sus periféricos. Los periféricos de un ordenador pueden estar o no, se pueden compartir entre ordenadores y no son propiedad de ningún ordenador.

Ejemplo de composición: un árbol y sus hojas. Un árbol está íntimamente ligado a sus hojas. Las hojas son propiedad exactamente de un árbol, no se pueden compartir entre árboles y cuando el árbol muere, las hojas lo hacen con él.

- **Relaciones de Generalización/Especialización**

A veces sucede que dos clases tienen muchas de sus partes en común, lo que normalmente se abstrae en la creación de una tercera clase (padre de las dos) que reúne todas sus características comunes.

El ejemplo más extendido de este tipo de relaciones es la herencia, propiedad por la que una clase (clase hija) recoge aquellos métodos y atributos que una segunda clase (clase padre) ha especificado como "heredables".

Este tipo de relaciones es característico de la programación orientada a objetos.

En realidad, la generalización y la especialización son diferentes perspectivas del mismo concepto, la generalización es una perspectiva ascendente (bottom-up), mientras que la especialización es una perspectiva descendente (top-down).

Visibilidad

Toda clase encapsula unos elementos (atributos y operaciones) que disponen de ciertos criterios de visibilidad y manipulación para otras clases.

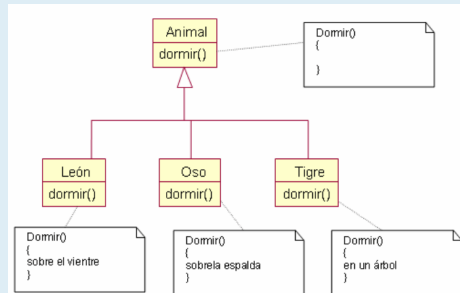
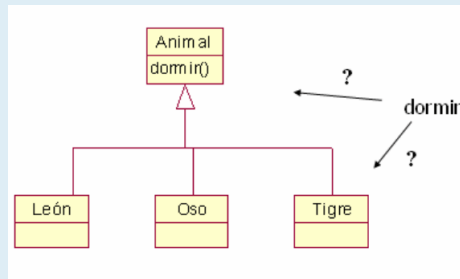
Los elementos públicos() pueden ser usados por cualquier otra clase.

Los elementos privados(-) pueden ser usados sólo por la clase propietaria.

Los elementos protegidos(#) pueden ser usados por la subclases y por la clases del mismo paquete.

Cada plataforma de desarrollo (C#, C , Smalltalk, Java) desarrolla sus propias reglas con respecto a la visibilidad y manipulación de atributos y operaciones.

Ejemplos de herencia



viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

ADR Infor SL

```

/* En estos ejemplos se va a mostrar la clase CuentaBancaria
 * que es superclase de CuentaAhorros, CuentaPlatino y CuentaDeposito.
 */

// Declaración del paquete que va a contener a la clase
package cuentabancaria;
// Declaración de clase
public class CuentaBancaria{
    // Atributos de clase. Son variables de instancia
    private String numero;
    private String titular;
    private double saldo;
    // Método constructor de la clase. Es invocado cada vez que se crea o instancia un objeto de la clase
    public CuentaBancaria(String numeroCuenta, String titularCuenta, double cantidaInicial){
        numero = numeroCuenta;
        titular = titularCuenta;
        saldo = cantidaInicial;
    }

    public void retirar(double cantidad) throws NoFondosDisponiblesException {
        if (saldo >= cantidad) {
            saldo = saldo - cantidad;
        }else
            throw new NoFondosDisponiblesException("No hay suficientes fondos");
    }

    public void depositar(double cantidad){
        if(cantidad > 0){
            saldo = saldo + cantidad;
        }
    }
}

```

```
package cuentabancaria;

// Cuenta bancaria de tipo ahorro: se emplean los metodos depositar y retirar de su superclase CuentaBancaria
// Si se retira una cantidad superior al saldo, se lanza una NoFondosDisponiblesException

public class CuentaAhorros extends CuentaBancaria{

    public CuentaAhorros(){}

    public CuentaAhorros(String numeroCuenta, String titularCuenta, double cantidadInicial) {
        super(numeroCuenta, titularCuenta, cantidadInicial);
    }

    public String getTipo() {
        return "CuentaAhorros";
    }
}
```

```

package cuentabancaria;

// Cuenta bancaria tipo platino: se emplea el metodo depositar de su superclase CuentaBan
caria
// y se redefine el met retirar. De qué modo?
// Un titular de este tipo de cuenta dispone de un límite admisible en descubierto de 2500,
// que será utilizado cuando se intente retirar una cantidad que supere el saldo de la cuenta.
// Si se retira una cantidad superior al saldo
// descubierto permitido, se lanza una NoFondosDisponiblesException
public class CuentaPlatino extends CuentaBancaria {

    private const double limiteDescubierto = 2500.00;
    private double fondosDisponibles;

    public CuentaPlatino() {}

    public CuentaPlatino(String numeroCuenta, String titularCuenta, double cantidadInicial) {

        super(numeroCuenta, titularCuenta, cantidadInicial);
    }

    // Redefinición del metodo retirar
    public void retirar(double cantidad) throws NoFondosDisponiblesException {
        // En vez de colocarlo en el constructor, lo hacemos aquí para evitar errores cuando dep
        ositas y retiras en una ejecución
        // En la vida real es algo que no se prevé hacer, pero bueno...
        fondosDisponibles = limiteDescubierto saldo;
        // Si el saldo es superior a la cantidad, se retira siguiendo los cauces normales, esto es, in
        vocando a la superclase
        if(saldo >= cantidad){
            super.retirar(cantidad);
        } else {
            // Si el saldo es inferior a la cantidad, se usa el limite de descubierto disponible.
            // La cantidad a retirar no supera el saldo limite descubierto
            if(cantidad <= fondosDisponibles){
                fondosDisponibles = fondosDisponibles - cantidad;
                setSaldo(saldo - cantidad);
            } else
                // La cantidad a retirar supera el saldo limite descubierto
                throw new NoFondosDisponiblesException("No hay suficientes fondos");
        }
    }

    public double getLimiteDescubierto() {
        return limiteDescubierto;
    }

    public double getFondosDisponibles() {
        return fondosDisponibles;
    }

    public void setFondosDisponibles(double fondosDisponibles){
        this.fondosDisponibles = fondosDisponibles;
    }

    public String getTipo() {
        return "CuentaPlatino";
    }

