# Sumario

UT 02: Introducción a la Orientación a Objetos....2
1 Antecedentes y características de Java....2
1.1 ¿Qué es Java?....2
1.2 Historia de Java....3
1.3 La Programación Orientada a Objetos (POO) y Java....5
1.4 Fortalezas de Java: Independencia de la plataforma , trabajo en red, seguridad y simplicidad....6
1.5 Los Bytecodes....7
1.6 Tipos de aplicaciones en Java....8
2 Principios básicos de la programación orientada a objetos....9
2.1 Beneficios de la Programación Orientada a Objetos....10
2.2 Características de la Programación Orientada a Objetos....11
3 Los objetos....12
3.1 Características de los Objetos. Estado, comportamiento e identidad....12
3.2 Propiedades y métodos....13
3.3 Interacción entre objetos....13
4 Clases, atributos y métodos....14
5 Encapsulación y visibilidad....15
6 Relaciones entre clases. Herencia....15
6.1 Relaciones entre clases y objetos....15
6.2 Herencia....15
7 Introducción a la API de Java....16

# UT 02: Introducción a la Orientación a Objetos

## 1 Antecedentes y características de Java

### 1.1 ¿Qué es Java?

Java es un lenguaje sencillo de aprender, con una sintaxis parecida a la de C++, pero en la que se han eliminado elementos complicados y que pueden originar errores. Java es orientado a objetos, con lo que elimina muchas preocupaciones al programador y permite la utilización de gran cantidad de bibliotecas ya definidas, evitando reescribir código que ya existe. Es un lenguaje de programación creado para satisfacer nuevas necesidades que los lenguajes existentes hasta el momento no eran capaces de solventar.

Una de las principales virtudes de Java es su independencia del hardware, ya que el código que se genera es válido para cualquier plataforma. Este código será ejecutado sobre una máquina virtual denominada Maquina Virtual Java (MVJ o JVM – Java Virtual Machine), que interpretará el código convirtiéndolo a código específico de la plataforma que lo soporta. De este modo el programa se escribe una única vez y puede hacerse funcionar en cualquier lugar. Lema del lenguaje: “Write once, runeverywhere”.

Antes de que apareciera Java, el lenguaje C era uno de los más extendidos por su versatilidad. Pero cuando los programas escritos en C aumentaban de volumen, su manejo comenzaba a complicarse. Mediante las técnicas de programación estructurada y programación modular se conseguían reducir estas complicaciones, pero no era suficiente. Fue entonces cuando la Programación Orientada a Objetos (POO) entra en escena, aproximando notablemente la construcción de programas al pensamiento humano y haciendo más sencillo todo el proceso. Los problemas se dividen en objetos que tienen propiedades e interactúan con otros objetos. Así el programador puede centrarse en cada objeto para programar internamente los elementos y funciones que lo componen.

Las características principales de lenguaje Java se resumen a continuación:

- El código generado por el compilador Java es independiente de la arquitectura.
- Está totalmente orientado a objetos.
- Su sintaxis es similar a C y C++.
- Es distribuido, preparado para aplicaciones TCP/IP.
- Dispone de un amplio conjunto de bibliotecas.
- Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
- La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema

## 1.2 Historia de Java

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos estaba en que su diseño era muy cambiante. Para que un programa funcionara en el siguiente dispositivo aparecido, había que reescribir el código. La empresa Sun quería crear un lenguaje independiente del dispositivo. Pero no fue hasta 1995 cuando pasó a llamarse Java, dándose a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esa filosofía y su facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

