MitoCode

JAVA BACKEND DEVELOPER

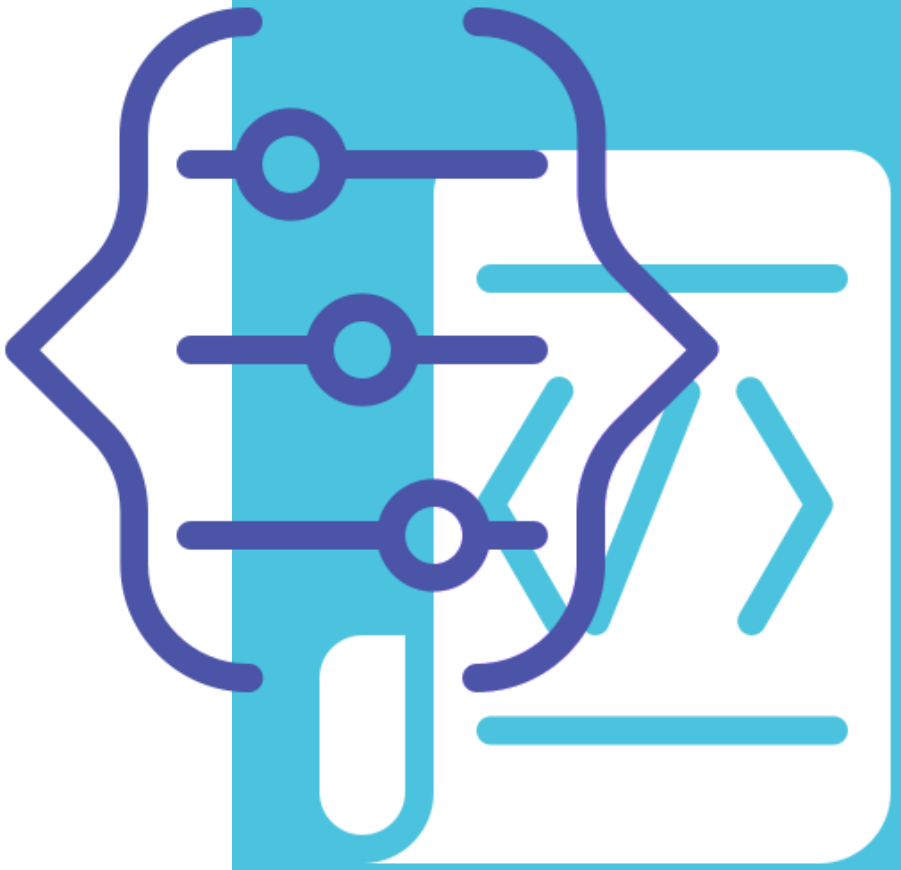
Profesor: Jaime Medina

Correo:
cursos@mitocodenetwork.com

www.mitocode.com

OBJETIVO

El objetivo principal es aplicar patrones de diseño para poder diseñar un backend API REST, terminarás conociendo y aplicando esto en un proyecto el cual te servirá para tu portafolio y futuras postulaciones laborales.



