TAREA Herramienta CASE:

Java desde UML con XML y CLIPS

Iván Falcón Monzón

ÍNDICE

Objetivo de la tarea

El objetivo principal de esta tarea es desarrollar una aplicación que **convierta modelos UML** en **código Java de manera automatizada**. Para ello, se debe integrar el procesamiento de archivos XMI, la generación de reglas en CLIPS y la ejecución de CLIPS con la librería **clipspy**, además de incorporar una **interfaz web con Flask** para facilitar la interacción.

Conclusiones

1. **Automatización del proceso**: Se logra una conversión directa desde diagramas UML a código Java sin intervención manual.
2. **Integración de tecnologías**: Se combinan **XML (XMI), Python, CLIPS y Flask** para crear la herramienta.
3. **Modularidad y escalabilidad**: La aplicación permite fácilmente agregar nuevas reglas y mejorar la generación de código.
4. **Uso práctico en desarrollo de software**: Este enfoque puede ser útil para generar código base desde diagramas UML, optimizando el proceso de desarrollo.

Primeros pasos

En **app.py** se añade un fragmento de código al principio para ejecutar de manera automática el archivo requirements.txt, que contiene los nombres de las dependencias que son necesarias para el funcionamiento del código.

app.py

app.py > process_diagram

```python
# Instalar las librerías automáticamente desde requirements.txt
import subprocess
import sys

# Verificar e instalar dependencias automáticamente
def install_requirements():
    try:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"])
        print("Dependencias instaladas correctamente.")
    except subprocess.CalledProcessError:
        print("Error al instalar las dependencias.")

install_requirements()
##########################################################################################
```

requirements.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
flask
flask-cors
clipspy
```

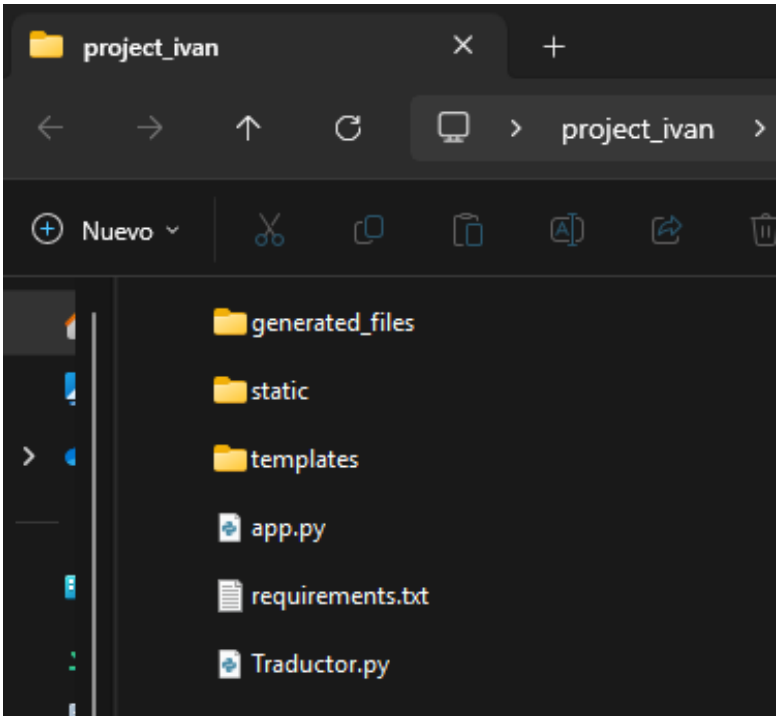
Github:

Por si el archivo subido falla, esta en mi github junto al pdf:
https://github.com/IvanFalconMonzon/TA4_CLIPS_IvanFalconMonzon.git

Archivos del proyecto

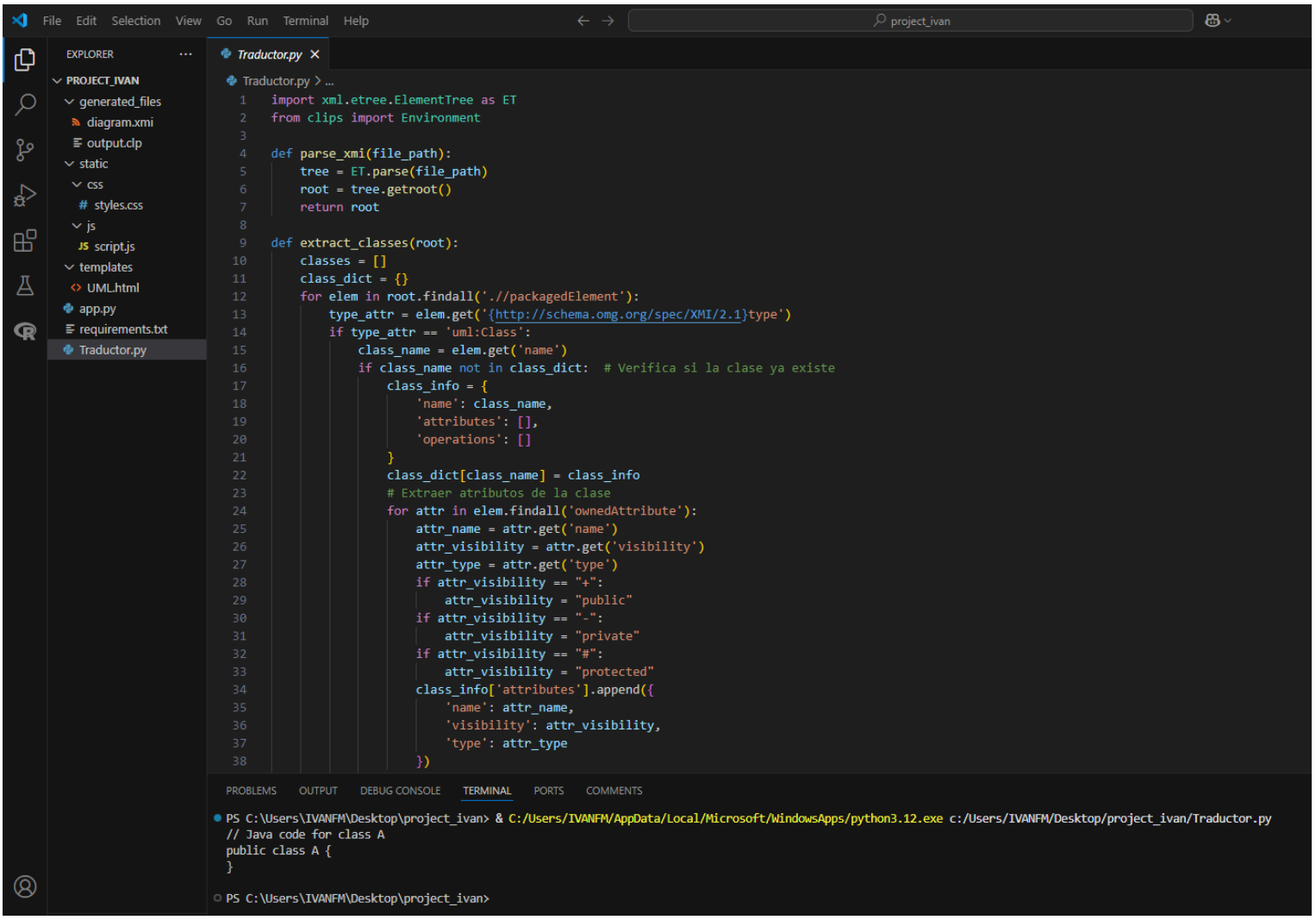
Nombre del archivo principal: project_ivan (subido con este pdf para probar funcionamiento)