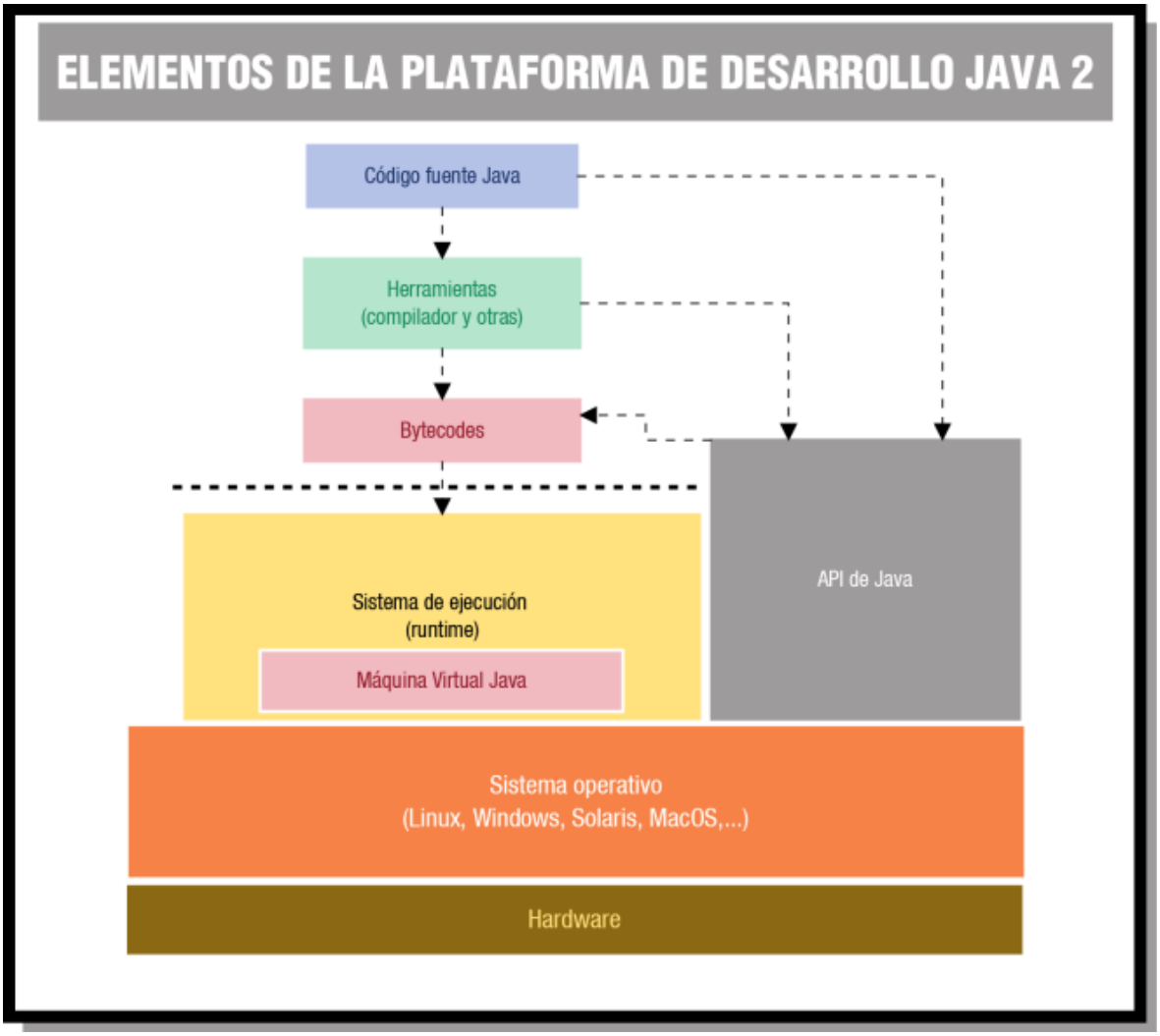
El factor determinante para su expansión fue la incorporación de un intérprete Java en la versión 2.0 del navegador Web Netscape Navigator, lo que supuso una gran revuelo en Internet. A principios de 1997 apareció Java 1.1, proporcionando sustanciales mejoras al lenguaje. Java 1.2, más tarde rebautizado como Java 2, nació a finales de 1998.

El principal objetivo del lenguaje Java es llegar a ser el nexo universal que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor Web, en una base de datos o en cualquier otro lugar.

Para el desarrollo de programas en lenguaje Java es necesario utilizar un entorno de desarrollo denominado JDK (Java Development Kit), que provee de un compilador y un entorno de ejecución (JRE – Java RunEnvironment) para los bytecodes generados a partir del código fuente. Igual que las diferentes versiones del lenguaje han incorporado mejoras, el entorno de desarrollo y ejecución también ha sido mejorado sucesivamente.

Java 2 es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluyendo los siguientes elementos:

- Un lenguaje de programación: Java.
- Un conjunto de bibliotecas estándar que vienen incluidas en la plataforma y que son necesarias en todo entorno Java. Es el Java Core.
- Un conjunto de herramientas para el desarrollo de programas, como es el compilador de bytecodes, el generador de documentación, un depurador, etc.
- Un entorno de ejecución, que en definitiva es una máquina virtual que ejecuta los programas traducidos a bytecodes.




Actualmente hay tres ediciones de la plataforma Java 2:

- J2SE: Entorno de Sun relacionado con la creación de aplicaciones y applets en lenguaje Java.
- J2EE: Pensada para la creación de aplicaciones Java empresariales y del lado del servidor.
- J2ME: Pensada para la creación de aplicaciones Java para dispositivos móviles.

## 1.3 La Programación Orientada a Objetos (POO) y Java

En Java, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. El objeto tendrá un comportamiento (su código interno) y un estado (los datos). Los objetos permiten la reutilización del código y pueden considerarse, en sí mismos, como piezas reutilizables en múltiples proyectos distintos. Esta característica permite reducir el tiempo de desarrollo de software.

