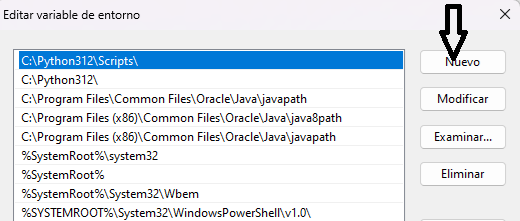
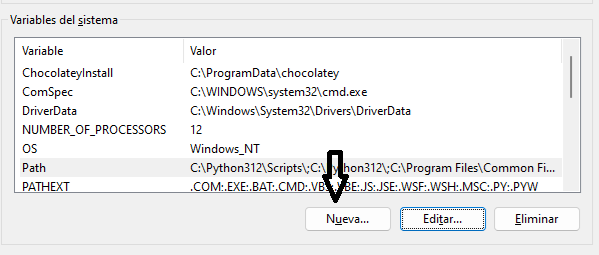
[gchaldu (Gabriel Chaldu)](https://github.com/gchaldu?tab=repositories) repositorio de github

descarge de la pagina [Java Downloads | Oracle Colombia](https://www.oracle.com/co/java/technologies/downloads/#jdk23-windows) el jkd2301 para window en zip

* fui a la pagina de descargas y lo descomprimí
* luego llega la carpeta jdk-23.0.1 a el disco duro C/java
* tenemos que ir a variables de entorno y decirle al sistema operativo donde está ubicado mi JDK
* En el buscador Imagen que contiene Texto

  Descripción generada automáticamente escribimos variables y vamos a la pestaña (editar las variables de entorno del sistema)
* Entramos a variables de entorno
* Vamos a ver las variables del sistema
* Doble click en (path) y debemos agregar la ubicación de nuestro jdk incluyendo la carpeta bin
* Buscamos donde esta nuestro jdk-23.0.1 y la carpeta bin, copiamos la ubicación
* En la ventana de las variables de entorno 
* En nuevo, y agregamos nuestra ubicación, en mi caso fue (C:\java\jdk-23.0.1\bin)
* Ahora en la ventana de variables del sistema vamos a agregar una nueva
* 
* Escribimos en el cuadro de texto : nombre de variable JAVA-HOME, en valor de la variable: (la ruta de jdk) C:\java\jdk-23.0.1
* Aceptamos en las dos ventanas
* Verificamos que este bien instalado
* Abrimos powershell o la consola

PS C:\Users\USER> java -version

java version "21.0.2" 2024-01-16 LTS

Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)

Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)

PS C:\Users\USER> javac -version

javac 21.0.2

PS C:\Users\USER>

:13:38 video 1 JAVA

Como me apareció la versión 21.0.2 entre a (agregar o quitar programas ) y desinstales todas la versiones de java instalada y realice la comprobación de nuevo

PS C:\Users\USER> java -version

java version "23.0.1" 2024-10-15

Java(TM) SE Runtime Environment (build 23.0.1+11-39)

Java HotSpot(TM) 64-Bit Server VM (build 23.0.1+11-39, mixed mode, sharing)

PS C:\Users\USER> javac -version

javac 23.0.1

PS C:\Users\USER>

Ahora si aparece la que se instaló la jkd-23.0.1

Video 2 7:01

Crear nuestro primer proyecto con INTELLIJ IDEA

1. Entramos al programa
2. Nuevo proyecto
3. Nombre del proyecto (HolaJava)
4. Ubicación c:/java/proyectos
5. Build system: intellij
6. En el JDK debe estar seleccionado el que instalamos jkd-23.0.1
7. Las otras casillas se dejan seleccionadas como se encuentran por defecto
8. Todo lo que crea el programa son archivos de configuración y librerías de JDK nosotros vamos a trabajar solo con lo que esta en SRC
9. Siempre que creamos un proyecto hay una clase principal (public class Main) y un método principal que es (public static void main) no se puede cambiar el nombre de main pues no se podría ejecutar el programa
10. Se ejecuta todo lo que esta dentro del método main

Video 3

Variables

Es una posición en memoria donde voy a almacenar un dato, en java como es tipado se debe definir el tipo de dato que se va a almacenar.

Tipodato nombreVariable;

Dependiendo el tipo de variable son las operaciones que se pueden hacer con ella

Categorías:

Primitivos y Referncia

Primitivos: contiene un solo valor como entero, flotante, carácter, booleanos

Referencia: son los arreglos las clases y las interfaces.

Texto

Descripción generada automáticamente

Primitivos

* boolean

boolean a= false;

boolean b= true

solo puede almacenar esos do valores

* char: un dato unicode

char a=’a’;

char b= ‘1’;

char c = ‘\u0021’;

* entero

Texto

Descripción generada automáticamente

* float

números con coma flotante

Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

La clase equivalente es el tipo correspondiente por referencia a ese tipo de variable primitivo y al declararlo se pueden utilizar diferentes métodos

La variables no pueden utilizar los siguientes nombres, por ser palabras reservadas del lenguaje

Tabla

Descripción generada automáticamente

Sout= system out println()

Nuestro ejemplo quedaría así

public class Main {  
 public static void main(String[] args) {  
  
 String mensaje= "Hola";  
 Main m= new Main();  
 m.Saludar(mensaje.toUpperCase());  
 int numero=2;  
 Integer numero2 = 5;  
 System.*out*.println("el numero de referencia es; "+numero2+"\n"+"el numero es: "+numero);  
  
 }  
 public void Saludar(String mensaje)  
 {  
 System.*out*.println(mensaje);  
 }  
}

resultado

HOLA

el numero de referencia es; 5

el numero es: 2

el salto de linea lo da \n

de la versión 10 en adelante se pueden declarar las variables con (var) y automática mente según el dato que se le de asigna el tipo de variable

var numero=2; asinga a numero que es un int

var numero = “Juan” asigna a numero que es un String

var numero3='\u0012'; asigna un char

Primitivos

Números Enteros

Pueden ser positivos y negativos Z = {..-2,-1,0,1,2…}

Byte: hasta 127

Short

Int

Long

“Método abreviado (soutv) = system out prinln y muestra nombre de variables para imprimir y las variables”

System.*out*.println("Cuantos BYTES tiene un Byte: "+Byte.*BYTES*);

Esta linea me imprime la cantidad de Bytes del tipo de dato Byte= 1

19-38 video 4

Definir las variables con (var) no es recomendable pues asigna muchos recursos y hace que la aplicación utilice demasiada memoria en una variable

Entrada de datos al sistema VIDEO 5

Creamos un nuevo proyecto ScannerEjemplo con su clase main y su public static void

