	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	52/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 04: Variables y arreglos




Elaborado por:

M.C. M. Angélica Nakayama C.
Ing. Jorge A. Solano Gálvez

Autorizado por:

M.C. Alejandro Velázquez Mena

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	53/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 04: Variables y arreglos

Objetivo:

Utilizar variables y arreglos con tipos de datos primitivos y con bibliotecas propias del lenguaje.

Actividades:

- Crear variables con diferentes tipos de datos.
- Crear arreglos con diferentes tipos de datos.


Introducción

Los **tipos de datos** hacen referencia al tipo de información que se trabaja, donde la unidad mínima de almacenamiento es el dato, también se puede considerar como el rango de valores que puede tomar una variable durante la ejecución del programa. Un tipo de datos define un conjunto de valores y las operaciones sobre estos valores.

Casi todos los lenguajes de programación explícitamente incluyen la notación del tipo de datos, aunque lenguajes diferentes pueden usar terminologías diferentes. La mayor parte de los lenguajes de programación permiten al programador definir tipos de datos adicionales, normalmente combinando múltiples elementos de otros tipos y definiendo las operaciones del nuevo tipo de dato.

Los **valores** que pueden adquirir los **tipos de datos** se manipulan durante la ejecución de un programa a través de **variables** o **arreglos**.

NOTA: En esta guía se tomará como caso de estudio el lenguaje de programación JAVA, sin embargo, queda a criterio del profesor el uso de éste u otro lenguaje orientado a objetos.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	54/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Variable

Una **variable** es un **nombre** que contiene un valor que **puede cambiar** a lo largo del programa. De acuerdo con el tipo de información que contienen, en Java hay dos tipos principales de variables:

1. Variables de tipos primitivos.
2. Variables de referencia.

Desde el punto de vista del papel o misión en el programa, las variables pueden ser:

- Variables **miembro** de una clase: Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias.
- Variables **locales**: Se definen dentro de un método o, de forma más general dentro de cualquier bloque entre llaves { }. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque.

Una variable se define especificando el **tipo de dato** y el **nombre** de dicha variable. Las variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase perteneciente al API de Java o creada por el usuario.

tipoDeDato *nombreVariable*;

Ejemplos:

int *miVariable*;


float *area*;

char *letra*;

String *cadena*;

MiClase *prueba*;

Si no se especifica un valor en su declaración, las variables miembro primitivas se inicializan a **cero** (salvo boolean y char, que se inicializan a false y '\0', respectivamente). Así mismo,

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	55/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

las variables miembro de tipo referencia son inicializadas por defecto a un valor especial: **null**.

Es importante distinguir entre la referencia a un objeto y el objeto mismo. Una referencia es una variable que indica dónde está guardado un objeto en la memoria (a diferencia de C/C++, Java no permite acceder al valor de la dirección, pues en este lenguaje se han eliminado los apuntadores). Al declarar una referencia todavía no se encuentra “apuntando” a ningún objeto en particular (salvo que se cree explícitamente un nuevo objeto en la declaración), y por eso se le asigna el valor **null**.

Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el objeto utilizando el operador **new**. Este operador reserva espacio en la memoria para el objeto (variables y funciones).

También es posible igualar la referencia declarada a otra referencia a un objeto existente previamente.

Ejemplo:


```
MyClass unaRef;
unaRef = new MyClass();
MyClass segundaRef = unaRef;
```

Un tipo particular de referencias son los **arrays** o **arreglos**, sean éstos de variables primitivas (por ejemplo, de enteros) o de objetos. En la declaración de una referencia de tipo **array** hay que incluir los **corchetes** [].

