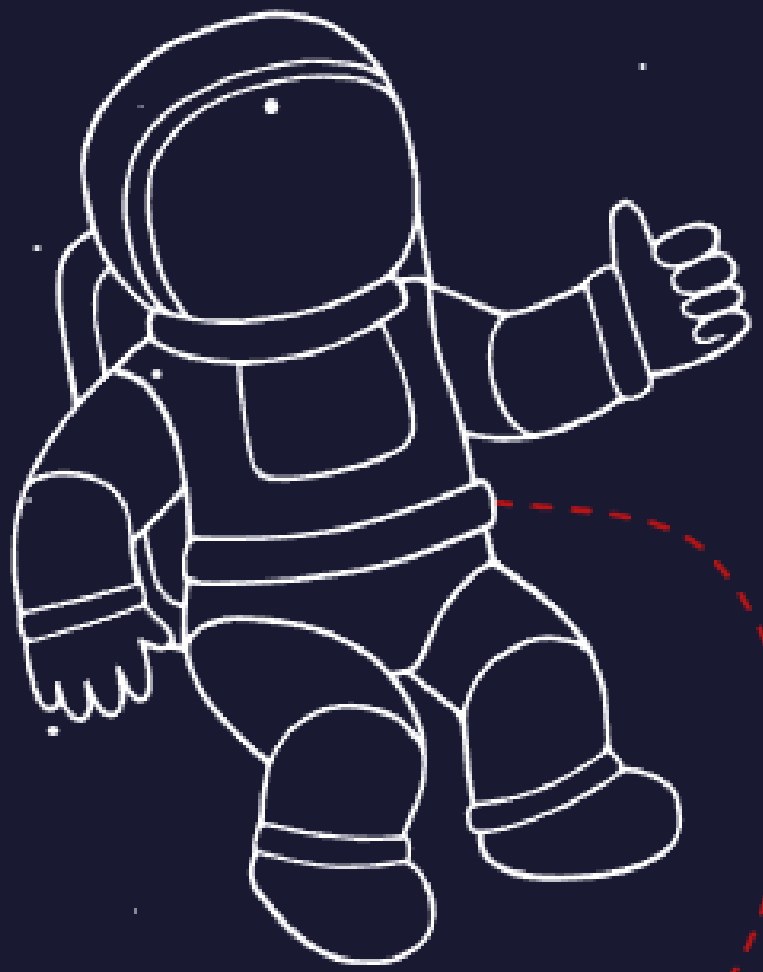


Programa académico CAMPUS

Trainer

Ing. Carlos H. Rueda C.


Java

Sesión 1

Introducción, configuración inicial y variables básicas

1. ¿Qué es Java?

Java es una plataforma informática de lenguaje de programación creada por **Sun Microsystems en 1995** y actualmente en posesión de Oracle. Ha evolucionado desde sus humildes comienzos hasta impulsar una gran parte del mundo digital actual, ya que es una plataforma fiable en la que se crean muchos servicios y aplicaciones. Los nuevos e innovadores productos y servicios digitales diseñados para el futuro también siguen basándose en Java.

1. ¿Qué es Java?

Java es un lenguaje de programación de **propósito general**, tipado, orientado a objetos. Permite el desarrollo desde aplicaciones básicas, pasando por aplicaciones empresariales hasta aplicaciones móviles. Java nació como un lenguaje de programación que pudiese ser multiplataforma y multidispositivo, bajo el paradigma **“Write Once Run Anywhere” (WORA)**

1. ¿Qué es Java?

De esta forma un programa Java escrito una vez, puede **ejecutarse sobre diferentes plataformas**, siendo soportados los sistemas operativos Windows, MacOs y UNIX. Y a su vez en diferentes tipos de dispositivos. Para poder seguir este paradigma la compilación de un programa Java no genera código fuente, si no que genera bytecodes. Estos bytecodes son interpretados por una máquina virtual o JVM (Java Virtual Machine). Dicha máquina ya está escrita para cada uno de los sistemas operativos en cuestión.

1.1. Características de Java

1.1.1. Independiente de plataforma

Cuando compilamos código fuente Java no se genera código máquina específico, si no que se generan bytecodes, los cuales son interpretados por la Java Virtual Machine (JVM), posibilitando que un mismo código fuente pueda ser ejecutado en múltiples plataformas.

1.1. Características de Java

1.1.2. Orientado a objetos

En Java, todo es considerado como un objeto. Esto implica que .

1.1. Características de Java

1.1.3. Sencillo

Java está enfocado para ser un lenguaje fácil de aprender. Simplemente se deberán entender los conceptos básicos de la programación orientada a objetos (POO). Una de las principales ventajas de Java es que, debido a su sintaxis y reglas, es uno de los lenguajes más simples de utilizar. Por ejemplo, sus operadores y apuntadores son realmente sencillos.

1.1.4. Ejecución en dos pasos

Java funciona mediante la compilación e interpretación simultánea durante el proceso de ejecución de órdenes. Esto hace que el código sea utilizable en muchos sistemas operativos, ya que se interpreta para cada uno de ellos.

1.1. Características de Java

1.1.5. Seguro

Java ofrece un alto nivel de seguridad gracias a su ejecución en dos pasos. Debido a que el código tiene que pasar por ambos procesos, es mucho más difícil hackear o modificarlo de una forma maliciosa.

1.1. Características de Java

1.1.6. Arquitectura neutral

Independientemente de que se ejecute en una arquitectura de 32bits o de 64bits. En Java los tipos de datos siempre ocupan lo mismo.

1.1. Características de Java

1.1.7. Multi-hilo

Java permite la programación concurrente, de tal manera que un único programa puede abrir diferentes hilos de ejecución.

1.1. Características de Java

1.1.8. Distribuido

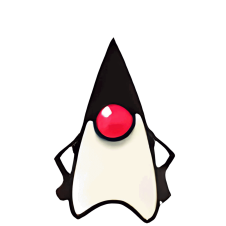
Este lenguaje está pensado para distribuir operaciones entre diferentes equipos. Al emplear soluciones online, es posible dividir tareas y funcionalidades entre dos o más sistemas y compartir información para que el programa corra eficientemente. Esto favorece sobre todo las arquitecturas orientadas a la WEB.