Luego vamos a la carpeta src (source)damo click derecho y creamos un paquete, le asignamos nombre del ejemplo com.youtics.scanner, esto crea una ruta,

Ahora en esa ubicación creamos una clase (MainScanner) y dentro de esta clase creamos el main (main.) como ya tenemos el main en nuestra clase se puede eliminar el main que estaba por defecto (click derecho delete)

Dentro del mismo paquete creamo una nueva clase (ScannerClass), creamos los atributos todos privados

Luego creamos un método publico, cuando llamamos el método Scanner, automáticamente importa la librería java.util.Scanner;

package com.youtics.scanner;  
  
import java.util.Scanner;  
  
public class ScannerClass {  
 private int numero;  
 private String username;  
 private Character letra;  
  
 public void ScannerNumber(){  
 System.*out*.println("Ingrese un numero...");  
 Scanner scanner = new Scanner(System.*in*);  
 this.numero= scanner.nextInt();  
 }  
  
}

ahora vamos a crear un getter para retornar informacion, click derecho después del método ScannerNumber y seleccionamos generate y después getter de numero queda asi:

public int getNumero() {  
 return numero;  
}

vamos al método principal o mainscanner

vamos a instanciar la clase

public class MainScanner {  
 public static void main(String[] args) {  
 ScannerClass scannerClass = new ScannerClass();  
 }  
}

al instanciar la clse ScannerClass pero no tenemos ese método constructor en nuestra clase java sabe que por defecto hay un constructor vacío.

Asi que no vamos a utilizar constructor en nuestra clase ScannerClass

Ahora publicamos el resultado por medio de método getter

System.*out*.println("id de usuario = " + scannerClass.getNumero());

Realizamos lo mismo para crear un método para solicitar el nombre de usuario.

16:30 video 5

Para solicitar por el método scanner un carácter se utiliza esta linea pues no existe (nextChar) :” this.letra= scanner.next().charAt(0);” se le esta diciendo que devuelva un dato de tipo char en la posición 0

El método para solicitar el dato y el getter son iguales a los dos anteriores, solo cambia la linea anterior

Es buena practica cuando se utiliza un objeto de tipo scanner cuando finalicemos su uso cerrar ese objeto. Pero no se puede finalizar en cada método porque eso traería problemas de funcionamiento, lo que normalmente se hace es trabajar el método scanner con una propiedad o atributo de la clase, tenemos nuestra scannerClass asi:

package com.youtics.scanner;  
  
import java.util.Scanner;  
  
public class ScannerClass {  
 private int numero;  
 private String username;  
 private Character letra;  
  
  
  
 public void ScannerNumber(){  
 System.*out*.println("Ingrese un numero...");  
 Scanner scanner = new Scanner(System.*in*);  
 this.numero= scanner.nextInt();  
 }  
  
 public int getNumero() {  
 return numero;  
 }  
 public void ScannerName(){  
 System.*out*.println("Ingrese su nombre de usuario...");  
 Scanner scanner = new Scanner(System.*in*);  
 this.username= scanner.nextLine();  
 }  
  
 public String getUsername() {  
 return username;  
 }  
 public void ScannerChar(){  
 System.*out*.println("Ingrese una letra...");  
 Scanner scanner = new Scanner(System.*in*);  
 this.letra= scanner.next().charAt(0);  
 }  
  
 public Character getLetra() {  
 return letra;  
 }  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

vamos a cambiar algunas partes para mejor el codigo, empezando por el método Scanner

atributo

private Scanner scanner;

y en cada método se llama este:

scanner = new Scanner(System.*in*);

creamos un getter para ese atribuo

se llama desde el main para finalizar o cerrar el metodo scanner

scannerClass.getScanner().close();//con esta linea se cierra el metodo scanner

tambien vemos que la linea : scanner=new Scanner(System.in) se repite en cada método, esto se mejora creando un constructor vacío y allí inicializamos ese método, al realizar esto y lo ejecutamos vemos un error que se ingresa e numero y se salta a solicitar la letra, esto pasa pues al crear metodo contructor y llamara a nextInt ingresa un espacio que se ingresa como si fuera otro carácter, esto se corrige llamando al método escanner y realizando un next(line) antes de llamar el método para ingresar el nombre

scannerClass.getScanner().nextLine();

esta linea absorve el dato que se crea por error,

nuestro main queda:

package com.youtics.scanner;  
  
public class MainScanner {  
  
 public static void main(String[] args) {  
 ScannerClass scannerClass = new ScannerClass();  
 scannerClass.ScannerNumber();  
 System.*out*.println("id de usuario = " + scannerClass.getNumero());  
 scannerClass.getScanner().nextLine();  
 scannerClass.ScannerName();  
 System.*out*.println("nombre de usuario = " + scannerClass.getUsername());  
 scannerClass.ScannerChar();  
 System.*out*.println("scannerClass.getLetra() = " + scannerClass.getLetra());  
 scannerClass.getScanner().close();  
 }  
}

y nuestra clase:

package com.youtics.scanner;  
  
import java.util.Scanner;  
  
public class ScannerClass {//parametros  
 private int numero;  
 private String username;  
 private Character letra;  
 private Scanner scanner;  
  
  
 public ScannerClass() {//constructor  
 scanner= new Scanner(System.*in*);  
 }  
  
 public Scanner getScanner() {//getter  
 return scanner;  
 }  
  
  
 public void ScannerNumber(){//metodo para escanear numero  
 System.*out*.println("Ingrese un numero...");  
 this.numero= scanner.nextInt();  
 }  
  
 public int getNumero() {//getter de numero  
 return numero;  
 }  
 public void ScannerName(){//metodo para escanear nombre  
 System.*out*.println("Ingrese su nombre de usuario...");  
 this.username= scanner.nextLine();  
 }  
  
 public String getUsername() {//getter de nombre  
 return username;  
 }  
 public void ScannerChar(){//metodo para ingreso o escanear caracter  
 System.*out*.println("Ingrese una letra...");  
 this.letra= scanner.next().charAt(0);  
 }  
  
 public Character getLetra() {//getter de caracter  
 return letra;  
 }  
  
}

video 6

TIPOS DE DATOS COMO CASTEAR (CASTING)

Castear es transformar datos que tenemos por ejemplo como un string a un number

int num = Integer.*parseInt*(variable-string);

cambiar string a double

double real = Double.*parseDouble*(numStr);

y de la misma forma se hace con los otro datos primitivos, siempre utilizando el wrapper o el dato envolvente que es para int es Integer para double es Double

ahora vamos a pasar de integer a string

//pasar de integer a string  
Integer numInteger = 25;//el programa dice que debemos utilizar el primitivo que es int no el wrap que es el envolvente  
String datoNum = Integer.*toString*(numInteger);  
System.*out*.println("datoNum = " + datoNum);  
  