```
├── app.py                  # Archivo principal de Flask
├── Traductor.py            # Programa Python para traducir diagram.xmi
├── templates/
│   └── UML.html            # HTML de la aplicación web
├── static/
│   ├── js/
│   │   └── script.js       # Lógica en JavaScript
│   └── css/
│   └── styles.css          # Estilos CSS
├── generated_files/
│   ├── diagram.xmi         # Salida generada por la aplicación web
│   └── output.clp          # Archivo generado por Traductor.py
└── requirements.txt        # Dependencias del proyecto
```



Primera pregunta

1. **(5 Puntos) Unir en un solo programa Python, Traductor.py y la parte encargada de generar el código Java al ejecutar en CLIPS el archivo output.clp. Para ello usar la librería clipspy (ver documento EjemploCLIPSPy.pdf).**



Flujo de trabajo del código de traductor.py

Parseo de XMI:

- Se extrae la información de clases UML (atributos, operaciones).
- Se detectan las relaciones UML: **asociaciones**, **generalizaciones**, **composiciones** y **agregaciones**.

Generación de hechos en CLIPS:

- Se crean las **plantillas** (deftemplate) y **hechos** (deffacts) para representar las clases y relaciones en CLIPS.

Reglas para generación de código Java:

- Se definen las reglas CLIPS (defrule) para **imprimir clases en Java** con sus atributos y métodos.

Resultado gráfico

Segunda pregunta

2. **(3 Puntos) Modificar Traductor.py para que genere las plantillas, hechos, y reglas correspondientes a los otros tipos de relaciones (asociación bidireccional, composición y agregación). Para distinguir el código generado, los atributos de tamaño dinámico asociados a las relaciones de composición pueden codificarse usando ArrayList<> o TreeSet<> y para los de agregación LinkedList<>.**

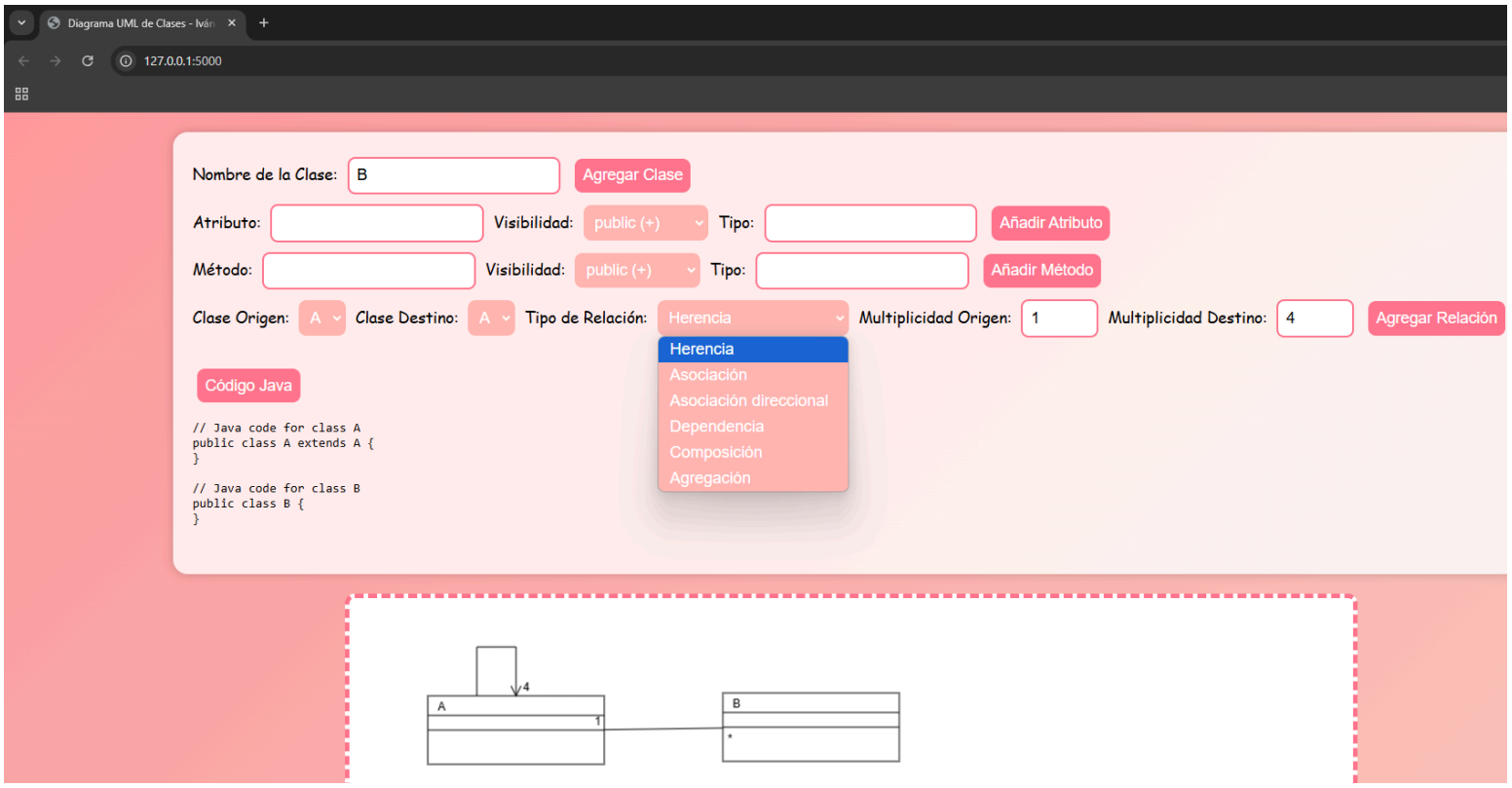
Se ha modificado el archivo Traductor.py para generar las plantillas, hechos y reglas correspondientes a los otros tipos de relaciones: asociación bidireccional, composición y agregación.

Para diferenciar el código generado según el tipo de relación:

- **Composición**: Se ha utilizado **ArrayList y TreeSet** para representar atributos de tamaño dinámico asociados a esta relación.
- **Agregación**: Se ha empleado **LinkedList** para manejar la colección de objetos agregados.
- **Asociación bidireccional**: Se ha ajustado la generación del código para reflejar correctamente las referencias mutuas entre clases.