1.2. ¿Cómo funciona Java?

Java es un lenguaje portable que se puede ejecutar en cualquier sistema operativo, pero, para esto ocurren los siguientes procesos:

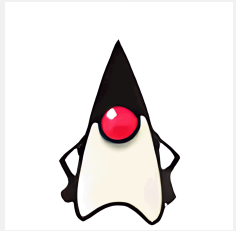
1.2. ¿Cómo funciona Java?

1. **Compilación**: En Java, el código fuente se escribe en archivos con extensión ".java". Estos archivos contienen las definiciones de clases y los métodos asociados. Una vez que se ha escrito el código, se debe compilar utilizando el compilador de Java (javac). El compilador convierte el código fuente en un formato de nivel inferior llamado bytecode en archivos con extensión “.class”, que es independiente de la plataforma.



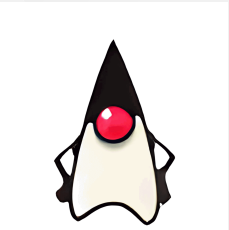
1.2. ¿Cómo funciona Java?

2. **Máquina virtual Java (JVM)**: Después de la compilación, el bytecode generado se puede ejecutar en cualquier plataforma que tenga instalada una JVM. La JVM es una máquina virtual que interpreta y ejecuta el bytecode. Actúa como un entorno de ejecución para los programas Java y se encarga de gestionar la memoria, la seguridad y otros aspectos de la ejecución del programa.



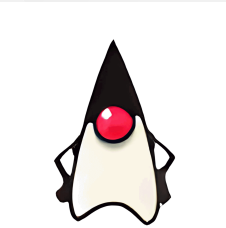
1.2. ¿Cómo funciona Java?

3. **Carga y ejecución del bytecode:** Cuando se ejecuta un programa Java, la JVM carga el bytecode en la memoria y lo ejecuta línea por línea. La JVM interpreta el bytecode o, en algunos casos, utiliza técnicas de compilación en tiempo de ejecución, como la just-in-time compilation (JIT), para mejorar el rendimiento del programa.



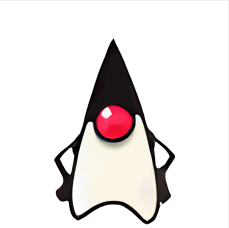
1.2. ¿Cómo funciona Java?

4. **Administración de memoria**: La JVM se encarga de administrar la memoria para los programas Java. Utiliza un recolector de basura (garbage collector) para liberar automáticamente la memoria ocupada por objetos que ya no son accesibles, lo que ayuda a evitar fugas de memoria y simplifica el desarrollo.



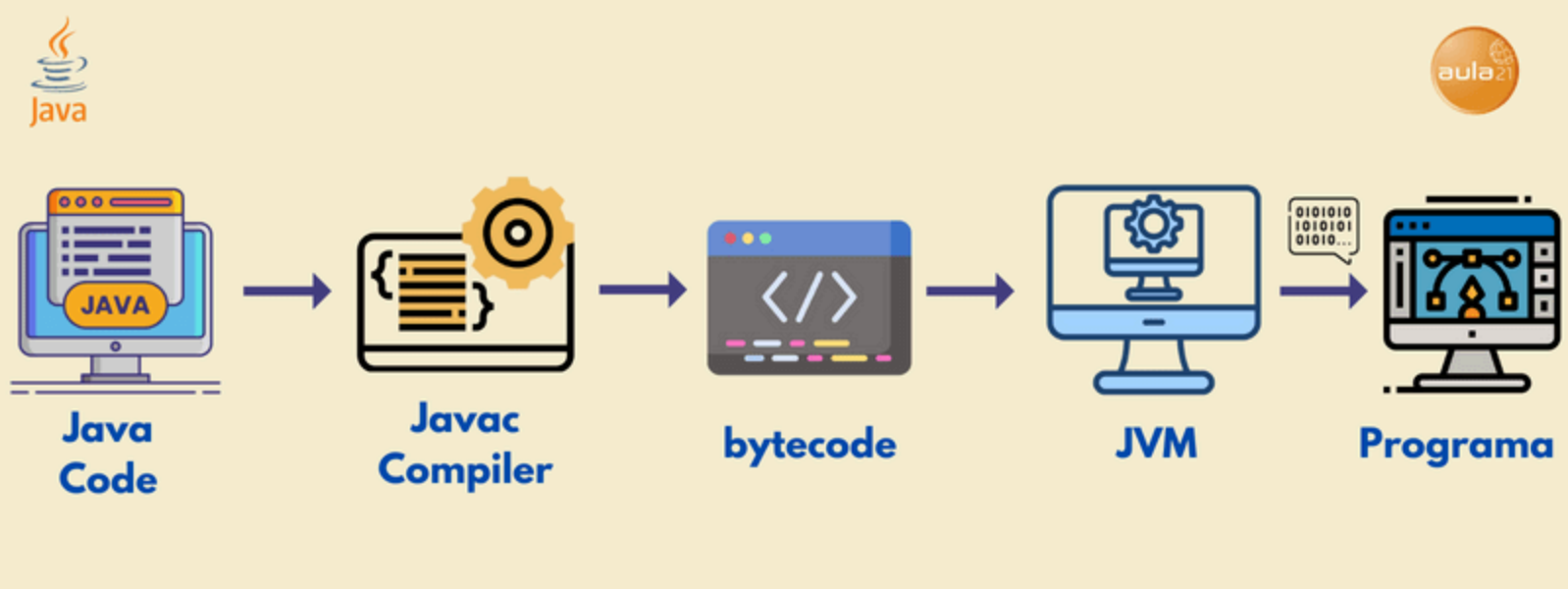
1.2. ¿Cómo funciona Java?

