Lanbide
Euskal Enplegu Zerbitzua
Servicio Vasco de Empleo

Euskadi, auzolana, bien común
EUSKO JAURLARITZA
GOBIERNO VASCO
ENPLEGUKO ETA GIZARTE POLITIKETAKO SAILA
DEPARTAMENTO DE EMPLEO Y POLITICAS SOCIALES

MANUAL

JAVA

Ipartek, Servicios Informáticos
Mazustegi, 9
48006, Bilbao
Tel. 94 432 92 88
www.formacion.ipartek.com
formacion@ipartek.com

JAVA SE

INDICE

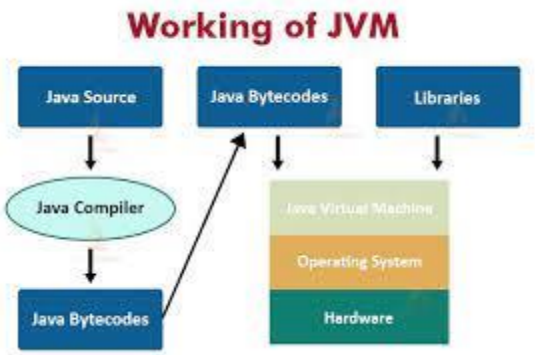
Tabla de contenido

Historia de Java

Java nació en 1995 como un lenguaje de programación por la empresa Sun Microsystems. Se buscaba obtener un lenguaje de programación fácil de programar, rápido y que no dependiese tanto del hardware sobre el que fuera a ser ejecutado

Para poder realizar esta tarea, James Gosling, su creador, hizo que una vez que el programa se compilase, diese un resultado en **bytecode**, un paso intermedio al programa final. Este archivo, que tiene una extensión **.class** no es ejecutable aun por el sistema operativo, debe poder ser ejecutado en un programa que se encargue de convertirlo a código ejecutable en la maquina anfitriona. Este programa es la **JVM** (Java Virtual Machine)

Dicha máquina virtual es lo que lo diferencia del resto de lenguajes de programación y lo que le da la versatilidad que otros lenguajes como C++ no poseen, ya que, por emplear la máquina virtual, el código compilado una única vez puede ejecutarse en cualquier ordenador, siempre que tenga una máquina virtual de java instalada, con independencia de la arquitectura del hardware



Esta característica es una de las que mayor popularidad le ha dado al lenguaje, dándole un impulso y una fama entre los programadores.

Otra de las características de java es su constante evolución. Desde sus comienzos ha paso por un numero amplio de revisiones, ampliando así su funcionalidad con cada una de ellas.

El número de versiones y sus nombres, no ha sido algo constante, y cambio su nomenclatura varias veces. La versión actual seria JAVA SE 17, que se liberó en 2021, y será una **versión LTS** que tendrá soporte hasta 2024. Aunque esta sea la versión más moderna, a la hora de trabajar y en las empresas, suelen emplearse versiones más antiguas, siendo las más empleadas la **JAVA SE 7** y **8** debido a que constan con todas las principales características, y aunque no tengan algunas de las novedades que aportan versiones más modernas, cumplen más que de sobra su funcionalidad

Instalación de Java

Para instalar el programa, deberemos cumplir una serie de requisitos previos:

- Debe estar instalada una máquina virtual de java
- La máquina virtual debe ser compatible con la versión del eclipse

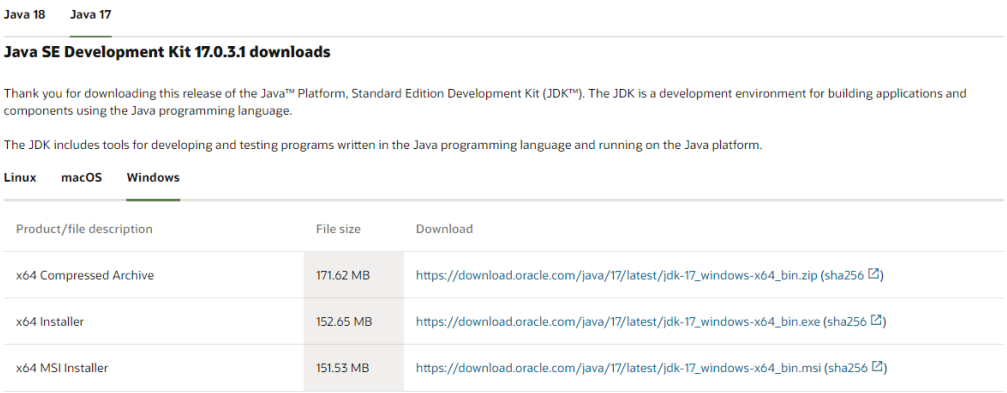
La máquina virtual de java podemos bajarla desde la propia página de Java. Aquí tendremos nuestra primera decisión que tomar, ya que todas las máquinas virtuales tienen dos versiones distintas: la **JRE** (Java Runtime enviroment) y el **JDK** (Java Development Kit)

Es preferible instalar EL JDK. Esto es debido a que el JDK ya de por sí, incorpora el JDK y además, cuenta con algunas herramientas exclusivas como el generador de documentación Javadoc, generadores de keys, …

Una vez seleccionado que maquina vamos a instalar, procederemos a ir a la página de Oracle a descargárnosla. Este proceso puede variar, ya que la pagina puede sufrir remodelaciones, pero normalmente con acceder a esta url nos bastara

https://www.oracle.com/java/technologies/downloads/

una vez en ella, buscaremos la versión 17 y descargaremos el instalador para nuestro sistema operativo



El proceso de instalación es el habitual, siguiente en todos los botones, aceptar la licencia y escoger la ruta de instalación, que se recomienda no cambiar y esperar a que termine

Una vez instalado, solo tendremos que abrir una consola de comandos en Windows y lanzar el siguiente comando para comprobar su correcta instalación:

- Java -version

Una vez lanzado, aparecerá un texto informativo que nos indicará que versión es la que tenemos instalada.

```
java version "1.8.0_271"
Java(TM) SE Runtime Environment (build 1.8.0_271-b09)
Java HotSpot(TM) Client VM (build 25.271-b09, mixed mode)
```

El mensaje podrá diferir un poco, siendo el numero de la versión superior. Se recomienda que la versión de java sea superior a la 1.8, ya que ofrece grandes ventajas que la 1.7 no contempla, como la inclusión del tipo String como caso de la estructura de control switch y expresiones lamda.

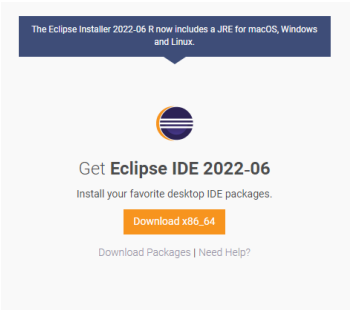
Entornos de desarrollo (Eclipse)

Instalación

Una vez comprobado este paso, proseguiremos descargándonos el **entorno de desarrollo (IDE)** Eclipse. Este software será el que utilizaremos para realizar nuestros programas con java. No es el único que hay en el mercado, ni el más potente, pero si es el IDE más difundido, y el que más soporte tiene en foros como stackoverflow

Otras alternativas podrían ser IntelliJ o Netbeans, no tan utilizados, pero dan muy buen resultado también a la hora de programar

Para descargar Eclipse, iremos al sitio oficial, y descargaremos el instalador para nuestro equipo. Recordad que el enlace a la descarga puede variar ya que suelen remodelar la página de vez en cuando.

