Java SE

Curso básico de Java Standard Edition

Arturo Rivas Rojas

1 de enero de 2019

Índice general

CAPÍTULO 1

Introducción

1.1. ¿Qué es?

Java es un lenguaje de programación de **alto nivel**.

Caracteristicas:

- Simple
- Orientado a Objetos
- Distribuido
- Multihilo
- Arquitectura Neutral
- Portable
- Alto desempeño
- Seguro

Java es un lenguaje compilado e interpretado.

1.2. Filosofía

Escribe una vez y correlo donde quieras.

1.3. Origen

Surge en 1991 creado por James Goslin, y fue absorvido por la compañia Sun microsystems.

Nacio con el proposito de comunicar dispositivos de forma configurable y portable.

En el 2009 Java llego a su apogeo y se volvio propiedad de Oracle. Nacieron las certificaciones Java.

1.3.1. Clasificación

- Java SE (Standard Edition) : base y sintaxis del lenguaje.
- Java EE (Enterprise Edition) : ultima generación.

1.4. Herramientas de desarrollo

Para poder trabajar con Java es necesario tener estos 2 componentes:

- JDK (Java Developer Kit): kit de desarrollo.
- JRE (Java Run Enviroment): para correr las aplicaciones.

1.4.1. Maquina Virtual

Una vez que tenemos nuestro codigo java y lo compilamos con el comando **javac** se genera una archivo .class que contine ByteCode.

La maquina virtual de Java a la que se llama con el comando **java** recibe de entrada ByteCode y lo interpreta.

1.5. Metodo Main

Es el punto de entrada de toda aplicación Java. los argumentos que se envian al programa se reciben en el main.

1.6. Entorno de Desarrollo Integrado (IDE)

Un entorno de desarrollo integrado es una herraminta que tiene todo lo necessario en una sola aplicación la terminal, editor de texto , buscador de archivos, etc.

CAPÍTULO 2

Tipos de Datos

Saber que tipo de dato ocupar nos ayuda a evitar el desperdicio de memoria.

2.1. Primitivos

Se pueden identificar porque se escriben con minúsculas.

2.1.1. Enteros

- **byte** (1 byte) -2^7 a 2^7-1.

```
byte age = 127;
```

- **short** (2 bytes) -2^15 a 2^15-1.

```
short year = 2019;
```

- **int** (4 bytes) -2^32 a 2^32-1.

```
int id_user = 102301;
```

- **long** (8 bytes) -2^64 a 2^64-1.

```
long id_twitter = 123423552532L;
```

Es necesario agregar una **L** al final del dato para señalar que es del tipo long.

2.1.2. Flotantes

- **float** (4 bytes).

```
float diameter = 34.25f;
```

Se debe añadir una f al final para decirle al compilador que es un float y no un double.

- **double** (8 bytes).

```
double price = 1234.256618177892;
```

2.1.3. Texto

- **char** (2 bytes) unicode.

```
char gender = 'M';
```

Se deben trabajar entre comillas simples ’.

2.1.4. Logicos

- **boolean** (2 bytes) true or false.

```
boolean isVisible = true;
```

CAPÍTULO 3

Variables

Una variable es un espacio de memoria a la que se le asigna un valor.

3.1. Naming

- Sensible a minúsculas y mayúsculas.
- Su expresión regular seria ^[a-z$_][\w$]*

Las clases se escriben con el estandar **upper camel case** mientras que los metodos y variables con **lower camel case**.

3.2. Cast

Es hacer una conversión entre tipos de datos.

De un tipo de dato que ocupa menos memoria a uno con mayor memoria el casteo es automatico.

En el caso opuesto se debe de definir a que tipo de dato se desea castear.

```
// primer caso
byte b = 6;
short s = b;
// segundo caso
s = 100;
b = (byte) s;
```

Cuando se castea un float a entero este se trunca, es decir desaparece lo que exista después del punto decimal.

En el caso de los char, castear un entero a un char nos regresa el caracter que ese entero representa en el código ASCII.