5. **Bibliotecas estándar**: Java proporciona una amplia biblioteca estándar (Java Standard Library) que contiene clases y métodos predefinidos para realizar tareas comunes, como manejo de archivos, manejo de cadenas, entrada y salida, redes, entre otros. Estas bibliotecas facilitan el desarrollo de aplicaciones Java al proporcionar una funcionalidad lista para usar.



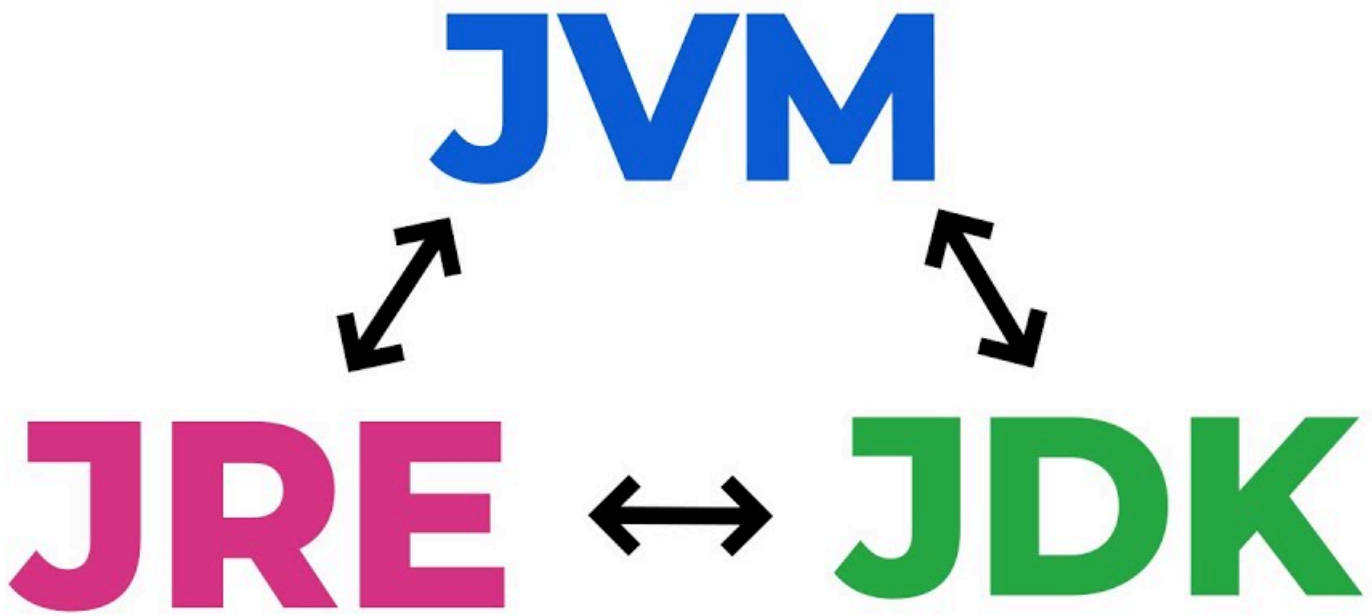
1.2. ¿Cómo funciona Java?

Estructura básica de un programa en Java



1.2. ¿Cómo funciona Java?

1.3. JDK, JRE y JVM



1.3. JDK, JRE y JVM

Como se ha mencionado anteriormente, Java es un lenguaje tanto compilado como interpretado. Por lo tanto, existen tres componentes que debemos tener en cuenta para el funcionamiento de Java en los diferentes ciclos de vida del desarrollo del software con este lenguaje.

1.3. JDK, JRE y JVM

1.3.1. Java Development Kit (JDK)

El Java Development Kit (JDK) es un **entorno de desarrollo de software** utilizado para desarrollar aplicaciones y applets de Java. Incluye Java Runtime Environment (JRE), un intérprete/cargador (Java), un compilador (javac), un archivador (jar), un generador de documentación (Javadoc) y otras herramientas necesarias para el desarrollo de Java.

1.3. JDK, JRE y JVM

1.3.2. Java Runtime Environment (JRE)

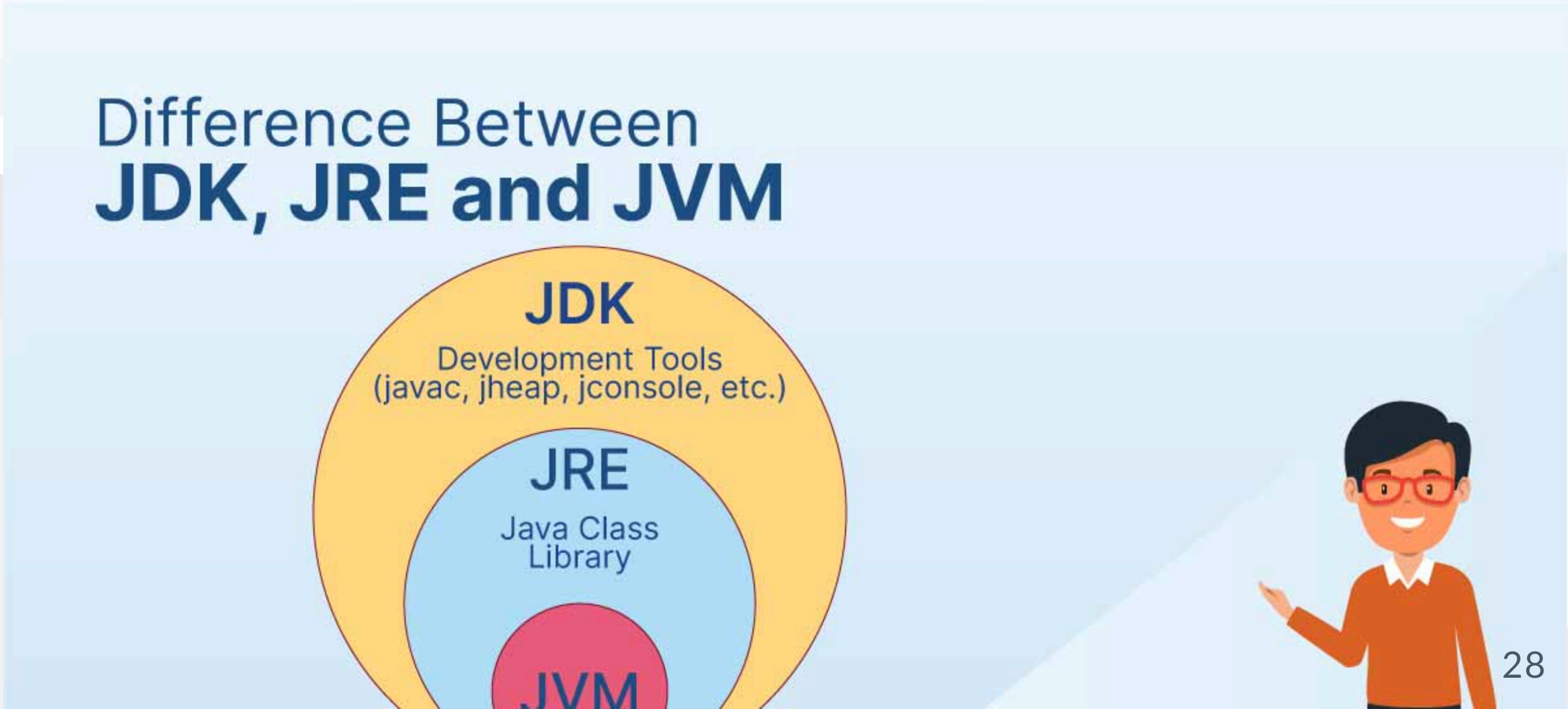
JRE significa “Java Runtime Environment” y también puede escribirse como “Java RTE“. Java Runtime Environment proporciona los **requisitos mínimos para ejecutar una aplicación Java**; consiste en Java Virtual Machine (JVM), clases principales y archivos auxiliares.

