

Informe Laboratorio 3 - Paradigma Orientado a Objetos

Paradigmas de Programación

Nombre: Felipe Cubillos Arredondo
Profesor: Gonzalo Martinez

Sección: B-2
Fecha: 25/06/2024

Índice

1. Introducción	2
1.1. Descripción del Problema	2
1.2. Descripción del Paradigma	2
2. Desarrollo	3
2.1. Análisis del Problema	3
2.2. Diseño de la Solución	3
2.3. Aspectos de Implementación	4
2.4. Instrucciones de uso	4
2.5. Resultados y Autoevaluación	5
3. Conclusiones	5
3.1. Anexos	6



Figura 1: Logo De Java

1. Introducción

1.1. Descripción del Problema

Una red de metro, también conocida como sistema de transporte subterráneo o ferrocarril metropolitano, está constituida por una serie de elementos interconectados que permiten el desplazamiento eficiente de pasajeros dentro de un área urbana. [1]

Se nos ha propuesto **diseñar, modelar y emular un software de manejo de sistemas ferroviarios de metro-tren**. Tomando en consideración los componentes básicos que componen un metro: **estaciones, líneas, carros y trenes**. El objetivo principal es hacer consultas respecto al metro, con información significativa para el usuario.

1.2. Descripción del Paradigma

Programación Orientada a Objetos: Este paradigma tiene sus orígenes en el MIT en los años 60, el primer lenguaje orientado a objetos fue Simula, ideado en 1962. Los conceptos y características más importantes que hay que tener en consideración al utilizar este paradigma son:

- **Clases:** Una clase es una plantilla para un conjunto de datos.
- **Objetos:** Un objeto es una instanciación de una clase. Por ejemplo, dada una clase Perro, una instancia sería Ayudante de Santa.
- **Atributos:** Los atributos de una clase son los datos que contiene, estos pueden ser de todo tipo de dato. Por ejemplo, id, nombre, entre otros.
- **Métodos:** Los métodos son funciones dentro de una clase, dados ciertos parámetros o atributos en si misma.
- **Encapsulamiento:** Pretende esconder los detalles de implementación de algún método, este busca enfocarse en el qué y no en el cómo.
- **Herencia:** Dado un método en una clase superior, que extiende una clase inferior, este es **heredado** de la clase superior. Por ejemplo, la clase Animal tiene el método pedirComida, si las subclases Perro y Gato son extensiones de Animal, entonces por herencia pueden llamar a pedirComida().
- **Polimorfismo:** Siguiendo con la idea de extensiones y herencia, el polimorfismo tiene por finalidad cambiar la implementación de algún método en una subclase, quedando por defecto la implementación en la superclase inicial.

Lenguaje, Java 11: El lenguaje a utilizar es considerado multi-paradigma, aun que para efectos del laboratorio se utilizará sus propiedades orientadas a objetos.

1. **Java Development Kit:** Para este Laboratorio, por convención, se utilizará la versión 11 de Java, a la fecha está disponible hasta la versión 22.

Amazon Corretto es una distribución sin costo, multiplataforma y lista para producción de Open Java Development Kit (OpenJDK). Corretto cuenta con soporte a largo plazo que incluye mejoras de rendimiento y correcciones de seguridad. Corretto dispone de una certificación de conformidad con el estándar Java SE y se utiliza internamente en Amazon para muchos servicios de producción. Con Corretto, puede desarrollar y ejecutar aplicaciones Java en sistemas operativos como Amazon Linux 2, Windows y macOS. [2]

2. Gradle Build System:

Gradle, es una herramienta que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben utilizando Groovy o Kotlin DSL (Domain Specific Language). [3]

2. Desarrollo

2.1. Análisis del Problema

Similar a los experiencias pasadas de laboratorio, vemos que podemos descomponer el problema en tipos de datos abstractos. Cómo el paradigma se basa en clases, para cada TDA asignamos una clase. Así tendríamos que ir dividiendo desde los más primordiales hasta los más complejos con sus atributos característicos:

1. **TDASStation:** Unidad de la línea, tiene atributos cómo id, nombre, tipo de estación y tiempo de detención.
2. **TDASSection:** Información entre dos estaciones, cómo son la longitud y coste entre estas.
3. **TDALine:** La línea en cuestión contiene las dos clases previamente designadas, además contiene un id, un nombre y el tipo de riel.

