Santiago Altafaj Iglesias

Este documento no pasa de ser un manual muy básico de programación en JAVA con las últimas incorporaciones de funcionalidades de las versiones 8 y 11

JAVA 8-11

Resumen curso UDEMY

# [Tipos de Datos](#Datos)

# [Estruturas de Control de Flujo](#Flujos)

# [POO](#POO)

# [Arrays y Colecciones](#Arrays)

# [Modularidad](#Modularidad)

# [Lambdas y Streams](#Lambdas)

# [Entrada-Salida y Bases de Datos](#io)

# [Control de Excepciones](#Excepciones)

# [Multitarea](#Multitarea)

# [Anotaciones,Localización y Seguridad](#Seguridad)

## Tipos de Datos

### Declarar e inicializar variables

Identificadores: se pueden utilizar cualquier combinación de letras, números,$ y \_ pero no pueden comenzar por carácter numérico, ni utilizar palabras reservadas

Las variables que pueden compartirse con todos los métodos se les conoce como atributos o campos; las variables de un método se les conoce como locales y solo son visibles a ese método, las variables locales no se inicializan por defecto, es necesario asignarles un valor para utilizarlas, las variables atributo se inicializan por defecto.

En los tipos primitivos la variable con tine al dato; los tipo objeto la variable contine una referencia la objeto(instanciar el objeto)

Los tipos primitivos cada varibles tine un dato o una copia del dato

En los tipo objeto cada variable (instancia) si igualmos dos variables apuntaran al mismo objeto

### Tipos primitivos y tipos objeto

Tipos primitivos: números, datos lógicos y carácter

Tipos objetos:cualquier objeto java

Si queremos que un int sea un long añadimos (l) al final del dato

Si queremos que in int sea un doublé o floata añadimos (f) al final del dato

Todos los tipos primitivos son convertibles en otros, salvo boolean

Las converisones pueden ser implícitas entonces el destino tiene que ser igual o mayor tamaño que el de origen

Int x =45;

Doublé n=x;

O explicitas

Doublé r= 4.5f;

Int x=(int)r; //casting

Los tipos objeto son todos los objetos de cualquier clase java, se manejan a través de variables de su tipo(clase), la variable contiene una referencia al objeto, mediante la variable se accede a los métodos del objeto

String s = new String(“hola”);

s.lengtn();

no se puede hacer conversión ni implícita ni explicita entre tipos primitivos y objetos

### Ciclo de vida de los objetos

Creación de un objeto:

Clase 1 c=new Clase 1();

String s = new String(“hola”);

Object ob=new Object();

Un objeto ose crea a partir del opereador new, seguido del nombre de la clase y se devuelve una referencia la objeto que se guarda en una variable de clase.

Constructores :

Se ejecutan durante la creación del objeto, pueden sobrecargarse (varios constructores con distintos parámetros)

### Clases envoltorio

Encapsulan tipos primitivos como objetos:

Integer integ= new Integer(200);

Double db =new Double(30.4);

Tienen métodos propios:

Int k=integ.intValue();

Doublé d=db.doubleValue();

Métodos estáticos para convertir String en tipo primitivo u objeto:

Int p=Integer.parseInt(“300”);

Int n =Integer.paresInt(“100011”,2); //35

Integer num=Integer.ValueOf(“100011,2);

Autoboxing/unboxing:

Se puede asignar directamente el tipo primitivo a la variable objeto(autoboxing):

Integer en =200; //autoboxing

Doublé db=45.5; //autoboxing

Se puede recuperar el tipo primitivo asignando directamente la varible objeto a lal variable primitiva(unboxing):

Int n =ent; //unboxing

Integer k=30; //autoboxing

K++; //unboxing más autoboxing

Los objetos de las clases de envoltorio son inmutables, no se pueden modificar

Integer ent=200; //autoboxing

Ent =ent +100; //genera un nuevo objeto, unboxing+autoboxing

### Inferencia de tipos

Característica incorporada en java 10, consiste en declarar variables locales sin indicar explícitamente el tipo.

Se emplea la palabrea VAR

Var num=100; //entero

Var datos =new ArrayList<Integer>();ArrayList de enteros

El tipo es inferido por el compiladore a partir del valor asignado a la variable

Simplifica la escritura de código, ni mejora ni empeora el rendimiento de la aplicación

Únicamente puede utilizarse con variables locales(en métodos).