1.3. JDK, JRE y JVM

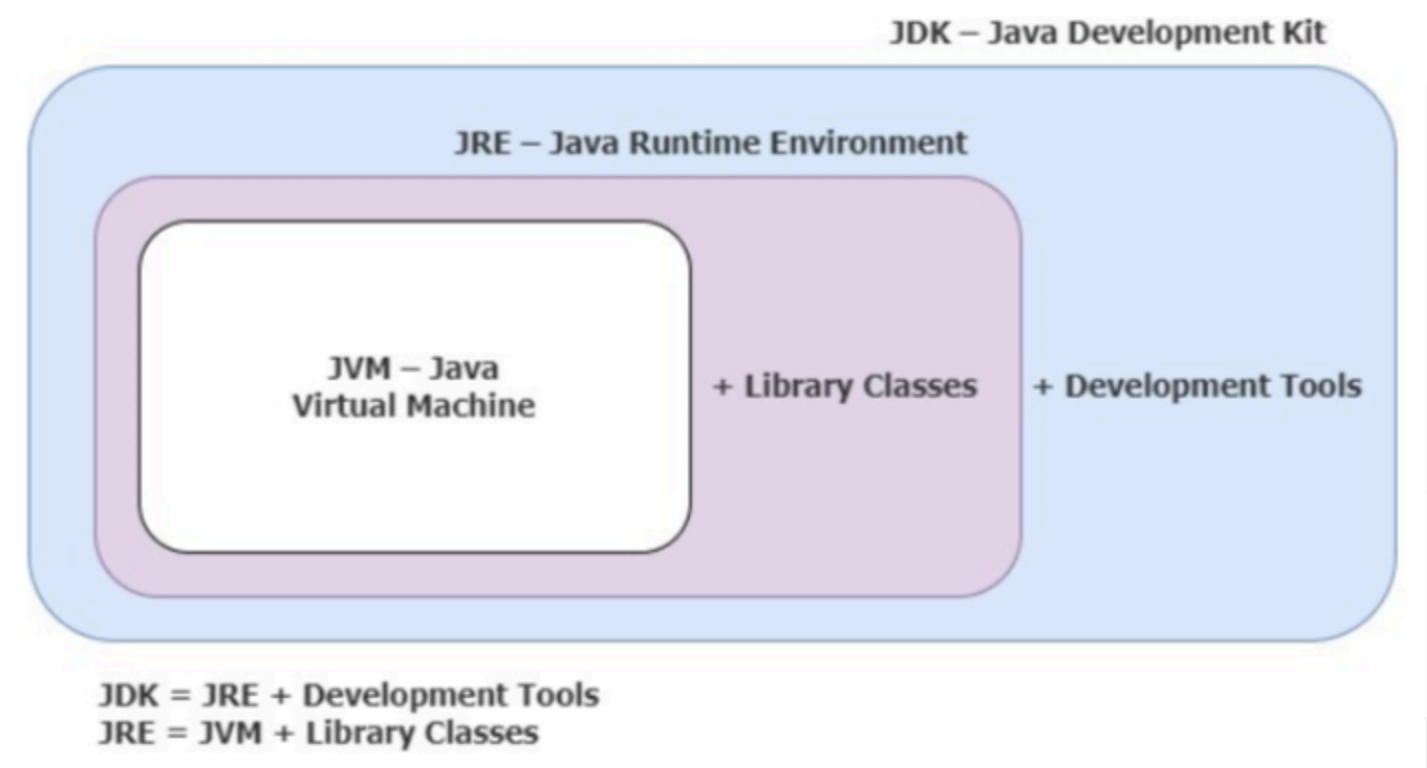
1.3.3. Java Virtual Machine (JVM)

es una parte muy importante de JDK y JRE porque está contenida o incorporada en ambos. Cualquier programa Java que ejecute utilizando JRE o JDK entra en la JVM y la JVM es **responsable de ejecutar el programa Java línea por línea**, por lo que también se lo conoce como intérprete.