//string a caracter  
  
String caracter = "AB";  
char cart = caracter.charAt(0);//si cambiamos el 0 por el 1 carga el segundo carácter seria B  
System.*out*.println("cart = " + cart);

15:19

int i =2000000000;//debemos tener cuidado con los rangos pues esto nos  
//puede ocasionar errores como este  
System.*out*.println("i = " + i);  
Short s = (short)i;//el wrap Short y el primitivo short  
System.*out*.println("s = " + s);//nos muestra es -27648 pues es un valor  
//dentro del rango de los short pore eso al castiar debemos tener cuidado  
//con los rangos los short van de -32767 a 32768  
System.*out*.println(s.*MAX\_VALUE*); 32767

System.*out*.println(s.*MIN\_VALUE*); -32768

Por eso si vamos a cambiar un integer a short debemos estar seguros de que esta dentro del rango

ARRAY (ARREGLOS)

Son tipos de datos de referencia que contiene varios elementos ordenados, una colección

Sus elementos o valores están asociados a una llave o índice y puede contener tipos de referencia o primitivos

Pero siempre asociado a un solo tipo de datos

Siempre el primer dato comienza en el índice 0

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Siempre se le tiene que decir cuantos datos se va a almacenar

Length: total elementos que tiene=10 para el ejemplo anterior

Índices del 0 al 9 y tamaño 10 o length

Como se crea un array

Int[] lista = new int[10]; array lista con 10 elementos

Para mostrar el elemento

System.*out*.println("lista[0] = " + lista[0]);// muestra el primer elemento

System.*out*.println("lista[1] = " + lista[1]);// muestra el segundo elemento

Ahora para recorrer el array se puede con un for

Ejemplo con for

for (int i = 0; i < lista.length ; i++) {  
 System.*out*.println("lista " + i+"= "+ lista[i]);  
}

resultado

lista 0= 20

lista 1= 10

lista 2= 0

lista 3= 0

lista 4= 0

lista 5= 0

lista 6= 0

lista 7= 0

lista 8= 0

lista 9= 0

ejemplo con for each

for (int numero:lista){//*todo lo que esta en la lista lo carga en numero* System.*out*.println("numero = " + numero);  
}

en java no se escribe for each solo for y la forma de la expresión es la que se asemeja a otros lenguajes

otra forma de crear un array

String[] palabras = {"pepe","Carlos","Maria","Ana","Julio"};//de esta manera se define tambien el array, la cantidad de elementos que se incluyan será el length definido para el mismo  
for (String palabra:palabras){  
 System.*out*.println("palabra = " + palabra);  
}

14:36 video 7

Instanceof: nos sirve para verificar el tipo de dato

String nombre = "Gaby";  
if(nombre instanceof String){  
 System.*out*.println("es un String");  
}else{  
 System.*out*.println("No es un String");  
}

Para un int

If (numero instanceof Integer)//mirar si lo que tengo en numero es un Int

OPERADORES TERNARIOS

operador ternario es un if pero mas simplificado

String mensaje = (numero2 instanceof Integer)? "Es un numero": "No es un numero";  
System.*out*.println("mensaje = " + mensaje);

mensaje = (numero2==20)? "numero2 igual que 20":"numero2 diferente a 20";  
System.*out*.println("mensaje = " + mensaje);

ARRAY AVANZADO (video 8)

Ordenar array

Ordenar alfabéticamente: array.sort(productos);

//iterar arreglo  
for (String producto: productos){  
 System.*out*.println("producto = " + producto);  
}

Buscar en arreglo

Scanner buscar = new Scanner(System.*in*);//este bloque me hace la función de scanner para digitar el dato a  
//buscar en el arreglo  
System.*out*.println("Ingrese el valor a buscar...");  
String valorBuscado = buscar.next();  
  
//bloque para recorrer el arreglo y encontrar la palabra  
int pos;  
for(pos=0; pos< palabras.length && !palabras[pos].equalsIgnoreCase(valorBuscado) ;pos++){  
 //en la línea anterior lo que hace es que si en la posición (pos) está el valor buscado se salga del bucle  
 //y queda en (pos) cargada la posición del valorBuscado  
 // en el for buscamos la posición solamente  
}  
if(pos== palabras.length){//por medio del if determinamos que si la posición que va para el arreglo palabras  
 //de 0 a 4 y el length es de 5 nunca se salto del ciclo llegando al mismo valor del tamaño del array, indica que la palabra  
 // o el dato no se encontró  
 System.*out*.println("valor no encontrado...");  
}else{  
 System.*out*.println("palabra buscada: "+palabras[pos]);  
 System.*out*.println("posición en el array: "+ pos);  
}

Tambien se puede utilizar la instrucción (equals) en la linea

for(pos=0; pos< palabras.length && !palabras[pos].equals(valorBuscado) ;pos++)

{}  
pero esto me diferencia entre mayúsculas y minúsculas equalsIgnoreCase me permite buscar palabras iguales sin importar si están en mayúsculas ominusculas

MODIFICAR EL ARREGLO

Vamos a eliminar la palabra que se busco en el punto anterior por tal motivo ya tenemos la posición que contiene esa palabra

Lo que se hace es que en la posición donde estaba la palabra se corren los otros datos del arreglo de derecha a izquierda y luego se genera un arreglo nuevo pero sin el ultimo dato, que quiere decir si tenemos un arreglo de 5 el nuevo tendrá un arreglo de 4 se elimina siempre el último dato porque esa posición va a quedar duplicada al momento de desplazar los datos a la posición donde eliminamos

//bloque para correr los datos de derecha a izquierda eliminando el dato buscado o remplazando la informacion que contenía por la que existe en el resto del arreglo

for (int i=pos; i<palabras.length-1; i++){  
 palabras[i] = palabras[i+1];

}

arreglo resultante despues de remplazar el dato buscado

Gaby

Maria

Ana

Stela

Stela

Ahora creamos un nuevo array eliminando la última posición o con una posición menos

String [] nuevo = new String[palabras.length-1];//de esta forma le decimos el tamaño del array resultante

Va a ser del tamaño de palabras menos un dato

Utilizamos

System.arraycopy(arreglo a copiar, desde que posición, arreglo destino, desde que posición del arreglo destino vamos a copiar, tamaño del arreglo donde vamos a copiar)

System.*arraycopy*(palabras,0,nuevo,0,nuevo.length);

Video 9 String

Si creamos dos string de forma diferente no van hacer iguales pero si los creamos de la misma forma y el contenido es el mismo apuntara a la misma direccion de memoria y serán iguales