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Figura 3.1: código ASCII

3.3. Arreglos (Arrays)

Un arreglo es un objeto donde se pueden guardar multiples variables de un mismo tipo.

Entonces podemos decir que un arreglo es un contenedor finito de un tipo de variable definido.

Un arreglo de una solo dimensión es una fila, uno de dos dimensiones es una matriz, uno de tres dimensiones es un cubo.

Para poder imaginarnos un array de n-dimensiones lo haremos visualizando a cada array como una matryoshka de n-1 y una caja donde hay varias de estas matryoshka.

Figura 3.2: matryoshka

3.4. Declaración e Inicialización

Para declarar un arreglo se utilizan los corchetes “[]”, un par por cada dimensión.

- único arreglo

```
// solo la variable con los corchetes a su lado
// sera un arreglo del tipo de dato definido.
int arregloInt[] = new int[5], entero = 4;
```

- múltiples Arreglos

```
// todas variables son arreglos del tipo de dato definido.
char[][] arreglo1, arreglo2[], arreglon;
arreglo1 = new char[12][2];
// notamos que arreglo2 es un arreglo de arreglos
// de 2 dimensiones eso lo vuelve un arreglo de 3 dimesiones
arreglo2 = new char[3][2][3];
arreglo3 = new char[21][4];
```

3.5. Consultar

Para consultar los elementos de un arreglo se utilizan indices que comienzan en el 0 y van al tamaño del arreglo-1.

```
// un arreglo de tamaño 2
int[2] arreglo = new int[2];
// inicializariamos sus elementos de la siguiente manera:
// primera posición
arreglo[0] = 5;
// segunda posición
arreglo[1] = 87;
// podemos obtener e imprimir el valor
System.out.println(arreglo[0]);
System.out.println(arreglo[1]);
```

CAPÍTULO 4

Operadores

4.1. Aritmeticos

- **Signo +**
  - suma de 2 numeros.

```
int a = 3, b = 5;
int suma = a + b;
```

  - concatenación de 2 cadenas.

```
concatenacion = "Hola" + "mundo";
```

- **Signo -**
  - resta de 2 numeros.

```
resta = a - b;
```

- **Signo ´*´**
  - multiplicación de 2 numeros.

```
multiplicacion = a * b;
```

- **Signo /**
  - división de 2 numeros

```
division = a / b;
```

- **Signo %**
  - residuo de la división de 2 numeros

```
modulo2 = a%2;
```

4.2. De Asignación

- **Signo =**
  - asignar un valor o dirección de memoria.

```
int a = 5;
int arreglo[] = new int[5]
```

- **Signo +=**
  - acumulador mediante suma.

```
a += b
//equivalencia
a = a + b;
```

- **Signo -=**
  - acumulador mediante resta.

```
a -= b + c / d;
//equivalencia
a = a - (b + c / d);
```

- **Signo *=**
  - acumulador por multiplicación.

```
a *= b - c * d;
//equivalencia
a = a * (b - c * d);
```

- **Signo /=**
  - acumulador por división.

```
a /= b
//equivalencia
a = a / b;
```

- **Signo %=**
  - acumulador por modulo.

```
a %= b
//equivalencia
a = a % b;
```

4.3. Incremento y Decremento

4.3.1. Prefijo

- **++**

```
int i = 0;
// incrementar en 1
++i;
// equivalencia
i = i + 1;
```

- -

```
int i = 0;
// decrementar en 1
--i;
// equivalencia
i = i - 1;
```

4.3.2. Posfijo

- ++

```
int i = 0;
// incrementar en 1
i++;
// equivalencia
i = i + 1;
```

- -

```
int i = 0;
// decrementar en 1
i--;
// equivalencia
i = i - 1;
```

4.3.3. Diferencia

```
int i = 3;
System.out.println(i++); // se imprimirá 3 y luego se hará el incremento.
i = 5;
System.out.println(++i); // se hará el incremento primero, por lo que se imprimirá un 6.
```

4.4. Relacionales

Todo los operadores relacionales se utilizan de manera infija, es decir entre 2 operandos.