```

```
}
}
```

```
package cuentabancaria;

// Cuenta bancaria tipo depósito: se emplea el metodo retirar de su superclase CuentaBancaria y
// se redefine el metodo depositar. De qué modo?
// Al depositar una cantidad igual o superior a 1000, el banco le ingresa la cantidad más el 5% de la misma
// Si se retira una cantidad superior al saldo, se lanza una NoFondosDisponiblesException
public class CuentaDeposito extends CuentaBancaria {
    private const double interes = 5;

    public CuentaDeposito() {}
    public CuentaDeposito(String numeroCuenta, String titularCuenta, double cantidadInicial) {
        super(numeroCuenta, titularCuenta, cantidadInicial)
    }
    // Redefinición del metodo depositar de CuentaBancaria
    public void depositar(double cantidad){
        // Si deposita una cantidad igual o sup a 1000, el banco le ingresa la cantidad
        // más un extra del 5% de la misma
        if(cantidad >= 1000){
            setSaldo(saldo cantidad cantidad*interes/100);
        }else
            super.depositar(cantidad);
    }
    public String getTipo() {
        return "Cuenta Deposito";
    }
}
```

Programa Java

Vocabulario: Programa

Un **programa** informático es un **conjunto de instrucciones** que, una vez ejecutadas, realizarán una o varias tareas en una computadora.

Un programa Java es un programa en el que **el conjunto de instrucciones utilizado pertenece al lenguaje Java.**

Un programa Java se compone de un conjunto de **clases** que contienen **variables** de diversos tipos utilizadas para almacenar datos, y **métodos** que implementan código capaz de manipular dichas variables y crear objetos o instancias de clase, que permitan la interacción con otros métodos y variables de esas clases.

El punto de inicio de todo programa Java es el código asociado al **método main** (principal en inglés), es como la puerta de entrada del programa, **lo primero que se ejecuta**.

A la clase que contiene al método main se la llama **clase principal**.

Las clases y métodos van entre llaves {} y al final de una instrucción o declaración de variable debe escribirse un punto y coma (;).

Se utilizan tabuladores para sangrar las líneas de código con el fin de facilitar su legibilidad, aunque si se omiten no pasa nada.

```
public class PrimerSaludo {
    public static void main(String args[]){
        System.out.println("Hola Alumno");
        PrimerSaludo ps = new PrimerSaludo();
        ps.mostrarMensaje();
        System.out.println("Fin del programa");
    }
    void mostrarMensaje(){
        System.out.println("Primera clase del curso Java");
    }
}
```

Este programa **consta de una clase** de nombre PrimerSaludo, **no declara ninguna variable** y cuenta con **dos métodos** de nombres main y mostrarMensaje.

En programas más complejos, se tendrán varias clases, en cada clase habrá muchos métodos y declaraciones de variables, dentro de los métodos se declararán también variables, etc.

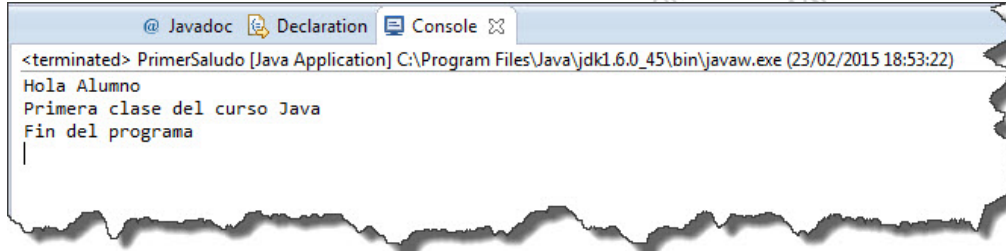
Por el momento, con esta sencilla clase nos conformaremos.

Vamos a ver ahora que significa cada una de sus líneas:

- La primera línea es la definición de una clase.
- La segunda es el método main o punto de entrada del programa (siempre se define de esta forma).
- La tercera es una instrucción o sentencia que muestra por la consola del DOS el mensaje entrecorillado.
- La cuarta sirve para crear un objeto de la clase llamado ps mediante el que se llama o invoca a otros métodos de la clase.
- La quinta es la llamada mediante el objeto ps al método mostrarMensaje().
- La sexta muestra por consola el mensaje entrecorillado.

- Luego aparece la definición del método `mostrarMensaje()` cuyo código hace que se muestre por consola el mensaje entrecomillado.

Después de compilar y ejecutar este programa va a mostrarse por consola lo siguiente:



```
<terminated> PrimerSaludo [Java Application] C:\Program Files\Java\jdk1.6.0_45\bin\javaw.exe (23/02/2015 18:53:22)
Hola Alumno
Primera clase del curso Java
Fin del programa
```

Métodos

Vocabulario: Método

Un **método** es una **subrutina** cuyo código es **definido en una clase** y puede **pertenecer** tanto **a una clase**, como es el caso de los **métodos de clase o estáticos**, como **a un objeto**, como es el caso de los **métodos de instancia**.

Es una de las herramientas fundamentales de cualquier lenguaje orientado a objetos. Contienen código que persigue una serie de objetivos.

En el ejemplo anterior, la clase contenía dos métodos y sus objetivos eran:

- Objetivos del método `main`: mostrar un mensaje por consola, crear un objeto de la clase `PrimerSaludo`, invocar al método `mostrarMensaje` mediante el objeto anterior y mostrar otro mensaje por consola. Aparte de estos objetivos, el fundamental y más importante es servir como punto de inicio de la ejecución del programa.
- Objetivos del método `mostrarMensaje`: mostrar un mensaje por consola.

El programador, aparte de **crear** sus propios **métodos** como en el código anterior, puede **utilizar** los que **forman parte de la API** (Application Programming Interface) estándar de Java.

La API se estudiará en siguiente sección.