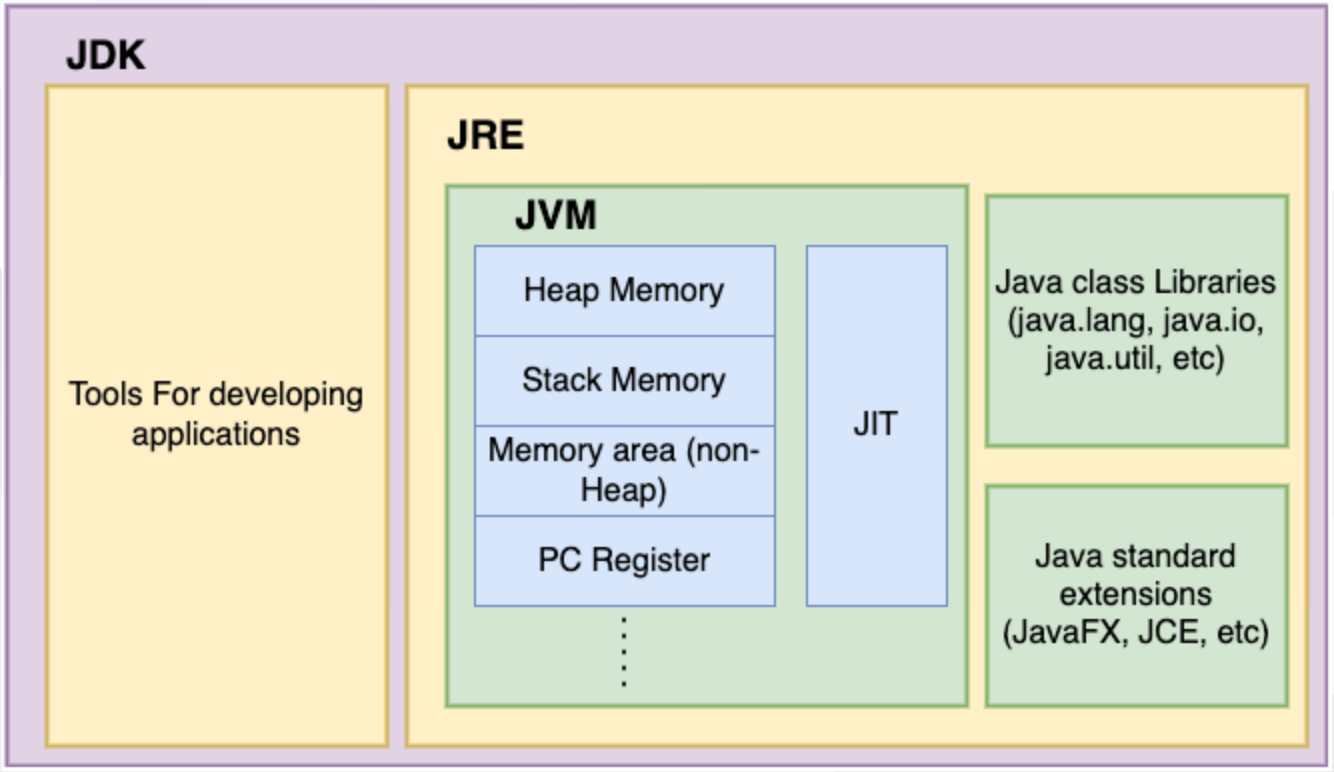
1.3. JDK, JRE y JVM



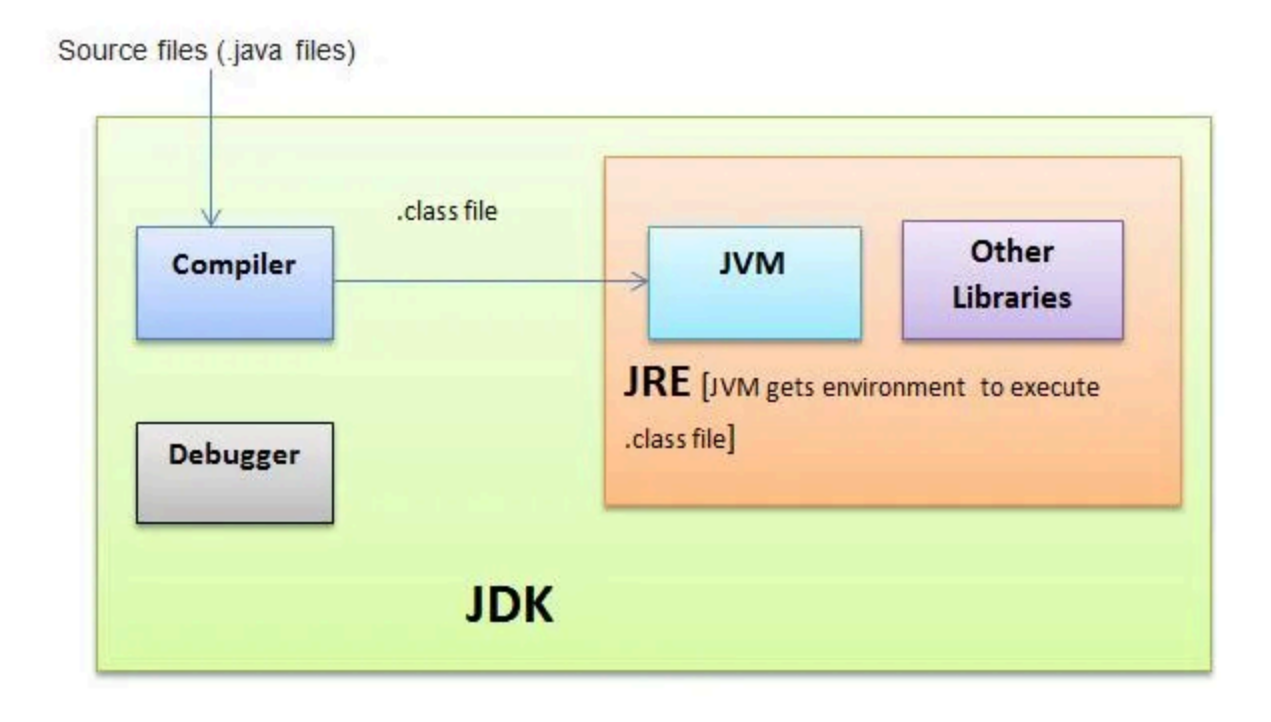
1.3. JDK, JRE y JVM



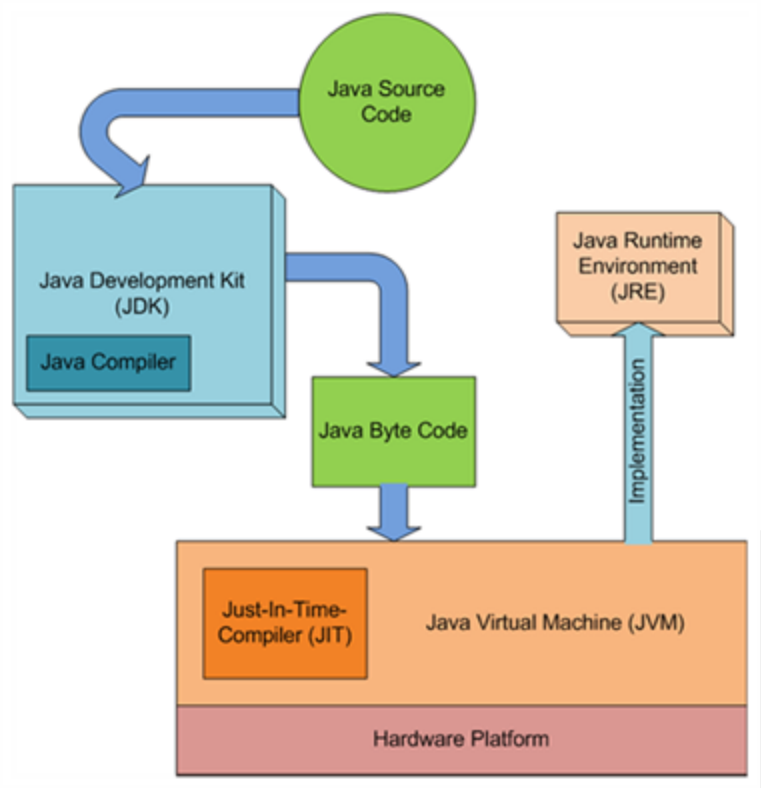
1.3. JDK, JRE y JVM



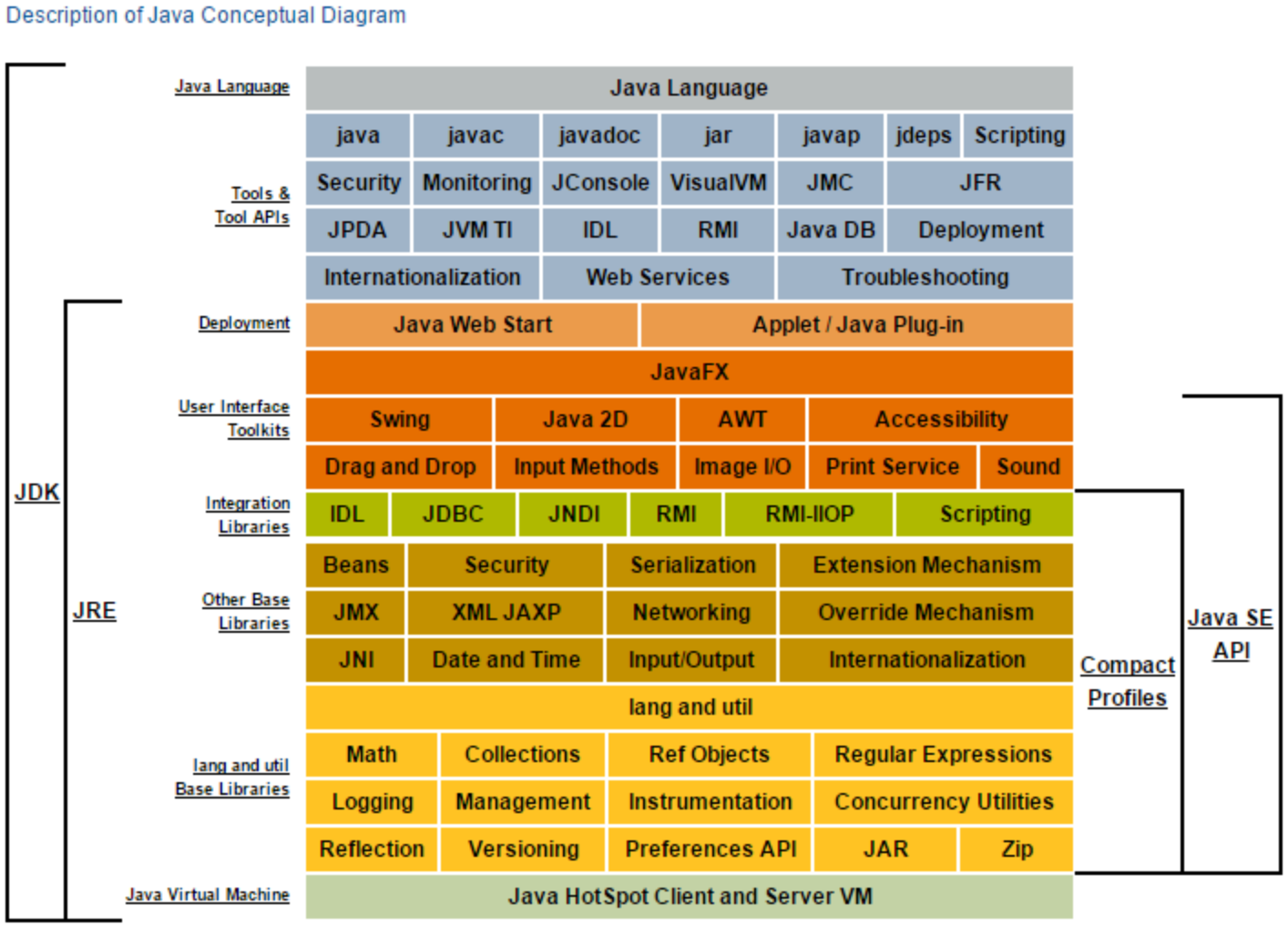
1.3. JDK, JRE y JVM



1.3. JDK, JRE y JVM



1.3. JDK, JRE y JVM



2. Entorno de desarrollo Java

2.1. Netbeans

- **Descripción:** NetBeans es otro IDE gratuito y de código abierto que, en un principio, fue desarrollado por Sun Microsystems y luego adquirido por Oracle. Está diseñado para facilitar el desarrollo de aplicaciones Java SE y Java EE.
- **Características:** Incluye herramientas de diseño visual, especialmente útiles para crear interfaces gráficas. Además, tiene un sistema de gestión de proyectos intuitivo y una buena