- **==**: igualdad.
- **!=**: desigualdad.
- **<**: menor que.
- **>**: mayor que.
- **<=**: menor o igual que.

- **>=**: menor o igual que.

los resultados de estos operadores son de tipo boolean. Cunado se cumplen regresan un **true** y en caso contrario un **false**

4.5. Logicos

Los operadores logicos **&&** y **||** son operadores infijos y nos sirven para trabajar con mas de una condición.

- **&&** (compuerta AND): solo si ambas condiciones a sus costados son **true**, retorna true.
- **||** (compuerta OR): mientras haya un **true** en cualquiera de sus extremos el resultado sera **true**.
- **!** (compuerta NOT): nega la condición que le sigue. es decir si es un **true** retorna un **false** y si es un **false** retorna un **true**.

CAPÍTULO 5

Control de Flujo

El flujo se controla a travez de condiciones y bucles.

5.1. If / Else

```
if(condicion){
    System.out.println("TRUE");
    if (condicion2){
        instrucciones2;
    }
}
else if(condicion2){
    System.out.println("FALSE");
}
else{
    instrucciones3;
}
```

La estructura **if / else** nos permite tomar decisiones, si la condición es verdares, en este caso se imprimirá “TRUE” y si es falsa se evaluara la condicion2, si esta es verdadera se imprimirá “FALSE”. Podemos aplicar esta lógica a cualquier decisión que queramos tomar.

Ademas se pueden anidar ya sea en el if o en el else.

5.2. Switch

Nos facilita generar un menú. pues nos evita el tener que escribir multiples if / else.

```
int estación = 8;
switch (mes) {
    case 1:
        System.out.println("Primavera");
        break;
    case 2:
        System.out.println("Verano");
        break;
    case 3:
        System.out.println("Otoño");
        break;
    case 4:
        System.out.println("Iniverno");
        break;
    default:
        System.out.println("Solo hay 4 estaciones");
        break;
}
```

5.3. While

Esta estructura es un bucle que mientras se cumpla la condición se ejecutara el código dentro del while.

```
int i = 0;
while(i < 5){
    instrucciones;
    i++;
}
```

En el ejemplo de arriba, las instrucciones se repetiran 5 veces. Si fuera una condición que siempre es verdadera este seria un ciclo infinito, en el caso contrario si la condición es falsa desde un principio entonces nunca se ejecutarian las instrucciones dentro del while.

5.4. Do While {#do while}

```
int i = 0;
do{
    instrucciones;
    i++;
}while(i < 5);
```

Este ciclo es muy similar al while, pero nos asegura que las instrucciones se ejecuten por lo menos una vez.

5.5. For

```
// sintaxis
for(inicialización; condición; paso){
instrucciones;
// ejemplo
for(int i=5; i > 0; i--){
System.out.println("cuenta regresiva:" + i);
}
```

Los ciclos **for** son para evitar tener multiples lineas cuando se tratan de condiciones que se cumplen por el paso del ciclo, mientras que los ciclos while se utilizan cuando se desconoce cuando se cumplira la condición.

En el ejemplo queda bastante claro cual es la estructura del ciclo **for**, primero va la inicialización que puede o no ser una declaración, la segunda es nuestra condición a evaluar y por ultimo es nuestro paso que podria ser ++, -, de n en n, i/e (logaritmico), etc.

5.6. Foreach

```
// sintaxis
for(variable : Iterador){
instrucciones;
}
int arreglo[] = {3,5,6,9,5};
for (int i : arreglo){
    System.out.println(i);
}
```

El ciclo foreach lo que nos ofrece es una herramienta para iterar sobre objetos de una manera mas sencilla y eficiente.