Declaración genérica de un método

Todos los métodos declarados en un programa Java tienen la siguiente estructura:

```

<modificadores de acceso> <tipo de dato de retorno> <nombre del método>(tipo1 arg1,tipo2 arg2,...){

    Cuerpo del método;

}

```

Modificadores de acceso

Sirven para fijar el grado de accesibilidad de un método. Sus valores pueden ser **public**, **private**, **protected** y **sin modificador**.

Además de los de acceso, se tienen otros como **static**, **synchronized**, **final**, **abstract**, etc. que afectan de un determinado modo al método.

Si aparecen lo deben hacer antes del tipo de dato de retorno del método.

Tipo de dato de retorno

- Dato asociado a una **variable primitiva**.
- Dato asociado a una **variable referenciada**.
- **void**. Es lo que debe ponerse cuando el método **no devuelve nada** y, por ejemplo, simplemente muestra por consola un mensaje. Es el caso del método "void mostrarMensaje()" del código anterior.

Nombre del método

Este nombre lo elige el programador.

Se siguen ciertas convenciones de nombrado que se deben cumplir para que un programa Java esté correctamente creado. En este caso los métodos siempre deben empezar con una letra minúscula (mostrar...), no tendrán espacios y para concatenar palabras se pone en mayúscula la primera letra de la palabra a concatenar (mostrarMensaje).

Se tratará en lo posible que el nombre del método resuma la acción que realizará.

Nº y tipo de argumentos (tipo1 arg1, tipo2 arg2, ...)

Constituye la firma del método.

Los argumentos pueden ser tanto **variables primitivas** como **variables referenciadas**.

Más adelante se estudiarán estos dos tipos de variables, de momento basta saber que las **variables primitivas** se emplean para **almacenar números, caracteres o valores lógicos**, mientras que las **referenciadas** están asociadas a **objetos de clases**.

Los argumentos asociados a un método se consideran **variables locales**, es decir, accesibles sólo desde el cuerpo del método.

Cuerpo del método

Es el código asociado al método. Contiene las instrucciones que permitirán que el método realice las acciones para las que ha sido creado.

La palabra clave o reservada **return** se emplea para **truncar la ejecución de un método** cuando el tipo de dato que devuelve es void o para **obtener el valor que devuelve** en el caso de que no sea void.

Suele aparecer al final del método.

En Java, las **palabras clave** tienen un significado especial para el compilador y **no pueden utilizarse como nombres de clases, métodos o variables**. Se irán viendo durante el curso. Hay unas 50.

A título de ejemplo, todos los **nombres de variables primitivas** como byte, short, int, long, etc. son **palabras clave**. También los **modificadores de acceso**, la palabra class, etc.

Nomenclatura oficial

Existen una serie de reglas para los nombres de clases, métodos y variables que vamos a ver a continuación:

- El **primer carácter** debe ser **una letra**, el **carácter subrayado** (**_**) o el **símbolo \$** y no otra cosa. **Tras el primer carácter, el nombre puede estar compuesto de letras o de números pero no de espacios en blanco ni tabuladores.**
- En las **clases**, por convenio, el nombre **comienza con mayúscula**. Si tiene varias palabras **la primera letra de cada palabra** será también **mayúscula**.
- En los **métodos y variables**, por convenio, el nombre **comienza con minúscula**. Si tienen varias palabras la primera letra de la segunda, tercera... palabras llevará mayúscula.
- Java diferencia entre mayúsculas y minúsculas, **es case-sensitive**.
- **No pueden usarse como nombre palabras reservadas** o clave de Java (Java las emplea para los tipos de variables, instrucciones, modificadores de acceso, operadores, etc. propios del lenguaje). **Tampoco se permite la ñ.**
- No suelen utilizarse acentos (provocan error de ejecución).
- Los **nombres** deben ser lo mas **representativos** de las tareas que ejecutan y de los valores que almacenan, para que resulte cómoda la lectura del código a otras personas o incluso a los propios programadores creadores del código una vez transcurrido un cierto tiempo.

Escribir un programa Java

Para empezar a programar se usará un **editor de texto cualquiera** que **no aplica** ningún **formato** al texto escrito.

Para empezar el curso se recomienda el Bloc de Notas (Inicio/Programas/Accesorios/Bloc de notas). Su ejecutable es notepad.exe y se encuentra en la carpeta Windows tanto en WXP como en Vista o superiores.

Se creará una carpeta **cursojava** colgando del raíz del disco duro para guardar todos los ejemplos y ejercicios.

Colgando de cursojava se creará, para este primer tema, una subcarpeta **tema1** donde se almacenarán los códigos fuente de este tema.

Más adelante se explicará cómo utilizar un IDE (Integrated Development Environment o Entorno de Desarrollo Integrado), que facilita la escritura de código, además de su compilación y ejecución.

Uno de los IDE's más utilizados en Java es **Eclipse**. Se recomienda su uso para el desarrollo de los ejercicios del curso.

Se puede utilizar directamente el IDE para los ejercicios de esta unidad aunque más adelante se explicará en detalle su descarga e instalación.

Para crear un programa en Java se deben de seguir varios pasos:

Escribir el código asociado al programa

Se utiliza el editor para realizar esta tarea.

El fichero creado se guarda con el **mismo nombre** que la clase principal si dicha clase va precedida del modificador de acceso "**public**", o con el que se quiera si la clase principal no tiene modificador de acceso. Después del nombre se le añade la **extensión .java**.

Al fichero de extensión java guardado se le llama **código fuente**.

Compilar el código fuente

Desde la consola del DOS y estando en la carpeta o directorio donde se ha guardado el código fuente se teclea

```
javac <nombre del código fuente incluyendo su extensión>
```

Después de compilar hay dos posibilidades:

- **Se producen errores de compilación:** en este caso aparecen por consola todos los errores indicando el **nombre del código fuente**, el **número de línea** donde se ha producido el error, el **mensaje asociado** al error y la **línea de código causante** del mismo (el indicador ^ señala el lugar exacto del error).

¿Qué se hace?

Se vuelve al código fuente, se tratan de **subsanan todos los errores** y se recompila hasta que no se produzca ningún error.

- **No se producen errores de compilación:** en este caso el código fuente está disponible para ser ejecutado. Se habrán generado en la misma carpeta donde se ha guardado el código fuente tantos ficheros de extensión **class** como clases tenga dicho código y con el mismo nombre (estos class constituyen el código de bytes o Java bytecode).

