

Unidad

1

DIPLOMATURA EN PROGRAMACION JAVA

Capítulo 1



Introducción

Introducción

En Este Capítulo

- ¿Qué es Java?
- ¿Cómo se crea un programa en Java?
- Compiladores e intérpretes
- Recolección de basura (Garbage Collection)
- ¿Cómo se maneja un programa en memoria?
- ¿Dónde se almacenan y qué son las variables?
- Comentarios
- Identificadores
- Tipos primitivos
- Tipos referenciados
- Dónde se almacenan y que son los métodos
- Clases en Java
- Uso de Bloques, Espacios en Blanco y Finalización de Sentencia
- Constructores
- Objetos y Mensajes
- Strings
- Ocultamiento de la información
- Encapsulado
- ¿Dónde empiezan los programas y por qué?
- Paquetes
- Construcción e Inicialización de Objetos
- Palabras clave

¿Qué es Java?

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
- Un entorno de desarrollo
- Un entorno para aplicaciones
- Un entorno para despliegue de aplicaciones

La sintaxis es similar al C++ pero el manejo y la semántica son parecidos al SmallTalk. Se utiliza para desarrollar tanto applets como aplicaciones locales y distribuidas

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.

¿Qué es la máquina virtual?

La máquina virtual de Java tiene una especificación que la describe de la siguiente manera:

Es una máquina imaginaria que se implementa emulando en ella el software y hardware de una máquina real. El código para una máquina virtual de Java se almacena en archivos .class, cada uno de los cuales contiene código para a lo sumo una clase pública

Este enunciado provee a la vez las especificaciones de las plataformas de hardware en las cuales todo código para la tecnología Java se compila. Con esta especificación se asegura que el código sea independiente de la plataforma porque la compilación se realiza para una máquina genérica, la Java Virtual Machine, mientras que en cada plataforma debe poseer una implementación específica para ella de la máquina virtual.

La especificación Java Virtual Machine declara que a puntos importantes de la siguiente manera:

Es el componente de la tecnología responsable de la independencia del hardware y del sistema operativo, que sea pequeño tamaño de su código compilado, y su capacidad para proteger a los usuarios de programas maliciosos.

También establece

La máquina virtual de Java no sabe nada del lenguaje de programación Java, sólo de un formato binario en particular, el formato de archivo de clase (.class). Un archivo de clase Java contiene las instrucciones de máquina virtual (o códigos de bytes – bytecodes –) y una tabla de símbolos, así como otra información complementaria. En aras de la seguridad, la máquina virtual Java impone el formato fuerte y las limitaciones estructurales del código en un archivo de clase. Sin embargo, cualquier lenguaje con funcionalidad que se pueda expresar en términos de un archivo de clase válido puede ser albergado por la máquina virtual Java. Atraídos la disponibilidad en general, independiente de la máquina que sirve de plataforma, los implementadores de otros idiomas están recurriendo a la máquina virtual de Java como un vehículo de entrega para sus lenguajes.

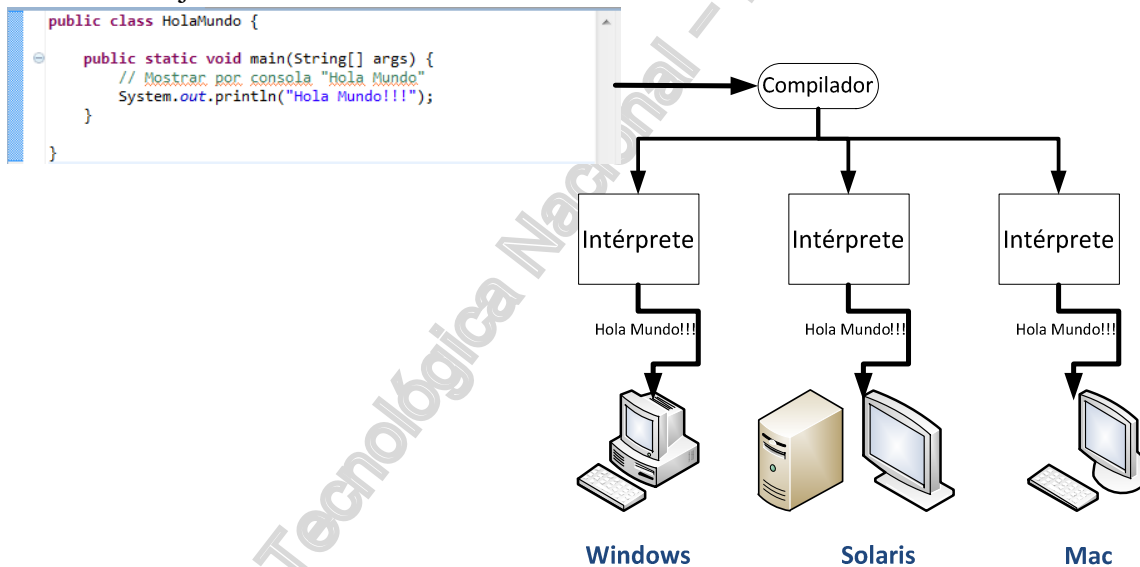
Por eso la máquina virtual de Java es un entorno virtual de ejecución completo y se lo entrega en su formato más simple para ejecución de programas bajo el nombre JRE (Java Runtime Environment)

¿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 compilador 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.

Un ejemplo de esto se muestra en la figura para la creación de un programa llamado "HolaMundo.java"



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!!!");  
    }  
}
```

```
}
```

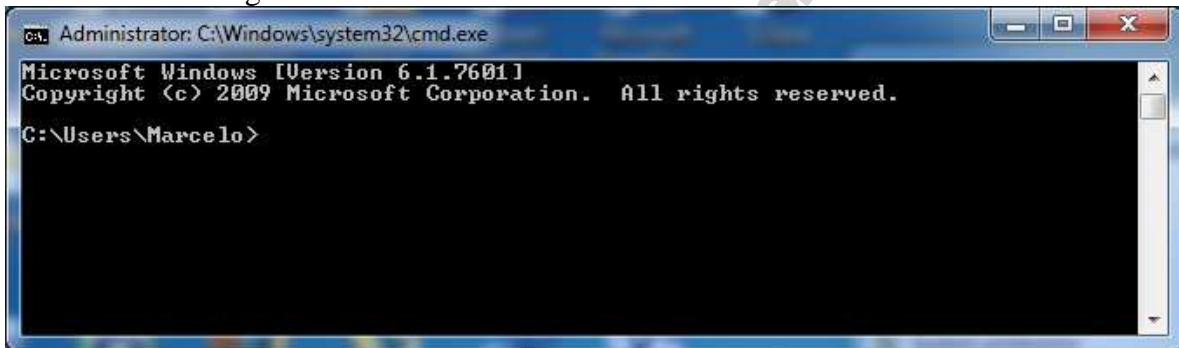
El nombre con el cual se deberá guardar el archivo debe ser HolaMundo.java
Por otra parte, se debe tener cuidado al escribir el código puesto que el lenguaje es sensible al caso, lo cual quiere decir que las mayúsculas y la minúsculas se interpretan diferente, por ejemplo

HOLAMUNDO.JAVA ~~≠~~ **HolaMundo.java**

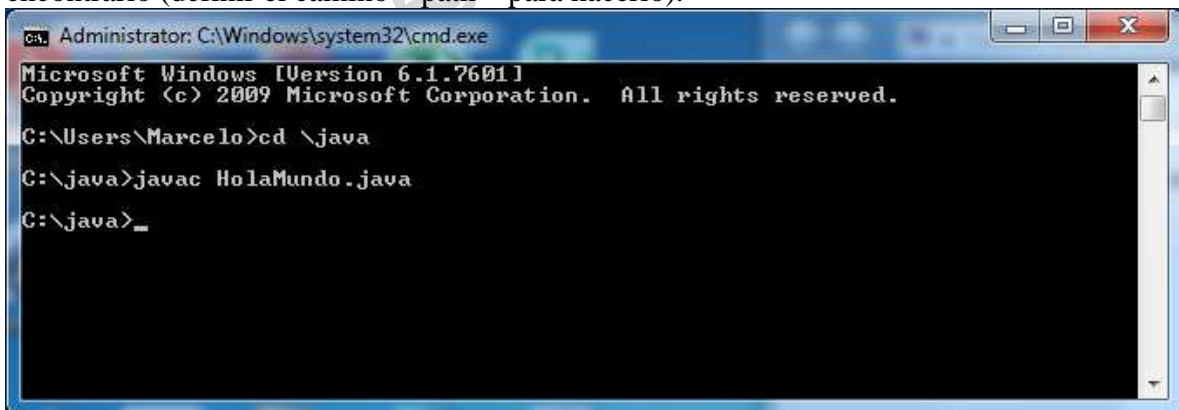
¿Cómo se compila y ejecuta?

Compilar el archivo fuente en un archivo de bytecodes

Por ejemplo, si se encuentra en un sistema operativo Windows, desde el botón Inicio (Start si la plataforma está en inglés) ejecutar la aplicación de consola, cmd.exe. Se despliega una ventana como la siguiente:



Para compilar el archivo fuente, se debe cambiar al directorio en el que se encuentra el archivo y ejecutar el compilador, **javac.exe**, con el cuidado que el sistema operativo pueda encontrarlo (definir el camino – path – para hacerlo).



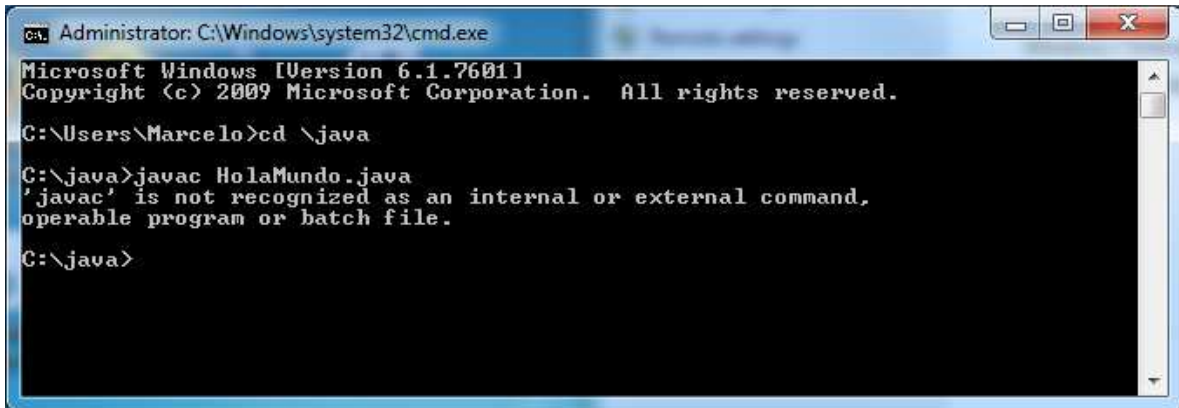
Ahora se puede compilar. En la línea de comandos, escribir el siguiente comando y pulsar Enter:

```
javac HelloWorldApp.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.class, se puede ejecutar el programa

Possible error

'javac' is not recognized as an internal or external command, operable program or batch file.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Marcelo>cd \java