Ahora, para el elemento del tren, vemos que se puede descomponer en:

1. **TDAPassengerCar:** Unidad del tren, tiene atributos cómo el id, la capacidad de sostener personas, el creador del tren, el tipo de tren y el modelo.
2. **TDATrain:** El tren en cuestión contiene un conjunto de carros, además de otros atributos característicos cómo el id, creador, velocidad y tiempo de estadía.

Finalmente, la unidad que engloba los otros es el TDASubway, que contiene a todas las otras clases.

1. **TDADriver:** Una unidad del metro, este interactúa con los trenes dentro del mismo sistema, tienen atributos cómo el id, el nombre y el creador del tren correspondiente.
2. **TDASubway:** Contiene trenes, líneas y conductores, es la clase de mayor capacidad jerárquica. Tiene atributos cómo id y nombre.

2.2. Diseño de la Solución

Comparando con las dos experiencias pasadas de laboratorio, este paradigma tiene un enfoque bastante distinto al momento de la resolución de problemas. Notamos que la sintaxis de Java tiene muchas similitudes con lenguajes de programación previamente vistos en otras asignaturas, cómo lo son C y Python. Comparten métodos y sentencias similares, cómo lo son el while, for, if, entre otros; por lo que se toma un enfoque imperativo-procedural.

Ahora, veremos que los TDA's pueden ser implementados cómo clases, antes de realizar cualquier codificación, es importante utilizar la herramienta del UML

A class diagram is a diagram used in designing and modeling software to describe classes and their relationships. Class diagrams enable us to model software in a high level of abstraction and without having to look at the source code.

Classes in a class diagram correspond with classes in the source code. The diagram shows the names and attributes of the classes, connections between the classes, and sometimes also the methods of the classes. [4]

Antes de confeccionar cualquier línea de código es importante establecer la relación e interacciones que tienen los métodos y clases entre sí dentro del proyecto, en la sección de anexos se encuentra el diagrama de análisis, cuyo proposito es de organizar el proyecto y hacer cuenta de la estructura en general. 2

2.3. Aspectos de Implementación

Una vez completado el análisis y diseño de la implementación, empezamos a idear la implementación, que resulta bastante elemental, cómo fué mencionao en la sección de diseño ahora tenemos herramientas iterativas y existen menos requerimientos al momento de proponer una implementación de los requisitos funcionales. Java al soportar multiples paradigmas es posible aplicar todos los conocimientos previamente aprendidos para resolver lo pedido. Tomemos por ejemplo `fetchCapacity`, que dado un tren, este devuelve la capacidad total que tiene para contener personas.

```
1 public class TDATrain{
2     List<TDAPassengerCar> pcarList;
3
4     /* Sabiendo que existe TDAPCar y respectivo getter */
5     public int fetchCapacity(){
6         int output = 0;
7         for(TDAPassengerCar curPcar : pcarList){
8             output += curPcar.getPassengerCapacity();
9         }
10        return output;
11    }
12 }
```


Vemos que la solución sigue una lógico imperativa, pero al mismo utiliza conceptos del orientado objetos con los métodos y getters.

Uno de los aspectos más importantes de la implementación es la carga por archivos, osea dejamos los atributos de los objetos a instanciar en archivos de texto, por ejemplo, cómo los otros archivos de texto tienen por separador el guión (-), para el TDA Line se tiene en su respectivo archivo el id, el nombre, el material y el rango de secciones que contiene; la misma estrategia es implementada para el TDA Train. Estos, una vez registrados se añaden a un arreglo con todas las entidades instanciadas, por lo que existe un arreglo para cada clase con todas los objetos. Es importante acotar que la lectura de archivos se realiza dentro de la clase y el método main.

Finalmente, se designa una clase para la implementación del menú, esta no posee atributos pero si un constructor para ser inicializado en la clase Main.

2.4. Instrucciones de uso

Cómo requisito previo es necesario tener descargado Java apropiadamente, teniendo las variables de entorno correctamente configuradas y los SDK junto a Gradle.

Todos los archivos necesarios se encuentran en el repositorio compartido con la cuenta de [paradigmasdiinf](#) y adjuntos en este párrafo. 

