



3. Utilización de objetos

Autor	(X) Xerach Casanova
Clase	Programación
Fecha	@Dec 15, 2020 10:25 AM

1. [Introducción](#)
2. [Fundamentos de la programación orientada a objetos](#)
 - 2.1. [Conceptos](#)
 - 2.2. [Beneficios](#)
 - 2.3. [Características.](#)
3. [Clases y objetos. Características de los objetos](#)
 - 3.1. [Propiedades y métodos de los objetos](#)
 - 3.2. [Interacción entre objetos](#)
 - 3.3. [Clases](#)
4. [Utilización de objetos](#)
 - 4.1. [Ciclo de vida de los objetos](#)
 - 4.2. [Declaración](#)
 - 4.3. [Instanciación](#)
 - 4.4. [Manipulación](#)
 - 4.5. [Destrucción de objetos y liberación de memoria](#)
5. [Utilización de métodos](#)
 - 5.1. [Parámetros y valores devueltos](#)
 - 5.2. [Constructores](#)
 - 5.3. [El operador This](#)
 - 5.4. [Métodos estáticos](#)
6. [Librerías de objetos \(paquetes\)](#)
 - 6.1. [Sentencia import](#)
 - 6.2. [Compilar y ejecutar clases con paquetes](#)
 - 6.3. [Jerarquía de paquetes](#)
 - 6.4. [Librerías java](#)
7. [Programación de la consola: entrada y salida de la información](#)
 - 7.1. [Conceptos sobre la clase System](#)
 - 7.2. [Entrada por teclado. Clase System.](#)
 - 7.3. [Entrada por teclado. Clase Scanner](#)
 - 7.4. [Salida por pantalla](#)
 - 7.5. [Salida de error.](#)

1. Introducción

Los programas son el resultado de la búsqueda y obtención de una solución para un problema del mundo real. La programación orientada a objetos (POO), establece una serie de técnicas que permiten trasladar problemas del mundo real a nuestro sistema informático.

2. Fundamentos de la programación orientada a objetos

Dentro de la programación se destacan dos paradigmas fundamentales:

Programación estructurada: se crean funciones y procedimientos que definen las acciones a realizar y que posteriormente forman los programas.

Programación orientada a objetos: considera los programas en términos de objetos y todo gira alrededor de ellos.

La programación estructurada consiste en descomponer el problema en unidades más pequeñas hasta llegar a acciones o verbos muy simples y fáciles de codificar. Un ejemplo sería resolver una ecuación de primer grado (pedir, calcular, mostrar):

1. Pedir valor de coeficientes
2. Calcular el valor de la incógnita
3. Mostrar el resultado.

La programación orientada a objetos, en lugar de descomponer en acciones, lo hace en objetos. Se trata de trasladar la visión del mundo real a nuestros programas.

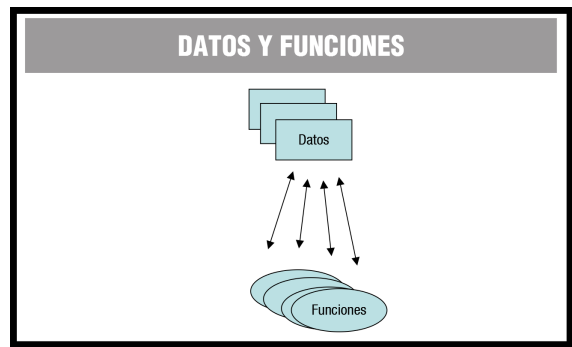
En definitiva, mientras la programación estructurada se centra en el conjunto de acciones a realizar en un programa, haciendo una división de procesos y datos, la programación orientada a objetos se centra en la relación que existe entre los datos y las acciones a realizar con ellos, y los encierra dentro del concepto objeto, tratando de realizar una abstracción lo más cercana al mundo real. Ambas no son excluyentes, ya que parte del paradigma orientado a objetos es implementado siguiendo los principios de la programación estructurada.

Se trata de representar entidades y objetos que nos encontramos en el mundo real mediante componentes de una aplicación.

2.1. Conceptos

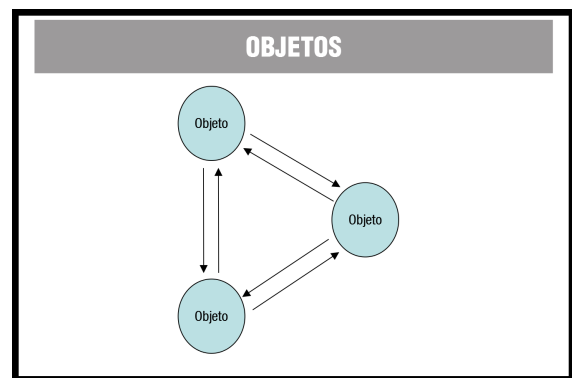
En la programación estructurada, dentro de las funciones se situaban las instrucciones del programa que manipulaban los datos. Funciones y datos se encontraban separados e independiente. Esto genera dos problemas:

- Los programas se creaban y estructuraban de acuerdo a la arquitectura del ordenador donde se van a ejecutar.
- Al estar todos los datos separados de las funciones, estos eran visibles en toda la aplicación y cualquier modificación de datos requería modificaciones en todas las funciones del programa.



En POO los objetos permiten mayor abstracción:

- El programador organiza su programa en objetos, representaciones del mundo real, más cercanas a la forma de pensar de las personas.
- Los datos y las funciones que los manipulan son parte interna de los objetos y no son accesibles al resto. Los cambios en los datos afectan solo a las funciones de ese objeto, pero no al resto de la aplicación.



La POO es la que presenta mayor facilidad para el desarrollo de programas basados en interfaces gráficas.

