Santiago Altafaj Iglesias

[Capte la atención del lector con un resumen atractivo. Este resumen es una breve descripción del documento. Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

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

### Uso de SWITCH

### Uso de FOR y WHILE

### Uso de BREAK y CONTINUE

## POO

### Creación de métodos y sobrecarga

### Paso de parámetros

### Miembros estáticos

### 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