

Ingeniería de Sistemas



ESPECIALIZACION EN INGENIERIA DE SOFTWARE MODULO PATRONES DE DISEÑO DE SOFTWARE



EL DOCENTE



JAIRO FRANCISCO SEOANES LEON

jairoseoanes@unicesar.edu.co (300) 600 06 70



Educación formal

- ✓ Ingeniero de sistemas, Universidad Popular del Cesar sede Valledupar, Feb 2002 Jun 2009.
- ✓ MsC en Ingeniería de Sistemas y Computación, Universidad Nacional de Colombia, Bogotá, Feb 2011 Mar 2015
- PhD Ciencia, Tecnologia e innovación, Urbe, Venezuela, Mayo 2024

Formación complementaria

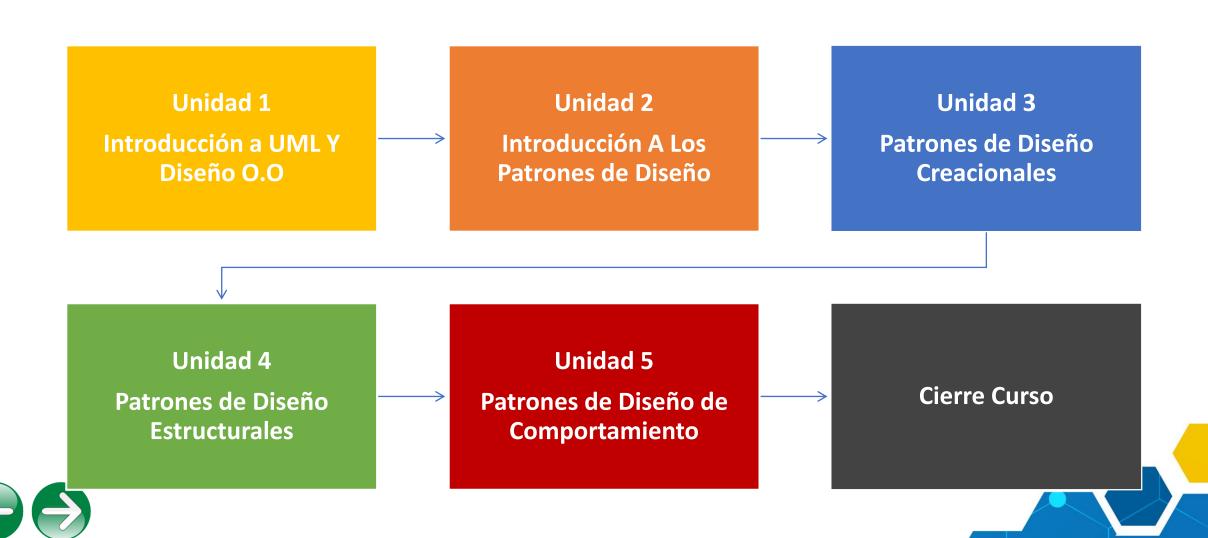
- ✓ AWS Academy Graduate AWS Academy Cloud Foundations, 2022 https://www.credly.com/go/p3Uwht36
- ✓ **Associate Cloud Engineer Path -** Google Cloud Academy, 2022 https://www.cloudskillsboost.google/public profiles/c7e7936c-3e37-4bad-b822-74d40c49d0db
- ✓ Fundamentos De Programación Con Énfasis En Cloud Computing AWS Academy y Misión Tic 2022
- ✓ Google Cloud Computing Foundations Google Academy, 2022
- ✓ Aplicación de cloud: retos y oportunidades de mejora para las empresas de software gestionando la computación en la nube – Fedesoft, 2023
- ✓ **Desarrollo De Aplicaciones Web En Angular, Para El Nivel Frontend** Universidad EAFIT, 2023
- ✓ Microsoft Scrum Foundations Intelligent Training MinTic , 2023

Experiencia profesional

- ✓ **Docente Universitario**, Universidad Popular del Cesar sede Valledupar, marzo del 2013.
- **Técnico de Sistemas Grado 11**, Rama judicial Seccional Cesar, SRPA Valledupar, Junio del 2009

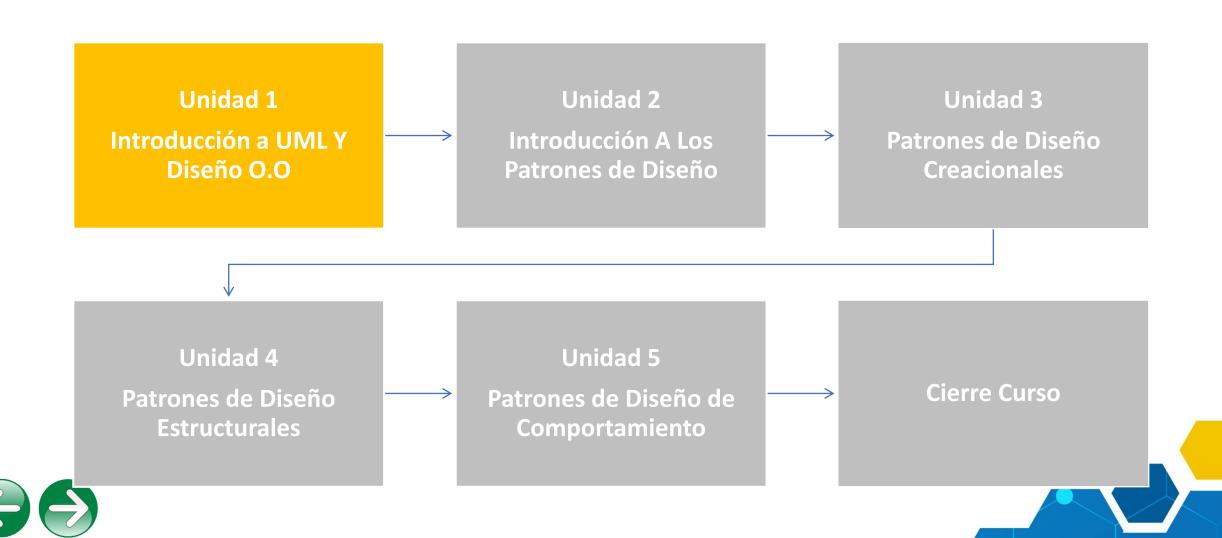


MODULO DE PATRONES DE DISEÑO DE SOFTWARE





MODULO DE PATRONES DE DISEÑO DE SOFTWARE







Unidad 1. Introducción a UML y diseño O.O

- 1.1 Introducción a UML.
- 1.2 Introducción a la Programación orientada a Objetos OOP
- **1.3** Conceptos básicos: objeto, atributo, método, miembro, mensaje, clase, comunicaciones y eventos.
- **1.4** Características de la Programación Orientada a Objetos: Abstracción, Encapsulamiento, Principio de Ocultación, Herencia, Polimorfismo
- 1.5 Utilización de diagramas UML para el análisis de requisitos: casos de uso y secuencia.
- 1.6 Diagramas de análisis UML: clases, paquetes, actividad, etc.





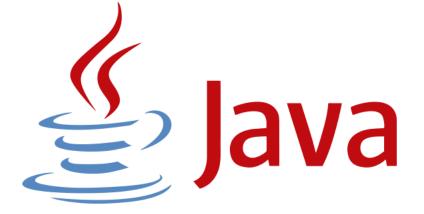




















La complejidad del software

Se exige el uso de un equipo de desarrolladores

se producen comunicaciones entre ellos complejas

El dominio del problema

Difíciles interacciones entre los usuarios y sus desarrolladores

diferentes perspectivas de la naturaleza del problema y de la solución.

Los requisitos de un sistema software cambian durante su desarrollo Gestionar el Proceso de Desarrollo

Complejidad del Software Difícil coordinación del grupo de trabajo

El equipo está disperso geográficamente

La Flexibilidad del Software

Las necesidades cambian con el tiempo, aun cuando el software se este desarrollando, entonces es importante poder hacer cambios posteriores al software

Comportamiento impredecible del software

El software presenta el comportamiento de un sistema discreto

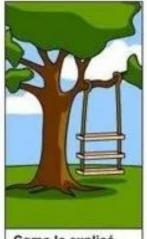
Alta sensibilidad al ruido



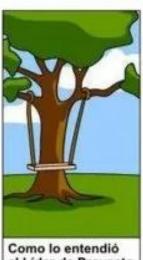




La complejidad del software



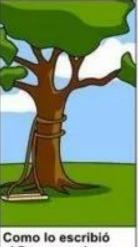
Como lo explicó el Cliente



el Lider de Proyecto



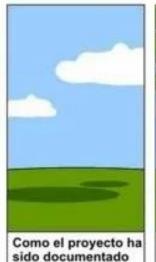
Como lo diseñó Ingenieria

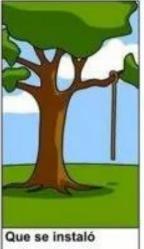


el Programador

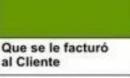


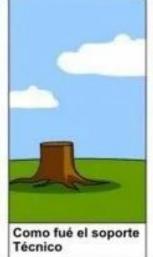
el Vendedor













La causa número uno en fallas de proyectos de software es la complejidad

Requisitos poco claros o ambiguos Falta de comunicación con los stakeholders Cambios frecuentes en los requisitos Prioridades mal definidas Mala documentación

Consecuencias:

Incremento de costos Retrasos Sistemas que no entregan valor Fracaso total





Herramientas para tratar la complejidad

Recolección y análisis exhaustivo de requisitos

Iteraciones ágiles

Gestión de cambios

Prototipos y pruebas tempranas

Comunicación constante



Descomponer

Descomposición algortimica

Descomposición en objetos

Abstraer

Construir un modelo simplificado de la realidad

Jerarquizar

Ordenar los elementos de un sistema







Paradigmas de programación

Modelo conceptual para desarrollar programas

Orientado al procedimiento

Orientado a funciones

Orientado a la lógica

Orientado a objetos







Factores de calidad del software

- Funcionalidad: Capacidad del software para satisfacer los requisitos especificados.
- **Confiabilidad:** Capacidad del software para mantener su nivel de rendimiento bajo condiciones específicas y durante un período de tiempo determinado.
- **Usabilidad:** Capacidad del software para ser entendido, aprendido y utilizado de manera efectiva y eficiente por los usuarios.
- **Eficiencia**: Capacidad del software para realizar las funciones requeridas con la menor cantidad de recursos posibles.
- **Mantenibilidad:** Capacidad del software para ser modificado y mejorado de manera efectiva y eficiente.
- Portabilidad: Capacidad del software para ser transferido de un entorno a otro.



