Sumario

Colección de apuntes de Java

No poseo los derechos de autor de todo el material expuesto en esta colección de apuntes. Utilice innumerables fuentes de datos. Algunas partes las escribí o las modifique para adaptarlas a las versiones Java 8 y Java 11.

Muchas gracias a todas las fuentes de datos que enriquecieron este material de estudio no comercial.

Java Primeros Pasos

¿Qué es Java?

JAVA es una tecnología pensada para desarrollo de aplicaciones de gran envergadura, altamente escalables, de gran integración con otras tecnologías y sumamente robustas.
Java no es tan solo un lenguaje, es una tecnología para el desarrollo de aplicaciones que conforma la plataforma sobre la cual se ejecutarán las mismas y en conjunto esta se compone de:

- Un lenguaje de programación. (javac)
- Un entorno de desarrollo. (ide)
- Un entorno para aplicaciones. (java runtime)
- Un entorno para despliegue de aplicaciones (Servlet container). (java EE Apache Tomcat, Apache TomEE, RedHat Jboss, Eclipse Jetty, Oracle GlassFish, Oracle WebLogic, IBM WebSphere).

Sus principales características son presentadas a continuación:

Lenguaje orientado a objetos

Respeta el paradigma de orientación a objetos, permitiendo utilizar los fundamentos del mismo:

- Herencia
- Polimorfismo
- Abstracción
- Encapsulamiento

Sintaxis basada en C/C++

La sintaxis es similar al C++ pero el manejo y la semántica son parecidos al SmallTalk. Se utiliza para desarrollar aplicaciones locales y distribuidas.
Aporta gran simplicidad ya que es una de las formas de escribir código más reconocidas y difundidas, y permite incorporar rápidamente a los programadores que conocen este lenguaje.

Multiplataforma

Significa que su código es portable, es decir se puede transportar por distintas plataformas. De esta manera es posible codificar una única vez una aplicación, y luego ejecutarla sobre cualquier plataforma y/o sistema operativo.
"Write once, run anywhere" es la política desde el primer día de JAVA, es decir construir la aplicacion una vez y ejecutarla en “cualquier lado”.

Manejo automático de memoria

No hay que preocuparse por liberar memoria manualmente ya que un proceso propio de la tecnología se encarga de monitorear, y por consiguiente eliminar el espacio ocupado que no esta siendo utilizado. El proceso encargado de realizar este trabajo se denomina Garbage Collector.

Evolución permanente

La tecnología está en constante evolución debido a la gran cantidad de “consumidores” que poseen, JAVA es uno de los lenguajes más utilizados en el mundo, y SUN (Oracle) pretende estar a la altura de la situación ofreciendo constantemente nuevas entregas.

Organización

La tecnología está organizada en tres grandes áreas bien definidas:

JME Java Micro Edition

Esta área tiene como objetivo el desarrollo de aplicaciones móviles, tales como GPS, Handhelds (por ejemplo la conocida Palm), celulares y otros dispositivos móviles programables. JME significa Java Micro Edition.

JSE Java Standard Edition

Esta área tiene como objetivo el desarrollo de aplicaciones de escritorio, similares a las aplicaciones tipo ventanas creadas con Visual Basic o Delphi. Incluye la funcionalidad básica del lenguaje como manejo de clases, colecciones, entrada/salida, acceso a base de datos, manejo de sockets, hilos de ejecución, etc. JSE significa Java Standard Edition.

JEE Java Enterprise Edition

Esta área tiene como objetivo el desarrollo de aplicaciones empresariales, de gran envergadura. Contempla ambientes web, como los ambientes manejados por servidores de aplicación. Las tecnologías principales incluidas en esta área son Servlets, JSP y EJB, entre otras. JEE significa Java Enterprise Edition.

La información recopilada en este apunte es sobre Java Standard Edition.

Exámenes de certificación Oracle

Existen dos tipos de exámenes de certificación del lenguaje OCA y OCP, siendo el primero el de nivel inicial y el segundo más avanzado.

Un **Asociado certificado por Oracle (OCA, por su sigla en inglés)** posee destrezas fundamentales y conocimientos básicos, y uno o dos años de experiencia.

Un **Profesional certificado por Oracle (OCP, por su sigla en inglés)** es la credencial insignia de nuestro programa. Esta certificación mide un nivel más profundo y avanzado de destrezas. Convertirse en un OCP demuestra sus capacidades y sus sólidos conocimientos en un área específica de la tecnología de Oracle, y dos o tres años de experiencia.

La Historia

El comienzo

En el año 1990 nace Java, bajo el diseño y la implementación de la empresa Sun Microsystems. El padre-fundador de la tecnología es el James Gosling, a través de una filial dentro de Sun llamada First Person Inc.
Gosling tuvo la visión inicial de construir una lenguaje de programación capaz de ejecutar su código sobre cualquier set de instrucciones, de distintos procesadores. Inicialmente el proyecto apuntó a la programación unificada de distintos electrodomésticos, es decir programar una sola vez y que el programa generado fuera útil para cualquier dispositivo.
El proyecto inicial de Java fue técnicamente un éxito, aunque comercialmente no tuvo el rendimiento esperado, y debió ser relegado unos años.

Aparición de Internet

En el año 1993, Internet da el gran salto, y se convierte de una interfaz textual a una interfaz gráfica. Java ve una oportunidad y entra fuertemente a internet con los Applets, pequeños programitas construidos en Java – con todos sus beneficios – capaces de ejecutarse dentro de un navegador. Es aquí donde Java comienza a dar sus primeros pasos firmes como lenguaje a difundir masivamente.
En el año 1995, el navegador Netscape Navigator comienza formalmente a soportar los Applets Java.
Adicionalmente, el lenguaje podía adaptarse fácilmente a las múltiples plataformas, con lo cual surge una de las primeras aplicaciones multiplataformas más conocidas: WebRunner (hoy HotJava), un navegador multiplataforma construido en Java.

Por qué el nombre JAVA

Inicialmente la intención fue nombrar al lenguaje de programación con el nombre de Oak, pero este ya estaba registrado. La leyenda cuenta que una visita a la cafetería le dio rápida solución al problema.
En las confiterías norteamericanas hay un café denominado Java, en el cual esta inspirado el nombre del lenguaje de programación. El logotipo de Java es justamente una taza café.

El Java Development Kit (JDK)

Oracle JDK

El Java Development Kit es el kit de desarrollo propuesto por Sun Microsystems para realizar desarrollos en JAVA. Se puede bajar de forma gratuita de la pagina http://www.oracle.com.
El kit incluye herramientas de desarrollo tales como un compilador, un debugger, un documentador para documentar en forma casi automática una aplicacion, un empaquetador para crear archivos de distribucion, y otras herramientas mas.
El kit no incluye un entorno de desarrollo interactivo (o IDE) como pueden ser Netbeans, JDeveloper, IntelliJ o Eclipse.
El 20 de marzo de 2018 se publicó la que es la versión 10 de Java siguiendo el nuevo calendario de publicar una nueva versión cada seis meses, Java 9 fue publicado en septiembre del año anterior. Con este nuevo calendario no pasarán tantos años entre cada nueva versión que era la queja de algunos desarrolladores y el motivo de que la plataforma Java no evolucionar tan rápidamente como algunos desarrolladores desean, quizá ahora la queja sea al contrario que se publican demasiadas versiones y no da tiempo a asimilar los cambios. Para dar cabida a ambas necesidades y garantizar un soporte prolongado cada año y medio será declarada una versión como de soporte a plazo largo o LTS para que las empresas tengan seguridad en las aplicaciones que desarrollen.
Lo mejor de este nuevo calendario de publicaciones cada seis meses es que las empresas y programadores tienen predictibilidad de cuándo se lanzará la siguiente versión aunque las nuevas características que tenga no está predeterminado ya no pasarán varios años entre versiones visibilizando que la plataforma evoluciona continuamente en pequeños saltos cada poco tiempo en vez de saltos grandes cada mucho tiempo que son más disruptivos y hace más difícil la adopción.
Es un principio no añadir características según lo que está de moda sino pensando en décadas futuras. Java 10 tiene una lista más reducida de cambios que Java 9 pero importantes y significativos. Java es el último en unirse a la fiesta de la inferencia de tipos pero ha sido de forma intencionada ya que el coste de implementarla de forma incorrecta supone un alto coste que hay que mantener en adelante. Otras ideas que ha sido implementadas el lenguajes de programación funcional y están listas para su uso masivo tomarán su propio camino en futuras versiones de Java.
El posible futuro **JDK 11**+, está planificado en seis meses después de Java 10 y con soporte extendido, el soporte de Java 10 durará tan solo hasta 2018.09, el de Java 11 al ser una LTS durará un periodo de 8 años hasta el 2026.09.