String nombre = "Gabriel";  
String nombre2 = new String("Gabriel");  
boolean comparar = nombre2==nombre;  
System.*out*.println("comparar = " + comparar);  
//resultado dice que es false pues uno se creo en forma literal y el otro  
// en forma de instancia. el contenido es el mismo como observamos con  
//equals pero su estructura es difrente  
comparar = nombre.equals(nombre2);  
System.*out*.println("comparar con equals = " + comparar);  
//resultado true

Ahora vamos a ver el resultado cuando se crea igual y su contenido es el mismo

String nombre3 = "Gabriel";  
comparar = nombre == nombre3;  
System.*out*.println("comparando estructuras y contenido:"+comparar);  
//comparando estructuras y contenido:true

Comentar bloque de codigo ctrl+shift+/

CONCATENACION

String curso = "Java 2023 -String";  
String docente = "Gabriel Shadu";  
String mensaje = curso + " con "+ docente;  
System.*out*.println("mensaje = " + mensaje);

9:33 video9

int n=10;  
int n2=20;  
//si quisieramos mostrar la suma de estos dos numeros  
System.*out*.println("la suma es: "+ n + n2);

* la suma es: 1020

porque java entiende que lo primero que tiene e un mensaje y pues después utiliza el símbolo de concatenación (+) asume que el resto de variables hacen parte del mensaje,

solución 1:

System.*out*.println("la suma es: "+ (n+n2));

Solución 2:

System.*out*.println(n+n2 + " es el resultado de la suma");

Porque java lee de arriba abajo y de izquierda a derecha por eso le da prioridad a la suma y después concatena

Solución 3://esta no la muestra en el video, pues me imagino que utiliza mas recursos

int suma = n+ n2;  
System.*out*.println(" el resultado de la suma es= " + suma);

CONCAT

Une dos o más string sin modificar su contenido el resultado se puede almacenar o mostrar

System.*out*.println(curso.concat(" con ").concat(docente));  
String mensaje2 = curso.concat(" con: ").concat(docente);  
System.*out*.println("mensaje2 = " + mensaje2);

CLASE Y OBJETOS EN JAVA

POO (programación orientada a objetos)

Lo primero es que debemos entender es que a partir de un problema debemos definir que objetos contiene y este objeto puede contener otros objetos, siempre que estos tengan atributos y comportamientos

La Clase: es un molde,

* primero se define un diseño (molde) de como va a ser la entidad que queremos crear
* una abstracción de un objeto de la realidad
* en base a ese molde podemos crear diferentes tipos de ese molde
* una clase es un molde o prototipo que define las variables y los métodos comunes a un cierto tipo de objeto

ejemplo de crear la clase automovil

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Diagrama UML

El símbolo menos (-) y el (+) son modificadores de acceso

Crear Objetos

Operador new

El operador new reserva memoria dinámica para un objeto

Automovil auto = new Automovil()// se crea un objeto de la clase Automovil

El objeto es una instancia de la clase

Abrimos un nuevo projecto y dentro de este dejamos la clase main quieta y creamos un paquete que llame Modelo dentro de este cree una java class que llave Automovi, le agregamos atributos y se ve asi:

package Modelo;  
  
public class Automovil {  
 private String color; //private es un modificador de acceso   
 private Integer tanque;  
 private String persona;  
 private double cilindrada;  
  
}

15:54

normalmente, se utiliza siempre el atributo private  
para modificar las características se hace a través de métodos  
igual para crear un automóvil se hace a través de un método

Para crear un objeto o en este caso un automóvil se utiliza un constructor

public Automovil(){//constructor vacio  
  
}

y para acceder a las características se utilizan los métodos getter y setter GET Y SET

Get: lee atributo

Set: modifica atributo

Para decir para ese objeto cuales son sus características particulares de ese objeto

22:24

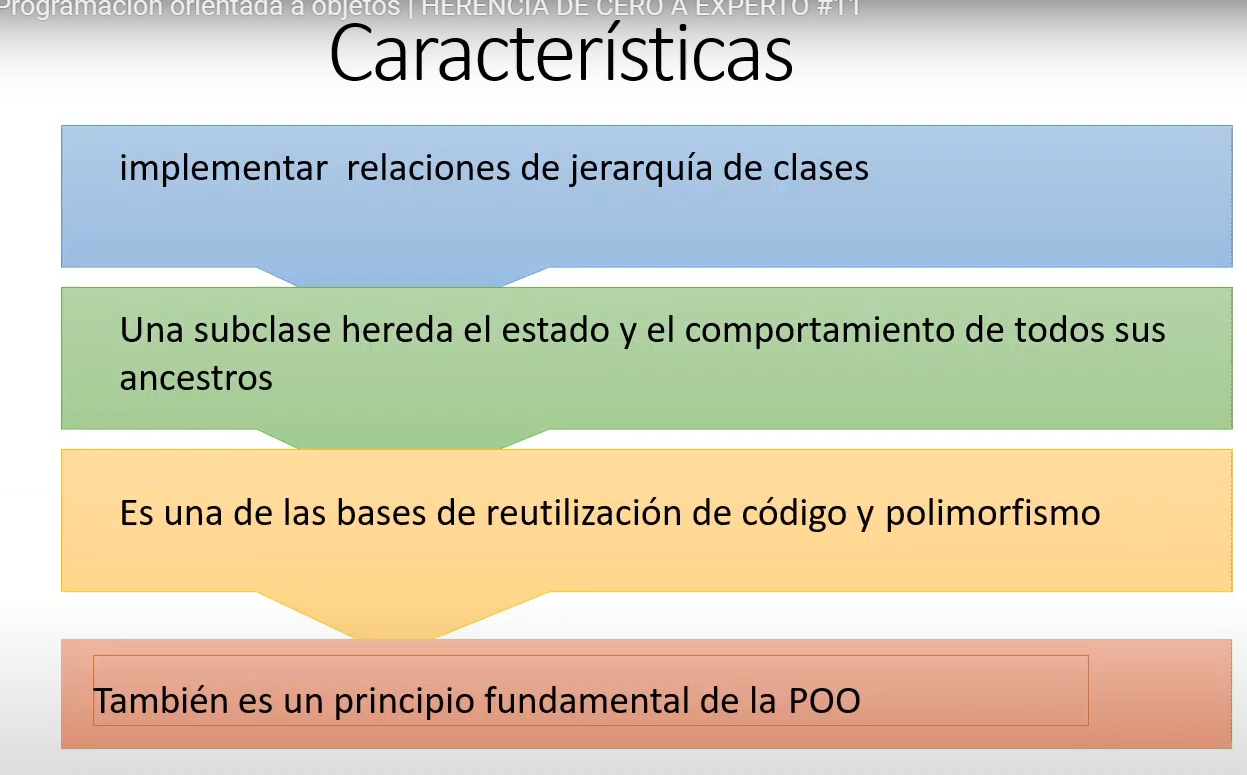
Para ver todas las propiedades de manera rápida se utiliza el método tostring

En la clase automóvil damos click derecho y en generate creamos el método toString que nos pregunta que información queremos mostrar y nos genera el codigo necesario que cuando llamemos al método nos muestre todo lo que seleccionamos

Automovil {color='Blanco', tanque=50, persona='Fernando Ortiz', cilindrada=2000.0}

HERENCIA

Es una relación de parentesco entre dos clases, una es padre de las otras, las otras son hijas de esta, si tenemos que programar una clase superior o raíz y solamente vamos a tener una subclase o clase hija no tiene ningún sentido.