La norma ISO 9126







Factores de calidad del software











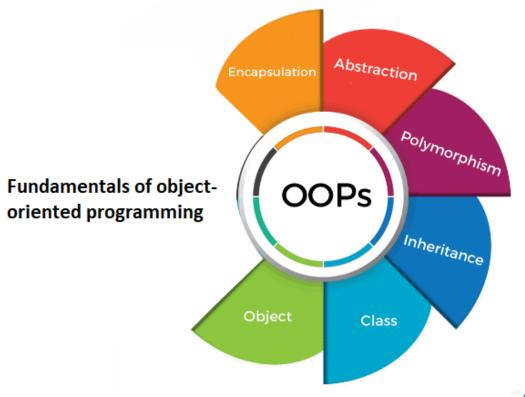








Programación orientada a objetos - POO



La programación orientada a objetos - POO es un paradigma de programación en el cual los programas expresan en un conjunto de objetos que colaboran entre si para resolver un problema.

nttps://www.iinkeain.com/puise/tunaamentais-object-orientea-programming-usman-maiik-gd0gf

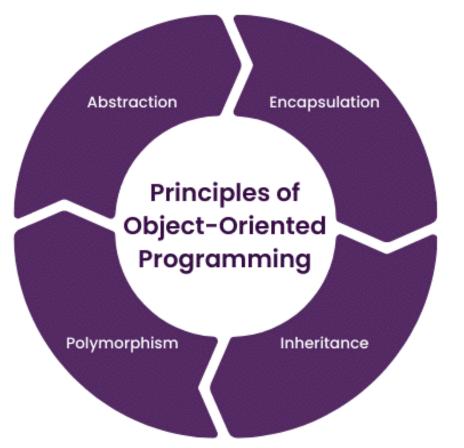








Principios fundamentales de la POO



https://www.theknowledgeacademy.com/blog/principles-of-object-oriented-programming/



La Abstracción es el modelo de un objeto o fenómeno del mundo real, limitado a un contexto específico, que representa todos los datos relevantes a este contexto con gran precisión, omitiendo el resto.

La *encapsulación* es la capacidad que tiene un objeto de esconder partes de su estado y comportamiento de otros objetos, exponiendo únicamente una interfaz.

La herencia es la capacidad de crear nuevas clases sobre otras existentes. La principal ventaja de la herencia es la reutilización de código.

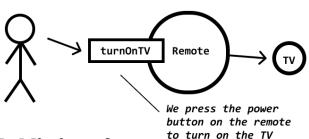
El polimorfismo es la capacidad de un objeto para dar diferentes respuestas a un mismo mensaje





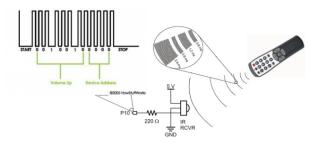
Principios fundamentales de la POO – La Abstracción

Abstraction example: Using a remote control to turn on a TV



Public interface

Public Declarative Human-centered Highest level of abstraction



Implementation

Private
Imperative
More technical
Lower levels of abstraction

https://khalilstemmler.com/articles/object-oriented/programming/4-principles/



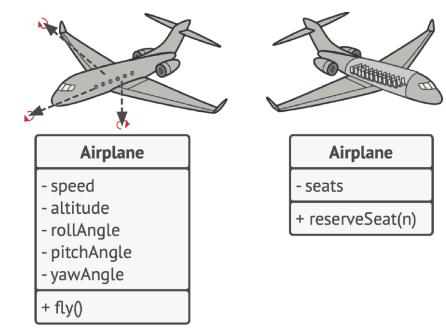
¿ Realmente eres consciente de cómo se enciende tu televisor cuando presionas el botón ON del control remoto? ¿Usted, como usuario, necesita saber la secuencia específica de 0 y 1 que emite su control remoto para indicarle al receptor del televisor que debe encenderse? ¿O es suficiente presionar el botón ON ?





Principios fundamentales de la POO – La Abstracción

- ☐ La abstracción en un principio fundamental de diseño
- Consiste en definir las características y comportamientos esenciales de un objeto en un contexto.
- Permite reducir los niveles de complejidad en los sistemas
- ☐ Permite simplificar la realidad, permitiendo a los desarrolladores centrarse en lo que hace un objeto (su interfaz) en lugar de cómo lo logra.
- Mejora la legibilidad y el mantenimiento del código, ofrece una vista de alto nivel que facilita a los desarrolladores comprender y trabajar con estructuras complejas.



Shvets, A. "Sumergete en los patrones de diseño", 2019.



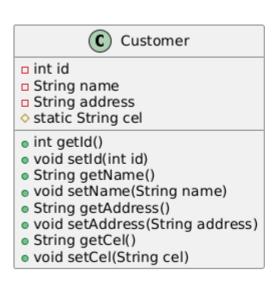




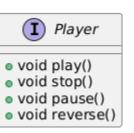
Principios fundamentales de la POO – La Abstracción

Tipos de abstracciones en OOP:

Clases, Clases abstractas, interfaces







Medida de la calidad de las abstracciones

- Acoplamiento
- Cohesión
- Suficiencia y completitud
- Primitividad







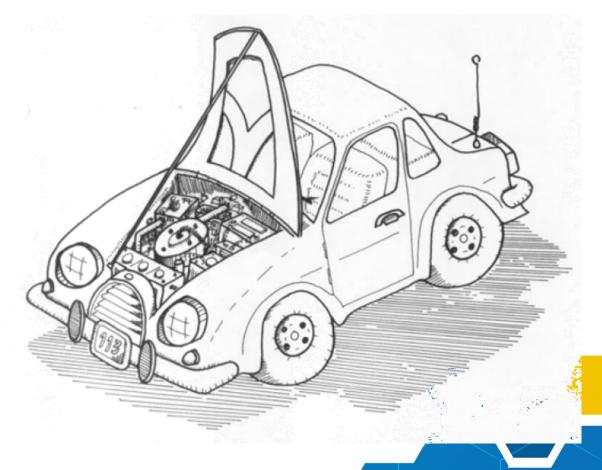


Principios fundamentales de la POO – Encapsulamiento

¿ Para arrancar el motor de un auto, tan solo debes girar una llave o pulsar un botón O necesitas conectar cables bajo el capó, rotar el cigüeñal y los cilindros, e iniciar el ciclo de potencia del motor. ?

El objetivo de encapsular es la ocultación de la implementación, para que ninguna parte del sistema dependa de la forma en que se implementa otra.





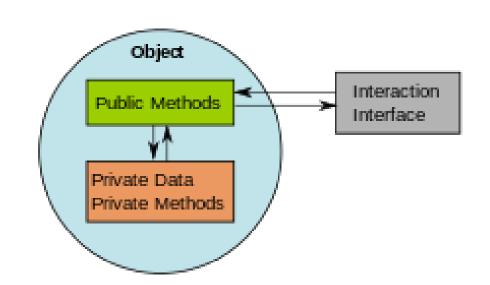




Principios fundamentales de la POO – Encapsulamiento

- ☐ El encapsulamiento es la propiedad que permite mantener oculto los detalles de implementación de una abstracción ante sus usuarios.
- ☐ La abstracción y el encapsulamiento son propiedades complementarias.
- ☐ El encapsulamiento permite que se pueda cambiar el funcionamiento interno de una abstracción y que sus usuarios no lo noten.
- ☐ Promover la integridad de los datos, la seguridad y la organización eficiente del código.