2.2. Beneficios

- **Comprensión:** Los conceptos del espacio del problema se refleja en el código, la lectura del código describe la solución del problema en el mundo real.
- **Modularidad:** aplicaciones mejor organizadas y más fáciles de entender al estar en módulos o archivos independientes.
- **Fácil mantenimiento.** Modificar las acciones hace que se refleje en los datos al estar estrechamente relacionados. Además, al estar las apps mejor organizadas es fácil de localizar cualquier elemento a modificar y corregir. El mayor coste de software se encuentra en el mantenimiento y no en su desarrollo.
- **Seguridad.** Se reduce la probabilidad de cometer errores, ya que no podemos modificar datos de objetos directamente, sino mediante acciones definidas para el objeto.
- **Reusabilidad.** Los objetos se definen como entidades reutilizables en programas que trabajan con la misma estructura de información.

2.3. Características.

- **Abstracción.** Definimos las características del objeto sin preocuparnos de como se escribirán en el código del programa. Simplemente se define de una forma general. En POO la herramienta de abstracción más importante es la clase, el cual es un tipo de dato que agrupa las características comunes de un conjunto de objetos. Por ejemplo. Pensando en una clase "vehículo" que agrupa las características comunes de todos ellos, podemos crear a partir de dicha clase objetos como coche, camión, moto... Vehículo es una abstracción de coche, camión y moto.
- **Modularidad.** Una vez representado el escenario del problema en nuestra aplicación, obtendremos un conjunto de objetos software a utilizar, que se crean a partir de una o varias clases. Cada clase es un archivo distinto, lo cual nos permite modificar las características de los objetos sin afectar al resto de clases.
- **Encapsulamiento (ocultación de la información).** Se ocultan las partes internas del objeto a los demás objetos y al mundo exterior, o se restringe el acceso para no ser manipulado de manera inadecuado. Por ejemplo. Persona y coche. El objeto persona utiliza el objeto coche para llegar a su destino utilizando sus acciones, pero no tiene por qué saber como funciona internamente.

- **Jerarquía.** Definimos jerarquías entre clases y objetos, las dos más importantes son: **"es un"** (generalización o especialización) y **"es parte de"** (llamada agregación).
 - **La generalización o especialización** es conocida como herencia, la cual permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases (múltiple). Ejemplo: Coche - Coche de carreras. Coche de carreras hereda las características de coche y además tiene las suyas propias.
 - **La agregación o inclusión,** permite agrupar objetos relacionados entre sí dentro de una clase. Por ejemplo, motor, ruedas, frenos y ventanas son agregados de coche

2.4. Lenguajes de programación orientados a objetos

- **Simula (1962).** El primero en introducir el concepto clase como elemento que incorpora datos y sus operaciones sobre estos. En 1967 surge Simula 67 con mayor número de tipos de datos.
- **SmallTalk (1972).** Basado en Simula 67, la primera fue SmallTalk 72 y la siguiente **SmallTalk 76.** Soporta propiedades de POO y posee un entorno que facilita el rápido desarrollo de aplicaciones. Su contribución más importante es el modelo-vista-controlador.
- **C++ (1985).** Diseñado por Bjarne Stroustrup Deriva de C y se añaden mecanismos que lo convierten en lenguaje orientado a objetos. No tiene recolector de basura automática. En este lenguaje aparece el concepto de clase tal y como los conocemos actualmente
- **Eiffel (1986).** Creado por Bertrand Meyer, sintaxis parecida a C, no logró la aceptación de C y Java.
- **Java (1995).** Diseñado por Gosling de Sun Microsystems, fue diseñado desde cero, su característica principal es que produce un bytecode que se interpreta en una máquina virtual.
- **C# (2000).** Creado por Microsoft como ampliación de C, basado en C++ y Java. Evita muchos problemas de diseño de C++.

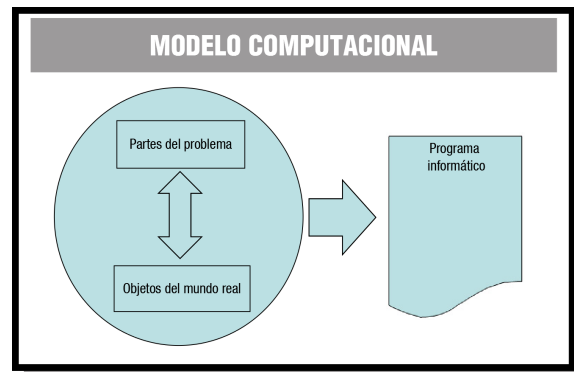
Lenguajes actuales no concebidos como POO, tales como JavaScript o PHP están migrando a POO.

3. Clases y objetos. Características de los objetos

Un objeto de software es una representación de un objeto en el mundo real, compuesto por una serie de características y comportamiento.

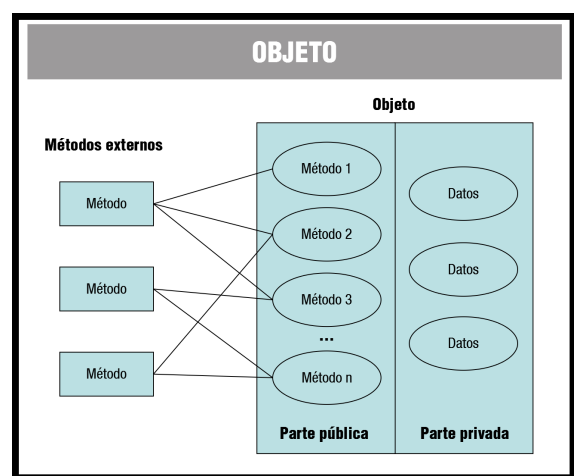
Es un conjunto de datos con las operaciones definidas para ellos. Los objetos tienen un estado y un comportamiento. Sus características son:

- **Identidad.** Permite diferenciar un objeto de otro, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Por ejemplo dos vehículos que salen de la misma cadena de montaje.
- **Estado.** Viene determinado por los parámetros o atributos que lo describen: marca, modelo, color, cilindrada...
- **Comportamiento.** Acciones que se pueden realizar sobre el objeto: arrancar(), parar(), acelerar(), frenar()...