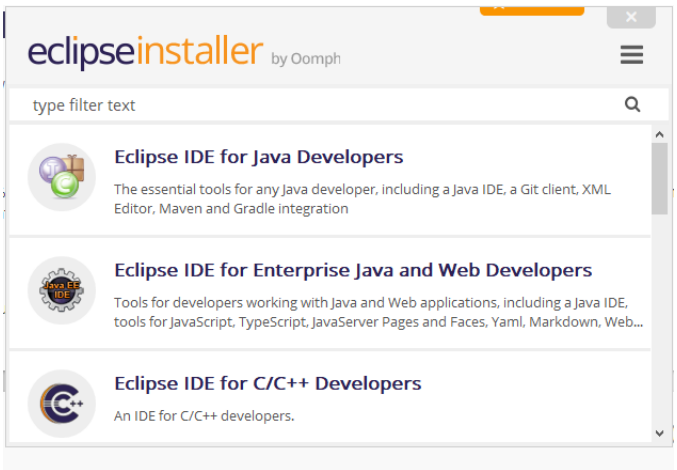


Una vez localizado el enlace para la descarga, obtendremos un archivo ejecutable que deberemos ejecutar para comenzar la instalación del propio editor

Las últimas versiones disponen incluido dentro del proceso de instalación una versión de la JRE, pero es preferible haberla descargado e instalado nosotros con anterioridad, ya que así dispondremos de algunas herramientas extras que si instalamos solo el JRE en lugar del JDK no tendremos

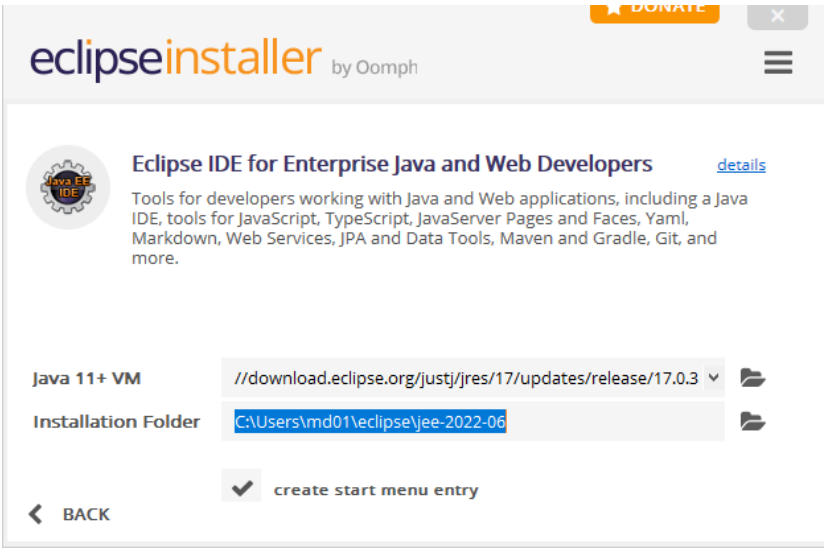
El archivo del instalador pesara alrededor de 120 megas, lo cual es muy ligero, pero necesitara conexión a internet para poder ejecutarse ya que algunas librerías las descarga desde internet en el proceso de instalación.

Para instalarlo, el proceso es relativamente sencillo y está casi automatizado. Cuando iniciemos el proceso, tras una pantalla de bienvenida, nos llevara a un asistente donde deberemos escoger que versión de eclipse instalaremos



de todas las opciones, las que nos permiten trabajar con Java serian **Eclipse IDE for java Developers**, que es la versión más básica con solo la funcionalidad más básica de java, que nos permitirá crear aplicaciones de consola, y si descargamos los plug-ins adecuados aplicaciones de escritorio

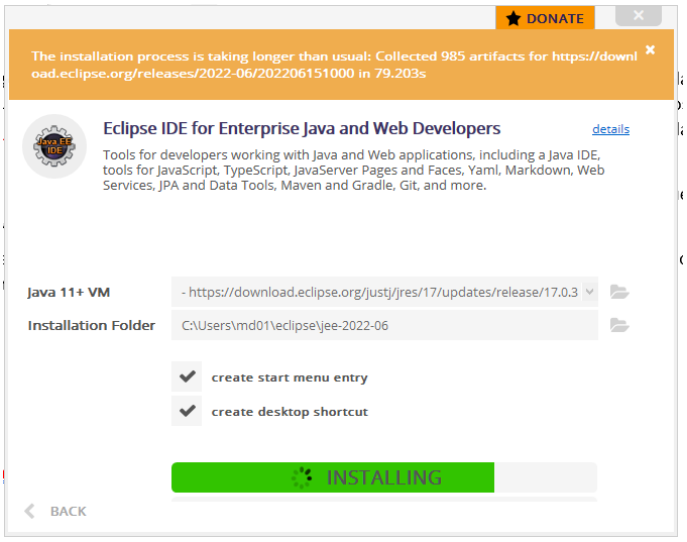
otra opción un poco más avanzada **seria Eclipse IDE for Enterprise java and Web developers**. Esta versión cuenta con las mismas opciones que el ide for java developers, y además soporte para java EE, que nos permitirá crear aplicaciones con interfaces HTML. Es aconsejable instalar esta última ya que es más completa y cuenta con todas las herramientas que necesitaremos, y algunas que en caso que queramos dar el salto a diseñar aplicaciones con servlets nos evitara tener que realizar el proceso de instalación del IDE de nuevo



En la siguiente ventana del asistente, escogeremos la localización de donde tendremos instalada la máquina virtual de Java. Las últimas versiones pueden, como en el caso de arriba, solicitarnos que la JRE que tengamos sea de una versión en concreto o superior. Arriba en el ejemplo, en la versión 2022-6 de eclipse solicita que la versión sea mínima la 11, o superior.

También indicaremos la ruta donde se instalará el entorno. Aquí podéis indicar la dirección que queráis, aunque rutas como C: suelen dar problemas.

Una vez iniciado el proceso de instalación, tardar a un rato en completarse. Se recomienda no navegar por internet, ni tener ninguna aplicación en segundo plano para que el proceso sea lo más rápido posible



Una vez acabada la instalación, ya podremos abrir nuestro programa, y comprobar que todo funciona correctamente.

Primeros pasos y comprobación

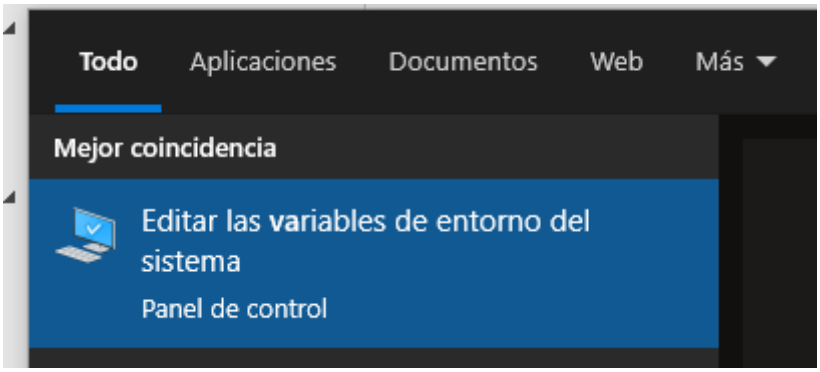
Al arrancar, debería verse una pantalla que nos indica cual será nuestro Workspace, la carpeta donde se guardaran los proyectos que hagamos. De momento aceptaremos, aunque se puede cambiar más adelante.