Carrera Java

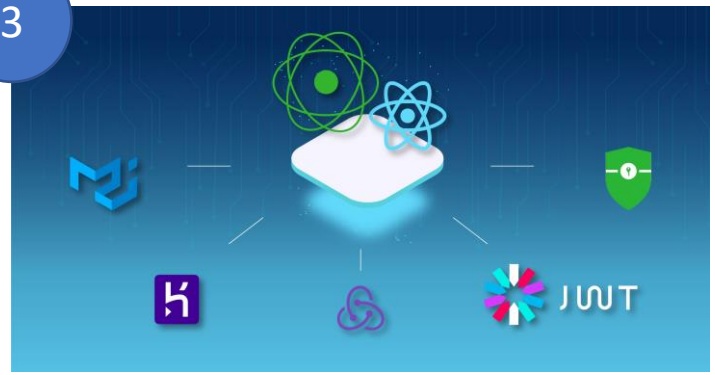
1

Java 11 Backend

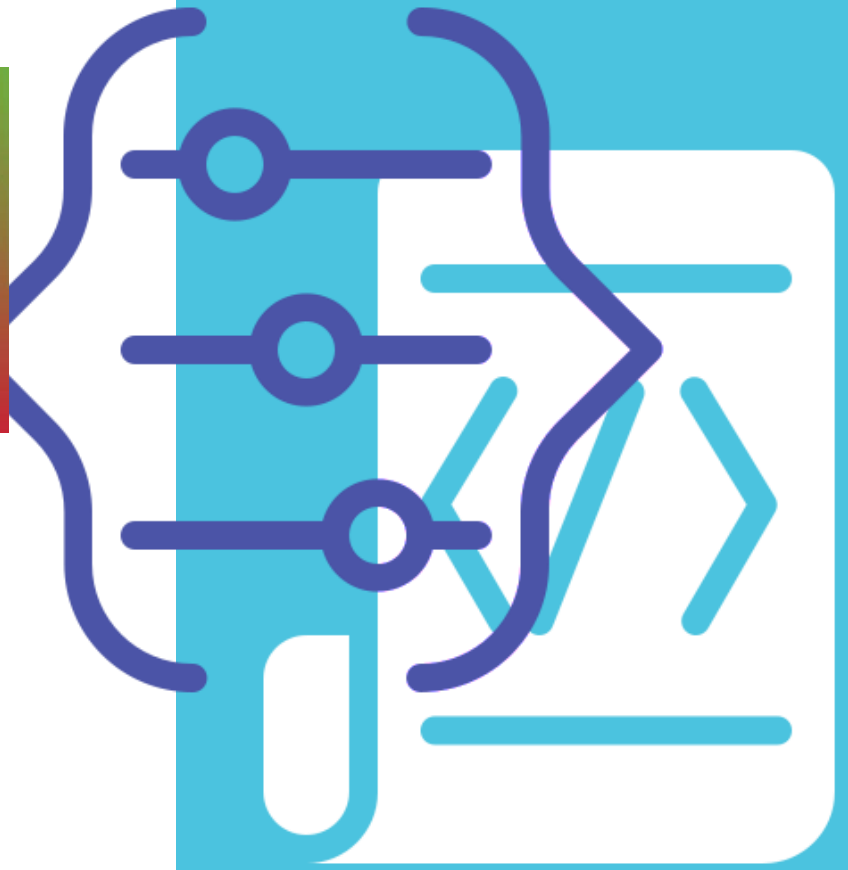
2

Java Full Stack

3



Spring Webflux & React.js



MitoCode

TEMARIO

- Programación Orientada a Objetos, Maven & Git
- Programación Funcional
- Spring Boot & API REST
- Spring Data JPA - Parte 1
- Spring Data JPA - Parte 2
- Spring Security
- Pruebas Unitarias: JUnit y Mock
- Integraciones con frameworks javascript (Angular, React.js & Vue.js)

MitoCode

PROGRAMACIÓN ORIENTADA A OBJETOS MAVEN & GIT

Profesor: Jaime Medina

Correo:
cursos@mitocodenetwork.com

www.mitocode.com

JAVA 11

Desde que Java cambió la forma en que va liberando versiones, cada 6 meses tenemos una nueva disponible, que incluye algunas funcionalidades. En la actualidad, Java 11 es una versión LTS (Long Term Support) siendo una de las versiones más adecuadas para usar en producción.

- Lenguaje de programación
- Maquina virtual / Entorno de ejecución
- Bibliotecas / API

ELIMINACIÓN DE MÓDULOS
JAVA EE Y COBRA

Se eliminan del JDK paquetes ya desaconsejados hace varias versiones anteriores y que no eran muy usador en cualquier caso. Estos paquetes son los de COBRA una forma de llamada a procedimientos que se utilizó como alternativa a RMI, pero nunca tuvo un uso extendido prefiriéndose SOAP o más recientemente interfaces REST.

La lista de paquetes eliminados son los siguientes:

- *java.xml.ws (JAX-WS,plus the related technologies SAAJ and Web Services Metadata*
- *java.xml.bind (JAXB)*
- *java.activation (JAF)*
- *java.xml.ws.annotation (Common Annotations)*
- *java.corba (COBRA)*
- *java-transaction (JTA)*
- *java.se.ee (Jaggregator module for the six modules above)*
- *jdk.xml.ws (Tools for JAX-WS)*
- *jdk.xml.bind (Tools for JAXB)*

SINTÁXIS DE VARIABLES LOCALES PARA PARÁMETROS EN LAMBDAS

Ahora los parámetros de una lambda pueden declararse con *var* con inferencia de tipos. Esto proporciona uniformidad en el lenguaje al declarar los parámetros, permite usar anotaciones en los parámetros de la función lambda como @NotNull.

Esta funcionalidad tiene algunas restricciones. No puede mezclar el uso y no uso de *var* y no se puede mezcla el uso de *var* y tipos de lambdas explícitas. Son consideradas ilegales por el compilador y producirá un error en tiempo de compilación.

```java
(x, y) -> x.process.(y)

(var x, var y) -> x.process.(y)

(@NotNull var x, @NotNull var y) -> x.process.(y)

(var x, y) -> x.process.(y) // no se puede mezclar var y no var
(var x, int y) -> x.process.(y) // no se puede mezclar var y tipos en lamdas explícitas
```