Es obligatorio asingar explícitamente un valor a la variable, valor que no puede ser null

Var data; //error de compilación

Var n =null;//error de compilación

No es posible utilizar inferencia de tipos en declaraciones multiples

Var a,c=10; // incorrecto

Var b=4,x=30; //incorrecto

Se puede utilizar inferencia de tipos en buclels de tipo for:

For(var i=0;i<10;i++){

}

For(var s:datos){

}

En Arrays, no puede utilizarse con inicialización abreviada

Var s={1,2,3,4}; //incorrecto

Var d =new int[]{1,2,3,4}; //correcto

### Operadores JAVA

Aritméticos, se emplean con tipos numéricos primitivos para realizar operaciones aritméticas en un programa:+,-,\*,/,%,++,--

Los operadores ++ y – - se aplican con tipos numéricos y pueden ir delante o detrás de la variable:

Int a =3,b;

b=a++; //b toma el valor 3 (primero asigna y luego incrementa )

b=++a; //b toma el valor 4

el operador = Asigna el resultado de una expresión a una variable

los operadores +=,-=,/=,%=, realizan la operación entre un dato y la variable y signan el resultado a la variable

int a =3;

a+=10; //equivale a a=a+10

byte b=10;

b+=5; //ojo, equivale a b=(byte)(b+5);

b=b+5;(esto provoca error de compilación pues una operación con int(los literales enteros son int)siempre da como resultado int.

Los condicionales evalúan dos operendos y dan como resultado un valor boolean;

<,>,<=,>=,==,!=,

Salvo ==, que puede utilizarse con objetos, los demás solo se pueden emplear con tipos primitivos, y entre tipos compatibles.

Int a=3;

Doublé c=9.7;

Boolean x=false;

If (a>c); //ok

Ifa>x); //error de compilación

Lógicos evalúan expresiones de tipo boolean:

&&,||,!

Los operadores && y || funcionan en modo cortocircuito si la primera condición no se cumple ya no se evalúan las siguientes

Otros :

New .Creacion de objetos a partir de la clase

ArrayList lista = new ArrayList();

Instanceof. Comprueba si un objeto es de untipo dado

String s =”hello”;

//muestra true

System.out.println(s instanceof String);

### Igualdad de objetos

Operador ==

Se utiliza para comprobar la igualdad con tipos primitivos

En variables de tipo objeto compara referencias, no los objetos

String n1= new String(“cadena”);

String n2=new String(“cadena”);

//el resultado es falso

If (n1==n2){

}

Al apuntar a objetos diferentes , las referencias son diferentes

Igualdad de Strings, para comparar dos cadenas de caracteres utilizamos el método equals();

String n1= new String(“cadena”);

String n2=new String(“cadena”);

//el resultado es verdadero

If(n1.equals(n2)){

}

El método equals distingue mayúsculas y minúsculas, para ignorar la diferencia, utilizamos equalsIgnoreCase();

Igualdad objetos de envoltorio

Diferentes objetos

Integer int1=new Integer(20);

Integer int2= new Integer(20);

//falso

If(in t1==int n2){

}

Mismo objeto

Integer int1=20;

Integer int2=20;

//verdadero

If(int1==int2){

}

Se puede utilizar el método equals() para comparar los valores envueltos por el objeto

Iguladad de StringBuilder

Representa cadenas mutables(modificables)

No sobreescribe equals(), por lo que == y equals() producen el mismo efecto. Solo verdadero cuando apuentan al mismo objeto

StringBuilder n1 =new StringBuilder(“cadena”);

StringBuilder n2 =new StringBuilder(“cadena”);

StringBuilder n3=n2;

If(n1==n2){ //el resultado es falso

}

If(n1.equals(n2)){ //el resulado es igualmente falso

}

If(n2==n3){ //el resultado es verdadero

}

Inmutabilidad de objetos String

Un objeto String representa una cadena de caracteres inmutable, es decir, no se puede modificar

En la concatenación, no se modifica ningún objeto existente, se crea uno nuevo.

## Estruturas de Control de Flujo

### Operador IF y TERNARIO

If comprueba una condición de tipo boolean y ejecuta un bloque de sentencias si se cumple .Opcionalmente, se puede indicar bloque else.