Tras esto, nos mostrara una pantalla de bienvenida y ya estaremos listos para empezar a programar

Problemas comunes a la hora de la instalación

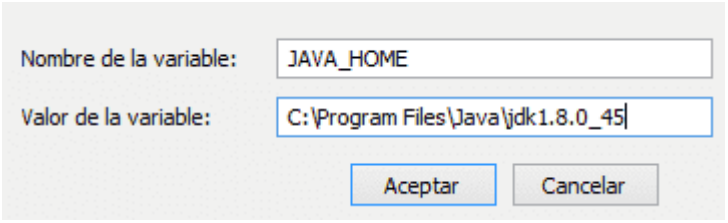
Uno de los problemas más comunes tras instalar el Eclipse es que por algún motivo, no se haya creado una variable de entorno en Windows con la ruta del JRE de java. Esto nos impedirá abrir Eclispe y ponernos a programar

Por suerte, es un fallo que tiene fácil solución. Deberemos comprobar que la variable de entorno este creada bien. Para ello buscaremos en el buscador de Windows la palabra **variables** hasta que nos aparezca en Windows la opción de editar las variables de entorno



Una vez clicada esta opción, nos abrirá una ventana donde tendremos la posibilidad de registrar variables para que el propio Windows pueda tener un rápido acceso a ciertos datos. Pulsaremos el botón editar las variables de entorno que aparece en la parte inferior del cuadro de dialogo y en el asistente que sale, veremos que tenemos dos secciones, una para las variables del usuario con el que se inició sesión en Windows y otra para las del sistema.

Deberemos buscar una variable que se denomine JAVA_HOME, si no estuviese, debería crearse. Esta variable tendrá como dato la dirección del JDK o JRE de java instalado en nuestro ordenador



Normalmente este suele ser el fallo más común del proceso de instalación de java. Para fallos más graves, deberá consultarse la información que proporcione Eclipse en internet, sonde nos llevará a foros especializados como Stack overflow donde podremos consultar la solución a ese problema concreto

Lenguaje Java

Java es uno de los lenguajes de programación más populares y versátiles de hoy día. Es difícil encontrarlo fuera del Top 3 de lenguajes mas utilizados y de los mejor pagados

Esto es debido a que java se creo teniendo en mente la estructura de programación de lenguajes como C/C++, simplificando muchas de las partes más difíciles y haciendo invisibles para el usuario cosas como los punteros.

Debido a tales cambios y a como esta diseñado el compilador para realizar aplicaciones ejecutables, java pierde potencia con respecto a C, pero suple esa carencia con una versatilidad que le permite hacer aplicaciones de Consola, escritorio con interfaz gráfica, con interfaz HTML, servicios web, … y un largo etcétera

Aprender Java, además, sirve como lenguaje puente para aprender otros lenguajes como pueden ser C#, un lenguaje muy similar a java, de Microsoft, que al igual que java, goza de gran popularidad.

La versatilidad que da el propio lenguaje en ocasiones también puede ser una desventaja. Java es un lenguaje que, debido a su creciente adaptabilidad, ha sufrido un gran numero de revisiones. Estas revisiones han ido introduciendo mejoras de manera paulatina en el lenguaje pero que a la larga pueden llegar a generar incompatibilidades y problemas con el hardware.

Actualmente java se encuentra en la versión 18. En los entornos laborales la que se utiliza es la versión 1.7 o 1.8, debido a que esta es la que se utilizan en entornos empresariales. El hecho que muchas empresas hayan decidido mantenerse en estas versiones es debido a que no les son necesarias las nuevas opciones, o tienen miedo que las versiones nuevas no sean compatibles completamente con el hardware y otros programas

Java cuenta con la característica que a diferencia de otros programas, se ejecuta sobre una maquina virtual intermedia, con lo cual, un mismo código puede ejecutarse en distintas maquinas, siempre con algunas limitaciones. Esto hace que el rendimiento global se vea un poco lastrado, pero la ventaja de abstraer el código del hardware en el que se ejecuta es ventaja más que suficiente

Variables

Las variables son una de las estructuras básicas de la programación en Java. Podemos verlas como pequeñas cajas en las que almacenaremos información. Para poder sacarle un buen provecho, las variables deben cumplir una serie de normas:

- Los datos que podemos asignar a una variable están tipados, esto es, si creamos una variable que almacenara números enteros, solo podremos guardar números enteros en ella. Los tipos de datos que podemos guardar son los siguientes
    - Números enteros
    - Numero reales
    - Caracteres
    - Valores booleanos
    - Texto
    - Datos combinados (Objetos)
- Para crear una variable, debe asignársele un espacio en la memoria ram. Para ello deberá asignársele uno de los tipos de datos disponibles por el propio lenguaje java o uno que hayamos creado nosotros. La creación de variables se vera en el apartado tipo de datos con mas detalle, incluyendo la manera de creación y los diferentes tipos de datos que podrán definirse
- Las variables son un tipo de datos que pueden cambiar su valor. Una vez creadas, es recomendable asignarles un valor, ya sea el valor inicial si lo conocemos o un valor neutro en caso de desconocerlo. Esto nos garantiza tener una seguridad en el programa y una estabilidad contra excepciones

Debido a que la programación es algo personal, cada programador escribe su código con pequeñas variaciones y aportaciones. Al crear las variables se les debe asignar un nombre y dicho nombre al quedar al criterio del programador, nos encontraremos una gran cantidad de variaciones tales como:

> Numero_de_Alumnos, num_alumnos, nAlumnos, contador, …

este es un problema que tiene una difícil solución, pero los creadores del lenguaje java, recomiendan una serie de normas, algunas son obligatorias y otras son opcionales para darles nombres a las variables, veámoslas:

- El nombre debe ser descriptivo, con verlo debemos saber para que se usara la variable
- No se pueden usar espacios en el nombre
- El nombre debe empezar siempre por una letra o guion bajo, nunca por un numero
- No se deben utilizar símbolos en el nombre de la variable. Están admitidos los símbolos guion bajo y el símbolo del dólar, pero este último se desaconseja utilizarlo
- La letra Ñ se puede utilizar en las variables en las últimas versiones, pero se desaconseja. En caso de querer utilizarla, para escribir por ejemplo “año” se recomienda usar una variación fonética como “anyo”, “anio”, “anno”
- Las variables en caso de comenzar por letra, siempre será minúscula
- Si los nombres son compuestos por varias palabras, lo recomendable es escribirlo con la primera letra de la primera palabra en minúsculas y la primera letra de cada palabra adicional en mayúsculas.

Ámbito de vida

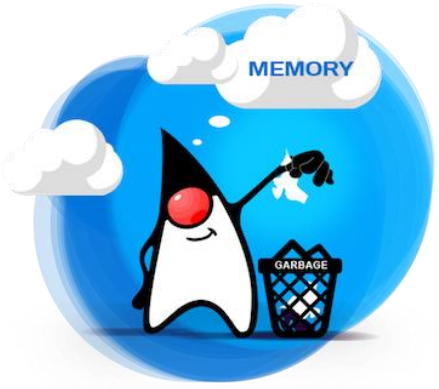
Las variables, como hemos visto antes, se guardan dentro de la memoria RAM. Al ser esta una memoria limitada en tamaño, cuando los programas se ejecutan, cada vez que crean variables van ocupando mas y mas espacio dentro de ella, con el riesgo de que la memoria se ocupe del todo.

Java cuenta con unos mecanismos para poder liberar la memoria de manera transparente al usuario (sin que este tenga que limpiarla como haría un usuario de lenguajes como c/c++).