Java no permite herencias múltiples

Para que exista herencia tiene que haber una relación de parentesco entre las clases

Hay dos formas de saber si es necesario utilizar la herencia

La generalización y la Especialización

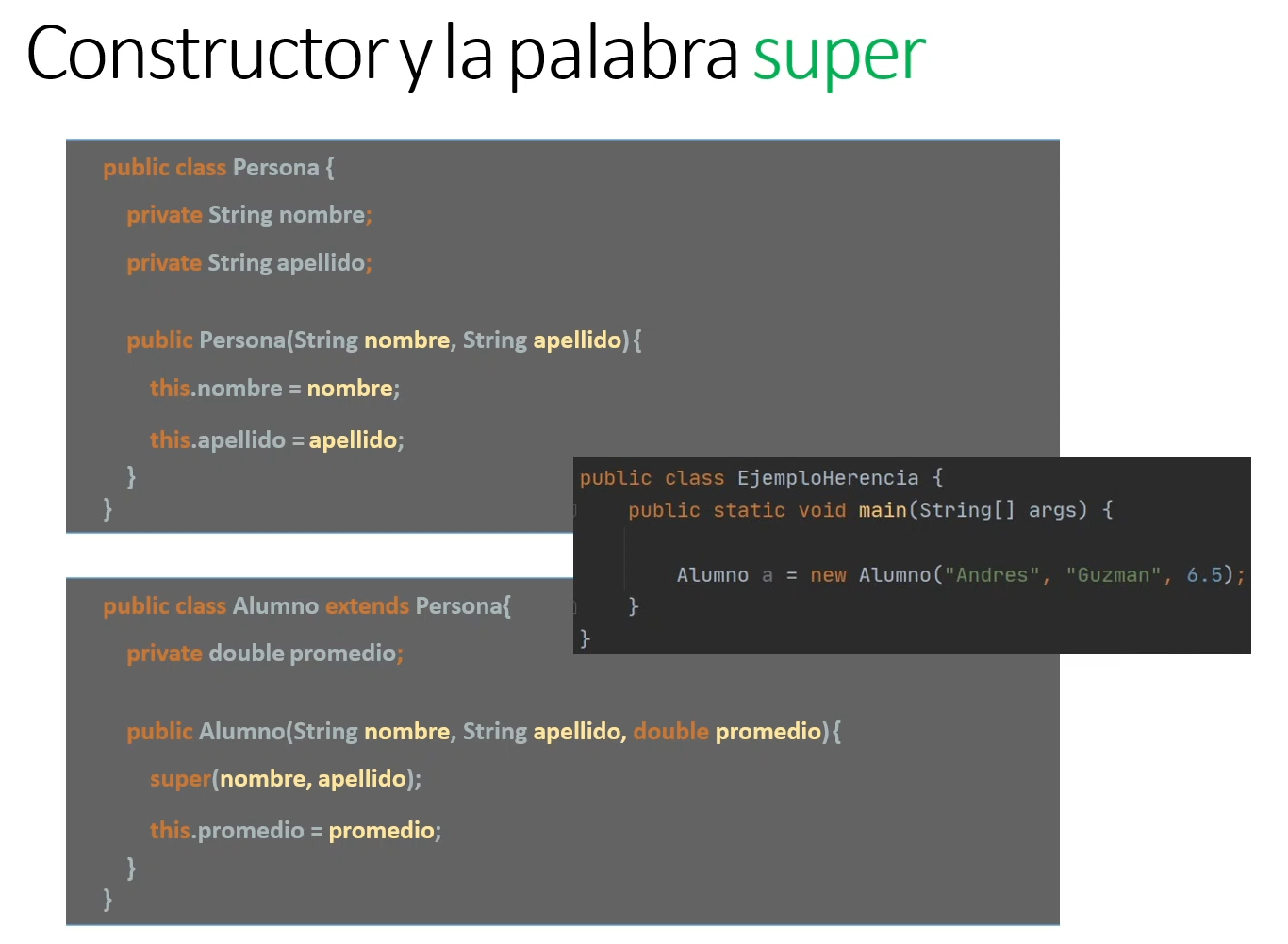
Especialización: son clases especiales que surgen a partir de una clase general, donde vemos la necesidad de utilizar los mismos datos en cada clase.

Generalización: a partir de las clases donde se repite codigo vemos la necesidad de crear una clase superior que contenga todo esto.

Constructor y la palabra Super

Constructor es el método que se llama igual que la clase y lo que hace es construir esa clase, genera el espacio en memoria para almacenar todos los atributos que hayamos declarado en ese constructor.

Super: es una palabra que esta relacionada con el constructor y con la herencia



La clase persona es la clase padre, la clase alumno es la que hereda los atributos de la clase persona y tambien tiene un constructor que contiene un atributo propio de esa clase y vemos que por medio de super de clase hijo nos permite utilizar el constructor de la clase padre (persona)

Sobre escritura de métodos:

Los métodos que se heredan de la clase padre se pueden sobrescribir o modificar para la clase hija, esto se hace redefiniendo en la clase hija un método con el mismo nombre

Interfaz de usuario gráfica

Descripción generada automáticamente

Al hacer esto aparece la palabra @Override (sobre escritura de método)

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ejemplo

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

To string del padre

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Se utilizo el super para el toString en el hijo

14:39 video 11

Sobreecritura de métodos / Restringir la herencia

* para impedir la herencia usamos la palabra final:

final public class persona{

[]….