CAPÍTULO 6

Modificadores de Acceso

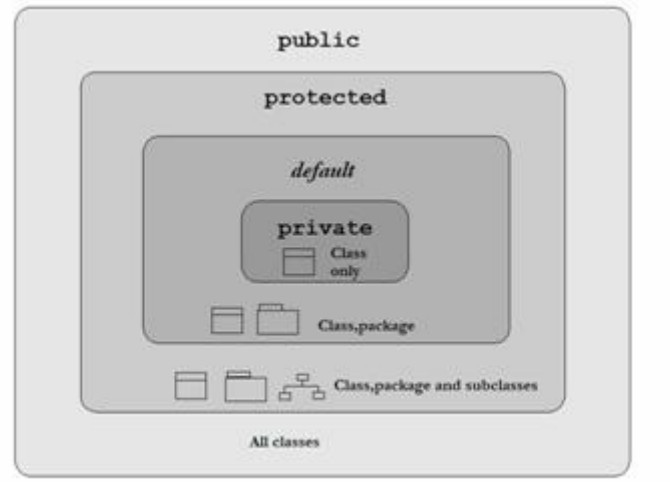

Figura 6.1: Modificadores de acceso

CAPÍTULO 7

Colecciones de datos

Se diferencian de los arreglos pues su tamaño no es fijo.

Tienenn operaciones para añadir, eliminar, obtener, encontrar o recorrer.

Todas las colecciones se encuentran en el paquete **java.utils**, se utiliza para almacenar colecciones de objetos, no se pueden coleccionar datos primitivos.

7.1. Estructura o Familia

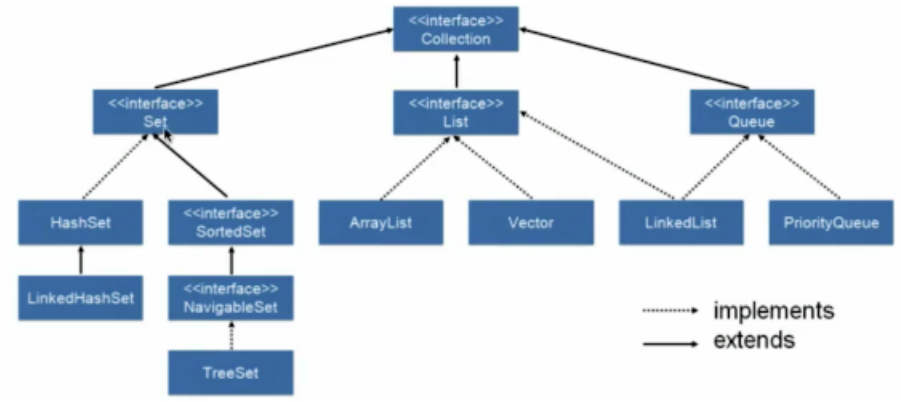

Figura 7.1: Arbol de implementación y herencia

7.2. Miembros

- **List** : Los elementos se van a encontrar ordenados y pueden repetirse. **Interfaz**.
    - **add(Object a)** : añade un objeto al final de la lista.
    - **add(índice i, Object a)** : añade un objeto en el índice i de la lista.

    - **get(índice i)** : devuelve el objeto en el índice i de la lista.
    - **remove(índice i)** : elimina el objeto en el índice i de la lista.
    - **clear()** : Vacía la lista.
    - **indecxOf(Object o)** : devuelve el índice de la primera coincidencia, de no encontrar el objeto devuelve -1.
    - **lastIndexOf(Object o)** : devuelve el índice de la última ocurrencia del objeto en la lista, de no encontrar el objeto devuelve -1.
    - **size()** : devuelve el numero de elementos de la lista.
    - **contains(Object o)** : devuelve un booleano dependiendo si existe el objeto en la lista o no.

- **ArrayList** : es un arreglo de objetos de tamaño dinámico. **Clase**.
- **Vector** : es similar a un ArrayList, la diferencia es que utiliza hilos y esta sincronizado. **Clase**.
- **Set** : Los elemnetos pueden o no estar ordenados y cada elemento es único.**Interfaz**.