Este mecanismo es que una variable será accesible desde la apertura de llaves que la precede la declaración de la variable hasta la llave que la cierra (caracteres"{" y "}"). Dicha variable no se podrá acceder desde una zona fuera de esta especificación.

Esto es uno de los aspectos del ámbito de vida, pero una vez que su ámbito de vida se ha acabado, no significa que esa variable libere automáticamente el espacio que tiene ocupado. Esa área de memoria aun tendrá los datos y para el sistema operativo, contará como que aún está en uso

Al no poder liberar la memoria de manera manual como en c/c++ debemos esperar que sea la propia maquina virtual de java la libere. Esa decisión la toma la propia maquina virtual, y aunque puede llegar a ordenársele realizar la tarea a voluntad del programador, no es conveniente hacerlo, es mejor dejar que esta se encargue de hacerlo cuando el decida.



Garbage collector

El encargado de liberar estos espacios recibe el nombre de Garbage collector. Los fundamentos internos de esta rutina son complejos, pero su labor es importante.

Liberar la memoria que ocupan variables que ya no se usan es una tarea que a la maquina virtual de java le cuesta cierto trabajo, por eso espera ciertas condiciones.

De manera sencilla, podemos decir que el garbage collector se iniciara en cuando se cumplan estas condiciones (aunque no se puede asegurar que pase justo cuando se cumplan)

- Haya ya unas cuantas
- El programa no este realizando una labor de proceso pesada
- Necesite espacio adicional

Constantes

Las constantes son un tipo particular de variable. Mientras que las variables pueden cambiar el valor de la información almacenada, una constante no podrá hacerlo. El valor que se le define al inicio es el que mantendrán durante toda la ejecución del programa.

Tipos de datos

En java podemos encontrar diferentes maneras de representar la información, desde los datos más simples a información más compleja y detallada.

Combinando todos ellos podemos almacenar cualquier tipo de información que necesitemos procesar, pero la tendencia actúa es hacer uso del paradigma de programación orientada a objetos

En orden de complejidad, nos encontraremos con los siguientes tipos de datos:

- Primitivas estándar
- Arrays
- Objetos

Datos primitivos (Primitivas)

Estos son los tipos de datos mas sencillos de cualquier lenguaje. Aquí nos encontramos con distintos números, texto, caracteres y variables binarias.

Es el único tipo de información que no son tratados por objetos en java, aunque en verdad hay una clase de la que se encarga su gestión. Estas clases se denominan clases envoltorio o Wrapper.

Para poder almacenarlas y usarlas de manera provisional se almacenarán en variables y dichas variables están tipadas, a diferencia de otros lenguajes. Una variable tipadas es aquella que solo puede tener en su contenido unos valores concretos. Si en ella metemos un número, solo podrá haber números

Datos complejos (Arrays)

Con los datos primitivos podemos juntarlos en formatos de tabla. Dichas tablas pueden llamarse arrays, matrices o arreglos en según qué zonas.

Un array no es mas que una serie de datos primitivas, organizados en una tabla. Todos son del mismo tipo, y a la hora de almacenarse en la memoria RAM están contiguos, por lo cual se accede más rápidamente a ellos.

Pueden ser de una o dos dimensiones, aunque lo normal es encontrarlos de una dimensión, y cada vez es mas habitual utilizar otro tipo de estructuras como listas o vectores para tratar los datos por las ayudas extras que proporcionan en comparación a los arrays

Datos complejos (Objetos)

Aquí ya entraríamos a la base de la programación orientada a objetos. Este paradigma de programación nos permite crear nuestros tipos de datos personalizados, establecer jerarquías y organización entre ellos

Nos permite poder trabajar de una manera muy fácil con datos compuestos complejos y agrupar la información

Casteos de datos

Antes hablamos que las variables debían tener un tipo de datos. En java existe la posibilidad de que ciertos tipos de datos, por ejemplo, los que son de tipo numérico, se comporten como los de otra clase de un grado superior. Esto es: un numero que ocupe 2 bytes en memoria (un short) podrá almacenarse en una variable de tipo int (que ocupa 4 bytes). Pero el caso contrario no sería posible.

Algunos datos numéricos, cuando se operan con textos, se pueden llegar a tratar como si fueran textos en lugar de números, realizándose así una concatenación en lugar de la operación de suma.

Clases envoltorio

En ocasiones es muy conveniente poder tratar los datos primitivos (int, boolean, etc.) como objetos. Por ejemplo, los contenedores definidos por el API en el package java.util (Arrays dinámicos, listas enlazadas, colecciones, conjuntos, etc.) utilizan como unidad de almacenamiento la clase Object. Dado que Object es la raíz de toda la jerarquía de objetos en Java, estos contenedores pueden almacenar cualquier tipo de objetos. Pero los datos primitivos no son objetos, con lo que quedan en principio excluidos de estas posibilidades.

Para resolver esta situación el API de Java incorpora las clases envoltorio (wrapper class), que no son más que dotar a los datos primitivos con un envoltorio que permita tratarlos como objetos. Por ejemplo, podríamos definir una clase envoltorio para los enteros, de forma bastante sencilla

Operadores

Ya hemos tratado las distintas maneras de integrar la información. Pero la información por si ola no sirve de nada, deberemos poder operara con ella. Para ello vamos a tener una serie de operadores en java que nos permite realizar distintas operaciones con los datos que disponemos.

Estas operaciones están reservadas a primitivas estándar, pero se podrán aplicar a objetos, pero no al objeto en sí, sino a los datos que tiene en su interior

Estas operaciones tienen una prioridad a la hora de aplicarse, lo mismo que las operaciones matemáticas, algunas de las cuales son parte de estos operadores.

Deberemos entender estas operaciones y cual es la prioridad con la que se aplican

Prior.	Operador	Tipo de operador	Operación
1	++ -- +, - ~ ¡	Aritmético Aritmético Aritmético Integral Booleano	Incremento previo o posterior (unario) Incremento previo o posterior (unario) Suma unaria, Resta unaria Cambio de bits (unario) Negación (unario)
2	(tipo)	Cualquiera	
3	*, /, %	Aritmético	Multiplicación, división, resto
4	+, - +	Aritmético Cadena	Suma, resta Concatenación de cadenas
5	<< >> >>>	Integral Integral Integral	Desplazamiento de bits a izquierda Desplazamiento de bits a derecha con inclusión de signo Desplazamiento de bits a derecha con inclusión de cero
6	<, <= >, >= instanceof	Aritmético Aritmético Objeto, tipo	Menor que, Menor o igual que Mayor que, Mayor o igual que Comparación de tipos
7	== ¡= == ¡=	Primitivo Primitivo Objeto Objeto	Igual (valores idénticos) Desigual (valores diferentes) Igual (referencia al mismo objeto) Desigual (referencia a distintos objetos)
8	& &	Integral Booleano	Cambio de bits AND Producto booleano
9	^ ^	Integral Booleano	Cambio de bits XOR Suma exclusiva booleana
10	| |	Integral Booleano	Cambio de bits OR Suma booleana
11	&&	Booleano	AND condicional
12	||	Booleano	OR condicional
13	? :	Booleano, cualquiera, cualquiera	Operador condicional (ternario)
14	= *=, /=, %= +=, -= <<=, >>= >>>= &=, ^=, |=	Variable, cualquiera	Asignación Asignación con operación

Tipos de operadores

Podemos separar los operadores en cuatro grandes grupos:

- Aritméticos: son los que en su base operan con valores numéricos, con indiferencia de su base, para obtener valores numéricos. En estas categorías nos encontramos las sumas, restas, multiplicaciones, paréntesis para ordenar las operaciones, …
- Lógicos: estos operadores operan sobre variables booleanas, siguiendo las normas del algebra booleana.
- De asignación: estos son los que se encargan de almacenar en memoria, en las variables los resultados de las operaciones
- Unarios: estos operadores operan a nivel de bit dentro de las variables, sirven para gestionar la información de cada octeto de manera única a nivel de bit. Vienen heredados de C para el manejo de registros de CPU

Estructuras de control

En programación, las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa.