C:\java>javac HolaMundo.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

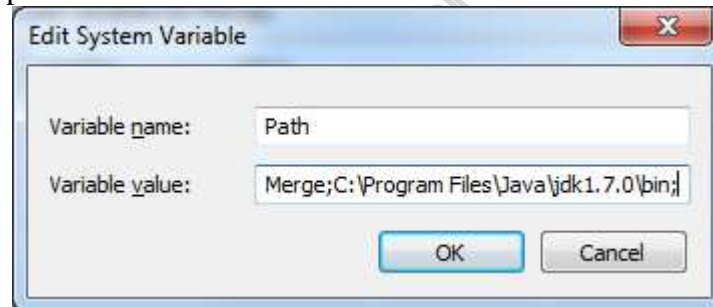
C:\java>
```

Si se produce este error, Windows no puede encontrar el compilador de Java, el javac.

Solución

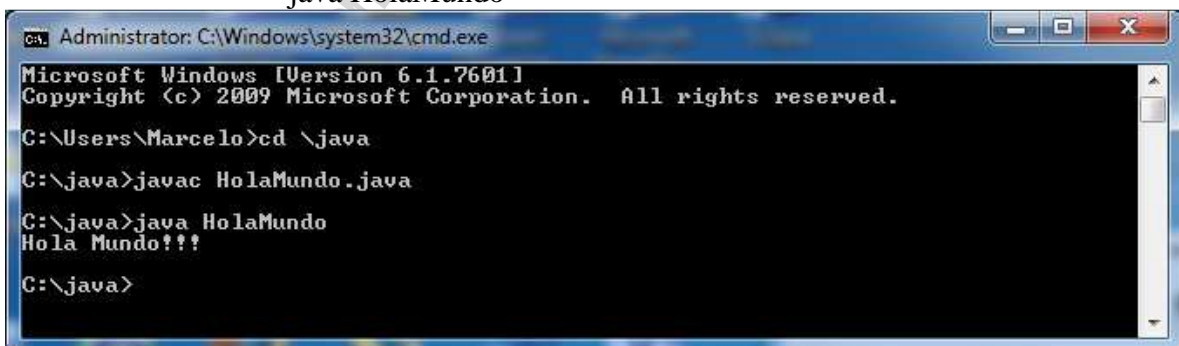
Incluir en la variable PATH el camino (ruta) al directorio donde se encuentra el compilador o invocar al compilador escribiendo el camino (ruta) completo de la unidad y directorio donde se encuentra el compilador javac.exe.

Por ejemplo, una posibilidad en un entorno Windows



Ejecutar le programa contenido en el archivo de bytecodes. En el mismo directorio, escribir en la línea de comandos:

java HolaMundo



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Marcelo>cd \java

C:\java>javac HolaMundo.java

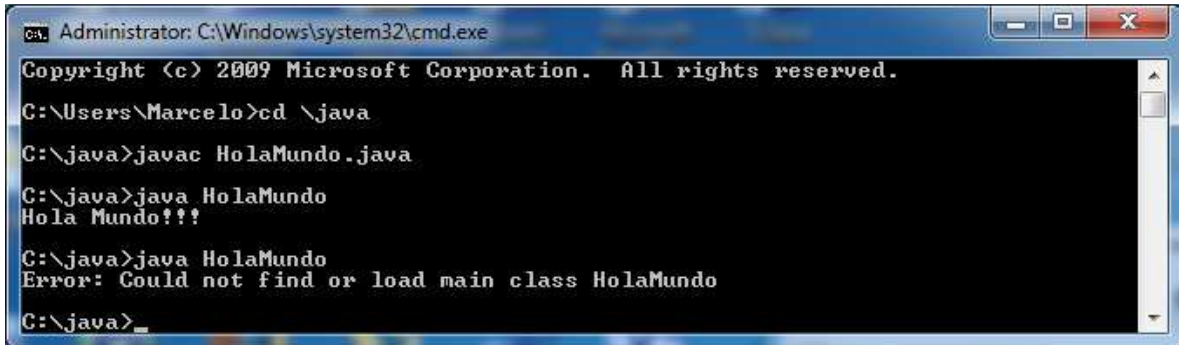
C:\java>java HolaMundo
Hola Mundo!!!

C:\java>
```

Possible error

Error: Could not find or load main class HolaMundo

Si aparece este error, java no puede encontrar el archivo de bytecodes, HolaMundo.class. Esto se puede verificar fácilmente si se borra el archivo y se trata de ejecutar nuevamente, como se muestra a continuación



```
Administrator: C:\Windows\system32\cmd.exe
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Marcelo>cd \java
C:\java>javac HolaMundo.java
C:\java>java HolaMundo
Hola Mundo!!!
C:\java>java HolaMundo
Error: Could not find or load main class HolaMundo
C:\java>
```

Uno de los lugares donde java intenta buscar el archivo de bytecodes es el directorio donde se invoca al intérprete. Por eso, si el archivo de bytecodes está en C:\java, se debería cambiar a ese directorio como directorio actual.

Si todavía persiste el problema, la causa puede ser la variable CLASSPATH. En ella se indica donde el intérprete debe buscar los archivos .class cuando los ejecuta. Si se está en el directorio donde reside el archivo .class y la variable CLASSPATH almacena un directorio diferente, esta puede ser la causa. Se puede limpiar el contenido de la variable ejecutando
set CLASSPATH=

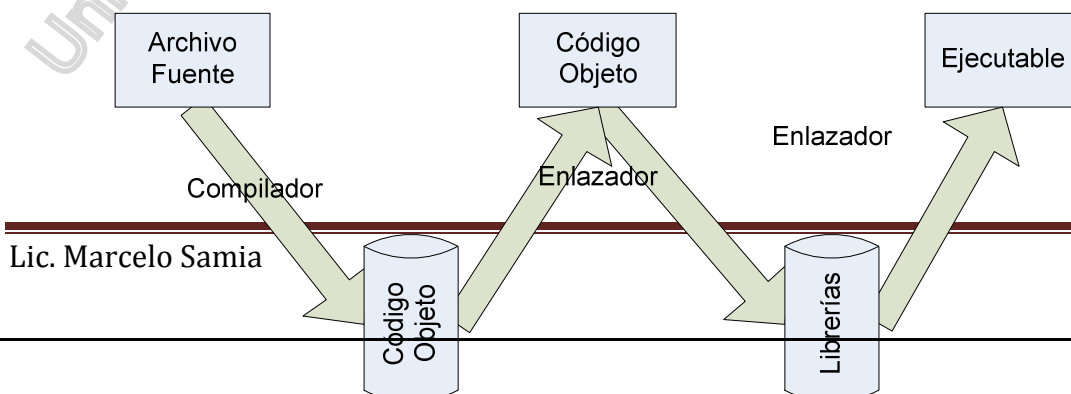
Una vez realizado esto y estando parado en el directorio donde se encuentra el archivo .class, escribir de nuevo java HolaMundo. Si el programa funciona, es señal de que la variable CLASSPATH tiene almacenado un directorio diferente cuando arranca la consola de comando. Ir a la parte del sistema operativo en particular que se tenga y cambiar su valor.

Nota: para realizar todos estos pasos se puede utilizar un entorno de desarrollo integrado. 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.

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

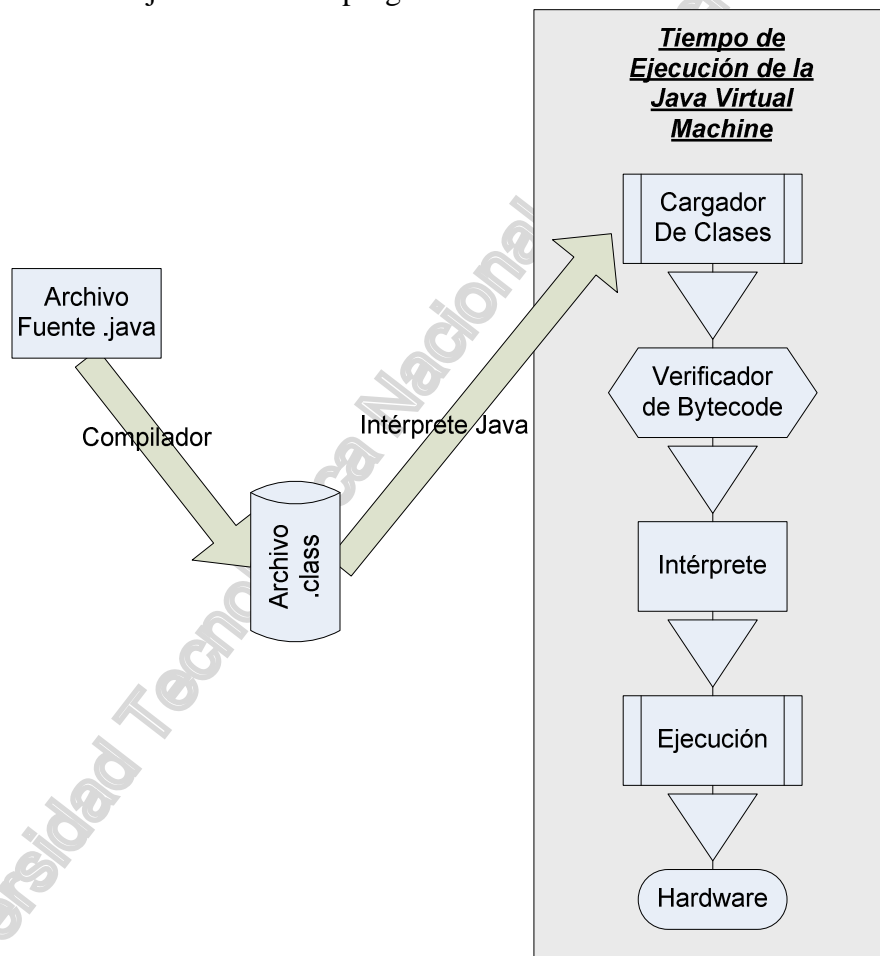


almacenadas allí cuyos llamados se encuentre en el archivo .obj. Además, genera la estructura básica de todo programa ejecutable para volcarlo como salida final del procesamiento. Este último paso es el que deja el archivo totalmente en binario y ejecutable por la plataforma en donde se desarrolló la 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 siguiente gráfico muestra un esquema de cómo se realiza el proceso completo, desde la compilación hasta la ejecución de un programa en Java



A continuación se definen los elementos que intervienen en la ejecución de un programa en Java.

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.

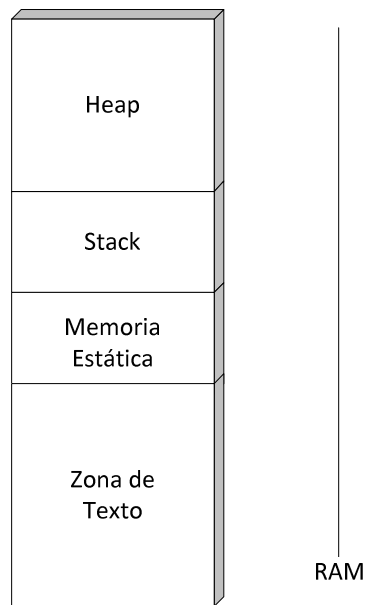
Una vez que no se utiliza más la memoria que se alojó se debe liberar este recurso solicitado por el programa, caso contrario, el recurso queda tomado y no estará disponible para otros requerimientos de memoria.

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 preexistentes 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 preexistentes. 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, los cuales se pueden clasificar según su uso en los siguientes grupos:

- Tipos enteros
 - **byte**
 - **short**
 - **int**
 - **long**
- Tipos de punto flotante
 - **float**
 - **double**
- Tipo texto
 - **char**
- Tipo lógico
 - **Boolean**

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

/* Comienza el comentario
La combinación barra - asterisco para comenzar y asterisco- barra para
finalizar se utiliza el comentario abarca más de una línea
Finaliza el comentario */
```

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

```
/** Comienza el comentario
La combinación barra - asterisco - asterisco es para que lo lea el
javadoc.
Finaliza el comentario */
```

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. Más adelante se retomará el tema de esta utilidad.

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 preestablecidas 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.

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 sensible al caso (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.

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 inicializar;

Clases:

class Horario

Interfaces:

interface MiInterfaz

Métodos:

agregarDias(**int** masDias)

Variables:

nuevaFecha

Constantes:

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;
```

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

Tipos Primitivos: Lógico

Este tipo sólo puede almacenar dos valores: `true` o `false`. Los valores que almacenan son considerados palabras clave en el lenguaje y se deben utilizar para cualquier asignación a variables de este tipo. Generalmente este tipo es utilizado para la toma de decisiones.

Ejemplo

```
boolean verdadero = true;
```

Declara una variable llamada verdadero y le asigna el valor `true`.

Los valores booleanos no se pueden convertir a ningún otro tipo (salvo String), así como tampoco ningún otro tipo se puede convertir a booleano.

Tipos Primitivos: `char`

Este tipo se utiliza para almacenar un solo caracter. Tiene la capacidad de almacenar cualquier caracter en formato Unicode (formato estándar de cualquier caracter en Java que ocupa 16 bits de almacenamiento). Se debe notar que sólo almacena un caracter, para almacenar una palabra o tan sólo más de un caracter, se debe utilizar otro tipo de datos que provee el lenguaje, el tipo String.

Otro punto importante es que cuando se asigna en el código un literal caracter, este debe ir rodeado de comillas simples, como se muestra a continuación:

```
char c = 'a';
```

La capacidad de almacenamiento de un tipo `char` es de dos bytes. La diferencia fundamental respecto de los tipos enteros es que éste se maneja sin el signo como valor numérico, lo cual incrementa su capacidad de almacenamiento. Esto es lógico ya que si bien es como los otros tipos enteros internamente, su finalidad es representar caracteres Unicode. La siguiente tabla especifica su capacidad.

Nombre del tipo	Longitud	Rango
char	16 bits	-2^{15} a $2^{15}-1$ ó 0 a 65,535

Se puede utilizar la siguiente notación en los literales de asignación:

Literal de Asignación	Descripción
'a'	La letra a
'\t'	Una tabulación
'\u????'	Un caracter Unicode específico con valor: ????