Por simplificar un poco las cosas, un programa en Java será como una representación teatral en la que debemos preparar primero cada personaje, definir sus características y qué va a saber hacer. Cuando esta fase esté terminada, la obra se desarrollará sacando personajes a escena y haciéndoles interactuar.

Al emplear los conceptos de la Programación Orientada a Objetos (POO), Java incorpora las tres características propias de este paradigma: **encapsulación** (En programación modular y más específicamente en programación orientada a objetos, se denomina así al ocultamiento de los datos y elementos internos de un objeto. Sólo se puede modificar un objeto a través de las operaciones definidas para éste.), **herencia** (mecanismo que permite derivar una clase de otra, de manera que extienda su funcionalidad) y **polimorfismo** (capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente). Por ejemplo, podemos crear dos clases distintas: Pez y Ave que heredan de la superclase Animal. La clase Animal tiene el método abstracto mover que se implementa de forma distinta en cada una de las subclases (los peces se mueven nadando, mientras que las aves lo hacen volando).

Los patrones o tipos de objetos se denominan **clases** (Es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y el comportamiento que todos los objetos de la clase comparten) y los **objetos** que utilizan estos patrones o pertenecen a dichos tipos, se identifican con el nombre de instancias.

Una **instancia** se produce con la creación de un objeto perteneciente a una clase (se dice que se instancia la clase). El objeto que se crea tiene los atributos, propiedades y métodos de la clase a la que pertenece. Los objetos y sus características se usan en la construcción de programas, ya sea como contenedores de datos o como partes funcionales del programa.)

## 1.4 Fortalezas de Java: Independencia de la plataforma , trabajo en red, seguridad y simplicidad.

Las dos principales características que distinguen a Java de otros lenguajes son la independencia de la plataforma y la posibilidad de trabajar en red o, mejor, la posibilidad de crear aplicaciones que trabajan en red. Adicionalmente, Java ofrece dos características muy valiosas, su seguridad y simplicidad:

a) **Independencia**: Los programas escritos en Java pueden ser ejecutados en cualquier tipo de hardware. El código fuente es compilado, generándose el código conocido como Java Bytecode (instrucciones máquina simplificadas que son específicas de la plataforma Java), el bytecode será interpretado y ejecutado en la Máquina Virtual Java (MVJ o JVM – Java Virtual Machine) que es un programa escrito en código nativo de la plataforma destino entendible por el hardware. Con esto se evita tener que realizar un programa diferente para cada CPU o plataforma. Por tanto, la parte que realmente es dependiente del sistema es la Máquina Virtual Java, así como las librerías o bibliotecas básicas que permiten acceder directamente al hardware de la máquina.

b) **Trabajo en red**: Esta capacidad del lenguaje ofrece múltiples posibilidades para la comunicación vía TCP/IP. Para poder hacerlo existen librerías que permiten el acceso y la interacción con protocolos como http, ftp, etc., facilitando al programador las tareas del tratamiento de la información a través de redes.

c) **Seguridad**: En primer lugar, los posibles accesos a zonas de memoria “sensibles” que en otros lenguajes como C y C++ podían suponer peligros importantes, se han eliminado en Java. En segundo lugar, el código Java es comprobado y verificado para evitar que determinadas secciones del código produzcan efectos no deseados. Los test que se aplican garantizan que las operaciones, operandos, conversiones, uso de clases y demás acciones son seguras. Y en tercer lugar, Java no permite la apertura de ficheros en la máquina local, tampoco permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra. En definitiva, podemos afirmar que Java es un lenguaje seguro.

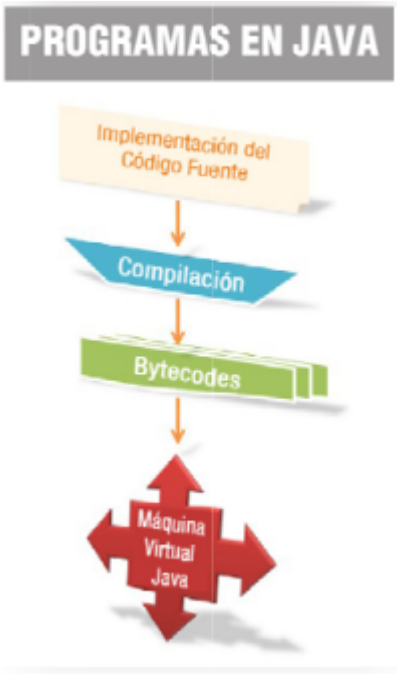
d) **Simplicidad**: Aunque Java es tan potente como C o C++, es bastante más sencillo. Posee una curva de aprendizaje muy rápida y, para alguien que comienza a programar en este lenguaje, le resulta relativamente fácil comenzar a escribir aplicaciones interesantes. En relación a C o C++, Java es algo más sencillo de entender porque elimina la aritmética de punteros (un puntero o apuntador es una variable que referencia una región de memoria; en otras palabras es una variable cuyo valor es una dirección de memoria), los registros, la definición de tipos, la gestión de memoria, etc., lo que reduce la posibilidad de cometer errores comunes