OpenJDK

OpenJDK es la versión libre de la plataforma de desarrollo Java bajo concepto de lenguaje orientado a objetos. Esta implementación se encuentra catalogada dentro de la licencia GPL de GNU con una excepción de enlaces, por lo que algunos de los componentes de los folders de clases y sitios web de Java se ultiman de los términos de la licencia para poder ser considerados dentro de la versión estipulada como GNU.

Cronograma de liberación de versiones de JDK

Según el nuevo cronograma de Oracle, cada 6 meses se libera una nueva versión no LTS. Y tiene soporte solo por 6 meses. Y cada 36 meses se libera una nueva versión LTS de soporte extendido por 8 años.
LTS: Long Term Support.

Ej:

JDK10 se libera en Marzo 2018 con soporte hasta Septiembre 2018.
JDK11 LTS se libera en Septiembre 2018 con soporte hasta Septiembre 2026.
JDK12 se libera en Marzo 2019 con soporte hasta Septiembre 2019.
JDK13 se libera en Septiembre 2019 con soporte hasta Marzo 2020.
JDK14 se libera en Marzo 2020 con soporte hasta Septiembre 2020.
JDK15 se libera en Septiembre 2020 con soporte hasta Marzo 2021.
JDK16 se libera en Marzo 2021 con soporte hasta Septiembre 2021.
JDK17 LTS se libera en Septiembre 2021 con soporte hasta Septiembre 2029

Para desarrollo en producción se recomienda usar versiones **LTS**.

Java 11 es la primera versión de soporte extendido publicada o LTS bajo el nuevo ciclo de publicaciones que adoptó Java en la versión 9. Añade varias novedades importantes en cuanto a seguridad y elimina otras que en versiones anteriores ya fueron marcadas como desaconsejadas.

En la plataforma Java era habitual que entre versión y versión mayor pasasen varios años normalmente 3 o más. Este ciclo de publicaciones no se adapta a las necesidades de todas las empresas, organizaciones y usuarios. Algunas empresas y usuarios desean ciclos de publicaciones más cortos quizá con no tantas novedades en cada uno de ellos pero sí de forma más constante incorporando mejoras. Otras organizaciones necesitan confiar en una versión que tenga un ciclo de soporte largo para sus aplicaciones.
Para adaptarse a ambos requerimientos Java a partir de Java 9 adoptó un nuevo ciclo de publicaciones siendo cada seis meses para los que desean mejoras de forma regular y cada tres años para los que necesitan soporte extendido. Java 9 incorporó la esperada modularización que cambia la forma de desarrollo de las aplicaciones con numerosas mejoras. Java 10 añadío la inferencia de tipos para variables locales con la palabra reservada var. Ahora Java 11 siendo una versión de soporte extendido o LTS, el soporte de Java 11 está planificado que dure hasta 2023 y hasta 2026 de forma extendida lo que son 8 años de soporte.
En esta nueva versión de Java 11 publicada el 25 de septiembre de 2018 las novedades no son tan relevantes como lo fuero Java 8 con las lamdas y Java 9 con los módulos pero continúa con las mejoras incrementales y proporciona una versión LTS en la que empresas grandes confiarán como base para sus desarrollos.

Nueva forma de licenciamiento de Oracle JDK

El **JDK 11** inicia una nueva era en la licencia de uso. Hasta ahora podías descargar y programar con el Kit de Desarrollo de Java oficial de **Oracle** y luego poner tu aplicación en producción o distribuirla sin tener que pagar nada al gigante del software. Sin embargo, a partir de Java 11 y del JDK 11, aunque puedes seguir desarrollando con él, tendrás que pagar una licencia a Oracle si quieres utilizarlo para poner las aplicaciones en producción. En Diciembre de 2018, el coste es de 2,5 dólares al mes por cada usuario de escritorio, y de 25 dólares por procesador en el caso de aplicaciones de servidor.
Esto no afecta a versiones anteriores del JDK, por lo que si usas Java 8, 9 o 10 sigue siendo gratuito.

Opciones de Java de software libre.

Que no cunda el pánico. No es necesario que te quedes con el JDK 8 o 9 para no tener que pagar licencia, perdiéndote además las novedades que surjan en la plataforma. Además, es importante seguir avanzando...Hasta hace poco, el **OpenJDK** era el patito feo de los SDKs de Java. El JDK oficial tenía más cosas, era más estable y además gratuito, por lo que el JDK siempre ha sido la opción por defecto para programar con Java.

Sin embargo, sabiendo los cambios de licencias que estaban preparando, Oracle ha trabajado muy duro para hacer que el **OpenJDK** se haya equiparado en todos los aspectos al JDK, hasta el punto de que se puede decir que el **OpenJDK** y el JDK son idénticos desde un punto de vista técnico, desde la versión 11.

Debido a ello, ahora el **OpenJDK** debería ser tu opción por defecto para **programar en Java**.

Esto no significa que Java deje de ser gratuito, ni es una gran tragedia para la plataforma tampoco. Para la mayoría de desarrolladores y empresas no habrá una gran diferencia. El desarrollo se trasladará a OpenJDK (que ahora es equiparable al JDK) en lugar de al tradicional JDK de Oracle, y el soporte se obtendrá de la comunidad. Para ciertas grandes empresas y administraciones, el hecho de pagar no les importará y de hecho será una ventaja porque tendrán garantizada la estabilidad y el soporte directo de Oracle durante unos años.

Aún así Java sigue y seguirá siendo el lenguaje más popular del mundo, y el que más puestos de trabajo genera, así que merece la pena aprenderlo bien.

Eclipse OpenJ9

OpenJ9 es una implementación de java de alto rendimiento, escalable.
El Eclipse OpenJ9 JVM es totalmente compatible con la especificación Java JVM. La misma versión de la JVM puede ser utilizado en OpenJDK 8 y versiones posteriores, lo que significa que muchas características y mejoras pueden ser explotados por las aplicaciones que se ejecutan en diferentes versiones de Java.
Eclipse OpenJ9 incrusta **Eclipse OMR** , que proporciona componentes de ejecución de la base que se pueden utilizar para construir entornos de ejecución para diferentes lenguajes de programación. En el proyecto OpenJ9, una capa adicional de código añade la semántica del lenguaje para proporcionar un entorno de ejecución de aplicaciones Java.

