

Bienvenidos

Java Programming: A Comprehensive Hands-On

David R. Luna G.
davidrlunag@gmail.com
0412-3111011



Temario del Curso

- ✓ Introducción al Lenguaje Java
- ✓ **Estructura del Lenguaje Java**
- ✓ Herramientas de Desarrollo Java
- ✓ Programación Orientada a Objetos con Java
- ✓ Manejo de Datos con Archivos
- ✓ Trabajando con Base de Datos Relaciones
- ✓ Desarrollando GUIs

Estructura del Lenguaje Java

Comentarios

Los comentarios nos permiten introducir notas y aclaraciones en el momento de programar para resaltar alguna acción que se que se realiza en ciertas partes del código, para indicar un posible foco de problema o bien como una norma de estilo en la programación para ir completando la documentación interna del programa.

Existen tres formas diferentes de introducir comentarios entre el código de *Java*.

.- Comentarios de una Sola Linea (//)

```
//Esto es un comentario
```

```
i=8; //Se inicializa la Variable
```

.- Comentarios de Varias Lineas

```
/* Esta segunda forma es mucho más cómoda para comentar un número elevado de  
líneas ya que sólo requiere modificar el comienzo y el final. */
```

.- Comentarios para la Documentación

```
/**
```

```
 * @author
```

```
 * @version
```

```
 * @param
```

```
 * @return
```

```
 **/
```



Estructura del Lenguaje Java

Identificadores

Los identificadores nos permiten dar nombres a las variables, métodos, clases y objetos, sin olvidar que Java distingue entre mayúsculas y las minúsculas.

Los identificadores se forman de la siguiente manera:

- 1.- Comienzan con una letra (mayúscula o minúscula), un guión bajo (_) o un símbolo dólar (\$).
- 2.- A continuación, los siguientes caracteres pueden ser letras o dígitos.
- 3.- No existe una longitud máxima de caracteres.

Válidos

\$variable1
ClasePrueba
_uno
prueba
esVacio
edad
Persona
llorar

No Válidos

&prueba
1eraVez
!variable
<variable>
-sexo
~prueba



Estructura del Lenguaje Java

Palabras Claves

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



Estructura del Lenguaje Java

Tipos de Datos

En el lenguaje de programación Java, existen dos categorías de datos:

Primitivos

Una variable de tipo primitivo contiene un valor simple del tamaño y formato apropiado para su tipo: un número, un carácter, o un valor booleano (condicional verdadero ó falso).

Referenciales

Las variables referencia son referencias o nombres de una información más compleja: *arrays* u *objetos* de una determinada clase



Estructura del Lenguaje Java

Tipos de Datos Primitivos

Se llaman primitivos por el hecho de ser una herencia que dejaron lenguajes anteriores.

Tipo	Tamaño en Bites	Valor Mínimo	Valor Máximo
boolean	1 bit		
byte	8 bits	-2^7	2^7
char	16 bits	0	2^{16}
short	16 bits	-2^{15}	2^{15}
int	32 bits	-2^{31}	2^{31}
long	64 bits	-2^{63}	2^{63}
float	32 bits		
double	64 bits		



Estructura del Lenguaje Java

Variables

Una *variable* es un *nombre* que contiene un valor que puede cambiar a lo largo del programa, y se utilizan para almacenar los datos que se manejarán en nuestros programas.

Según el tipo de información que almacenen pueden ser:

Variables de Tipos Primitivos

Variables de Referencia

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

1.- Variables *miembro de una clase*: Se definen en una clase, fuera de cualquier método; pueden ser *tipos primitivos* o *referencias*.

2.- Variables *locales*: Se definen dentro de un método o más en general *dentro de cualquier bloque* entre *llaves* {}. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque. Pueden ser también *tipos primitivos* o *referencias*.



Estructura del Lenguaje Java

Variables

Antes de poder utilizar una variable es necesario declararla, para que el compilador le asigne un espacio en memoria para ser usado por ella, cuyo espacio dependerá del tipo de dato que se vaya a almacenar en ella.

[modificadores_acceso] tipo_dato nombre_variable;

```
int numero;          boolean raso_examen;  
char edocivil;       float sueldo;      String Cadena;
```

A las variables locales es necesario siempre inicializarlas. Las variables de clases o miembro se inicializan de manera automática

```
Numero=900;          paso_examen=false;  
Edocivil='C';         sueldo=900_000,50;
```

Puedo inicializarlas al momento de declararlas:

```
int numero=900;      boolean raso_examen=false;  
char edocivil='c';   float sueldo=90000,50f;
```



Estructura del Lenguaje Java

Variables

Tipo	Tamaño en Bites	Valor Mínimo	Valor Máximo
boolean	1 bit		
byte	8 bits	-2^7	2^7
char	16 bits	0	2^{16}
short	16 bits	-2^{15}	2^{15}
int	32 bits	-2^{31}	2^{31}
long	64 bits	-2^{63}	2^{63}
float	32 bits		
double	64 bits		