If(condición){

Sentencias;

}esle(opcional){

sentencias

}

Si la condición no se boolean se produce un error de compilación

Las llaves solo son obligatorias se el bloque contiene más de una instrucción

El operador ternario es una forma abreviada de la instrucción if

Variable=expresión?valor\_si:valor\_no;

Evalúa una expresión de tipo boolean, si el resultado es verdadero, se devuelve el valor indicado después de la interrogación, si no, devolverá el valor a continuación de los dos puntos;

Int a=3,b=5,c;

C=(a>b)?a\*a:b--;

//c vale 5,pues primero se asigna b a c y luego se decrementa la variable

System.out.println(c);//5

### Uso de SWITCH

Evalúa una expresión cuyo resultado debe ser entero(int).Se ejecutarán diferentes loques de sentencias en función de los posibles resultados

Switch(expresión){ switch(a\*2){

Case valor1: case 0:

//sentencias system.out.println(“nada”);

Break; break;

Case valor2: case4:

//sentencias System.out.prinln(“cerca”);

Break; break;

Default case 8:

//sentencias System.out.prinln(“acierto”);

} break;

Default:

System.out.println(“error”);

}

Si el resultado de la expresión coincide con uno de los valores indicados en los case, ejecutará el bloque correspondiente de sentencias,si no, entrará en el bloque default(opcional).

La instrucción break al final de cada bloque case es opcional. Si no se indica, el programa entrará en el siguiente bloque

Int a=10;

Switch(a){

Case 10:

System.out.println(“es 10”);

Default:

Sytem.out.prinln(“sin valor”);

}

El programa muestra:

(Es 10

Sin valor)

Los valores de los case deben ser literales o constantes enteras int,o convertibles implícitamente en int

Int p=5;

Final int k=30;

Int n=3;

Switch(p){

Case 10: // ok, es un literal int

Case k : // ok, es una constant

Case p: //error de compilación, no es una constante. Ojo si fuese 5 si valdría aquí utilizamos el nombre de la variable

Case ‘@’: // ok, char convertible implícitamente a int

El bloque default es opcional y no tiene que aparecer necesariamente al final

Int p=5;

Switch(p){

Case 10:

System.out.println(“es 10”);

Default:

System.out.pirntln(“default”);

Case2:

System.out.prinln(“es 2”);

}

El programa muestra

(default

Es 2) porque no hemos puesto break después del case default

Desde java 7 es posible evaluar en un switch expresiones cuyo resultado sea una cadean de caracteres(String)

Si la expresión es evaluada como string, no se admiten valores enteros en los case y viceversa

Uso de FOR y WHILE

La instrucción for ejecuta un grupo de instrucciones un número determinado de veces

For(inicialización;condición;incremento){

//instrucciones

}

Desde la variable toma el valor de inicialización y hasta que la condición deje de cumplirse, se ejecutará el bloque de instrucciones

For(int i=1;i<10;i++){

System.out.println(i);

}

Las llaves son obligatorias si hay más de una instrucción.

Las tres instrucciones de control son opcionales.

Int i=1;

For(;i<10;) // los ;se ponen igualmente

System.out.println(i);

I++;

}

Las instrucciones de control pueden contener más de una senencia,separadas por una coma:

For(int a=0, b=10;a<b;a++,b--){

//sentencias

}

La instrucción enhanced for o for extendido se utiliza para el recorrido, en modo lectura , de arrays y colecciones.

For (tipo variable:array){

//instrucciones

}

La variable de control va pasando por todas las posiciones del array,NO es un índice

Int[] nums={4,5,1,8};

For (int n:nums){

System.out.println(n); //n toma los valores(4,5,1,8) y los imprime

}

La instrucción WHILE ejecuta un grupo de instrucciones mientras se cumpla una condición(resultado sea true)

La condición puede evaluarse al principio, o después de ejecutar el bloque de instrucciones:

While(condición){ do{

//instrucciones //instrucciones

} }while(condición);

Las acciones dentro del bloque provocarán que en algún momento la condición deje de cumplirse, sino bucle infinito

Int n=0;s=0; int n=0;

While(s<1000){ do{

S+=n++; n=leerNumero();

} }while(n<0);

### Uso de BREAK y CONTINUE

Provocan una salida forzada.