Otros JDK alternativos

- RedHat OpenJDK con soporte RedHat.
- IBM OpenJDK con soporte IBM

Descarga e instalación del JDK

La descarga e instalación de la herramienta JDK varia según la plataforma. Pero una vez instalada debemos verificar que la misma este correctamente instalada.
Para verificar la instalación desde terminal de sistema:

```
javac -version      //verifica la versión instalada del compilador.
java -version       //verifica la versión instalada del java Runtime.
```

El compilador

El compilador viene incluido como una herramienta dentro de la JDK, en el sistema operativo Windows viene presentado como javac.exe
El compilador transforma los archivos de código fuente de java, es decir los archivos de texto con extensión .java, en archivo compilados, también denominados bytecode. Los archivos compilados tienen la extensión .class, y son archivos binarios.

Que son los Bytecodes?

En el lenguaje de programación Java, se puede "escribir una vez, ejecutar en cualquier parte". Esto significa que cuando se compila un programa, no se generan instrucciones para una plataforma específica. En su lugar, se generan bytecodes Java, que son instrucciones para la Máquina Virtual Java (Java VM). Si la plataforma- sea Windows, UNIX, MacOS o un navegador de Internet-- tiene la Java VM, podrá entender los bytecodes.

El Java Runtime Environment (JRE)

Java Runtime Environment es el ambiente de ejecución de Java, y también esta incluido en la JDK. Tiene como componentes más importantes a la Java Virtual Machine y a las class libraries, que son las que contienen las clases base del lenguaje de programación JAVA.
El JRE se distribuye también en forma independiente, es decir sin la JDK, ya que cuando es necesario desplegar una aplicación hecha en JAVA en el cliente, no es necesario instalare herramientas que son propias del proceso de desarrollo, como ser el compilador, empaquetador, documentador, y otros.
Sin una JRE instalada no es posible ejecutar una aplicación construida en JAVA.
En Windows, el comando para invocarlo es el java.exe

La Java Virtual Machine (JVM)

La Java Virtual Machine viene incluida dentro de la Java Runtime Environment, y tiene como principal objetivo la ejecución de código JAVA compilado, es decir de los archivo .class .
La JVM se encarga de interpretar el bytecode y convertirlo a código nativo en tiempo de ejecución, lo cual hace que la ejecución sea un poco más lenta pero garantiza la portabilidad, es decir que el lenguaje sea multiplataforma. De esta manera el código compilado JAVA se puede ejecutar en cualquier plataforma (arquitectura + sistema operativo) que tenga instalada el JRE.

¿Cómo se crea un programa en Java?

Para crear un programa:

- ***Crear un archivo fuente Java:*** *Un archivo fuente contiene texto, escrito en el lenguaje de programación Java, que los programadores pueden entender. Se puede usar cualquier editor de texto para crear y editar archivos fuente.*

- ***Compilar el archivo fuente en un archivo de bytecodes:*** *El compilador de Java, javac, toma el archivo fuente y lo traduce en instrucciones que la Máquina Virtual Java (Java VM) puede entender. El compilar pone estas instrucciones en un archivo de bytecodes.*

- ***Ejecutar le programa contenido en el archivo de bytecodes:*** *La máquina virtual Java está implementada por un intérprete del lenguaje. Este intérprete toma el archivo de bytecodes y lleva a cabo las instrucciones traduciéndolas a instrucciones que el computador puede entender.*

Crear un archivo fuente Java

Se puede utilizar cualquier editor de texto cualquiera. El único cuidado en este punto es que el nombre del archivo debe coincidir con el nombre de la clase que se declare dentro de él y la extensión debe ser .java. Sino, el programa dará un error al tratar de compilarlo. Por ejemplo, si se escribe el archivo del ejemplo anterior su contenido será:

```
public class HolaMundo {
	public static void main(String[] args) {
		// Mostrar por consola "Hola Mundo"
		System.out.println("Hola Mundo!!!");
	}
}
```

El nombre con el cual se deberá guardar el archivo debe ser HolaMundo.java, es importante que el nombre del archivo sea igual al nombre de la public class.
Por otra parte, se debe tener cuidado al escribir el código puesto que el lenguaje es sensible al caso, lo cual quiere decir que las mayúsculas y la minúsculas se interpretan diferente.

¿Cómo se compila y ejecuta?

Compilar el archivo fuente en un archivo de bytecodes, Para compilar el archivo fuente, desde terminal (consola) de sistemas, se debe cambiar al directorio en el que se encuentra el archivo y ejecutar el compilador, javac, con el cuidado que el sistema operativo pueda encontrarlo (definir el camino – path – para hacerlo).

```
javac HolaMundo.java
```

Si el prompt reaparece sin mensajes de error, el programa se ha compilado con éxito. El compilador ha generado un archivo de bytecodes Java, HolaMundo.class. Ahora que existe un archivo .class, se puede ejecutar el programa.

Nota: *para realizar todos estos pasos se puede utilizar un entorno de desarrollo integrado (IDE). Este tipo de entornos definen internamente la variable CLASSPATH y permiten editar, compilar, ejecutar y depurar los programas en Java, pero más allá del entorno que se posea, siempre invocan a los comandos aquí explicados aunque en forma transparente para el desarrollador.*

La variable de entorno CLASSPATH

La variable de entorno CLASSPATH se utiliza para referenciar el directorio donde estarán ubicadas todas las clases construidas en JAVA, para que el JRE al ejecutar una clase sepa dónde ubicar el resto de las clases o archivos empaquetados que contienen clases.

Ejecutar el archivo.class de java

Para lograr ejecutar el archivo.class (HolaMundo.class) generado por el compilador escribir desde terminal de sistema.

```
java HolaMundo
```

Y el programa se ejecutara.

Como armar el ejecutable .jar?

El ejecutable de java .jar es un archivo comprimido .rar que lee y ejecuta la JVM, la ventaja de este ejecutable es que funciona en cualquier plataforma o sistema operativo que disponga de la JVM.

Primero armar un archivo Manifest.mf adentro de la carpeta del proyecto con el siguiente texto.

```
Main-Class: HolaMundo
```

Esto le indica cual es el punto de entrada al proyecto.

En la terminal del sistema dentro del directorio del programa HolaMundo:

```
 jar cmf Manifest.mf HolaMundo.jar HolaMundo.class
```

La opción "**c**" indica que queremos crear un fichero.jar nuevo. Si ya existía, se borrará, así que hay que tener cuidado.
La opción "**m**" para indicar que vamos a añadir un fichero de manifiesto.
La opción "**f**" sirve para indicar el nombre del fichero, que va inmediatamente detrás. En nuestro caso, **fichero.jar**. Finalmente se pone una lista de HolaMundo.class (o de cualquier otro tipo) que queramos meter en nuestro jar. Se pueden usar comodines, estilo *.class para meter todos los .class de ese directorio.

Este comando armara el archivo.jar que puede ser ejecutado desde un entorno grafico haciendo doble clic o desde consola de sistema, de la siguiente manera:

```
java -jar HolaMundo.jar
```

Ejecución desde archivo de código fuente único JAVA 11

Para ejecutar un programa Java es necesario compilarlo a *bytecode* y posteriormente ejecutarlos. Se necesitan dos pasos. Para facilitar la ejecución de los programas que se componen de un único archivo de código fuente se añade la posibilidad de lanzar un programa desde el archivo de código fuente. Esto es útil par programas pequeños o para los casos de estar aprendiendo el lenguaje.

```
public class Test{
```

```
            public static void main(String[] args){
                  System.out.println("Hola Mundo!!");
            }
      }

//Desde consola ejecutamos con java Test.java en un solo paso
//Revisar permiso de acceso a los archivos en el Sistema Operativo.
```

Compiladores e intérpretes

Los compiladores funcionan leyendo el programa de un archivo fuente almacenado en disco, resuelven todas las sentencias definidas para el compilador y generan un archivo en formato .obj. Estos archivos no están todavía en binario, sólo están en ese formato las instrucciones que pudo resolver el compilador. El resto se dejan como marcas para que las resuelva el enlazador.
El enlazador recorre el archivo generado por el compilador, enlaza los llamados a procedimientos y/o funciones y busca en las librerías las funciones que puedan encontrarse almacenadas allí cuyos llamados se encuentre en el archivo .obj. Además, genera la estructura básica de todo programa ejecutable para volcarlo como salida final del procesamiento. Este último paso es el que deja el archivo totalmente en binario y ejecutable por la plataforma en donde se desarrolló la aplicación
Cuando el código se interpreta en lugar de compilarlo y enlazarlo, el intérprete lee cada instrucción, la traduce e intenta ejecutarla siguiendo el mismo esquema de funcionamiento que un programa compilado y enlazado, emulando la estructura que diseña el enlazador para ese tipo de programas. Esto deriva en que la ejecución de un programa interpretado sea muy lenta.
En Java, para acelerar este proceso, el código pasa por un proceso de compilación de manera de optimizar el trabajo de interpretación, disminuyendo considerablemente el tiempo de procesamiento.