en los programas. Muy relacionado con la simplicidad que aporta Java está la incorporación de un elemento muy útil: el **Recolector de Basura** (Garbage collector). Permite al programador liberarse de la gestión de la memoria y hace que ciertos bloques de memoria puedan reaprovecharse, disminuyendo el número de huecos libres (fragmentación de memoria o memoria que queda desperdiciada al usar los métodos de gestión de memoria. Puede ser interna o externa). Cuando realicemos programas, crearemos objetos, haremos que éstos interaccionen, etc. Todas estas operaciones requieren de uso de memoria del sistema, pero la gestión de ésta será realizada de manera transparente al programador. Todo lo contrario que ocurría en otros lenguajes. Podremos crear tantos objetos como solicitemos, pero nunca tendremos que destruirlos. El entorno de Java borrará los objetos cuando determine que no se van a utilizar más. Este proceso es conocido como recolección de basura.

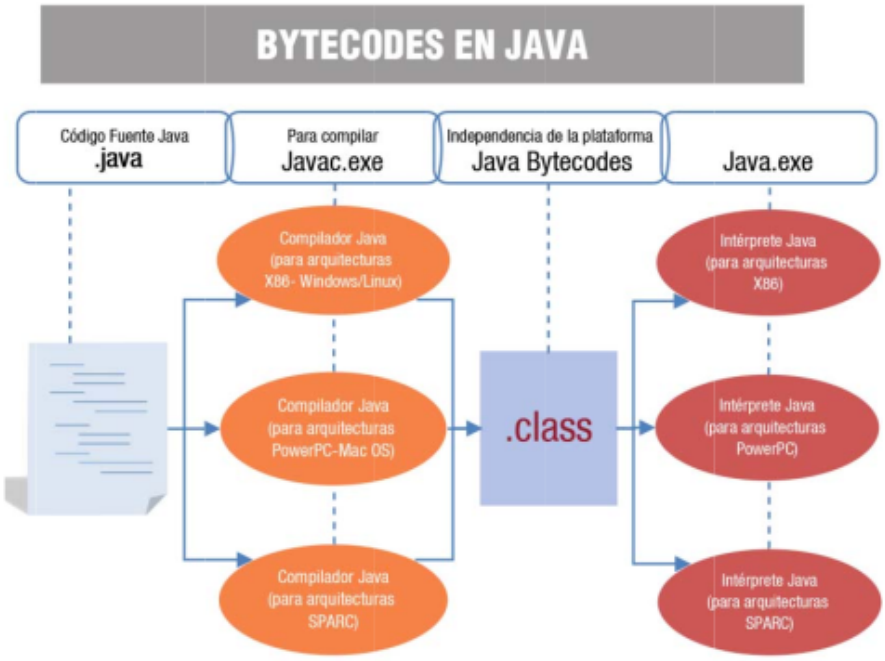
## 1.5 Los Bytecodes

Un programa escrito en Java no es directamente ejecutable. Es necesario que el código fuente sea interpretado por la Máquina Virtual Java. A continuación se detallan los pasos que sigue este proceso:

- Una vez escrito el código fuente (archivo con extensión .Java), se precompila generando el Bytecode o archivo compatible con el intérprete de la JVM. Este intérprete genera el código nativo para la plataforma sobre la que se ejecuta el programa.
- En el proceso de precompilación, existe un verificador de código de bytes que se asegura de que se cumplen ciertas condiciones.
  - El código satisface las especificaciones de la Máquina Virtual Java.
  - No existe amenaza contra la integridad del sistema.
  - No se producen desbordamientos de memoria.
  - Los parámetros y sus tipos son adecuados.
  - No existen conversiones de datos no permitidas.




Para que un bytecode pueda ser ejecutado en cualquier plataforma, es imprescindible que dicha plataforma cuente con el intérprete adecuado, es decir, la máquina virtual específica para esa plataforma. En general, la Máquina Virtual Java es un programa de reducido tamaño y gratuito para todos los sistemas operativos.