Con las estructuras de control se puede:

- De acuerdo con una condición, ejecutar un grupo u otro de sentencias **(If-Then-Else)**
- De acuerdo con el valor de una variable, ejecutar un grupo u otro de sentencias **(Switch-Case)**
- Ejecutar un grupo de sentencias mientras se cumpla una condición **(Do-While)**
- Ejecutar un grupo de sentencias hasta que se cumpla una condición **(Do-Until)**
- Ejecutar un grupo de sentencias un número determinado de veces **(For-Next)**

Todos los lenguajes de programación modernos tienen estructuras de control similares. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis; cada lenguaje tiene una sintaxis propia para expresar la estructura, aunque java suele utilizar una sintaxis.

If

es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves.

Puede anidarse con otra sección llamada else a continuación de las llaves. En este bloque se podrá definir la lógica que ejecutará antes de seguir ejecutando el programa principal si no se cumple la condición del if.

En java también es posible después de un bloque else abrir un bloque if nuevo, y continuar anidándolo de manera continuada

Switch

Al igual que if, switch..case controla el flujo del programa especificando en el programa que código se debe ejecutar en función de unas variables. En este caso en la instrucción switch se compara el valor de una variable sobre los valores especificados en la instrucción case.

For

La declaración for se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración for tiene tres partes separadas por (;). La inicialización de la variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier estamento válido en lenguaje C sin relación con las variables declaradas.

While

Un bucle del tipo while es un bucle de ejecución continua mientras se cumpla la expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor.

Do while

El bucle do while funciona de la misma manera que el bucle while, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutará al menos una vez.

Salida de datos

Hasta El momento, solo hemos hablado de guardar información en variables. Estas variables se alojan en la memoria RAM, y por lo tanto no son visibles en ningún momento.

Java, dependiendo del entorno que se ejecute puede mostrar esa información en distintos formatos, ya sea vía páginas web, aplicaciones de escritorio, pero para ello deberá contarse con un framework adecuado. Pero de manera común a todos estos, podemos hacer uso de la consola para sacar información,

Este método es relativamente sencillo de usar. Bastara con hacer uso de una función del sistema que se encargara de mostrar el mensaje por la salida estándar del sistema. Esta salida, normalmente será la consola del sistema, un terminal similar a CMD donde podremos revisar la información que saquemos

Para hacer uso de este, lo que tendremos que hacer será ejecutar la siguiente instrucción y pasarle como parámetro el string que queramos mostrar. Podrá pasársele cadenas de texto, objetos, variables, ya que es el propio java el que intentara darle formato.

Para objetos será recomendable insertar dentro de la clase un override del método toString (ver capítulo de herencia)

Entrada de datos

La entrada de datos es una labor mas compleja en Java. Para poder leer una información en java, necesitaremos leerla desde el teclado en la consola, y toda esa información será procesada como texto por lo que nos será imposible guardarla directamente en variables de tipo numérico.

Para poder trabajar con datos que introduzca desde teclado, haremos uso de varias clases, aunque la mas habitual es la clase Scanner, por todas las utilidades que trae

Aunque sea el método mas popular para gestionar la entrada de datos desde consola, este método no es del todo fiable. Si se usan las funciones de lectura de datos enteros como nextInt() o cualquier otra numérica, entra en conflicto, y si la siguiente instrucción de lectura de teclado es una para leer texto, nextLine(), esta fallará, deberá realizarse dos veces

Para solucionar este pequeño problema, suele realizarse una clase auxiliar con distintos métodos o funciones en los que gestionaremos la entrada de datos y su validación para un tipo de datos concreto.

Esto nos aporta fiabilidad, sencillez de código, reusabilidad y robustez, ya que mediante excepciones, o otros métodos, podemos solicitar la corrección de los datos si los usuarios no introducen lo requerido por el programa

Funciones

Los métodos son funciones que pueden ser llamadas dentro de la clase o por otras clases. La implementación de un método consta de dos partes, una declaración y un cuerpo. La declaración en Java de un método se puede expresar esquemáticamente como:

```
tipoRetorno nombreMetodo( [lista_de_atributos] ) {
    cuerpoMetodo
    }
```

Los métodos pueden tener numerosos atributos a la hora de declararlos, incluyendo el control de acceso, si es estático o no estático, etc. La sintaxis utilizada para hacer que un método sea estático y su interpretación, es semejante en Java y en C++. Sin embargo, la sintaxis utilizada para establecer el control de acceso y su interpretación, es muy diferente en Java y otros lenguajes.

La lista de argumentos es opcional, puede limitarse a su mínima expresión consistente en dos paréntesis, sin parámetro alguno en su interior. Los parámetros, o argumentos, se utilizan para pasar información al cuerpo del método.

Asi también podemos asignar antes de la declaración un modificador al comportamiento de la función.

- **Static**: indica que los métodos pueden ser accedidos sin necesidad de instanciar un objeto del tipo que determina la clase
- **Abstract**: indica que el método no está definido en la clase, sino que se encuentra declarado ahí para ser definido en una subclase (sobreescrito).
- **Final**: evita que un método pueda ser sobreescrito.

El control de acceso también tiene un significado especial cuando se programan funciones..

En la siguiente descripción, se indica cómo se trata el control de acceso cuando se tienen entre manos a los constructores:

- **Private**: Ninguna otra clase puede instanciar objetos de la clase. La clase puede contener métodos públicos, y estos métodos pueden construir un objeto y devolverlo, pero nadie más puede hacerlo.
- **Protected**: Solamente las subclases de la clase pueden crear instancias de ella.
- **Package**: el modificador para solo tener acceso desde paquete
- **Public**: Cualquier otra clase puede crear instancias de la clase.

Existe una cuarta opción que es ignorar la especificación de modificadores de acceso. En este caso el rango que se aplicara es el de paquete. La función podrá ser accedida desde cualquier clase que se encuentre dentro del paquete que este definida

Parámetros de funciones

En Java, todos los métodos deben estar declarados y definidos dentro de la clase, y hay que indicar el tipo y nombre de los argumentos o parámetros que acepta. Los argumentos son como variables locales declaradas en el cuerpo del método que están inicializadas al valor que se pasa como parámetro en la invocación del método.

En Java, todos los argumentos de tipos primitivos deben pasarse por valor, mientras que los objetos deben pasarse por referencia. Cuando se pasa un objeto por referencia, se está pasando la dirección de memoria en la que se encuentra almacenado el objeto.

Si se modifica una variable que haya sido pasada por valor, no se modificará la variable original que se haya utilizado para invocar al método, mientras que si se modifica una variable pasada por referencia, la variable original del método de llamada se verá afectada de los cambios que se produzcan en el método al que se le ha pasado como argumento.

Sobrecarga de métodos

Puede darse el caso que como en el caso de los constructores de una clase, deseemos tener una serie de métodos, todos con el mismo nombre pero que reciban distintos tipos de parámetro, y darle una codificación a ese método distinta en cada caso. Java nos permite realizar esta tarea.