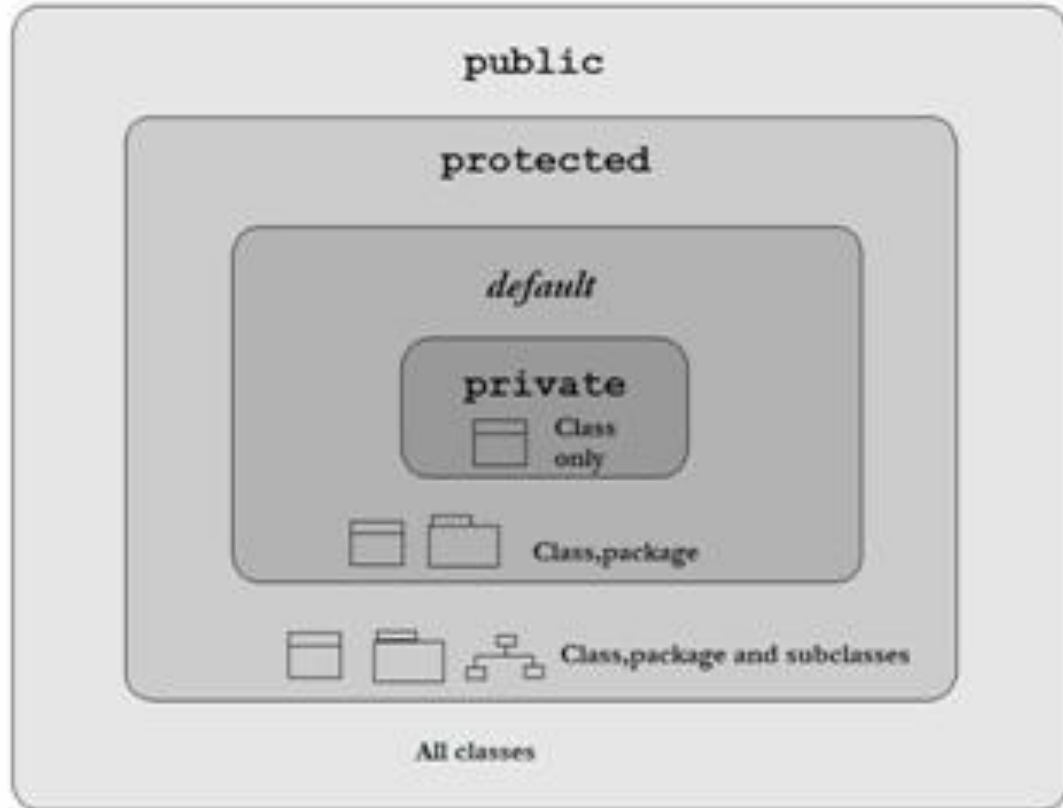
Estructura del Lenguaje Java

```
int money = 100_000_000;  
long creditCNumber = 3434_3423_4343_4232L;  
int mask = 0b0000_0000_0000_0000_0000_0000_1111_1111;
```



Introducción

Modificadores de Acceso



Modificadores de Clase

Solamente 2 modificadores son permitidos:

Public y default (sin modificador)

Si una clase es publica, entonces puede ser accesada desde cualquier parte

Si una clase no tiene modificador, entonces solo podrá ser accesada desde el mismo paquete

Modificadores de Variables y Métodos

public and no modifier – the same way as used in class level.

private – members CAN ONLY access.

protected – CAN be accessed from 'same package' and a subclass existing in any package can access.



Introducción

Modificadores de Acceso

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y



Estructura del Lenguaje Java

Enumeraciones

Los tipos enumerados sirven para restringir el contenido de una variable a una serie de valores predefinidos. Esto suele ayudar a reducir los errores en nuestro código.

Los tipos enumerados se pueden definir fuera o dentro de una clase. Otra ventaja que traen los tipos enum de Java es que al ser una “especie de clase” podemos añadirles métodos, variables de instancia, constructores, etc... lo que los hace muy potentes.

```
enum Test Enum{  
    Valor1, Valor2, Valor3  
}
```

Enum.Valor1



Estructura del Lenguaje Java

La Clase Scanner

La Clase Scanner nos va a ayudar a leer los datos de una forma más sencilla que el habitual manejo de Lectura de datos por Consola con Java.

La utilización de la clase Scanner es muy sencilla. Lo primero que tenemos que hacer es declarar un objeto Scanner [r](#) instanciándolo contra la consola, es decir, contra el objeto System.in. Para usarla debemos importarla del paquete java.util

```
Scanner reader = new Scanner(System.in);  
String sTexto = reader.next();
```



Estructura del Lenguaje Java

Casting

En muchas ocasiones hay que transformar una variable de un tipo a otro, por ejemplo de *int* a *double*, o de *float* a *long*.

En Java se realizan de modo automático conversiones implícitas de un tipo a otro de más precisión, por ejemplo de *int* a *long*, de *float* a *double*, etc.

Las conversiones de un tipo de mayor a otro de menor precisión requieren una orden explícita del programador, pues son conversiones inseguras que pueden dar lugar a errores (por ejemplo, para pasar a *short* un número almacenado como *int*, hay que estar seguro de que puede ser representado con el número de cifras binarias de *short*).

A estas conversiones explícitas de tipo se les llama *cast* o *casting*. El *casting* se hace poniendo el tipo al que se desea transformar entre paréntesis, como por ejemplo,

```
long result;  
result = (long) (a/(b+c));
```



Estructura del Lenguaje Java

Casting

```
short a=1, b=1, c=0;  
c=a+b;           //error de compilación
```

```
short a=1, b=1, c=0;  
c=(short)(a+b);  //Casting para evitar el error
```

No deben olvidar que para los tipos enteros el valor por defecto es INT y para los valores reales es DOUBLE



Estructura del Lenguaje Java

Operadores

Los operadores realizan algunas funciones en uno o dos operandos. Los operadores que requieren un operador se llaman operadores unarios. Por ejemplo, ++ es un operador unario que incrementa el valor su operando en uno.

Los operadores que requieren dos operandos se llaman operadores binarios. El operador = es un operador binario que asigna un valor del operando derecho al operando izquierdo.

Los operadores unarios en Java pueden utilizar la notación de prefijo o de sufijo.

operador op1 o op1 operador

Todos los operadores binarios de Java tienen la misma notación, es decir aparecen entre los dos operandos:

op1 operador op2



Estructura del Lenguaje Java

Operadores Aritméticos

Operador	Uso	Descripción
+	op1 + op2	Suma op1 y op2
-	op1 - op2	Resta op2 de op1
*	op1 * op2	Multiplica op1 y op2
/	op1 / op2	Divide op1 por op2
%	op1 % op2	Obtiene el resto de dividir op1 por op2
++	op ++	Incrementa op en 1; evalúa el valor antes de incrementar
++	++ op	Incrementa op en 1; evalúa el valor después de incrementar
--	op --	Decrementa op en 1; evalúa el valor antes de decrementar
--	-- op	Decrementa op en 1; evalúa el valor después de decrementar



Estructura del Lenguaje Java

Operadores Relacionales

Operador	Uso	Devuelve true si
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
!=	op1 != op2	op1 y op2 son distintos



Estructura del Lenguaje Java

Operadores Lógicos

Operador	Uso	Devuelve true si
&&	op1 && op2	op1 y op2 son verdaderos
	op1 op2	uno de los dos es verdadero
!	! op	op es falso



Estructura del Lenguaje Java

Operadores de Asignación

Operador	Uso	Equivale a
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2



Estructura del Lenguaje Java

Estructuras de Control

- Las estructuras de control de flujo alteran la naturaleza secuencial de un programa.
- Las estructuras de control son: seleccion e iteracion
- La seleccion previene condicionalmente la ejecucion de una instruccion.

