ENTORNOS DE DESARROLLO

CASO PRACTICO 3UD5

ALUMNO CESUR

24/25

Alejandro Muñoz de la Sierra

PROFESOR

Diego Tinedo Rodríguez

I N T R O D U C C I O N

Nos encontramos con varios ficheros Java, parte de una versión algo reducida de un juego tan famoso como Fortnite, sin seguir un plan fijo. Lo que hicimos fue tomar esos archivos como punto de partida para armar un diagrama de clases mediante ingeniería inversa, intentando descubrir cómo se organizan las clases y se relacionan entre sí, antes de siquiera pensar en mejoras o ampliaciones del código. Fue como echar un vistazo a la anatomía del programa, para entender de qué va realmente todo esto.

Para ello, usamos StarUML, una herramienta que resulta sorprendentemente completa. Con una extensión particular, conseguimos importar directamente los archivos .java, lo que nos permitió generar los elementos básicos del modelo UML sin tener que construirlo desde cero, algo que nos ahorró bastante trabajo, al menos en un primer momento.

02
EXPLORANDO EL CÓDIGO FUENTE EN ECLIPSE

Tras revisar los archivos, terminamos identificando tres clases centrales que estructuran el programa. En Main.java se encuentra el método main, punto de arranque del programa, donde se crean los jugadores y se ponen en marcha las lógicas del juego.

```java
package main;

public class Main {

    public static void main(String[] args) {
        FortniteClass fortnite = new FortniteClass();

        Player player1 = new Player("John", 10, 5);
        Player player2 = new Player("Jane", 12, 7);
        Player player3 = new Player("Bob", 8, 4);
        Player player4 = new Player("John", 9, 6);  // Jugador duplicado
        Player player5 = new Player("Mike", 11, 8);

        fortnite.addPlayer(player1);
        fortnite.addPlayer(player2);
        fortnite.addPlayer(player3);
        fortnite.addPlayer(player4);
        fortnite.addPlayer(player5);  // Intentar agregar más allá del límite
        fortnite.printPlayerList();

        fortnite.removePlayer(player2);
        fortnite.removePlayer(player4);  // Intentar eliminar un jugador que no está en la lista
        fortnite.printPlayerList();

        int playerCount = fortnite.getPlayerCount();
        System.out.println("Cantidad de jugadores: " + playerCount);

    }

}
```

Relaciones: Usa las clases FortniteClass y Player (Dependencia).

Por otro lado, FortniteClass.java es el núcleo del juego: maneja una lista (ArrayList) para almacenar los jugadores y ofrece métodos para añadir, eliminar, contar o visualizar la lista de participantes.

```java
package main;

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

public class FortniteClass {
    private List<Player> players;

    public FortniteClass() {
        players = new ArrayList<>();
    }

    public boolean addPlayer(Player player) {
        if (players.size() >= 5) {
            return false;
        }

        if (players.contains(player)) {
            return false;
        }

        players.add(player);
        return true;
    }

    public boolean removePlayer(Player player) {
        if (players.size() <= 1) {
            return false;
        }

        if (players.contains(player)) {
            players.remove(player);
            return true;
        } else {
            return false;
        }
    }
```

```java
    public void printPlayerList() {
        if (players.isEmpty()) {
            System.out.println("No hay jugadores en la lista.");
        } else {
            System.out.println("Lista de jugadores:");
            for (Player player : players) {
                System.out.println(player.toString());
            }
        }
    }

    public int getPlayerCount() {
        return players.size();
    }

}
```

Relación: Agregación/Asociación con Player. Contiene una **lista** de objetos Player.

Y, finalmente, está Player.java, la clase que define a cada jugador (con atributos como el nombre, la velocidad y la potencia) y que además sobrescribe métodos clave como equals, hashCode, clone y toString, facilitando la comparación, clonación e impresión de la información del jugador.

```java
package main;

public class Player {
    private String name;
    private int speed;
    private int power;

    public Player(String name, int speed, int power) {
        this.name = name;
        this.speed = speed;
        this.power = power;
    }

    public String getName() {
        return name;
    }

    public int getSpeed() {
        return speed;
    }

    public int getPower() {
        return power;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Player player = (Player) obj;
        return name.equals(player.name);
    }
```