Las únicas condiciones que pone es que todos los métodos deberán devolver obligatoriamente el mismo tipo de return, y en caso de contar con los mismos tipos de parámetro, no deberán estar los tipos en el mismo orden

La mayor funcionalidad que nos puede proveer la sobrecarga podemos encontrarla en combinación con la P.O.O. de una BD. un caso podría ser tener una colección de funciones llamadas insertar, que cada una recibirá un parámetro de tipos distinto y luego dentro de cada una de ellas montar la sentencia SQL Personalizada para cada uno

Programación orientada a objetos

La programación Orientada a Objetos es el paradigma actual de programación. Dicho de otra manera, la manera de estandarizar la programación incluso en diferentes lenguajes con tal de que su aprendizaje y uso sea similar para todos

Aunque las normas son bastante similares, vamos a centrarnos en explicar las peculiaridades de las que aplica java para realizar su implementación:

- A excepción de los datos primitivos, todo son objetos en java
- Los objetos son instancias de unos patrones sin datos llamadas clases. Los objetos tienen información (o estar vacíos)
- Las clases son patrones (recetas) que definen que información tienen los objetos y que cosas pueden hacer. A estos se los llama atributos y métodos respectivamente
- Para poder pasar de una clase a un objeto deben usarse unos métodos especiales llamados constructores. Estos se encargarán de reservar espacio en memora para poder utilizar luego el objeto
- Los constructores deben llamarse con el operador new, y siempre se guardarán en una variable con el nombre de la clase
- Existe la herencia dentro de java, pero esta siempre será herencia simple, solo se podrá heredar de una clase
- Para darle mayor versatilidad a la herencia simple, se desarrollo una suerte de herencia múltiple con interfaces

Esto no es más que un pequeño listado de las características mas comunes de la programación orientada a objetos, que iremos detallando cada una de ellas en su correspondiente categoría

Además de estas características, propias de las clases, para facilitar la labor de la seguridad, hay que trabajar con las clases de una manera concreta. Las clases no deben dejar acceder de manera directa a los atributos, estos deben estar encapsulados, y su acceso solo deberá hacerse a trabes de métodos, ya sea para modificar los valores o para leerlos

También las clases deben ser una abstracción de lo que queramos modelar

Veamos ahora como se crean las clases

Clases

Pongámonos ya a trabajar con un ejemplo. El proceso que deberemos hacer para hacer una buena clase es el siguiente

- Modelizar la entidad que queramos convertir a objeto: Para ello, escogeremos los atributos que nos harán falta para que el ordenador entienda que eso es lo que representa el objeto real. Intentaremos atenernos a los mínimos. Deberemos plantearnos también los tipos de esos datos, ya que los atributos pueden ser datos de tipo primitivos o datos de tipo objeto a su vez
- Una vez modelizados, procederemos a crear una clase nueva en nuestro proyecto. Y en ella definiremos los atributos con los tipos que hayamos decidido darles, pero además de ello, deberán estar precedidos por la palabra private. Esta no es la única opción. Mas adelante veremos que otras opciones es no especificar nada delante de los atributos, poner protected o incluso public. Por normas de estilo y del lenguaje se recomienda usar siempre private

- Una vez creados los atributos, justo debajo de ellos de deberán crear dos métodos especiales. Estos métodos se llamarán como la clase, y deberán estar precedidos de la palabra public. Uno de ellos llevará como parámetros todos los atributos, y el otro no recibirá ningún parámetro. Respectivamente se denominarán constructor completo y constructor vacío. El constructor vacío puede dejarse libre de contenido ya que la propia maquina virtual de java se encarga de gestionar su funcionamiento, pero el completo hay que rellenarlo. Es un proceso sencillo. Basta con asignarle a cada atributo el valor del parámetro adecuado del constructor. Si parámetro y constructor tienen el mismo nombre, se antepondrá el selector this. a el atributo para diferenciarlo. Veamos un ejemplo:

  ```
  public Fecha(int dia, int mes, int anho) {
      this.dia = dia;
      this.mes = mes;
      this.anho = anho;
  }
  ```

  Sigamos con los métodos que todas las clases deben tener. Al estar definidos los atributos como privados, deberemos definir por cada atributo dos métodos, uno para poder dar un valor nuevo a ese atributo en concreto y otro para poder comprobar que valor tiene ese atributo en concreto. Estos métodos se denominan métodos setter y getter respectivamente. Los métodos setter serán métodos en los que reciba por parámetro el nuevo valor y lo asigne al atributo. Lo mismo que en caso del constructor, si es necesario poner el selector this. delante del atributo, lo haremos si coinciden nombre de atributo y parámetro. El getter en cambio no tendrá parámetro alguno, pero devolverá el atributo correspondiente. Veamos un ejemplo de cada uno de ellos:

  ```
  // Getter
  public int getDia() {
          return dia;
  }

  // Setter
  public void setDia (int dia) {
          this.dia = dia;
  }
  ```

- Una vez creados los constructores, los métodos getters y setter, solo restara añadirle los métodos propios. Estos métodos serán todo aquel método que no encaje en la descripción de los de arriba y sirva para opera con nuestro objeto. Es normal encontrarnos aquí con funciones que heredaremos de otras clases, y sobrescribirlas para darle otra funcionalidad, o incluso funciones propias como funciones de presentación de datos. Una de las funciones habituales suele ser la función toString(), heredada desde la clase Object. Esta función se usará para mostrar una representación en formato texto de los datos del objeto, ya que no es tan simple mostrar los en consola por ejemplo al contar estos con varios datos simples

Modificadores de acceso

Ya nombramos estos modificadores de pasada a la hora de crear las clases, ahora vamos a entrar en detalle. En java existen cuatro modificadores de acceso que podemos utilizar para proteger el acceso a nuestras variables, los atributos de una clase o la visibilidad de una función

Estos modificadores son en orden de mayor a menor permisibilidad: public, package (no hace falta ponerlo, es el valor por defecto), protected y private.

Cada uno de ellos nos restringe el acceso a cada uno de ellos de diversa manera, siendo apropiado o completamente desaconsejado usarlos en según que casos, como explicaremos ahora

- **Public**: este modificador da completo acceso a la variable o a la función, desde cualquier lugar. Esto puede dar lugar a que se use desde cualquier sitio, o en caso de variables, que puedan modificarse o accederse por error desde otras partes del programa. Cualquier clase externa a la nuestra puede verla
- **Package**: las variables protegidas por este modificador solo podrán ser visibles desde el propio paquete donde estén. Cualquier clase externa en el mismo paquete podrá tener acceso a ellas, verlas y modificarla. En caso de las funciones, solo podrán usarse en ese paquete, quedando completamente invisibles y accesibles fuera de este
- **Protected**: aquí ya entramos con herencia. Cualquier método o atributo que haga uso de este modificador de acceso, solo permitirá acceder a las variables, atributos o métodos que lo definan desde la propia clase y las clases que heredan la clase en la que se definió. Normalmente se usa el modificador private antes que este por estándar
- **Private**: dentro de la orientación a objetos el que usaremos para definir el acceso a los atributos. Con ello solo se podrán modificar desde la propia clase, sin tener que hacer uso de los getters o setters. Cualquier función definida también como private, solo se podrá llamar y usar desde dentro de la propia clase donde esta definida, no se podrá tener acceso desde fuera de esta

En esta tabla resumen veremos el alcance de cada elemento modificado por estos

Modificador/Acceso	Clase	Paquete	Subclase	Todos
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
Package (default)	Sí	Sí	No	No
private	Sí	No	No	No

Encapsulación

Conociendo ya los modificadores de acceso, y sabiendo que acceso tiene cada uno de ellos, ahora podemos entender el motivo de por que los atributos de una clase los definiremos como privates. Esta manera de trabajar con las clases se llama encapsulamiento, y consiste en definir los atributos private o en su defecto protected, aunque es preferible por estándar hacerlo privado.