## 1.6 Tipos de aplicaciones en Java

La versatilidad del lenguaje de programación Java permite al programador crear distintos tipos de aplicaciones. A continuación, describiremos las características más relevantes de cada uno de ellos:

**Aplicaciones de consola:**

- Son programas independientes, como los creados con lenguajes tradicionales.
- Se componen como mínimo de un archivo .class que debe contar necesariamente con el método main.
- No necesitan un navegador web y se ejecutan cuando invocamos el comando Java para iniciar la Máquina Virtual de Java (JVM). De no encontrarse el método main la aplicación no podrá ejecutarse.
- Las aplicaciones de consola leen y escriben hacia y desde la entrada y salida estándar, sin ninguna interfaz gráfica de usuario.

**Aplicaciones gráficas:**

- Aquellas que utilizan las clases con capacidades gráficas, como Swing (biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE).
- Incluyen las instrucciones import, que indican al compilador de Java que las clases del paquete javax.swing se incluyan en la compilación.

**Applets:**

- Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador. Cuando el navegador carga una web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.
- Se pueden descargar de Internet y se observan en un navegador. Los applets se descargan junto con una página HTML desde un servidor web y se ejecutan en la máquina cliente.
- No tienen acceso a partes sensibles (por ejemplo: no pueden escribir archivos), a menos que uno mismo le dé los permisos necesarios en el sistema.
- No tienen un método principal.
- Son multiplataforma y pueden ejecutarse en cualquier navegador que soporte Java.

**Servlets:**

- Son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.
- Los servlets, al contrario de los applets, son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones Web que interactúen con los clientes.

**Midlets:**

- Son aplicaciones creadas en Java para su ejecución en sistemas de propósito simple como dispositivos móviles. Los juegos Java creados para teléfonos móviles son midlets.

# 2 Principios básicos de la programación orientada a objetos

Dentro de las distintas formas de hacer las cosas en programación, distinguimos dos paradigmas fundamentales:

- **Programación Estructurada**, en la que se crean funciones y procedimientos que definen las acciones a realizar, y que posteriormente forman los programas.
- **Programación Orientada a Objetos**, que considera los programas en términos de objetos y todo gira alrededor de ellos.

Pero ¿en qué consisten realmente estos paradigmas? Veamos estos dos modelos de programación con más detenimiento.

Inicialmente se programaba aplicando las técnicas de programación tradicional, también conocidas como Programación Estructurada. El problema se descomponía en unidades más pequeñas hasta llegar a acciones o verbos muy simples y fáciles de codificar. Por ejemplo, en la resolución de una ecuación de primer grado, lo que hacemos es

descomponer el problema en acciones más pequeñas o pasos diferenciados:

- Pedir valor de los coeficientes.
- Calcular el valor de la incógnita.
- Mostrar el resultado.

Si nos damos cuenta, esta serie de acciones o pasos diferenciados no son otra cosa que verbos; por ejemplo el verbo pedir, calcular, mostrar, etc.

Sin embargo, la Programación Orientada a Objetos aplica de otra forma diferente la técnica de programación "divide y vencerás". Para ello lo que hace es descomponer, en lugar de acciones, en objetos, propiedades y métodos. El principal objetivo sigue siendo descomponer el problema en problemas más pequeños, que sean fáciles de manejar y mantener, fijándonos en el escenario del problema e intentando reflejarlo en nuestro programa. Por ejemplo, podríamos pensar en un objeto "ecuación" con las propiedades "coeficientes" y el método "calcular resultado". Se dice que la Programación Orientada a Objetos aborda los problemas de una forma más natural, entendiendo como natural que está más en contacto con el mundo que nos rodea.

La Programación Estructurada se centra en el conjunto de acciones a realizar en un programa, haciendo una división de procesos y datos. La Programación Orientada a Objetos se centra en la relación que existe entre los datos y las acciones a realizar con ellos, encerrándolos en el concepto de **objeto**, tratando de realizar una abstracción lo más cercana al mundo real.

## 2.1 Beneficios de la Programación Orientada a Objetos

Las principales ventajas de la POO son:

- **Comprensión**. Los conceptos del espacio del problema se hayan reflejados en el código del programa, por lo que la mera lectura del código nos describe la solución del problema en el mundo real.
- **Modularidad**. Facilita la modularidad del código, al estar las definiciones de objetos en módulos o archivos independientes, hace que las aplicaciones estén mejor organizadas y sean más fáciles de entender.
- **Fácil mantenimiento**. Cualquier modificación en las acciones queda automáticamente reflejada en los datos, ya que ambos están estrechamente relacionados. Esto hace que el mantenimiento de las aplicaciones, así como su corrección y modificación sea mucho más fácil. Por ejemplo, podemos querer utilizar un algoritmo más rápido, sin tener que cambiar el programa principal. Por otra parte, al estar las aplicaciones mejor organizadas, es más fácil localizar cualquier elemento que se quiera modificar y/o corregir. Esto es importante ya que se estima que los mayores costes de software no están en el proceso de desarrollo en sí, sino en el mantenimiento posterior de ese software a lo largo de su vida útil.
- **Seguridad**. La probabilidad de cometer errores se ve reducida, ya que no podemos modificar los datos de un objeto directamente, sino que debemos hacerlo mediante

las acciones definidas para ese objeto. Imaginemos un objeto lavadora. Se compone de un motor, tambor, cables, tubos, etc. Para usar una lavadora no se nos ocurre abrirla y empezar a manipular esos elementos, ya que lo más probable es que se estropee. En lugar de eso utilizamos los programas de lavado establecidos. Pues algo parecido con los objetos, no podemos manipularlos internamente, sólo utilizar las acciones que para ellos hay definidas.