Ejemplo

- '\u03A6' Es la letra Griega Pi

Nota: Cuando se utiliza un literal de asignación con formato Unicode, el resultado final puede variar porque este puede tener asignado una página de códigos diferente a la propuesta en el ejemplo.

Literales de caracter

Existen caracteres que tienen un significado especial ya sea si se los utiliza en una variable char o como parte de una cadena. Estos caracteres están asociados a valores de control que se pueden utilizar en las salidas o representan información en los ingresos. Como se hará un uso extensivo de los mismos se enumeran en la siguiente tabla

Caracter	Significado
\n	Salto de línea
\t	Tabulador
\r	Retorno de carro

Para mostrar un ejemplo de utilización se usarán elementos todavía no explicados, las cadenas (String), la conversión de un tipo carácter a cadena y la concatenación de cadenas. Estos temas se aclararán posteriormente y se utilizan en este punto sólo a fin de demostrar el uso de estos caracteres especiales

Ejemplo

```
package caracteres;
public class CaracteresEspeciales {

    public static void main(String[] args) {
        char n= '\n';
        char t = '\t';
        char r = '\r';
        String cadena1 = "Hola Mundo\n" + n + "!!!!!!";
        String cadena2 = t + "\tHola Mundo";
        String cadena3 = "Hola Mundo\r" + r + "!!!!!!";

        System.out.println(cadena1);
        System.out.println(cadena2);
        System.out.println(cadena3);
    }
}
```



```
}  
Este programa produce la siguiente salida, la cual esta afectada por el uso de los caracteres  
literales especiales  
    Hola Mundo  
  
    !!!!!  
        Hola Mundo  
    Hola Mundo  
  
    !!!!!
```

Tipos Primitivos: String

No es exactamente un tipo de dato primitivo, es una clase de uso interno de Java, pero permite manejos de asignaciones como si lo fuera. Los literales que lo definen deben ir encerrados entre comillas dobles (" "), como se muestra a continuación:

```
"Este es un string"
```

Ejemplo

```
String saludo = "Hola !! \n";  
String MensajeDeAlerta = "Usar un número!";
```

Nota: Este tipo tiene una serie de detalles en los cuales se irá profundizando más adelante. La gestión y manipulación de String exige un apartado separado el cuál se irá presentando en los próximos módulos

Tipos Primitivos: Enteros

Todos los tipos enteros en Java se rigen estrictamente por la aritmética de complemento a dos (salvo el tipo `char`). Esto implica que son tipos con valores positivos y negativos y el bit más significativo en su formato interno indica el signo.

Existen cuatro tipos enteros y se diferencian en su capacidad de almacenamiento como muestra la tabla.

Nombre del tipo	Longitud	Rango
<code>byte</code>	8 bits	-2^7 a 2^7-1 ó -128 a 127
<code>short</code>	16 bits	-2^{15} a $2^{15}-1$ ó -32,768 a 32,767
<code>int</code>	32 bits	-2^{31} a $2^{31}-1$ ó -2,147,483,648 a 2,147,483,647
<code>long</code>	64 bits	-2^{63} a $2^{63}-1$ ó -9,223,372,036,854,775,808 a +9,223,372,036,854,775,807

Ejemplo

```
int a;  
int b=10;
```

El lenguaje por defecto convierte a todo literal de asignación a un tipo entero `int`, por lo tanto cuando se asigna un literal a otro tipo entero diferente se debe especificar con una letra o un molde que convierte el tipo (en inglés se denomina a dicha conversión “*cast*”), como se indica a continuación.

Ejemplo

```
long p = 10L;  
long n = (long)10;  
short s = (short)5;  
byte b = (byte)1;
```

Las formas utilizadas para p y n son análogas. La letra “L” utilizada en la asignación a la variable p es un camino corto que provee el lenguaje. En este caso en particular, como se asigna desde un espacio de almacenamiento menor a una mayor, el lenguaje provee una conversión automática llamada *promoción*.

Java 7

Inicialización de Número Grandes

A partir de la versión 7 se pueden inicializar variables con literales de asignación separados por símbolos de subrayado para facilitar su lectura.

De esta manera, declaraciones como las siguientes son perfectamente válidas

```
int var1 = 1_000_999_000;  
long var2 = 1_222_333_444_555_666_777L;
```

Si se muestra por pantalla el contenido de estas variables, se obtiene para la primera y segunda variable respectivamente

```
1000999000  
1222333444555666777
```

Java 7

Bases Numéricas en Java

Para comodidad del programador, Java permite la asignación de valores a números enteros en formato decimal, octal y hexadecimal. A partir de la versión 7 del lenguaje se permiten las asignaciones de números con formato binario, lo cual sumado al uso del carácter de subrayado brinda facilidad al momento de leer valores numéricos binarios, los cuales suelen ser largos. El siguientes es un ejemplo de utilización

```
public static void main(String[] args) {  
    int var1 = 153;  
    int var2 = 0x153;  
    int var3 = 0153;  
    int var4 = 0b1001_1001;  
  
    System.out.println("El valor de var1 asignado en decimal: " + var1);  
    System.out.println("El valor de var2 asignado en hexadecimal: " + var2);  
    System.out.println("El valor de var3 asignado en octal: " + var3);  
    System.out.println("El valor de var4 asignado en binario: " + var4);  
}
```

Este sencillo programa produce la siguiente salida en formato decimal, la cual varía según la base numérica utilizada en cada asignación:

```
El valor de var1 asignado en decimal: 153  
El valor de var2 asignado en hexadecimal: 339  
El valor de var3 asignado en octal: 107  
El valor de var4 asignado en binario: 153
```

Tipos Primitivos: Punto Flotante

Existen dos tipos primitivos de punto flotante y se diferencian por su capacidad de almacenamiento. Las variables de punto flotante en Java pueden utilizar notación científica, por eso no es posible asignar un rango de valores que pueden almacenar. La siguiente tabla muestra los tipos

Nombre del tipo	Longitud	Rango
Float	32 bits	1.40129846432481707E-45 a 3.40282346638528860E+38 (positivo o negativo)
Double	64 bits	4.94065645841246544E-324 a 1.79769313486231570E+308 (positivo o negativo)

Java toma la asignación de un literal numérico en punto flotante por defecto como un **double**, por lo tanto si la declaración es para un **float** hay que especificarlo de una de las siguientes formas:

```
float f = 10.5F;
float g = (float)10.5;
```

Los literales de punto flotante incluyen el punto decimal o una de las siguientes notaciones:

- E o e (agrega el exponencial)
- F o f (float)
- D o d (double)

Ejemplo

Literal de Asignación	Descripción
3.14	Valor simple de punto flotante (un double)
6.02E23	Valor grande de punto flotante
2.718F	Valor del tamaño de un float
123.4E+306D	Valor grande de punto flotante con una D redundante pero correcta

Valores Iniciales de las Variables

En java, todas las variables declaradas, cuando son creadas y almacenadas en memoria, tienen un valor inicial. La siguiente tabla enumera dichos valores

Variable	Valor
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	'\u0000'
boolean	false
Todos los tipos referenciados	null

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;  
    }  
}
```

Tipos referenciados

Cada vez que se crea una clase se define un nuevo tipo, pero para poder hacer uso de él, se debe crear un objeto. Esto es similar a declarar una variable de un tipo base con la diferencia que la memoria requerida para la operación será determinada por los elementos de dicha la clase.

Por otro lado, el lenguaje utiliza la definición de la clase como molde de la memoria a utilizar y en base a esto se realiza la creación del objeto. En Java cuando se declara un objeto se utiliza un operador diseñado para este fin: **new**. Es él quien toma la clase como molde y reserva la memoria para la creación del objeto.

Sin entrar en detalles acerca de la manera en que lo realiza, el operador efectúa las operaciones necesarias en memoria y luego devuelve el valor de la dirección, o referencia, del lugar en el cual las realizó, de ahí el nombre de estas declaraciones como tipo referenciados.

Evidentemente el valor devuelto debe almacenarse en algún lugar para su posterior uso y ese lugar es una variable de tipo referencia. Como este tipo de variables almacenan la dirección de memoria donde se creó el objeto y como toda clase genera un nuevo tipo, el lenguaje es consistente al especificar que la declaración deba efectuarse con la clase que define a dicha variable.

De esta manera, con el nombre de la clase seguido del identificador elegido, se crea la variable referencia.

Si se le desea dar una referencia inicial, se debe almacenar el valor devuelto por el operador **new**. La declaración tendría un formato tal que al identificador lo seguirá un operador de asignación "=", el operador **new** y por último en nombre de la clase que indica el tipo de objeto a crear.

El formato es el siguiente:

<nombre del tipo clase> <identificador> [= new nombre del tipo clase ()];

Al igual que las variables, se pueden crear tipos referenciados sin necesidad de inicializarlos con un valor (referencia al objeto creado por **new**), pero se debe usar el operador posteriormente para crear el objeto cuando se desee utilizarlo.

Ejemplo:

```
Empleado e = new Empleado();  
Persona p = new Persona();  
Persona p2;
```

Dónde se almacenan y que son los métodos

Casi todo código que se puede ejecutar en Java (salvo los bloques estáticos como se verá posteriormente) estará escrito dentro de un elemento especial del lenguaje al que se denomina método.

El código de los métodos se almacena en el segmento de texto (o se emula dicho almacenamiento por el intérprete), por lo tanto ahí se encuentran todas las instrucciones a ejecutar.

En Java sólo se pueden declarar métodos dentro de clases, cualquier intento de declarar un método fuera de una clase es un error que se detecta en tiempo de compilación.

Para acceder a un método con "notación de punto" debe estar declarado en su interfaz, por lo tanto deberá tener un modificador de tipo "**public**". Esto se explicará posteriormente.

Métodos en Java

Los métodos, o como se los llama en otros lenguajes "funciones", son los elementos de una clase mediante los cuales se define una operación que un objeto de su tipo puede realizar.

La declaración de un método puede adoptar el siguiente formato:

```
[<modificador>] <tipo retornado> <identificador>([lista de argumentos]){  
    [declaraciones y / o métodos a utilizar]  
    [return [valor retornado];]  
}
```

Si bien el formato de la declaración del método es incompleta (falta la posibilidad de declarar excepciones), se usará así de momento porque la parte faltante es un tema avanzado para este punto.

El formato quiere decir lo siguiente:

- **Modificador:** Opcional. Es el modificador de visibilidad del método (indica como se lo puede utilizar y desde donde). Las posibilidades son:

- **public** (pertenecer a la interfaz. Se accede con notación de punto)
- **private** (pertenecer a la clase. No se accede con notación de punto)
- **protected** (pertenecer a la cadena de herencia y se verá posteriormente. No se accede con notación de punto)
- Sin modificador (pertenecer al paquete. Se accede si está en el mismo paquete)

El formato quiere decir lo siguiente:

- **Tipo retornado:** se debe elegir entre un tipo primitivo o uno referenciado. Si el método no devuelve ningún valor se puede indicar poniendo en este lugar la palabra clave **void**.
- **Identificador:** es el nombre que se le asignará al método y mediante el cual se lo invocará
- **Lista de argumentos:** en este lugar se indican los parámetros que recibirá el método. Los argumentos se declaran igual que las declaraciones de variables con la diferencia que no se pone el “;” final. Si el método posee varios argumentos, se deben separar unos de otros con el operador de continuación de declaración (“,”). Si el método no recibe argumentos, se dejan sólo los paréntesis del mismo.
- **Comienzo del bloque de sentencias:** obligatoriamente poner la llave de apertura de bloque de sentencias
- **Sentencias:** opcionalmente agregar declaraciones e invocaciones a métodos. Existe la posibilidad de no poner nada a lo que se llama método de cuerpo vacío.
- **Valor retornado:** opcionalmente poner una sentencia **return** para terminar la ejecución del método. Si se indica que el método retorna un valor (recordar que puede ser **void**), este se debe poner a continuación
- **Fin del bloque de sentencias:** Poner obligatoriamente la llave de cierre

Ejemplo

```
public int calculaVolumen(int ancho, int largo, int alto) {  
    // bloque de sentencias  
    return 0;  
}
```

Este método es público, retorna un entero y recibe tres enteros como parámetros

Clases en Java

Tienen el siguiente formato:

```
<modificador> class <identificador> {  
    [declaraciones de variables y métodos]  
}
```

Donde este tipo de declaración se debe leer de la siguiente manera:

- 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

Ejemplo

```
public class Ejemplo {  
    private int var1;  
    private int var2;
```

```
public int getVar1() {  
    return var1;  
}
```

No se debe olvidar que cuando se declara una clase se crea un nuevo tipo, en este caso el tipo es **Ejemplo**. Las clases son sólo moldes para los objetos de su tipo que se creen. Se puede apreciar en ella la interfaz declarada con el modificador **public**. Esto indica que el método `getVar1` será accesible por notación de punto

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.

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;  
}
```

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

Constructores

Además, las clases tienen un método especial que se denomina constructor. Cuando se crea un objeto a través de crear una instancia de una clase, luego que se reserva en memoria los lugares de almacenamiento necesarios para el objeto, se ejecuta siempre el constructor.