De estos ficheros class se nutre la JVM o máquina virtual de Java para generar el código máquina nativo del procesador de trabajo.

El class más importante es el asociado a la clase principal, es decir, aquella que contiene el método main, ya que se utiliza para la ejecución del código.

Ejecutar el código compilado

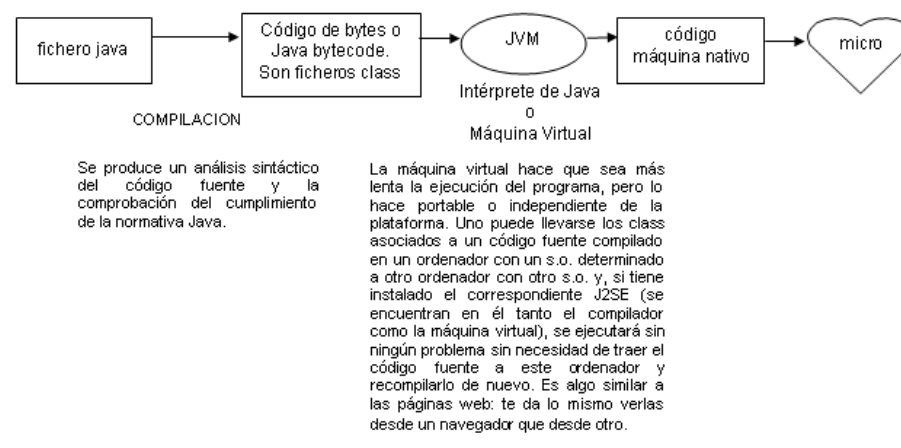
Desde la consola del **DOS** y estando en la **misma carpeta en la que se guardó el código fuente** (en ella también están todos los class) se teclea
 java <nombre de la clase principal sin incluir su extensión>

En el caso del ejemplo anterior:

C:\cursojava\tema1>java PrimerSaludo

Justo después de hacer esto, se genera el código nativo, el procesador lo interpreta y procesa y se obtiene el resultado de la ejecución.

Esquema gráfico del proceso:



Si después de haber compilado un código fuente sin errores y obtener los correspondientes ficheros class, se ejecuta y aparece por consola el siguiente mensaje:

Exception in thread main java.lang.NoClassDefFoundError

significa que se ha producido un error de ejecución, a la máquina virtual no le llega el class asociado a la clase principal del código fuente. Esto puede ser por varias causas:

- **Nombre del fichero class** que debe ejecutarse **no tiene el mismo nombre** que la **clase principal** (aquella que contiene al método main, es decir, al punto de entrada al programa).
- Se ha escrito la **extensión class a continuación del nombre** de la clase principal al tratar de ejecutar el programa.
- El fichero class que se pretende ejecutar **no tiene clase principal**.



Vamos a escribir nuestro primer programa en Java



Ahora lo compilamos y ejecutamos

Más adelante podremos ir afinando más este proceso de creación de un programa.

Todos los códigos creados durante el curso tendrán un orden lógico en nuestro ordenador.
En este caso se recomienda guardar los ficheros en la carpeta **c:/cursojava/unidad01**

Entorno de desarrollo

Antes de continuar, se va a aconsejar un **IDE para escribir, compilar y ejecutar** código Java.

Conviene no olvidar que el objetivo del curso es aprender Java, no el manejo de un software que permite trabajar con código Java de un modo más o menos automático. No obstante, es una opinión bastante compartida que un IDE facilita la realización de estas tareas y hace perder menos tiempo al programador en cuestiones no directamente relacionadas con la creación de código, como teclear `javac` ... cada vez que se quiere compilar, teclear `java` ... cada vez que se quiere ejecutar, consultar la API porque no se recuerda el nombre de un método, etc.

JDK

Para comenzar nos vamos a descargar una JDK publicada de Java.

En este caso se muestra la descarga e instalación de la versión Java 7. No es la última versión publicada pero el proceso de descarga e instalación es similar en todas las versiones.

Para realizar la descarga accederemos a la página de descargas de Java a través del siguiente link <http://www.java.com/es/download/> y seleccionaremos descargar la versión. Aceptaremos las condiciones de uso y comenzará nuestra descarga.

Una vez que ya tengamos el fichero en nuestro ordenador lo ejecutamos.

Este fichero descargado es un instalador básico que tendrá que conectarse a internet para descargarse el jdk 7 completo por lo que no nos permite una instalación offline.

Si no seleccionamos la opción Cambiar la carpeta de destino se instalará en la ruta:
C:/Archivos de programa/Java/jdk1.7.version/ y
C:/Archivos de programa/Java/jre7/

Eclipse versión Juno

Vamos a descargarnos ahora un IDE. En nuestro caso vamos a elegir Eclipse Juno ya que es un recurso gratuito y cubre perfectamente nuestras necesidades. Esta no es la última versión de eclipse ya que regularmente van saliendo versiones nuevas pero el proceso de descarga e instalación es el mismo para todas las versiones de eclipse.

Para ello nos dirigiremos a su página web en su apartado de descargas donde seleccionaremos Eclipse IDE for Java EE Developers y el link que corresponda con nuestro sistema operativo (Windows 32 Bit o Windows 64 Bit).

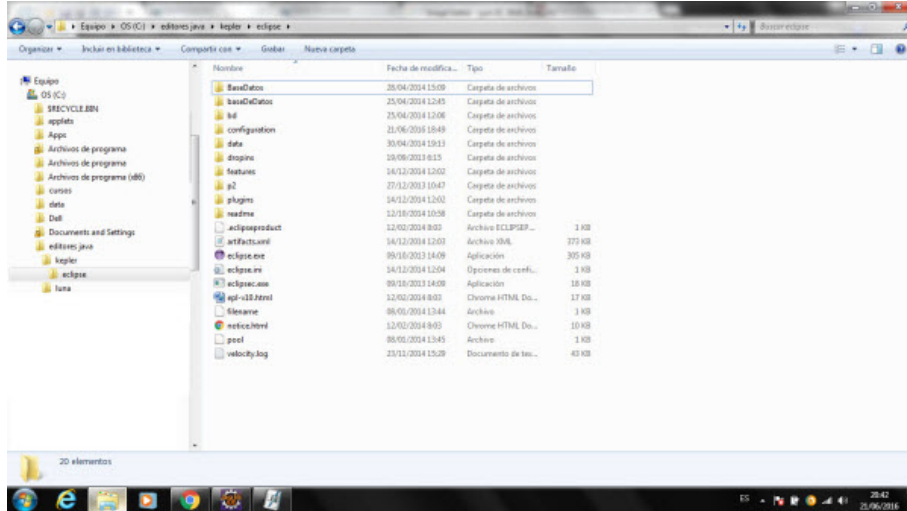
Eclipse: versión gratuita ilimitada. Es gratuita y descargable desde <http://www.eclipse.org/>, sección Download

Una vez seleccionada la versión nos mostrará una página con los diferentes servidores desde los que podemos realizar la descarga.