- **Reusabilidad**. Los objetos se definen como entidades reutilizables, es decir, que los programas que trabajan con las mismas estructuras de información, pueden reutilizar las definiciones de objetos empleadas en otros programas, e incluso las acciones definidas sobre ellos. Por ejemplo, podemos crear la definición de un objeto de tipo persona para una aplicación de negocios y deseamos construir a continuación otra aplicación, digamos de educación, en donde utilizamos también personas, no es necesario crear de nuevo el objeto, sino que por medio de la reusabilidad podemos utilizar el tipo de objeto persona previamente definido

## 2.2 Características de la Programación Orientada a Objetos

Cuando hablamos de Programación Orientada a Objetos, existen una serie de características que debe cumplir cualquier lenguaje. Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción**. Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la clase. Básicamente, **una clase es un tipo de dato que agrupa características comunes de un conjunto de objetos**. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global del conjunto. Por ejemplo, si pensamos en una clase Vehículo que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como Coche y Camión. Entonces se dice que Vehículo es una abstracción de Coche y de Camión.
- **Modularidad**. Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.
- **Encapsulación**. También llamada "ocultamiento de la información". La encapsulación o encapsulamiento es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que

contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto Persona y otro Coche. Persona se comunica con el objeto Coche para llegar a su destino, utilizando para ello las acciones que Coche tenga definidas como por ejemplo conducir. Es decir, Persona utiliza Coche pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.

- **Jerarquía**. Mediante esta propiedad podemos definir relaciones de jerarquías, entre clases y objetos. Las dos jerarquías más importantes son la jerarquía "es un", llamada generalización o especialización, y la jerarquía "es parte de", llamada agregación. Conviene detallar algunos aspectos:
  - La **generalización o especialización**, también conocida como herencia, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase CochedeCarreras a partir de la clase Coche, y así sólo tendremos que definir las nuevas características que tenga.
  - La **agregación,** también conocida como inclusión, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un Coche está formado por Motor, Ruedas, Frenos y Ventanas. Se dice que Coche es una agregación y Motor, Ruedas, Frenos y Ventanas son agregados de Coche.
- **Polimorfismo**. Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase Animal y la acción de expresarse. Nos encontramos que cada tipo de Animal puede hacerlo de manera distinta, los Perros ladran, los Gatos maullan, las Personas hablamos, etc.

# 3 Los objetos

## 3.1 Características de los Objetos. Estado, comportamiento e identidad.

Un objeto es un conjunto de datos con las operaciones definidas para ellos. Los objetos tienen un estado y un comportamiento.

Los objetos tienen unas características fundamentales que los distinguen:

- **Identidad**. Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- **Estado**. El estado de un objeto viene determinado por parámetros o atributos que lo describen y los valores de éstos. Por ejemplo, si tenemos un objeto Coche, el estado estaría definido por atributos como Marca, Modelo, Color, Cilindrada, etc.

- **Comportamiento**. Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto Coche, el el comportamiento serían acciones como: arrancar(), parar(), acelerar(), frenar(), etc.

## 3.2 Propiedades y métodos

Todo objeto tiene un estado y un comportamiento. Concretando un poco más, las partes de un objeto son:

- **Campos, Atributos o Propiedades**: Parte del objeto que almacena los datos. También se les denomina Variables Miembro. Estos datos pueden ser de cualquier tipo primitivo (boolean, char, int, double, etc) o ser a su vez ser otro objeto. Por ejemplo, un objeto de la clase Coche puede tener un objeto de la clase Ruedas.
- **Métodos o Funciones Miembro**: Parte del objeto que lleva a cabo las operaciones sobre los atributos definidos para ese objeto.

La idea principal es que el objeto reúne en una sola entidad los datos y las operaciones, y para acceder a los datos privados del objeto debemos utilizar los métodos que hay definidos para ese objeto. La única forma de manipular la información del objeto es a través de sus métodos. Es decir, si queremos saber el valor de algún atributo, tenemos que utilizar el método que nos muestre el valor de ese atributo. De esta forma, evitamos que métodos externos puedan alterar los datos del objeto de manera inadecuada. Se dice que los datos y los métodos están encapsulados dentro del objeto.