2. Entorno de desarrollo Java

2.1. Eclipse

- **Descripción:** Eclipse es un IDE de código abierto para Java que es ampliamente utilizado en la industria. Es modular y extensible, permitiendo instalar plugins para añadir funcionalidades adicionales.
- **Características:** Soporta múltiples lenguajes además de Java y tiene una comunidad activa que desarrolla plugins para integrarse con herramientas de control de versiones, bases de datos, y

2. Entorno de desarrollo Java

2.1. IntelliJ IDEA

- **Descripción:** Desarrollado por JetBrains, IntelliJ IDEA es uno de los IDEs más populares para Java. Su versión Ultimate (de pago) es especialmente útil para desarrollo empresarial, mientras que la versión Community (gratuita) ofrece muchas funcionalidades esenciales.
- **Características:** Destaca por su avanzada autocompletado de código, refactorización inteligente y herramientas integradas para

2. Entorno de desarrollo Java

2.1. Visual Studio Code (VS Code)

- **Descripción:** Aunque no es un IDE dedicado exclusivamente a Java, VS Code de Microsoft es un editor de código muy popular que, con la ayuda de extensiones, soporta Java de manera eficiente.
- **Características:** Las extensiones de Java para VS Code, como la extensión de Java para desarrolladores de Microsoft, permiten trabajar en proyectos de Java con autocompletado, depuración, y soporte para Maven y Gradle.

4. Variables y constantes en Java

Variables

Para definir una variable seguiremos la estructura:

```
[Tipo de variable] [Identificador de variable];
```

- Es fuertemente tipado.
- Todas las variables tendrán un tipo de dato y un nombre de identificador

4. Variables y constantes en Java

Variables

🖼️ Ejemplos de variables

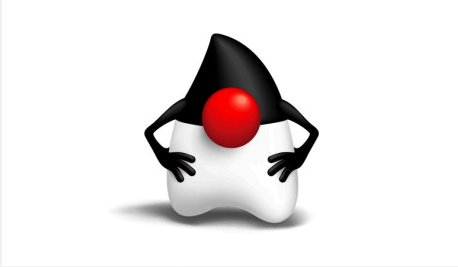
```
int numero = 3;

String cadena = "Hola a todos!";

double decimal = 57;

boolean bandera = true;
```

4. Variables y constantes en Java

Constantes

En Java para declarar que un valor es constante anteponemos la palabra reservada `final` así:

```
final [Tipo de variable] [Identificador de variable];
```



4. Variables y constantes en Java

Declarar variables con var

A partir de la versión **10** de JDK se incluyó **var** para la definición de variables por inferencia (deducción) y no expresando directamente el tipo de variable que es cada una; esto no quiere decir que estas variables tengan un tipado débil y puedan cambiar su tipo. El uso de **var** ahorra algunos caracteres y evita la redundancia en la declaración, pero, se puede considerar que esto otorga mucha libertad a la hora de escribir código y *por ende genera malas prácticas*.

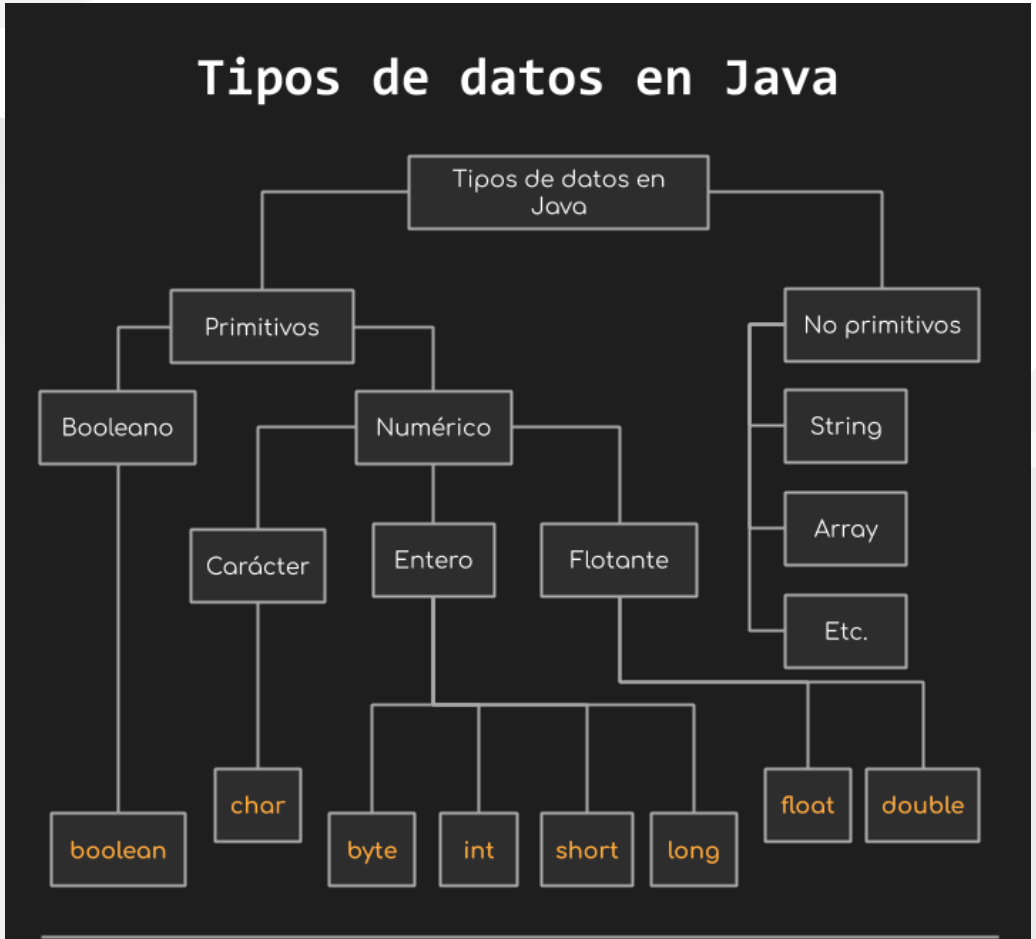
Declarar variables con var

 Ejemplo uso de `var`

```
int numero = 3;

numero = 25;
numero = "Hola mundo";

var v1 = 5;
var v2 = "Hola mundo";

var v1 = "Hola mundo";
var v2 = "Campus";
```