La condición es una
expresión booleana

```
if (condición)  
    estatuto;  
  
else  
    estatuto;
```

Cláusula if

Cláusula else
(opcional)



Estructura del Lenguaje Java

Estructuras de Control

```
if (condición)
{
    bloque de estatutos
}
else
{
    bloque de estatutos
}
```

Si se requiere poner más de un estatuto, los estatutos se deben encerrar entre llaves

El else es opcional



Estructura del Lenguaje Java

Estructuras de Control

Ejemplo:

```
if(calif >= 70)
    System.out.println("Aprobado");
else {
    System.out.println("Reprobado");
    System.out.println("DEBES CURSARLA");
}
```



Estructura del Lenguaje Java

Estructuras de Control

- Las opciones de una selección pueden no ser excluyentes. Ejemplo:

```
x=6;
```

```
if(x>5) x++;
```

```
if(x>4) x--;
```

```
if(x<10) x*=2;
```

- Para forzar la exclusion se requieren combinaciones else if

```
x=6;
```

```
if(x>5) x++;
```

```
else if(x>4) x--;
```

```
else if(x<10) x*=2;
```




Estructura del Lenguaje Java

Operador ?:

```
if (condition)  
    x = exp1;  
else x = exp2;
```

x = condition ? exp1 : exp2;



Estructura del Lenguaje Java

Selección Multiple - Switch

```
int llave;  
switch (clave) {  
    case 3 : .....  
        break;  
    case 2: .....  
        break;  
    .....  
    case <valor>: .....  
        break;  
    default:.....  
        break;  
}..
```

- clave puede ser de tipo
 - byte, char, short, int, long, String (desde la 1.7)
- Ejecuta a partir del caso que se cumple.
 - clave == valor
- Hay que insertar break para no ejecutar los siguientes casos.
- El default es opcional



Estructura del Lenguaje Java

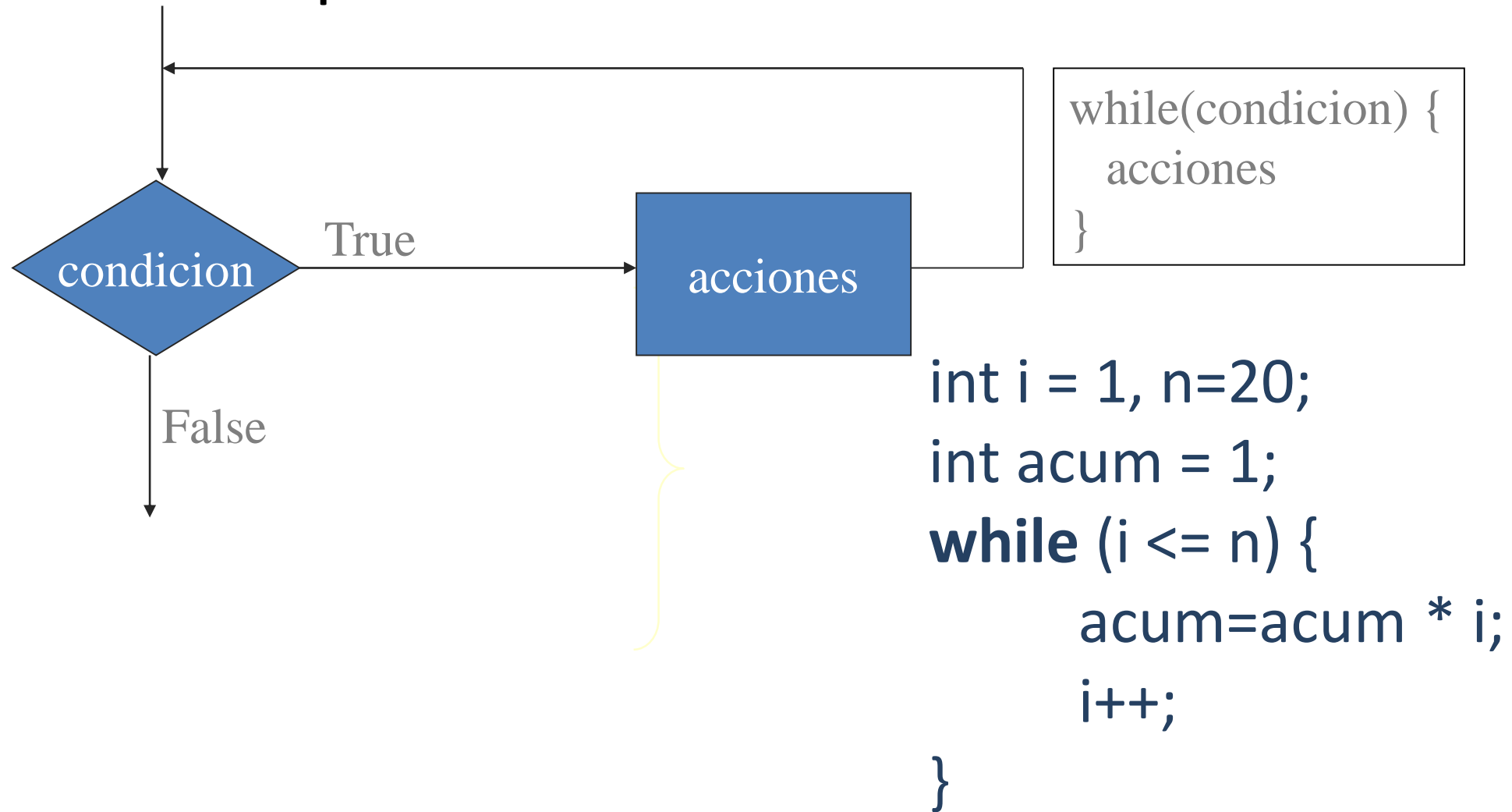
Selección Multiple - Switch

- El selector debe ser un valor constante
 - case 2
 - case 1+1
 - case uno + 1 // uno es un constante entera.
- Se pueden poner varios selectores anidados:
 - case 'd': case 'D'
 - se ejecutan las sentencias asociadas si la expresión de control coincide con alguna de los valores del caso