}

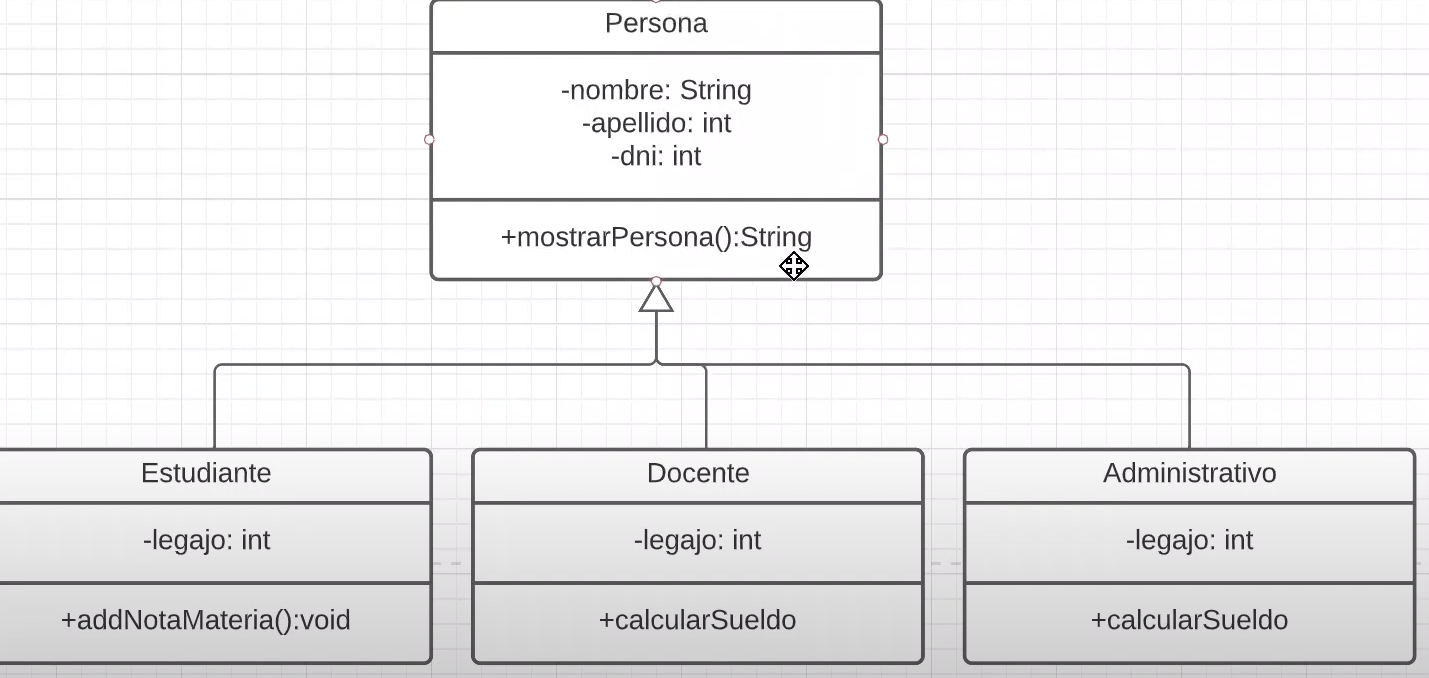
Esto hace que la clase declarada con final no se puede heredar por los hijos

Tambien se puede hacer con un método:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ejemplo de diagrama



POLIMORFISMO

Diagrama UML

Diagrama

Descripción generada automáticamente

En el diagrama la clase padre es “persona” y las clases hijo son “estudiante, docente y administrativo” que heredan los atributos nombre, apellido y dni y el método mostrar persona

El polimorfismo nos permite tratar al estudiante, docente y administrativo como se fueran una persona

Al crear instancias se hace los siguiente

Persona estudiante1= new Estudiante("Juan Perez", 1254,56);  
Persona docente = new Docente("Carlos Porras", 142563,30);

Podemos hacer una lista con objetos de clase persona y estudiante

List<Persona> list = new ArrayList<>();  
list.add(estudiante);  
list.add(estudiante1);  
list.add(docente);  
list.add(docente1);

Los mostramos

for (Persona p:list){  
 System.*out*.println("p.toString() = " + p.toString());  
}

resultado

p.toString() = Estudiante{numeroLegajo=56Persona{nomYApe='Juan Perez', dni=1254}}

p.toString() = Estudiante{numeroLegajo=56Persona{nomYApe='Juan Perez', dni=1254}}

p.toString() = Docente{cantHorasClase=30Persona{nomYApe='Carlos Porras', dni=142563}}

p.toString() = Docente{cantHorasClase=35Persona{nomYApe='Luis Prada', dni=8875665}}

si quisiéramos modificar un dato de la clase docente utilizando las instancias anteriores se procede así:

((Docente)docente).setCantHorasClase(45);//es un timo de casteo por lo que se instancio con la clase padre Persona

Si quisiéramos mostrar solo los docentes de la lista se utiliza Instanceof así:

for (Persona pe:list) {  
 if (pe instanceof Docente) {//solo muestra información si el objeto instanciado es docente  
 System.*out*.println("Docente = " + pe.toString());  
 }  
}

CLASES ABSTRACTAS

Es una clase en la que no se pueden declarar instancias

* cuando tenemos un conjunto de subclases en que algún comportamiento está presente en todas ellas, pero se implementan en forma distinta en cada una, entonces creamos una clase abstracta.
* El modificador **abstract**  declara que la clase es una clase abstracta.
* Una clase abstracta podría contener métodos abstractos (métodos sin implementación)
* Una clase abstracta esta diseñada para ser una super clase y no puede instanciarse.

Ejemplo

abstract public class Mamifero{

private String hábitat;

private float peso;

private String color;

abstract public void hacer ruido();

abstract public void comer();

abstract public void dormir();

abstract public void mover();

}

Todos los mamíferos tienen un hábitat un peso y un color además hacen ruidos, comen, duermen y se mueven, ¿pero todos no se mueven igual? “no”, la implementación de estos métodos va a variar según quien herede esta clase, pero todos van a heredar estos métodos.

3:58

Las clases abstractas no se pueden instanciar

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Nos dice que la clase Hoja es abstracta y no se puede instanciar pero si implementamos el método que nos indica nos permitiría instanciar

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

De esta forma se instanciaría un método abstracto, que no es algo que se haga en realidad pues no es lo más recomendado pues si se crea un método abstracto es para que se sobrescriban los métodos de la clase.