3.1. Propiedades y métodos de los objetos

- **Campos, atributos o propiedades.** Almacena los datos del objeto, también llamadas variables miembro. Pueden ser de tipo primitivo, o ser a su vez otro objeto, por ejemplo en un objeto coche, puede ser la clase ruedas.
- **Métodos o funciones miembro.** Lleva a cabo las operaciones sobre los atributos definidos para ese objeto.



Un objeto debe reunir en una sola entidad los datos y operaciones. Para acceder a esos datos (privados) debemos utilizar los métodos definidos para el objeto. De esta manera evitamos que métodos externos puedan alterar datos del objeto de manera inadecuada (métodos encapsulados dentro del objeto).

3.2. Interacción entre objetos

Los objetos se comunican entre ellos llamando a sus métodos. Estos métodos describen el comportamiento de un objeto cuando recibe una llamada: Objeto1 ejecuta el método de objeto2 y recibe el Objeto2 recibe un mensaje de Objeto1. Estos mensajes que reciben los objetos o a los que pueden responder reciben el nombre de protocolo de objeto.

El proceso de interacción entre objetos consiste en "envío de mensaje (petición a un objeto) y ejecución del código del método (determina que hacer con el mensaje).

Un programa se ejecuta generando las siguientes acciones:

- Creación de objetos a medida que se necesitan.
- Comunicación entre objetos mediante envíos de mensajes o el usuario a los objetos.
- Eliminación de objetos cuando no son necesarios y así dejar espacio libre en memoria.

3.3. Clases

Una clase es la descripción de un conjunto de objetos que comparten estructura y comportamiento común. A partir de esa clase se crean copias o instancias.

Por ejemplo. Tenemos la clase vehículo con un conjunto de propiedades: marca, potencia, velocidad máxima y conjunto de métodos: aumentarVelocidad, reducirVelocidad, etc... Podemos crear a partir de ahí el objeto objBMW318, objMercedes220.

La clase no existe en memoria, solo define la estructura de los datos, cuando un objeto es instanciado, se reserva espacio en memoria para alojar sus datos.

Una clase se comporta como un tipo de datos y el objeto es una instancia de esos tipos de datos (una variable de esos tipos de datos).

Las clases constan de atributos (comunes a todos los objetos de esa clase) y métodos (que se utilizan para manejar esos objetos). Un programa informático se compone por un conjunto de clases, a partir de las cuales se crean objetos que interactúan entre sí.

La declaración de una clase está compuesta por:

- **Cabecera.** Compuesta por una serie de modificadores, como public (indica que es pública y se puede acceder desde otras clases), la palabra class y el nombre de la clase.
 - **Cuerpo de la clase,** se especifica entre llaves los atributos y métodos de la clase.

```
1  /*
2  * Estructura de una clase en Java
3  */
4
5  Cabecera de la clase
6  public class NombreClase {
7      // Declaración de los atributos
8
9      // Declaración de los métodos
10
11     public static void main (String[] args) {
12         // Declaración de variables y/o constantes
13
14         // Instrucciones del método
15     }
16 }
17
18
19
```

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.

4. Utilización de objetos

El concepto objeto e instancia de clase es lo mismo. Los objetos se crean a partir de clases y representan casos individuales de estas.

Por ejemplo, una clase persona reúne características comunes de las personas: color de pelo, ojos, peso, altura... y las acciones que pueden realizar: dormir, comer, crecer..., dentro del programa podemos crear un objeto trabajador que puede estar basado en la clase persona. **Trabajador es una instancia de la clase persona y la clase persona es una abstracción del objeto trabajador.**

Cada objeto tiene una zona de almacenamiento propia en memoria, donde se guarda su información que será distinta a la de cualquier otro objeto. A las

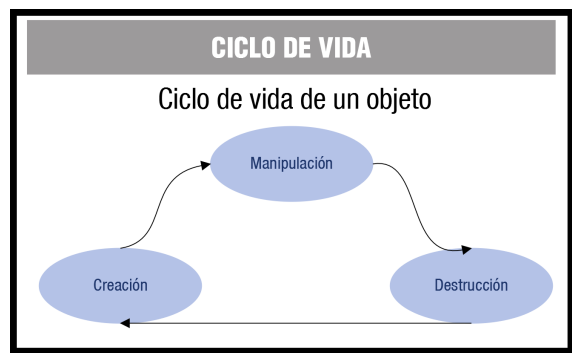
variables miembro instanciadas se les llama variables de instancia y a los métodos, métodos de instancia.

4.1. Ciclo de vida de los objetos

En java, todo programa parte de la clase principal, que ejecuta el contenido del método `main()`, el cual utilizará las demás clases del programa y lanzará mensajes a otros objetos.

Las instancias tienen un tiempo de vida determinado:

- Creación. Reserva de memoria e inicialización de atributos.
- Manipulación. Cuando se hace uso de los atributos y métodos.
- Destrucción. Eliminación y liberación de recursos.



4.2. Declaración

Para crear un objeto hay que declararlo (definir el tipo de objeto) e instanciarlo, con el operador `new`:

Para declarar un objeto se utiliza **<tipo> nombreobjeto;** - Por ejemplo:

```
Vehiculo coche;
```

Tipo es la clase a partir de la cual se creará el objeto y nombre_objeto es el nombre de la variable de referencia.

Los tipos referenciados o referencias se utilizan para guardar la dirección de los datos en memoria.

Cuando creamos una referencia, se encuentra vacía, no contiene ninguna instancia y su valor es `null`. La referencia está creada pero no el objeto.

Por ejemplo, en **`String mensaje;`** `String` es un tipo de dato referenciado en el que `String` es realmente la clase a partir de la cual creamos un objeto llamado `mensaje`. `Mensaje` aún no contiene el objeto porque no ha sido instanciado.