Cómo es un método especial se diferencia de los otros por las siguientes dos características:

- Se llaman igual que la clase

- Nunca se le pone el valor retornado

Si la clase no posee uno, como en el ejemplo anterior, se utiliza uno por defecto que el lenguaje provee más allá que no se especifique en el código. Además, si se lo desea, se puede poner más de uno.

Los constructores permiten que se les pase parámetros, pero cuando se declara un objeto se le deben pasar tantos parámetros como los que figure en alguno de sus constructores o la declaración será un error, así si se le quiere poner la clase anterior un constructor que reciba un entero, quedaría con el siguiente formato:

```
public class Ejemplo {  
    public Ejemplo(int v){ var1=v;}  
    private int var1;  
    private int var2;  
    public int getVar1(){return var1;}  
}
```

Y la declaración de un objeto de esta clase sería

```
Ejemplo obj = new Ejemplo(8);
```

Un resumen de lo que ocurre cuando se declara un objeto de una clase que tiene constructor es el siguiente:

- Se reserva el espacio en memoria para el objeto
- Se le pasa el parámetro 8 al constructor
- Se empieza a ejecutar el constructor y se asigna el valor del parámetro a la variable var1
- Se termina la ejecución del constructor
- Se devuelve la referencia al objeto y se almacena en la variable obj

Objetos y Mensajes

Para utilizar los servicios que presta un objeto se invocan los elementos declarados en su interfaz pública. En la clase Ejemplo el único elemento en dicha interfaz es el método getVar1. Para poder invocarlo se debe utilizar la notación de punto y la referencia almacenada en la variable obj, ya que esta indica el lugar de almacenamiento del mismo. El código en un programa se vería de la siguiente manera:

```
int aux;  
aux = obj.getVar1();
```

En este código se declara la variable entera aux y luego se almacena en ella el valor retornado por el método getVar1.

Cada vez que se utiliza un elemento declarado en la interfaz (parte pública) se le manda un mensaje al objeto

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 dos formas de asignar una cadena de caracteres: cuando se crea el objeto y mediante el operador de asignación

Ejemplo:

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

Nota: los String en Java son inmutables, lo cual implica que si se asigna una nueva cadena de caracteres a una variable de referencia, como la del ejemplo llamada nombre, se crea un nuevo objeto y se deja al anterior para que sea recolectado como basura

Ocultamiento de la información

Para proteger los datos que se almacenan dentro de un objeto, la clase utiliza la declaración de visibilidad **private** para que no se pueda acceder la variable, ya que esta declaración indica que la variable no pertenece a la interfaz de la clase. Cuando un elemento no pertenece a la interfaz no se puede acceder por notación de punto una vez creado un objeto, sólo lo podrán acceder aquellos elementos que pertenezcan a la clase.

Ejemplo:

```
package ocultamiento;

public class Persona {
    private String primerNombre;
    private String segundoNombre;
    private String apellido;
    private String documento;

    public Persona() {
    }

    public String getApellido() {
        return apellido;
    }

    public String getDocumento() {
        return documento;
    }

    public String getPrimerNombre() {
        return primerNombre;
    }

    public String getSegundoNombre() {
        return segundoNombre;
    }

    public void setApellido(String string) {
        apellido = string;
    }

    public void setDocumento(String string) {
        documento = string;
    }

    public void setPrimerNombre(String string) {
        primerNombre = string;
    }

    public void setSegundoNombre(String string) {
        segundoNombre = string;
    }
}
```

}

Encapsulado

Como se mostró con anterioridad, en toda clase existe una parte pública y una privada. La primera define los elementos de la clase que son accesibles a través de su interfaz. Muchas veces se hace referencia a este hecho como “accesible por el mundo exterior o el universo”. Esta frase puede simplificarse de la siguiente manera:

Los elementos públicos de una clase serán accesibles por notación de punto una vez creado un objeto de su tipo

En cambio, la parte privada, define todo lo contrario.

Los elementos declarados como privados en una clase no serán accesibles por ningún elemento salvo que este se encuentre en la misma clase.

Este último concepto es el que permite separar los servicios que brinda un objeto de la forma en que lo hace. Por lo tanto, en otras palabras, un objeto debe verse como una serie de servicios que presta a través de su interfaz y la forma en que lo hace se oculta del mundo exterior porque no es accesible. Cuando una clase oculta una serie de operaciones (métodos y variables de la clase que éstos usan) declarándolos privados y los utiliza posteriormente para brindar un servicio, se dice que este servicio está encapsulado dentro de la clase.

Se puede decir que el ocultamiento de la información y los métodos privados son en conjunto conocidos como “encapsulado”, estén presente uno de ellos o ambos.

Un ejemplo claro de esto es cuando en una clase se oculta la información pero se quiere brindar la posibilidad de acceder a los datos almacenados, ya sea para guardar valores como para leerlos. En este caso se deben crear métodos públicos que cumplan ese rol, como se mostró en el ejemplo anterior.

¿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 está 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

```
package ocultamiento;

public class UsaPersona {
    public static void main(String[] args) {
        Persona p = new Persona();
    }
}
```

Se debe tener en cuenta que sólo puede existir un método `main` por clase.

Paquetes

Los paquetes son los lugares físicos donde se almacenan las clases. Tienen una relación con los directorios de un sistema de archivos en un sistema operativo, de manera que el nombre

de un paquete donde se guardan las clases es igual al nombre del directorio en el sistema operativo donde residen las mismas.

Se definen al principio de toda declaración de clase y debe ser la primera sentencia cuando se crea una clase.

Es el lugar donde se guarda la clase compilada (archivos con extensión .class), no los archivos fuentes (que tiene extensión .java). Sin embargo, algunos entornos (sobre todo los más antiguos) de trabajo guardan en el mismo directorio ambos archivos

Cuando se declara un paquete, se define un espacio de nombres, por lo tanto no es sólo un lugar de almacenamiento, sino que también es un espacio que define una visibilidad. Por lo tanto, si se quiere utilizar una clase que se encuentra en un paquete, se debe definir explícitamente.

Se declara con la sentencia **package**. Si el paquete no existe, el compilador lo crea (así como también el directorio asociado al nombre en el sistema de archivos del sistema operativo). Si el paquete existe, sólo incorpora la clase compilada a éste.

Para acceder a una clase dentro de un paquete se debe utilizar la sentencia **import**, porque como se mencionó anteriormente, los paquetes definen visibilidades, por lo tanto, se debe indicar donde “mirar” para utilizar la clase.

Construcción e Inicialización de Objetos

Los objetos, como se mencionó anteriormente, se construyen e inicializan para poder utilizarlos. Para comprender mejor este concepto, se analizará el siguiente ejemplo:

```
package inicializar;
/**
 * Esta clase esta diseñada para manejar horarios de ingreso y egreso de médicos
 * de una clínica
 */
public class Horario {

    private int dia = 1;
    private int horaComienzo = 9;
    private int minutosComienzo = 30;
    private int horaFin = 18;
    private int minutosFin = 30;
    private int turnosPorHora = 4;

    public Horario(int d, int hc, int mc, int hf, int mf, int tph) {
        dia = d;
        horaComienzo = hc;
        minutosComienzo = mc;
        horaFin = hf;
        minutosFin = mf;
        turnosPorHora = tph;
    }

    public Horario(Horario f) {
        dia = f.dia;
        horaComienzo = f.horaComienzo;
        minutosComienzo = f.minutosComienzo;
        horaFin = f.horaFin;
        minutosFin = f.minutosFin;
        turnosPorHora = f.turnosPorHora;
    }
}
```

```
}

public Horario agregar(int masDias) {
    Horario nuevaFecha = new Horario(this);
    nuevaFecha.dia = nuevaFecha.dia + masDias;
    return nuevaFecha;
}

public void imprimir() {
    System.out.println("Horario: ");
    System.out.println("Día: " + dia);
    System.out.println("Hora de comienzo: " + horaComienzo);
    System.out.println("Minutos de comienzo: " + minutosComienzo);
    System.out.println("Hora de fin: " + horaFin);
    System.out.println("Minutos de fin: " + minutosFin);
    System.out.println("Turnos por hora:" + turnosPorHora);
}

public int getDia() {
    return dia;
}

public int getHoraComienzo() {
    return horaComienzo;
}

public int getMinutosComienzo() {
    return minutosComienzo;
}

public int getHoraFin() {
    return horaFin;
}

public int getMinutosFin() {
    return minutosFin;
}

public int getTurnosPorHora() {
    return turnosPorHora;
}
}
```

Y una clase cliente de la anterior que utiliza uno de sus servicios
package inicializar;

```
public class VerificarHorarios {

    public static void main(String[] args) {
        Horario h = new Horario(2, 8, 45, 12, 45, 3);
        h.imprimir();
        System.out.println("-----");
        Horario nuevoHorario = h.agregar(3);
        nuevoHorario.imprimir();
    }
}
```

```
}
```

Una declaración del tipo:

```
Horario h = new Horario(2, 8, 45, 12, 45, 3);
```

Aloja espacio en memoria para un nuevo objeto de la siguiente manera:

1. Se reserva el espacio en memoria para el nuevo objeto
2. Todas las variables de instancia son inicializadas a sus valores por defecto
3. Los atributos que tienen asignados valores explícitamente son inicializados
4. Se ejecuta el constructor
5. Se le asigna el valor a la variable de referencia

Alojamiento en Memoria

Una declaración aloja espacio en memoria sólo para la variable de referencia. En el ejemplo anterior previa ejecución del operador **new** se aloja espacio solo para h. Por lo tanto, si la declaración se realizó en un método, la variable h se alojará en el stack. Si la declaración se hizo como una variable de instancia de la clase, h se alojará en el heap. En el ejemplo anterior como h se encuentra dentro de main, se alojará en el espacio dedicado para main en el stack.

Un ejemplo de cómo es el proceso de construcción de un objeto, más allá de la ubicación de la variable h (heap o stack) es el siguiente:

1. Aloja espacio en el stack sólo para la variable de referencia

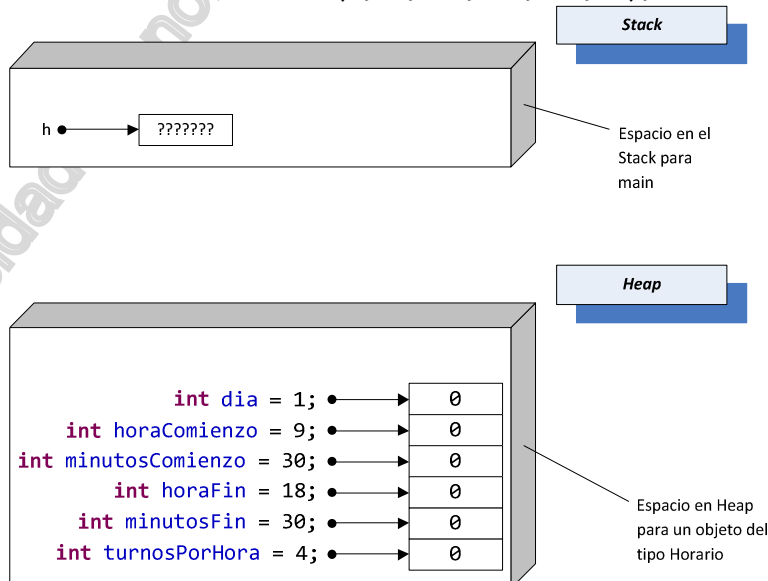
```
Horario h = new Horario(2, 8, 45, 12, 45, 3);
```

Cuando se resuelve la declaración de la referencia, en memoria sólo se tiene



2. Cuando se ejecuta el operador **new**, lo primero que se hace es alojar espacio para el objeto en el heap.

```
Horario h = new Horario(2, 8, 45, 12, 45, 3);
```



3. El próximo paso en la construcción del objeto, es la inicialización de los atributos. Como la declaración que crea el objeto indica que se le pasan valores al constructor, éste intentará asignarlos según un orden de inicialización preestablecido. El lugar mediante el cual se le pasan los valores al constructor es entre los paréntesis que están a continuación del nombre de la clase a la derecha de la declaración:

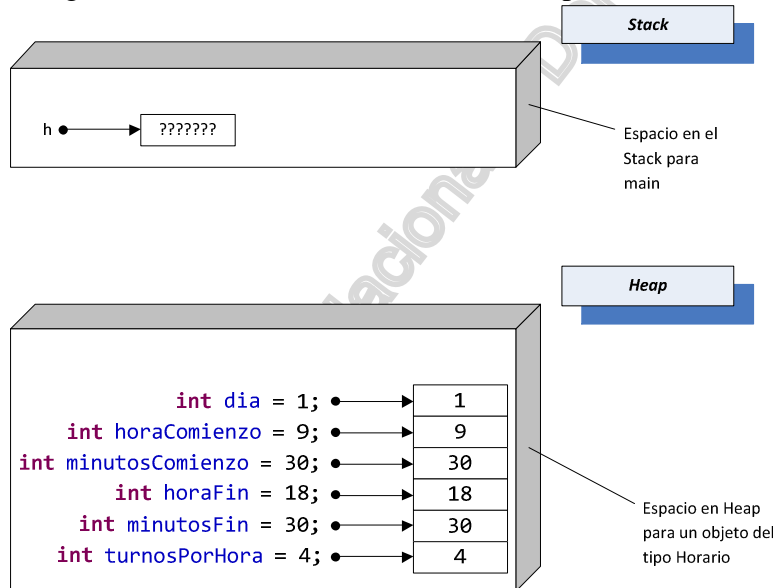
```
Horario h = new Horario(2, 8, 45, 12, 45, 3);
```

4. Cómo la clase declara variables de instancia con valores iniciales, el próximo paso es inicializar los valores declarados. Estos fueron declarados en la clase de la siguiente manera:

```
private int dia = 1;
private int horaComienzo = 9;
private int minutosComienzo = 30;
private int horaFin = 18;
private int minutosFin = 30;
private int turnosPorHora = 4;
```

Por lo tanto, se asignan los valores de inicialización declarados en la clase para los atributos (llamados valores por defecto)

El siguiente gráfico muestra la realización de estos pasos



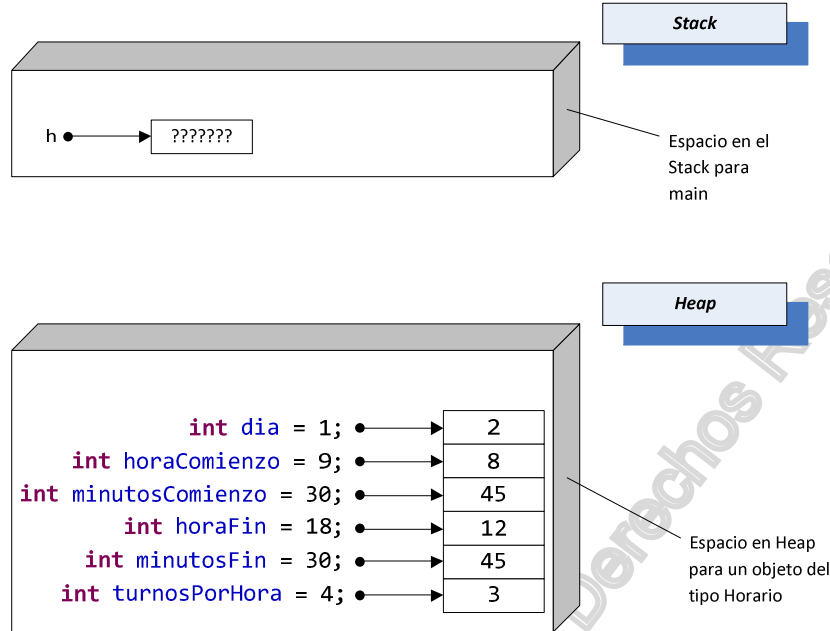
5. Una vez inicializados los valores de las variables de instancia, se ejecuta el conjunto de sentencias que se encuentran dentro del constructor que corresponda, por ejemplo:

```
Horario h = new Horario(2, 8, 45, 12, 45, 3);
```

Deberá pasar los parámetros 2,8,45,12,45,3 y empezar a ejecutar lo que está declarado dentro del constructor como se muestra a continuación

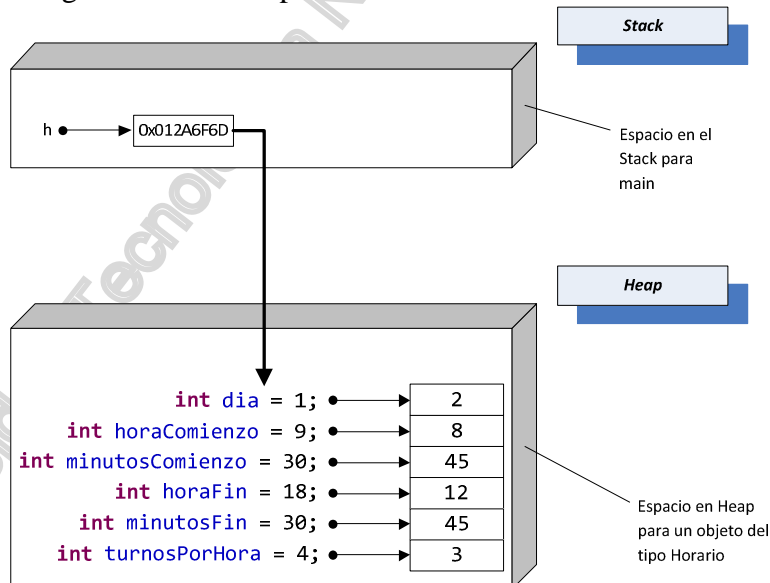
```
public Horario(int d, int hc, int mc, int hf, int mf, int tph) {
    dia = d;
    horaComienzo = hc;
    minutosComienzo = mc;
    horaFin = hf;
    minutosFin = mf;
    turnosPorHora = tph;
}
```


El siguiente gráfico muestra el estado de las variables de instancia una vez terminada la ejecución de sentencias del constructor de la clase



6. Cuando se termina de ejecutar el constructor, el objeto se encuentra creado en el heap. Lo único que queda por hacer es retornar la referencia que permite luego encontrarlo cuando se opera con él. Por lo tanto, se asigna el objeto que acaba de ser creado a la variable de referencia `h`
- ```
Horario h = new Horario(2, 8, 45, 12, 45, 3);
```

El siguiente gráfico ilustra el proceso



## Asignación de tipos referenciados

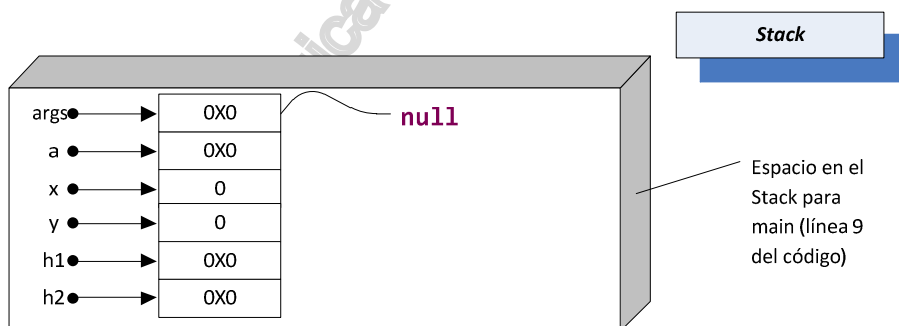
Como se demostró anteriormente, un objeto se crea mediante el operador **new** y su referencia se almacena en la variable declarada del tipo de la clase. Esta variable es tan sólo una referencia, **no el objeto en sí mismo**, el cual se encuentra almacenado en el heap.

Si, por ejemplo, se tiene el siguiente código:

```
1. public class AsignacionReferencias {
2. public void metodo(Horario h){
3. int x = 7;
4. int y = x;
5. Horario h1 = new Horario(2, 8, 45, 12, 45, 3);
6. Horario h2 = h1;
7. h2 = new Horario(2, 8, 45, 12, 45, 3);
8. }
9. public static void main(String[] args) {
10. AsignacionReferencias a = new AsignacionReferencias();
11. int x = 7;
12. int y = x;
13. Horario h1 = new Horario(2, 8, 45, 12, 45, 3);
14. Horario h2 = h1;
15. h2 = new Horario(2, 8, 45, 12, 45, 3);
16. a.metodo(h2);
17. }
18. }
```

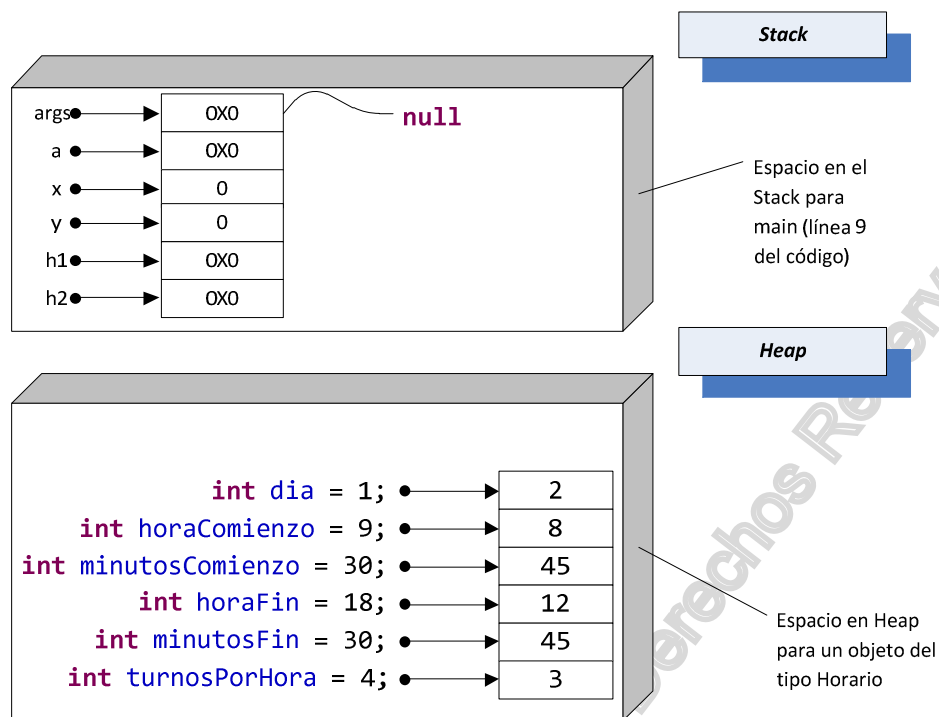
Se analizará el programa por cada línea de ejecución para comprender el manejo de memoria que se realiza a medida que se ejecuta

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

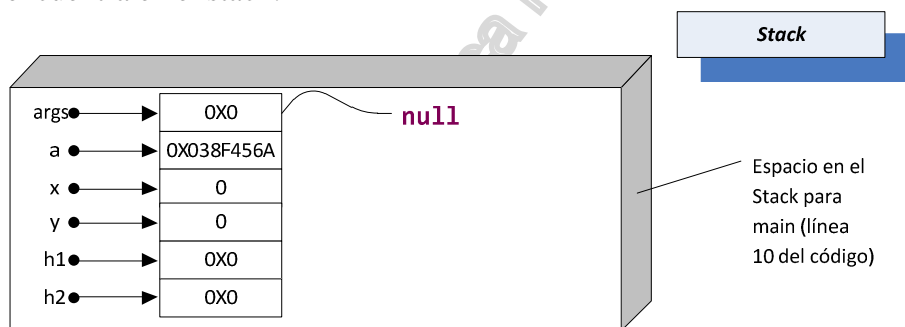


Cuando comienza el programa se reserva espacio en el stack para almacenar las variables locales a **main**. Como no recibe ningún argumento por línea de comandos en la variable args (esto se explicará posteriormente) el valor almacenado que queda es 0 en hexadecimal (**null**)

```
10. AsignacionReferencias a = new AsignacionReferencias();
```



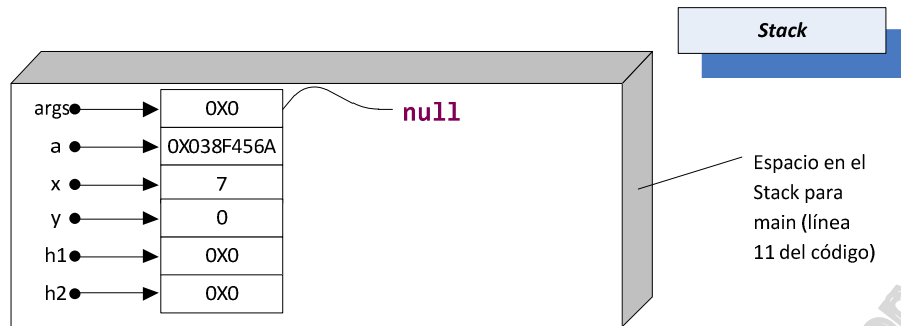
Se ejecuta el operador **new** el cual reserva y asigna valores de memoria en el heap según se explicó anteriormente. Como en este caso en particular la clase AsignacionReferencias no tiene variables de instancia, se asigna un lugar en el heap que representa el espacio de memoria asignado al objeto, pero no ocurre ningún proceso de reserva de memoria, inicialización o asignación de valores. El valor de la dirección de memoria, sin embargo, igualmente lo retorna el operador **new** y se asigna a la variable de referencia **a** que se encuentra en el stack.