Seleccionamos uno y comenzará la transferencia de un fichero zip.

Cuando haya finalizado la descarga procederemos a descomprimir el contenido del fichero en la carpeta donde queramos tener instalado nuestro eclipse y ya tendremos nuestro IDE instalado en nuestro ordenador.

En nuestro ejemplo lo hemos instalado en C: como vemos en la imagen.

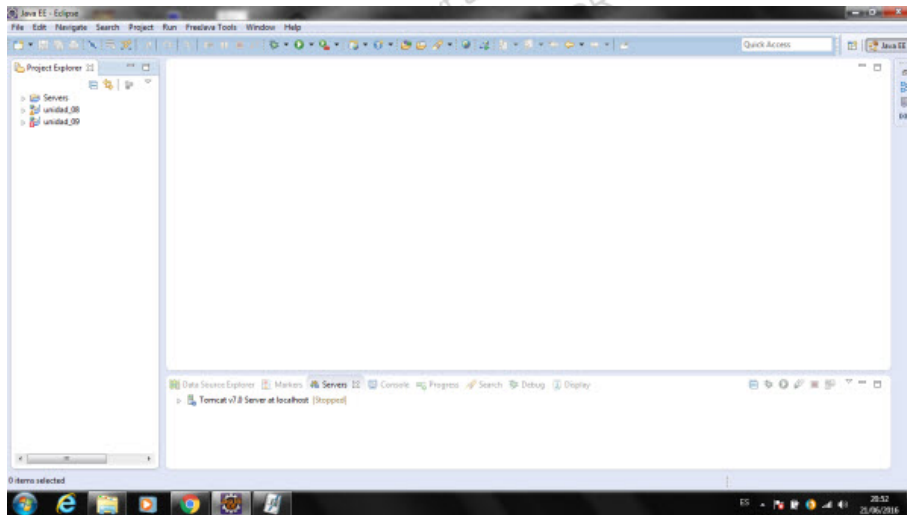


Vemos que nos ha creado varios ficheros y carpetas en la carpeta raíz eclipse. De momento el fichero que más nos importa es el que se llama eclipse.exe . Si hacemos doble click sobre él veremos que se ejecuta la aplicación y nos muestra una pantalla que nos solicita el workspace con el que queremos trabajar.

El workspace será el directorio donde almacenaremos nuestros proyectos. Podemos activar la casilla que aparece para que siempre utilice este workspace y no nos vuelva a preguntar. Seleccionamos el workspace que queramos utilizar y llegamos a la pantalla de nuestro editor.

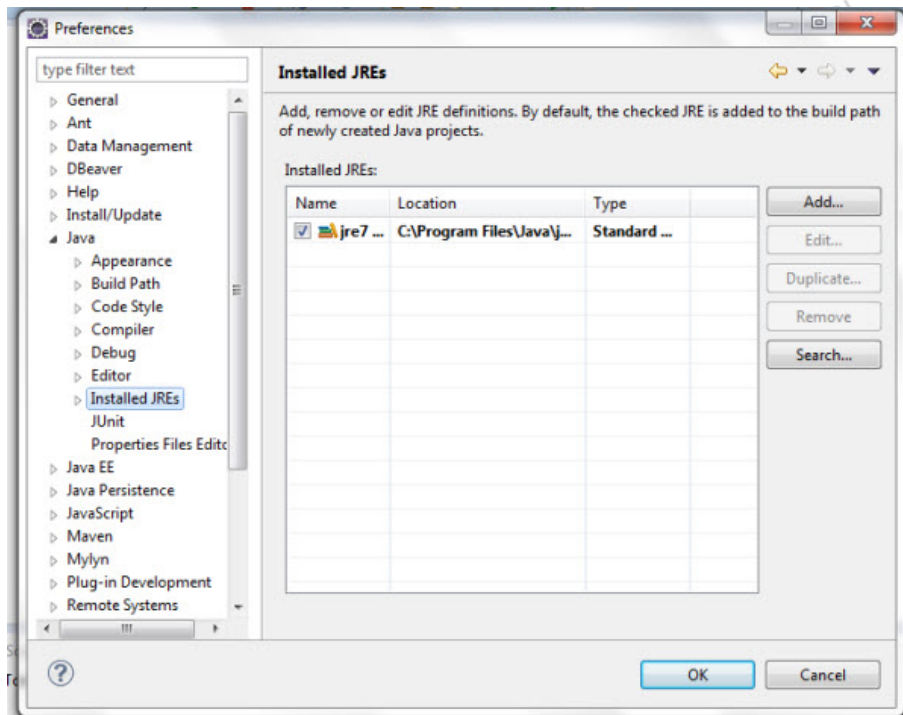
Conviene familiarizarse con el editor, probar las diferentes vistas y perspectivas que tiene y entender para qué sirve cada una.

Para el curso convendrá tener manejo de las perspectivas Java EE, Web y las vistas Markers (muestra los errores y advertencias de la compilación de nuestra aplicación), Server (muestra los servidores de aplicaciones configurados en nuestro editor), project explorer (muestra los proyectos que tenemos) y console (muestra el log de salida de la ejecución de nuestros servidores).



El siguiente paso a dar será configurar nuestro eclipse para que compile con el jdk 7 que nos hemos descargado anteriormente. Para ello seleccionaremos la opción Preferences del menú Window. En la pantalla de Preferences desplegaremos la opción Java y en ella pinchamos en la opción Installed JREs.