Cuando creamos un objeto hacemos uso de una variable que almacena la dirección de ese objeto en memoria. Esa variable es una referencia o tipo de dato referenciado, porque no contiene el dato sino la posición del mismo en memoria.

```
String saludo = new String ("Bienvenido a Java");

String s; //s vale null

s = saludo; //asignación de referencias
```

En las instrucciones anteriores, las variables `s` y `saludo` apuntan al mismo objeto de la clase `String`. **Esto implica que cualquier modificación en el objeto `saludo` modifica también el objeto al que hace referencia la variable `s`, ya que realmente son el mismo.**

Los nombres de la clase empiezan con mayúscula, como `String`, y los nombres de los objetos con minúscula, como `mensaje`, así sabemos qué tipo de elemento utilizando.

4.3. Instanciación

Cuando creamos la referencia al objeto debemos instanciarla. Para ello utilizamos `new`:

```
nombre_objeto = new <Constructor_de_la_Clase>([<par1>, <par2>, ..., <parN>]);
```

- Nombre es el objeto de la variable referencia.
- `new` es el operador que crea el objeto.
- constructor de la clase es el método especial de la clase que se llama igual que ella e inicia el objeto, dándole valores iniciales a sus atributos.
- `par1-parN` son parámetros que puede necesitar el constructor para dar valores iniciales a los atributos.

Cuando instanciamos el objeto se reserva memoria para él, de esta tarea se encarga java y el recolector de basura, que se encargará de eliminar de la memoria los objetos no utilizados.

Por ejemplo para `String` utilizaríamos lo siguiente:

```
mensaje = new String;
```

En este ejemplo, el objeto se crea con la cadena vacía (""), pero también podríamos darle contenido:

```
mensaje = new String ("El primer programa");
```

Java permite utilizar la clase String como si de un tipo de dato primitivo se tratara, por eso no hace falta utilizar el operador new para instanciar un objeto de la clase String.

Por último, debemos tener en cuenta que podemos declarar e instanciar un objeto en la misma instrucción:

```
String mensaje = new String ("El primer programa");
```

4.4. Manipulación

Una vez creado e instanciado podemos acceder a su contenido a través de sus atributos y métodos utilizando el nombre del objeto y el operador punto, seguido del nombre del atributo o método a utilizar.

Por ejemplo para acceder a las variables de instancia o atributos:

```
nombre_objeto.atributo;
```

Para acceder a métodos o funciones miembro del objeto:

```
nombre_objeto.metodo([par1, par2, ..., parN]);
```

par1, par2,... son parámetros que utiliza el método y son opcionales, dependiendo de si el método lo necesita o no.

Por ejemplo, para crear un rectángulo se utiliza la clase Rectangle, que se obtiene de la biblioteca o paquete de librería java.awt.

Instanciamos el objeto utilizando el método constructor, llamado igual que el objeto, indicando los parámetros correspondientes a la posición y dimensión del rectángulo:

```
Rectangle rect = new Rectangle(50, 50, 150, 150);
```

Si queremos cambiar el valor de los atributos utilizaremos:

```
rect.height = 100;  
rect.width = 100;
```

o podemos utilizar un método para hacer lo anterior:

```
rect.setSize(200,200);
```

```
public class Manipular {  
  
    public static void main(String[] args) {  
  
        // Instanciamos el objeto rect indicando posicion y dimensiones  
        Rectangle rect = new Rectangle( 50, 50, 150, 150 );  
  
        //Consultamos las coordenadas x e y del rectangulo  
        System.out.println( "----- Coordenadas esquina superior izqda. -----");  
        System.out.println("\tx = " + rect.x + "\n\ty = " + rect.y);  
  
        // Consultamos las dimensiones (altura y anchura) del rectangulo  
        System.out.println( "\n----- Dimensiones -----");  
        System.out.println("\tAlto = " + rect.height );  
        System.out.println( "\tAncho = " + rect.width);  
  
        //Cambiar coordenadas del rectangulo  
        rect.height=100;  
        rect.width=100;  
  
        rect.setSize(200, 200);  
        System.out.println( "\n-- Nuevos valores de los atributos --");  
        System.out.println("\tx = " + rect.x + "\n\ty = " + rect.y);  
        System.out.println("\tAlto = " + rect.height );  
        System.out.println( "\tAncho = " + rect.width);  
  
    }  
}
```

4.5. Destrucción de objetos y liberación de memoria

En java corre a cargo del recolector de basura (garbage collector). Es un sistema de destrucción automática de objetos que ya no son utilizados. Lo que hace es

liberar una zona de memoria reservada previamente con new.

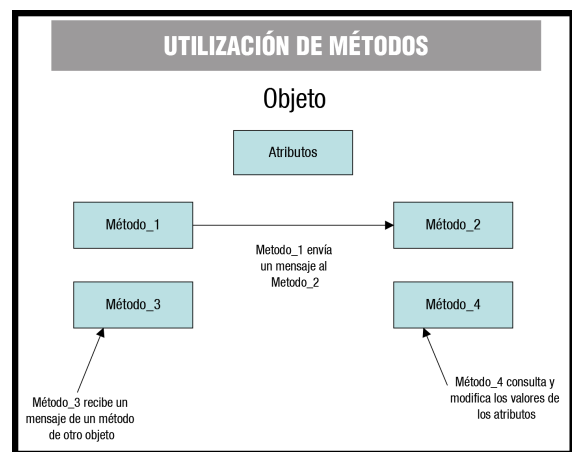
El recolector de basura se ejecuta en segundo plano y de manera muy eficiente para no afectar a la velocidad del programa. Periódicamente busca objetos que ya no son referenciado y cuando los encuentra los marca para ser eliminados, los cuales elimina en el momento que considere oportuno.

Un objeto eliminado se hace ejecutando el método finalize(), si por ejemplo queremos forzar la finalización de un objeto podemos utilizar el método runFinalization() de la clase System, la cual forma parte de la biblioteca de clases de Java y contiene diversas clases para entrada y salida de información, acceso a variables de entorno y otros métodos de utilidad.