4. Variables y constantes en Java

4.1. Datos primitivos en Java



Tipo de Dato	Subdivisión	Descripción	Límites
Entero	int	4 bytes de espacio de almacenamiento	-2147483648 hasta 2147483647
	Short	2 bytes de espacio de almacenamiento	-32768 hasta 32767
	Long	8 bytes de espacio de almacenamiento	Números enormes que va con sufijo **L**
	Byte	1 byte de espacio de almacenamiento	-128 hasta 127
Decimales o coma flotante	Float	4 bytes de espacio de almacenamiento	6 a 7 cifras decimales significativas que van con sufijo F (3.25F)
	Double	8 bytes de espacio de almacenamiento	Hasta 15 cifras decimales significativas
Caracteres	Char	Representa caracteres, van entre comillas simple ''	'a', 'z', etc.
Lógicos	Boolean	Son datos que pueden ser verdaderos o falsos	V=true, F=false.

4. Variables y constantes en Java

4.1. Datos primitivos en Java

Tipo de dato	Tamaño (Byte*)	Rango
byte	1	-128 a 127
short	2	-32,768 a 32,767
int	4	-2,147,483,648 a 2,147,483,647
long	8	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	4	Decimal de 6 o 7 dígitos

4. Variables y constantes en Java

4.1. Datos primitivos en Java

`Byte`: es una unidad de almacenamiento de datos que consiste en 8 bits contiguos. Un bit (acrónimo de binary digit) es la unidad básica de información en computación y puede representar un valor de 0 o 1.

4. Variables y constantes en Java

4.1. Datos primitivos en Java

🖼️ Ejemplo de valores primitivos en Java

```
byte miByte = 127;
short miShort = 32767;
int miInt = 2147483647;
long miLong = 9223372036854775807L;
float miFloat = 2.12345F;
double miDouble = 2.1245;
boolean miBoolean = true;
char miChar = 'A';
```

4.2. Impresión en consola

Para imprimir por pantalla, se usa la clase `System`, el atributo `out`, y su método `println()` o `print()` así: **`System.out.println()`** o **`System.out.print()`** notar que `System` es una clase y siempre debe ir con la `"S"` mayúscula.

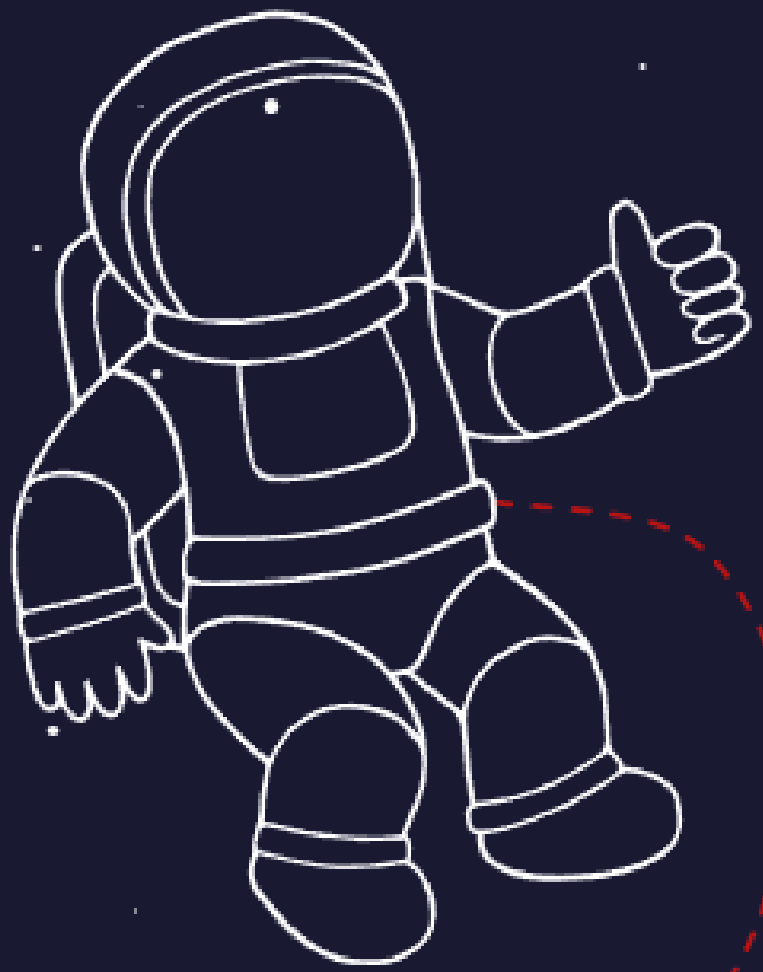
 Ejemplo de impresiones en consola

```java
int valor = 123;
System.out.println("Hola estudiantes!");
System.out.println(valor);
System.out.println("Mi edad es: " + valor);
```

4.2.1. Diferencia entre print() y println()

La primera pregunta que surge al ver los métodos útiles para salida de datos en Java es en qué se diferencian (`print()` vs `println()`), básicamente ambos son útiles para mostrar texto por pantalla y la única diferencia es que `println()` muestra el texto y luego deja un salto de línea (como si presionara enter) mientras que `print()` no lo hace.

 Ejemplo de diferencia entre print() y println()



Programa académico CAMPUS

Trainer

Ing. Carlos H. Rueda C.