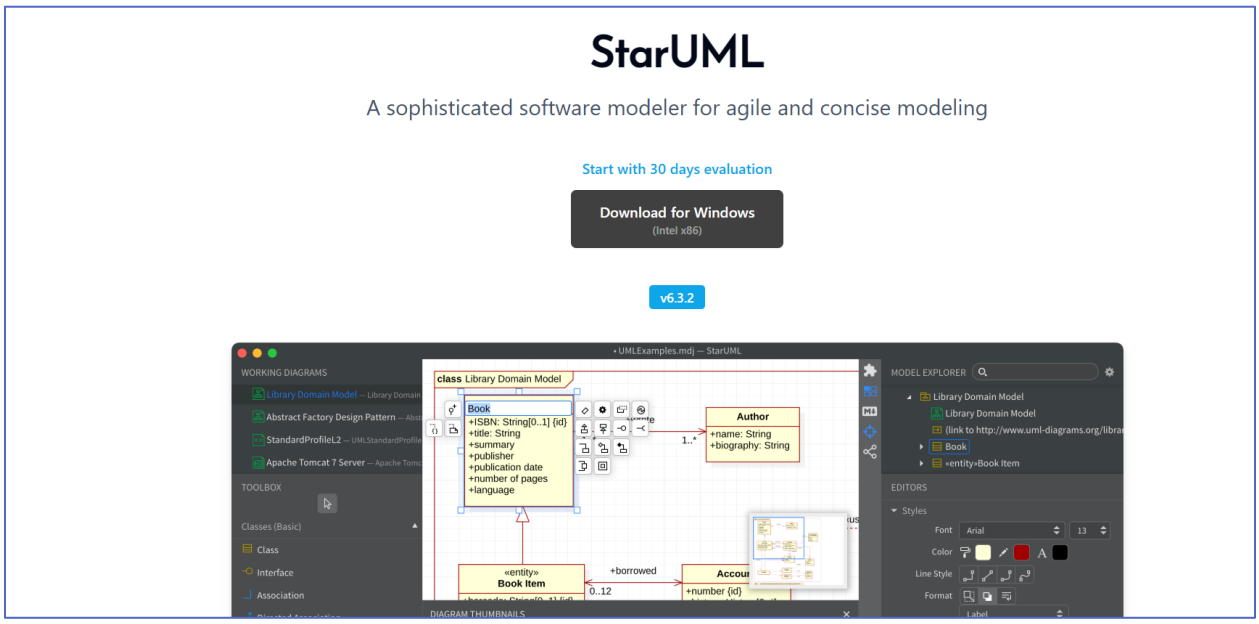
```java
    @Override
    public int hashCode() {
        return name.hashCode();
    }

    @Override
    public Player clone() {
        return new Player(this.name, this.speed, this.power);
    }

    @Override
    public String toString() {
        return "Nombre: " + name + ", Velocidad: " + speed + ", Potencia: " + power;
    }
}
```