5. Utilización de métodos

Los métodos contienen la declaración de variables locales y las operaciones que se pueden realizar para el objeto, que son ejecutadas cuando el método es invocado.

Se definen en el cuerpo de la clase y posteriormente son instanciados para convertirse en método instancia de objeto.



Los métodos se componen de cabecera y cuerpo. La cabecera también tiene modificadores. Dentro de un método nos encontramos el cuerpo, que contiene el código de la acción a realizar. Las acciones que un método puede realizar son:

- Inicializar atributos.
- Consultar valores de atributos.
- Modificar valores de atributos.
- Llamar a otros métodos del mismo objeto u objetos externos.

5.1. Parámetros y valores devueltos

Los métodos sirven para consultar información del objeto o para modificar su estado.

La información se devuelve a través de valores de retorno y la modificación se hace mediante la lista de parámetros.

La lista de parámetros se puede declarar de dos formas:

- **Por valor.** El valor no se devuelve al finalizar el método y la modificación de esos parámetros no tienen efecto una vez se salga de él. Dicho método recibe una copia de los argumentos, así que las modificaciones se hacen sobre la copia y no sobre las variables originales.
- **Por referencia.** Los valores de los parámetros sí tienen efecto tras la finalización del método. Cuando pasamos una variable a un método por referencia, en realidad estamos pasando la dirección de memoria del dato.

En java, todas las variables se pasan por valor, excepto los objetos, que se hace por referencia.

La declaración de un método en java tiene dos restricciones:

- **Un método siempre devuelve un valor** (no hay valor por defecto) y se llama valor de retorno. Se devuelve al método que lo llamó cuando este se termina de ejecutar. Puede ser tipo primitivo, referenciado o tipo void (que no devuelve ningún valor).
- **Tiene un número fijo de argumentos.** Los argumentos son variables a través que se pasan al método desde el lugar del que se llame, para que éste pueda utilizar sus valores. Reciben el nombre de parámetros cuando aparecen en la declaración del método.

La cabecera de un método se declara como sigue:

```
public tipo_de_dato_devuelto nombre_metodo (Tipo_Par1 NombrePar1, TipoPar2 NombrePar2...);
```

El tipo de dato que se devuelve se declara después del modificador (public) y se corresponde con el valor de retorno. La lista de parámetros aparece al final de la cabecera, encerrados entre paréntesis y separados por coma, indicando el tipo de dato.

La lista de argumentos en la llamada a un método debe coincidir en número, tipo y orden con los parámetros para que no produzca error.

5.2. Constructores

Un constructor es un método especial con el mismo nombre que la clase y que no devuelve ningún valor tras su ejecución.

Cuando instanciamos un objeto, el nombre de la clase a la que hacemos referencia realmente es el nombre del constructor, el cual es un método especial que sirve para inicializar valores.

Por ejemplo, la clase `Date` proporcionada por la biblioteca de clases de Java, se utiliza instanciando un objeto a partir de ella:

```
Date fecha = new Date();
```

Estamos creando un objeto `fecha` de tipo `Date`, el cual contiene la fecha y hora actual del sistema.

La estructura de los constructores es similar a la del método, salvo que no devuelve ningún valor. En el cuerpo, se contiene la inicialización de atributos y el resto de instrucciones del constructor.

- El constructor es invocado automáticamente en la creación de un objeto y solo una vez.
- No empiezan con minúscula al llamarse igual que la clase.
- Puede haber varios constructores en una sola clase.
- El constructor puede tener parámetros para definir qué valores dar a los atributos del objeto.
- El constructor por defecto es el que no tiene parámetros. Si no definimos un constructor por defecto e intentamos utilizarlo tendremos un error de compilación.
- Toda clase tiene al menos un constructor. Si no se define un constructor el compilador lo crea vacío por defecto e inicializa los atributos a sus valores

Estructura interna de un método constructor

```
public NombreClase (par1, par2, ..., parN)
{
    // Inicialización de atributos
    // Resto de instrucciones del constructor
} // Fin del método
```

parámetros opcionales

puede llamar a otros métodos de la clase aunque no es recomendable

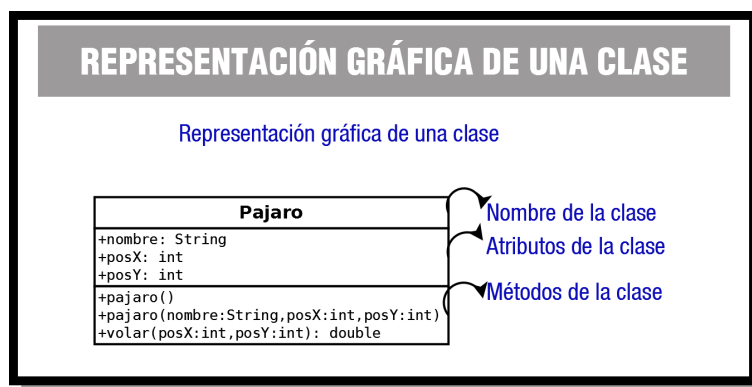
el modificador normalmente siempre es public

por defecto según el tipo que sean. El constructor al que se llama pertenece a su súper clase (clase de la que hereda) y si este no tiene constructor se produce error en la compilación.

5.3. El operador This

Constructores y métodos suelen utilizar este operador, que sirve para referirse a los atributos de un objeto cuando estamos dentro de él, sobre todo cuando existe ambigüedad entre el nombre de un parámetro y el de un atributo (`this.nombre_atributo`).