Ejemplo:

```
int [ ] vector;
vector = new int [10];
MyClass [ ] lista = new MyClass [5];
```

En Java todas las variables deben estar incluidas en alguna clase. En general las variables declaradas dentro de llaves { }, es decir, dentro de un bloque, son visibles y existen dentro de estas llaves. Por ejemplo las variables declaradas al principio de un método existen mientras se ejecute el método; las variables declaradas dentro de un bloque if no serán válidas al finalizar las sentencias correspondientes a dicho if y las variables miembro de una

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	56/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

clase (es decir declaradas entre las llaves { } de la clase pero fuera de cualquier método) son válidas mientras existe el objeto de la clase.

Constante

Una **constante** es una variable cuyo valor **no puede ser modificado**. Para definir una constante en Java se utiliza la palabra reservada **final**, antes de la declaración del tipo de dato, de la siguiente manera:

```
final tipoDato nombreDeConstante = valor;
```

Ejemplo:

```
final double PI = 3.1416;
```

Arreglo

Un **arreglo** es un objeto en el que se puede almacenar un conjunto de datos de un **mismo tipo**. Cada uno de los elementos del arreglo tiene asignado un **índice** numérico según su posición, siendo 0 el índice del primer elemento. Se declara de la siguiente manera:

```
tipoDeDato [ ] nombreVariable;      o      tipoDeDato nombreVariable[ ];
```


Como se puede apreciar, los corchetes pueden estar situados delante del nombre de la variable o detrás. Ejemplos:

```
int [ ] k;      String [ ] p;      char datos[ ];
```

Los arreglos pueden declararse en los mismos lugares que las variables estándar. Para asignar un tamaño al arreglo se utiliza la expresión:

```
variableArreglo = new tipoDeDato[tamaño];
```

También se puede asignar el tamaño al arreglo en la misma línea de declaración de la variable.

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	57/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
int [ ] k = new int[5];
```

Cuando un arreglo se dimensiona, todos sus elementos son inicializados explícitamente al **valor por defecto** del tipo correspondiente.

Para declarar, dimensionar e inicializar un arreglo en una misma sentencia se indican los valores del arreglo entre llaves y separados por comas. Ejemplo:

```
int [ ] nums = {10, 20, 30, 40};
```

El acceso a los elementos de un arreglo se realiza utilizando la expresión:

variableArreglo[índice]

Donde índice representa la posición a la que se quiere tener acceso y cuyo valor debe estar entre **0** y **tamaño - 1**

Todos los objetos arreglo exponen un atributo publico llamado **length** que permite conocer el tamaño al que ha sido dimensionado un arreglo.


Ejemplo:

```
int [ ] nums = new int[10];
for(int i=0; i < nums.length; i++)
    nums[i] = i * 2;
```

Los arreglos al igual que las variables, se pueden usar como argumentos, así como también pueden ser devueltos por un método o función.

En Java se puede utilizar una variante del ciclo for llamado **for-each**, para facilitar el recorrido de arreglos y colecciones, recuperando su contenido y eliminando la necesidad de usar una variable de control que sirva de índice. Su sintaxis es:

```
for(tipoDato variable: variableArreglo){
    //instrucciones
}
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	58/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo:

```
int [ ] nums = { 4, 6, 30, 15 };
for(int n : nums){
    System.out.println(n);
}
```

En este caso, sin acceder de forma explícita a las posiciones del arreglo, cada una de estas es copiada automáticamente a la variable auxiliar **n** al principio de cada iteración.

Los arreglos en Java también pueden tener más de una dimensión, al igual que en C/C++ para declarar un arreglo bidimensional se tendrían que usar dos pares de corchetes y en general para cada dimensión se usa un nuevo par de corchetes.

Ejemplo:

```
int [ ] [ ] matriz;
```


Argumentos por línea de comandos

Es posible suministrar parámetros al método **main** a través de la línea de comandos. Para ello, los valores a pasar deberán especificarse a continuación del nombre de la clase separados por un espacio:

```
>> java NombreClase arg1 arg2 arg3
```

Los datos llegarán al método **main** en forma de un arreglo de cadenas de caracteres.