NombreDeLaClase

propiedad1 : TipoA

propiedad2 : TipoB

+ propiedad3 : TipoC

- método1(): TipoX

método2(): TipoY

+ método3(): TipoZ

private public protected

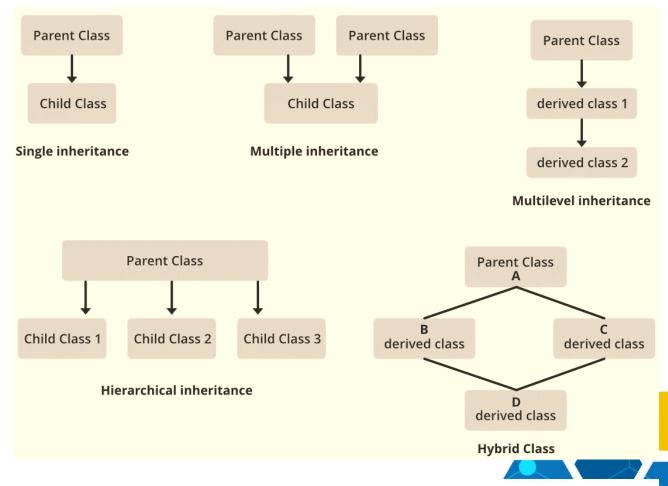






Principios fundamentales de la POO – Herencia

- ☐ La Herencia es la propiedad que permite crear nuevas abstracciones a partir de abstracciones ya existentes.
- ☐ Es el principio fundamental que promueve la reutilización de código.
- ☐ Permite jerarquizar abstracciones
- Clase base, padre o superclase
- Clase derivada, hija o subclase





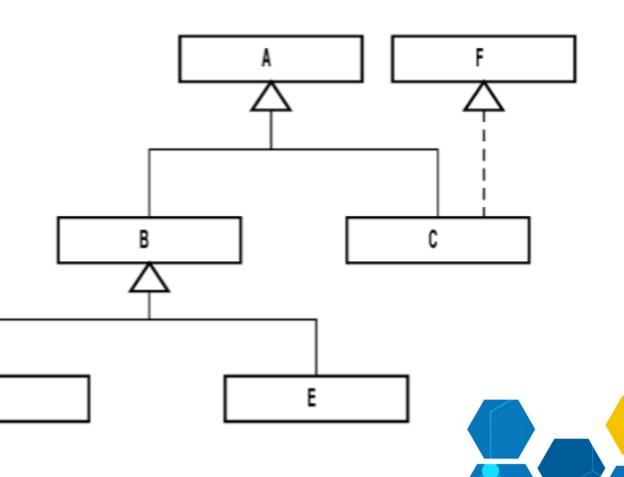




Principios fundamentales de la POO – Herencia

- ☐ Reduce el tiempo y el esfuerzo de desarrollo al tiempo que garantiza la coherencia entre las clases relacionadas.
- Permite la creación de clases especializadas que amplían o modifican la funcionalidad de la clase base, promoviendo la flexibilidad y adaptabilidad en el código base.
- No se debe abusar de la herencia, puede afectar el acoplamiento.
- Define la relación "es un tipo de"









Principios fundamentales de la POO – Polimorfismo

- ☐ La OOP permite tratar objetos de diferentes tipos como instancias de una clase base estándar. (gracias a la herencia).
- ☐ El polimorfismo es la propiedad de la OPP que otorga la capacidad a un mensaje de mostrarse en más de una forma.
- El programa no conoce el tipo de objeto concreto almacenado en una variable, pero gracias al **enlace dinámico**, puede realizar el llamado a la versión correcta de un método, para entregar la respuesta deseada.

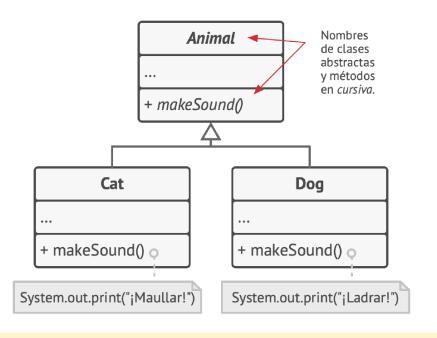








Principios fundamentales de la POO – Polimorfismo



Type of polymorphism

Compile time

Run time

Overloading method

Animal toby = new Cat or new Dog toby.makeSound() ?



El polimorfismo permite a los desarrolladores crear código genérico, promoviendo la reutilización, simplificando la implementación y contribuyendo a la adaptabilidad y escalabilidad de los sistemas de software.





Conceptos de la POO – Clases y Objetos

Persona

- +nombre
- +añoNacimiento
- +sexo
- -peso
- -estatura
- +Persona(CADENA nom, ENTERO an, CARACTER sex, REAL pe, REAL est)
- +calcularIndiceMasaCorporal()
- +calcularEdad()
- + ~Persona()

Clase

Plantilla utilizada para la creación de objetos.

Define los atributos y métodos que contendrán los objetos que se creen a partir de ella

Objeto1: Persona

nombre=SAMUEL añoNacimiento=2010 sexo=H peso=13.7 estatura=90.5

Persona(CADENA nom, ENTERO an, CARACTER sex, REAL pe, REAL est) calcularIndiceMasaCorporal() calcularEdad() ~Persona()

Objeto2: Persona

nombre=MARÍA ISABEL añoNacimiento=2002 sexo=M peso=45 estatura=140

Persona(CADENA nom, ENTERO an, CARACTER sex, REAL pe, REAL est) calcularIndiceMasaCorporal() calcularEdad() ~Persona()

Objeto 1 Objeto 2

Diagrama de objetos en UML

Objetos

<u>Un objeto es un ejemplar creado de una Clase.</u> Cada vez

que se construye un objeto de una clase, se crea una instancia de esa clase

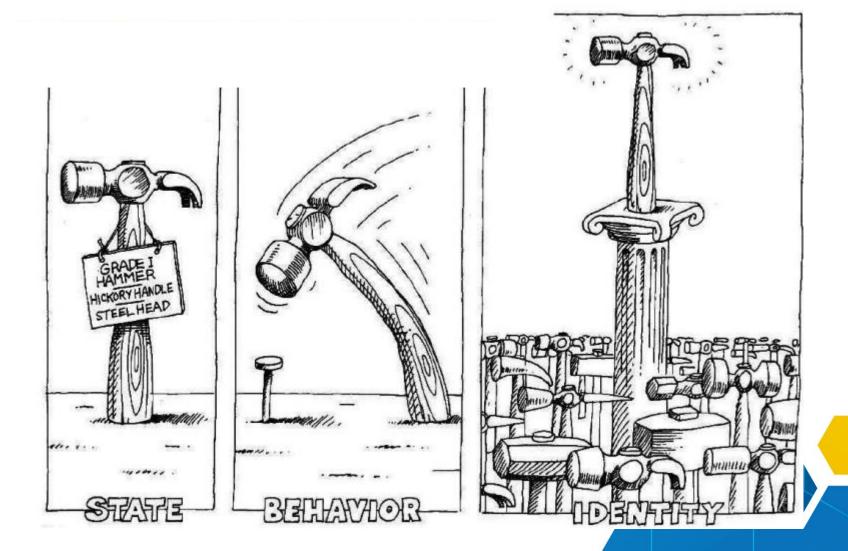






Conceptos de la POO – Clases y Objetos

Cada que un objeto es instanciado tiene una identidad, un estado y un comportamiento.



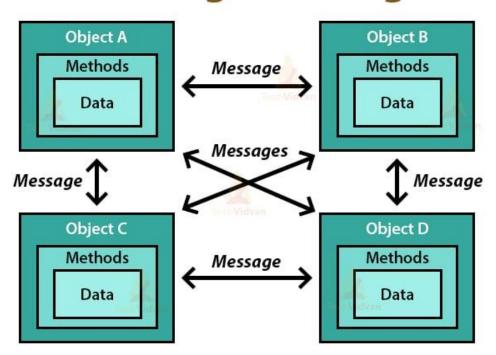






Conceptos de la POO – Mensaje y Método

Message Passing



Un **método** es el procedimiento o funcion que se invoca para actuar sobre un objeto.

Un **mensaje** es la indicación de la acción que debe hacer un objeto, es decir, la ejecución de uno de sus métodos.









Leguaje unificado de modelado - UML



UML es una notación gráfica unificada para representar múltiples características sistemas desarrollados con el paradigma de orientación a objetos.

Captura de requisitos - Análisis - Diseño - Implementación - Pruebas









Leguaje unificado de modelado - UML

Vista Lógica Vista de casos Vista de Vista de concurrencia componentes de uso Vista de implantación

Vista de casos de uso: Diagramas de Casos de Uso

Vista lógica:

Estructura: Diagrama de clases y diagramas de objetos

Interacción: Diagrama de estado, secuencia, colaboración y actividad

Vista de implantación: Diagrama de despliegue

Vista de componentes: Diagrama de componentes

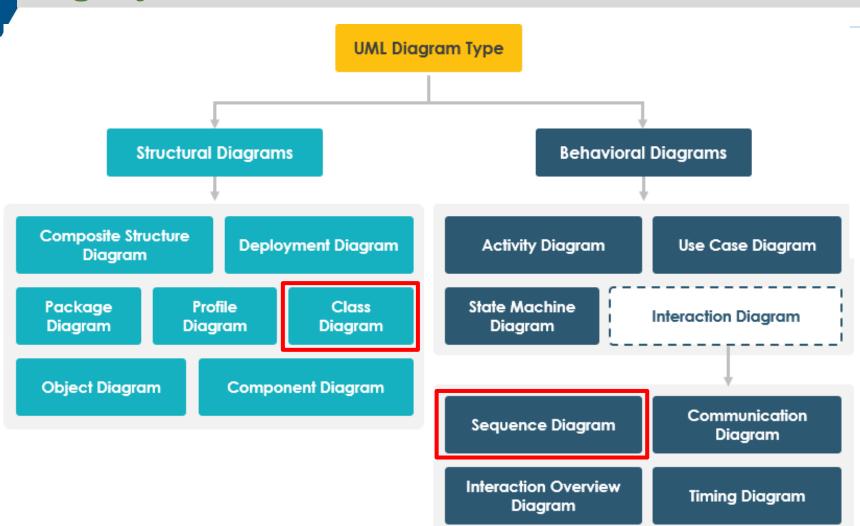
Vista de concurrencia:

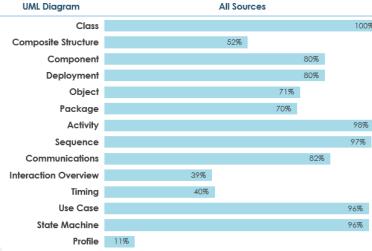
Vista lógica + Vista de componentes + Vista de implantación





Leguaje unificado de modelado - UML





ampliamente utilizado, si es \geq 60% de las fuentes escasamente utilizado si es \leq 40% de las fuentes

https://www.visual-paradigm.com/







Diagrama de clase - UML

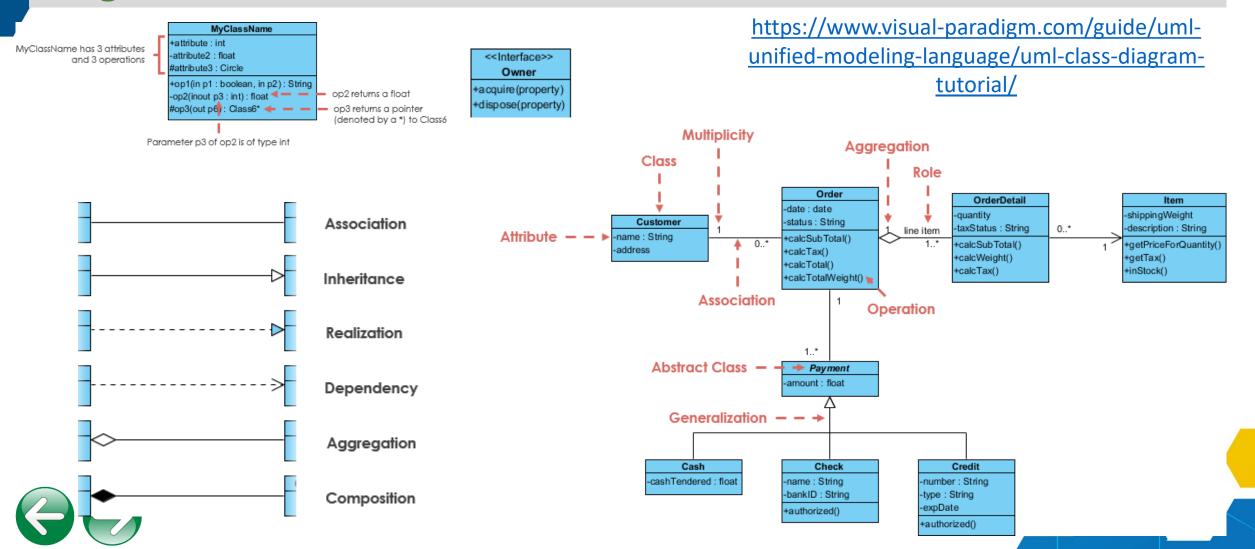
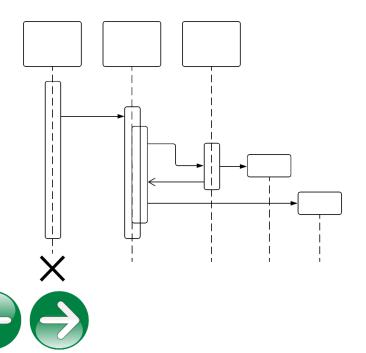


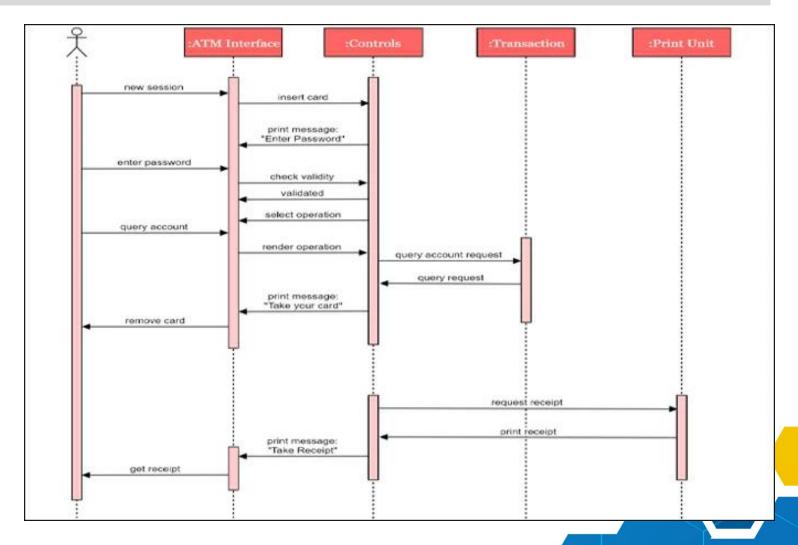




Diagrama de secuencia - UML

El diagrama de secuencia modela la colaboración de objetos basándose en una secuencia de tiempo. Muestra cómo los objetos interactúan con otros en un escenario particular de un caso de uso.









Java es un <u>lenguaje de programación orientado a objetos</u> (POO) independiente de la plataforma.

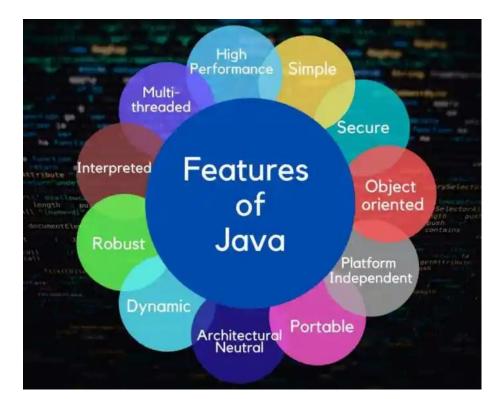
Debido a su confiabilidad y facilidad de uso, Java es uno de los lenguajes de programación más populares del mundo.

Es probable que sea uno de los primeros lenguajes de programación que encontrará como aspirante a desarrollador de aplicaciones.

Principales usos 2023:

- Creación y ejecución de aplicaciones móviles.
- Creación y ampliación de aplicaciones en la nube
- ✓ Desarrollar chatbots y otras herramientas de marketing.
- ✓ Impulsando aplicaciones web de nivel empresarial
- ✓ Compatible con dispositivos de inteligencia artificial (IA) e Internet de las cosas (IoT)





https://dailyjavaconcept.com/features-of-java/







Top Java Applications



The Mars Rover



Google













Netflix Minecraft

Uber











Spotify

eBay







CashApp Signal

NASA WorldWind

https://hackr.io/blog/java-application





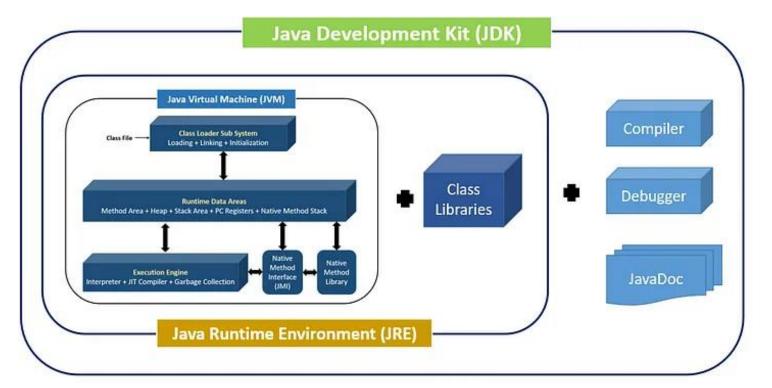


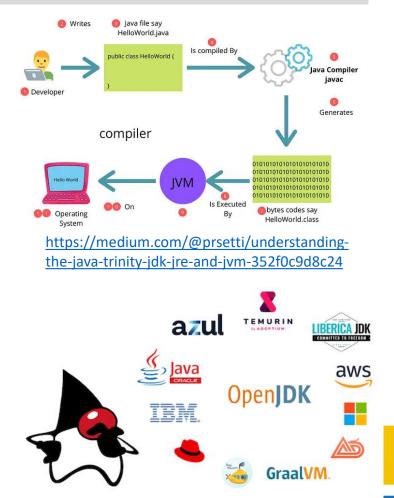




Programación con Java - Instalación

JDK o JRE o JVM?







https://sdkman.io/jdks





Programación con Java - Instalación

Verificación de la instalación de Java

C:\Users\Your Name>java -version

java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)

Pasos para la instalación:

- 1. Descargar e instalar JDK
- 2. Configurar variable de entorno

https://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK HowTo.html



Entornos de desarrollo IDE















Programación con Java – Compilación, Ejecución y Empaquetado

```
public class HolaMundo {

public static void main(String arg[]){

System.out.println("Hola Mundo en Java");

System.out.println("Hola Mundo en Java");

}
```

C:\example>javac HolaMundo.java

-> Compila

C:\example>java HolaMundo Hola Mundo en Java

-> Ejecuta

temp.mf

+

Main-Class: HolaMundo

Sealed: true

```
C:\example>jar cf HolaMundo.jar HolaMundo.class -> Empaqueta

C:\example>jar cmf temp.mf HolaMundo.jar HolaMundo.class -> Enlaza manifiesto

C:\example>java -jar HolaMundo.jar
Hola Mundo en Java -> Ejecuta
```