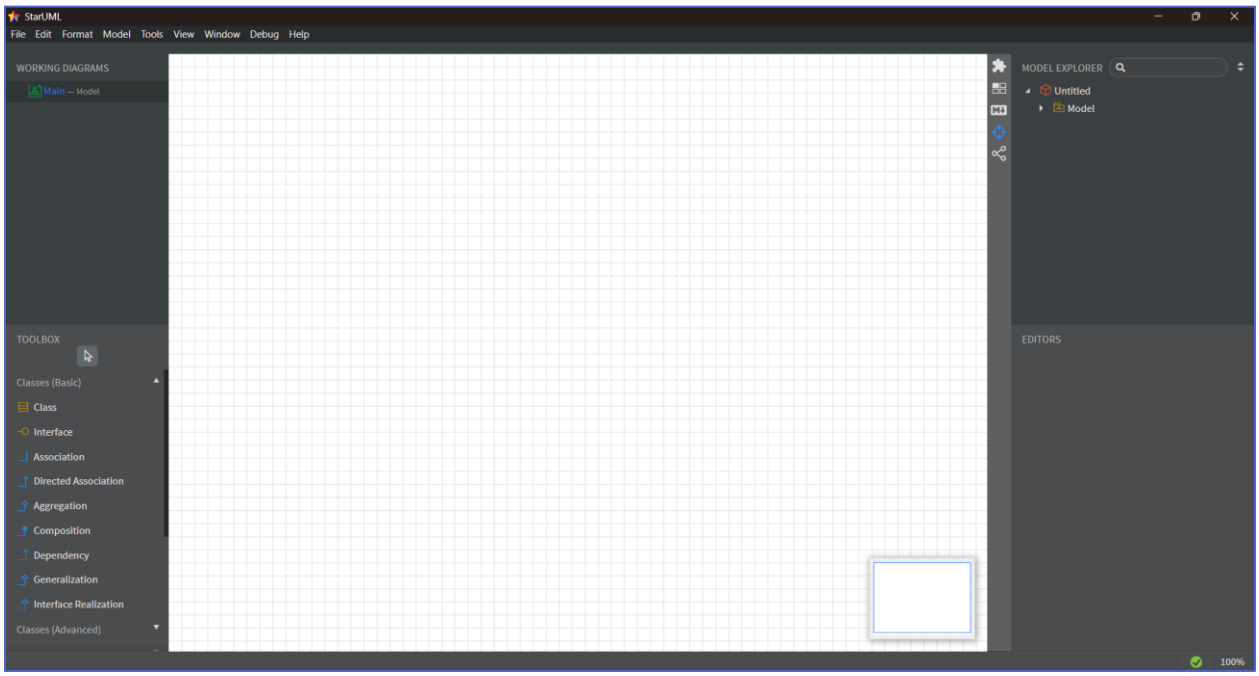
Cabe mencionar que FortniteClass, de forma casi natural, contiene varios objetos Player, mientras que Main se encarga de interactuar tanto con la clase del núcleo como con la de los jugadores para gestionar, en esencia, toda la lógica del juego.