De esta manera, los atributos solo se podrán acceder ya sea para consultar o modificar a través de los métodos public o protected que hayamos definido

Lo normal suele ser definir todos los métodos públicos, y si en la clase, hacemos uso de alguna función que su uso es solo interno en esa clase, si se podrán definir como privados

Paquetes

Los paquetes en java es la manera que tiene de ordenar el código en categorías para facilitar tener las cosas ordenadas

Cada clase deberá estar siempre guardada en al menos un paquete. Y aunque es posible no definir ninguno, es algo totalmente desaconsejable ya que los proyectos java crecen bastante y que la estructura del proyecto que de hecha un caos es una posibilidad

A nivel interno de sistema operativo, los paquetes se traducen como carpetas, donde solo alojaremos código fuente. Bajo ningún concepto se guardarán imágenes, bases de datos u otro tipo de recursos en estos.

No hay una estructura estándar de paquetes, podríamos decir, pero hay unas normas que se suelen tomar como estándar:

- Los nombres de paquetes solo tendrán letras minúsculas, y números, no usar caracteres especiales, letra Ñ y similar
- Se recomienda usar un nombre de dominio como paquete inicial. Si el proyecto es de la empresa informática fácil se recomienda usar algo similar a esto: com.informaticafacil
- Cada nombre que este separado por un punto será una nueva carpeta, y puede haber mas de dos niveles, que son los que tiene el ejemplo anterior
- Crear nombres de paquete descriptivos y tantos subpaquetes como necesitemos, y usad nombres descriptivos

Instanciación de objetos

En java las clases son las recetas de las que luego se pueden crear múltiples instancias del mismo tipo de objetos. La clase define las variables y los métodos comunes a las instancias de ese tipo (el tipo de la clase creada), pero luego, cada instancia de esta clase tendrá sus propios valores, siendo única.

Para crear instancias de objetos, que es lo que realmente utilizaremos en programación, debemos tener claras dos cosas indispensables, la primera es el nombre de la clase para la cual vamos a crear el objeto y segundo el constructor que dicha clase posee, es decir, si el constructor recibe o no parámetros.

Para crear objetos en Java, el lenguaje nos proporciona el comando new, con este comando le decimos a Java que vamos a crear un nuevo objeto de una clase en específico y le enviamos los parámetros (en caso de ser necesario) según el constructor, veamos un ejemplo.

Vamos a crear un objeto o instancia en Java para la clase que hemos creado al comienzo llamada MiClase. Esta clase tiene un constructor que no recibe parámetros, por lo cual no es necesario enviar algún tipo de valor al momento de crear el objeto, veamos entonces la sintaxis para crear un objeto del tipo MiClase en java.

MiClase miObjeto; //Declaramos una variable del tipo de la clase

miObjeto = new MiClase(); //Aquí ya hemos creado un objeto de MiClase

al crear la variable miObjeto, en memoria RAM, de esta manera se le reserva el espacio, el objeto existe, aunque no tienen ningún valor en sus atributos, debido al hecho que usamos el constructor vacío. Para darle valores ahora solo nos quedara rellenarlos con los métodos set

si ya hubiésemos contado con los datos que el objeto fuese a tener, podríamos haber tomado la decisión de usar el constructor completo en la construcción de este, y a la misma hora de crearlo ya tendría también los datos asignados, sin tener que hacer uso de los métodos set a posteriori

Clases envoltorio (Wrapper)

En ocasiones es muy conveniente poder tratar los datos primitivos (int, boolean, etc.) como si fuesen objetos. Por ejemplo, los contenedores de tipo Arrays dinámicos, listas, colecciones, conjuntos, y demás utilizan como unidad de almacenamiento la clase Object. Dado que Object es la raíz de toda la jerarquía de objetos en Java, estos contenedores pueden almacenar cualquier tipo de objetos. Pero los datos primitivos no son objetos, con lo que quedan en principio no pueden usarse para almacenarlos dentro de estas estructuras.

Para resolver esta situación el API de Java incorpora las clases envoltorio (wrapper class), que no son más que dotar a los datos primitivos con un envoltorio que permita tratarlos como objetos.

Para cada tipo de dato primitivo existe una clase Wrapper, y aunque el nombre suele ser igual al del dato primitivo pero en mayúsculas hay algunas excepciones: veamos cuales son los tipos

Las clases envoltorio existentes son:

- Byte para byte.
- Short para short.
- Integer para int.
- Long para long.
- Boolean para boolean
- Float para float.
- Double para double y
- Character para char.

Herencia

La herencia es el mecanismo por el que se crean nuevos objetos definidos en términos de objetos ya existentes. Por ejemplo, si se tiene la clase Ave, se puede crear la subclase Pato, que es una especialización de Ave.

```
class Pato extends Ave {
   int numero_de_patas;
   }
```

La palabra clave extends se usa para generar una subclase (especialización) de un objeto. Un Pato es una subclase de Ave. Cualquier cosa que contenga la definición de Ave será copiada a la clase Pato, además, en Pato se pueden definir sus propios métodos y variables de instancia. Se dice que Pato deriva o hereda de Ave.

Además, se pueden sustituir los métodos proporcionados por la clase base. Utilizando nuestro anterior ejemplo de MiClase, aquí hay un ejemplo de una clase derivada sustituyendo a la función Suma_a_i():

```
import MiClase;
public class MiNuevaClase extends MiClase {
   public void Suma_a_i( int j ) {
      i = i + ( j/2 );
      }
   }
```

Ahora cuando se crea una instancia de MiNuevaClase, el valor de i también se inicializa a 10, pero la llamada al método Suma_a_i() produce un resultado diferente:

```
MiNuevaClase mnc;
mnc = new MiNuevaClase();
mnc.Suma_a_i( 10 );
```

Java se diseñó con la idea de que fuera un lenguaje sencillo y, por tanto, se le denegó la capacidad de la herencia múltiple, tal como es conocida por los programadores C++. En este lenguaje se añade cierta complejidad sintáctica cuando se realiza herencia múltiple de varias clases, las cuales comparten una clase base común (hay que declarar dicha clase como virtual y tener bastante cuidado con los constructores), o también cuando las clases base tienen miembros de nombre similar (entonces hay que utilizar especificadores de acceso).

Por ejemplo, de la clase aparato con motor y de la clase animal no se puede derivar nada, sería como obtener el objeto toro mecánico a partir de una máquina motorizada (aparato con motor) y un toro (aminal). En realidad, lo que se pretende es copiar los métodos, es decir, pasar la funcionalidad del toro de verdad al toro mecánico, con lo cual no sería necesaria la herencia múltiple sino simplemente la compartición de funcionalidad que se encuentra implementada en Java a través de interfaces.

Interfaces

Una interfaz es como una clase Java, pero solo tiene constantes estáticas y método abstracto. Java usa la interfaz para implementar herencia múltiple. Una clase Java puede implementar múltiples interfaces Java. Todos los métodos en una interfaz son implícitamente públicos y abstractos.