--break. Abandona el bucle

--continue. Pasa a la siguiente iteración

Ambas instrucciones pueden utilizarse tanto en for como en while

La instrucción break provoca la salida de la instrucción repetitiva, pasando el control del programa a la siguiente instrucción

Int n=leerNumero();

Int s=0;

For(int I =1;i<n;i++){

S+=I;

If(s>100){

Break;

}

}

La instrucción continue pasa a la sigiente Iteracción del bucle. En el caso de un for, la llamada a continue noos llevaría directamente a la instrucción de incremento:

//muestra los números del 1 al 10,

//menos el 5

For(int i=1;i<10;i++){

If(i==5)

Continue;

System.out.println(i);

}

Bucles etiquetados, es possible asignar una etiqueta a una instrucción for o while

En bucles anidados, permite a las instrucciones break o continue indicar el bucle que se quiere abandonar:

Externo: for(…){

Wile(…){

Break externo; //sale del bucle principal

}

}

Si no se indica etiqueta despúes de break/continue, afectará al bucle más interno

Ejemplo práctico

Public class TestBuclesAnidados{

Public static void main(String[] args){

Int a=0,s=0,i=1;

Principal:

For(;i<10;i++){

While(a<5){

A=i++;

S=a+I;

If(s>=10){

Break principal;

}

}

}

System.out.println(i+”:”+a+”:”+s); //el resultado es 6:5:11

}

}

## POO

### Creación de métodos y sobrecarga

Definición y estructura:

Un método es una función que realiza una tarea. Puede recibir parámetros y de volver un resultado

Estructura:

Modificador tipo\_devolución nombre\_metodo(parámetros){

}

Ejemplo:

Public(modificador) int(tipo\_devolución) sumar(nombre\_metodo) (int a, intb) (parmetros){

Int s=a+b(tarea);

Return s (devolución);

}

Public void (los métodos que no devuelven resultado(void)no necesitan return) imprimir (int a){

System.out.println(a);

}

Llamada a método:

Los métodos se definen dentro de clases, para llamarlos se debe crear un objeto de la clase (instancia) y utilizar la sintaxis:

Objeto(instancia).metodo(argumentos);

Ejemplo:

Class Calc{

Public int sumar(int a ,int b){

Int s=a+b;

Return s; //devolución

Class Test{

Public static void main (String[] arg){

Clac cl = new Calc() (instancia o creación del objeto con el constructor por defecto);

Int s =cl.sumar(8 , 3) (los parámetros son del tipo definido en el método, y se guarda el resultado en una varible con el tipo definido en el método);

}

Sobrecarga de métodos

Una clase puede contener varios métodos con el mismo nombre, pero deben diferenciarse en el número o tipo de parámetros:

Public int sumar(int a, int b){…}

Public int sumar(int a){…}

Public int sumar(long b){…}

El nombre de los métodos suele ser un verbo.

El tipo de devolución no afecta en la sobrecarga, puede ser el mismo o diferente

Llamadas a métodos sobrecargados:

La versión del método que será llamado se determina en función de los argumentos de la llamada

Sumar(3,9)……..public int sumar (int a , int b){….}

Sumar(10)……….public int sumar(int a){…..}

Sumar(7L)……….public int sumar(long b){….}

Si tenemos dos métodos con el mismo nombre y mismos parámetros se produce un error de compilación

Si tenemos dos métodos con el mismo nombre pero alguna de sus letras está en mayúsculas no hay error pero tampoco hay sobrecarga pues son distintos.

Precaución

Cuando hay varios métodos que se pueden ejecutar en una llamada: primero se intenta coincidencia exacta, después promoción de tipos y en último lugar autoboxing:

Método(4)…………..void método(int a);

Void método(Integer e);

Método(4)…………..void método(long a);

void método(Iteger e);

Método(4)……………void método(Long a);

Void método(integer e);

### Paso de parámetros

Paso de tipos primitivos:

Al pasar un tipo primitivo a un método, estamos pasando una copia del dato.