Ejemplo de utilización de objetos y métodos con uso de parámetros y operador `this`:



- `pajaro()` es el constructor por defecto, no contiene instrucción, java inicializa las variables miembro automáticamente si no les damos valor.
- `parajo(String nombre, int posX, int posY)` es el constructor que recibe como argumentos una cadena de texto y dos enteros para inicializar el valor de los atributos.
- `volar(int posX, int posY)`. El método recibe como argumentos los dos enteros y devuelve un valor de tipo `double` como resultado, usando la palabra clave `return`. Ese valor devuelto es el resultado de aplicar un desplazamiento de acuerdo a la siguiente fórmula:

$$\text{desplazamiento} = \sqrt{\text{posX} \cdot \text{posX} + \text{posY} \cdot \text{posY}}$$

1. Creamos la clase pájaro con sus métodos y atributos:


```

public class Pajaro {

    String nombre;
    int posX, posY;

    public Pajaro() {

    }

    public Pajaro(String nombre, int posX, int posY) {
        this.nombre=nombre;
        this.posX=posX;
        this.posY=posY;
    }

    double volar (int posX, int posY) {

        double desplazamiento = Math.sqrt( posX*posX + posY*posY );
        this.posX = posX;
        this.posY = posY;

        return desplazamiento;
    }
}

```

Creemos un método main() en el cual situamos el código de nuestro programa. Este código consiste en crear una instancia de la clase pajaro y ejecutar sus métodos: el constructor y el método volar(). También podemos imprimir el resultado del método (crear, invocar, imprimir)

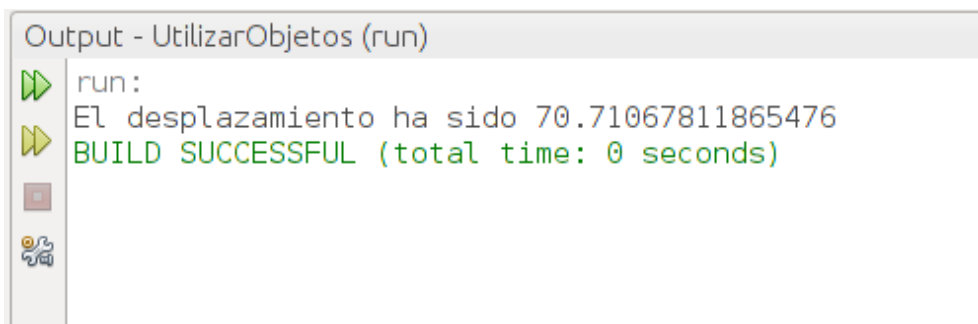
```

public static void main(String[] args) {

    Pajaro loro = new Pajaro("Lucy",50,50) ;
    double d = loro.volar(50,50);
    System.out.println("El desplazamiento ha sido " + d);
}

```

El resultado es:



```

Output - UtilizarObjetos (run)
run:
El desplazamiento ha sido 70.71067811865476
BUILD SUCCESSFUL (total time: 0 seconds)

```

5.4. Métodos estáticos

Los métodos estáticos son aquellos métodos definidos para una clase que se pueden utilizar sin necesidad de crear un objeto de dicha clase. También se llaman métodos de clase, por ejemplo los métodos de String son estáticos.

Para llamarlos utilizamos:

- Nombre del método si lo llamamos desde la misma clase en la que está definido.
- Nombre de la clase, seguido del operador punto, más el método estático, si lo llamamos desde una clase distinta.
- Nombre del objeto, seguido del operador punto, más el nombre del método estático, cuando tengamos un objeto instanciado de la clase en la que se encuentra un método estático.

Estos métodos no afectan al estado de los objetos instanciados (variables instancia) y suelen utilizarse para realizar operaciones comunes a todos los objetos de la clase. Sirve, por ejemplo para contar el número de objetos instanciados de una clase, a través de una variable entera de la clase que se incrementa conforme se van creando objetos.

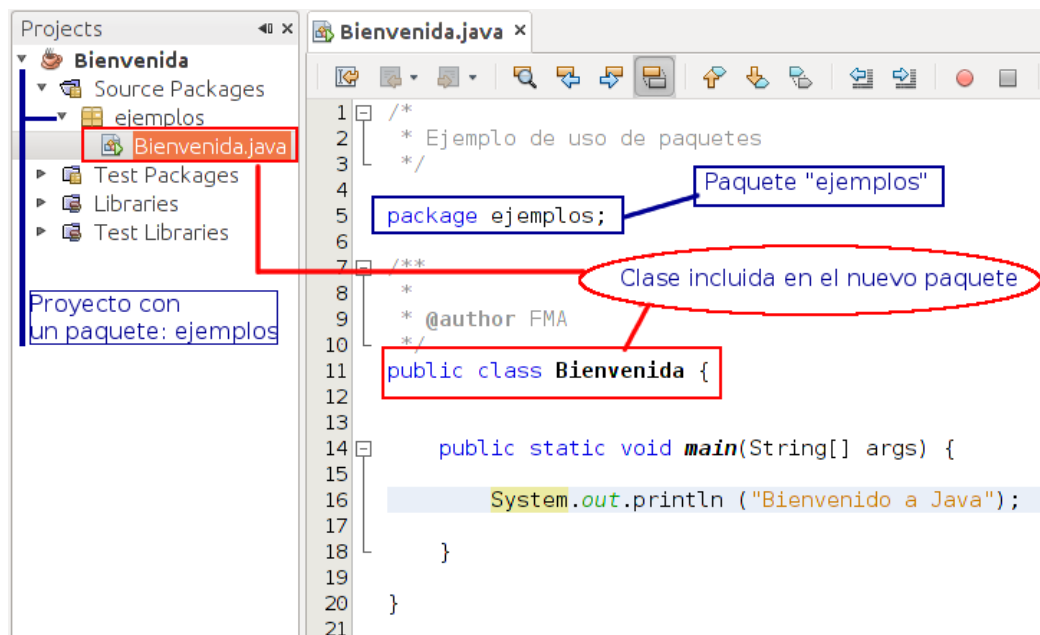
6. Librerías de objetos (paquetes)

Un paquete de clases es una agrupación de clases que consideramos que están relacionadas entre sí o tratan un tema común.