```python
def extract_compositions(root, class_dict):
    compositions = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Composition':
            member_end = elem.get('memberEnd')
            if member_end:
                whole, part = member_end.split()
                owned_ends = elem.findall('ownedEnd')
                multiplicity_target = None
                for owned_end in owned_ends:
                    end_type = owned_end.get('type')
                    if end_type == part:
                        multiplicity_target = owned_end.get('multiplicity')
                if whole and part:
                    compositions.append({
                        'type': 'composition',
                        'whole': whole,
                        'part': part,
                        'multiplicity': multiplicity_target
                    })
                    # Añadir atributo en la clase whole
                    class_name = whole
                    if class_name in class_dict:
                        if multiplicity_target != "*":
                            class_dict[class_name]['attributes'].append({
                                'name': f'{part.lower()}List',
                                'visibility': 'private',
                                'type': f'ArrayList<{part}>'
                            })
                        else:
                            class_dict[class_name]['attributes'].append({
                                'name': f'{part.lower()}List',
                                'visibility': 'private',
                                'type': f'TreeSet<{part}>'
                            })
    return compositions
```

Resultado gráfico



Tercera pregunta

3. **(2 Puntos) Lograr que la aplicación web desencadene la ejecución de Traductor.py cuando se genere el archivo diagram.xmi, puedes hacerlo utilizando Flask y combinando la funcionalidad de la aplicación web con la lógica de Python.**

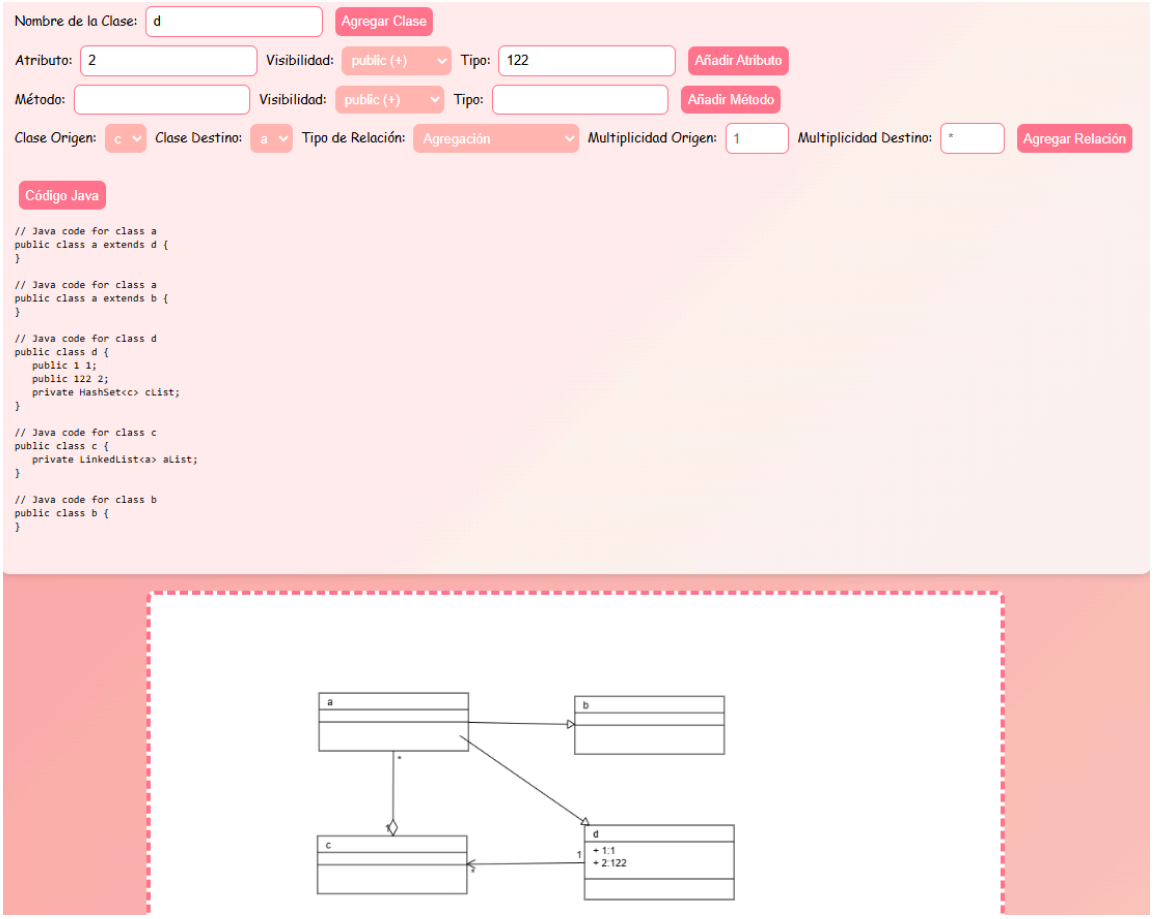
   Se ha modificado la aplicación web para que desencadene **la ejecución de Traductor.py cuando se genere el archivo diagram.xmi, utilizando Flask**.

   La implementación incluye los siguientes cambios:

   - Se ha agregado una **ruta en Flask** (/generated_files) que recibe el archivo diagram.xmi, **lo guarda en el servidor y verifica su existencia**.
   - Se ha implementado la ruta /mostrar_clp, que ejecuta Traductor.py utilizando subprocess.run(), captura su salida y la devuelve en formato JSON.
   - Se han configurado las rutas para permitir la descarga de archivos generados.
   - Se ha habilitado **CORS** para permitir el acceso desde distintas fuentes.

   Estos cambios aseguran que Traductor.py se ejecute automáticamente cuando se procesa diagram.xmi, integrando la funcionalidad de la aplicación web con la lógica de Python.

Ejemplo de esta funcionalidad:



Anexo de códigos (ADICIONAL)

app.py

```
# Instalar las librerías automáticamente desde requirements.txt
import subprocess
import sys

# Verificar e instalar dependencias automáticamente
def install_requirements():
    try:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"])
        print("Dependencias instaladas correctamente.")
    except subprocess.CalledProcessError:
        print("Error al instalar las dependencias.")

install_requirements()
##########################################################################
#########################

from flask import Flask, render_template, request, jsonify, send_from_directory
from flask_cors import CORS, cross_origin
import subprocess
import os

app = Flask(__name__)
CORS(app)  # Habilitar CORS para todas las rutas

app.config['UPLOAD_FOLDER'] = 'generated_files'
app.config['GENERATED_FOLDER'] = 'generated_files'

# Ruta para servir la página principal
@app.route('/')
def home():
    return render_template('UML.html')

# Ruta para manejar el archivo generado (diagram.xmi) y ejecutar
@app.route('/generated_files', methods=['POST'])
@cross_origin()
def process_diagram():
    try:
        # Guardar el archivo subido
        if 'xmi' in request.files:
            file = request.files['xmi']
            diagram_path = os.path.join(app.config['UPLOAD_FOLDER'], 'diagram.xmi')
            file.save(diagram_path)
            app.logger.info(f'Archivo guardado en: {diagram_path}')
```

```
        # Verificar si el archivo existe
        if not os.path.exists(diagram_path):
            return jsonify({'error': 'Archivo diagram.xmi no encontrado'}), 400

        # Confirmar éxito
        return jsonify({'message': 'Archivo procesado correctamente'}), 200

    except Exception as e:
        return jsonify({'error': f'Error inesperado: {e}'}), 500

@app.route('/mostrar_clp', methods=['GET'])
@cross_origin()
def mostrar_clp():
    try:
        # Ejecutar Traductor.py y capturar la salida
        result = subprocess.run(['python', 'Traductor.py'], capture_output=True, text=True)
        if result.returncode != 0:
            app.logger.error(f'Error al ejecutar Traductor.py: {result.stderr}')
            return jsonify({'error': f'Error al ejecutar Traductor.py: {result.stderr}'}), 500

        # Devolver la salida generada por Traductor.py
        return jsonify({'output': result.stdout}), 200

    except Exception as e:
        app.logger.error(f'Error al leer el archivo de código Java: {e}')
        return jsonify({'error': f'Error al leer el archivo de código Java: {e}'}), 500