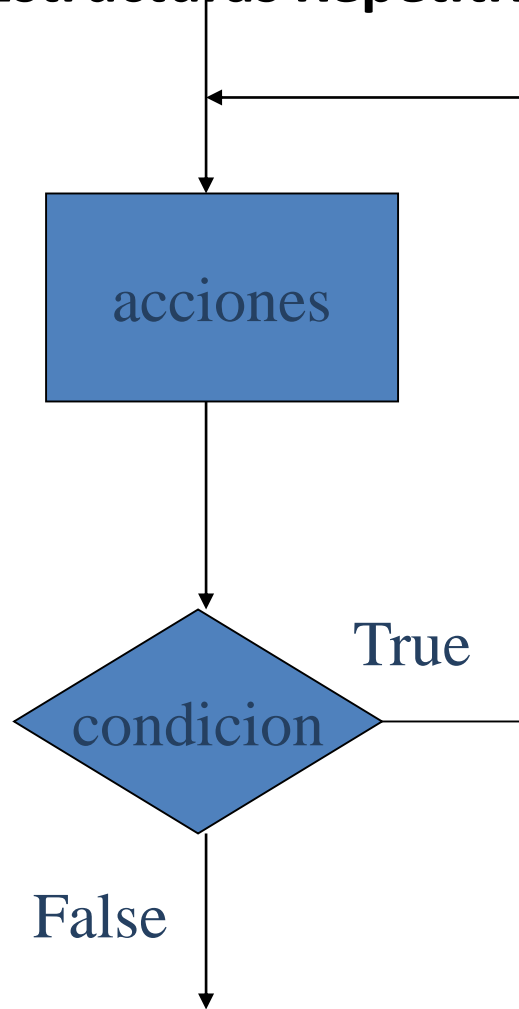
Estructura del Lenguaje Java

Estructuras Repetitivas



Estructura del Lenguaje Java

Estructuras Repetitivas



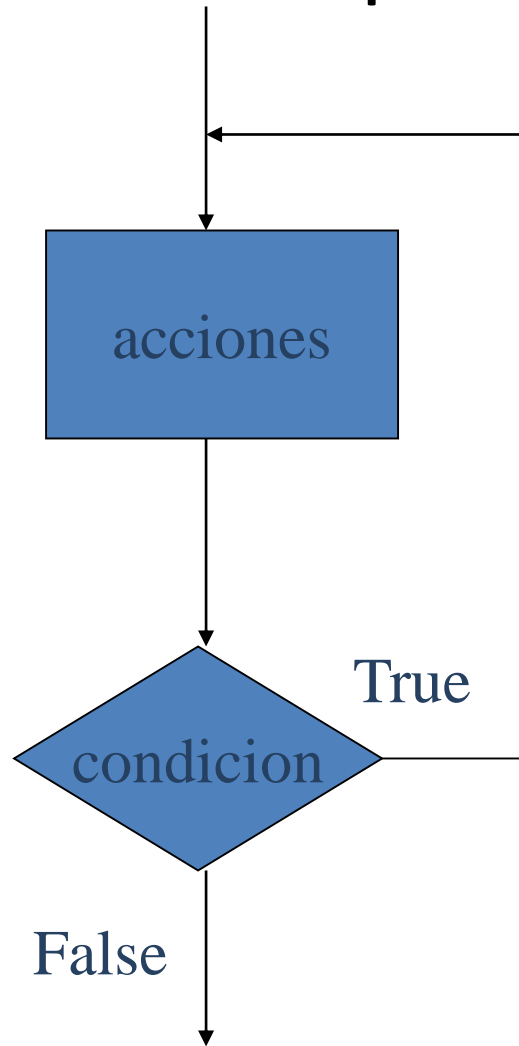
```
do {  
    acciones  
}while (condicion);
```

```
int i = 1, n=20;  
int acum = 1;  
do {  
    acum=acum * i;  
    i++;  
while (i <= n);
```



Estructura del Lenguaje Java

Estructuras Repetitivas



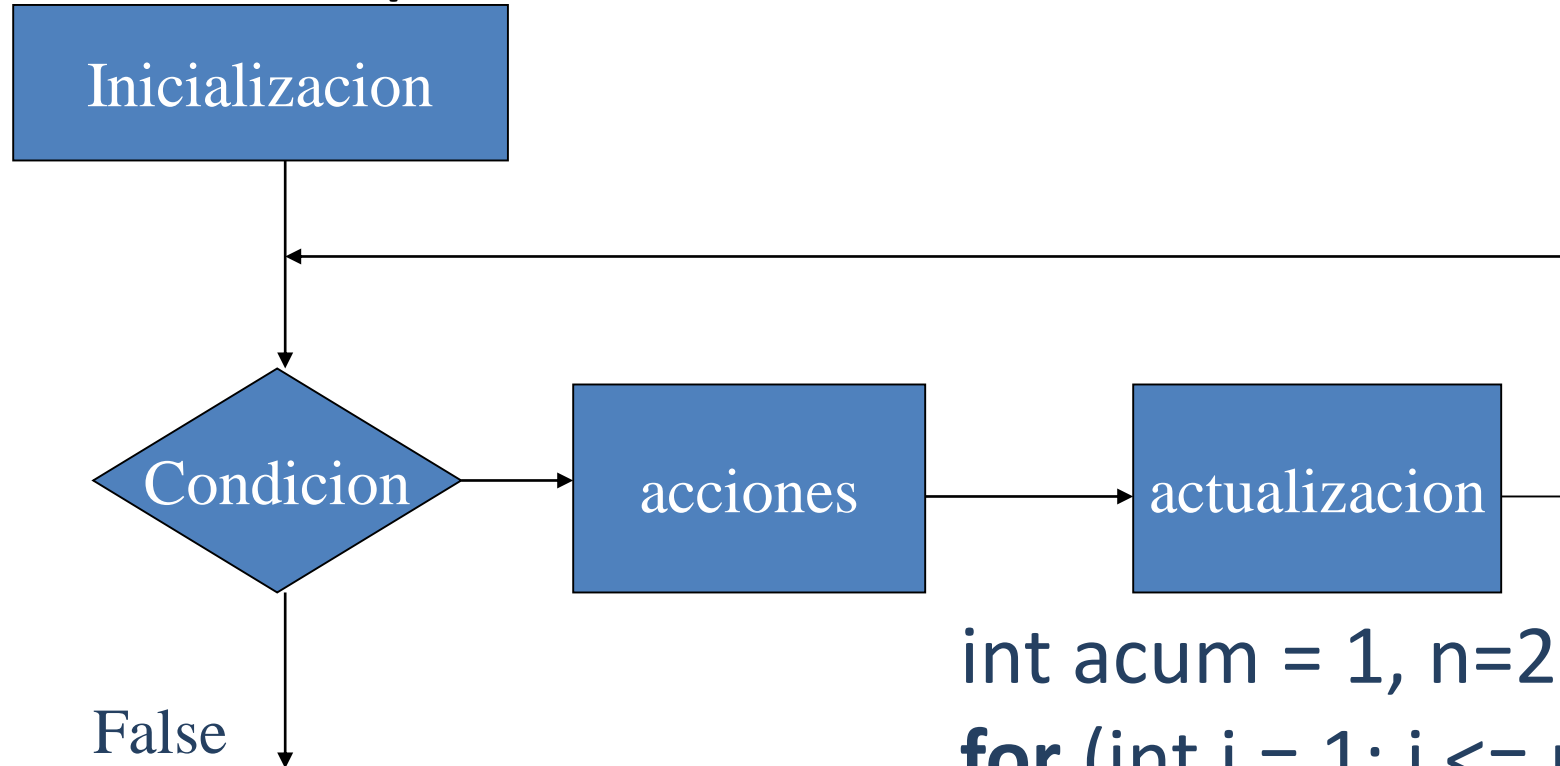
```
do {  
    acciones  
}while (condicion);
```

```
int i = 1, n=20;  
int acum = 1;  
do {  
    acum=acum * i;  
    i++;  
while (i <= n);
```