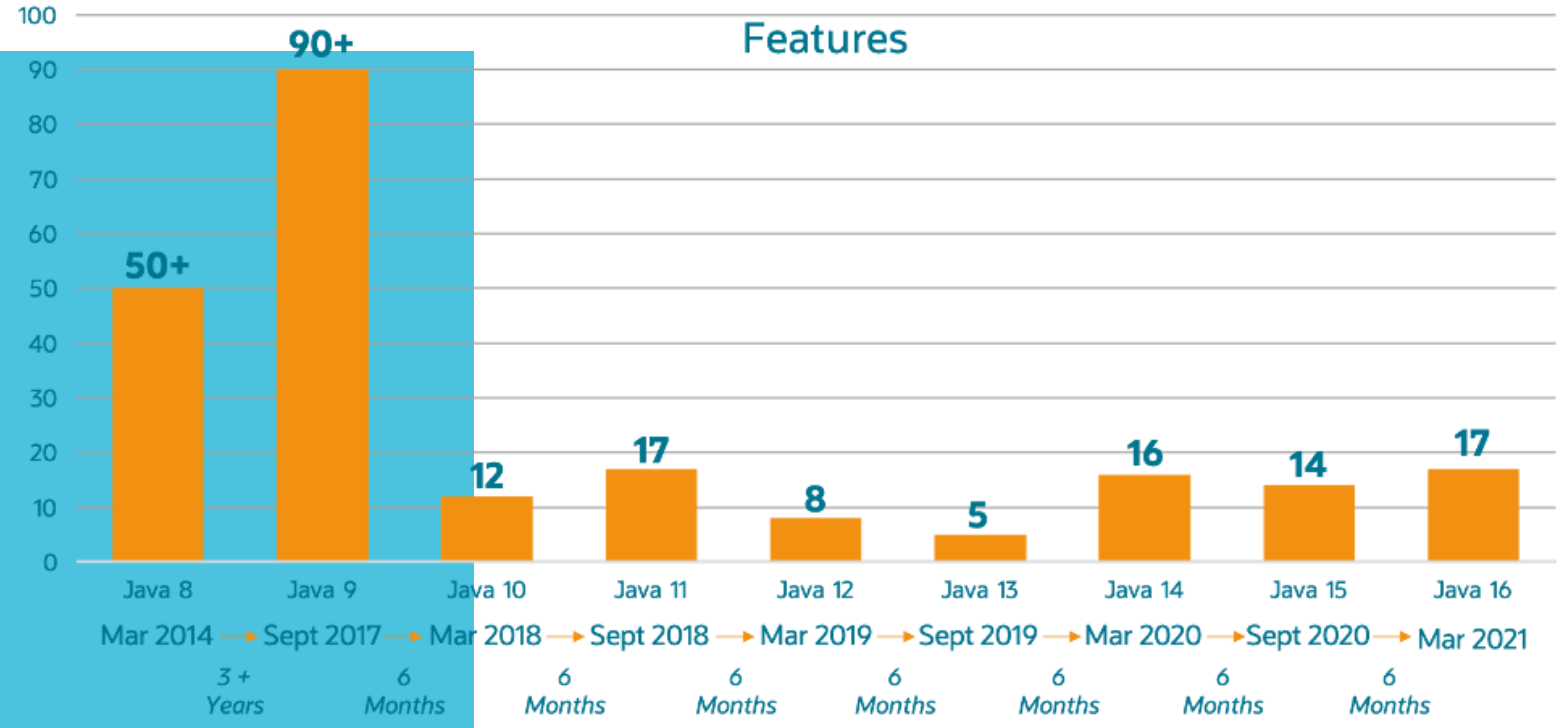
Lambda.java

CLIENTE HTTP

En Java 9 se incorporó de forma experimental un cliente HTTP con soporte para HTTP/2 en el Propio JDK. En Java 11 alcanza la categoría de estable. Este cliente HTTP es una forma sencilla de hacer llamadas a servicios web ya sean REST o **GraphQL**.

Las clases del nuevo cliente se encuentran en el paquete **Java.net.http.** Al estar este cliente HTTP incorporado en el JDK no será necesario depender de librerías de terceros.

NUEVO ESQUEMA
DE LICENCIAMIENTO DE JAVA



Java Backend Developer

MitoCode

Oracle Java SE Support Roadmap*†				
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**	**Sustaining Support**
7 (LTS)	July 2011	July 2019	July 2022*****	Indefinite
8 (LTS)**	March 2014	March 2022	December 2030*****	Indefinite
9 (non-LTS)	September 2017	March 2018	Not Available	Indefinite
10 (non-LTS)	March 2018	September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	September 2026	Indefinite
12 (non-LTS)	March 2019	September 2019	Not Available	Indefinite
13 (non-LTS)	September 2019	March 2020	Not Available	Indefinite
14 (non-LTS)	March 2020	September 2020	Not Available	Indefinite
15 (non-LTS)	September 2020	March 2021	Not Available	Indefinite
16 (non-LTS)	March 2021	September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 (non-LTS)	March 2022	September 2022	Not Available	Indefinite
19 (non-LTS)***	September 2022	March 2023	Not Available	Indefinite
20 (non-LTS)***	March 2023	September 2023	Not Available	Indefinite
21 (LTS)***	September 2023	September 2028	September 2031	Indefinite

Fuente: https://www.oracle.com/java/technologies/java-se-support-roadmap.html

MitoCode

NUEVO ESQUEMA
DE LICENCIAMIENTO DE JAVA

Version	Release date	End of Free Public Updates[1][5][6][7]	Extended Support Until
Java SE 11 (LTS)	September 2018	September 2027 for Zulu At least October 2024 for AdoptOpenJDK At least September 2027 for Amazon Corretto	September 2026, or September 2027 for e.g. Zulu[8]
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK, March 2023 for Zulu[8]	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	September 2030 for Zulu	TBA

Legend: Old version | Older version, still maintained | **Latest version** | Future release

NUEVO ESQUEMA DE LICENCIAMIENTO DE JAVA

- El cambio de licencia afectara a versiones superiores a Java 8u202
- Si se quiere utilizar Oracle JDK para Java 8 (superior a 8u202) o Java 11 en producción, es obligatorio pagar un soporte de $25 al mes
- Si se quiere utilizar Oracle JDK para Java 8 o Java 11 en desarrollo, no se paga nada
- Si no se quiere utilizar Oracle JDK las JVM son intercambiables con Open JDK ya que Oracle JDK es en si una compilación de Open JDK con soporte comercial de Oracle. Adicionalmente Oracle ya liberó sus características «premium» y están disponibles para todo el mundo
- Entre las ofertas de OpenJDK con soporte comercial (gratuito y pagado) encontramos a Oracle, Red Hat, AdoptOpenJDK/IBM, Azul Systems, Amazon, Liberica JDK, entre otros.

ALTERNATIVAS
A JDK

OpenJDK

Amazon Corretto

AdoptOpenJDK

THE TWELVE FACTOR APP

"The twelve-factor app" es una metodología para construir aplicaciones. Lo podríamos ver como un conjunto de prácticas de desarrollo y arquitectura, que presta especial atención a las dinámicas del escalamiento y cubre los puntos fundamentales que debería cumplir el desarrollo de una aplicación preparada para ser distribuida como un servicio. Se compone de doce reglas https://12factor.net/es/

¿Quién debería leer este documento?

Cualquier desarrollador que construya aplicaciones y las ejecute como un servicio. Ingenieros de operaciones que despliegen y gestionen dichas aplicaciones.

¿Cual es su objetivo?

Ofrecer un conjunto de soluciones conceptualmente robustas para esos problemas acompañados de su correspondiente terminología.

THE TWELVE-FACTOR APP

1

CÓDIGO BASE

Debemos tener un código base, es decir, tenemos que tener un sistema de control de versiones.

2

DEPENDENCIA

Nuestras dependencias tienen que ir declaradas mediante un fichero, ya que no se asume que nuestro entorno va a tener estas dependencias instaladas por defecto.