Si el método modifica su copia, no afecta al original

Class Test{

Public static void main(String[] arg){

Calc cl =new Calc();

Int n=5;

Cl.modif(n);

System.out.println(n); //devuelve 5

}

Class Calc{

Public void modif(int a){

A=a+3;}

}

}

N=5 -------🡪a=5 (se recibe una copia del dato en a)// n=5--------🡪a=8(después del método se modifica a pero se imprime n) si imprimiéramos a seria 8)

Paso de tipos objeto

Al pasar un tipo objeto a un método, estamos pasando una copia de la referencia al objeto

Class Test{

Public static void main(String[] arg){

Calc cl =new Calc();

StringBuider sb=new StringBuilder(“hello”);

cl.modif(sb);

system.out.println(sb.toString()); //hello bye

class Calc{

public void modif(StringBuilder d){

d.append(“ bye”);

}

}

Las variables sb y d apuntan al mismo objeto porque son de tipo String Builder si cambiamos StringBuilder por String no sería igual porque son inmutables; aunque inicialmente apuntan al mismo objeto;tras la concatenación se genera un nuevo objeto String al que apunta la variable parámetro; sb=”hello” y d= “hello bye”.

### Miembros estáticos

Métodos estáticos

Son métodos que no están asociados a ningún objeto(instancia) particular de la clase

Se declaran con la palabra “static”:

Class Calc{

Public static int cuadrado(int a){

Return a\*a;

}

}

No es necesario crear un objeto para llamar a estos métodos, se utiliza el nombre de la clase:

Int r =Calc.cuadrado(4)(el resultado del método se guarda en la variable r)

Aunque es la forma habitual de usarlos, también se les puede llamar con cualquier instancia de la clase.

Cosideraciones métodos estáticos.

Solo pueden llamar a otros miembros de su misma clase que también sean static

Class Test{

Int a=2;

Static int b=5;

Public static int método(){

Int c=a\*3; //error de compilación

Int n=b+1; //ok

Imprime(n); //ok

}

Static void imprime(int s){

System.out.printl(s);

}

No se puede usar en su interior ni “this” ni “super”

Atributos estáticos(variables estáticas):

Son compatibles por todos los objetos de la clase.

Se definen con la palabra “static”

Class Test{

Static int n=0;

Public void inc(){

N++;

}

Public int getN(){

Return n;

}

Class Prueba{

Public static void main (String[]arg){

Test t1=new Test();

T1.inc(); // n=1

Test t2=new Test();

T2.inc(); //n=2

System.out.println(t1.getN()); //devuelve 2

System.out.println(t2.getN()); //devuelve 2

Bloques estáticos:

Se ejecutan una vez durante la vida de una clase.

Solo pueden acceder a otros miembros estáticos:

Class Test{

Static int n=0;

Static{

N++;

}

Public int getN(){

Return n;

}

Class Prueba{

Public static void main(String[] arg){

Test t1 = new Test();

Test t2= new Test();

System.out.println(t1.getN()); //1

System.out.println(t2.getN()); //1

En este caso el bloque estatico solo se ejecuta al instanciar la clase con t1, con t2 ya no se ejecuta.

### Creación e inicialización de instancias

### Modificadores de acceso

### Encapsulación

### Herencia

### Constructores en la herencia

### Sobreescritura de métodos

### Tipo de objeto y tipo de referencia

### Clases abstractas y polimorfismo

### Interfaces

### Interfaces JAVA 8 Y 9

### Interfaces funcionales

### Clases anidadas

### Enumeraciones

## Arrays y Colecciones

### Arrays de una dimensión

### Arrays multidimensionales

### Genéricos

### Iterables y colecciones

### Listas

### Tablas

### Conjuntos

### Colas dobles

### Ordenación Arrays y listas I

### Ordenación Arrays y listas II

## Modularidad

### Conceptos básicos de modularidad

### Compilación y ejecución de aplicaciones modulares

### Empaquetado de modulos

### Módulos anónimos y automáticos

### Otras características de la modularidad

## Lambdas y Streams

### Expresiones LAMBDAS e INTERFACES FUNCIONALES

### Interfaces java.util.Function

### Introducción a STREAMS

### Métodos de STREAMS I

### Métodos de STREAMS II

### Métodos de STREAMS III

### Métodos de STREAMS IV

### Métodos de STREAMS V

### Referencias a métodos

### STREAMS paralelos

## Entrada-Salida y Bases de Datos

## Control de Excepciones

## Multitarea

## Anotaciones,localización y seguridad