Estructura del Lenguaje Java

Estructuras Repetitivas



```
int acum = 1, n=20;  
for (int i = 1; i <= n; i++) {  
    acum=acum * i;  
}
```



Estructura del Lenguaje Java

Estructuras Repetitivas

- Se pueden inicializar, e incluso declarar, cualquier número de variables dentro de la estructura separándolas con comas. Ejemplo:

- `//int i,j;`

`for(int j=2; condición, actualización)`

- Si la variable ha sido declarada antes del bucle, el valor que tiene al terminar sigue existiendo, en caso contrario deja de tener validez al terminar.

- Se pueden actualizar varias variables separadas por comas. Ejemplo:

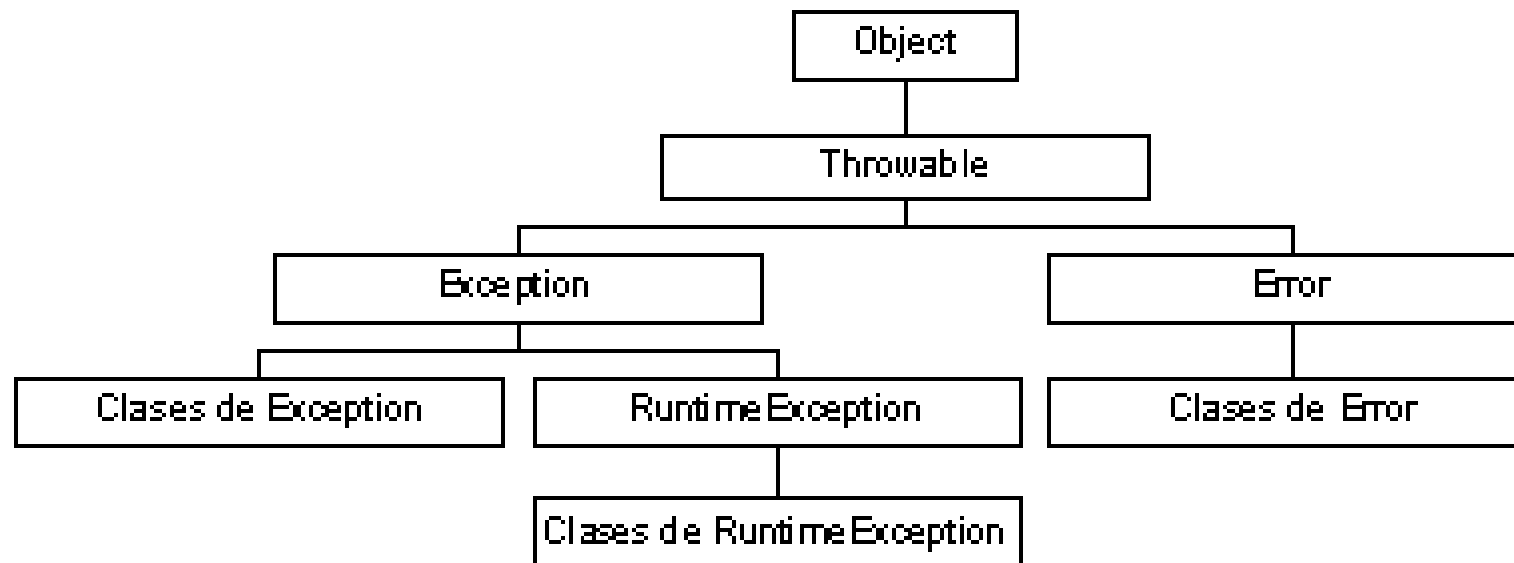
`for(int i=1, j=2; (j<10)&&(i>2); i++, j=j+i)`



Estructura del Lenguaje Java

Manejo de Excepciones

Java incorpora en el propio lenguaje la gestión de errores. En el lenguaje *Java*, una *Exception* es un cierto tipo de error o una condición anormal que se ha producido durante la ejecución de un programa. Algunas *excepciones* son *fatales* y provocan que se deba finalizar la ejecución del programa.



Estructura del Lenguaje Java

Manejo de Excepciones

Cuando el programador va a ejecutar un trozo de código que pueda provocar una excepción (por ejemplo, una lectura en un fichero), debe incluir este fragmento de código dentro de un bloque *try*:

```
try {  
    // Código posiblemente problemático  
}
```

Pero lo importante es cómo controlar qué hacer con la posible excepción que se cree. Para ello se utilizan las cláusulas *catch*, en las que se especifica que acción realizar:

```
try {  
    // Código posiblemente problemático  
} catch( tipo_de_excepcion e) {  
    // Código para solucionar la excepción e  
} catch( tipo_de_excepcion_mas_general e) {  
    // Código para solucionar la excepción e  
}
```



Estructura del Lenguaje Java

Métodos

Los métodos son subrutinas o funciones que manipulan los datos definidos por la clase, y en muchos casos, proveen acceso a esos datos. En código bien escrito de Java, cada método desempeña solo una tarea. Cada método tiene un nombre y este nombre se utiliza para llamar al método. Un método posee paréntesis después del nombre, para diferenciarlo de las variables, así como también recibir parámetros para su funcionamiento. La lista de parámetros es una secuencia de parejas tipo e identificador, separados por coma. Siempre deben devolver un valor.

