

TAREA CASE LLM:

Código Java desde UML a otros lenguajes



ÍNDICE

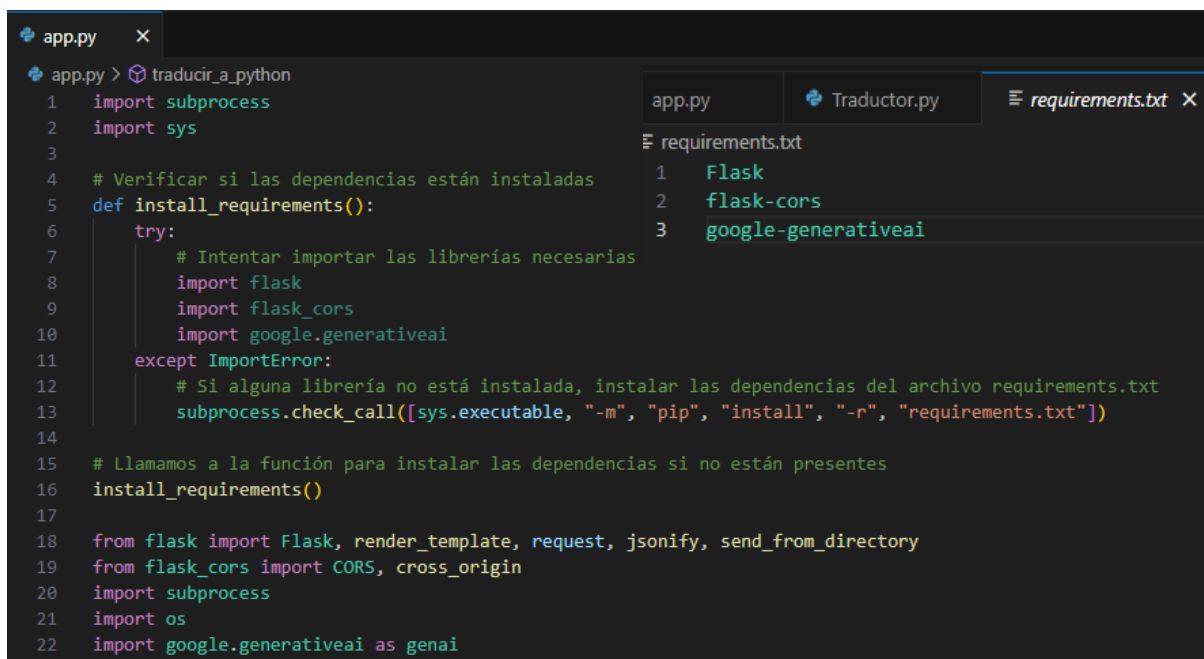
| | |
|---|-----------|
| Objetivo de la tarea..... | 3 |
| Primeros pasos..... | 3 |
| Github:..... | 3 |
| Estructura de archivos del proyecto..... | 4 |
| Descripción de la tarea:..... | 5 |
| 1. Añadir un nuevo botón a la interfaz de usuario que permita traducir el código Java generado en el área de texto a código Python..... | 5 |
| 2. Para ello, se utilizará un modelo de lenguaje grande (LLM) accesible mediante la API proporcionada por OpenRouter..... | 5 |
| Puntos de la actividad del 2 al 4..... | 6 |
| Importación y configuración..... | 6 |
| Ruta /traducir_a_python..... | 6 |
| Lógica de traducción..... | 6 |
| Llamada al modelo Geminini..... | 7 |
| Limpieza del resultado..... | 7 |
| Respuesta..... | 7 |
| Manejo de errores..... | 7 |
| 5. Se deberá documentar brevemente el funcionamiento de la API utilizada, incluyendo cómo se configura el api_key, api_base y cómo se hace la petición desde JavaScript.... | 8 |
| Documentación del funcionamiento de la API de Gemini en el proyecto..... | 8 |
| Cómo se realiza la llamada al modelo Gemini..... | 8 |
| Cómo se hace la petición desde JavaScript..... | 9 |
| Modificar la regla de clips de asociación bidireccional HashSet (Adicional)..... | 10 |
| Cambios en el estilo de la app (Adicional)..... | 12 |
| Resultado de ejemplo:..... | 14 |
| Anexo de códigos (ADICIONAL)..... | 15 |
| app.py..... | 15 |
| Traductor.py..... | 18 |
| UML.html..... | 27 |
| script.js..... | 30 |