**Nota:** Los valores hexadecimal que se presentan como representaciones de espacios de memoria son a fines didácticos y se deben considerar como suposiciones. El algoritmo interno que se dedica a establecer las direcciones de asignación en la máquina virtual es interno y se llama gestor de heap (heap manager). Éste es quien determina la dirección de memoria donde se almacenará el objeto.

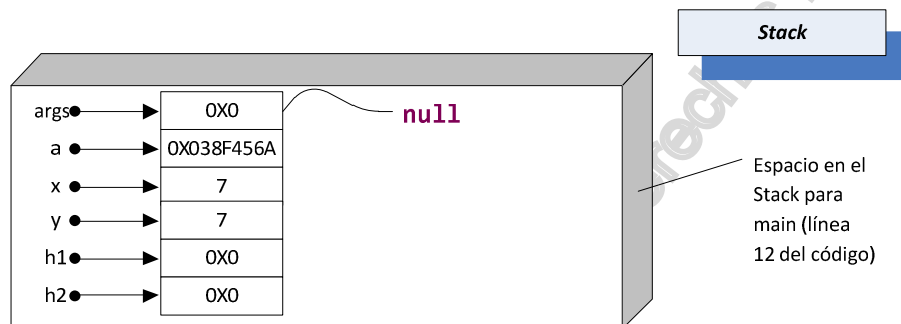
11. **int** x = 7;

Se inicializa la variable x en el stack asignándole el valor 7

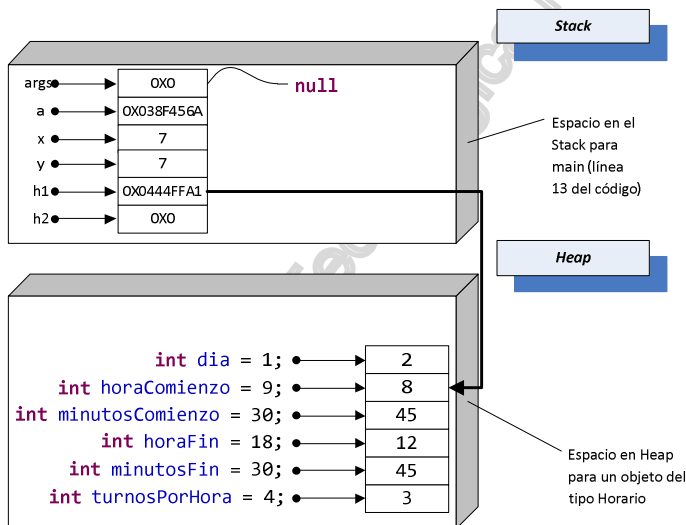


12. `int y = x;`

Se inicializa la variable y en el stack asignándole el valor que almacena la variable x



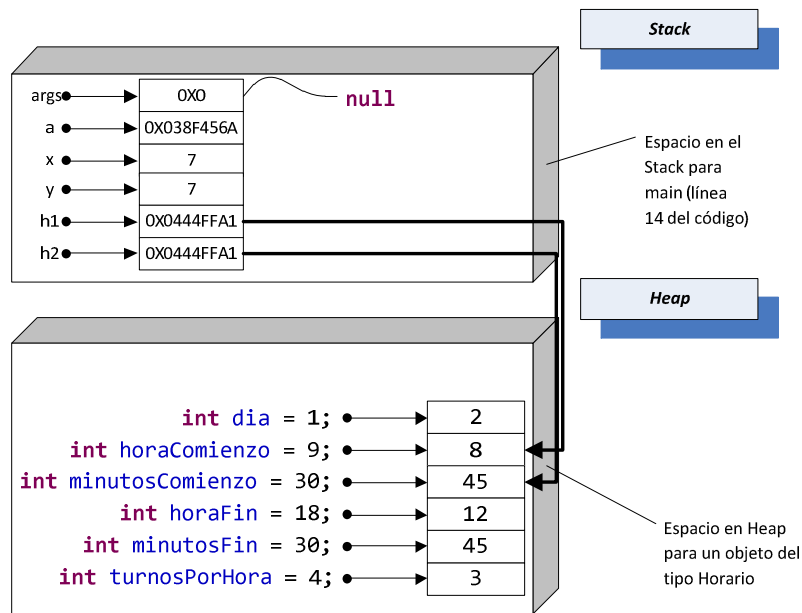
13. `Horario h1 = new Horario(2, 8, 45, 12, 45, 3);`



Se ejecuta el operador `new` el cual reserva y asigna valores de memoria en el heap según se explicó anteriormente. Como en este caso en particular la clase `Horario` tiene variables de instancia, se asigna un lugar en el heap que representa el espacio de memoria asignado al objeto, y ocurre el proceso de reserva de memoria, inicialización o asignación de valores. El valor de la dirección de memoria lo retorna el operador `new` y se asigna a la variable de referencia `h` que se

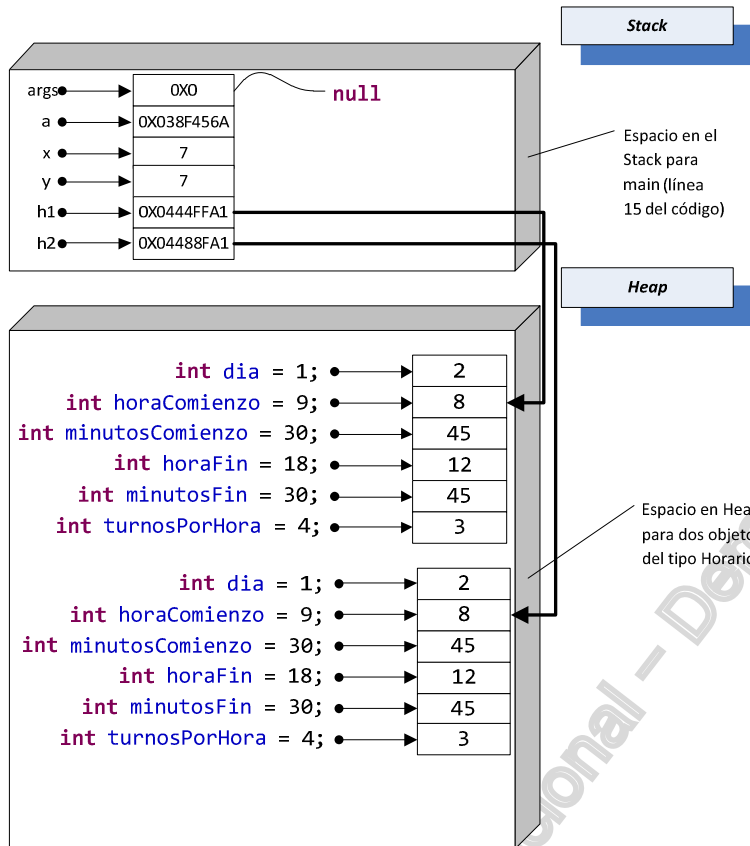
encuentra en el stack.

14. `Horario h2 = h1;`



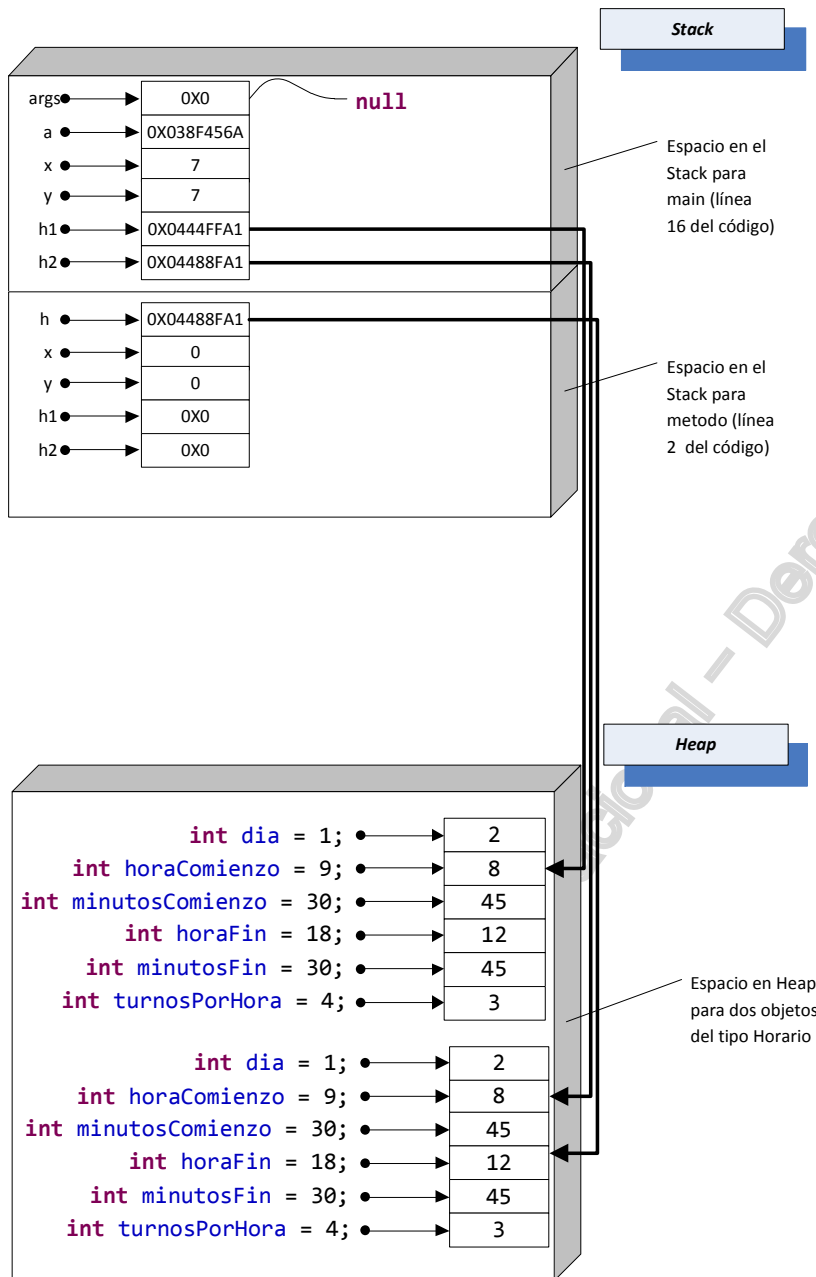
El resultado de la ejecución de esta línea es que se asignan dos referencias al mismo objeto del tipo `Horario`, por lo tanto dos variables de referencia a objetos del tipo `Horario` apuntan al mismo lugar de memoria en el heap. Por lo tanto, es muy importante no confundir el concepto de “variable de referencia” con “objeto”. La variable de referencia tan sólo indica el lugar donde se almacena el objeto.

15. `h2 = new Horario(2, 8, 45, 12, 45, 3);`



Para tener dos objetos diferentes con los mismos atributos debe ejecutarse `new` dos veces con los mismos valores como parámetros para el constructor. Sino, tendrán cada uno sus valores iniciales

16.                    a.metodo(h2);



Se invoca al método del objeto **a** que se define en la línea 2. Esto causa que se reserve espacio en el stack para el nuevo método a ejecutar y se almacene en dicho espacio los parámetros y variables locales del método.

2. **public void** metodo(Horario h){

Al empezar la ejecución del método, se encuentra reservado el espacio en el stack y las variables locales se inicializan a sus valores por defecto. El parámetro recibe un valor con la dirección de un objeto de tipo Horario (el que se encuentra almacenado en la variable **h2**) y comienza la ejecución con dicho valor asignado. En este punto hay dos referencias apuntando a un mismo objeto.