## 3.3 Interacción entre objetos

Dentro de un programa los objetos se comunican llamando a sus métodos. Los métodos están dentro de los objetos y describen el comportamiento de un objeto cuando recibe una llamada. En otras palabras, cuando un objeto, objeto1, quiere actuar sobre otro, objeto2, tiene que ejecutar uno de sus métodos. Entonces se dice que el objeto2 recibe un mensaje del objeto1.

Un **mensaje** es la acción que realiza un objeto. Un método es la función o procedimiento al que se llama para actuar sobre un objeto. Los distintos mensajes que puede recibir un objeto o a los que puede responder reciben el nombre de **protocolo** de ese objeto.

El proceso de interacción entre objetos se suele resumir diciendo que se ha "enviado un mensaje" (hecho una petición) a un objeto, y el objeto determina "qué hacer con el mensaje" (ejecuta el código del método). Cuando se ejecuta un programa se producen las siguientes acciones:

- Creación de los objetos a medida que se necesitan.
- Comunicación entre los objetos mediante el envío de mensajes unos a otros, o el usuario a los objetos.
- Eliminación de los objetos cuando no son necesarios para dejar espacio libre en la memoria del computador.

# 4 Clases, atributos y métodos

Hasta ahora hemos visto lo que son los objetos. Un programa informático se compone de muchos objetos, algunos de los cuales comparten la misma estructura y comportamiento. Si tuviéramos que definir la estructura y comportamiento cada vez que queremos crear un objeto, estaríamos utilizando mucho código redundante. Por ello lo que se hace es crear una clase, que es una descripción de un conjunto de objetos que comparten una estructura y un comportamiento común.

Y a partir de la clase, se crean tantas "copias" o "instancias" como necesitemos. Esas copias son los objetos de la clase.

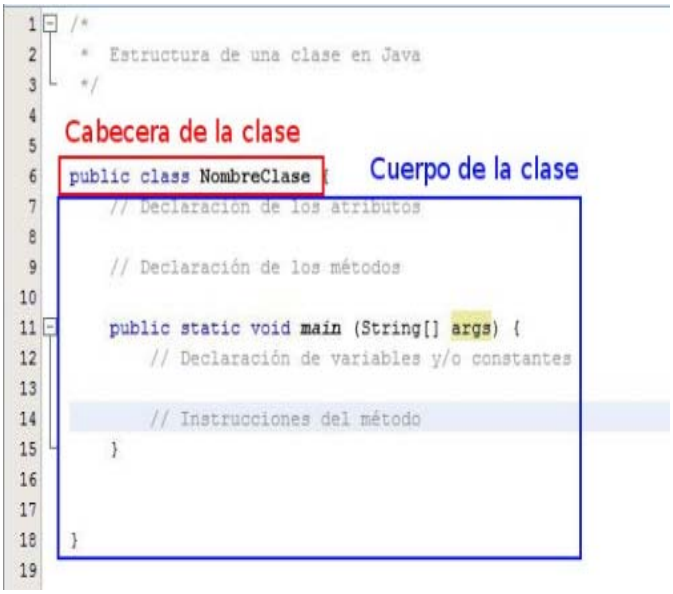
Las **clases** constan de datos y métodos que resumen las características comunes de un conjunto de objetos. Un programa informático está compuesto por un conjunto de clases, a partir de las cuales se crean objetos que interactúan entre sí.

En otras palabras, una clase es una plantilla o prototipo donde se especifican:

- Los **atributos** comunes a todos los objetos de la clase.
- Los **métodos** que pueden utilizarse para manejar esos objetos.

Para declarar una clase en Java se utiliza la palabra reservada **class**. La declaración de una clase está compuesta por:

- **Cabecera de la clase.** La cabecera es un poco más compleja que como aquí definimos, pero por ahora sólo nos interesa saber que está compuesta por una serie de modificadores, en este caso hemos puesto public que indica que es una clase pública a la que pueden acceder otras clases del programa, la palabra reservada class y el nombre de la clase.
- **Cuerpo de la clase.** En él se especifican encerrados entre llaves los atributos y los métodos que va a tener la clase.




El método **main()** se utiliza para indicar que se trata de una clase principal, a partir de la cual va a empezar la ejecución del programa. Este método no aparece si la clase que estamos creando no va a ser la clase principal del programa.

# 5 Encapsulación y visibilidad.

La **encapsulación** permite agrupar funcionalidades (métodos) y estado (datos) de una forma cohesiva. Los métodos proporcionan los mecanismos adecuados para modificar el estado y en algunos casos también serán la puerta de acceso a este. En lenguajes como Java, la forma de acceder al estado será mediante métodos tipo getXXX().