Cómo fue mencionado en la introducción, se utilizará Gradle para la compilación del código, osea, una vez descomprimido el archivo `Laboratorio3.21461391.FelipeCubillosArredondo.zip` 6 Abriendo una terminal en la carpeta, en el mismo directorio se debe correr la siguiente secuencia de comandos para ejecutar el código en consola:

En Windows 5

```
$ gradlew.bat build
$ gradlew.bat run
# En caso de no funcionar, configurar variables de entorno
# Si vuelve a fallar, borrar carpeta build en directorio del proyecto
```

En UNIX

```
$ ./gradlew build
$ ./gradlew run
```

Nota importante, si se presenta el error `ERROR JAVA_HOME is set to an invalid directory` entonces no está bien configurada la variable de entorno de Java, por lo que hay que anteponer:

```

$ export JAVA_HOME="path"
# Donde path es lo que resulte del siguiente comando
$ readlink -f $(which java)
# Después comandos faltantes...
$ ./gradlew build
$ ./gradlew run

```

En la sección de anexos se adjunta una imagen de lo que debería de aparecer en pantalla si todo se ejecuta correctamente. 4

2.5. Resultados y Autoevaluación

Este laboratorio fué probado en 2 sistemas distintos. En un Acer Nitro 5 AN515-42 con sistema operativo Windows 10 y un Lenovo Thinkpad L470 con sistema operativo Debian 12. Para ambos, siguiendo las instrucciones de uso muestra adecuadamente el menú y las opciones elegidas con los archivos entregan resultados coherentes. La autoevaluación con más detalle se encuentra en el directorio de archivos cómo un archivo de texto plano, pero a groso modo revisaremos los TDA's que tienen más precedencia

1. TDALine: Todos los requisitos funcionales se ejecutan de manera adecuada.
2. TDATrain: Todos los requisitos funcionales se ejecutan de manera adecuada.
3. TDASubway: Se logran todos los requisitos funcionales sin falla menos los siguientes:
 - assignTrainToLine
 - assignDriverToTrain
 - whereIsTrain
 - trainPath

Finalmente, una vez terminado el código, es importante realizar el **Diagrama de Diseño**, este da una visión mas realista del proyecto terminado a comparación del previo **Diagrama de Análisis**, que daba expectativas de la organización entre clases y métodos. 3

Cabe destacar que todas las clases y métodos fueron documentados en formato JavaDocs, este se encuentra en el mismo repositorio, para una mejor revisión de este se recomienda revisar el archivo que reside en el directorio `/JavaDocs/org/example/package-summary.html`.

3. Conclusiones

En conclusión, este laboratorio, a comparación de los pasados, provó ser más intensivo en términos de organización y comprensión del enunciado. Los requisitos no funcionales terminaron siendo más demandantes contrastando con los requisitos funcionales, que pudieron ser realizados con más rapidez debido a la flexibilidad del lenguaje Java.

La experiencia en general considero que es invaluable, ya que da un ejemplo de cómo es trabajar en un sistema que requiera del manejo de la programación orientada objetos, que es utilizada ampliamente en el mercado actual, además de enseñar herramientas de composición de código, cómo lo son los diagramas de estandar UML, Github y la documentación por medio de JavaDocs.

Nuevamente, comparando con los paradigmas anteriores, sabemos que Java al ser un lenguaje multi paradigma es posible aplicar los conocimientos previamente adquiridos para la resolución de problemas e implementación de métodos.