Nos permiten reutilizar el código

Trabajar más eficiente

Trabajar más ordenado



Estructura del Lenguaje Java

Métodos

```
private | protected | public ][static][abstract][final][native][synchronized]  
TipoDevuelto NombreMétodo( [tipo1 nombre1[, tipo2 nombre2 ]...] ) [throws  
excepción1 [,excepción2]... ]
```

Declaración	public protected package private	controla el nivel de acceso para el método	Componente obligatorio
	static	método de clase	
	abstract	método no implementado	
	final	método no puede ser sobrescrito por las subclases	
	native	método implementado en otro lenguaje	
	synchronized	método que requiere de un monitor para ejecutarse	
	tipo_retornado NombreDelMetodo	devuelve un objeto del tipo tipo_retornado. Los tipos pueden ser: Primitivos (int, float, boolean) o Referenciados (array, object o nombres de interfaces)	
	(lista de parámetros)	lista de argumentos (tipo nombre) separados por coma (","). Los tipos pueden ser: Primitivos (int, float, boolean) o Referenciados (array, object o nombres de interfaces)	
Cuerpo	throws exceptions	excepciones manejadas por el método	
	{ Cuerpo del Método }		



Estructura del Lenguaje Java

Métodos

```
public void MostrarData(){
    System.out.println("Mi Nombre es: David Luna");
}

public int getEdad(){
    return edad;
}

public float calculaArea(){
    int base=10, altura=5;
    return base*altura/2;
}
```

Los métodos pueden definir variables locales. Su visibilidad llega desde la definición al final del bloque en el que han sido definidas. No hace falta inicializar las variables locales en el punto en que se definen, pero el compilador no permite utilizarlas sin haberles dado un valor. A diferencia de las variables miembro, las variables locales no se inicializan por defecto.



Estructura del Lenguaje Java

Métodos

Los métodos pueden recibir parametros, es decir, valores de variables que serán pasados desde otro punto de nuestro programa y que son necesarios para el correcto funcionamiento de nuestro método.

La lista de parámetros es una secuencia de parejas tipo e identificador, separados por coma. Siempre deben devolver un valor

```
public float calculaArea(int base, int altura){  
    return base*altura/2;  
}
```

En Java los argumentos de los tipos primitivos se pasan siempre por valor. El método recibe una copia del argumento actual; si se modifica esta copia, el argumento original que se incluyó en la llamada no queda modificado. La forma de modificar dentro de un método una variable de un tipo primitivo es incluirla como variable miembro en una clase y pasar como argumento una referencia a un objeto de dicha clase.



Estructura del Lenguaje Java

Métodos

```
public class Metodos{
    public int factorial(int x){
        int fact=1;
        for (int i=1;i<=x;i++)
            fact*=i;
        return (fact);
    }
    public static void main(String a[]){
        int x=5, factor;
        System.out.println("Le voy a Calcular el Factorial a "+x);
        factor=factorial(x);
        System.out.println("El factorial de" + x + "es =" + factor);
        System.out.println("El factorial de" + x + "es =" +
factorial(x));
    }
}
```



Estructura del Lenguaje Java

Métodos

Como ya vimos poder utilizar el modificador static con los métodos, el cual nos va a permitir crear métodos que pueden ser utilizados directamente con la clase, no hará falta crear una instancia de la clase para usarlos. En otras palabras, para todas las instancias solo se cargará en memoria un método.

```
public static void mensaje(){
    System.out.println("Prueba");
}

public class Prueba{
    public static int factorial(int x){
        int fact=1;
        for (int i=1;i<=x;i++)
            fact*=i;
        return (fact);
    }
    public static void Mensaje(){
        System.out.println("Soy un Metodo Estatico");
    }
}

public class Prueba2{
    public static void main(String a[]){
        System.out.println("El factorial de 5
es=" +Prueba.factorial(5));
        Prueba.Mensaje();
    }
}
```



Estructura del Lenguaje Java

Constructores

Un constructor es un método especial que inicia un objeto inmediatamente después de su creación. De esta forma nos evitamos el tener que iniciar las variables explícitamente para su iniciación.

El constructor tiene exactamente el mismo nombre de la clase que lo implementa; no puede haber ningún otro método que comparta su nombre con el de su clase. Una vez definido, se llamará automáticamente al constructor al crear un objeto de esa clase (al utilizar el operador *new*).

El constructor no devuelve ningún tipo, ni siquiera *void*. Su misión es iniciar todo estado interno de un objeto (sus atributos), haciendo que el objeto sea utilizable inmediatamente; reservando memoria para sus atributos, iniciando sus valores...

Si no lo creamos, la máquina virtual de Java nos provee uno, el cual es llamado constructor por defecto, que tiene la siguiente estructura:

```
nombreClase(){  
}
```



Estructura del Lenguaje Java

Constructores

Si proveemos nuestro propio constructor, Java no creará el Constructor por Defecto, por lo que tenemos que tener cuidado a la hora de implementarlo, ya que podemos hacer que reciba parametros, y a la hora de crear objetos habrá que pasarle los parametros.

```
nombreClase(int x){  
}
```

A la hora de crear un objetos, debemos hacerlo llamando al constructor y pasarle un parametro entero, en caso contrario, no se podrá crear objeto de ese tipo.

```
NombreClase obj1 = new NombreClase(8);
```

Esto nos puede ayudar a obligar a quien quiera crear un objeto a tener que pasarle los parámetros para inicializarlo con esos valores.



Estructura del Lenguaje Java

Constructores

- Todas las clases en java tienen al menos uno.
- Si no se define ningún constructor, java activa el constructor por defecto que no utiliza argumentos y no realiza inicializaciones especiales.
- Si hay argumentos, busca el constructor adecuado y ejecuta el código contenido en él.
- Un constructor es un método especial usado para definir un objeto.
- Tiene el mismo nombre que la clase
- Debe ser publico para ser usado por otras clases
- Usualmente establece valores iniciales de las variables de instancia de un objeto
- En caso de no definirse explícitamente Java asigna un constructor vacío.
- Puede haber varios constructores
- No tiene ningún tipo de regreso, ni siquiera void
- Son invocados con el operador new
- El operador new crea la nueva instancia de la clase, sin inicializar, se llama al constructor al que se le pasa el objeto de modo implícito (this).