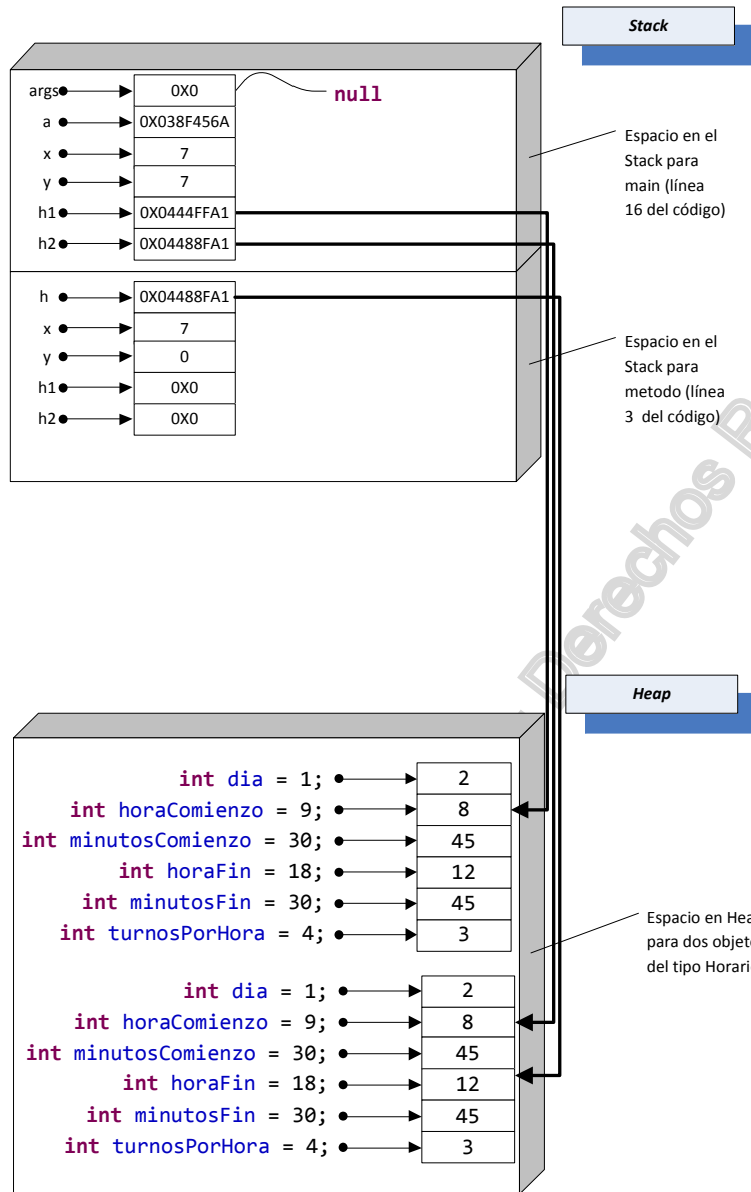
Notar que existen dos referencias a un mismo objeto. Una se encuentra almacenada en el espacio reservado en el stack para **main** y la otra se

encuentra en el espacio reservado para **método**.

3.                    **int** x = 7;

Se asigna en el espacio del stack el valor 7 a la variable **x**. Notar que no sólo esta variable no tiene nada que ver con la variable definida en **main** sino que se encuentra en un espacio de memoria totalmente diferente



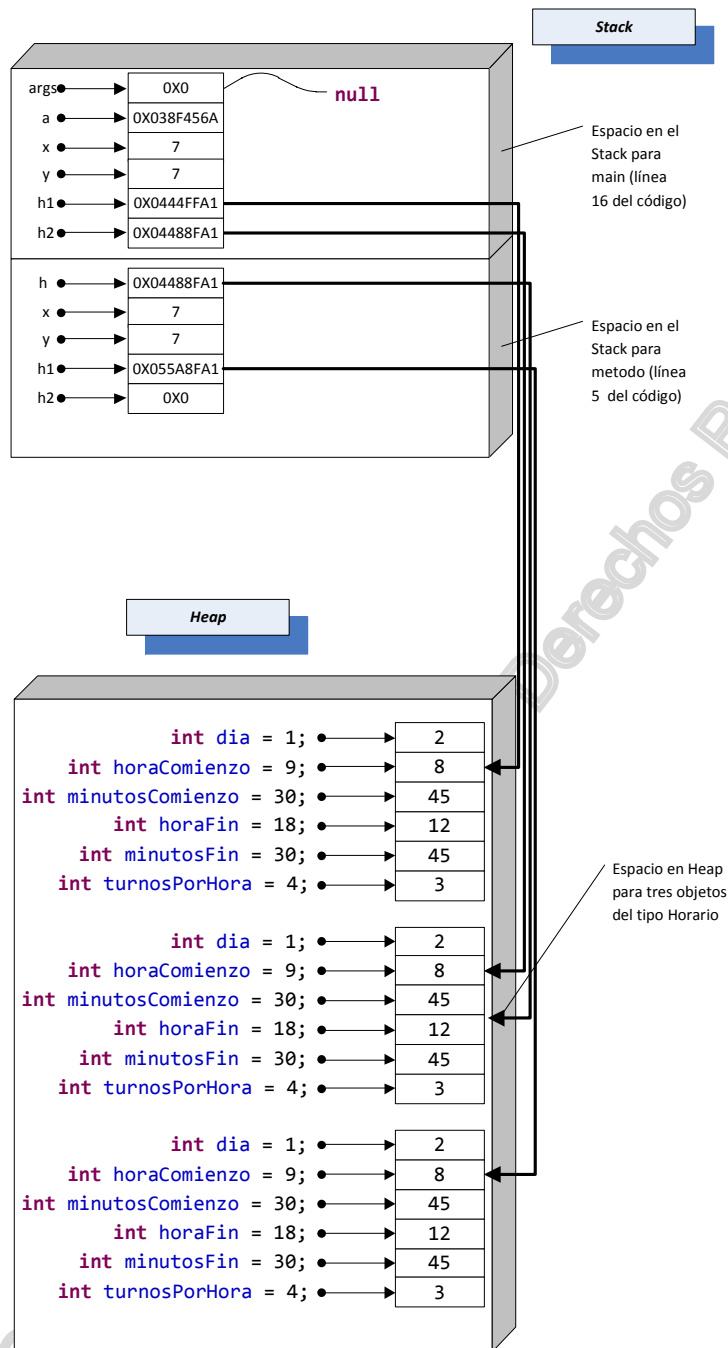


4. `int y = x;`

Al igual que en **main**, se le asigna el valor que almacena **x** a la variable **y**. Notar nuevamente los distintos lugares de memoria ocupados.

5. `Horario h1 = new Horario(2, 8, 45, 12, 45, 3);`

Se asigna a la variable de referencia **h1** la dirección de un nuevo objeto del tipo **Horario** que se encuentra en el heap. En este punto el stack y el heap quedan en el estado que muestra el gráfico.

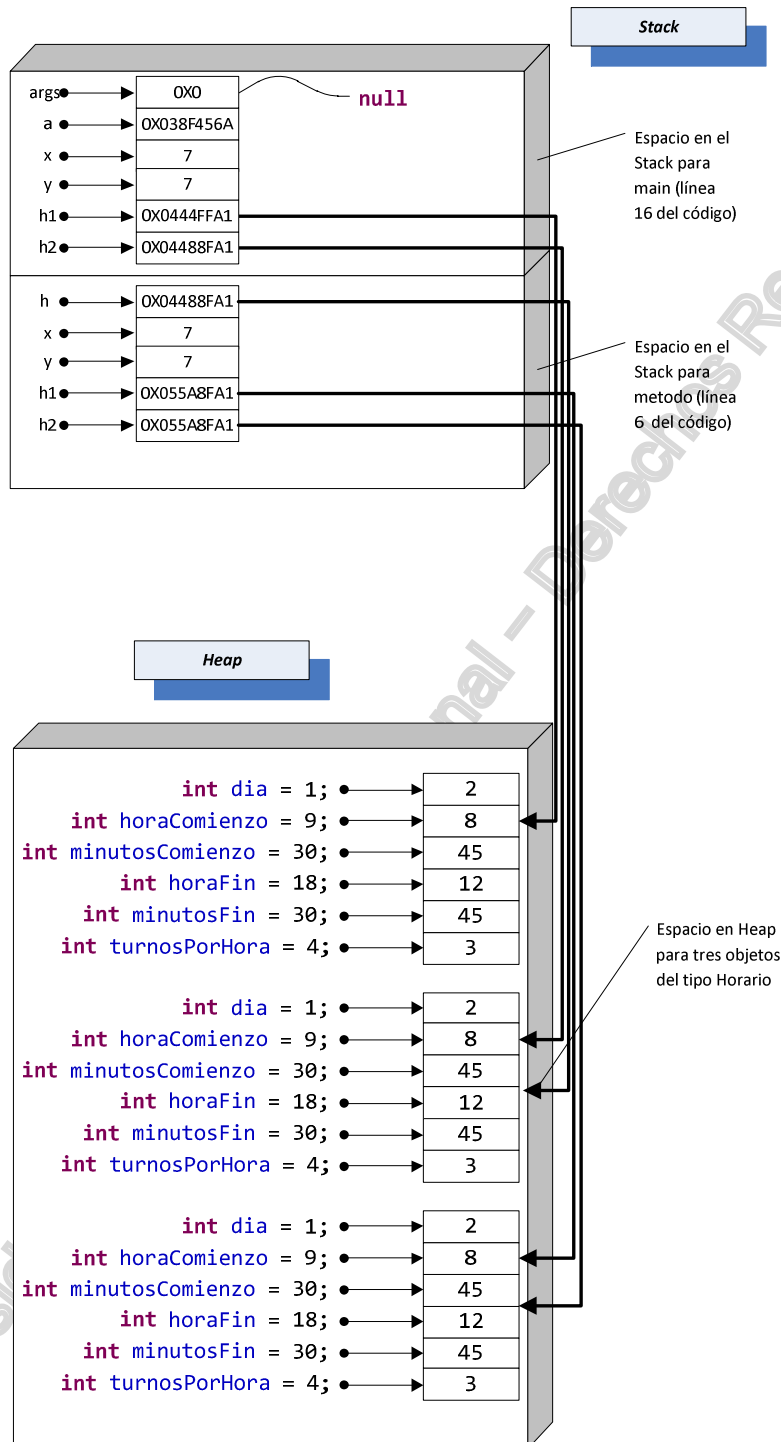


6. Horario h2 = h1;

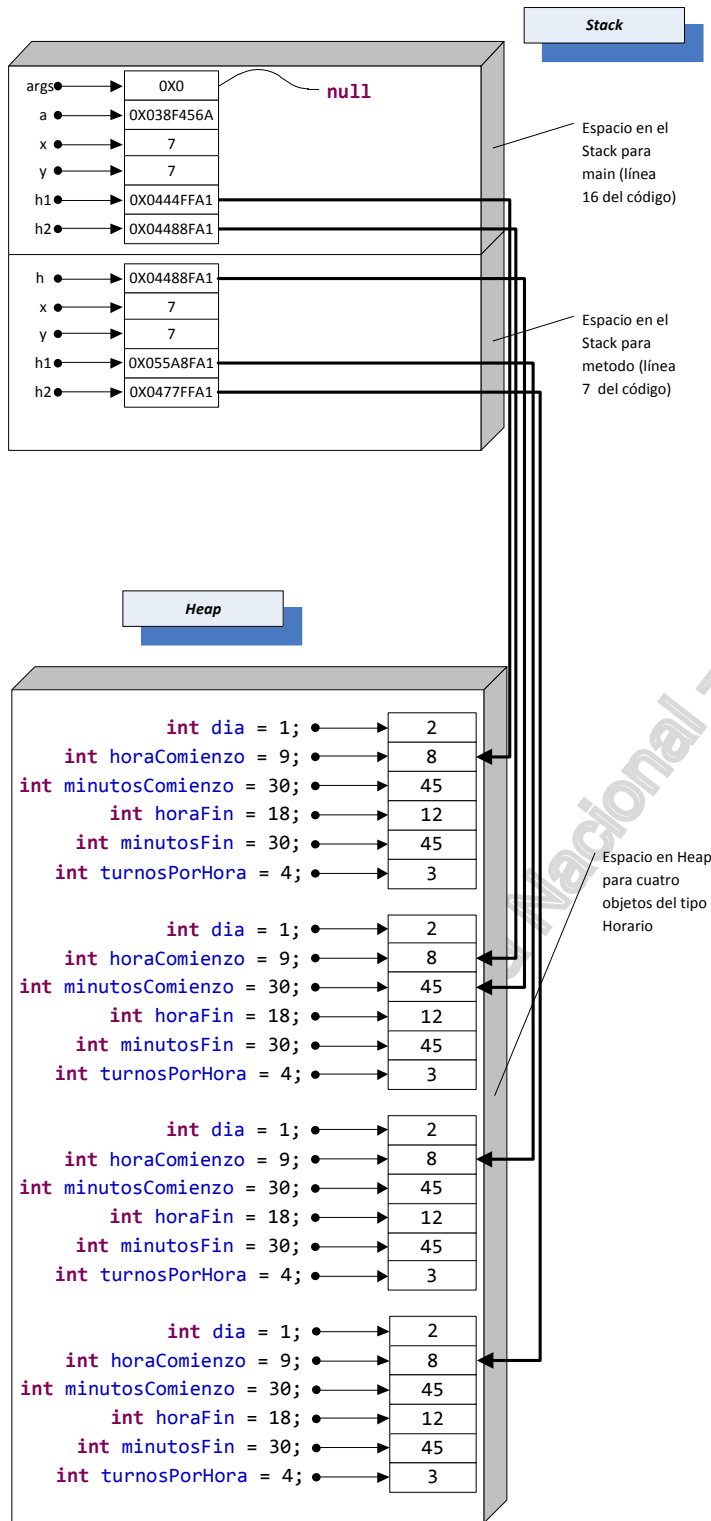
Nuevamente hay dos variables de referencia apunta a un mismo objeto. La diferencia es que estas se encuentran en espacio del stack reservado para **método**, por lo tanto no tienen nada que ver con las que se encuentran en el espacio reservado para **main**.