3

CONFIGURACIONES

Todo lo que cambie entre despliegues, es decir, si cogemos nuestra aplicación y la desplegamos en preproducción, en QA o en producción, posiblemente existan cambios, pues todos estos cambios tienen que ir dentro de un fichero de configuración.

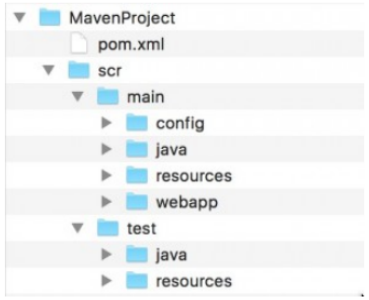
MAVEN

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zy

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy



Cada una de las carpetas de la jerarquía contiene lo siguiente:

- **src**: código fuente, ficheros de configuración, recursos y demás. Es decir, todo salvo el directorio de salida (target) y el pom.xml
- **main**: desarrollo de la aplicación, independientemente de los test
- **test**: desarrollo de los test de la aplicación
- **config**: ficheros de configuración en el proyecto
- **java**: clases java que contienen el código fuente de la aplicación
- **resources**: recursos que se incluirán en el empaquetado y serán usados por la aplicación, como pueden ser ficheros de texto o scripts
- **webapp**: contiene todos los ficheros correspondientes a la propia aplicación web que no sean código java

GIT

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente

comando	función
git init	inicializa un repositorio
git add	agrega cambios a staging area
git status	muestra el estado de los cambios
git reset HEAD	elimina un cambio del staging area
git commit	envía un cambio del staging al repositorio
git rm	borra cambio del repositorio
git checkout	trae la última versión del cambio, crea y cambia de rama
git branch	muestra las ramas
git merge	fusiona las ramas
git remote add origin	relacionar repo local a un repo remoto
git clone	clonar repositorio remoto
git pull	trae la última versión del repo remoto
git push	envía los cambio al repo remoto

CLASES Y OBJETOS

- Es la unidad básica que encapsula toda la información de un Objeto

(un objeto es una instancia de una clase).

- A través de ella podemos moldear el entorno que se desea estudiar

(una casa, un auto, una cuenta corriente, etc.).

- Generalmente representa un sustantivo (persona, lugar o cosa)