El compilador de Java

El compilador toma el código fuente Java y genera los bytecodes que componen el archivo de salida .class. Los bytecodes son código de instrucciones de máquina para la Java Virtual Machine, por lo tanto, todo componente de la tecnología de Java termina siendo un conjunto de estas instrucciones, las cuales están definidas para implementar una máquina virtual y se componen de:

- *Un conjunto de instrucciones*
- *Un conjunto de registros*
- *Un formato para los archivos de clases*
- *Un Stack*
- *Un heap con un recolector de basura (garbage collector)*
- *Un área de memoria*

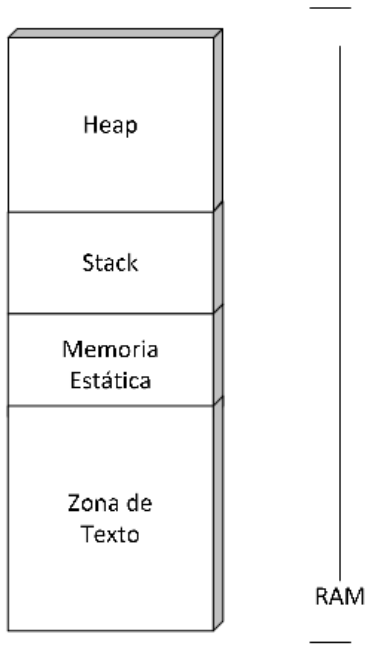
Recolección de basura (Garbage Collection)

Muchos lenguajes de programación permiten alojar memoria dinámicamente en tiempo de ejecución. Este proceso varía basado en la sintaxis del lenguaje, pero siempre involucra un puntero que almacena la dirección donde comienza el bloque de memoria.
Una vez que no se utiliza más la memoria que se alojó se debe liberar este recurso solicitado por el programa, caso contrario, el recurso queda tomado y no estará disponible para otros requerimientos de memoria.
La responsabilidad de liberar el recurso queda en muchos lenguajes en manos del programador. Si este omite el hecho de la liberación, el programa puede correr peligro de quedarse sin memoria e incluso de terminar anormalmente.

Para evitar este tipo de problemas, Java realiza la liberación de recursos automáticamente mediante un proceso llamado “garbage collector”, el cual se realiza automáticamente sobre toda la memoria pedida. La implementación de este proceso varía según la plataforma en la cual este implementada la máquina virtual.

¿Cómo se maneja un programa en memoria?

La estructura de todo programa en memoria es la que muestra la siguiente figura



Donde:
Zona de Texto: Es el lugar donde se almacenan todas las instrucciones de programa
Memoria Estática: Es donde se almacenan las variables globales (según el lenguaje) y las estáticas.
Stack: Es el lugar donde se almacenan los parámetros y variables locales de un método. Se agranda o reduce dinámicamente en cada invocación a un método.
Heap: Es la memoria asignada a un programa que no se encuentra en uso actualmente pero que se puede pedir dinámicamente.

¿Dónde se almacenan y qué son las variables?

Como el lenguaje es interpretado, el funcionamiento de la estructura básica que diseña un enlazador para un programa compilado y enlazado es emulado por el intérprete.
Las variables son los espacios de almacenamiento en memoria que provee el lenguaje.
Todas las variables declaradas en Java se almacenan en el stack. Esto quiere decir que salvo dos excepciones, el único espacio de almacenamiento será en el stack.
Una excepción, como se verá posteriormente, son las variables que se encuentran dentro de los objetos. Todos los objetos en Java se crean pidiendo memoria dinámicamente, por lo tanto, su espacio de almacenamiento es en el heap.
La segunda excepción son las variables estáticas, las cuales se almacenan en la memoria estática.
En un lenguaje de programación orientado a Objetos como Java, las variables son directamente asociadas a los atributos de las clases, aunque se utilicen en otros lugares que no sean directamente atributos, como los parámetros de un método. Si bien es cierto que en otros lenguajes orientados a

objetos existen variables fuera de las clases, en Java no. Por lo tanto en Java hablar de variables o atributos es lo mismo salvo que estén en un método.

Variables

En Java se puede hacer una primera clasificación de las variables según su procedencia en dos grupos:

- **Tipos primitivos de Java.**
- **Tipos Referenciados.**

La principal diferencia entre estos dos grupos es que el primero son los tipos de datos prexistentes definidos en el lenguaje mientras que los segundos son aquellos que pueden almacenar las referencias que se obtienen al crear un objeto de cualquier clase que se haya definido.
Es necesario mencionar un área gris en el lenguaje. Hay ciertos tipos referenciados, o sea, que provienen de clases definidas, que el lenguaje los maneja como tipos prexistentes. La razón de esto es que dichas clases se consideraron con la importancia suficientes como para que el lenguaje las maneje internamente. Este es el caso del tipo String, como se verá posteriormente.

Propiedades de las variables

Las variables en Java poseen propiedades que permiten su control y manejo a lo largo de un programa. Las propiedades de una variable son las siguientes:

- Son espacios en memoria que almacenan datos.
- Siempre tienen almacenado un valor desde su declaración.
- Siempre deben ser declaradas por un tipo primitivo o referenciado.
- Poseen visibilidad y alcance (lugares desde donde se pueden acceder y lugares desde los cuales no).
- Se pueden utilizar como parámetros de los métodos.
- Se puede retornar el valor que almacenen desde un método.

El manejo de estas propiedades es lo que garantiza el buen uso de las mismas, por lo tanto, si se puede asociar una variable con los atributos de una clase, es fundamental dominar el concepto para saber donde utilizarlas y como.

Variables: Tipos primitivos y referenciados

Todo elemento que tenga las propiedades de una variable es un tipo. Por ejemplo, un objeto se puede acceder porque es un tipo referenciado (para acceder a un objeto se almacena una referencia en una variable del tipo de la clase del objeto a referenciar) y tiene las mismas propiedades de una variable. Además de los tipos referenciados, existen los tipos primitivos de Java.

Tipos de datos primitivos.