Ejemplo:

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	59/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


```

public class Ejemplo {

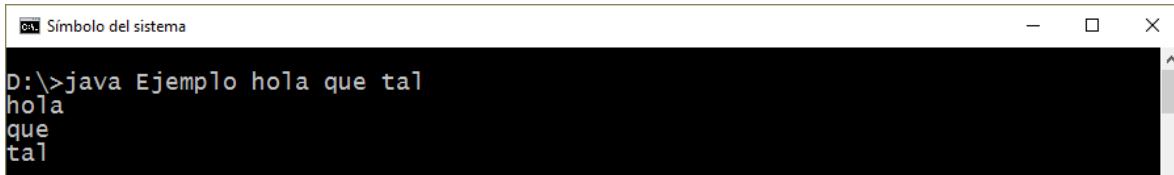
    public static void main(String[] args) {
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
    }

}

```


	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	60/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Se ejecuta utilizando la siguiente expresión en la línea de comandos:



```
D:\>java Ejemplo hola que tal
hola
que
tal
```


API de JAVA

La API Java (**A**pplication **P**rogramming **I**nterface) es una interfaz de programación de aplicaciones provista por los creadores del lenguaje, que da a los programadores los medios para desarrollar aplicaciones Java.

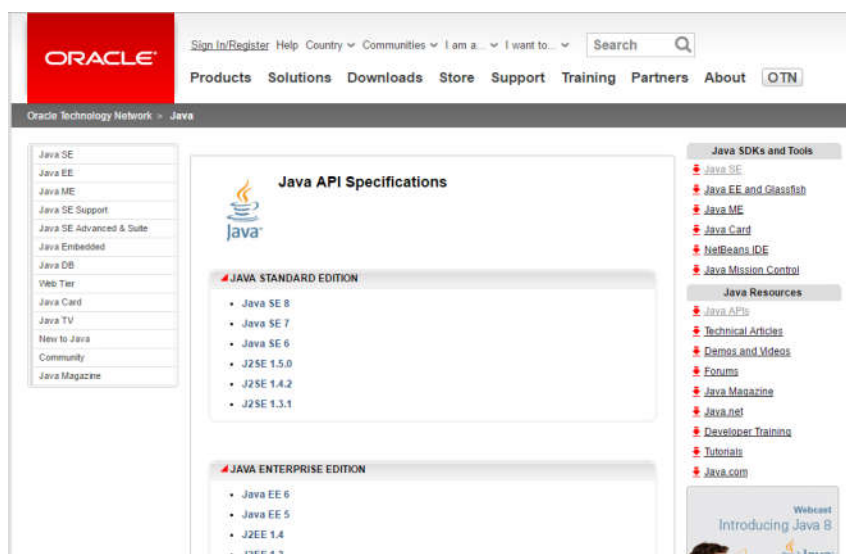
Como el lenguaje Java es un lenguaje orientado a objetos, la **API** de Java provee de un conjunto de clases utilitarias para efectuar toda clase de tareas necesarias dentro de un programa. La **API** Java está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.

La información completa del **API** de java se denomina **especificación** y sirve para conocer cualquier aspecto sobre las clases que contiene la **API**. Esta especificación es de crucial importancia para los programadores ya que en ella se pueden consultar los detalles de alguna clase que se quiera utilizar y ya no sería necesario memorizar toda la información relacionada.