Las clases de un mismo paquete tienen acceso privilegiado a los atributos y métodos de otras clases del mismo paquete. Por ello, se considera a los paquetes, en cierto modo, unidades de encapsulación y ocultación de información.

Cada fichero .java se crea dentro del paquete que indiquemos. Los paquetes se declaran utilizando la palabra clave package, seguida del nombre del paquete:

```
package nombre_de_paquete;
```



En Netbeans se pueden crear paquetes en el paquete raíz "Source Packages" y dentro de un paquete ya existente.

6.1. Sentencia import

Para utilizar una clase que está en un paquete distinto a la clase que estamos utilizando, usamos la sentencia import. Por ejemplo, la clase Scanner se encuentra en el paquete java.util de la biblioteca de clases java. Podremos importar esa clase, o todas las clases del paquete:

```
import java.util.scanner;
import java.util.*;
```

Esta sentencia debe aparecer después de la sentencia package, si existiese.

También se puede utilizar la clase sin la sentencia import, pero debemos indicar su ruta completa cada vez que queramos utilizarla:

```
java.util.Scanner teclado = new java.util.Scanner(System.in);
```

Trabajar con paquetes implica organizar directorios, compilar y ejecutar de cierta forma para que todo funcione bien.

6.2. Compilar y ejecutar clases con paquetes

Por ejemplo, para el archivo `Bienvenida.java`, que pertenece al paquete `ejemplos`, se debe crear el subdirectorio `ejemplos` y meter dentro el archivo.

Debemos prestar atención a las mayúsculas y minúsculas en las carpetas, utilizando el mismo nombre para todo.

Para compilar la clase `Bienvenida.java` debemos situarnos en el directorio padre del paquete y compilar desde ahí. Ejemplo estando en linux, teniendo la siguiente ruta:

```
/<directorio_usuario>/Proyecto_Bienvenida/ejemplos/Bienvenida.java
```

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
$ javac ejemplos/Bienvenida.java
```

Si todo va bien, en el directorio `ejemplos` nos aparece la clase compilada `Bienvenida.class`. El nombre del paquete es: `paquete.clase`, es decir: `ejemplos.Bienvenida`.

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
$ java ejemplos/Bienvenida

Bienvenido a Java
```

6.3. Jerarquía de paquetes

Un paquete puede contener subpaquetes y así sucesivamente. Esto permite agrupar paquetes relacionados en un paquete más grande.

A nivel de sistema operativo, debemos crear subdirectorios con los nombres de los subpaquetes y meter dentro las clases que correspondan a cada una.

Para compilar, en el directorio del proyecto habría que compilar poniendo toda la ruta hasta llegar a la clase. Por ejemplo, teniendo el paquete "basicos" dentro del paquete `ejemplos`, la estructura de una clase `HolaMundo` dentro de este paquete sería: **`ejemplos.basicos.HolaMundo`** y la ruta en S.O. sería:

```
/<directorio_usuario>/Proyecto_Bienvenida/ejemplos/basicos/HolaMundo.java
```

Su compilación y ejecución sería:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
```

```
$ javac ejemplos/basicos/HolaMundo.java

$ java ejemplos/basicos/HolaMundo

Hola Mundo
```

La biblioteca de clases de Java se organiza haciendo uso de esta jerarquía. Ejemplo, para la clase `Date`, la cual se encuentra dentro del paquete `util` que a su vez está dentro del paquete `java`.

```
import java.util.Date;
```

6.4. Librerías java

En el entorno de compilación y ejecución de Java, tenemos incluida la API de Java.

Utilizar las clases y métodos de esta biblioteca nos ayuda a reducir tiempo de desarrollo, es importante saber consultarla y conocer sus clases más utilizadas. Los paquetes más importantes son:

- **java.io.** Gestionan entrada y salida para manipular ficheros, leer o escribir en pantalla, en memoria, etc...
- **java.lang.** Contiene las clases básicas del lenguaje, no es necesario importarlo, ya que se importa automáticamente por el entorno de ejecución. En él se encuentra la clase `Object`, que es la raíz de jerarquía de clases en Java, o `System`, también se encuentran las clases que envuelven tipos primitivos de datos.
- **java.util.** De utilidad general para el programador, por ejemplo, la clase `Scanner`, la clase `Date`, etc...
- **java.math.** Herramientas de manipulaciones matemáticas
- **java.awt.** Clases relacionadas con la construcción de interfaces de usuario: ventanas, cajas de texto, botones...
- **java.swing.** Clases de construcción de interfaces avanzadas. Alternativa más potente que `awt`.
- **java.net.** Clases para programación en red local e internet.
- **java.sql** para programar acceso a bases de datos.
- **java.security** para implementar mecanismos de seguridad.

7. Programación de la consola: entrada y salida de la información

Los programas a veces necesitan acceder a recursos del sistema, como por ejemplo la entrada/salida estándar, para recoger datos del teclado y mostrarlos por pantalla.

La entrada por teclado y la salida por pantalla se realiza con la clase `System` del paquete `java.lang`.

Los atributos de `System` son tres objetos que se utilizan para entrada y salida estándar.

- `System.in` entrada estándar: teclado
- `System.out` salida estándar: pantalla
- `System.err` salida de error estándar, también por pantalla pero con un fichero distinto para distinguir la salida normal del programa de los mensajes de error y poder mostrarlos.

No se puede crear objetos con `System`, se utiliza directamente utilizando punto para llamar a sus métodos:

```
System.out.println("Bienvenido a Java");
```

7.1. Conceptos sobre la clase `System`

`System.in` es un atributo de la clase `System`, pero consultando la biblioteca de clases, observamos que es un objeto y como tal, debe ser instanciado, volviendo a consultar la biblioteca, `System.in` es una instancia de una clase de java llamada `InputStream`

Field Summary	
Fields	
Modifier and Type	Field and Description
static <code>PrintStream</code>	<code>err</code> The "standard" error output stream.
static <code>InputStream</code>	<code>in</code> The "standard" input stream.
static <code>PrintStream</code>	<code>out</code> The "standard" output stream.

7.2. Entrada por teclado. Clase `System`.

`InputStream` permite leer bytes, desde el teclado, un archivo o cualquier dispositivo de entrada. También podremos utilizar desde esta clase el método

`read()` que permite leer un byte de la entrada o `skip(long n)` que salta `n` bytes de la entrada.

Pero lo realmente interesante es poder leer texto o números y se utilizan estas clases:

- **InputStreamReader** convierte bytes leídos en caracteres y sirve para convertir el objeto `System.in` en otro tipo de objeto que permita leer caracteres
- **BufferedReader**. Lee hasta un fin de línea, la cual tiene un método (`readLine()`) que nos permite leer caracteres hasta el final de línea.