Es importante saber que estos son tipos de datos del lenguaje y que no representan objetos. Cosa que sí sucede con el resto de elementos del lenguaje Java.

boolean
El tipo de dato boolean se utiliza para almacenar las palabras claves true o false, es decir verdadero o falso. Ocupan 1 byte en memoria.

char
El tipo de dato char se utiliza para almacenar un solo caracter, del tipo Unicode. Ocupan 2 bytes en memoria.
Unicode 10 Java 11
Tratar texto es una parte importante casi de cualquier aplicación, este soporte de Unicode 10 añade 16018 nuevos caracteres soportados, 128 nuevos emojis y 19 símbolos nuevos para el estándar en televisiones 4K.

byte
El tipo de dato byte es un tipo de dato numérico y entero, se utiliza para almacenar números comprendidos entre -128 y 127. Ocupa 1 byte de memoria.

short
El tipo de dato short es un tipo de dato numérico y entero, se utiliza para almacenar números comprendidos entre -32768 y 32767. Ocupa 2 bytes de memoria.

int
El tipo de dato int es un tipo de dato numerico y entero, se utiliza para almacenar numeros comprendidos entre -2.147.483.648 y 2.147.483.647. Ocupa 4 bytes de memoria.

```
//Impresión de valores mínimos y máximos de un tipo de datos
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE);
```

long
El tipo de dato long es un tipo de dato numérico y entero, se utiliza para almacenar numeros comprendidos entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Ocupa 8 bytes de memoria.

float
El tipo de dato float es un tipo de dato numérico y de punto flotante, se utiliza para almacenar números comprendidos entre -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38. Ocupa 4 bytes de memoria, y maneja entre 6 y 7 cifras decimales.

double
El tipo de dato float es un tipo de dato numérico y de punto flotante, se utiliza para almacenar números comprendidos de 1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308. Ocupa 8 bytes de memoria, y maneja unas 15 cifras decimales.
Nota: Los tipos de datos primitivos de Java no son nulleables, es decir no se le puede asignar null

```
int x=null; //no compila
```

Comentarios

En Java existen tres formas de poner comentarios dentro del código y son los siguientes

```
// La doble barra se usa si el comentario abarca una línea
/*
      Bloque
      de
```

```
            Comentarios
      */
```

La tercera alternativa es igual que la segunda. La primera diferencia radica es que comienza con /** , y la segunda es que este tipo de comentarios los lee un programa de utilidad llamado javadoc y lo utiliza para documentar la clase, como muestra el siguiente ejemplo

```
/**
      Comentario JavaDoc
*/
```

Es para que lo lea el javadoc, lo que el programa de utilidad lee en comentarios como el anterior, lo incluye en archivos HTML que conforman la documentación de la o las clases sobre las que se ejecuto javadoc.
Este comentario es muy importante para la documentación de un proyecto, Las IDEs se valen del javaDoc para asistir al programador, y a la vez es un comentario visible para actores de proyecto que no tienen acceso al código fuente (analistas, diseñadores y testers).

Identificadores

Hay una regla muy simple para determinar si algo es un identificador:

> *Cuando en el código un programador debe decidir que nombre ponerle a un elemento, dicho elemento es un identificador.*

Como se puede apreciar, dentro de esta característica entran varios elementos antes mencionados, como ser, nombres de clases, variables, métodos, etc...
Hay otros elementos del lenguaje Java todavía no mencionados que son identificadores.
Estos se irán descubriendo como tales a medida que se aprenda más del lenguaje siguiendo la simple regla antes mencionada.
Es preciso aclarar que cualquier palabra que el lenguaje defina como “reservada o clave”, no podrá ser utilizada como identificador.
Por otro lado, crear un identificador es asignarle un nombre a un elemento que permite definir un lenguaje de programación. Esta asignación de nombres debe seguir ciertas normas prestablecidas en el formato del mismo que variarán de lenguaje a lenguaje. En el caso de Java, las reglas a seguir son las siguientes:

El primer carácter de un identificador debe ser uno de los siguientes:

- **Una letra en mayúsculas (A~Z)**
- **Una letra en minúsculas (a~z)**
- **El carácter de subrayado (_)**
- **El símbolo pesos o dólar ($)**

Del segundo carácter en adelante:

- **Cualquier elemento de los que sirve para el primer carácter**
- **Caracteres numéricos (0~9)**

Vale la pena mencionar que el espacio en blanco no es un carácter permitido como se puede apreciar, por lo tanto no debe utilizarse para nombrar un identificador.

```
        int _nro1=2;    //valido
```

```
int _1nro=2;    //valido
int $1nro=2;    //valido
//int 1nro=2;   //no valido
int nro1_=2;    //valido
//int nro 1=2;  //no valido
```

Además existen ciertas normas estandarizadas para crear identificadores, que si bien no son obligatorias, han sido adoptadas por la mayoría de los programadores y seguirlas ayudan mucho a la legibilidad del código. Algunas de ellas son las siguientes

- Si es una clase, el nombre debe comenzar con mayúsculas
- Si es un método o una variable, el nombre debe comenzar con minúsculas.
- Si el nombre tiene más de una palabra, a partir de la segunda palabra separarlas sólo comenzando con mayúsculas (primera letra de cada palabra a partir de la segunda)
- Las constantes se escriben con mayúsculas y las palabras se separan con un símbolo de subrayado
- Los paquetes se escriben en minúsculas

No se debe olvidar el hecho de que Java es un lenguaje case sensitive (diferencia entre mayúsculas y minúsculas), por lo tanto si dos identificadores son iguales en su significado a la lectura pero difieren tan sólo en el caso de una letra, el lenguaje los considerará identificadores distintos.

Palabras Reservadas

Como en todo lenguaje existen palabras reservadas que no se pueden usar como identificador.

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	false	final	finally
float	for	goto	if	implements
import	instanceof	int	interface	long
native	new	null	package	private
protected	public	return	short	static
strictfp	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while		

Convenciones para codificar

Si bien muchos temas se irán desarrollando posteriormente, para sentar bases, se pueden mostrar algunos ejemplos de las convenciones que normalmente se utilizan al codificar Ejemplo

```
Paquetes:       package curso.java.inicializar;

Modulos:        module ejemplo {}       //A partir de Java 9

Clases:         class Horario {}

Interfaces:     interface MiInterfaz {}

Métodos:        agregarDias(int masDias){}

Variables:      Date nuevaFecha;
```

Constantes: `int VALOR_MAXIMO;`

Tipos primitivos

Las declaraciones de las variables se realizan con el siguiente formato:

```
[<modificador>] <tipo primitivo> <identificador> [= valor inicial];
```

En otras palabras:
Si aparece “<>”, quiere decir “elegir uno entre los posibles”
Si aparece “[]”, quiere decir que es opcional
Si una palabra o símbolo aparece sin ninguna otra cosa, indica que ponerlo es obligatorio

Este formato debe leerse de la siguiente manera:
Modificador: Opcional. Es el modificador de visibilidad de la variable (indica como se lo puede utilizar y desde donde). Recordar que no poner nada se interpreta como el modificador por defecto (o sea, se declara un modificador implícitamente). Las posibilidades son:

- public
- private
- protected
- Sin modificador

Es obligatorio poner un tipo, pero se debe elegir uno de los posibles.
Es obligatorio poner un identificador, se debe elegir el nombre a poner.
Se posee la opción de elegir poner un “=” y un valor si se desea que la variable tenga un valor inicial.
Siempre hay que finalizar la declaración con un “;” que indica fin de línea de programa.