3.1. Anexos

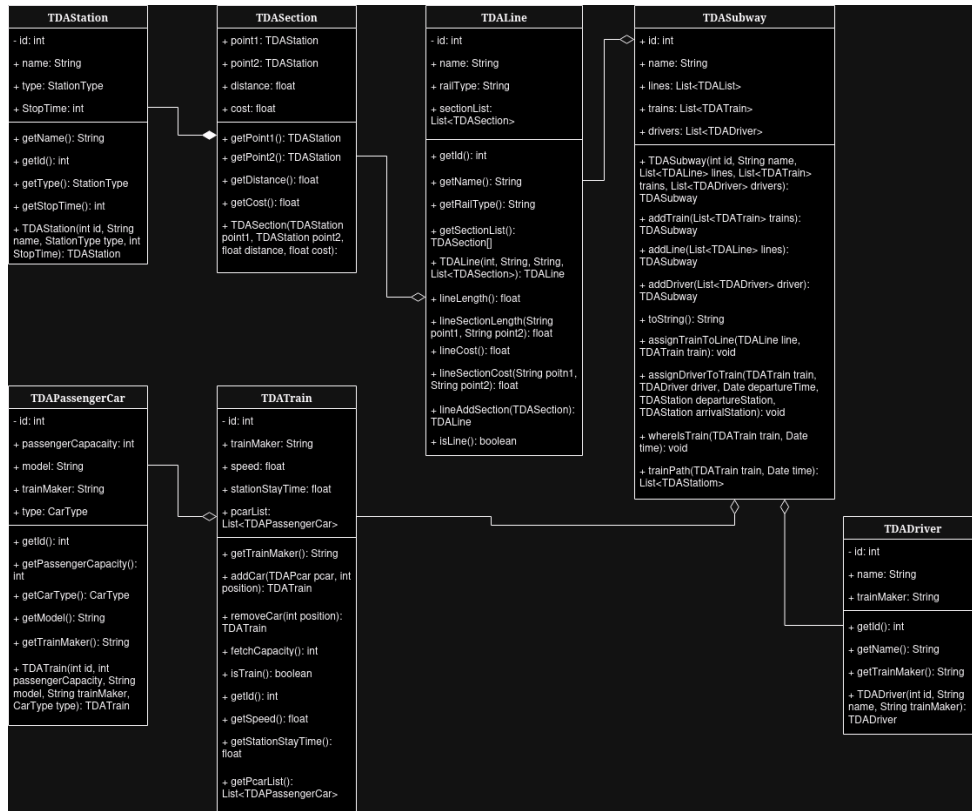


Figura 2: Diagrama de Análisis