}  
public static void ImprimirAbs(Hoja hoja){//la clase abstracta me permite modificar métodos  
 System.*out*.println("hoja.imprimir() = " + hoja.imprimir());  
}

En el ejemplo este método se crea en el main para imprimir la información utilizando las propiedades de la clase abstracta que me permite modificar el método y el polimorfismo

13:44

*ImprimirAbs*(cv);//utilizamos el mé*todo abstracto* System.*out*.println("============================================");  
 Informe inf = new Informe("Contenido del informe", "Dibu Martinez","Rodrigo de Paul");  
 *ImprimirAbs*(inf);  
 }  
 public static void ImprimirAbs(Hoja hoja){//la clase abstracta me permite modificar métodos  
 System.*out*.println("hoja.imprimir() = " + hoja.imprimir());  
 }  
}

El método y la clase abstracta nos permite realizar lo anterior

Interfaces

Las clases abstractas puede tener métodos que se implementan en cambio las interfaces no se implementa ningún método, las interfaces es un comportamiento y ese puede pertenecer a un conjunto de clases independientemente de su relación de parentesco

Y para poder utilizar el método de imprimir sin necesidad de extender o de heredar de la clase Hoja se hace una interface pues todos comparten el mismo comportamiento que es imprimir

25:33

API DE COLECCIONES DE JAVA

Estructuras de Datos

Colecciones:

* clases utilizadas para agrupar y almacenar objetos.
* Dos grandes grupos: Collection y Map
* Son dinámicas: gestion de memoria dinámica (no hay que asignar un tamaño como los array)

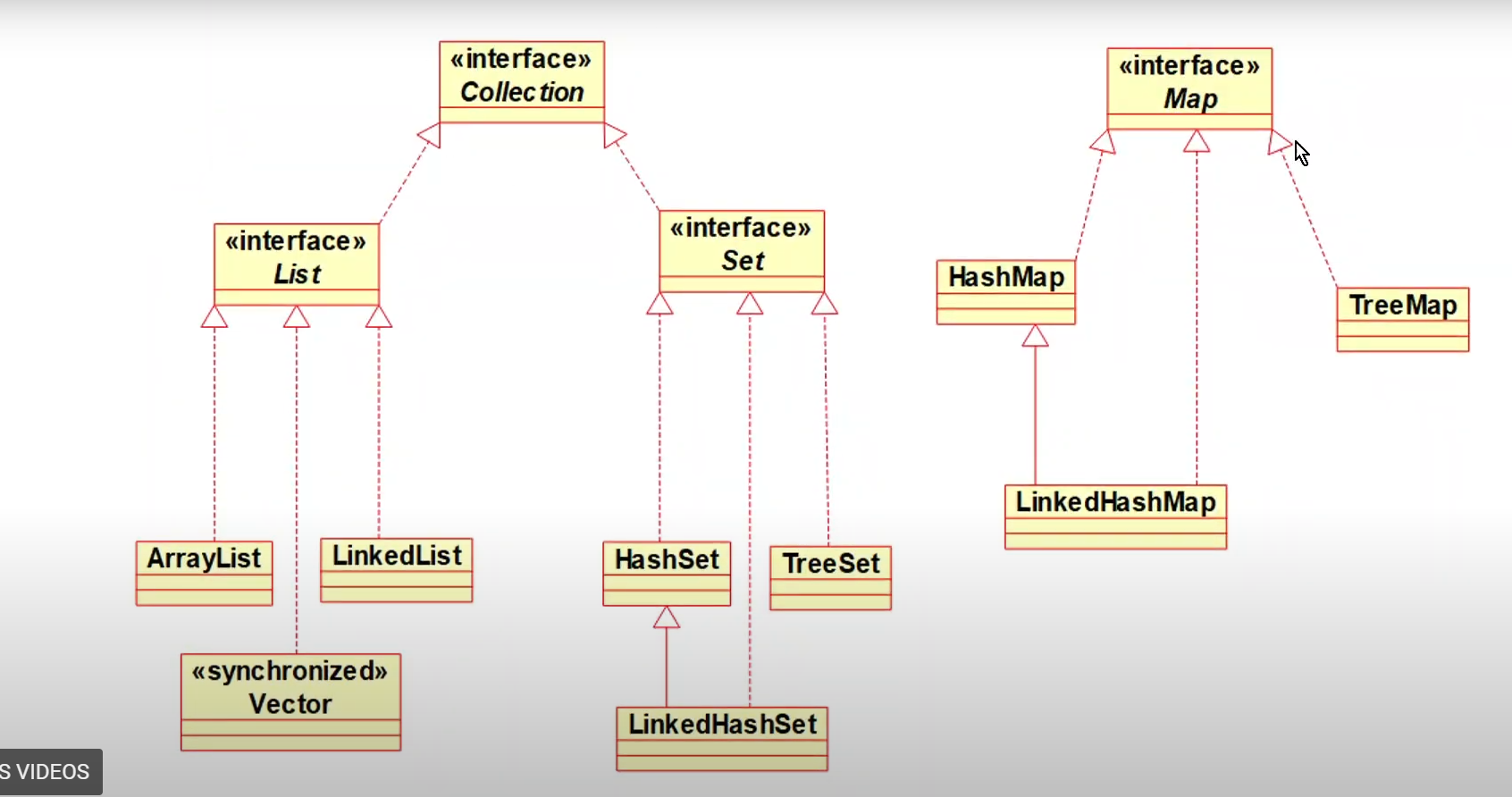
Tipos de Agrupaciones

El API de colecciones se utiliza para agrupar objetos y consta de 3 tipos: List, Set y Map

List: almacena objetos en una secuencia determinada. Cada vez que se agrega un objeto respeta un orden de ingreso tiene posición, permite elementos duplicados.

Set: no permite elementos duplicados y no mantiene el orden de los elementos

Maps: los mapas almacenan información en base a pareja de llaves (key) y valores (values).



Crear List

List<String> array = new ArrayList<>();

Cuando se hace esto se importan estas librerías

import java.util.ArrayList;  
import java.util.List;

* Agregar elementos: arreglo.add("Pedro");
* Acceder a elentos: System.*out*.println("arreglo.get(2) = " + arreglo.get(2));
* Eliminar elementos: arreglo.remove(2);
* Remplazar elementos: arreglo.set(2,"Nelson").
* //buscar elementos  
  Boolean existe = arreglo.contains("Juan");  
  System.*out*.println("existe Juan en el Arreglo = " + existe);

//agrego elemetos  
arreglo.add("Ana");  
arreglo.add("Luis");  
arreglo.add("Zuly");  
System.*out*.println("arreglo = " + arreglo);

arreglo = [Pedro, Maria, Nelson, Ana, Luis, Zuly]

Ordeno los elementos

Collections.*sort*(arreglo);// se importa la librería import java.util.Collections;

System.*out*.println("arreglo = " + arreglo);

Resultado:

arreglo = [Ana, Luis, Maria, Nelson, Pedro, Zuly]

20:44

Ahora vamos a remplazar la ArrayList por LinkedList en la linea donde se creo el arreglo

List<String> arreglo = new ArrayList<>();

Pasamos a LinkedList asi:

List<String> arreglo = new LinkedList<>();

Observamos si todo lo que hicimos con el array list tiene algún error y vemos que funcionan todos los métodos de la misma forma, vamos a ejecutar y observar el resultado,…. Vemos que el resultado es exactamente el mismo.