Nota: *En este punto es bueno señalar que en Java toda sentencia termina con un “;”. El otro símbolo que indica el fin de algo es la “}”, pero se utiliza para ciertos tipos de declaraciones que se llaman bloques.*

Ejemplo

```
int nroEmpleados=10;
int edad=20;
float sueldoFijo;
long documento;
int a=4, b=3, d=66;     //declaración y asignación multiple
```

Nota: *Cuando a una variable se le asigna un valor numérico dentro del código, dicho número se lo identifica como “literal de asignación”. El lenguaje toma esos caracteres y los convierte al valor binario que debe almacenar en la memoria para la variable.*

Strings
Las cadenas de caracteres o “strings” en Java se manejan con una clase interna del lenguaje, por lo tanto, siempre será un tipo referenciado.
El lenguaje permite varias formas de asignar una cadena de caracteres:

Ejemplo:

```
String nombre = new String("Juan");
```

```
	String apellido = "Perez";
	char[] vector = {‘h’,’o’,’l’,’a’};
	String saludo = new String(vector);
	String texto = null;    //Los Strings son nulleables
```

Internamente un String almacena un array de caracteres (unicode 2 bytes). A partir de java 9 hay un cambio importante llamado “Compact String” que almacena un array de bytes.
Hasta java 8 el String era almacenado como:

```
	private final char[] value;
```

A partir de java 9 el String es almacenado como:

```
	private final byte[] value;
```

```
A partir de Java 10 se añade soporte para unicode10
```

Inicialización de los tipos de datos

Un ejemplo de manejo de valores de asignación es el que se muestra a continuación.

```
public class Asignacion {
	public static void main(String args[]) {
		// declaración de variables enteras
		int x, y;
		// declaración y asignación de variables de punto flotante
		float z = 3.414f;
		// declaración y asignación de double
		double w = 3.1415;
		// declaración y asignación de boolean
		boolean verdadero = true;
		// declaración de variable de caracter
		char c;
		// declaración de variable String
		String str;
		// declaración y asignación de String
		String str1 = "chau";
		// asignación de valores a un char
		c = 'A';
		// asignación de valores a un String
		str = "Hola!";
		// asignación de valores a un int
		x = 6_003_334;
		y = 1000;

		//Asignación de enteros a partir de java 7
		int nro2=1_000_000;           //java 7
		//int nro3=_123;              //no compila
		//int nro4=123_;              //no compila
		int nro5=1_00_000;            //compila


	}
}
```

Tipo de datos var JDK10

Java 10 adiciona la nueva palabra reservada *var*, esto ayuda a no tener que repetir varias veces los tipos en la construcción de un objeto. La inferencia de tipos es la idea que permite al compilador obtener el tipo estático sin que sea necesario escribirlo de forma explícita.

```
//Tipo de datos var JDK10 variables locales.
//Solo es permitido su uso en variables locales.

var v1=1;         // int
var v2=4.25;      // double
var v3=true;      // boolean
var v4='a';       // char
var v5="Hola";    // String
var v6=100000L;   // long
var v7=4.25f;     // float
var v8=1_000_000; // int java 7

//v1=3.45;        // Error no puedo cambiar el tipo de datos
v1=4;

System.out.println(v1);
System.out.println(v2);
System.out.println(v3);
System.out.println(v4);
System.out.println(v5);
System.out.println(v6);
System.out.println(v7);
System.out.println(v8);


//Ejemplo de var para referenciar objetos o vectores
var objeto=new Object();
System.out.println(objeto);
var vector=new int[4];
vector[0]=3;
for(int i:vector) System.out.println(i);


//Ejemplo de var no se puede usar como tipo de datos de parámetros
//de métodos o funciones. Ni tampoco como tipo de datos de atributo.
public class Test {
      //private static var varX=23;
      //error no se puede usar var como tipo de datos de atribuo
      public static void main(String[] args) {
            var var="Hola";
            System.out.println(var);
            funcion2(var);
      }

      //public static void funcion1(var var){
      //No se permite usar var como tipo datos en parametros de
      //entrada.
      //    System.out.println(var);
      //}
      public static void funcion2(String var){
            System.out.println(var);
      }
}
```

Uso de Bloques, Espacios en Blanco y Finalización de Sentencia

Una sentencia se compone de una o más líneas terminadas con un punto y coma (;):

```
totales = a + b + c + d + e + f;
```

El compilador separa las sentencias o llamados a función por cada “;” que encuentre.
Por otra parte, las sentencias del lenguaje se colocan dentro de bloques, los cuales se definen con un par de llaves. Por lo tanto, se puede definir un bloque como una colección de sentencias limitadas por la apertura y cierre de llaves:

```
{
      x = y + 1;
      y = x + 1;
}
```

Otra característica de los bloques es que definen sus propias visibilidades, por lo tanto las variables declaradas dentro de ellos tienen alcance del bloque.
Los bloques se pueden anidar, por lo tanto, cuando se anidan bloques los que están anidados ven las variables de los bloques que los contienen.

```
{
      int x = 1;
      System.out.println(x);
}
// System.out.println(x);     //esta sentencia da error, por que la variable
                              //esta fuera de scope (alcance).
```

Los bloques se utilizan para separaciones sintácticas, como por ejemplo, el contenido de una clase:

```
public class Fecha {
      private int dia;
      private int mes;
      private int anio;       //no es adecuado usar caracteres especiales
                              //(ñ o Ñ), usamos los caracteres que están por
                              //dejado del 127 en la tabla ascii
}
```

Se pueden utilizar los espacios en blanco que se necesiten sin que afecten el código.

¿Dónde empiezan los programas y por qué?

En Java, el comienzo de un programa se coloca dentro de una clase en un método. A diferencia de otros métodos escritos por el programador este tiene un nombre preestablecido: main . Este método es el punto de entrada para el comienzo de un programa.
Como no esta definido que un programa deba comenzar en una clase en particular, esto indica que puede haber muchas clases que posean el método main , pero sólo puede haber un método de este tipo por clase. El intérprete de Java sabe por donde arrancar el programa porque el primer argumento que recibe para empezar a ejecutarlo debe ser la clase que posee el método main que se desea utilizar como comienzo de programa.

Ejemplo

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

```
            System.out.println("Inicio del programa.");
            System.out.println("Hola Mundo!!");
            Persona p = new Persona();
      }
}
```

Caracteres especiales – Secuencias de Escape

La representación de caracteres especiales como el salto de línea o la tabulación, se logran a partir de la barra (\). Los caracteres especiales más utilizados son:

- `\n : Nueva línea.`
- `\t : Tabulador.`
- `\’ : Comilla simple.`
- `\” : Comilla doble.`

Entorno de desarrollo integrado (IDE)

Los IDE están diseñados para maximizar la productividad del programador proporcionando componentes muy unidos con interfaces de usuario similares. Los IDE presentan un único programa en el que se lleva a cabo todo el desarrollo. Generalmente, este programa suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de software. Esto contrasta con el desarrollo de software utilizando herramientas no relacionadas.
Uno de los propósitos de los IDE es reducir la configuración necesaria para reconstruir múltiples utilidades de desarrollo, en vez de proveer el mismo set de servicios como una unidad cohesiva. Reduciendo ese tiempo de ajustes, se puede incrementar la productividad de desarrollo, en casos donde aprender a usar un IDE es más rápido que integrar manualmente todas las herramientas por separado.