# Ruta para servir el archivo generado
@app.route('/generated_files/<path:filename>')
@cross_origin()
def download_file(filename):
    return send_from_directory(app.config['GENERATED_FOLDER'], filename)

if __name__ == '__main__':
    app.run(debug=True)
```

Traductor.py

```
import xml.etree.ElementTree as ET
from clips import Environment

def parse_xmi(file_path):
    tree = ET.parse(file_path)
    root = tree.getroot()
    return root

def extract_classes(root):
    classes = []
    class_dict = {}
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Class':
            class_name = elem.get('name')
            if class_name not in class_dict:  # Verifica si la clase ya existe
                class_info = {
                    'name': class_name,
                    'attributes': [],
                    'operations': []
                }
                class_dict[class_name] = class_info
                # Extraer atributos de la clase
                for attr in elem.findall('ownedAttribute'):
                    attr_name = attr.get('name')
                    attr_visibility = attr.get('visibility')
                    attr_type = attr.get('type')
                    if attr_visibility == "+":
                        attr_visibility = "public"
                    if attr_visibility == "-":
                        attr_visibility = "private"
                    if attr_visibility == "#":
                        attr_visibility = "protected"
                    class_info['attributes'].append({
                        'name': attr_name,
                        'visibility': attr_visibility,
                        'type': attr_type
                    })
                # Extraer operaciones de la clase
                for op in elem.findall('ownedOperation'):
                    op_name = op.get('name')
                    op_visibility = op.get('visibility')
                    op_type = op.get('type')
                    if op_visibility == "+":
                        op_visibility = "public"
```

```
                if op_visibility == "-":
                    op_visibility = "private"
                if op_visibility == "#":
                    op_visibility = "protected"
                class_info['operations'].append({
                    'name': op_name,
                    'visibility': op_visibility,
                    'type': op_type
                })
            classes.append(class_info)
    return list(class_dict.values()), class_dict

def extract_directed_associations(root, class_dict):
    directed_associations = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:DirectedAssociation':
            member_end = elem.get('memberEnd')
            if member_end:
                source, target = member_end.split()
                owned_ends = elem.findall('ownedEnd')
                multiplicity_source = None
                multiplicity_target = None
                for owned_end in owned_ends:
                    end_type = owned_end.get('type')
                    if end_type == source and multiplicity_source is None:
                        multiplicity_source = owned_end.get('multiplicity1')
                    if end_type == target and multiplicity_target is None:
                        multiplicity_target = owned_end.get('multiplicity2')
                if source and target:
                    directed_associations.append({
                        'type': 'directedAssociation',
                        'source': source,
                        'target': target,
                        'multiplicity1': multiplicity_source,
                        'multiplicity2': multiplicity_target
                    })
                    # Añadir atributo en la clase source
                    class_name = source
                    if class_name in class_dict:
                        if multiplicity_target != "*":
                            class_dict[class_name]['attributes'].append({
                                'name': f'{target.lower()}List',
                                'visibility': 'private',
                                'type': f'{target}[]'
                            })
                        else:
```

```
                    class_dict[class_name]['attributes'].append({
                        'name': f'{target.lower()}List',
                        'visibility': 'private',
                        'type': f'HashSet<{target}>'
                    })
    return directed_associations

def extract_generalizations(root):
    generalizations = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Generalization':
            memberEnd = elem.get('memberEnd')
            parent_name, child_name = memberEnd.split()
            if parent_name and child_name:
                generalizations.append({
                    'type': 'generalization',
                    'parent': parent_name,
                    'child': child_name
                })
    return generalizations

def extract_associations(root):
    associations = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Association':
            member_end = elem.get('memberEnd')
            if member_end:
                source, target = member_end.split()
                owned_ends = elem.findall('ownedEnd')
                multiplicity_source = None
                multiplicity_target = None
                for owned_end in owned_ends:
                    end_type = owned_end.get('type')
                    if end_type == source:
                        multiplicity_source = owned_end.get('multiplicity')
                    elif end_type == target:
                        multiplicity_target = owned_end.get('multiplicity')
                if source and target:
                    associations.append({
                        'type': 'association',
                        'source': source,
                        'target': target,
                        'multiplicity1': multiplicity_source,
                        'multiplicity2': multiplicity_target
                    })
```

```
    return associations

def extract_dependencies(root):
    dependencies = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Dependency':
            memberEnd = elem.get('memberEnd')
            if memberEnd:
                client, supplier = memberEnd.split()
                if client and supplier:
                    dependencies.append({
                        'type': 'dependency',
                        'client': client,
                        'supplier': supplier
                    })
    return dependencies

def extract_compositions(root, class_dict):
    compositions = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Composition':
            member_end = elem.get('memberEnd')
            if member_end:
                whole, part = member_end.split()
                owned_ends = elem.findall('ownedEnd')
                multiplicity_target = None
                for owned_end in owned_ends:
                    end_type = owned_end.get('type')
                    if end_type == part:
                        multiplicity_target = owned_end.get('multiplicity')
                if whole and part:
                    compositions.append({
                        'type': 'composition',
                        'whole': whole,
                        'part': part,
                        'multiplicity': multiplicity_target
                    })
                    # Añadir atributo en la clase whole
                    class_name = whole
                    if class_name in class_dict:
                        if multiplicity_target != "*":
                            class_dict[class_name]['attributes'].append({
                                'name': f'{part.lower()}List',
                                'visibility': 'private',
                                'type': f'ArrayList<{part}>'
```

```
                })
            else:
                class_dict[class_name]['attributes'].append({
                    'name': f'{part.lower()}List',
                    'visibility': 'private',
                    'type': f'TreeSet<{part}>'
                })
    return compositions

def extract_aggregations(root, class_dict):
    aggregations = []
    for elem in root.findall('.//packagedElement'):
        type_attr = elem.get('{http://schema.omg.org/spec/XMI/2.1}type')
        if type_attr == 'uml:Aggregation':
            member_end = elem.get('memberEnd')
            if member_end:
                whole, part = member_end.split()
                owned_ends = elem.findall('ownedEnd')
                multiplicity_target = None
                for owned_end in owned_ends:
                    end_type = owned_end.get('type')
                    if end_type == part:
                        multiplicity_target = owned_end.get('multiplicity')
                if whole and part:
                    aggregations.append({
                        'type': 'aggregation',
                        'whole': whole,
                        'part': part,
                        'multiplicity': multiplicity_target
                    })
                    # Añadir atributo en la clase whole
                    class_name = whole
                    if class_name in class_dict:
                        class_dict[class_name]['attributes'].append({
                            'name': f'{part.lower()}List',
                            'visibility': 'private',
                            'type': f'LinkedList<{part}>'
                        })
    return aggregations

def generate_clips_facts(classes, relationships):
    clips_facts = []

        clips_facts.append('(deftemplate class\n    (slot name)\n    (multislot attributes)\n    (multislot operations))')
    clips_facts.append('(deftemplate attribute\n    (slot id)\n    (slot class-name)\n    (slot name)\n    (slot visibility)\n    (slot type))')
```