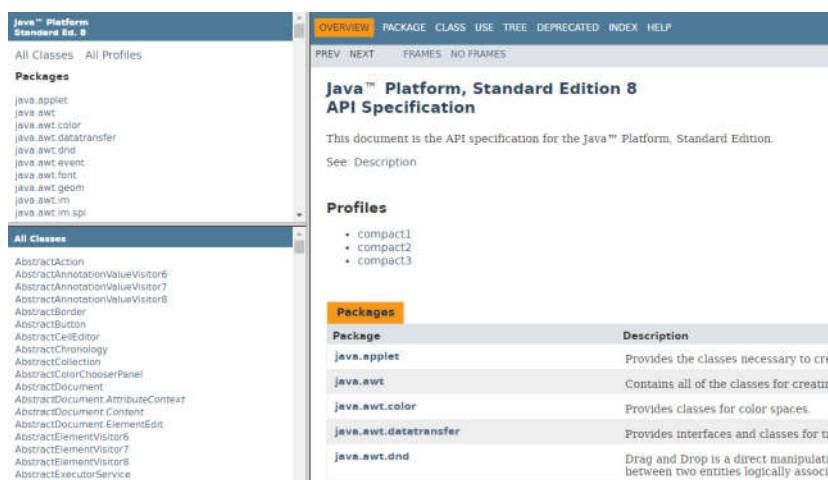
La especificación del **API** de java se encuentra en el sitio de Oracle: <http://www.oracle.com/technetwork/java/api-141528.html>.


	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	61/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En este sitio se puede consultar la especificación de las últimas versiones (ya que en cada versión se agregan o modifican algunas clases).



Una vez seleccionada la versión, se puede consultar el detalle de todas las clases que integran el API.



	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	62/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Manejo de cadenas

En Java las cadenas de caracteres no son un tipo de datos primitivo, sino que son objetos pertenecientes a la clase **String**.

La clase **String** proporciona una amplia variedad de métodos que permiten realizar las operaciones de manipulación y tratamiento de cadenas de caracteres habituales en un programa.

Para crear un objeto **String** podemos seguir el procedimiento general de creación de objetos en Java, utilizando el operador **new**. Ejemplo:

```
String s = new String("Texto de prueba");
```

Sin embargo, dada la amplia utilización de estos objetos en un programa, Java permite crear y asignar un objeto String a una variable de la misma forma que se hace con los tipos de datos primitivos. Entonces el ejemplo anterior es equivalente a:


```
String s = "Texto de prueba";
```

Una vez creado el objeto y asignada la referencia al mismo a una variable, puede utilizarse para acceder a los métodos definidos en la clase String (se pueden revisar en la documentación del API con java.lang.String). Los más usados son: length, equals, charAt, substring, indexOf, replace, toUpperCase, toLowerCase, Split, entre otros.

Ejemplo:

```
s.length();           //Devuelve el tamaño de la cadena
```

```
s.toUpperCase();      //Devuelve la cadena en mayúsculas
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	63/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Las variables de tipo String se pueden usar en una expresión que use el operador + para concatenar cadenas. Ejemplo:

```
String s = "Hola";
```

```
String t = s + " qué tal";
```

En Java las cadenas de caracteres son objetos **inmutables**, esto significa que una vez que el objeto se ha creado, no puede ser modificado.

Cuando escribimos una instrucción como la del ejemplo anterior, es fácil intuir que la variable *s* pasa a apuntar al objeto de texto *"Hola que tal"*. Aunque a raíz de la operación de concatenación pueda parecer que el objeto *"Hola"* apuntado por la variable *s* ha sido modificado en realidad no sucede esto, sino que al concatenarse *"Hola"* con *" que tal"* se está creando un nuevo objeto de texto *"Hola que tal"* que pasa a ser referenciado por la variable *s*. El objeto *"Hola"* deja de ser referenciado por dicha variable.


Java provee soporte especial para la concatenación de cadenas con las clases **StringBuilder** y **StringBuffer**. Un objeto **StringBuilder** es una secuencia de caracteres **mutable**, su contenido y capacidad puede cambiar en cualquier momento. Además, a diferencia de los **Strings**, los **builders** cuentan con una capacidad (capacity), la cantidad de espacios de caracteres asignados. Ésta es siempre mayor o igual que la longitud (length) y se expande automáticamente para acomodarse a más caracteres.

Los métodos principales de la clase **StringBuilder** son **append** e **insert**. Cada uno convierte un dato en **String** y concatena o inserta los caracteres de dicho String al **StringBuilder**. El método **append** agrega los caracteres al final mientras que **insert** los agrega en un punto específico.

Para hacer la misma concatenación que el ejemplo con String, quedaría:

```
StringBuilder sb = new StringBuilder("Hola");
```

```
sb.append(" que tal");
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	64/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Wrappers

Los **wrappers** o clases envoltorio son clases diseñadas para ser un complemento de los tipos primitivos. En efecto, los tipos primitivos son los únicos elementos de Java que no son objetos. Esto tiene algunas ventajas desde el punto de vista de la eficiencia, pero algunos inconvenientes desde el punto de vista de la funcionalidad.

Por ejemplo, los tipos primitivos siempre se pasan como argumento a los métodos por valor, mientras que los objetos se pasan por referencia. No hay forma de modificar en un método un argumento de tipo primitivo y que esa modificación se trasmita al entorno que hizo la llamada.

Una forma de conseguir esto es utilizar un **wrapper**, esto es un objeto cuya variable miembro es el tipo primitivo que se quiere modificar. Las clases **wrapper** también proporcionan métodos para realizar otras tareas con los tipos primitivos, tales como conversión con cadenas de caracteres en uno y otro sentido.

Existe una clase **wrapper** para cada uno de los tipos primitivos, las clases son: Byte, Short, Character, Integer, Long, Float, Double y Boolean (obsérvese que los nombres empiezan por mayúscula, siguiendo la nomenclatura típica de Java). Todas estas clases se encuentran en *java.lang*.

Todas las clases **wrapper** permiten crear un objeto de la clase a partir de tipo básico.


```
int k = 23;
```

```
Integer num = new Integer(k);
```

A excepción de Character, las clases **wrapper** también permiten crear objetos partiendo de la representación como cadena del dato.

```
String s = "4.65";
```

```
Float ft = new Float(s);
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	65/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Para recuperar el valor a partir del objeto, las ocho clases **wrapper** proporcionan un método con el formato `xxxValue()` que devuelve el dato encapsulado en el objeto donde xxx representa el nombre del tipo en el que se quiere obtener el dato.

```
float dato = ft.floatValue( );
```

```
int n=num.intValue( );
```

Las clases numéricas proporcionan un método estático `parseXxx(String)` que permite convertir la representación en forma de cadena de un número en el correspondiente tipo numérico donde xxx es el nombre del tipo al que se va a convertir la cadena de caracteres.

```
String s1 = "25, s2="89.2";
```

```
int n = Integer.parseInt(s1);
```

```
double d = Double.parseDouble(s2);
```

Auto boxing y unboxing


El **autoboxing** consiste en la encapsulación automática de un dato básico en un objeto wrapper mediante la utilización del operador de asignación.

Por ejemplo:

```
int p = 5;
Integer n = new Integer(p);
```


Equivale a:

```
int p = 5;
Integer n = p;
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	66/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Es decir, la creación del objeto **wrapper** se produce implícitamente al asignar el dato primitivo a la variable objeto. De la misma forma, para obtener el dato básico a partir del objeto **wrapper** no será necesario recurrir al método *xxxValue()*, esto se realizará implícitamente al utilizar la variable objeto en una expresión. A esto se le conoce como **autounboxing**. Para el ejemplo anterior:

```
int a = n;
```

	Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos	Código:	MADO-21
		Versión:	02
		Página	67/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Bibliografía

Martín, Antonio

Programador Certificado Java 2.

Segunda Edición.

México

Alfaomega Grupo Editor, 2008

Sierra Katy, Bates Bert

SCJP Sun Certified Programmer for Java 6 Study Guide

Mc Graw Hill

Dean John, Dean Raymond.

Introducción a la programación con Java

Primera Edición.

México

Mc Graw Hill, 2009