0 3

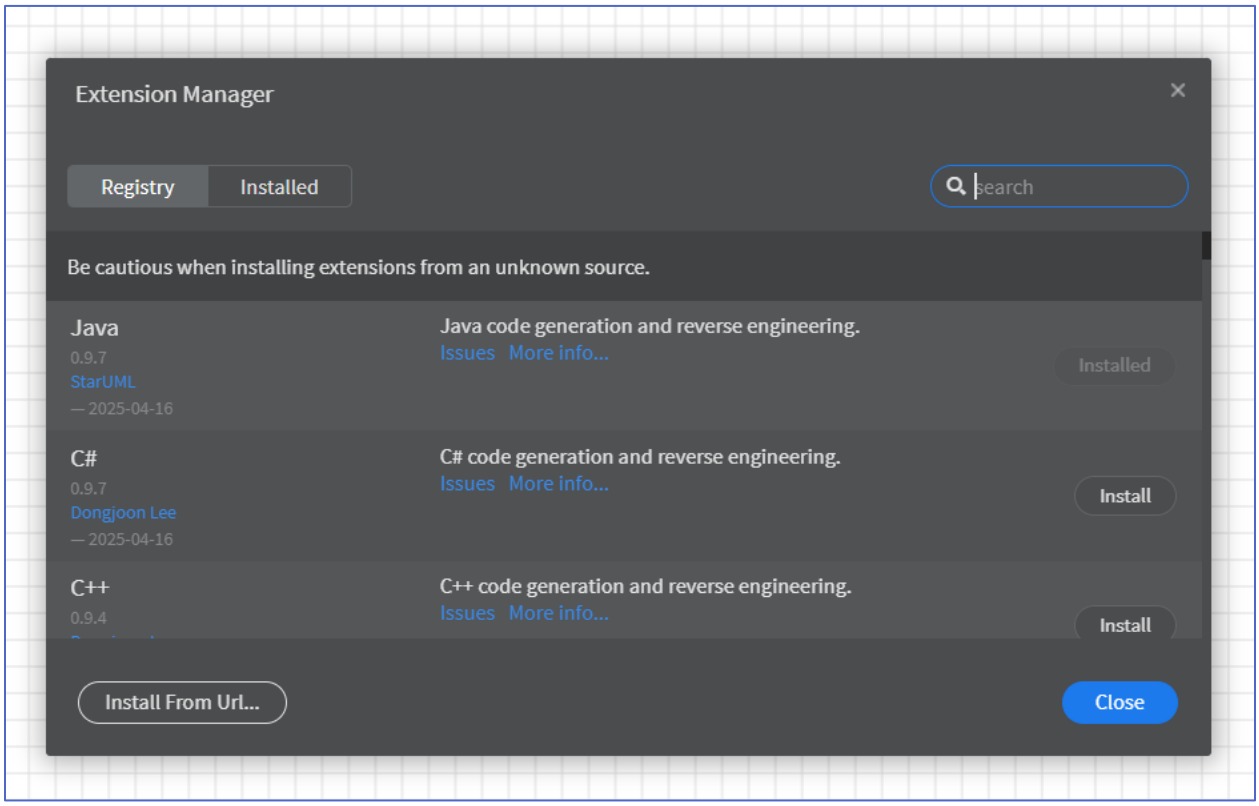
INSTALACIÓN Y PROCESO CON STARUML

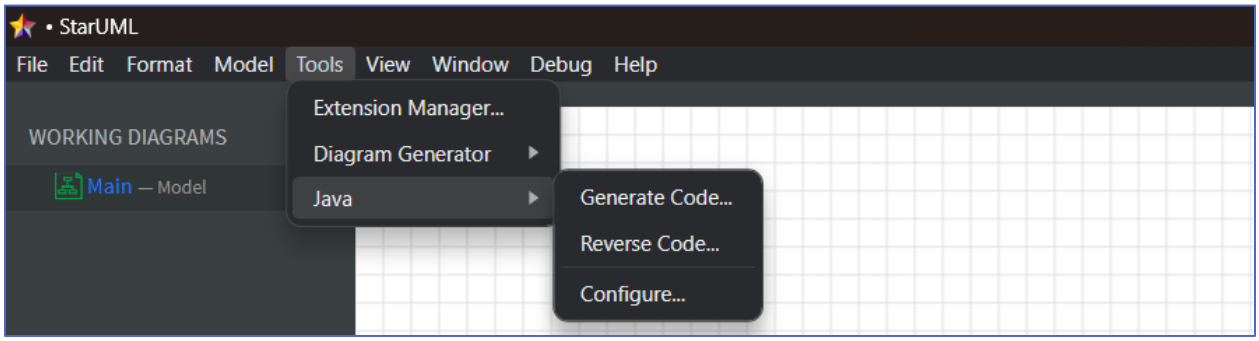
El recorrido con StarUML fue menos lineal de lo que uno espera. Primero instalamos la herramienta