```
    clips_facts.append('(deftemplate operation\n    (slot id)\n    (slot class-name)\n    (slot name)\n    (slot visibility)\n   (slot type))')
   clips_facts.append('(deftemplate dependency\n   (slot client)\n   (slot supplier))')
   clips_facts.append('(deftemplate generalization\n   (slot parent)\n   (slot child))')
       clips_facts.append('(deftemplate directedAssociation\n      (slot source)\n      (slot target)\n      (slot multiplicity1)\n  (slot multiplicity2))')
    clips_facts.append('(deftemplate association\n    (slot source)\n    (slot target)\n    (slot multiplicity1)\n (slot multiplicity2))')
   clips_facts.append('(deftemplate composition\n   (slot whole)\n   (slot part)\n   (slot multiplicity))')
   clips_facts.append('(deftemplate aggregation\n   (slot whole)\n   (slot part)\n   (slot multiplicity))')

   clips_facts.append('(deffacts initial-facts')

   attribute_id = 1
   operation_id = 1

   for cls in classes:
       attributes = []
       operations = []

       for attr in cls['attributes']:
           attr_id = f'attr{attribute_id}'
           attributes.append(attr_id)

                clips_facts.append(f'   (attribute (id {attr_id}) (class-name {cls["name"]}) (name {attr["name"]}) (visibility {attr["visibility"]}) (type "{attr["type"]}"))')
           attribute_id += 1

       for op in cls['operations']:
           op_id = f'op{operation_id}'
           operations.append(op_id)
            clips_facts.append(f'  (operation (id {op_id}) (class-name {cls["name"]}) (name {op["name"]}) (visibility {op["visibility"]}) (type "{op["type"]}"))')
           operation_id += 1

       attributes_str = ' '.join(attributes)
       operations_str = ' '.join(operations)
                clips_facts.append(f'    (class (name {cls["name"]}) (attributes {attributes_str}) (operations {operations_str}))')

   for rel in relationships:
       if rel['type'] == 'generalization':
           clips_facts.append(f'  (generalization (parent {rel["parent"]}) (child {rel["child"]}))')
       elif rel['type'] == 'directedAssociation':
            clips_facts.append(f'  (directedAssociation (source {rel["source"]}) (target {rel["target"]}) (multiplicity1 {rel["multiplicity1"]}) (multiplicity2 {rel["multiplicity2"]}))')
       elif rel['type'] == 'association':
```

```
                clips_facts.append(f'   (association (source {rel["source"]}) (target {rel["target"]}) (multiplicity1 {rel["multiplicity1"]}) (multiplicity2 {rel["multiplicity2"]}))')
        elif rel['type'] == 'dependency':
            clips_facts.append(f'  (dependency (client {rel["client"]}) (supplier {rel["supplier"]}))')
        elif rel['type'] == 'composition':
                clips_facts.append(f'    (composition (whole {rel["whole"]}) (part {rel["part"]}) (multiplicity {rel["multiplicity"]}))')
        elif rel['type'] == 'aggregation':
                clips_facts.append(f'    (aggregation (whole {rel["whole"]}) (part {rel["part"]}) (multiplicity {rel["multiplicity"]}))')

    clips_facts.append(')')

    return clips_facts

def write_clips_file(clips_facts, file_path):
    with open(file_path, 'w') as file:
        for fact in clips_facts:
            file.write(f'{fact}\n')

        file.write('''
(defrule generate-java-code

   ?class <- (class (name ?class-name) (attributes $?attributes) (operations $?operations))
   (generalization (parent ?class-name) (child ?x))
   =>
   (printout t "// Java code for class " ?class-name crlf)
   (printout t "public class " ?class-name " extends " ?x " {" crlf)

   ;; Imprimir atributos
   (do-for-all-facts ((?attr attribute))
      (and
         (member$ (fact-slot-value ?attr id) $?attributes)
         (eq (fact-slot-value ?attr class-name) ?class-name))
      (bind ?visibility (fact-slot-value ?attr visibility))
      (bind ?type (fact-slot-value ?attr type))
      (bind ?name (fact-slot-value ?attr name))
      (printout t  "    " ?visibility " " ?type " " ?name ";" crlf))

   ;; Imprimir métodos
   (do-for-all-facts ((?op operation))
      (and
         (member$ (fact-slot-value ?op id) $?operations)
         (eq (fact-slot-value ?op class-name) ?class-name))
      (bind ?visibility (fact-slot-value ?op visibility))
      (bind ?type (fact-slot-value ?op type))
      (bind ?name (fact-slot-value ?op name))
```

```
      (printout t "    " ?visibility " " ?type " " ?name "()" " {" crlf
              "      // method body" crlf "    }" crlf))

   (printout t "}" crlf crlf)
)

(defrule generate-java-code-no-inheritance
   ?class <- (class (name ?class-name) (attributes $?attributes) (operations $?operations))
   (not (generalization (parent ?class-name)))
    =>
   (printout t "// Java code for class " ?class-name crlf)
   (printout t "public class " ?class-name " {" crlf)

   ;; Imprimir atributos
   (do-for-all-facts ((?attr attribute))
      (and
         (member$ (fact-slot-value ?attr id) $?attributes)
         (eq (fact-slot-value ?attr class-name) ?class-name))
      (bind ?visibility (fact-slot-value ?attr visibility))
      (bind ?type (fact-slot-value ?attr type))
      (bind ?name (fact-slot-value ?attr name))
      (printout t  "    " ?visibility " " ?type " " ?name ";" crlf))

   ;; Imprimir métodos
   (do-for-all-facts ((?op operation))
      (and
         (member$ (fact-slot-value ?op id) $?operations)
         (eq (fact-slot-value ?op class-name) ?class-name))
      (bind ?visibility (fact-slot-value ?op visibility))
      (bind ?type (fact-slot-value ?op type))
      (bind ?name (fact-slot-value ?op name))
      (printout t "    " ?visibility " " ?type " " ?name "()" " {" crlf
              "      // method body" crlf "    }" crlf))

   (printout t "}" crlf crlf)
)
''')

import tempfile
import os
from clips import Environment

def ejecutar_clips(clp_path):
    env = Environment()
    env.load(clp_path)
    env.reset()
    env.run()
```

```
if __name__ == '__main__':
    # Ruta del archivo XMI de entrada
    xmi_path = 'generated_files\\diagram.xmi'
    # Ruta del archivo CLIPS generado
    clips_file = 'generated_files\\output.clp'

    try:
        root = parse_xmi(xmi_path)
        classes, class_dict = extract_classes(root)
        generalizations = extract_generalizations(root)
        directed_associations = extract_directed_associations(root, class_dict)
        associations = extract_associations(root)
        dependencies = extract_dependencies(root)
        compositions = extract_compositions(root, class_dict)
        aggregations = extract_aggregations(root, class_dict)

         relationships = generalizations + directed_associations + associations + dependencies + compositions +
aggregations

        clips_facts = generate_clips_facts(classes, relationships)
        write_clips_file(clips_facts, clips_file)

        # Ejecutar CLIPS para generar el código Java y guardarlo en un archivo
        ejecutar_clips(clips_file)

    except ET.ParseError as e:
        print(f"Error al parsear el archivo XMI: {e}")
```

UML.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Diagrama UML de Clases - Iván Falcón Monzón</title>
    <!--Estilo nuevo agregado en la carpeta css-->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>
    <div class="form-container">
        <label for="classNameInput">Nombre de la Clase:</label>
        <input type="text" id="classNameInput">
        <button onclick="addClass()">Agregar Clase</button>
        <br>
        <label for="attributeInput">Atributo:</label>
        <input type="text" id="attributeInput">
        <label for="attributeVisibility">Visibilidad:</label>
        <select id="attributeVisibility">
            <option value="+">public (+)</option>
            <option value="-">private (-)</option>
            <option value="#">protected (#)</option>
        </select>
        <label for="attributeType">Tipo:</label>
        <input type="text" id="attributeType">
        <button onclick="addAttribute()">Añadir Atributo</button>
        <br>
        <label for="methodInput">Método:</label>
        <input type="text" id="methodInput">
        <label for="methodVisibility">Visibilidad:</label>
        <select id="methodVisibility">
            <option value="+">public (+)</option>
            <option value="-">private (-)</option>
            <option value="#">protected (#)</option>
        </select>
        <label for="methodType">Tipo:</label>
        <input type="text" id="methodType">
        <button onclick="addMethod()">Añadir Método</button>
        <br>
        <label for="fromClassSelect">Clase Origen:</label>
        <select id="fromClassSelect"></select>
        <label for="toClassSelect">Clase Destino:</label>
        <select id="toClassSelect"></select>
        <label for="relationType">Tipo de Relación:</label>
        <select id="relationType">
            <option value="herencia">Herencia</option>
            <option value="asociación">Asociación</option>
                                <option value="asociaciónDireccional">Asociación direccional</option>
```

```
        <option value="dependencia">Dependencia</option>
                                <option value="composición">Composición</option>
                                <option value="agregación">Agregación</option>
    </select>
    <label for="multiplicityFrom">Multiplicidad Origen:</label>
    <input type="text" id="multiplicityFrom" placeholder="1" size="3">
    <label for="multiplicityTo">Multiplicidad Destino:</label>
    <input type="text" id="multiplicityTo" placeholder="*" size="3">
    <button onclick="addRelation()">Agregar Relación</button>
    <!--Nuevo botón para mostrar el código de diagram.xmi en formato java-->
    <br><br><button onclick="mostrarCLP()">Código Java</button>
    <pre id="clpOutput"></pre>

  </div>
  <canvas id="umlCanvas" width="1024" height="768"></canvas>
  <!--Llamar al archivo script.js-->
  <script src="{{ url_for('static', filename='js/script.js') }}"></script>
</body>
</html>
```