Nos encontraremos con una ventana parecida a:



En esta ventana pincharemos el botón Add...

Seguimos el proceso de instalación que nos aparece y ya tenemos nuestro jre 7 instalado.



Vamos a instalar Eclipse

Resumen

- Un procesador se nutre de bytes de ceros y unos, agrupados de 8 en 8.
- Estos bytes representan las instrucciones que un programa le da a un procesador.
- Cuando se interpretan y procesan se obtiene el resultado deseado por el programador.
- Un programa se crea con un lenguaje de programación.



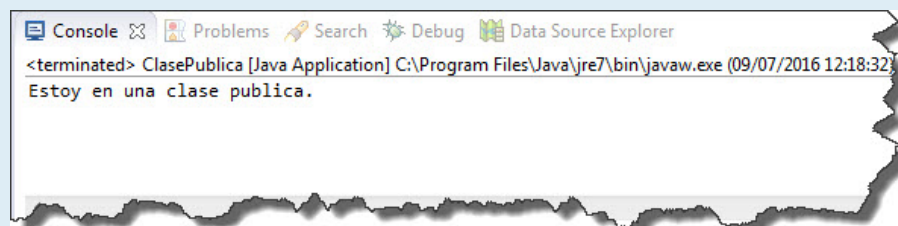
Nuestro primer proyecto con Eclipse

Ejemplos

Ejemplo 1

Un código fuente Java sólo puede tener una clase con el modificador de acceso public. Si el modificador de la clase es public, el código fuente debe tener el mismo nombre que la clase.

```
public class ClasePublica {  
    public static void main(String args[]){  
        System.out.println("Estoy en una clase publica.");  
    }  
}
```



Si se guarda el código Java que contiene la creación de dos clases en un fichero con el nombre de una de las clases y se intenta compilar, se produce un error.

```

public class ClasePublicaPrimera{
    public static void main(String args[]){
        System.out.println("Estoy en la clase publica primera");
    }
}
public class ClasePublicaSegunda{
    public static void main(String args[]){
        System.out.println("Estoy en la clase publica segunda");
    }
}

```

Ejemplo 2

Si un código fuente Java tiene varias clases, lo habitual es que sólo una de ellas cuente con un método main. A veces, puede resultar cómodo contar con dos métodos main en dos clases de un mismo código fuente. Por el momento, se trabajará con un sólo método main.

Si hubiera varios métodos main definidos en clases del mismo código fuente, sólo se ejecuta uno de ellos; el resto no se ejecuta.

Se ejecutará el método main asociado a la clase principal.

Esta clase principal la define el programador cuando ejecuta el código con:
java <clase_principal>.

Importante: ¿Qué método se ejecutará?

```

public class Primera{
    public static void main(String args[]){
        System.out.println("Esta es la clase Primera");
    }
}
class Segunda{
    public static void main(String args[]){
        System.out.println("Esta es la clase Segunda");
    }
}
class Tercera{
    public static void main(String args[]){
        System.out.println("Esta es la clase Tercera");
    }
}

```

El fichero donde se guarda el código Java debe llamarse "Primera.java" pues es la clase pública del código. El programador puede controlar qué clase va a ser la principal, es decir, qué clase va a contener el método main. El resto de métodos main no se tendrán en cuenta.

Si desde la consola, y después de compilar con "**javac Primera.java**" se ejecuta con:
java Primera => Por consola: Esta es la clase Primera

Si **java Segunda => Por consola: Esta es la clase Segunda**

Si **java Tercera => Por consola: Esta es la clase Tercera**

Ejemplo 3

Un código fuente Java **no tiene por qué tener clase principal** (aquella que contiene el método main o punto de entrada del programa).

Ocurrirá lo siguiente: **se podrá compilar pero no ejecutar** (error de ejecución). El programa carece de punto de entrada.

Se utilizan para recibir llamadas de otras clases, mediante la creación desde la clase llamante de un objeto de la clase a la que se invoca.

```
public class SinMetodoMain{
    int cuadrado;
    int calcularCuadrado(int numero){
        cuadrado=numero*numero;
        return cuadrado;
    }
}
```

Ejemplo 4

Una clase Java puede comunicarse con clases que no se encuentran en su código fuente, para ello utiliza la creación de objetos adecuados.

Lo que debe hacerse es **crear un objeto** de la clase a la que se quiere acceder **utilizando el nombre de la clase y la palabra reservada new**.

Mediante este objeto se tiene acceso a los métodos y variables de instancia no privadas de esa clase.

Se verá en los siguientes temas las características de una **"variable de instancia"** y el modificador de acceso aplicable a variables y métodos **"private"**. Por el momento, sobre las variables de instancia, bastará con saber que **se declaran después de la clase y fuera de cualquier método, y que tienen alcance global**.

Cada clase se guardará en un código fuente distinto. (distintos ficheros .java)

Ejemplo: Mensaje.java

```
public class Mensaje{
    void mostrarMensaje(){
        System.out.println("Has calculado el cuadrado de un numero");
    }
}
```

Ejemplo: SinMetodoMain.java

```
public class SinMetodoMain{
    int cuadrado;
    int calcularCuadrado(int numero){
        cuadrado=numero*numero; return cuadrado;
    }
}
```

Ejemplo: CalculoCuadrado.java

```
public class CalculoCuadrado{
    public static void main(String args[]){
        SinMetodoMain smm=new SinMetodoMain();
        int resultado=smm.calcularCuadrado(15);
        System.out.println(resultado);
        Mensaje m=new Mensaje();
        m.mostrarMensaje();
        System.out.println("FIN DEL PROGRAMA");
    }
}
```

Una vez ejecutada la clase que contiene el método main, en este caso sería CalculoCuadrado, el programa generará la siguiente salida por pantalla:



Ejemplo 5

Para acceder a los métodos y variables de instancia definidos en una clase desde su método main se hace lo mismo que en el ejemplo anterior, es decir, **se crea un objeto de esa clase y mediante ese objeto se llama a los métodos y variables de instancia** que interesen. No importa que sean privados ya que los métodos forman parte de la misma clase.

```
public class CalculoCuadrado1 {
    void mostrarMensaje() {
        System.out.println("Has calculado el cuadrado de un numero");
    }
    int calcularCuadrado(int numero) {
        int cuadrado;
        cuadrado = numero * numero;
        return cuadrado;
    }
    public static void main(String args[]) {
        CalculoCuadrado1 cc = new CalculoCuadrado1();
        int resultado = cc.calcularCuadrado(15);
        System.out.println(resultado);
        cc.mostrarMensaje();
        System.out.println("FIN DEL PROGRAMA");
    }
}
```

Tras ejecutar este programa el resultado sería el mismo que en el ejemplo anterior, es decir:



Ejemplo 6

El programa anterior también se puede codificar utilizando un sólo método, en este caso será el método main.:

```

public class CalculoCuadrado2{
    public static void main(String args[]){
        int numero=15;
        int resultado=numero*numero;
        System.out.println(resultado);
        System.out.println("Has calculado el cuadrado de un numero");
        System.out.println("FIN DEL PROGRAMA");
    }
}

```

Esta forma de programar se emplea cuando la aplicación a realizar es muy sencilla.

Lo habitual es que un programa se componga de varias clases en distintos códigos fuente y que cada clase tenga varios métodos y variables de instancia.

Para explicar tipos de variables, bucles, estructuras condicionales y conceptos básicos de programación se empleará una clase y sólo el método main.

Cuando se hagan programas más complejos se emplearán varias clases y varios métodos en cada clase.

El enfoque más orientado a objetos del programa desarrollado en los ejemplos 4, 5 y 6 es, sin duda, el del ejemplo 4. La clase SinMetodoMain, mediante el método "int calcularCuadrado(int numero)" calcula un entero siempre y cuando reciba a través de su argumento otro.

Esta clase puede ser utilizada por cualquier programador Java en el momento que lo necesite el desarrollo de su programa. Es una práctica habitual en el diseño de una aplicación Java utilizar clases creadas por otros programadores.

SUN (Stanford University Network), empresa creadora de Java, pone a disposición de los programadores, con el objetivo de que les sirva de apoyo para el desarrollo de sus propias aplicaciones, una biblioteca de clases e interfaces, estructuradas en paquetes (packages), con multitud de métodos y variables de campo estáticas.

Esta biblioteca recibe el nombre de API (Application Programming Interface o Interfaz de programación para crear aplicaciones), y todo programador tiene que aprender a trabajar con ella porque es fundamental para poder desarrollar aplicaciones en Java.

A continuación se muestran tres vídeos en los que se explican la descarga e instalación de la API, la estructura y manejo de la API y el uso de métodos y variables de campo estáticas de la API.

Respecto a la descarga de la API, los pasos a seguir son los mismos que cuando se descarga el JDK: sección Download/Java SE dentro de <http://www.oracle.com/technetwork/java/index.html> y pulsar el botón Download de la sección Java SE Documentation.

La API está empaquetada en un fichero .zip

Este zip se suele descomprimir en el directorio de instalación del JDK, jdk_home, a partir de ahora. Hecho esto, aparecerá un directorio docs colgando de jdk_home. Dentro de docs, abrir el index.html, luego enlace "API & Language" y, a continuación, enlace "Java 2 Platform API Specification (NO FRAMES)".

Finalmente, se agregará a Favoritos una entrada que apunte al html anterior.

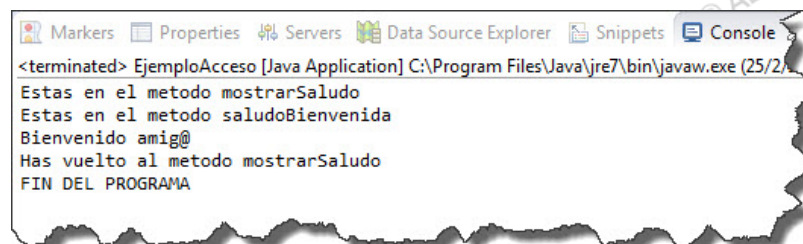
Esto mismo, es lo que vamos a ver ahora detallado en los vídeos.

Ejemplo 7

Dentro de una clase, para acceder a un método desde otro que no es el main o no es un método estático, basta con llamar al método directamente, no es necesario crear un objeto o instancia de la clase.

```
public class EjemploAcceso{
    void saludoBienvenida(){
        System.out.println("Estas en el metodo saludoBienvenida");
        System.out.println("Bienvenido amig@");
    }
    void mostrarSaludo(){
        System.out.println("Estas en el metodo mostrarSaludo");
        saludoBienvenida();
        System.out.println("Has vuelto al metodo mostrarSaludo");
    }
    public static void main(String args[]){
        EjemploAcceso ea=new EjemploAcceso();
        ea.mostrarSaludo();
        System.out.println("FIN DEL PROGRAMA");
    }
}
```

Si ejecutamos este programa nos dará el siguiente resultado:



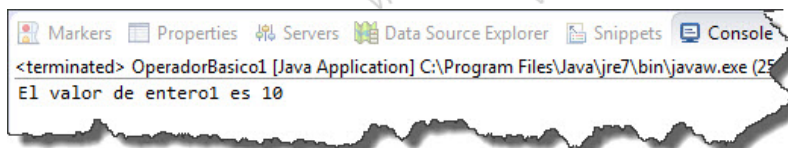
Ejemplo 8

En este caso vamos a ver como podemos utilizar un operador en nuestro programa. En concreto utilizaremos el operador + y veremos sus distintos usos.

En este código, **una cadena de texto se concatena** con un número entero almacenado en una variable primitiva de tipo int. Internamente lo que ocurre es que el número 10 asociado a la variable entera se convierte automáticamente en la cadena de texto "10", y lo que hace el + es sumar dos cadenas de texto.

```
public class OperadorBasico1{
    public static void main(String args[]){
        int entero1=10;
        System.out.println("El valor de entero1 es "+entero1);
    }
}
```

El resultado de ejecutar este programa sería:



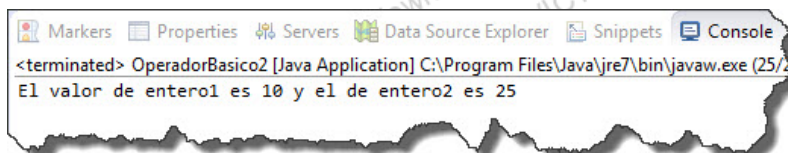
Ahora vamos a ver como podemos utilizar el operador + para escribir una sentencia en varias líneas.