Recapitulando, la variable **h2** que se encuentra en el espacio del stack para **main** y el parámetro que recibe **método** apuntan a un mismo objeto en el heap. Las variables **h1** y **h2** que pertenecen a **método** apuntan también a un mismo objeto diferente al que apunta el

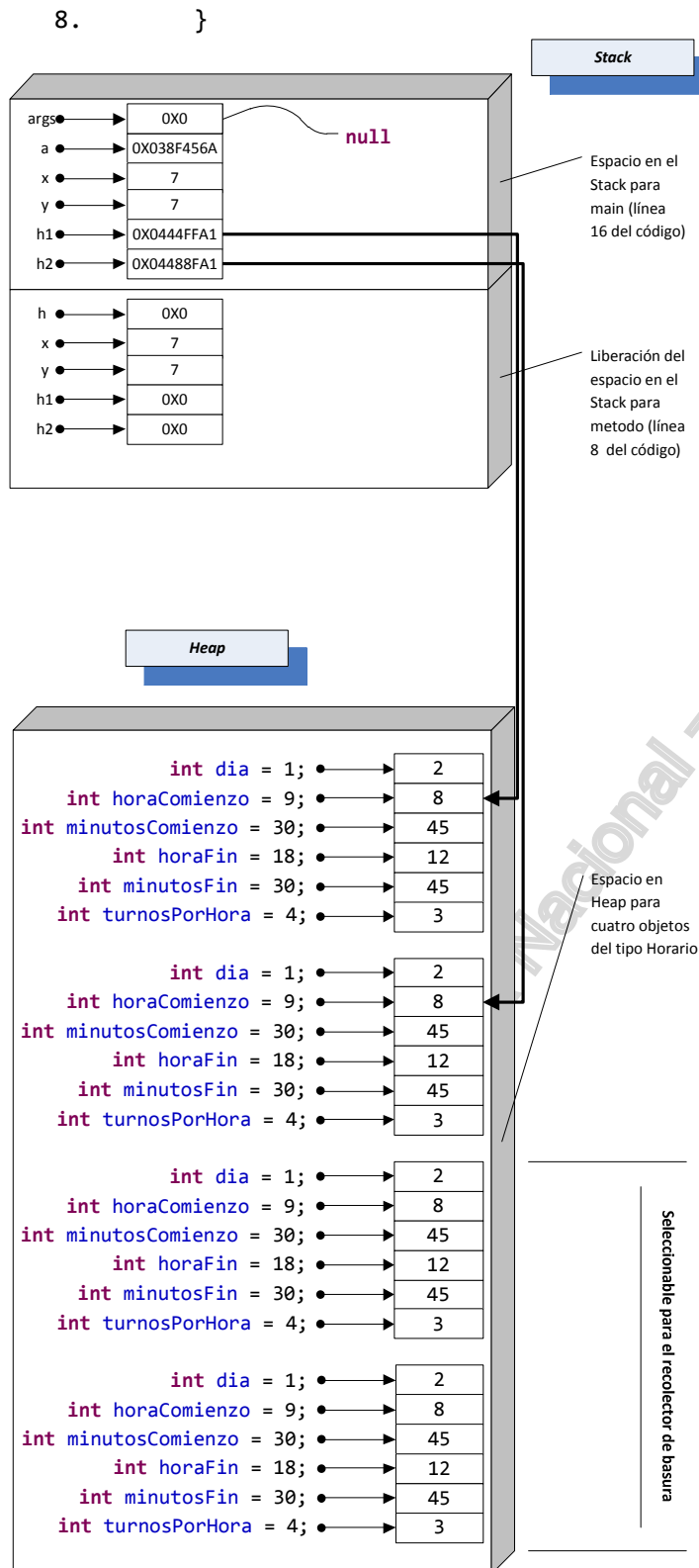
parámetro y la variable de **main**, pero todos son objetos del tipo *Horario* (mismo tipo, objetos diferentes).



7. `h2 = new Horario(2, 8, 45, 12, 45, 3);`



Una vez más se crea un nuevo objeto en el heap, esta vez para asignarlo a la variable de referencia **h2** de **metodo**. En este punto existen cuatro objetos en el heap y cinco referencias activas en el stack.



Esta línea indica el fin del método, lo cual acarrea como consecuencia la liberación de los recursos que estaban siendo utilizados por el método invocado. Las consecuencias son las siguientes:

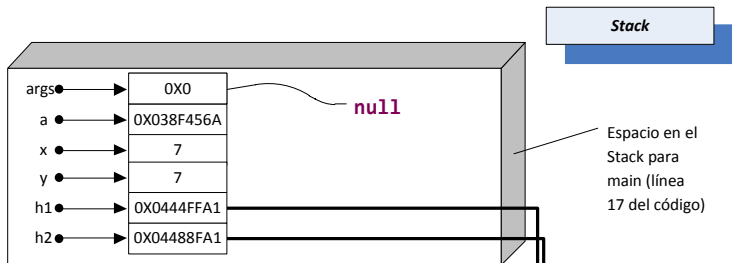
- Toda referencia al heap se libera, lo cual implica que si un objeto que se encuentra en el heap **no tiene una referencia activa que apunte hacia él en algún lugar del código que se sigue ejecutando en el programa, el mismo es elegible para que lo libere el recolector de basura (garbage collector)**. Los objetos siguen estando en el heap, pero no son accesibles y el recolector liberará el espacio que ocupan en un momento en el cual el algoritmo que lo gestiona considere oportuno (la máquina virtual posee un criterio para liberar objetos a medida que transcurre el tiempo. Cuanto más tiempo estuvo en memoria sin referencia activa, más rápido lo elimina).

- Se libera el espacio asignado al método en el stack, con lo cual dicho espacio se puede asignar a cualquier nuevo método que lo requiera.

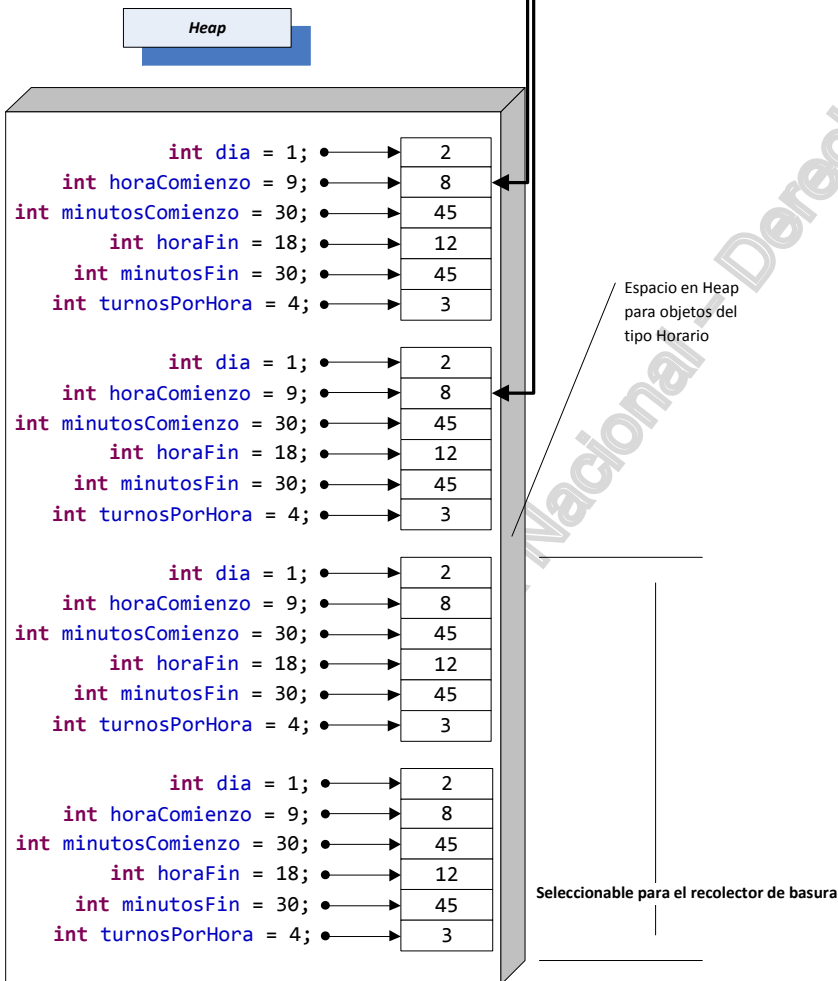
- El control del código retorna a la siguiente instrucción posterior al lugar donde se realizó la invocación del método

17.        }

Justo antes de ejecutar esta línea que indica el fin de **main**, y por lo tanto del programa, la



memoria queda en el estado que muestra el gráfico. Al ejecutar esta línea, se liberan todos los recursos que aún están tomados, tanto en el heap como el stack, y la memoria queda “limpia”.



## Pasajes de parámetros

En Java los pasajes de parámetros *siempre* se realizan por valor. Esto

quiere decir que cuando se invoca a un método, el parámetro se copia en el espacio del stack que corresponde a dicho método.

Sin embargo, se debe prestar especial atención cuando el valor pasado como parámetro es una referencia, ya que esta apunta al heap y si se modifican valores estos se reflejarán en el objeto al cual referencia la variable.

El siguiente ejemplo demuestra como varían los valores según se utilizan, se rescriben o se pasan a otros lugares de memoria.

Ejemplo

```
package inicializar;

public class PasajesDeParametros {

 // Métodos para cambiar los valores actuales
 public void cambiarEntero(int valor) {
 valor = 55;
 }

 public void cambiarRefObjeto(Horario ref) {
 ref = new Horario(1, 7, 30, 11, 00, 4);
 }

 public void cambiarAtributoObjeto(Horario ref) {
 ref.agregarDias(4);
 }

 public static void main(String args[]) {
 PasajesDeParametros p = new PasajesDeParametros();
 Horario h;
 int val;
 // Asignarle un valor al entero
 val = 11;
 // Intento de cambiarlo
 p.cambiarEntero(val);
 // ¿Cuál es el valor actual?
 System.out.println("El valor del entero es: " + val);
 System.out.println("-----");

 // Asignar un objeto del tipo Horario
 h = new Horario(2, 8, 45, 12, 45, 3);
 // Intento por cambiarlo
 p.cambiarRefObjeto(h);
 // ¿Cuál es el valor actual?
 h.imprimir();
 System.out.println("-----");

 // Cambiando el atributo day
 // a través de la referencia al objeto
 p.cambiarAtributoObjeto(h);
 // ¿Cuál es el valor actual?
 h.imprimir();
 }
}
```

La salida producida es la siguiente:

El valor del entero es: 11

-----

Horario:

Día: 2

Hora de comienzo: 8

Minutos de comienzo: 45

Hora de fin: 12

Minutos de fin: 45



Turnos por hora:3

-----

Horario:

Día: 2

Hora de comienzo: 8

Minutos de comienzo: 45

Hora de fin: 12

Minutos de fin: 45

Turnos por hora:3

### Palabras clave

Las palabras clave en java son las que se muestran a continuación:

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

Sin embargo, algunas de las palabras clave son sólo reservadas y no las utiliza el lenguaje, como por ejemplo:

- **const**
- **goto**

También, **true** y **false** para los tipos **boolean**, no son sentencias sino por el contrario, son literales constantes que se utilizan como palabras reservadas.