IntellijIdea

Programación con Java – Compilación, Ejecución y Empaquetado

```
Netbeans
                   public class HolaMundo {
             13
                        * @param args the command line arguments
             16
                       public static void main(String[] args) -
             17
              18
                           System.out.println("Hola Mundo en Java co
             19
             21
Hola Mundo en Java con netbeans
BUILD SUCCESSFUL (total time: 0 seconds)
```





Programación con Java

Sintaxis básica

- 1. Variables y tipos de datos
- 2. Operadores y expresiones
- 3. Estructuras de control
- 4. Métodos
- 5. Arreglos

POO en Java

- 1. Clases y objetos
- 2. Clases de api de Java
- 3. Asociación, agregación y composición
- 4. Herencia y polimorfismo
- 5. Interfaces
- 6. Colecciones
- 7. Lambdas
- 8. Record









Variables y tipos de datos

tipo	descripción	rango de valores
boolean	valor lógico	true o false
char	carácter unicode (16 bits)	los caracteres internacionales
byte	entero de 8 bits con signo	-128127
short	entero de 16 bits con signo	-3276832767
int	entero de 32 bits con signo	-2.147.483.648 2.147.483.647
long	entero de 64 bits con signo	aprox. 9.0*10 ¹⁸
float	nº real de 32 bits	unos 6 dígitos
double	nº real de 64 bits	unos 15 dígitos

- Java es un lenguaje fuertemente tipado
- Java implementa la inferencia de tipos
- Java permite la conversión de tipos (
- Java implementa clases envoltorio (Wrapper) para cada tipo
- Autoboxing Unboxing









Operadores y expresiones

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
=	Operador asignación	n = 4	n vale 4

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
_	operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 - 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2 7 / 2	0.25 3
ક	Resto de la división entera	20 % 7	6

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	7 == 38	false
!=	distinto que	'a' != 'k'	true
<	menor que	'G' < 'B'	false
>	mayor que	'b' > 'a'	true
<=	menor o igual que	7.5 <= 7.38	false
>=	mayor o igual que	38 >= 7	true

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	!false !(5==5)	true false
I	Suma lógica – OR (binario)	true false (5==5) (5<4)	true true
^	Suma lógica exclusiva – XOR (binario)	true ^ false (5==5) (5<4)	true true
&	Producto lógico – AND (binario)	true & false (5==5)&(5<4)	false false
11	Suma lógica con cortocircuito: si el primer operando es true entonces el segundo se salta y el resultado es true	true false (5==5) (5<4)	true true
3.3	Producto lógico con cortocircuito: si el primer operando es false entonces el segundo se salta y el resultado es false	false && true (5==5)&&(5<4)	false false

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
++	Incremento i++ primero se utiliza la variable y luego se incrementa su valor ++i primero se incrementa el valor de la variable y luego se utiliza	4++ a=5; b=a++; a=5; b=++a;	a vale 6 y b vale 5 a vale 6 y b vale 6
	decremento	4	3

Descripción





Ejemplo

Resultado del

Programación con Java – Sintaxis básica

Operadores y expresiones

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
=	Producto combinado	a=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
?:	operador condicional	a = 4; b = a == 4 ? a+5 : 6-a; b = a > 4 ? a*7 : a+8;	b vale 9 b vale 12



- 1				ejempio
			expresión	
	~	Negación ó complemento binario (unario)	~12	-13
	1	Suma lógica binaria – OR (binario)	12 10	8
	^	Suma lógica exclusiva – XOR (binario)	12^10	6
	&	Producto lógico binario – AND (binario)	12&10	14
	<<	Desplaza a la izquierda los bits del 1º operando tantas veces como indica el 2º operando (por la derecha siempre entra un cero)	7<<2 -7<<2	28 -28
	>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando (por la izquierda entra siempre el mismo bit más significativo anterior)	7>>2 -7>>2	1 -2



Operador

AND

A	В	Output	
0	0	0	
0	1	0	
1	0	0	
1	1	1	



OR

A	В	Output
0	0	0
0	1	1
1	0	1
1	1	1



XOR

A	В	Output
0	0	0
0	1	1
1	0	1
1	1	0





Operadores y expresiones

```
int x = 4;
int y = 5;
int z = 10;
int total = 12;
```

```
int x = 4, y = 5, z = 10;
int total = x + y * z;
```

```
int x = 4, y = 5, z = 10;
int total = z / (x * y);

System.out.println("The total is " + total + ".");
```

```
double volume = 3.14 * radius * radius * height * 1 / 3;
```

- Los operadores aritméticos tiene un orden de aplicación.
- En una expresión, java asume que el tipo de dato resultado, es el tipo mas grande.









Estructuras de control - condicional



```
if (condición){
  // código
}
```



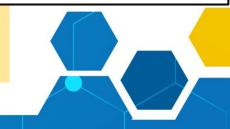
```
Falso
         Verdadero
                      Condición
Bloque de instrucciones 1
                                 Bloque de instrucciones 2
        if (condición){
               // código
         else{
              // código
```

```
switch (valor) {
  case value1:
  // secuencia de sentencias.
   break;
  case value2:
  // secuencia de sentencias.
   break;
  case valueN:
  // secuencia de sentencias.
  break;
  default:
  // secuencia de sentencias.
```

```
return switch (num % 2) {
    case 0:
        yield "par";
    case 1:
        yield "impar";
    default:
        yield "uknnow";
};

return switch(5%2){
    case 0->"Es par";
```

```
return switch(5%2) {
    case 0->"Es par";
    case 1->"Es impar";
    default->"desconocido";
};
```

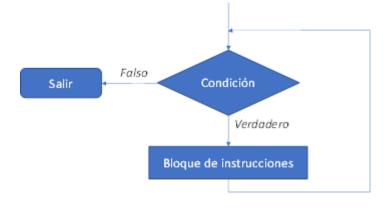


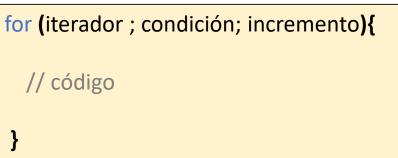




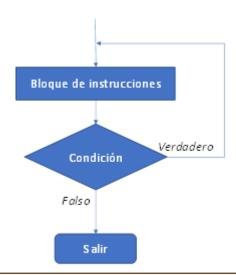
Estructuras de control - ciclos











```
do {

// código

} while ( condición );
```







Métodos

```
Declaración de métodos

public tipo_de_dato_devuelto nombre_metodo(tipo_de_dato parametro1, tipo_de_dato parametro2) {
    // código a ejecutar
    return resultado;
}
```

Invocación de métodos

```
public int multiplicar(int a, int b) {
    return a * b;
}

public static void main(String[] args) {
    int resultado = multiplicar(4, 5);
    System.out.println(resultado);
}
```

- Sin retorno, sin parámetros:

```
public void saludar() {
    System.out.println("¡Hola, mundo!");
}
```

- Con retorno, con parámetros:

```
public int sumar(int a, int b) {
    return a + b;
}
```

- Sin retorno, con parámetros:

```
public void saludar(String nombre) {
    System.out.println(";Hola, " + nombre + "!");
}
```

Con retorno, sin parámetros:

```
public static String obtenerNombreUsuario() {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Por favor, ingresa tu nombre: ");
    String nombre = scanner.nextLine();

    // Cerramos el scanner para liberar recursos
    scanner.close();

    return nombre;
}
```

- Métodos estáticos:

```
public static void mostrarMensaje() {
    System.out.println("¡Hola, soy un ejemplo de método estático!");
}
```







Arrays

Declaración de array unidimensional

```
tipo_dato nombre_array[];
nombre_array = new tipo_dato[tamanio];
```

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

- El primer elemento de un array esta en el índice 0.
- El ultimo elemento en el índice (tamaño-1)
- El tamaño de un array esta dado por la propiedad lenght
- Se pueden inicializar mediante un recorrido, o utilizando un iniciador
- Se pueden definir métodos que retornen y/o reciban arrays
- Una matriz puede que no sea cuadrada (esta compuesta por multiples arrays)

Declaración de array multidimensional

```
tipo_dato nombre_array[][];
nombre array = new tipo dato[tamanio] [tamanio];
```



char abecedario [] [];
abecedario = new char[3] [10];







Programación con Java

Sintaxis básica

- 1. Variables y tipos de datos
- 2. Operadores y expresiones
- 3. Estructuras de control
- 4. Métodos
- 5. Arreglos

POO en Java

- 1. Clases y objetos
- 2. Clases de api de Java
- 3. Asociación, agregación y composición
- 4. Herencia y polimorfismo
- 5. Interfaces
- 6. Colecciones
- 7. Lambdas
- 8. Record