Para usar una interfaz en su clase, agregue la palabra clave “implements” después del nombre de su clase seguido del nombre de la interfaz. Esto hará

Mediante una interfaz indicamos que debe hacerse pero no como se debe implementar. Veremos con un ejemplo la creación de una interfaz y la implementación de la misma en dos clases distintas.

```
public interface Juego {
int UN_JUGADOR=1;
int DOS_JUGADORES=2
   void iniciar();
   void jugar();
   void finalizar();
}
```

En esta interfaz se pueden ver los dos apartados de los que hablamos con anterioridad, las constantes y las definiciones de métodos. No es necesario que ambas estén presentes, pudiendo una interfaz solo tener constantes, solo métodos o ambas. En cualquiera de los casos, el que mayor problemas nos puede dar es cualquera que tenga métodos, ya que la clase que implemente la interfaz deberá definir esos códigos

Para ello, lo que deberá hacer será darles código, incluyendo una nueva definición de la función el la clase donde se implementa, precedida de directiva @override, para indicarle que es la sobreescritura del método, justo encima de la definición de este

Ya con esto, tenemos desarrollada un simulacro de herencia múltiple

Polimorfismo

El Polimorfismo es uno de los 4 pilares de la programación orientada a objetos (POO) junto con la Abstracción, Encapsulación y herencia.

el término "Polimorfismo" es una palabra de origen griego que significa "muchas formas". Este término se utiliza en la POO para "referirse a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos".

El polimorfismo es una característica de la programación orientada a objetos que permite llamar a métodos con igual nombre pero que pertenecen a clases distintas.

En Java es necesario que las clases compartan una superclase común para implementar el polimorfismo, luego veremos que también se puede implementar el polimorfismo en Java mediante interfaces.

Con el polimorfismo podemos implementar programas que luego son fácilmente extensibles.

Abstracción

Sobreescritura

Para entender en su totalidad el código fuente escrito en Java, es imprescindible tener muy claro el concepto de la redefinición o sobreescritura de métodos. Tanto es así, que a continuación se vuelve a insistir sobre la cuestión, a pesar de haberla tratado varias veces a lo largo de este documento.

La sobreescritura de métodos es una característica más de la herencia en Java. Es decir, en Java las nuevas clases se pueden definir extendiendo clases ya existentes. Aquí surgen los conceptos de subclase que sería la clase obtenida, y superclase, que sería la clase que está siendo extendida, tal como también ya se ha explicado.

Cuando una nueva clase se extiende desde otra que ya existía, todas las variables y métodos que son miembros de la superclase (y todos aquellos miembros de los antecesores de la superclase) serán también miembros de la subclase.

En el supuesto de que en el entorno en que se va a mover la nueva subclase, alguno de los métodos de la superclase (o alguno de sus antecesores) no sea adecuado para los objetos originados de la subclase, es posible reescribir el método en la subclase para que se adapte en su funcionamiento a los objetos del tipo de la subclase.

La reescritura del método dentro de la subclase es lo que se conoce por sobreescritura de métodos (overriding methods). Todo lo que se requiere en Java para poder sobeescribir un método es utilizar el mismo nombre del método en la subclase, el mismo tipo de retorno y la misma lista de argumentos de llamada, y en el cuerpo de ese método en la subclase proporcionar un código diferente y específico para las acciones que vaya a realizar sobre objetos del tipo que origina la nueva subclase.

Aunque no se ha discutido todavía la capacidad de multihilo de Java, marcada por la clase Thread; si se puede indicar que hay un método en esta clase, run(), que es de vital importancia. La implementación de este método run() en la clase Thread está completamente vacía, indicando que no se hace nada sino que se limita a definir un método interfaz. No tiene sentido para el método run() definir nada por defecto, porque en último lugar será utilizado para realizar los que el hilo de ejecución necesite y los programadores de la librería de Java no pueden anticipar esas necesidades.

Pero, por otro lado, este método no se puede declarar como abstracto, porque esto podría influir en la instanciación de objetos de la clase Thread, y la instanciación de estos objetos es un aspecto sumamente crítico en la programación multihilo en Java. Luego el resultado ha sido que los programadores de Sun han definido a run() como un método vacío.

Se puede reemplazar completamente la implementación de un método heredado indicando el mismo nombre, la misma lista de argumentos y el mismo tipo de retorno; y colocar en el cuerpo del método el código que realice la función que sea menester en su nueva situación. En el caso del método run(), se podría hacer algo como:

```
class MiThread extends Thread {
   void run() {
      // código del método
      }
   }
```

En este fragmento de código, el método run() de la clase MiThread sobrescribe al método run() de la clase Thread y proporciona una nueva implementación.

Hay que tener cuidado con la diferencia entre sobrecargar y sobrescribir un método, y entenderla correctamente. Para sobrecargar un método hay que duplicar el nombre el método y el tipo que devuelve, pero utilizar una lista de argumentos diferente al original. Para sobre escribir un método, no solamente el nombre y el tipo de retorno deben ser iguales, sino que ha de serlo también la lista de argumentos. Es decir, que hay que estar atentos a la lista de argumentos que se indica para un método, en evitación de la sobrecarga cuando en realidad se cree que se está sobrescribiendo, o viceversa.

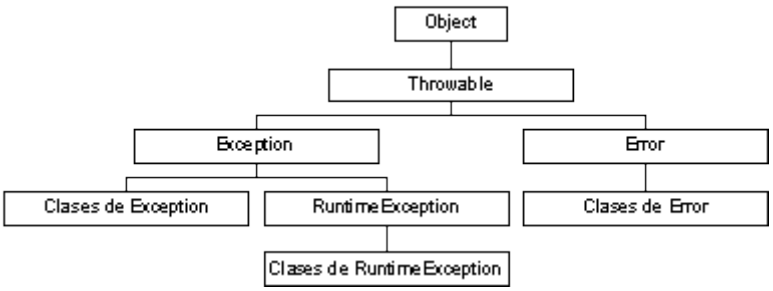
Excepciones

En Java los errores en tiempo de ejecución (cuando se está ejecutando el programa) se denominan excepciones, y esto ocurre cuando se produce un error en alguna de las instrucciones de nuestro programa, como por ejemplo cuando se hace una división entre cero, cuando un objeto es 'null' y no puede serlo, cuando no se abre correctamente un fichero, etc. Cuando se produce una excepción se muestra en la pantalla un mensaje de error y finaliza la ejecución del programa.

En Java (al igual que en otros lenguajes de programación), existen muchos tipos de excepciones y enumerar cada uno de ellos sería casi una labor infinita. En lo referente a las excepciones hay que decir que se aprenden a base experiencia, de encontrarte con ellas y de saber solucionarlas.

Cuando en Java se produce una excepción se crear un objeto de una determina clase (dependiendo del tipo de error que se haya producido), que mantendrá la información sobre el error producido y nos proporcionará los métodos necesarios para obtener dicha información. Estas clases tienen como clase padre la clase Throwable, por tanto, se mantiene una jerarquía en las excepciones.

A continuación, mostramos algunas de las clases para que nos hagamos una idea de la jerarquía que siguen las excepciones, pero existen muchísimas más excepciones que las que mostramos:



Tratamiento de excepciones

Dado el carácter impredecible y bloqueante de estas, es importante saber preveer donde puede saltar y darle un tratamiento

Para realizar esta tarea no tendremos mas que rodear la o las sentencias que puedan soltar excepciones con un bloque try catch como el del ejemplo inferior

```
try {
        int[] myNumbers = {1, 2, 3};
        System.out.println(myNumbers[10]);
} catch (Exception e) {
        System.out.println("Something went wrong.");
}
```

Es verdad que la captura de excepciones superior cumple con su objetivo, pero no diferencia del tipo de excepción que se captura, tratando la excepción que sea de la misma manera. Es una mejor practica en el bloque catch capturar todas las excepciones que puedan surgir.