script.js

```
const canvas = document.getElementById('umlCanvas');
const ctx = canvas.getContext('2d');
let selectedClass = null;
let offsetX, offsetY;
let selectedRelation = null;
let draggingRelation = false;
let startDragX, startDragY;

const classes = [];
const relations = [];

class UMLClass {
  constructor(name, x, y) {
    this.name = name;
    this.x = x;
    this.y = y;
    this.width = 180;
    this.height = 70; // Incluye espacio para las líneas de separación
    this.attributes = [];
    this.methods = [];
  }

  draw() {
    ctx.strokeRect(this.x, this.y, this.width, this.height);
    ctx.fillText(this.name, this.x + 10, this.y + 15);
    ctx.beginPath();
```

```
        ctx.moveTo(this.x, this.y + 20); // Línea bajo el nombre de la clase
        ctx.lineTo(this.x + this.width, this.y + 20);
        ctx.stroke();

        let yPosition = this.y + 35;
        this.attributes.forEach(attr => {
            ctx.fillText(attr, this.x + 10, yPosition);
            yPosition += 15;
        });

        ctx.beginPath();
        ctx.moveTo(this.x, yPosition); // Línea bajo los atributos
        ctx.lineTo(this.x + this.width, yPosition);
        ctx.stroke();

        yPosition += 15;
        this.methods.forEach(meth => {
            ctx.fillText(meth, this.x + 10, yPosition);
            yPosition += 15;
        });

        this.height = Math.max(70, yPosition - this.y + 10); // Actualizar la altura de la clase
    }

    addAttribute(attr) {
        this.attributes.push(attr);
    }

    addMethod(method) {
        this.methods.push(method);
    }
}

class Relation {
    constructor(fromClass, toClass, type, fromMultiplicity, toMultiplicity) {
        this.fromClass = fromClass;
        this.toClass = toClass;
        this.type = type;
        this.fromMultiplicity = fromMultiplicity;
        this.toMultiplicity = toMultiplicity;
        this.offset = 0; // Offset inicial
    }
draw() {
    const { fromX, fromY, toX, toY } = calculateLinePoints(this.fromClass, this.toClass, this.offset);

    ctx.beginPath();
    ctx.moveTo(fromX, fromY);
```

```
    if (this.type === 'dependencia') {
        ctx.setLineDash([4, 4]); // Línea discontinua para dependencia
    }

    ctx.lineTo(toX, toY);
    ctx.stroke();
    ctx.setLineDash([]);

    // Dibuja las flechas o adornos según el tipo de relación
    if (this.type === 'herencia') {
        drawInheritanceArrow(toX, toY, fromX, fromY);
    } else if (this.type === 'composición') {
        drawCompositionDiamond(fromX, fromY, toX, toY);
    } else if (this.type === 'agregación') {
        drawAgregationDiamond(fromX, fromY, toX, toY);
    } else if (this.type === 'dependencia' || this.type === 'asociaciónDireccional') {
        drawFlecha(fromX, fromY, toX, toY);
    }

    // Mostrar multiplicidades (excepto para herencia y dependencia)
    if (this.type !== 'herencia' && this.type !== 'dependencia') {
        ctx.font = '12px Arial';
        ctx.fillText(this.fromMultiplicity, fromX - 10, fromY - 5);
        ctx.fillText(this.toMultiplicity, toX + 5, toY + 15);
    }
}
   draw() {
         if (this.fromClass === this.toClass) {
           drawReflexiveArrow(this.fromClass, this.toMultiplicity);
        } else {
           const { fromX, fromY, toX, toY } = calculateLinePoints(this.fromClass, this.toClass, this.offset);

           ctx.beginPath();
           ctx.moveTo(fromX, fromY);
           if (this.type === 'dependencia') {
               ctx.setLineDash([4, 4]); // Define el patrón de la línea discontinua
           }
           ctx.lineTo(toX, toY);
           ctx.stroke();
           ctx.setLineDash([]);

           if (this.type === 'herencia') {
               drawInheritanceArrow(toX, toY, fromX, fromY);
           }
           if (this.type === 'composición') {
               drawCompositionDiamond(fromX, fromY, toX, toY);
```

```
            }
            if (this.type === 'agregación') {
                drawAgregationDiamond(fromX, fromY, toX, toY);
            }
            if (this.type === 'dependencia' || this.type === 'asociaciónDireccional') {
                drawFlecha(fromX, fromY, toX, toY);
            }
            if ((this.type !== 'herencia') && (this.type !== 'dependencia')) {
                ctx.font = '12px Arial';
                ctx.fillText(this.fromMultiplicity, fromX - 10, fromY - 5);
                ctx.fillText(this.toMultiplicity, toX + 5, toY + 15);
            }
        }
    }

    setOffset(offset) {
        this.offset = offset;
    }
}

function drawReflexiveArrow(cls, multiplicity) {
    const startX = cls.x + cls.width / 2;
    const startY = cls.y;
    const loopWidth = 40;
    const loopHeight = 50;

    ctx.beginPath();
    ctx.moveTo(startX, startY);
    ctx.lineTo(startX, startY - loopHeight);
    ctx.lineTo(startX - loopWidth, startY - loopHeight);
    ctx.lineTo(startX - loopWidth, startY);
    ctx.moveTo(startX,startY);

            const arrowWidth = 5;
    const arrowHeight = 10;


    ctx.lineTo(startX-arrowWidth,startY-arrowHeight);
    ctx.moveTo(startX, startY);
    ctx.lineTo(startX+arrowWidth,startY-arrowHeight);

    ctx.stroke();

    // Dibujar la multiplicidad cerca de la flecha
    ctx.font = '12px Arial';
    ctx.fillText(multiplicity, startX+7, startY -5);
}
```

```
function drawInheritanceArrow(toX, toY, fromX, fromY) {
    const headLength = 10;
    const angle = Math.atan2(toY - fromY, toX - fromX);

    ctx.beginPath();
    ctx.moveTo(toX, toY);
     ctx.lineTo(toX - headLength * Math.cos(angle - Math.PI / 6), toY - headLength * Math.sin(angle - Math.PI
/ 6));
     ctx.lineTo(toX - headLength * Math.cos(angle + Math.PI / 6), toY - headLength * Math.sin(angle + Math.PI
/ 6));
    ctx.closePath();
    ctx.fillStyle = 'white';
    ctx.fill();
    ctx.stroke();
    ctx.fillStyle = 'black';
}

function drawFlecha(fromX, fromY, toX, toY) {
    const arrowWidth = 10;
    const arrowHeight = 20;
    const angle = Math.atan2(toY - fromY, toX - fromX);

    ctx.save();

    ctx.translate(toX, toY);
    ctx.rotate(angle);

    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.lineTo(-arrowWidth, -arrowHeight / 4);
    ctx.moveTo(0, 0);
    ctx.lineTo(-arrowWidth, arrowHeight / 4);
    ctx.closePath();
    ctx.stroke();
    ctx.restore();
}

function drawCompositionDiamond(fromX, fromY, toX, toY) {
    const diamondWidth = 10;
    const diamondHeight = 20;
    const angle = Math.atan2(toY - fromY, toX - fromX);

    ctx.save();
    ctx.translate(fromX, fromY);
    ctx.rotate(angle - Math.PI / 2);
```

```
    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.lineTo(-diamondWidth / 2, diamondHeight / 2);
    ctx.lineTo(0, diamondHeight);
    ctx.lineTo(diamondWidth / 2, diamondHeight / 2);
    ctx.closePath();
    ctx.fillStyle = 'black';
    ctx.fill();
    ctx.restore();
}

function drawAgregationDiamond(fromX, fromY, toX, toY) {
