

El lenguaje de programación de Java es...

- **Uso general:** está diseñado para ser utilizado para escribir software en una amplia variedad de dominios de aplicación y carece de funciones especializadas para cualquier dominio específico.
- **Basado en clase:** su estructura de objeto se define en clases. Las instancias de clase siempre tienen esos campos y métodos especificados en sus definiciones de clase (ver [Clases y Objetos](#)). Esto contrasta con lenguajes no basados en clases como JavaScript.
- **Escritura estática:** el compilador verifica en el momento de la compilación que se respetan los tipos de variables. Por ejemplo, si un método espera un argumento de tipo `String`, ese argumento debe ser de hecho una cadena cuando se llama al método.
- **Orientado a objetos:** la mayoría de las cosas en un programa Java son instancias de clase, es decir, paquetes de estado (campos) y comportamiento (métodos que operan sobre datos y forman la *interfaz* del objeto con el mundo exterior).
- **Portátil:** se puede compilar en cualquier plataforma con `javac` y los archivos de clase resultantes se pueden ejecutar en cualquier plataforma que tenga una JVM.

Creando tu primer programa Java

Cree un nuevo archivo en su [editor de texto](#) o [IDE](#) llamado `HelloWorld.java`. Luego pegue este bloque de código en el archivo y guarde:

```
public class HelloWorld {  
    public static void main(String[] args)  
    {  
        System.out.println("Hello, World!");  
    }  
}
```

El programa "Hello World" contiene un solo archivo, que consiste en una definición de clase `HelloWorld`, un método `main` y una declaración dentro del método `main`.

```
public class HelloWorld {
```

La palabra clave de `class` comienza la definición de clase para una clase llamada `HelloWorld`.

Cada aplicación Java contiene al menos una definición de clase ([Más información sobre clases](#)).

```
public static void main(String[] args) {
```

Este es un método de punto de entrada (definido por su nombre y firma de `public static void main(String[])`) desde el cual JVM puede ejecutar su programa. Cada programa de Java debería tener uno. Es:

- `public`: lo que significa que el método también se puede llamar desde cualquier lugar desde fuera del programa. Ver [Visibilidad](#) para más información sobre esto.
- `static`: significa que existe y se puede ejecutar por sí mismo (a nivel de clase sin crear un objeto).
- `void`: significa que no devuelve ningún valor. **Nota:** *Esto es diferente a C y C++ donde se espera un código de retorno como `int` (la forma de Java es `System.exit()`).*

Este método principal acepta:

- Una [matriz](#) (normalmente llamada `args`) de `String` `s` pasa como argumentos a la función principal (por ejemplo, desde los [argumentos de la línea de comando](#))

Casi todo esto es necesario para un método de punto de entrada de Java.

Piezas no requeridas:

- El nombre `args` es un nombre de variable, por lo que puede llamarse como quieras, aunque normalmente se llama `args`.
- Si su tipo de parámetro es una matriz (`String[] args`) o [Varargs](#) (`String... args`) no importa porque las matrices se pueden pasar a `varargs`.

Nota: una sola aplicación puede tener varias clases que contengan un método de punto de entrada (`main`). El punto de entrada de la aplicación está determinado por el nombre de clase pasado como un argumento al comando `java`.

Dentro del método principal, vemos la siguiente declaración:

```
System.out.println("Hello, World!");
```

Vamos a desglosar esta declaración elemento por elemento:

Elemento	Propósito
<code>System</code>	esto denota que la siguiente expresión llamará a la clase <code>System</code> , desde el paquete <code>java.lang</code> .
Elemento	Propósito
<code>.</code>	este es un "operador de puntos". Los operadores de puntos le brindan acceso a una clase de miembros ¹ ; Es decir, sus campos (variables) y sus métodos. En este caso, este operador de punto le permite hacer referencia al campo estático de <code>out</code> dentro de la clase <code>System</code> .
<code>out</code>	este es el nombre del campo estático del tipo <code>PrintStream</code> dentro de la clase <code>System</code> contiene la funcionalidad de salida estándar.
<code>.</code>	este es otro operador de puntos. Este operador de puntos proporciona acceso al método <code>println</code> dentro de la variable <code>out</code> .
<code>println</code>	este es el nombre de un método dentro de la clase <code>PrintStream</code> . Este método, en particular, imprime el contenido de los parámetros en la consola e inserta una nueva línea después.
<code>(</code>	este paréntesis indica que se está accediendo a un método (y no a un campo) y comienza a pasar los parámetros al método <code>println</code> .
<code>"Hello, World!"</code>	este es el literal de cadena que se pasa como parámetro al método <code>println</code> . Las comillas dobles en cada extremo delimitan el texto como una cadena .
<code>)</code>	este paréntesis significa el cierre de los parámetros que se pasan al método <code>println</code> .
<code>;</code>	este punto y coma marca el final de la declaración.

Nota: Cada declaración en Java debe terminar con un punto y coma (;).

El cuerpo del método y el cuerpo de la clase se cierran.

```

    } // end of main function scope
} // end of class HelloWorld scope

```

Aquí hay otro ejemplo que demuestra el paradigma OO. Vamos a modelar un equipo de fútbol con un miembro (¡sí, uno!). Puede haber más, pero lo discutiremos cuando lleguemos a los arreglos.

Primero, definamos nuestra clase de `Team`:

```
public class Team {    Member member;    public
Team(Member member) {    // who is in this Team?
this.member = member;    // one 'member' is in this Team!
    }
}
```

Ahora, definamos nuestra clase de `Member`:

```
class Member {
    private String name;
    private String type;
    private int level; // note the data type here
    private int rank; // note the data type here as well

    public Member(String name, String type, int level, int
rank) {
        this.name = name;
        this.type = type;
        this.level = level;
        this.rank = rank;
    }
}
```

¿Por qué usamos `private` aquí? Bueno, si alguien desea saber su nombre, debe preguntarle directamente, en lugar de buscar en su bolsillo y sacar su tarjeta de Seguro Social. Este `private` hace algo así: impide que las entidades externas accedan a sus variables. Solo puede devolver miembros `private` través de las funciones de obtención (que se muestran a continuación).

Después de ponerlo todo junto, y de agregar los métodos de obtención y el método principal, como se mencionó anteriormente, tenemos:

```

public class Team {
    Member member;

    public Team(Member member) {
        this.member = member;
    }

    // here's our main method

    public static void main(String[] args) {
        Member myMember = new Member("Aurieel", "light", 10, 1);
        Team myTeam = new Team(myMember);
        System.out.println(myTeam.member.getName());
        System.out.println(myTeam.member.getType());
        System.out.println(myTeam.member.getLevel());
        System.out.println(myTeam.member.getRank());
    }
}

class Member {
    private String name;
    private String type;
    private int level;
    private int rank;

    public Member(String name, String type, int level, int rank)
    {
        this.name = name;
        this.type = type;
        this.level = level;
        this.rank = rank;
    }

    /* let's define our getter functions here */
    public String getName() { // what is your name?
        return this.name; // my name is ...
    }
}

```

```
        return this.type; // my type is ...
    }

    public int getLevel() { // what is your level?
        return this.level; // my level is ...
    }

    public int getRank() { // what is your rank?
        return this.rank; // my rank is
    }
}
```

Salida:

```
Aurieel light
10
1
```

Una vez más, el método `main` dentro de la clase de `Test` es el punto de entrada a nuestro programa. Sin el método `main`, no podemos decirle a la Máquina Virtual Java (JVM) desde dónde comenzar la ejecución del programa.