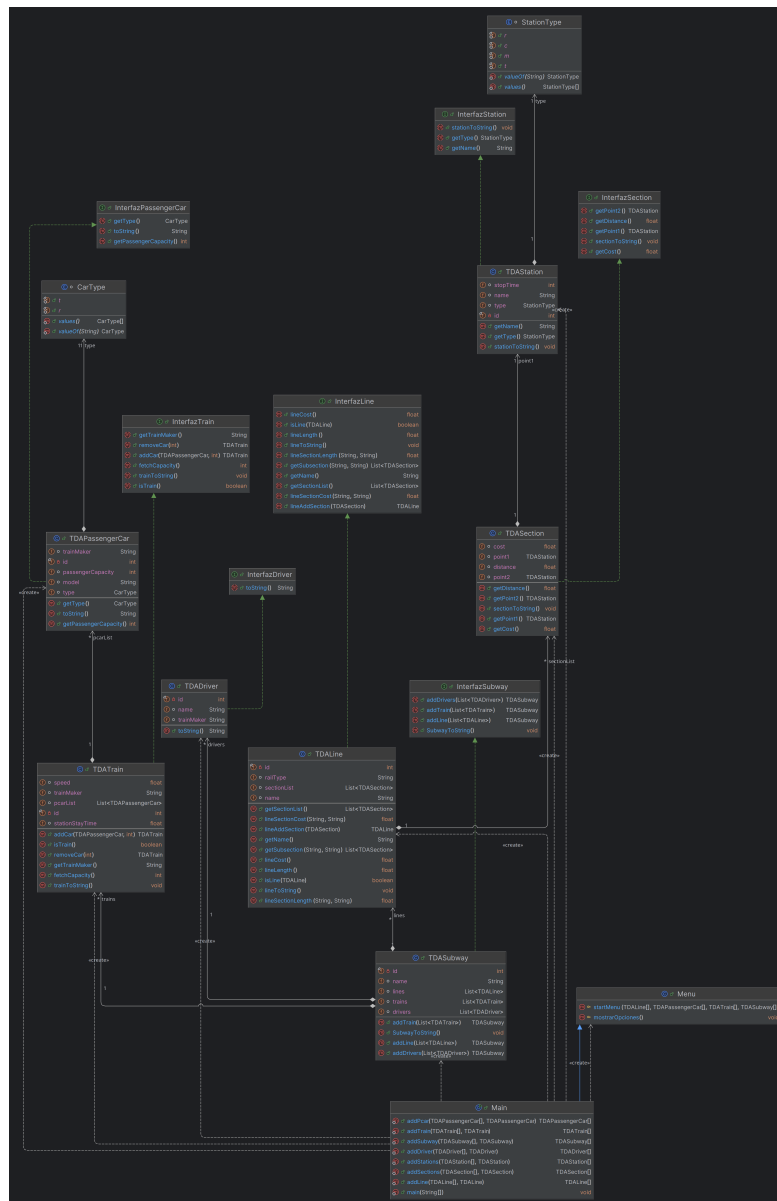
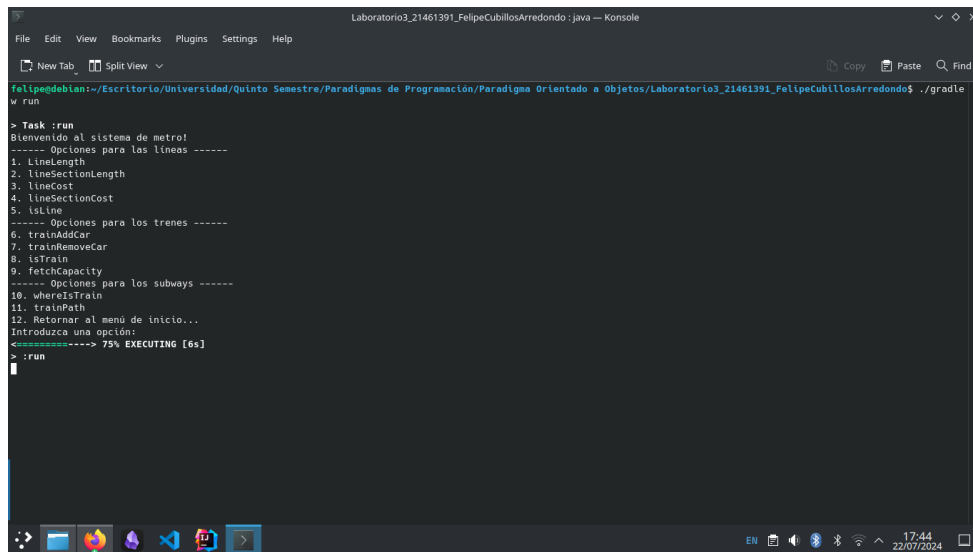
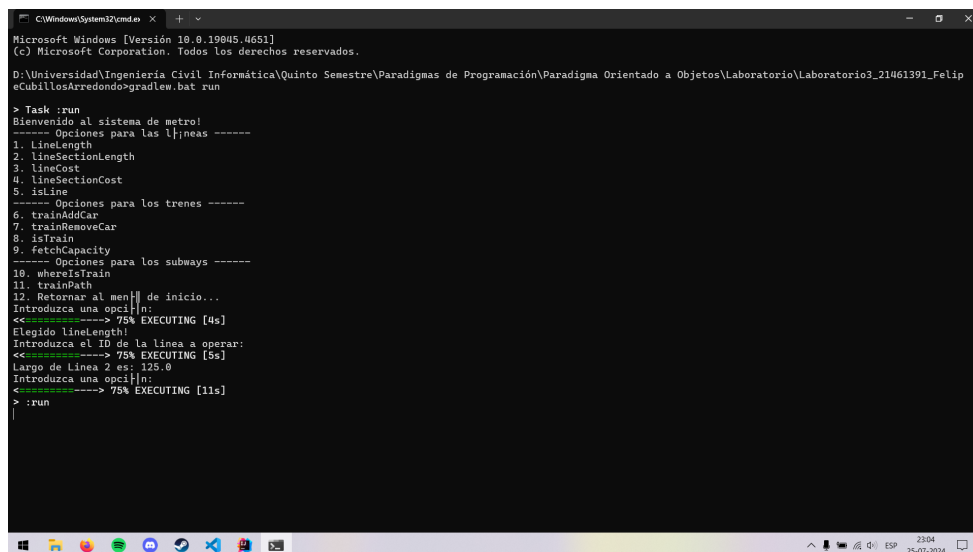