    const diamondWidth = 10;
    const diamondHeight = 20;
    const angle = Math.atan2(toY - fromY, toX - fromX);

    ctx.save();
    ctx.translate(fromX, fromY);
    ctx.rotate(angle - Math.PI / 2);

    ctx.beginPath();
    ctx.moveTo(0, 0);
    ctx.lineTo(-diamondWidth / 2, diamondHeight / 2);
    ctx.lineTo(0, diamondHeight);
    ctx.lineTo(diamondWidth / 2, diamondHeight / 2);
    ctx.closePath();

    ctx.fillStyle = 'white';
    ctx.fill();

    ctx.strokeStyle = 'black';
    ctx.stroke();

    ctx.restore();
}

function addClass() {
    const className = document.getElementById('classNameInput').value;
    const existingClass = classes.find(c => c.name === className);
    if (existingClass) {
        alert('La clase ya existe');
        return;
    }
    const newClass = new UMLClass(className, 50, 50);
    classes.push(newClass);
    updateClassSelects();
    drawDiagram();
```

```
}

function addAttribute() {
    const className = document.getElementById('classNameInput').value;
    const attribute = document.getElementById('attributeInput').value;
    const visibility = document.getElementById('attributeVisibility').value;
    const type = document.getElementById('attributeType').value;
    const attr = `${visibility} ${attribute}:${type}`;

    const cls = classes.find(c => c.name === className);
    if (cls) {
        cls.addAttribute(attr);
        drawDiagram();
    }
}

function addMethod() {
    const className = document.getElementById('classNameInput').value;
    const method = document.getElementById('methodInput').value;
    const visibility = document.getElementById('methodVisibility').value;
    const type = document.getElementById('methodType').value;
    const meth = `${visibility} ${method}():${type}`;

    const cls = classes.find(c => c.name === className);
    if (cls) {
        cls.addMethod(meth);
        drawDiagram();
    }
}

function addRelation() {
    const fromClass = document.getElementById('fromClassSelect').value;
    const toClass = document.getElementById('toClassSelect').value;
    const type = document.getElementById('relationType').value;
    const fromMultiplicity = document.getElementById('multiplicityFrom').value || '1';
    const toMultiplicity = document.getElementById('multiplicityTo').value || '*';

    const fromCls = classes.find(c => c.name === fromClass);
    const toCls = classes.find(c => c.name === toClass);

    if (fromCls && toCls) {
        const newRelation = new Relation(fromCls, toCls, type, fromMultiplicity, toMultiplicity);
        relations.push(newRelation);
        drawDiagram();
    }
}
```

```
function updateClassSelects() {
    const fromClassSelect = document.getElementById('fromClassSelect');
    const toClassSelect = document.getElementById('toClassSelect');

    fromClassSelect.innerHTML = '';
    toClassSelect.innerHTML = '';

    classes.forEach(cls => {
        const optionFrom = document.createElement('option');
        optionFrom.value = cls.name;
        optionFrom.text = cls.name;
        fromClassSelect.add(optionFrom);

        const optionTo = document.createElement('option');
        optionTo.value = cls.name;
        optionTo.text = cls.name;
        toClassSelect.add(optionTo);
    });
}

function drawDiagram() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    relations.forEach(relation => relation.draw());
    classes.forEach(cls => cls.draw());
}

function calculateLinePoints(fromClass, toClass, offset) {
    // Centro horizontal y vertical de la clase origen
    const fromXCenter = fromClass.x + fromClass.width / 2;
    const fromYCenter = fromClass.y + fromClass.height / 2;

    // Centro horizontal y vertical de la clase destino
    const toXCenter = toClass.x + toClass.width / 2;
    const toYCenter = toClass.y + toClass.height / 2;

    // Ancho y alto de la clase destino
    const toWidth = toClass.width;
    const toHeight = toClass.height;

    // Dirección de la línea desde el centro de la clase origen hacia el centro de la clase destino
    const dx = toXCenter - fromXCenter;
    const dy = toYCenter - fromYCenter;

    // Normalización de la dirección para obtener la unidad
    const length = Math.sqrt(dx * dx + dy * dy);
    const unitDx = dx / length;
    const unitDy = dy / length;
```

```
// Punto de origen de la relación (moviéndose desde el centro hacia el borde de la caja de la clase origen)
const fromX = fromXCenter + unitDx * (fromClass.width / 2 + offset);
const fromY = fromYCenter + unitDy * (fromClass.height / 2 + offset);

// Calcular el punto de intersección con el borde de la clase destino
let intersectionX, intersectionY;

// Calcular las intersecciones con los bordes de la caja de la clase destino
const cx = fromXCenter;
const cy = fromYCenter;
const cw = fromClass.width / 2 + offset;
const ch = fromClass.height / 2 + offset;

const tx = toXCenter;
const ty = toYCenter;
const tw = toWidth / 2;
const th = toHeight / 2;

// Se calcula la intersección con los cuatro bordes posibles de la caja de la clase destino
let intersections = [];

// Intersección con el borde izquierdo de la caja destino
  let intersection = intersectionWithLineSegment(cx, cy, tx, ty, toClass.x, toClass.y, toClass.x, toClass.y +
toClass.height);
if (intersection) intersections.push(intersection);

// Intersección con el borde superior de la caja destino
  intersection = intersectionWithLineSegment(cx, cy, tx, ty, toClass.x, toClass.y, toClass.x + toClass.width,
toClass.y);
if (intersection) intersections.push(intersection);

// Intersección con el borde derecho de la caja destino
 intersection = intersectionWithLineSegment(cx, cy, tx, ty, toClass.x + toClass.width, toClass.y, toClass.x +
toClass.width, toClass.y + toClass.height);
if (intersection) intersections.push(intersection);

// Intersección con el borde inferior de la caja destino
 intersection = intersectionWithLineSegment(cx, cy, tx, ty, toClass.x, toClass.y + toClass.height, toClass.x +
toClass.width, toClass.y + toClass.height);
if (intersection) intersections.push(intersection);

// Encontrar la intersección más cercana al centro de la clase destino
let minDistance = Number.MAX_SAFE_INTEGER;
intersections.forEach(inter => {
    const dist = distance(cx, cy, inter.x, inter.y);
    if (dist < minDistance) {
```

```
            minDistance = dist;
            intersectionX = inter.x;
            intersectionY = inter.y;
        }
    });

    // Si no se encontró ninguna intersección (esto debería ser imposible en condiciones normales)
    // se toma el centro de la clase destino como punto de llegada
    if (isNaN(intersectionX) || isNaN(intersectionY)) {
        intersectionX = toXCenter;
        intersectionY = toYCenter;
    }

    return { fromX, fromY, toX: intersectionX, toY: intersectionY };
}

function intersectionWithLineSegment(x1, y1, x2, y2, x3, y3, x4, y4) {