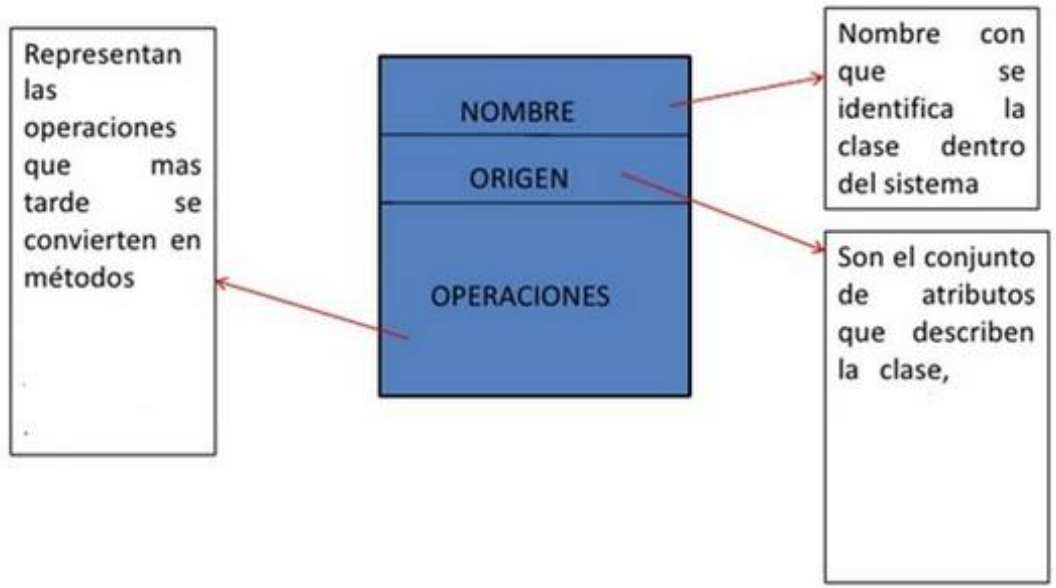


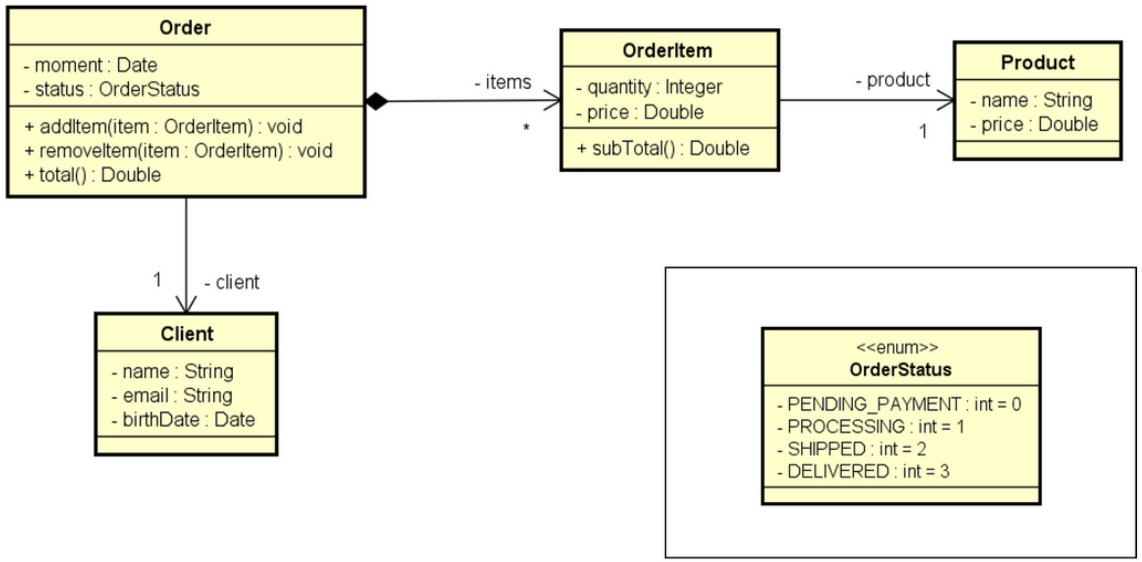
DIAGRAMA DE CLASES
Y ASOCIACIONES

Los diagramas de clases son uno de los tipos de diagramas más útiles. En UML, ya que trazan claramente la estructura de un sistema concreto al modelar sus clases, atributos, operaciones y relaciones Entre objetos. Con nuestro **software de generación de diagramas UML.** La creación de estos diagramas no es tan abrumadora como podría parecer. Esta guía te ayudará a entender, planificar y crear tu propio diagrama de clases

DEMO : MITOCODE-MOD1-POO

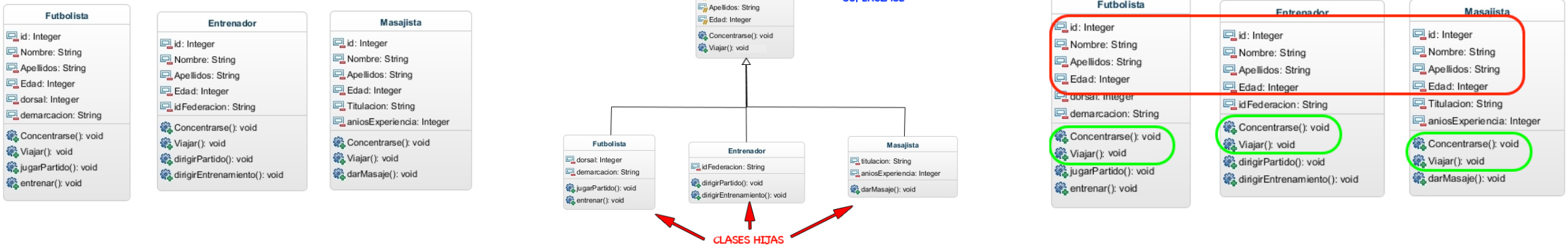
Leer datos de un pedido con N artículos (N suministrados por el usuario).
Luego, muestre un resumen del pedido como se muestra en el ejemplo.
Nota: la hora del pedido debe ser la hora del sistema

```
Enter client data:
Name: Henry Mendoza Puerta
Email: hmendo81@gmail.com
Birth date (DD/MM/YYYY): 2021-08-19
Enter order data:
Status: PROCESSING
How many items to this order? 2
Enter #1 item data:
Product name: TV
Product price: 10000.00
Quantity: 1
Enter #2 item data:
Product name: Mouse
Product price: 40.00
Quantity: 2

Order moment: 2021-08-19
Order status: PROCESSING
Client: Henry Mendoza Puerta (2021-08-19) - hmendo81@gmail.com
Order items:
TV, $10000.00, Quantity: 1, Subtotal: $10000.00
Mouse, $40.00, Quantity: 2, Subtotal: $80.00
Total price: $10080.00
```



HERENCIA

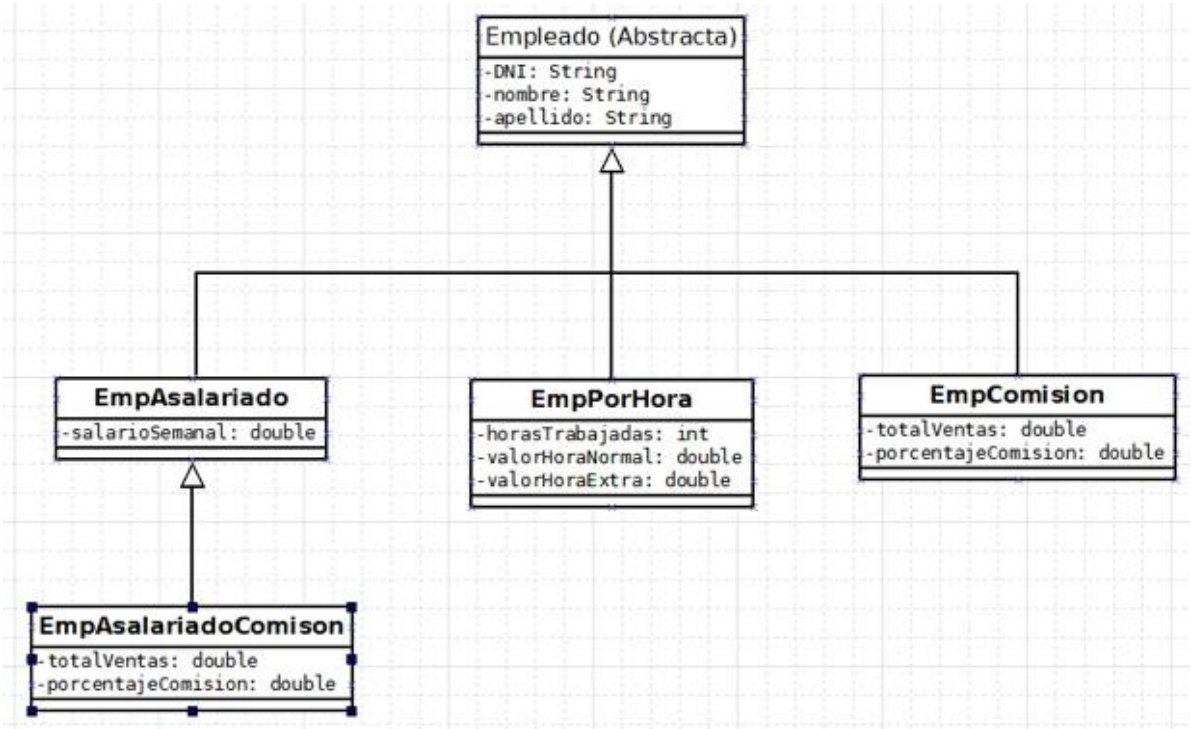
La Herencia es uno de los 4 pilares de la programación orientada a objetos (POO) junto con la Abstracción, Encapsulación y Polimorfismo. Al principio cuesta un poco entender estos conceptos característicos del paradigma de la POO porque solemos venir de otro paradigma de programación como el paradigma de la programación estructurada pero se ha de decir que la complejidad está en entender este nuevo paradigma y no en otra cosa.
En esta entrada vamos a explicar de la mejor manera posible que es la herencia y lo vamos a explicar con un ejemplo.