Cuando hablamos de **visibilidad** nos referimos a la ocultación o publicación tanto de la información del estado como de la implementación de la funcionalidad de una clase. La visibilidad nos permite ocultar al exterior detalles del estado o de la implementación que no queremos que sean expuestos fuera de nuestra "unidad de encapsulación".

Un ejemplo muy sencillo sería el siguiente: en un juego no queremos que se sepa la vida de los personajes, únicamente queremos que se sepa si el personaje está vivo, herido o muerto. Podríamos recurrir a una propiedad privada que almacene los puntos de vida del personaje y un método público que nos permita obtener su estado. El método calcularía el estado en función de los puntos de vida del jugador pero los puntos de vida no quedarían expuestos en ningún momento.

El concepto de visibilidad suele acompañar al de encapsulación hasta el punto de que a veces se asume que encapsulación y visibilidad son sinónimos.

# 6 Relaciones entre clases. Herencia.

## 6.1 Relaciones entre clases y objetos

Las clases, igual que los objetos, no existen de modo aislado. La Orientación a Objetos (POO) intenta modelar aplicaciones del mundo real tan fielmente como sea posible y por lo tanto debe reflejar estas relaciones entre clases y objetos. Veremos estas relaciones detalladamente en posteriores Unidades de Trabajo.

## 6.2 Herencia

De todas las relaciones posibles entre las distintas clases y objetos, hay que destacar por su importancia en O.O. la relación de herencia. La relación de herencia es una relación entre clases que comparten su estructura y el comportamiento.

- Se denomina **herencia simple** a la relación en que una clase comparte la estructura y comportamiento de una sola clase.
- Se denomina **herencia múltiple** a la relación en que una clase comparte la estructura y comportamiento de varias clases.

Para que un lenguaje de programación pueda ser considerado orientado a objetos, debe implementar el mecanismo de herencia.

La clase superior de la jerarquía, en cada relación, se denomina **superclase**, clase base ó clase padre y, la clase que hereda de la superclase, se denomina **subclase**, clase derivada ó clase hija.

La herencia es:

HERENCIA = TRANSMISIÓN + REDEFINICIÓN + ADICIÓN

Las implicaciones de la herencia sobre los objetos de las clases involucradas son las siguientes:

- Sobre los objetos de la superclase A: ninguna
- Sobre los objetos de la subclase B:
  - Contienen todos los atributos que contienen los objetos de la superclase.
  - Contiene los atributos añadidos de la subclase.
  - Responden a los mensajes que corresponden con métodos transmitidos a la subclase; es decir, como dicta el método de la superclase.
  - Responden a los mensajes que corresponden con métodos añadidos a la subclase; es decir, como dicta el método de la subclase.
  - Responden a los mensajes que corresponden con métodos redefinidos en la subclase; es decir, como dicta el método de la subclase anulando el método de la superclase.

**Sintaxis de la herencia en Java.**

```
modificador NombreSubClase extends NombreSuperClase{
                              …
}
```

La subclase puede redefinir tanto los atributos cómo los métodos heredados. Los métodos de la superclase redefinidos pueden ser utilizados a través de la referencia super en el método correspondiente (al principio):

```
super.métodoRedefinido(…)
```

# 7 Introducción a la API de Java

La **API (Application Programming Interface)** de Java contiene lo que se conoce como Bibliotecas de Clases Java. Este conjunto de bibliotecas proporciona al programador paquetes de clases útiles para la realización de múltiples tareas dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente (Referencia a los aspectos del significado, sentido o interpretación del significado de un determinado elemento, símbolo, palabra, expresión o representación formal. En principio cualquier medio de expresión (lenguaje formal o natural) admite una correspondencia entre expresiones de símbolos o palabras, y situaciones o conjuntos de cosas que se encuentran en el mundo físico o abstracto, que puede ser descrito por dicho medio de expresión).

En décadas pasadas una biblioteca era un conjunto de programas que contenían cientos de rutinas (una rutina es un procedimiento o función bien verificados, en determinado lenguaje de programación). Las rutinas de biblioteca manejaban las tareas que todos o

casi todos los programas necesitaban. El programador podía recurrir a esta biblioteca para desarrollar programas con rapidez.

Una biblioteca de clases es un conjunto de clases de programación orientada a objetos. Esas clases contienen métodos que son útiles para los programadores. En el caso de Java cuando descargamos el JDK obtenemos la biblioteca de clases API. Utilizar las clases y métodos de las APIs de Java reduce el tiempo de desarrollo de los programas. También, existen diversas bibliotecas de clases desarrolladas por terceros que contienen componentes reutilizables de software, y están disponibles a través de la Web.

Enlace a la API: https://docs.oracle.com/javase/7/docs/api/