Figura 3: Diagrama de Diseño



```
Laboratorio3_21461391_FelipeCubillosArredondo : java - Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab Split View
felipe@debian:~/Escritorio/Universidad/Quinto Semestre/Paradigmas de Programación/Paradigma Orientado a Objetos/Laboratorio3_21461391_FelipeCubillosArredondo$ ./gradlew run
> Task :run
Bienvenido al sistema de metro!
----- Opciones para las líneas -----
1. LineLength
2. LineSectionLength
3. LineCost
4. LineSectionCost
5. IsLine
----- Opciones para los trenes -----
6. TrainAddCar
7. TrainRemoveCar
8. IsTrain
9. FetchCapacity
----- Opciones para los subways -----
10. WhereIsTrain
11. TrainPath
12. Retornar al menú de inicio...
Introduzca una opción:
<-----> 75% EXECUTING [6s]
> :run
```

Figura 4: Código Corriendo en Thinkpad L470 con Debian 12



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19045.4651]
(c) Microsoft Corporation. Todos los derechos reservados.
D:\Universidad\Ingeniería Civil Informática\Quinto Semestre\Paradigmas de Programación\Paradigma Orientado a Objetos\Laboratorio\Laboratorio3_21461391_FelipeCubillosArredondo>gradlew.bat run
> Task :run
Bienvenido al sistema de metro!
----- Opciones para las líneas -----
1. LineLength
2. LineSectionLength
3. LineCost
4. LineSectionCost
5. IsLine
----- Opciones para los trenes -----
6. TrainAddCar
7. TrainRemoveCar
8. IsTrain
9. FetchCapacity
----- Opciones para los subways -----
10. WhereIsTrain
11. TrainPath
12. Retornar al menú de inicio...
Introduzca una opción:
<-----> 75% EXECUTING [4s]
Elegido LineLength!
Introduzca el ID de la línea a operar:
<-----> 75% EXECUTING [5s]
Largo de línea 2 es: 125.0
Introduzca una opción:
<-----> 75% EXECUTING [11s]
> :run
```

Figura 5: Código corriendo en Acer Nitro 5 con Windows 10

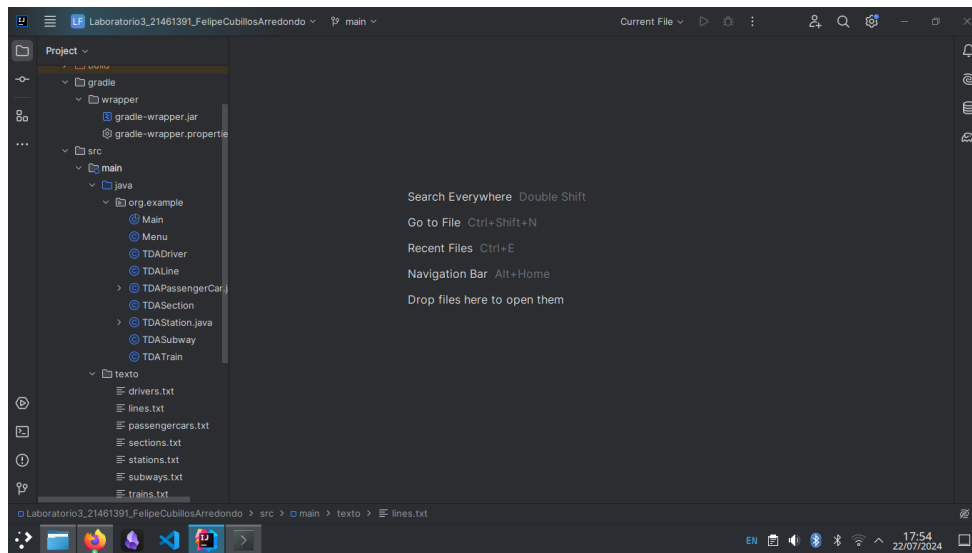


Figura 6: Directorios del proyecto y distribución de archivos

Referencias

- [1] P. de Programación, “2024 01 laboratorio general,” *Plataforma Moodle, UVirtual*, 2024.
- [2] Amazon, “¿qué es amazon corretto 11?” <https://docs.aws.amazon.com/es-es/corretto/latest/corretto-11-ug/what-is-corretto-11.html>, 2024.
- [3] O. Webinars, “Qué es gradle: La herramienta para ser más productivo desarrollando,” <https://openwebinars.net/blog/que-es-gradle/>, 2024.
- [4] U. de Helsinki, “Class diagrams,” <https://java-programming.mooc.fi/part-11/1-class-diagrams>, 2024.