Clase

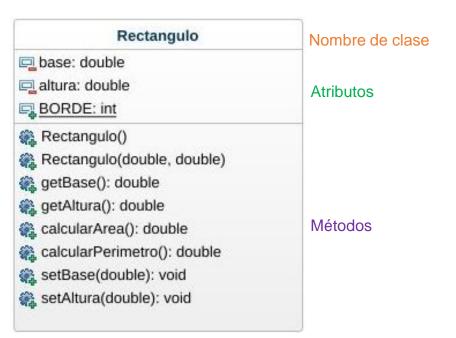


Diagrama de clase UML



```
public class Rectangulo {
                                                           public class [name]{
           private double base;
           private double altura;
           private static final int BORDE = 10;
           public Rectangulo() { }
10
           public Rectangulo(double base, double altura) {
12
                this.base = base;
13
                this.altura = altura;
15
           public double getBase() { return base;}
           public void setBase(double base) {this.base = base;}
18
           public double getAltura() {return altura;}
19
20
21
           public void setAltura(double altura) { this.altura = altura;}
22
           public double calcularArea() {
23
24
               return this.base*this.altura;
26
           public double calcularPerimetro() {
27
               return this.base*2 + this.altura*2;
28
29
```





```
public class Rectangulo {
    private double base;
    private double altura;
    private static final int BORDE = 10;
   public Rectangulo() { }
    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    public double getBase() { return base;}
   public void setBase(double base) {this.base = base;}
    public double getAltura() {return altura;}
    public void setAltura(double altura) { this.altura = altura;}
    public double calcularArea() {
        return this.base*this.altura;
    public double calcularPerimetro() {
        return this.base*2 + this.altura*2;
```

Objetos y Constructores

```
Rectangulo r = new Rectangulo();
r.setBase(20);
r.setAltura(40);
System.out.println( r.getAltura() );
System.out.println( r.getBase() );
System.out.println( r.calcularArea() );
System.out.println( r.calcularPerimetro() );
System.out.println( Rectangulo.BORDE );
```



30



Programación con Java – Programación O.O

```
public class Rectangulo {
    private double base;
    private double altura;
    private static final int BORDE = 10;
   public Rectangulo() { }
    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    public double getBase() { return base;}
   public void setBase(double base) {this.base = base;}
    public double getAltura() {return altura;}
    public void setAltura(double altura) { this.altura = altura;}
    public double calcularArea() {
        return this.base*this.altura;
    public double calcularPerimetro() {
        return this.base*2 + this.altura*2;
```

Objetos y Constructores

```
Rectangulo r = new Rectangulo(20,10);

System.out.println(r.getAltura());

System.out.println(r.getBase());

System.out.println(r.calcularArea());

System.out.println(r.calcularPerimetro());

System.out.println(Rectangulo.BORDE);
```







```
public class Rectangulo {
           private double base;
           private double altura;
           private static final int BORDE = 10;
           public Rectangulo() { }
           public Rectangulo(double base, double altura) {
                this.base = base;
               this.altura = altura;
           public double getBase() { return base;}
           public void setBase(double base) {this.base = base;}
           public double getAltura() {return altura;}
           public void setAltura(double altura) { this.altura = altura;}
           public double calcularArea() {
               return this.base*this.altura;
           public double calcularPerimetro() {
                return this.base*2 + this.altura*2;
30
```

Atributos de instancia

```
private double base;
private double altura;
```

```
Rectangulo r = new Rectangulo();

System.out.println(r.getAltura());

System.out.println(r.getBase());
```

```
Rectangulo c = new Rectangulo();

System.out.println(c.calcularPerimetro());

System.out.println(c.getBase());
```



30



Programación con Java – Programación O.O

```
public class Rectangulo {
    private double base;
    private double altura;
    private static final int BORDE = 10;
   public Rectangulo() { }
    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    public double getBase() { return base;}
   public void setBase(double base) {this.base = base;}
   public double getAltura() {return altura;}
    public void setAltura(double altura) { this.altura = altura;}
    public double calcularArea() {
        return this.base*this.altura;
    public double calcularPerimetro() {
        return this.base*2 + this.altura*2;
```

Atributos de clase (static)

```
public static final int BORDE = 10;
System.out.println( Rectangulo.BORDE );
```

```
private static int BORDE = 10;

public static int getBorde(){
    return Rectangulo.BORDE;
}

public static void setBorde(int borde){
    Rectangulo.BORDE = borde;
}
```

System.out.println(Rectangulo.getBorde());





```
public class Rectangulo {
    private double base;
    private double altura;
    private static final int BORDE = 10;
   public Rectangulo() { }
    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    public double getBase() { return base;}
    public void setBase(double base) {this.base = base;}
    public double getAltura() {return altura;}
    public void setAltura(double altura) { this.altura = altura;}
    public double calcularArea() {
        return this.base*this.altura;
    public double calcularPerimetro() {
        return this.base*2 + this.altura*2;
```

Sobrecarga de métodos (overload)

```
public Rectangulo ( ){
}

Rectangulo r = new Rectangulo();
```

```
public Rectangulo (double base, double altura){
    this.base = base;
    this.altura = altura;
}
```

Rectangulo p = new Rectangulo(10,5);





```
public class Rectangulo {
    private double base;
    private double altura;
    private static final int BORDE = 10;
   public Rectangulo() { }
    public Rectangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    public double getBase() { return base;}
    public void setBase(double base) {this.base = base;}
    public double getAltura() {return altura;}
    public void setAltura(double altura) { this.altura = altura;}
    public double calcularArea() {
        return this.base*this.altura;
    public double calcularPerimetro() {
        return this.base*2 + this.altura*2;
```

Sobrecarga de métodos (overload)

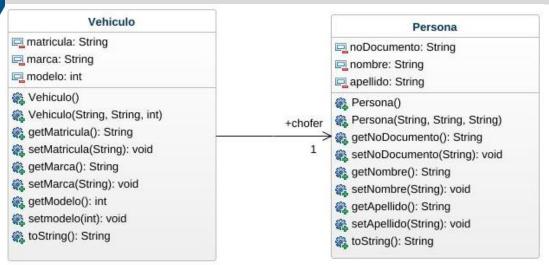
```
public double sumaArea (Rectangulo r ){
   return this.calcularArea() + r.calcularArea();
}
```

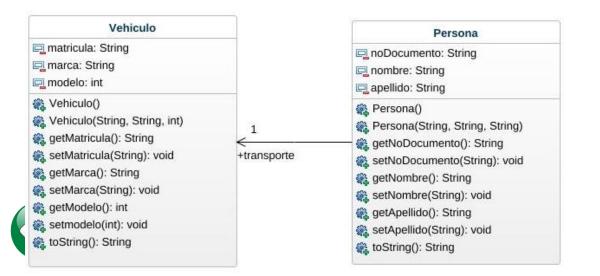
```
public double sumaArea (double b, double a ){
   Rectangulo r = new Rectangulo(b,a);
   return this.sumaArea(r);
}
```

```
public double sumaArea (double b){
   return this.sumaArea(b,0);
}
```









```
public class Vehiculo {
    private String matricula;
    private String marca;
    private String modelo;
    private Persona chofer;
    //
}
```

```
public class Persona {
    private String noDocumento;
    private String nombre;
    private String apellido;
    private Vehiculo transporte;
    //
}
```





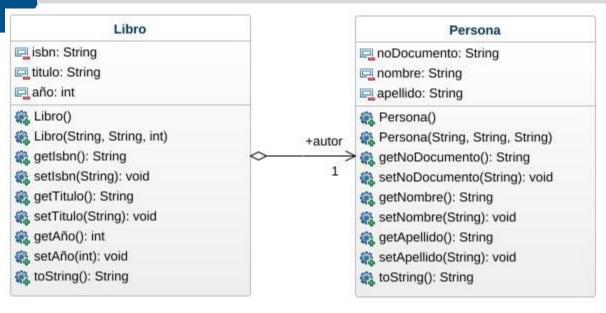
Vehiculo Persona matricula: String noDocumento: String marca: String nombre: String modelo: int apellido: String Wehiculo() Rersona() Wehiculo (String, String, int) +pasajeros 🏟 Persona(String, String, String) getMatricula(): String getNoDocumento(): String setMatricula(String): void +transporte a setNoDocumento(String): void getMarca(): String a getNombre(): String setMarca(String): void a setNombre(String): void getModelo(): int a getApellido(): String setmodelo(int): void a setApellido(String): void toString(): String toString(): String

```
public class Vehiculo {
         private String matricula;
         private String marca;
         private String modelo;
         private Persona pasajeros[];
public class Persona {
         private String noDocumento;
         private String nombre;
         private String apellido;
         private Vehiculo transporte;
```









```
public class Persona {
    private String noDocumento;
    private String nombre;
    private String apellido;
    //
}
```

```
public class Libro {
         private String isbn;
         private String titulo;
         private int año;
         private Persona autor;
         public Libro(String i, String t, int y, Persona a){
             this.autor = a;
```





```
public class Libro {
          private String isbn;
          private String titulo;
          private int año;
          private Persona autor;
          //
          public Libro(String i, String t, int y, Persona a){
                //
                this.autor = a;
          }
}
```

```
Persona p = new Persona("123", "Luis", "Perez");

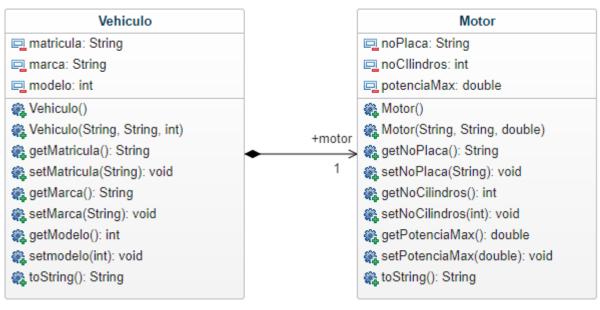
Libro libroA = new Libro("xxx", "POO", 2020, p);

Libro libroB = new Libro("yyy", "PD", 2023, p);
```









```
public class Motor {
    private String noPlaca;
    private int noCilindros;
    private double potenciaMax;
    //
}
```

```
public class Vehiculo {
          private String matricula;
          private String marca;
          private int modelo;
          private Motor motor;
          public Vehiculo (String mat, String marca, int
          mod, String nPlaca, int noCil, double pot){
            this.motor = new Motor(nPlaca, noCil, pot);
```







```
public class Vehiculo {
           private String matricula;
           private String marca;
           private int modelo;
           private Motor motor;
           public Vehiculo (String mat, String marca, int
           mod, String nPlaca, int noCil, double pot){
             this.motor = new Motor(nPlaca, noCil, pot);
public class Motor {
           private String noPlaca;
           private int noCilindros;
           private double potenciaMax;
```

```
Vehiculo v = new Vehiculo("123", "Toyota", 2020, "XBC-1", 4, 1500.8);
```