HERENCIA CLASE

ABSTRACTA

Una clase abstracta es una clase que no se puede instanciar (crear un objeto de esa clase) pero si se pueden definir atributos e implementar métodos en ella para que sus clases hijas los puedan utilizar



POLIFORMISMO

Polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación.

DEMO: MITOCODE-MOD1 HERENCIAPOLIMORFISMO

Elaborar un programa para leer los datos de N contribuyentes (N proporcionados por el usuario), que pueden ser personas físicas o jurídicas, y luego mostrar el monto del impuesto pagado por cada uno, así como el total del impuesto recaudado.

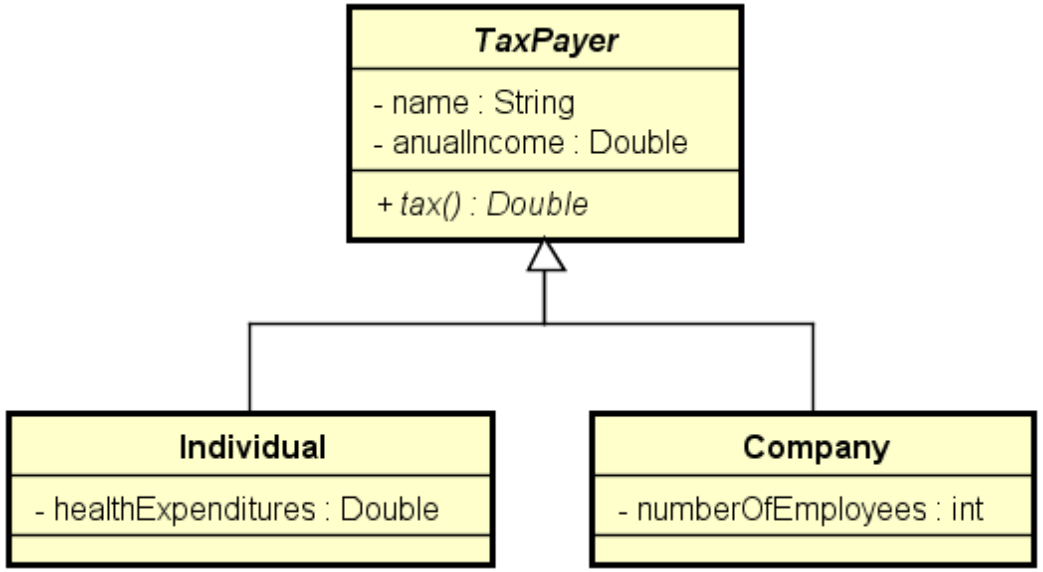
Los datos individuales son: nombre, ingresos anuales y gastos de salud. Los datos de la entidad legal son el nombre, los ingresos anuales y el número de empleados. Las reglas para el cálculo de impuestos son las siguientes:

Persona natural: las personas con ingresos inferiores a 20000,00 pagan el 15% de impuesto. Las personas con ingresos desde 20000.00 en adelante pagan el 25% de impuesto. Si la persona tuvo gastos de atención médica, el 50% de estos gastos se deducen del impuesto. Ejemplo: una persona cuyo ingreso fue 50000.00 y tuvo 2000.00 en gastos de salud, el impuesto es: (50000 * 25%) - (2000 * 50%) = 11500.00

Persona jurídica: las personas jurídicas pagan el 16% de impuestos. Sin embargo, si la empresa tiene más de 10 empleados, paga el 14% de impuestos. Ejemplo: una empresa cuyos ingresos fueron 400000,00 y tiene 25 empleados, el impuesto es: 400000 * 14% = 56000,00