Esto se puede hacer porque declaramos de forma polimórfica no declaramos de esta forma

LinkendList<String> arreglo = new LinkedList<>(); si no que declaramos como List y eso se puede porque tanto ArrayList como LinkedList pertenecen a ese tipo de interfaz List.

Los cambios son internamente pues el comportamiento lo vamos a ver en los siguientes ejemplos

Ahora si queremos acceder a los métodos propios de la LinkedList o de ArrayList debemos declarar explícitamente

LinkendList<String> arreglo = new LinkedList<>();

Ya que las linkend nos permite agregar elementos al principio del arreglo mientras que el Arraylist no

Como el método addFirst(), agrega al principio del arreglo

arreglo.addFirst("Fernando");

agregar al final:

arreglo.addLast("Salomon");

se puede utilizar el método Iterator

ListIterator<String> iterator = arrglo.listIterator() // se declara con mayúscula y minúscula este método.

if (iterator.hasNext()){//el hasNext es para preguntar si hay un elemento en la siguiente posición.

String cadena = iterator.next();//carga la posición en cadena

cadena = iterator.next();//carga la segunda posición

String cadena1 = iterator.previous();// en iterator esta cargado ya el valor de la tercera posición  
//por eso si mostramos el previo va mostrar de nuevo la segunda posición

Set y Map

Los set y los map se parecen en que ambos implementan un algoritmo que se llama hascode

Que hace que no se permitan duplicados en estas colecciones aunque son de interfaces diferentes pues los set pertenecen a collection y los map a su propia interfaz map

Set

HashSet y TreeSet

HashSet: no permite duplicados y sus datos están en desorden

TreeSet: no permite duplicados pero sus datos si están ordenados, es un árbol

Los LinkedHashSet deriva del HashSet es implementado con una lista doblemente enlazada.

Map

HashMap y TreeMap

LinkedHashMap hereda de HashMap

La diferencia de los map con los set es que estos tienen dos valores la llave y el valor

Creamos un HashSet

Set<String> cadenas = new HashSet<>();

Agregamos elementos

cadenas.add("Pepe");  
cadenas.add("Carlos");  
cadenas.add("Juan");  
cadenas.add("Ana");  
cadenas.add("Maria");

mostramos en pantalla

System.*out*.println("cadenas= " + cadenas);

Resultado

cadenas= [Ana, Juan, Maria, Carlos, Pepe]

como vemos no respeta el orden de inserción de los elementos.

9:27

Los hashset no se puede ordenar como se hacia en las LIST

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Me dice que el método requiere una LIST no un Set, por ese motivo se requiere realizar una transformación del Set

List<String> lista = new ArrayList<>(cadenas);  
Collections.*sort*(lista);//permite ordenar pero ya no es un Set si no una lista  
System.*out*.println("lista = " + lista);

//para buscar un elemento

boolean esta = cadenas.contains("Pepe");

* Para cambiar un dato, se debe buscar, eliminar y añadir un nuevo elemento

cadenas.remove("Pepe"); //elimina elemento

caden.add // agrega elemento

TreeSet

Como se crea

Set<String> cadenas = new TreeSet<>();

* Agregar elementos

cadena.add("Pepe");

* Eliminar

cadena.remove(“String”).

* Buscar

cadena.contains(“String”)

la diferencia que se observa es que los elementos quedan ordenados cuando se muestran y pues al ordenar cada elemento nuevo esto genera un costo de hardware y hace que sea un proceso mas demorado que se llama el costo de balanceo

a diferencia de la lista no existe una posición por eso no existen los métodos Get ni Set

esto pasa los dos tipos de Set no tiene posición

Map

Para crear un HashMap

Map<Integer, String> mapa = new HashMap<>();

En la creación pide especificar que tipo de dato es el key y que tipo de dato es el vale

Agregar datos:

mapa.put(1, "Pepe")

los Map permiten el mismo VALUE, pero la key tiene que se siempre diferente

// crear un mapa con llave y un objeto a partir de la clase Persona que tiene dos parámetros Dni y Nombre  
Map<Integer, Persona> mapaPersonas = new HashMap<>();  
  
  
//creo los objetos  
Persona p1 = new Persona(88205852, "Fernando");

//adiciono

mapaPersonas.put(2,p1);

vamos a buscar un dato

Persona busqueda = mapaPersonas.get(2);  
System.*out*.println("busqueda = " + busqueda);

busqueda = Persona{dni=88205852, nombre='Fernando'}

vamos a eliminar un dato

mapaPersonas.remove(“el key del dato a eliminar”);

si agregamos un numero de key menor a los existentes este se ordena, por tal motivo los map tienen un costo de ordenamiento

//actualizar un dato  
//se utiliza lo mismo que para agregar simplemente que se cambia el valor  
//tenemos este mapa:

mapaPersonas = {1=Persona{dni=1043958263, nombre='Juan'}, 2=Persona{dni=88205852, nombre='Fernando'}, 3=Persona{dni=88251814, nombre='Carlos'}}

System.*out*.println("mapaPersonas = " + mapaPersonas);  
  
//creamos un nuevo objeto que vamos a actualizar en la posición 2 del mapa  
Persona p4 = new Persona(45125362, "Jose");  
//vamos a actulizar la poscición dos con este dato  
mapaPersonas.put(2,p4);  
System.*out*.println("mapaPersonas = " + mapaPersonas);

mapaPersonas = {1=Persona{dni=1043958263, nombre='Juan'}, 2=Persona{dni=45125362, nombre='Jose'}, 3=Persona{dni=88251814, nombre='Carlos'}}

Otros Metodos importantes

containsKey containsValue

30:02

boolean existe = mapaPersonas.containsKey(2);  
System.*out*.println("existe = " + existe);// existe = true  
  
existe = mapaPersonas.containsValue(p4);  
System.*out*.println("existe = " + existe);// existe = true

Iterar el Map

Collection<Persona> personas = mapaPersonas.values();//se importa la clase Collection  
for(Persona persona:personas){  
 System.*out*.println("Nombre: " + persona.getNombre()+ " Dni: " + persona.getDni());  
}

//Iterar con el mapa para mostrar las llaves o keys  
  
Collection<Integer> keysPersona = mapaPersonas.keySet();  
for(Integer key:keysPersona){  
 System.*out*.println("key = " + key);  
}

TREEMAP

Toso es exactamente igual que el hashmap vamos a crear una clase y copiamos y pegamos el contenido de hashmap en la clase pero cambiamos el nombre del Map ahora será treeMap y vemos que no tiene ningún error y todos los métodos aplican de la misma manera.