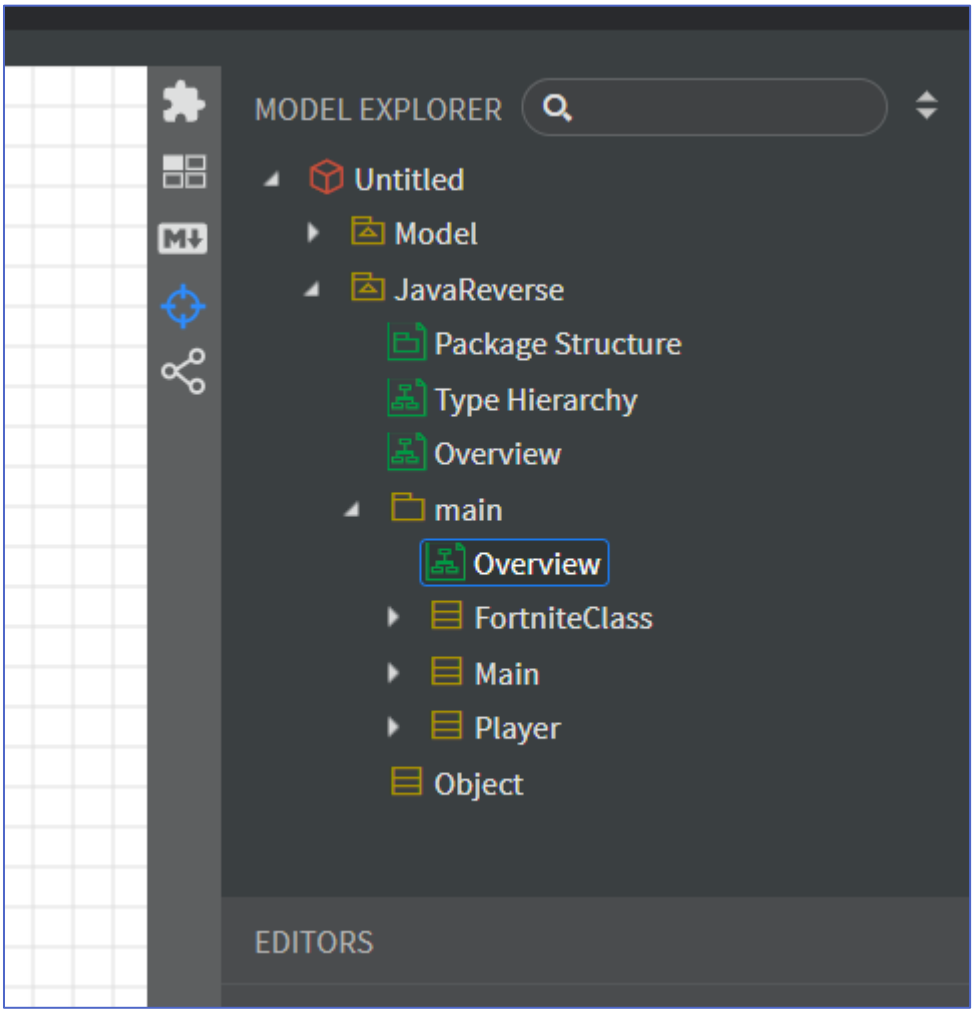


desde su sistema de complementos, le agregamos la extensión de Java.



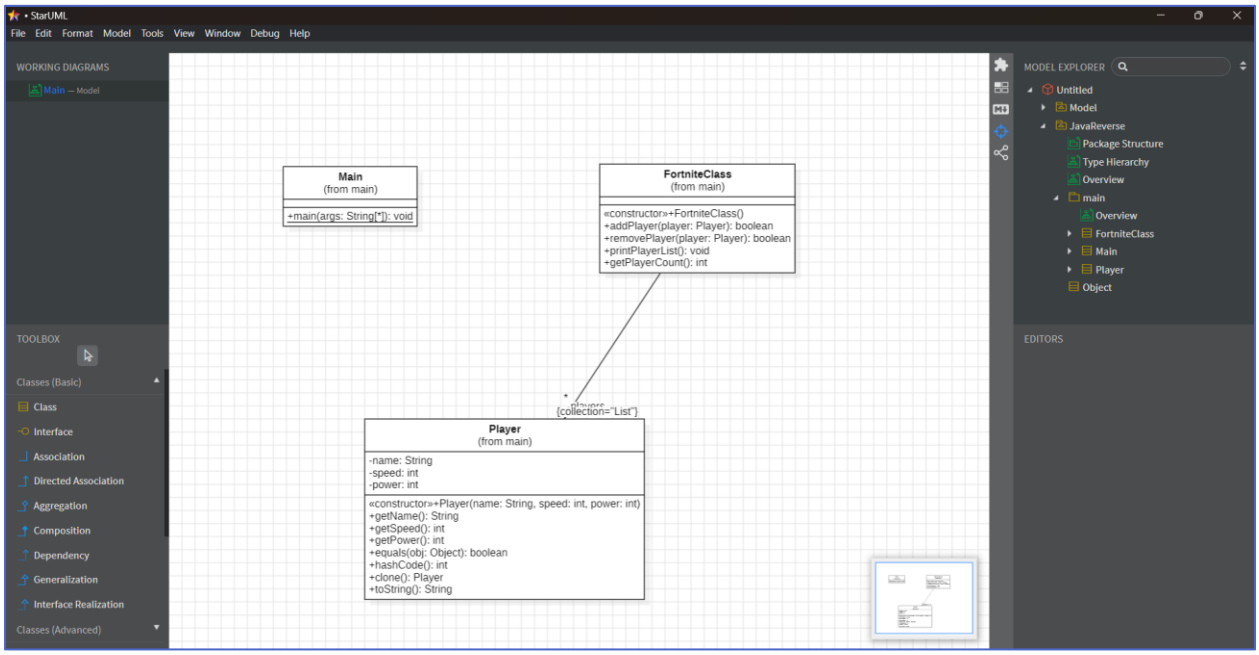


luego creamos un proyecto nuevo y vamos a tools , java, reverse code y agregamos la carpeta donde están nuestros archivos java.