Objetivo de la tarea

Ampliar la funcionalidad de una **herramienta CASE** de generación automática de código Java a partir de diagramas UML, incorporando un botón que permita **traducir el código Java generado a otro lenguaje orientado a objetos**, como Python, utilizando un modelo de lenguaje grande (LLM) a través de una API.

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 x
app.py > traducir_a_python
1 import subprocess
2 import sys
3
4 # Verificar si las dependencias están instaladas
5 def install_requirements():
6     try:
7         # Intentar importar las librerías necesarias
8         import flask
9         import flask_cors
10        import google.generativeai
11    except ImportError:
12        # Si alguna librería no está instalada, instalar las dependencias del archivo requirements.txt
13        subprocess.check_call([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"])
14
15 # Llamamos a la función para instalar las dependencias si no están presentes
16 install_requirements()
17
18 from flask import Flask, render_template, request, jsonify, send_from_directory
19 from flask_cors import CORS, cross_origin
20 import subprocess
21 import os
22 import google.generativeai as genai
```

```
requirements.txt
1 Flask
2 flask-cors
3 google-generativeai
```

Github:

Por si el archivo subido falla, esta en mi github junto al pdf:

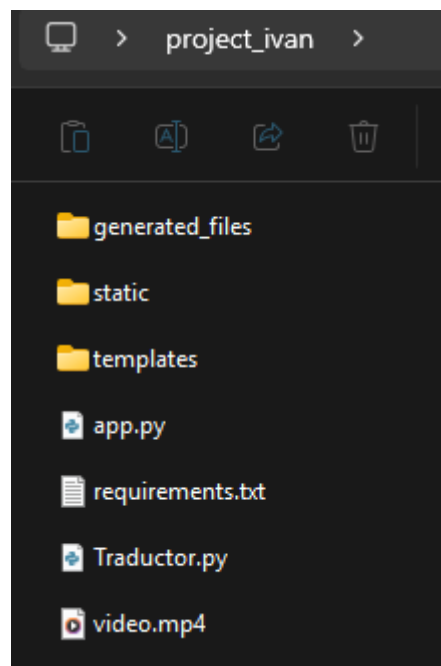
https://github.com/IvanFalconMonzon/TA_CASE-LLM_IvanFalconMonzon.git

Nota: En la carpeta del proyecto y en el github, hay un video.mp4 probando el funcionamiento de la aplicación.

Estructura de 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.xml |
| — templates/ | |
| — UML.html | # HTML de la aplicación web |
| — static/ | |
| — js/ | |
| — script.js | # Lógica en JavaScript |
| — css/ | |
| — styles.css | # Estilos CSS |
| — generated_files/ | |
| — diagram.xml | # Salida generada por la aplicación web |
| — output.clp | # Archivo generado por Traductor.py |
| — requirements.txt | |
| — video.mp4 | # Vídeo probando la aplicación |



Descripción de la tarea:

A partir de una herramienta web ya existente que permite dibujar diagramas de clases UML y generar automáticamente código Java mediante un sistema experto basado en CLIPS, se propone:

1. Añadir un nuevo botón a la interfaz de usuario que permita traducir el código Java generado en el área de texto a código Python.