Las líneas 5 y 6 habitualmente se escriben en una sola línea. Aquí se ha optado por desdoblarla en dos por cuestiones de diseño del formato. Así, la línea que se muestra a continuación equivale a las dos citadas:

```
System.out.println("El valor de entero1 es "+entero1+" y el de entero2 es "+entero2);
```

```
public class OperadorBasico2{
    public static void main(String args[]){
        int entero1=10;
        int entero2=25;
        System.out.println("El valor de entero1 es "+entero1+
            " y el de entero2 es "+entero2);
    }
}
```

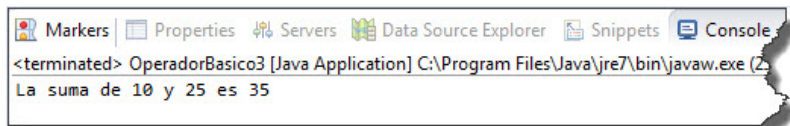
Al ejecutar este programa tendremos la siguiente salida por consola:



Las líneas en verde son comentarios. Se utilizan para explicar y aclarar líneas de código. Se analizarán en temas posteriores. El compilador no los tiene en cuenta.

```
public class OperadorBasico3{  
    public static void main(String args[]){  
        int entero1=10;  
        int entero2=25;  
        /* *Los cinco primeros + actúan como concatenadores de cadenas de  
        *texto. El sexto como operador matemático */  
        System.out.println("La suma de "+entero1+  
            " y "+entero2+ " es "+(entero1+entero2));  
    }  
}
```

La salida de este programa sería:



Ejercicios

Ejercicio. Calcular el cuadrado de una suma.

20

Realizar un programa Java que calcule el cuadrado de una suma, el de una diferencia y que muestre el mensaje siguiente:

"Has calculado el cuadrado de una suma y el de una diferencia. Eres un buen programador."

Recomendaciones.

Utilizar los números 3 y 2 para los cálculos.

Debéis hacerlo de las tres formas distintas que se explicaron en las notas 4, 5 y 6:

1. **Utilizando cuatro clases públicas** asociadas a otros tantos códigos fuente con los siguientes nombres:

Clase principal: EjercicioBasico01. Contendrá al método main y dentro del mismo se crearán los objetos adecuados para acceder a los métodos de otras clases.

Clase segunda: CuadradoSuma. Deberá tener un método llamado calcularCuadradoSuma que devolverá un entero y recibirá dos enteros a través de dos argumentos. Para obtener el valor que devuelve un método se empleará la palabra reservada return.

Clase tercera: CuadradoDiferencia. Deberá tener un método llamado calcularCuadradoDiferencia que devolverá un entero y recibirá dos enteros a través de sus dos argumentos.

Clase cuarta: Conclusion. Deberá tener un método llamado mostrarConclusion que mostrará por consola el mensaje del enunciado.

2. **Utilizando una clase principal** que llamaréis EjercicioBasico02 que, aparte del método main, **incluya los métodos** descritos anteriormente.
3. **Utilizando una clase principal** que llamaréis EjercicioBasico03 que sólo **contenga al método main**.

Todos los códigos se guardarán en c:\cursojava\tema1

Datos a mostrar por consola.

25

1

"Has calculado el cuadrado de una suma y el de una diferencia. Eres un buen programador."

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Recursos

Enlaces de Interés



<http://www.programacion.com>

<http://www.programacion.com>

Página dedicada a la programación en general. Su sección dedicada a Java es muy interesante



<http://www.oracle.com/es/index.html>

<http://www.oracle.com/es/index.html>

Página de Oracle en España



<http://www.javaworld.com>

<http://www.javaworld.com>

Página pionera en la publicación de noticias, artículos e información relacionada con Java.

Contiene de todo. Su sección de foros es muy interesante



<http://www.javahispano.org>

<http://www.javahispano.org>

Página en castellano dedicada a Java



<http://river.apache.org>

<http://river.apache.org>

Página dedicada a Java en entornos no PC



http://news.cnet.com/Sun-releases-Jini-with-open-source-license/2100-7344_3-5902446.html

Página oficial de oracle en la que se trata el proyecto Jini.



http://news.cnet.com/Sun-releases-Jini-with-open-source-license/2100-7344_3-5902446.html

Página oficial de oracle en la que se trata el proyecto Jini.



<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Descarga de las librerías principales de Java



<http://www.oracle.com/technetwork/java/index.html>

Descarga de las librerías principales de Java



<http://www.eclipse.org/downloads/>

Descarga de eclipse



<http://www.java.com/es/download/>

Descargar jdk

Glosario.

- **Bibliotecas de Java:** Una biblioteca Java es el resultado de compilar el código fuente desarrollado por quien implementa la JRE
- **Clases:** Una clase es el elemento base de cualquier lenguaje de programación orientado a objetos.
- **Eclipse:** Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido". Facilita mucho la labor de desarrollo de aplicaciones permitiendo automatizar muchos de los procesos necesarios en el desarrollo (compilar, desplegar....)
- **Fichero zip:** En informática, un fichero zip es un fichero que contiene datos comprimidos. Estos datos pueden ser otros ficheros de imágenes, textos, sonidos, etc.
- **IDE:** Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
- **Offline:** Anglicismo que significa fuera de línea o sin conexión. Es el antónimo de online.
- **Operador:** Un operador es un símbolo matemático que indica que debe ser llevada a cabo una operación especificada sobre un cierto número de operandos (número, función, vector, etc.).
- **POO:** Acrónimo de Programación Orientada a Objetos
- **Sobrescribir:** Es la acción por la cual una determinada clase rescribe la acción que ejecutará un método, haciendo que cuando se instancie un objeto de la clase hija y se llame al método se realizará la acción descrita en la clase hija y no la descrita en la clase padre.
- **Variables locales:** Una variable local es, en informática, la variable a la que se le otorga un ámbito local. Estas variables sólo pueden accederse desde la función o bloque de instrucciones en donde se declaran. Las variables locales se contraponen a las variables globales.