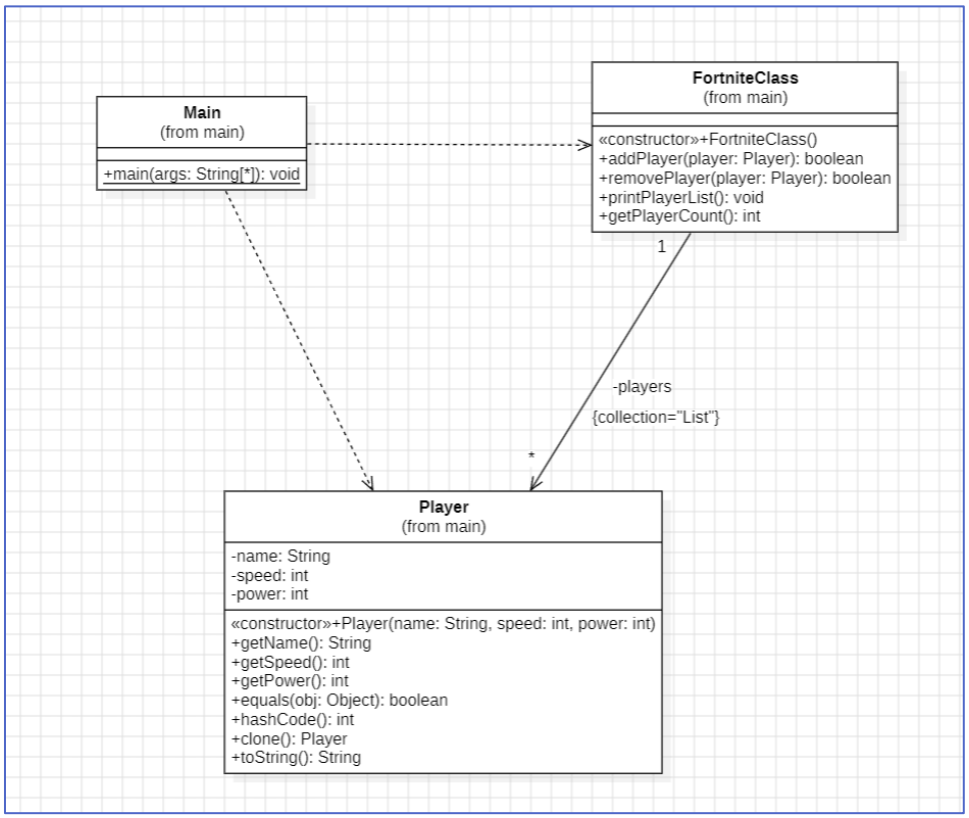


StarUML sorprendió detectando las clases automáticamente y mostrándolas en un panel del modelo.

Movemos manualmente esas clases al diagrama visual, lo cual le dio un toque más manual al proceso.



Tuvimos que ajustar algunas relaciones, ya que ciertos detalles como el uso de listas (ArrayList) no se generan de forma automática. Arreglamos, por ejemplo, la relación de agregación entre FortniteClass y Player (indicando que puede haber varios jugadores, es decir, 1..*), y enlazamos Main con Dependency con ambas clases para reflejar su papel controlador.



El resultado fue un diagrama de clases bastante limpio y claro, en el que sobresalen las tres clases principales. Se pueden ver con detalle sus atributos, métodos y las relaciones entre ellas, dejando entrever la estructura interna del programa.

0 5

C O N C L U S I O N E S

Esta práctica nos ayudó a aplicar lo que sabíamos de modelado UML y programación orientada a objetos. Al partir del código y construir el modelo, comprobamos que, generalmente, los diagramas facilitan muchísimo entender una aplicación, sobre todo cuando la documentación es casi inexistente. Además, el ejercicio nos permitió familiarizarnos con StarUML, una herramienta muy útil tanto en fases de diseño como en el análisis posterior. En definitiva, la experiencia fue bastante enriquecedora, pues nos permitió visualizar de forma clara la estructura interna del programa y valorar la importancia de contar con herramientas de modelado a lo largo del ciclo de vida del software—even si a veces el proceso se sintió un poco torpe o improvisado.

0 6

R E F E R E N C I A S

https://docs.staruml.io/working-with-uml-diagrams/class-diagram

https://docs.staruml.io/user-guide/managing-extensions

https://stackoverflow.com/questions/6167266/generate-uml-class-diagram-from-java-project

https://www.youtube.com/watch?v=ewuauF25DDI

https://github.com/staruml/staruml-java