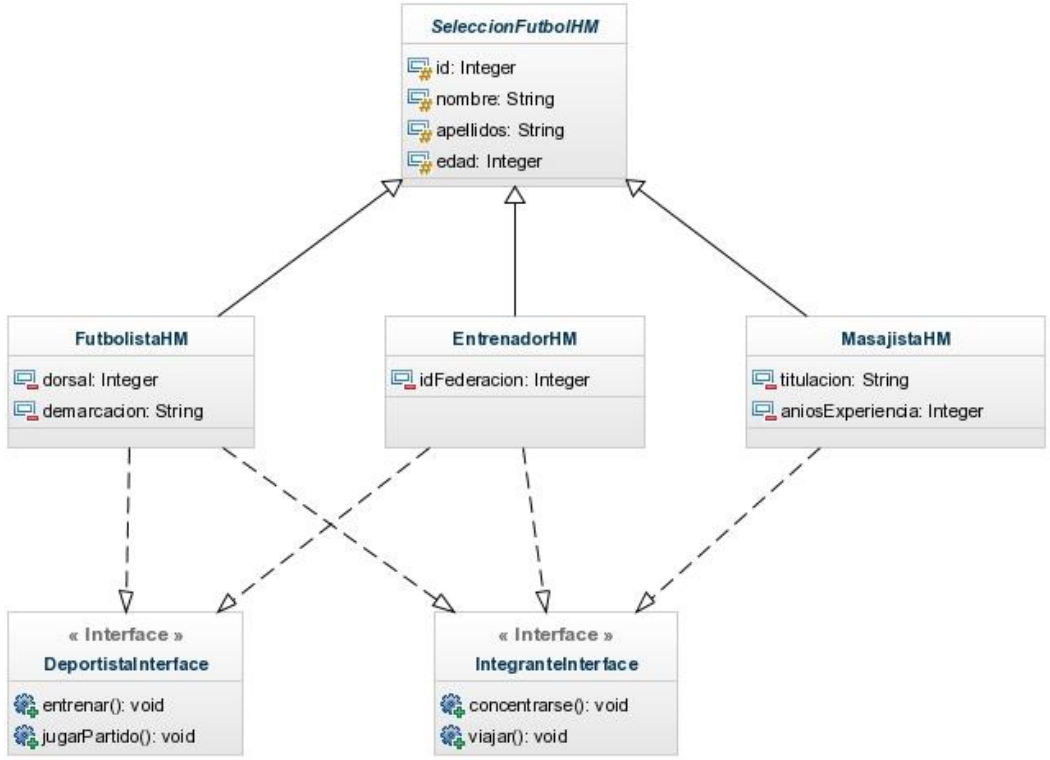
DEMO: MITOCODE-MOD1 HERENCIAPOLIMORFISMO

```
Enter the number of taxpayers: 3
Taxpayer #1 data:
Individual or company (i/c)? i
Name: Henry
Anual income: 50000.00
Health expenditures: 2000.00
Taxpayer #2 data:
Individual or company (i/c)? c
Name: SoftTech
Anual income: 400000.00
Number of employees: 25
Taxpayer #3 data:
Individual or company (i/c)? i
Name: Bob
Anual income: 120000.00
Health expenditures: 1000.00

TAXES PAID:
Henry: $ 11500.00
SoftTech: $ 56000.00
Bob: $ 29500.00

TOTAL TAXES: $ 97000.00
```



INTERFACES

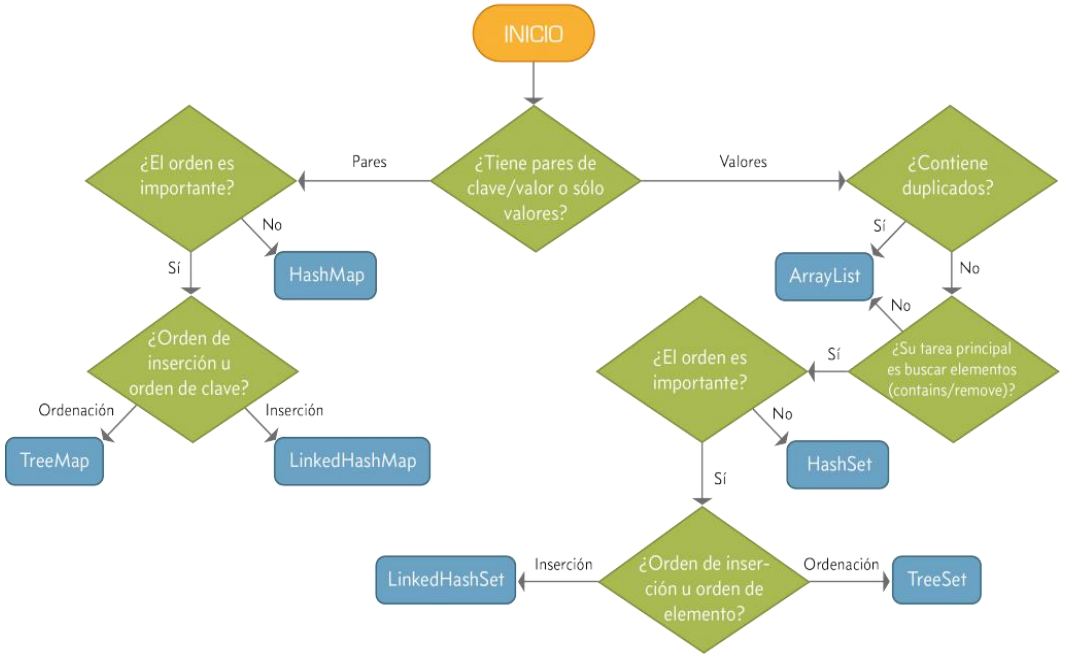
Una Interface es una clase abstracta pura en la que todos sus métodos son abstractos y
por tanto no se pueden implementar en la clase Interface.



COLECCIONES

Una colección representa un grupo de objetos. Esto objetos son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos. En Java, se emplea la interfaz genérica **Collection** para este propósito. Gracias a esta interfaz, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser: añadir, eliminar, obtener el tamaño de la colección... Partiendo de la interfaz genérica **Collection** extienden otra serie de interfaces genéricas. Estas subinterfaces aportan distintas funcionalidades sobre la interfaz anterior.

Diagrama de decisión para uso de colecciones Java



DEMO: MITOCODE-MOD1 COLECCIONLIST

Haga un programa para leer un entero N y luego el datos (cédula, nombre y salario) de N empleados, no debería haber id repetir.

Luego haga el aumento del X por ciento en el salario de un empleado en particular. Para ello, el programa debe leer una identificación y el valor X. Si la identificación ingresada no existe, mostrar un mensaje y abortar la operación. Al final, muestra el listado actualizado de empleados.

Recuerde aplicar la técnica de encapsulación para no Permitir que el salario se cambie libremente. Uno El salario solo se puede aumentar en base a una operación. de aumento por porcentaje dado.