```
InputStreamReader isr = new InputStreamReader(System.in);  
  
BufferedReader br = new BufferedReader (isr);
```

En el código anterior hemos creado un `InputStreamReader` a partir de `System.in` y pasamos dicho `InputStreamReader` al constructor de `BufferedReader`. El resultado es que las lecturas que hagamos con el objeto `br` son en realidad realizadas sobre `System.in`, pero con la ventaja de que podemos leer una línea completa. Así, por ejemplo, si escribimos una A, con:

```
String cadena = br.readLine();
```

Obtendremos en cadena una "A"

Sin embargo, seguimos necesitando hacer la conversión si queremos leer números. Por ejemplo, si escribimos un entero 32, en cadena obtendremos "32". Si recordamos, para convertir cadenas de texto a enteros se utiliza el método estático `parseInt()` de la clase `Integer`, con lo cual la lectura la haríamos así:

```
int numero = Integer.parseInt (br.readLine());
```

A continuación, ejemplo completo, en el que se incluye una excepción por si falla algo. Se captura esa excepción y avisamos al usuario de lo que ha pasado. Eso se realiza con `try {}`, colocando entre llaves el código que puede fallar y `catch {}` colocando el tratamiento de la excepción.

```

1 import java.io.BufferedReader;
  import java.io.IOException;
  import java.io.InputStreamReader;

  /*
   * Ejemplo de entrada por teclado con la clase System
   */

1 /**
   *
   * @author FMA
   */
  public class entradateclado {
1 |   public static void main(String[] args) {
      try
      {
          InputStreamReader isr = new InputStreamReader(System.in);
          BufferedReader br = new BufferedReader(isr);

          System.out.print("Introduce el texto: ");
          String cad = br.readLine();

          //salida por pantalla del texto introducido
          System.out.println(cad);

          System.out.print("Introduce un numero: ");
          int num = Integer.parseInt(br.readLine());

          // salida por pantalla del numero introducido
          System.out.println(num);

          } catch (Exception e) {
              // System.out.println("Error al leer datos");
              e.printStackTrace();
          }
      }
  }
}

```

7.3. Entrada por teclado. Clase Scanner

La entrada y salida por System tiene el inconveniente de solo poder leer de manera fácil datos String. En cambio, la clase Scanner permite leer distintos tipos de datos: String, int, long, etc:

```

Scanner teclado = new Scanner (System.in);

int i = teclado.nextInt ();

```

O para una línea completa de texto, número o lo que sea:


```
String cadena = teclado.nextLine();
```

Ejemplo:

```
import java.util.Scanner;
/*
 * Ejemplo de entrada de teclado con la clase Scanner
 */

/**
 *
 * @author FMA
 */
public class EntradaTecladoScanner {

    public static void main(String[] args) {

        // Creamos objeto teclado
        Scanner teclado = new Scanner(System.in);

        // Declaramos variables a utilizar
        String nombre;
        int edad;
        boolean estudias;
        float salario;

        // Entrada de datos
        System.out.println("Nombre: ");
        nombre=teclado.nextLine();
        System.out.println("Edad: ");
        edad=teclado.nextInt();
        System.out.println("Estudias: ");
        estudias=teclado.nextBoolean();
        System.out.println("Salario: ");
        salario=teclado.nextFloat();

        // Salida de datos
        System.out.println("Bienvenido: " + nombre);
        System.out.println("Tienes: " + edad + " años");
        System.out.println("Estudias: " + estudias);
        System.out.println("Tu salario es: " + salario + " euros");
    }
}
```

7.4. Salida por pantalla

La salida por pantalla se hace con el objeto `System.out`. El cual es una instancia de la clase `PrintStream` del paquete `java.lang`. Entre sus métodos podemos destacar:

- **void print(String s):** escribe una cadena de texto.
- **void println(String x):** escribe una cadena de texto y termina la línea.

java.io

Class PrintStream

java.lang.Object

java.io.OutputStream

java.io.FilterOutputStream

java.io.PrintStream

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

LogStream

- **void printf(String format, Object... args)** escribe una cadena de texto utilizando formato.

Para concatenar mensajes con valores de variables se utiliza el operador de concatenación +

```
System.out.println("Bienvenido, " + nombre);
```

La orden printf() utiliza unos códigos de conversión para indicar el tipo que es. Por ejemplo:

- **%c** escribe un carácter.
- **%s** escribe una cadena de texto.
- **%d** escribe un entero.
- **%f** escribe un número en punto flotante.
- **%e** escribe un número en punto flotante en notación científica.

Si por ejemplo queremos escribir el número float 12345.1684 con el punto de los miles y solo dos cifras sería:

```
System.out.printf("% ,.2f\n", 12345.1684);
```

Esta orden escribiría el número 12.345,17 por pantalla.

7.5. Salida de error.

Es una instancia de la clase PrintStream, así que podemos utilizar los mismos métodos.

En la consola de varios entornos integrados de desarrollo la salida de err se ve de un color distinta a la de out:

```
System.out.println("Salida estándar por pantalla");
System.err.println("Salida de error por pantalla");
```

Mapa conceptual