    const ua = ((x4 - x3) * (y1 - y3) - (y4 - y3) * (x1 - x3)) / ((y4 - y3) * (x2 - x1) - (x4 - x3) * (y2 - y1));
    const ub = ((x2 - x1) * (y1 - y3) - (y2 - y1) * (x1 - x3)) / ((y4 - y3) * (x2 - x1) - (x4 - x3) * (y2 - y1));

    if (ua >= 0 && ua <= 1 && ub >= 0 && ub <= 1) {
        const intersectionX = x1 + ua * (x2 - x1);
        const intersectionY = y1 + ua * (y2 - y1);
        return { x: intersectionX, y: intersectionY };
    }
    return null;
}

function distance(x1, y1, x2, y2) {
    return Math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2);
}

canvas.addEventListener('mousedown', function(e) {
    const mouseX = e.offsetX;
    const mouseY = e.offsetY;

      selectedClass = classes.find(cls => mouseX > cls.x && mouseX < cls.x + cls.width && mouseY > cls.y &&
mouseY < cls.y + cls.height);

    if (selectedClass) {
        offsetX = mouseX - selectedClass.x;
        offsetY = mouseY - selectedClass.y;
    } else {
        const closeRelation = relations.find(relation => {
                    const { fromX, fromY, toX, toY } = calculateLinePoints(relation.fromClass, relation.toClass,
relation.offset);
```

```
            const distance = Math.abs((toY - fromY) * mouseX - (toX - fromX) * mouseY + toX * fromY - toY *
fromX) / Math.sqrt(Math.pow(toY - fromY, 2) + Math.pow(toX - fromX, 2));
            return distance < 5;
        });

        if (closeRelation) {
            selectedRelation = closeRelation;
            draggingRelation = true;
            startDragX = mouseX;
            startDragY = mouseY;
        }
    }
});

canvas.addEventListener('mousemove', function(e) {
    if (selectedClass) {
        selectedClass.x = e.offsetX - offsetX;
        selectedClass.y = e.offsetY - offsetY;
        drawDiagram();
    } else if (draggingRelation && selectedRelation) {
        const offsetX = e.offsetX - startDragX;
        const offsetY = e.offsetY - startDragY;
        selectedRelation.setOffset(selectedRelation.offset + offsetX);
        startDragX = e.offsetX;
        startDragY = e.offsetY;
        drawDiagram();
    }
});

canvas.addEventListener('mouseup', function(e) {
    selectedClass = null;
    if (draggingRelation) {
        draggingRelation = false;
        selectedRelation = null;
    }
});

function escapeXML(value) {
    return value.replace(/</g, '&lt;').replace(/>/g, '&gt;');
}

function generateXMI() {
    let xmi = `<?xml version="1.0" encoding="UTF-8"?>\n`;
                xmi += `<XMI xmi.version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:uml="http://www.omg.org/spec/UML/20090901">\n`;
    xmi += `  <uml:Model xmi:type="uml:Model" name="UMLModel">\n`;
```

```
    classes.forEach(cls => {
        xmi += `    <packagedElement xmi:type="uml:Class" name="${cls.name}">\n`;
        cls.attributes.forEach(attr => {
            const [visibility, rest] = attr.split(' ');
            const [name, type] = rest.split(':');
            const escapedType = type ? escapeXML(type.trim()) : '';
                xmi += `      <ownedAttribute visibility="${visibility}" name="${name.trim()}" type="${escapedType}"
/>\n`;
        });
        cls.methods.forEach(meth => {
            const [visibility, rest] = meth.split(' ');
            const [name, returnType] = rest.split(':');
            const escapedReturnType = returnType ? escapeXML(returnType.trim()) : '';
                    xmi += `          <ownedOperation visibility="${visibility}" name="${name.replace('()', '').trim()}"
type="${escapedReturnType}" />\n`;
        });
        xmi += `    </packagedElement>\n`;
    });

    relations.forEach(rel => {
        let relationType = 'Association';
        if (rel.type === 'herencia') {
            relationType = 'Generalization';
        } else if (rel.type === 'composición') {
            relationType = 'Composition';
        } else if (rel.type === 'agregación') {
            relationType = 'Aggregation';
        } else if (rel.type === 'dependencia') {
            relationType = 'Dependency';
        } else if (rel.type === 'asociaciónDireccional') {
            relationType = 'DirectedAssociation';
        }

            xmi += `      <packagedElement xmi:type="uml:${relationType}" memberEnd="${rel.fromClass.name}
${rel.toClass.name}">\n`;
        if ((relationType !== 'Generalization') && (relationType !== 'Dependency')) {
            xmi += `      <ownedEnd type="${rel.fromClass.name}" multiplicity1="${rel.fromMultiplicity}" />\n`;
            xmi += `      <ownedEnd type="${rel.toClass.name}" multiplicity2="${rel.toMultiplicity}" />\n`;
        }
        xmi += `    </packagedElement>\n`;
    });

    xmi += `  </uml:Model>\n</XMI>`;
    return xmi;
}

function downloadXMI() {
```

```
    const xmi = generateXMI();
    const blob = new Blob([xmi], { type: 'application/xml' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = 'diagram.xmi';
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
}

function saveXMIToServer() {
    const xmi = generateXMI();
    const blob = new Blob([xmi], { type: 'application/xml' });
    const formData = new FormData();
    formData.append('xmi', blob, 'diagram.xmi');
    console.log(formData);
    fetch('/generated_files', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        if (data.error) {
            alert(`Error: ${data.error}`);
        } else {
            alert('Archivo procesado correctamente');
        }
    })
    .catch(error => {
        alert(`Error: ${error}`);
    });
}

function mostrarCLP() {
    saveXMIToServer();
    fetch('/mostrar_clp')
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(data => {
            document.getElementById('clpOutput').textContent = data.output;
        })
```

```
        .catch(error => {
            console.error('Error al obtener el CLP:', error);
        });

    }
```

styles.css

```
/* Estilos generales */
body {
    font-family: 'Comic Sans MS', cursive, sans-serif;
    background: linear-gradient(135deg, #ff9a9e, #fad0c4);
    text-align: center;
    margin: 0;
    padding: 20px;
}

/* Contenedor del formulario */
.form-container {
    background: rgba(255, 255, 255, 0.8);
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
    display: inline-block;
    text-align: left;
}

/* Inputs y selectores */
input, select, button {
    font-size: 16px;
    padding: 8px;
    border-radius: 8px;
    border: none;
    margin: 5px;
}

input {
    border: 2px solid #ff758c;
}

select {
    background-color: #ffb7b2;
    color: #fff;
}

button {
    background-color: #ff758c;
    color: white;
```

```
    cursor: pointer;
    transition: transform 0.2s, background-color 0.3s;
}

button:hover {
    background-color: #ff5277;
    transform: scale(1.1);
}

/* Canvas para UML */
canvas {
    background-color: white;
    border: 5px dashed #ff758c;
    border-radius: 10px;
    margin-top: 20px;
}
```