DEMO: MITOCODE-MOD1 COLECCIONLIST

```
Emplyoee #1:
Id: 333
Name: Maria Brown
Salary: 4000.00

Emplyoee #2:
Id: 536
Name: Henry Mendoza Puerta
Salary: 3000.00

Emplyoee #3:
Id: 772
Name: Bob Mendez
Salary: 5000.00

Enter the employee id that will have salary increase : 536
Enter the percentage: 10

List of employees:
333, Maria Brown, 4000.00
536, Henry Mendoza Puerta, 3300.00
772, Bob Mendez, 5000.00
```

```
Emplyoee #1:
Id: 333
Name: Henry Mendoza Puerta
Salary: 40000.00

Emplyoee #2:
Id: 536
Name: Alex Mendez
Salary: 3000.00

Enter the employee id that will have salary increase : 776
This id does not exist!

List of employees:
333, Henry Mendoza Puerta, 40000.00
536, Alex Mendez, 3000.00
```

GENERICOS

Generics permiten que las clases, interfaces y métodos puedan ser parametrizados por tipo. Sus beneficios son:

- Reuso
- Type safety
- Performance

Uso común: Colecciones

CONVENCIÓN DE NOMBRES

Dado que se trata de tipos genéricos, su nomenclatura no afecta su comportamiento y podría designarse cualquier nombre para un genérico en Java. Sin embargo, existen convenciones para los nombres de estos tipos cuyo objetivo es mejorar la legibilidad de interpretación del código. Algunas de importancia son:

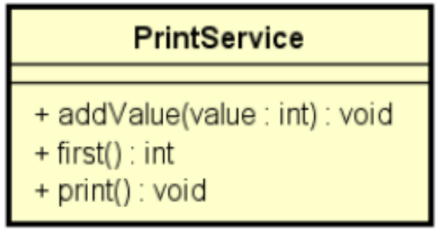
- E – Element.
- K – Key.
- V – Value.
- N – Number.
- T – Type.
- S,U,V, y así sucesivamente, para más tipos.

GENERICOS

Reuso

Se desea hacer un programa que lee una cantidad N, luego N números. Al final, imprima estos números de forma organizada de acuerdo al ejemplo. A continuación, indicar cuál fue el Primer valor informado.

```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```

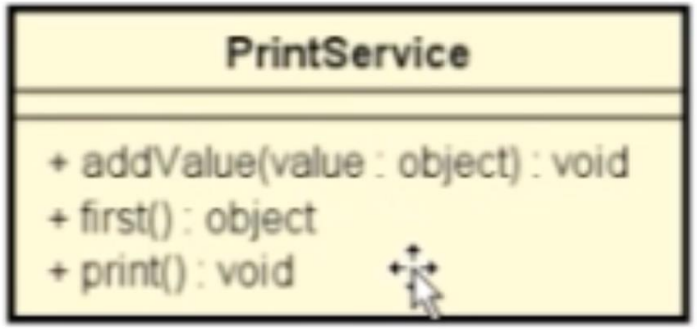


DEMO : MITOCODE-MOD1-GENERICO

Type safety & performace

Se desea hacer un programa que lee una cantidad N, luego N números. Al final, imprima estos números de forma organizada de acuerdo al ejemplo. A continuación, indicar cuál fue el Primer valor informado.

```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```

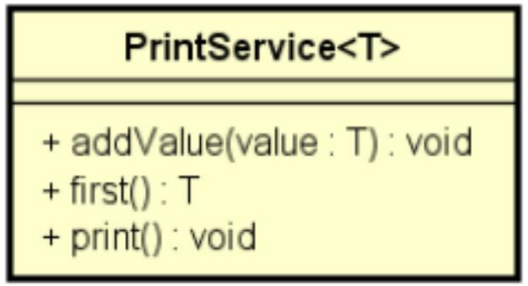


GENERICOS

Solución con Generics

Se desea hacer un programa que lee una cantidad N, luego N números. Al final, imprima estos números de forma organizada de acuerdo al ejemplo. A continuación, indicar cuál fue el Primer valor informado.

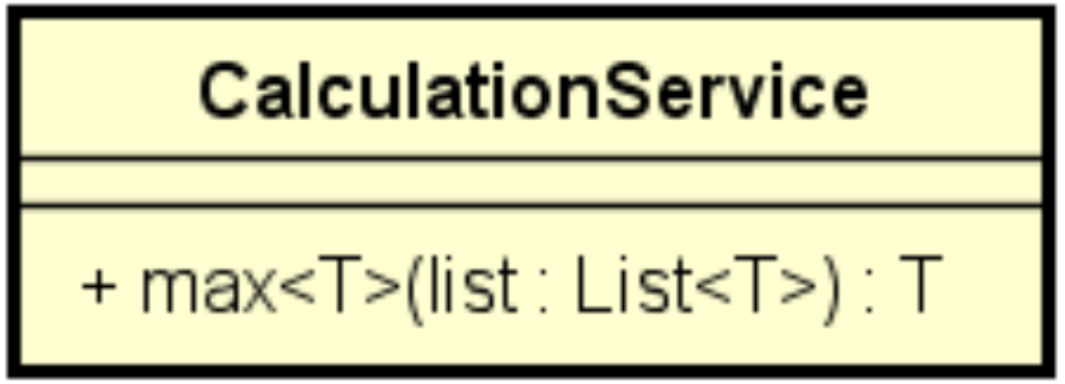
```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```



DEMO : MITOCODE-MOD1-GENERICO

Una empresa de consultoría quiere evaluar el desempeño de productos, empleados, entre otras cosas. Uno de los cálculos que necesita es encontrar el máximo valor de un conjunto de elementos. Cree un programa que lea un conjunto de productos de un archivo, como ejemplo, y luego muestre el mayor precio de ellos

```
Computer,890.50
IPhone X,910.00
Tablet,550.00
Most expensive:
IPhone, 910.00
```



MitoCode Network

DESCUBRE TU POTENCIAL

www.mitocode.com.pe