EmpleadoBasico

nHorasLaboradas: int

valorHora: double

EmpleadoBasico()

EmpleadoBasico(int, double)

getNHorasLaboradas(): int

getValorHora(): double

🚓 setNHorasLaboradas(int): void

setValorHora(double): void

🖚 calcularSalario(): double

EmpleadoPorComision

porcentajeComision: float

ventasRealizadas: double

EmpleadoPorComision()

EmpleadoPorComision(int, double, float, double)

a calcularComision(): double

a calcularSalario(): double

a getPorcentajeComision(): float

getVentasRealizadas(): double

a setPorcentajeComision(float): void

🐔 setVentasRealizadas(double): void

Herencia entre Clases

```
public class EmpleadoBasico {
  private int nHorasLaboradas;
   private double valorHora;
  public EmpleadoBasico(){ }
  public EmpleadoBasico(int h, double v){
     this.nHorasLaborradas = h;
     this.valorHora = v;
  // getter y setter
  public double calcularSalario(){
     return this.nHorasLaborada * this.valorHora;
```

EmpleadoBasico eb = new EmpleadoBasico(40, 30000.5);

System.out.println(eb.calcularSalario());

Super clase









EmpleadoBasico

nHorasLaboradas: int

valorHora: double

EmpleadoBasico()

EmpleadoBasico(int, double)

getNHorasLaboradas(): int

getValorHora(): double

a setNHorasLaboradas(int): void

a setValorHora(double): void

a calcularSalario(): double

EmpleadoPorComision

porcentajeComision: float

ventasRealizadas: double

EmpleadoPorComision()

EmpleadoPorComision(int, double, float, double)

calcularComision(): double

acalcularSalario(): double

getPorcentajeComision(): float

getVentasRealizadas(): double

a setPorcentajeComision(float): void

setVentasRealizadas(double): void

Herencia entre Clases

```
public class EmpleadoPorComision extends EmpleadoBasico {
    private float porcentajeComision;
    private double ventasRealizadas;
    public EmpleadoPorComision(){
    public EmpleadoPorComision(int h, double p, float c, double v){
       super(h,p);
                                                  EmpleadoPorComision ec;
                                                 ec = new EmpleadoPorComision(40, 30000.5, 0.1,7000000);
    // getter y setter
     public double calcularComision(){
        return this.porcentajeComision * this.ventasRealizadas;
      @Override
      public double calcularSalario(){
       return this.calcularComision() + super.calcularSalario();
```

```
public class EmpleadoBasico {
  private int nHorasLaboradas;
  private double valorHora;
  public EmpleadoBasico(){ }
  public EmpleadoBasico(int h, double v){
     this.nHorasLaborradas = h;
     this.valorHora = v;
  // getter y setter
  public double calcularSalario(){
    return this.nHorasLaborada * this.valorHora;
```

System.out.println(ec.calcularSalario());



Clase derivada



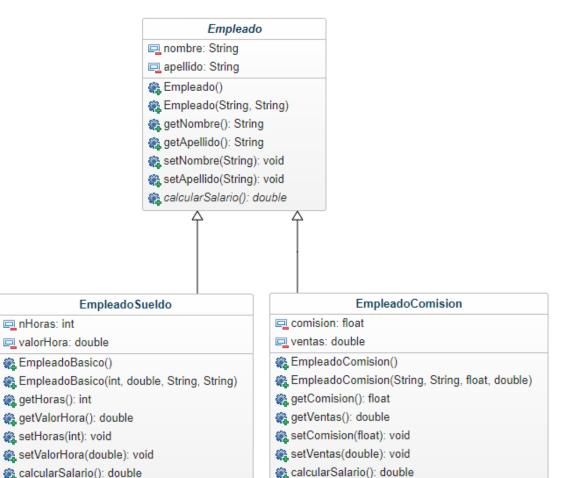


nHoras: int



Programación con Java – Programación O.O

Clases abstractas y Polimorfismo



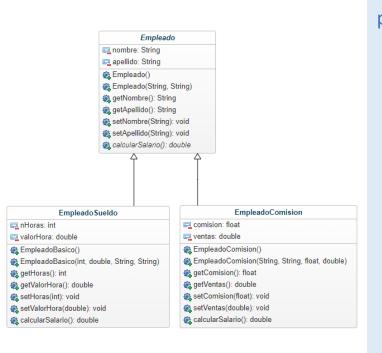
```
public abstract class Empleado {
    private String nombre;
    private String apellido;
    public Empleado ( ){
    public Empleado ( String nombre, String apellido){
       this.nombre = nombre;
       this.apellido = apellido;
     public String getNombre(){
        return this.nombre;
     // getter y setter
     public asbtract double calcularSalario();
```

Superclase Abstracta





Clases abstractas y Polimorfismo



```
public class EmpleadoSueldo extends Empleado {
    private int nHoras;
    private double valorHora;

public EmpleadoSueldo(){ }
    public EmpleadoSueldo(int h, double v, String n, String a){
        super(n,a);
        this.nHoras = h;
        this.valorHora = v;
        Empleado e = new EmpleadoS
```

```
public abstract class Empleado {
    private String nombre;
    private String apellido;

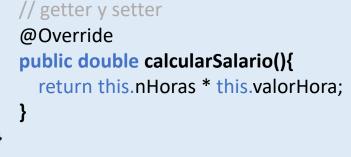
    public Empleado ( ){
    }
    public Empleado ( String nombre, String apellido){
        this.nombre = nombre;
        this.apellido = apellido;
    }

    public String getNombre(){
        return this.nombre;
    }
    // getter y setter

    public asbtract double calcularSalario();
}
```

Empleado e = new EmpleadoSueldo(40, 30000.5, "Pedro", "Paz");

System.out.println(e.calcularSalario());





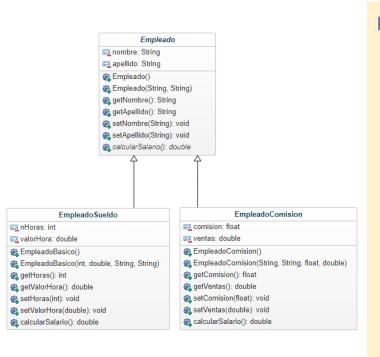








Clases abstractas y Polimorfismo



```
public class EmpleadoComision extends Empleado {
   private float comision;
   private double ventas;

public EmpleadoComision(){ }
   public EmpleadoComision(float c, double v, String n, String a){
        super(n,a);
        this.comision = c;
        this.ventas = v;
        Fmpleado e = new EmpleadoComis
```

```
public abstract class Empleado {
    private String nombre;
    private String apellido;

    public Empleado ( ){
    }
    public Empleado ( String nombre, String apellido){
        this.nombre = nombre;
        this.apellido = apellido;
    }

    public String getNombre(){
        return this.nombre;
    }
    // getter y setter

    public asbtract double calcularSalario();
}
```

Empleado e = new EmpleadoComision(40, 30000.5, "Pedro", "Paz");

System.out.println(e.calcularSalario());



Clase derivada

public double calcularSalario(){

return this.nHoras * this.valorHora;

// getter y setter

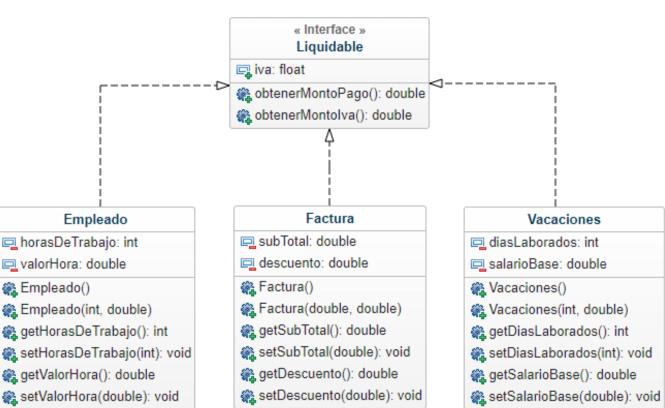
@Override







Interfaces y Polimorfismo



```
public interface Liquidable {
    float IVA=0.16;
    double obtenerMontoPago();
    double obtenerMontolva();
}
```

Liquidable objetoLiquidable;