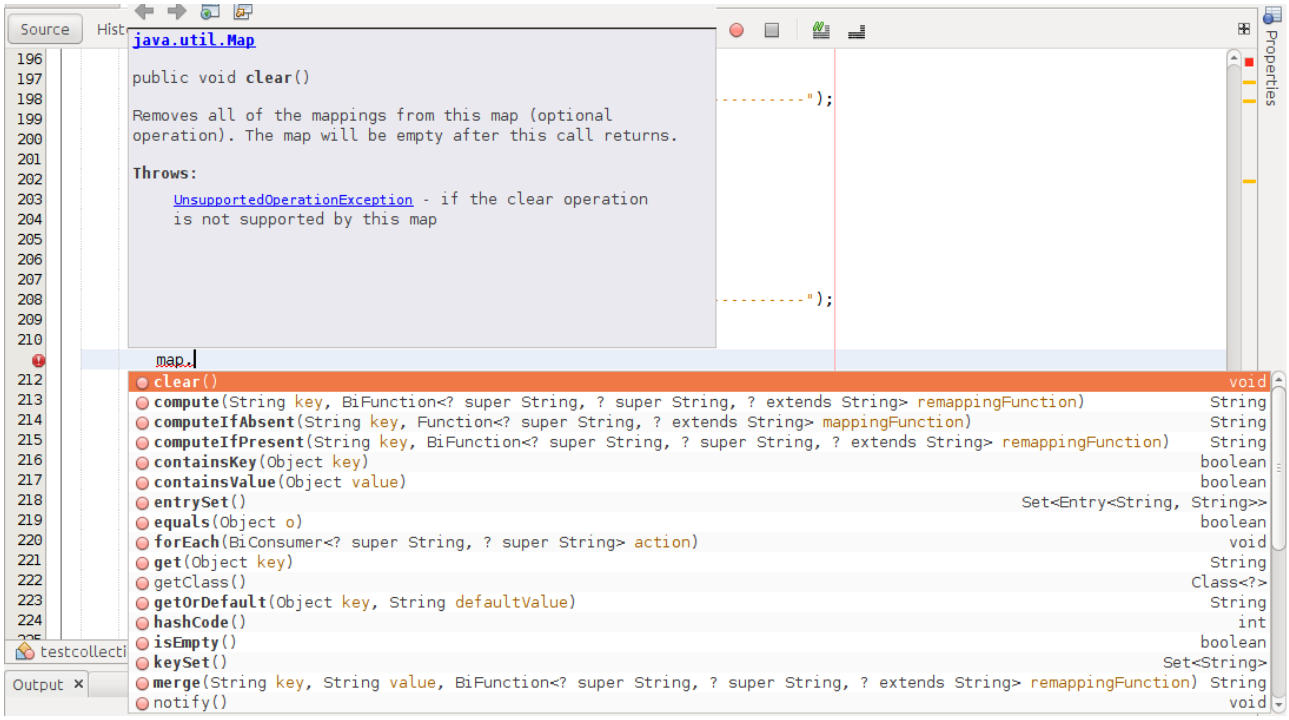
IDEs para Java

En java tenemos varios IDEs de desarrollo libre, Eclipse, Netbeans IDE, IntelliJ.
Para lograr un mejor aprovechamiento de este material, se recomienda tener un IDE instalado.
Los códigos fuentes de este apunte están desarrollado con Netbeans IDE, pero el código java generado es estándar y puede leerse desde cualquier IDE, utilizando un asistente de migración de proyecto.

Ventajas de trabajar un IDE

- Debugear el código.
- Predictivo de lenguaje (InteliSense).
- Ir al archivo donde esta declarada cierta función o variable y explorar código interno del Java.
- Validación de código (Error de sintaxis).
- Conocer los ficheros en los que tenemos algún error de sintaxis.
- Refactorización de código.
- Conocer las funciones declaradas en una clase u script.
- Por lo general tendrá soporte para lenguajes o script como: javaScript, css, sql, etc.
- Cliente de SVN, GIT integrado (En algunos).
- Maven integrado.
- Cliente FTP (En algunos ya sea por default o con algún plugin).
- Posibilidad de agregar plugins o escribir plugins con utilidades nuevas.
- Graficadores UML activos.

En la imagen se ve como funciones el predictivo (intelSense) en un IDE.



Calendario de liberación de versiones de Apache Netbeans.

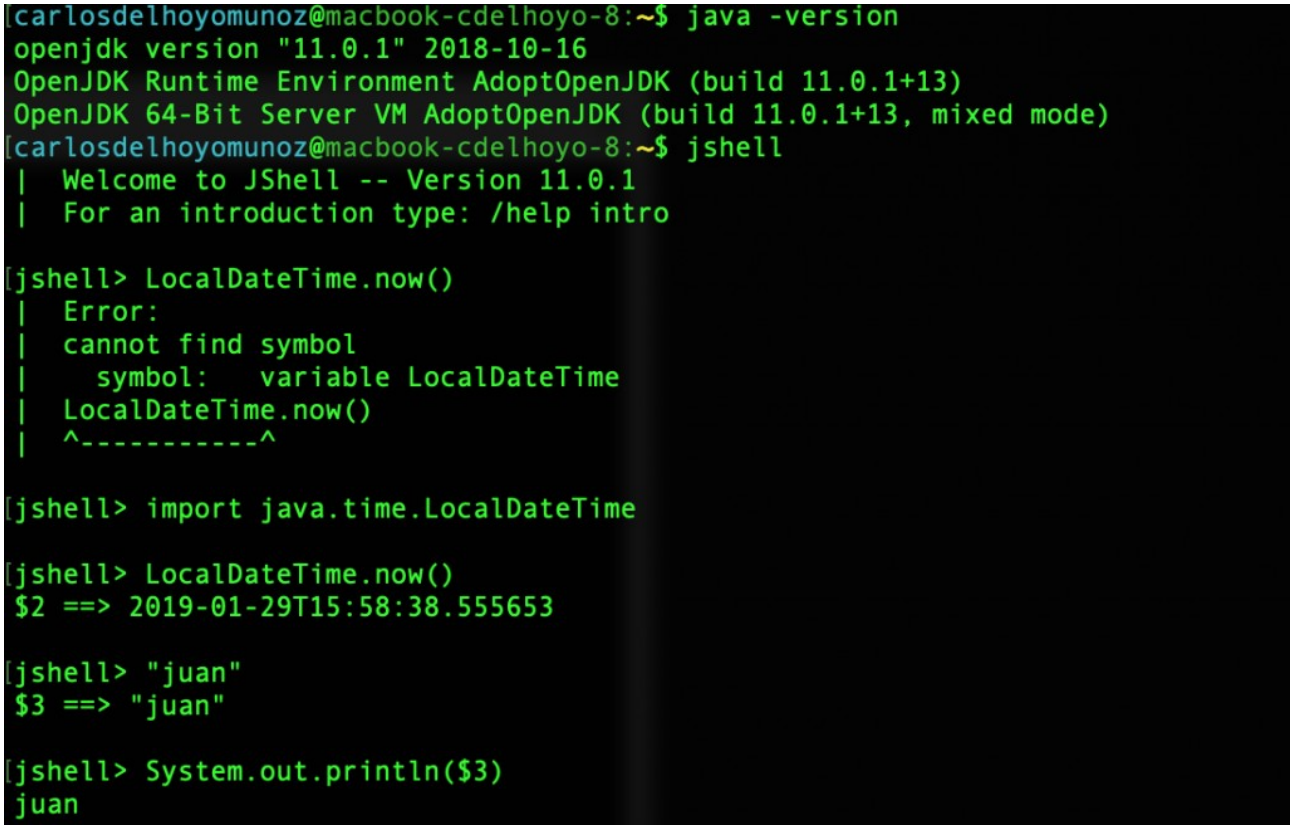
Netbeans es uno de los IDEs más utilizados, el mismo es libre, hasta la versión 8.2 tenia soporte de Oracle y en la versión 9 Apache le brinda soporte a este IDE.
Al día de la fecha Apache no lanzo una versión LTS del producto pero tiene un cronograma de liberación de versiones publicado en el sitio https://netbeans.org/