Estructura del Lenguaje Java

Sobrecarga de Métodos

Es posible que necesitemos crear más de un método con el mismo nombre, pero con listas de parámetros distintas. A esto se le llama *sobrecarga del método*. La sobrecarga de método se utiliza para proporcionar a Java un comportamiento *polimorfo*. Esto nos permite brindar una sola interfaz para un mismo método, a quien vaya a utilizar nuestras clases.

nombreMetodo(listaParametros)>>> FIRMA

```
public int factorial(int x){  
    int fact=1;  
    for (int i=1;i<=x;i++)  
        fact*=i;  
    return (fact);  
}
```

```
public int factorial(double x){  
    int fact=1;  
    for (int i=1;i<=x;i++)  
        fact*=i;  
    return (fact);  
}
```

La Máquina Virtual de Java, hace un chequeo de la firma que se está llamando y la busca en la clase, por lo que el podrá determinar cual es el método que se esté llamando. No pueden existir métodos con la misma firma, dará un error de compilación, al igual que si no existe el método con la firma que se está llamando.



Estructura del Lenguaje Java

Arreglos

1. Estructuras de Datos
2. Contienen varios elementos relacionados del mismo tipo
3. Estáticos
Conservan el mismo tamaño
4. Existen clases dinámicas tipo arreglo
Pueden variar su tamaño
5. En Java los arreglos son objetos



Estructura del Lenguaje Java

Arreglos

Arreglo

Grupo de localidades consecutivas de memoria

Mismo nombre y tipo

Para referirse a un elemento, se debe especificar

Nombre del Arreglo

Posición del Arreglo


Formato:

nombrearreglo[posicion numero]


vector[0]

Primer elemento en la posición **0**

El enésimo elemento del arreglo **c**: **c[0], c[1]...c[n-1]**



c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78



Estructura del Lenguaje Java

Arreglos

Como otras variables, antes de poder utilizar un array primero se debe declarar. De nuevo, al igual que otras variables, la declaración de un array tiene dos componentes primarios: el tipo del array y su nombre

```
tipo_dato nombre_arreglo[];
```

```
tipo_dato []nombre_arreglo;
```

```
int vector[]; int []vector;
```

```
vector
```

```
null
```

La parte **int[]** de la declaración indica que vector es un array de enteros. La declaración no asigna ninguna memoria para contener los elementos del array. Si se intenta asignar un valor o acceder a cualquier elemento de **vector** antes de haber asignado la memoria para él, el compilador dará un error como este y no compilará el programa



Estructura del Lenguaje Java

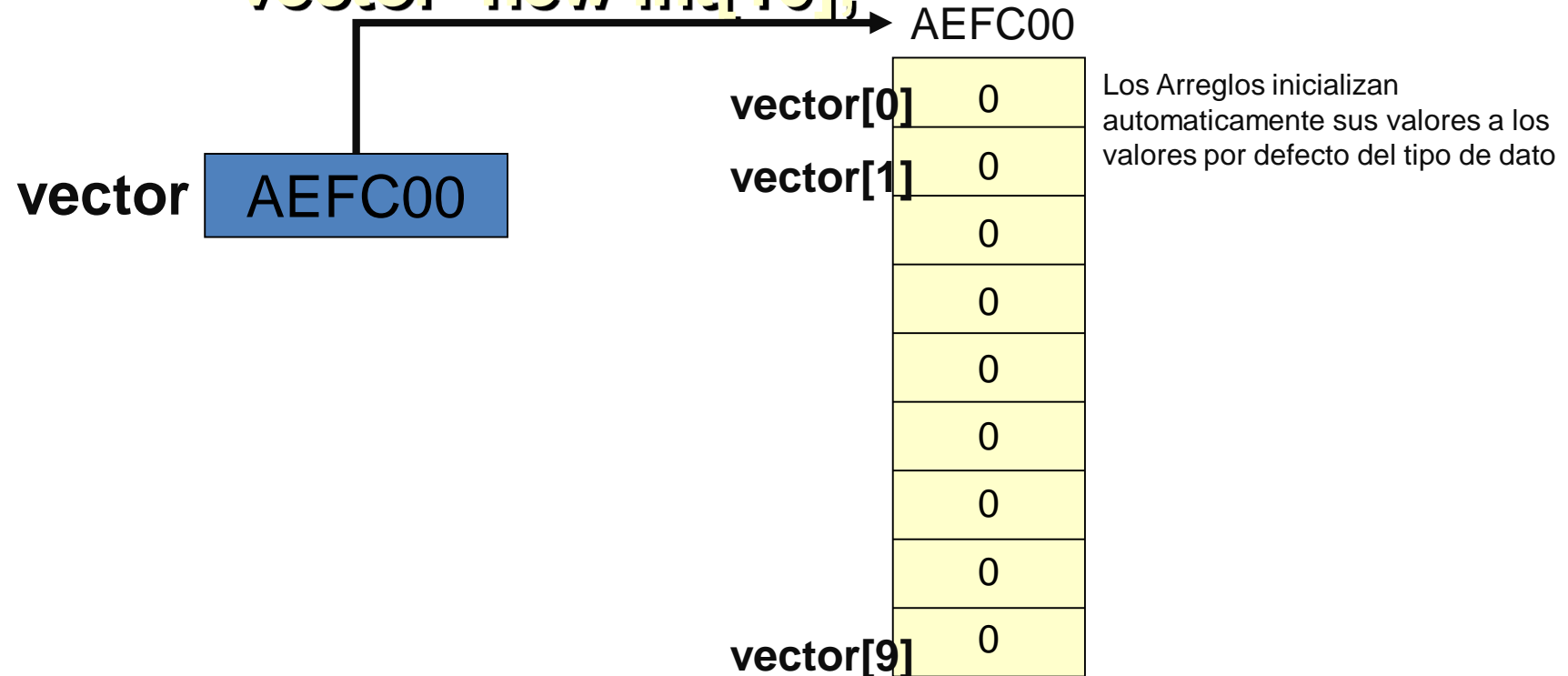
Arreglos

Para poder usar los arreglos es necesario asignarles un tamaño. Se puede lograr esto de dos formas:

Usando el operador new

`new tipo_dato[tamaño_vector]`

`vector=new int[10];`



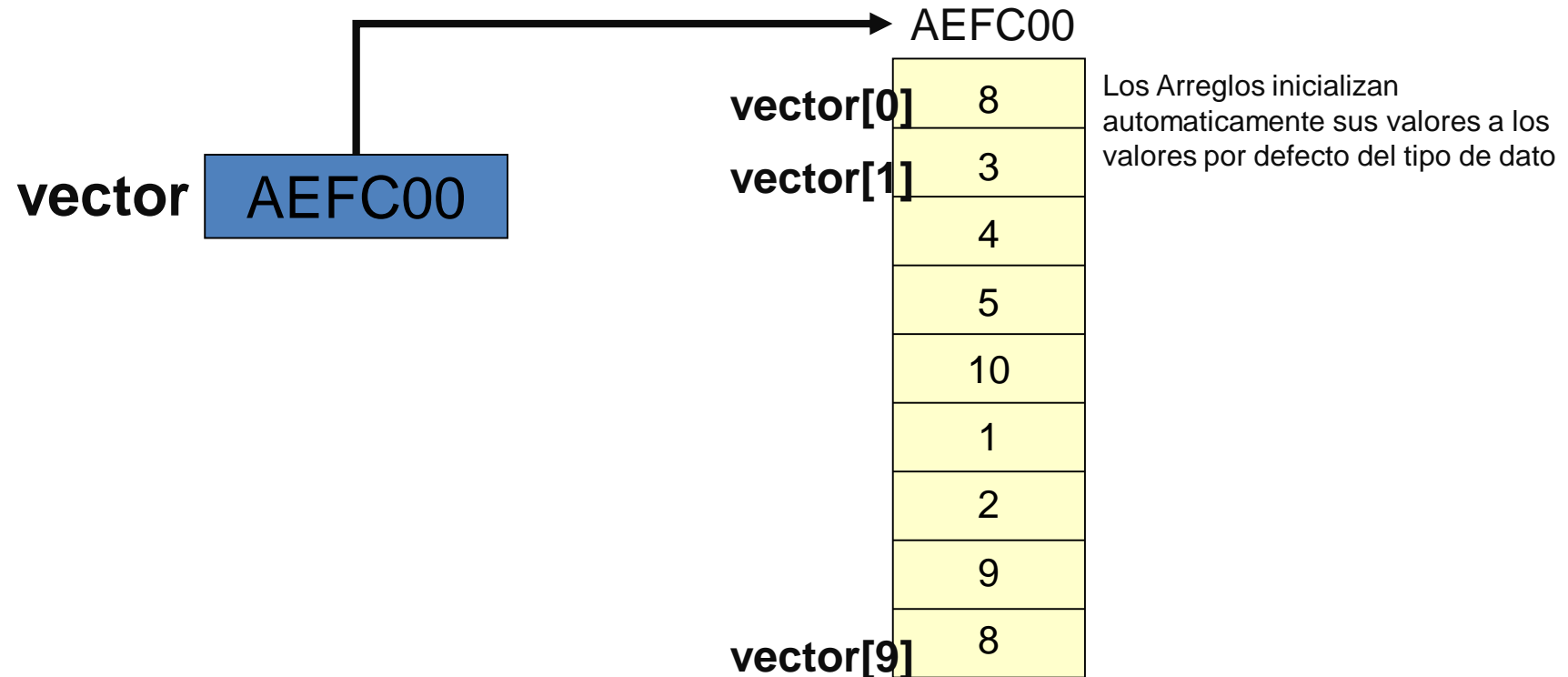
Estructura del Lenguaje Java

Arreglos

Para poder usar los arreglos es necesario asignarles un tamaño. Se puede lograr esto de dos formas:

Asignando los Valores Directamente

```
Int vector[]={8,3,4,5,10,1,2,9,7,8}
```



Estructura del Lenguaje Java

Arreglos

Podemos Inicializar el arreglo en el mismo momento de declararlo

```
Tipo_dato nombre_arreglo=new tipo_dato[longitud]
```

```
int vector[]=new int[10];
```

```
int vector[]={8,3,4,5,10,1,2,9,7,8}
```

Declarando varios Arreglos del Mismo Tipo

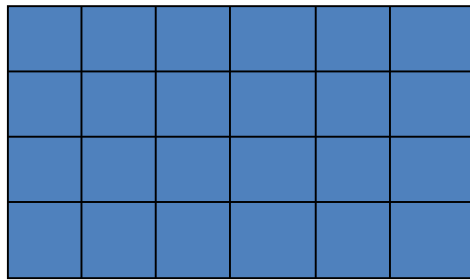
```
int [] vector1, vector2, vector3;
```



Estructura del Lenguaje Java

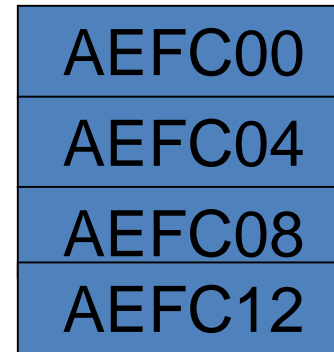
Arreglos

En Java una matriz es un vector de vectores fila, o más en concreto un vector de referencias a los vectores fila. Con este esquema, cada fila podría tener un número de elementos diferente.

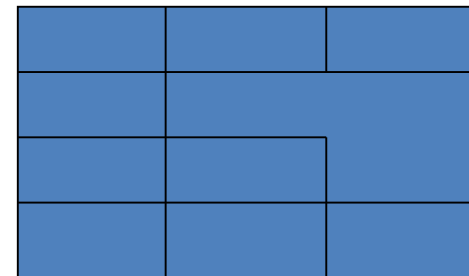


Matriz

Matriz



AEFC00	AEFC08
8	8
3	3
4	
AEFC04	AEFC12
8	8
	3
	4



Matriz



Estructura del Lenguaje Java

Arreglos

Para poder usar los arreglos es necesario asignarles un tamaño. Se puede lograr esto de dos formas:

Usando el operador new

```
new tipo_dato[filas][columnas]
```

```
new tipo_dato[filas][]
```

```
matriz=new int[10][10];
```

```
matriz=new int[10][];
```

matriz AAAC00

AAAC00

AEFC00
AEFC04
AEFC08
AEFC12

AEFC00

8
3
4

AEFC04

8

AEFC08

8
3

Los Arreglos inicializan automáticamente sus valores a los valores por defecto del tipo de dato