Interface

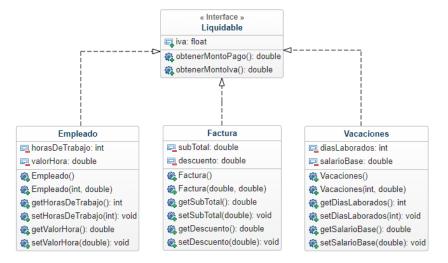








Interfaces y Polimorfismo



```
public interface Liquidable {
  float IVA=0.16;
  double obtenerMontoPago();
  double obtenerMontolva();
```

```
public class Empleado implements Liquidable {
   private int horasDeTrabajo;
   private double valorHora;
                            Liquidable objeto=new Empleado(8, 12000);
    public Empleado ( ){
                                 System.out.println( objeto.obtenerMontoPago() );
    public Empleado ( int h, int v){
      this.horasDeTrabajo = h;
                                 System.out.println(objeto.obtenerMontolva());
      this.valorHora = v;
    // getter y setter
    @Override
    public double obtenerMontoPago(){
      return this.nHoras * this.valorHora;
   @Override
    public double obtenerMontolva(){
      return Liquidable.IVA * this.obtenerMontoPago();
        Implementación de interface
```







Interfaces y Polimorfismo

```
« Interface »
                                           Liquidable
                                    强 obtenerMontoPago(): double
                                   🙉 obtenerMontolva(): double
        Empleado
                                            Factura
                                                                              Vacaciones
horasDeTrabajo: int
                                   subTotal: double
                                                                     diasLaborados: int
valorHora: double
                                   descuento: double
                                                                     salarioBase: double
                                   Factura()
Empleado()
                                                                      Wacaciones()
Empleado(int, double)
                                   Ractura(double, double)
                                                                      Vacaciones(int, double)
getHorasDeTrabajo(): int
                                   getSubTotal(): double
                                                                      agetDiasLaborados(): int
                                   setSubTotal(double): void
setHorasDeTrabajo(int): void
                                                                      setDiasLaborados(int): void
getValorHora(): double
                                   getDescuento(): double
                                                                      getSalarioBase(): double
a setValorHora(double): void
                                   setDescuento(double): void
                                                                      🙊 setSalarioBase(double): void
```

```
public interface Liquidable {
  float IVA=0.16;
  double obtenerMontoPago();
  double obtenerMontolva();
```

```
public class Factura implements Liquidable {
   private double subTotal;
   private double descuento;
    public Factura ( ){
    public Factura ( double s, double d){
                                     Liquidable objeto=new Factura(1000, 100);
      this.subTotal = s;
      this.descuento = d;
                                 System.out.println( objeto.obtenerMontoPago() );
    // getter y setter
                                 System.out.println( objeto.obtenerMontolva() );
    @Override
    public double obtenerMontoPago(){
      return this.subTotal - this.descuento;
   @Override
    public double obtenerMontolva(){
         Implementación de interface
```

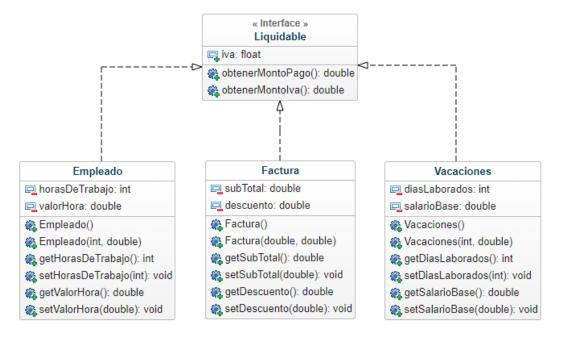






Código genérico

```
public class Main {
     public static void main(String[] arg){
           List<Liquidable> list = new ArrayList();
           list.add(new Empleado(30, 5000));
           list.add(new Factura(30000,5000));
           verValorLiquidado(list);
     public static double verValorLiquidado(List<Liquidable> list){
       for(Liquidable I : list){
           System.out.println("Valor= " + I.obtenerMontoPago());
           System.out.println("Iva= " + I.obtenerMontolva() );
```



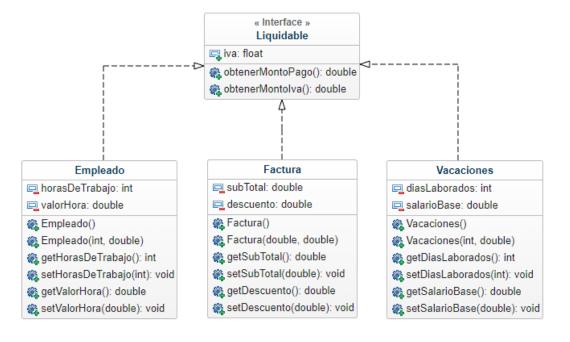






Código genérico

```
public class Main {
     public static void main(String[] arg){
           List<Liquidable> list = new ArrayList();
           list.add(new Empleado(30, 5000));
           list.add(new Factura(30000,5000));
           verValorLiquidado(list);
     public static double verValorLiquidado(List<Liquidable> list){
       for(Liquidable I : list){
           System.out.println("Valor= " + I.obtenerMontoPago());
           System.out.println("Iva= " + I.obtenerMontolva());
```



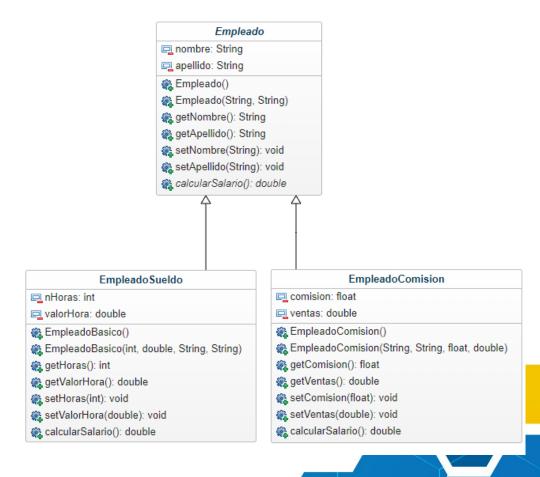






Código genérico

```
public class Main {
     public static void main(String[] arg){
           List<Empleado> list = new ArrayList();
           list.add(new EmpleadoSueldo(30, 5000, "Pipe", "Paz" ));
           list.add(new EmpleadoComision(0.3,100000, "Luis", "Diaz"));
           verNomina(list);
     public static double verNomina(List<Empleado> list){
       for(Empleado e : list){
           System.out.println("Nombre= " + e.getName() );
           System.out.println("Apellido= " + e.getApellido() );
           System.out.println("Salario= " + e.calcularSalario());
```







Colecciones - Listas

ArrayList

List <String> palabras=new ArrayList();

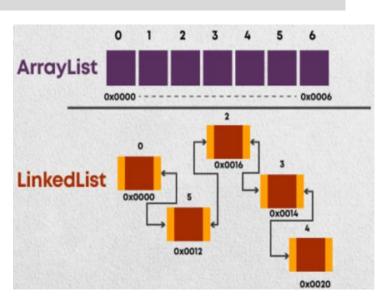
LinkedList

List <String> palabras=new LinkedList();

Métodos:

Object get (int index)
Object set (int index, Object element)
void add (int index, Object element)
Object remove (int index)





Operations	ArrayList	LinkedList
	Complexity	Complexity
get(int index)	O(1)	O(n/4)
add(E element)	O(1)->O(n)	O(1)
remove(int index)	O(n/2)	O(n/4)
add(int index, E element)	O(n/2)	O(n/4)
Iterator.remove()	O(n/2)	O(1)
ListIterator.add(E element)	O(n/2)	O(1)

Tiempo de complejidad





Colecciones - Mapas

HashMap

Map <int,String> palabras=new HashMap();

TreeMap

Map <int,String> palabras=new TreeMap();

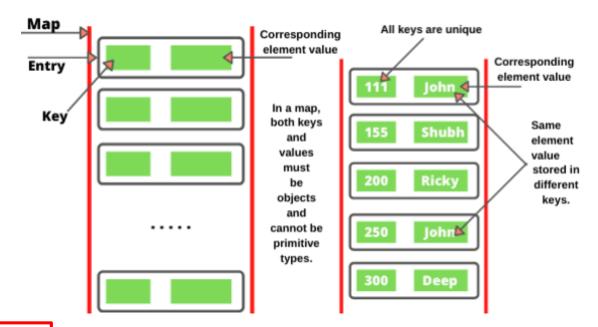
Métodos:

V put(K key, V value)

V remove(Object key)

V get(Object key)

int size()



https://www.scientecheasy.com/2020/10/map-in-java.html/









Colecciones - Conjuntos

HashSet

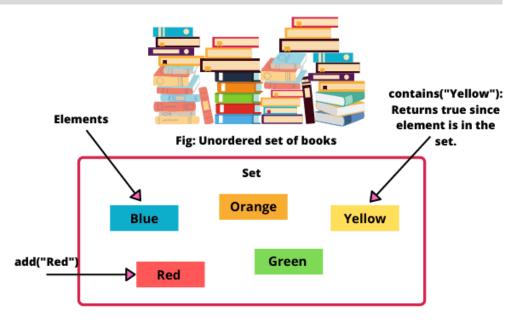
Set <String> palabras=new HashSet();

TreeSet

Map <String> palabras=new TreeSet();

Métodos:

boolean add(Object o)
boolean remove(Object o)
boolean isEmpty()
int size()



https://www.scientecheasy.com/2020/10/map-in-java.html/









Colecciones - API Java

Clases utilitarias

String
Scanner
LocalDate
LocalDateTime
Random
Math

Arreglos y Colecciones

Collections Arrays

Interfaces

Serializable Comparable Cloneable Comparator

Avanzado

Lambdas Streams Records









Recursos para aprender POO con Java

coursera

Introducción a la programación orientada a objetos en Java

Universidad de los Andes

Introducción a UML

Universidad de los Andes

ORACLE

Academy Ruta de aprendizaje Java