Date	Release	Description

June 15th		Feature freeze. Merge window closes. Only fixes intended for next release to be merged into master. Release branch <version>.1 created and beta build(s) made available. NetCAT lite???
July 7th		NB <version>.1 voting candidate created and release vote commences.
July 15th	**NetBeans 11.1 (in 2019)** **NetBeans 12.1 (in 2020)**	**NB <version>.1 released.**
July 21st		NB <version>.2 merge window opens.
September 15th		Feature freeze. Merge window closes. Release branch <version>.2 created and beta build(s) made available.
October 7th		NB <version>.2 voting candidate created and release vote commences.
October 15th	**NetBeans 11.2 (in 2019)** **NetBeans 12.2 (in 2020)**	**NB <version>.2 released.**
October 21st		NB <version>.3 merge window opens.
December 15th		Feature freeze. Merge window closes. Release branch <version>.3 created and beta build(s) made available.
January 7th		NB <version>.3 voting candidate created and release vote commences.
January 15th	**NetBeans 11.3 (in 2020)** **NetBeans 12.3 (in 2021)**	**NB <version>.3 released.** (Start NetCAT with this release here???)
January 21st		NB <version+1>.0 merge window opens. NetCAT announced and signups begin.
February 15th		Feature freeze. Merge window closes. Release branch <version+1>.0 created and beta build(s) made available. NetCAT test spec review starts.
February 21st		NetCAT testing phase starts.
March 21st		NetCAT testing ends. NetCAT community acceptance vote.
April 7th		NB <version+1>.0 voting candidate created and release vote commences.
April 15th	**LTS** **NetBeans 12.0 (in 2020)** **NetBeans 13.0 (in 2021)**	**NB <version+1>.0 LTS released.**
April 21st		NB <version+1>.1 merge window opens. GOTO 10

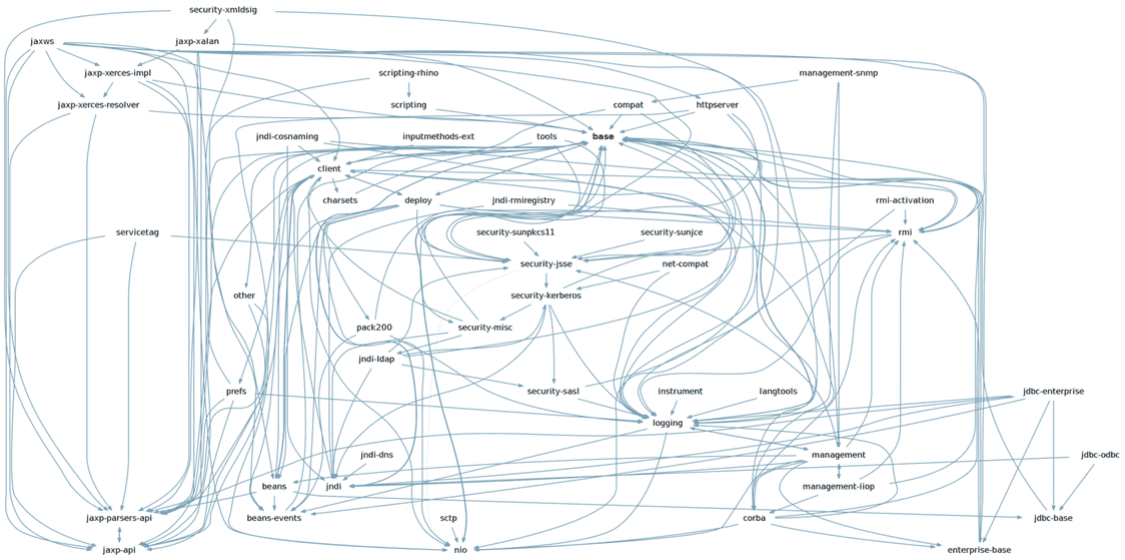
Comando jshell del terminal JDK 9

En java 9 tenemos un nuevo comando disponible en nuestro terminal, el comando *jshell.* Con este comando podemos probar directamente por consola cualquier sentencia de java sin necesidad de un IDE. Como vemos en el siguiente ejemplo, podemos usar variables, realizar imports o cualquier sentencia que se nos ocurra. Además nos da detalles si encuentra algún error en el código.

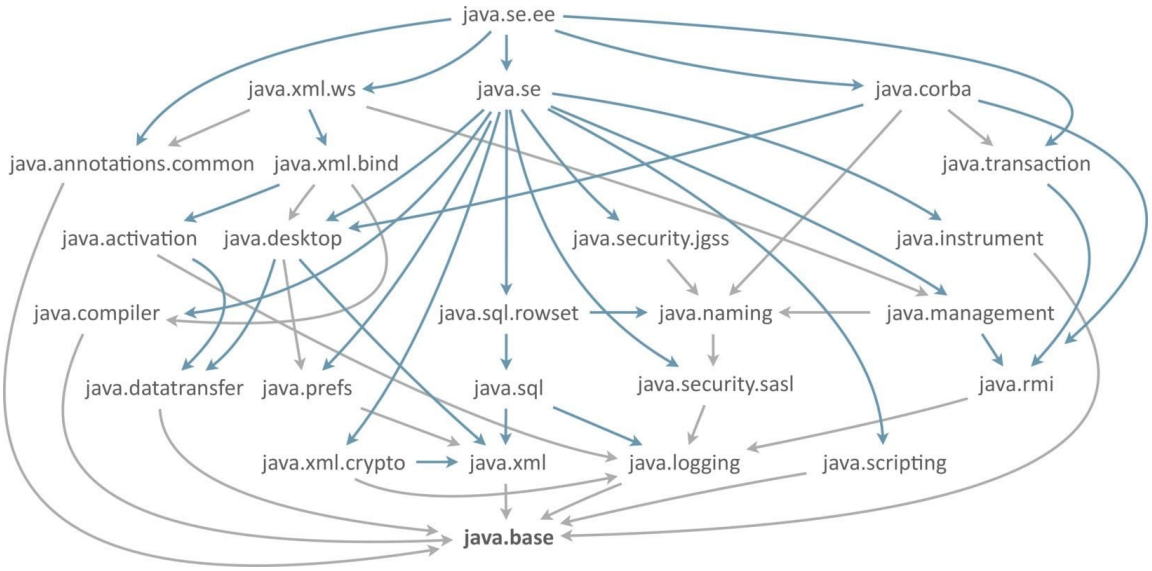


Modularidad JDK 9

En el año 2000 el JDK tenía múltiples dependencias entre paquetes no relacionados:



Esto ocurre porque en un proyecto en el que no haya límites claros –como una división en librerías– es fácil tomar atajos que creen dependencias sin que nadie lo note. Incluso en un proyecto con desarrolladores disciplinados como el JDK, hicieron falta años para refactorizar hacia un sistema modular.
En 2017, el JDK modularizado tiene este aspecto (versión simplificada):



El JDK 9 tiene 94 módulos. Puedes listarlos ejecutando:

Shell

```
$ java --list-modules java.activation@9 java.base@9 java.compiler@9 java.corba@9
...
  $ java --list-
  modules

  java.activation@9
  java.base@9
  java.compiler@9
  java.corba@9
  ...
```

Verás que los módulos tienen los siguientes prefijos:

`java`	Módulos del núcleo de la plataforma Java. Estos son los módulos del lenguaje, módulos enterprise, o *módulos agregadores* que agrupan los dos anteriores.
`javafx`	Módulos de Java FX.
`jdk`	APIs y herramientas del JDK como el compilador, javadoc, y consola. No son parte de la especificación del lenguaje.
`jdk.incubator`	Módulos experimentales sujetos a cambios. Por ejemplo, `jdk.incubator.httpclient`.
`jdk.unsupported`	Tipos no soportados y que pueden ser eliminadas en cualquier momento. Por ejemplo, `sun.misc.Unsafe` .
`oracle`	Módulos específicos de la implementación de Oracle del JDK.

Los **módulos agregadores** son módulos sin código propio que agrupan a otros módulos mediante sentencias `requires transitive`. En el JDK hay dos:

- java.se: Java Standard Edition
- java.se.ee: Java Enterprise Edition

Un proyecto bien diseñado no debería requerir los módulos agregadores, a menos que necesite la plataforma entera, lo cual es improbable.