```
<!-- Botón para traducir el código Java a Python -->
<br><br>
<br><br><button onclick="traducirJavaAPython()">Traducir a Python</button>

<!-- Área de texto donde se mostrará el resultado en Python -->
<pre id="pythonOutput" style="background-color: #f4f4f4; border: 1px solid #ccc; padding: 10px;"></pre>
```

Se añade en el archivo de UML.html un nuevo botón llamado "Traducir a Python", donde al pulsar en este botón saldrá en el área de texto añadida el código en python.

2. Para ello, se utilizará un modelo de lenguaje grande (LLM) accesible mediante la API proporcionada por OpenRouter.

Yo utilizo en vez de Openrouter la API gratuita de Geminis.

Integración de código en el archivo app.py:

```
79 # clave API
80 genai.configure(api_key="AIzaSyBsrUVJXR2jN4YG4lR3_qmhsiCZ11007sE")
81
82 # Traducir Java a Python con Gemini
83 @app.route('/traducir_a_python', methods=['POST'])
84 @cross_origin()
85 def traducir_a_python():
86     try:
87         java_code = request.json.get('codigo_java', '')
88
89         if not java_code.strip():
90             return jsonify({'error': 'Código Java vacío'}), 400
91
92         model = genai.GenerativeModel("gemini-2.0-flash")
93         response = model.generate_content(
94             f"Traduce este código Java a Python. Solo responde con el código Python, sin explicaciones ni comentarios:\n\n{java_code}"
95         )
96
97         # Limpiar respuesta de bloque markdown si es necesario
98         codigo_python = response.text.strip()
99         if codigo_python.startswith("```"):
100             codigo_python = "\n".join(codigo_python.split("\n")[1:-1]).strip()
101
102         return jsonify({'codigo_python': codigo_python})
103
104     except Exception as e:
105         return jsonify({'error': f'Error al traducir código: {e}'}), 500
106
107 if __name__ == '__main__':
108     app.run(debug=True)
109
```

Puntos de la actividad del 2 al 4

Este código define un endpoint en una API web que permite traducir código Java a Python usando el modelo Gemini de Google.

Importación y configuración

```
import google.generativeai as genai

# clave API
genai.configure(api_key="AIzaSyBsrUYJXR2jN4YG4lR3_qmhsiCZ1l0O7sE")
```

- Importa la librería de Gemini (google.generativeai) para usar modelos de IA generativa.
- Configura la clave de acceso a la API de Gemini.

Ruta /traducir_a_python

```
# Traducir Java a Python con Gemini
@app.route('/traducir_a_python', methods=['POST'])
@cross_origin()
```

- Define un endpoint HTTP POST al que puedes enviar código Java.
- @cross_origin() permite que otros orígenes (como aplicaciones web) puedan llamar a esta ruta (CORS).

Lógica de traducción

```
def traducir_a_python():
    try:
        java_code = request.json.get('codigo_java', '')
```

- Obtiene el código Java que se envía en el cuerpo JSON bajo la clave "codigo_java".

```
    if not java_code.strip():
        return jsonify({'error': 'Código Java vacío'}), 400
```

- Verifica que no esté vacío. Si está vacío, devuelve un error 400.

Llamada al modelo Geminini

```
model = genai.GenerativeModel("gemini-2.0-flash")
response = model.generate_content(
    f"Traduce este código Java a Python. Solo responde con el código Python, sin explicaciones ni comentarios:\n\n{java_code}"
)
```

- Crea una instancia del modelo gemini-2.0-flash.
- Le pide al modelo que traduzca el código Java a Python, pidiéndole que no dé explicaciones, solo el código traducido.

Limpieza del resultado

```
# Limpiar respuesta de bloque markdown si es necesario
codigo_python = response.text.strip()
if codigo_python.startswith("`"):
    codigo_python =
"\n".join(codigo_python.split("\n")[1:-1]).strip()
```

- A veces Gemini responde con el código dentro de un bloque Markdown ("python").
- Este bloque se limpia para dejar solo el código.

Respuesta

```
return jsonify({'codigo_python': codigo_python})
```

- Devuelve el código Python traducido en formato JSON.

Manejo de errores

```
except Exception as e:
    return jsonify({'error': f'Error al traducir código: {e}'}), 500
```

- Si algo falla (problema con la API, formato incorrecto, etc.), devuelve un error 500 con el mensaje correspondiente.

5. Se deberá documentar brevemente el funcionamiento de la API utilizada, incluyendo cómo se configura el `api_key`, `api_base` y cómo se hace la petición desde JavaScript.

Documentación del funcionamiento de la API de Gemini en el proyecto

Configuración del API Key y conexión con Gemini

Para utilizar el modelo Gemini de Google en Python, se utiliza la librería `google.generativeai`.

Pasos de configuración:

```
import google.generativeai as genai

# Configurar la clave API para acceder al modelo
genai.configure(api_key="TU_API_KEY")
```

- **api_key**: es una clave secreta que proporciona Google para autenticar al usuario. Se debe obtener desde Google AI Studio.
- Actualmente, la librería de Gemini no requiere configurar `api_base` como otros servicios (por ejemplo, OpenAI). Solo es necesario el `api_key`.

Importante: en aplicaciones reales, nunca debes dejar el `api_key` visible en el código fuente. Se recomienda usar variables de entorno para mayor seguridad.

Cómo se realiza la llamada al modelo Gemini

Dentro de la función de Flask, se genera una respuesta del modelo con:

```
model = genai.GenerativeModel("gemini-2.0-flash")
response = model.generate_content("Tu mensaje aquí")
```

- `"gemini-2.0-flash"` es el nombre del modelo.
- `.generate_content(...)` envía un prompt (instrucción) al modelo para que lo procese y genere una respuesta.

En este caso, el prompt contiene el código Java y una instrucción clara para traducirlo a Python.

Cómo se hace la petición desde JavaScript

Desde el lado del cliente (por ejemplo, una página web en HTML/JavaScript), se puede hacer una petición a este endpoint con `fetch()`:

```
fetch("http://localhost:5000/traducir_a_python", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    codigo_java: `public class Hola { public static void main(String[]
args) { System.out.println("Hola Mundo"); } }`
  })
})
.then(response => response.json())
.then(data => {
  console.log("Código traducido a Python:", data.codigo_python);
})
.catch(error => {
  console.error("Error al traducir:", error);
});
```

Detalles clave:

- Se usa `fetch()` para hacer una solicitud POST al servidor local (`localhost:5000`).
- En el body, se envía el código Java en formato JSON.
- La respuesta es un JSON con el código Python traducido, bajo la clave `"codigo_python"`.

Modificar la regla de clips de asociación bidireccional HashSet (Adicional)

Se ha mejorado la gestión de asociaciones dirigidas añadiendo automáticamente relaciones bidireccionales entre clases.

Cuando una **clase A** tiene una asociación dirigida hacia una **clase B**, ahora ambas clases generan un atributo privado con tipo `HashSet<Clase>` que refleja esa relación.

¿Qué hace?

Si `ClaseA → ClaseB` y la multiplicidad del destino no es `*`, se añade:

- En ClaseA: `private HashSet<ClaseB> clasebList;`
- En ClaseB: `private HashSet<ClaseA> claseaList;`

¿Por qué HashSet?

Se usa `HashSet` para representar colecciones sin elementos duplicados y con acceso eficiente, lo cual es ideal para relaciones múltiples entre clases.

Código cambiado en el archivo `traductor.py`:

```
# =====
# ASOCIACIONES DIRIGIDAS
# =====
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 el atributo a la clase source
    if source in class_dict:
        if multiplicity_target != "*":
            class_dict[source]['attributes'].append({
                'name': f'{target.lower()}List',
                'visibility': 'private',
                'type': f'HashSet<{target}>'
            })

    # Añadir relación bidireccional en la clase target
    if target in class_dict:
        class_dict[target]['attributes'].append({
            'name': f'{source.lower()}List',
            'visibility': 'private',
            'type': f'HashSet<{source}>'
        })

    return directed_associations

```

Cambios en el estilo de la app (Adicional)

Se cambia el archivo styles.css con el fin de que sea diferente de la actividad anterior.

En la actividad anterior tenía colores rosados y ahora se ha cambiado a tonos azules y tonos verdes, siendo más limpio de visualizar y he intentado que se vea más moderno.

Código completo del archivo modificado styles.css:

```
/* Estilos generales */
body {
    font-family: 'Comic Sans MS', cursive, sans-serif;
    background: linear-gradient(135deg, #4e54c8, #8f94fb); /* Degradado azul-violeta */
    text-align: center;
    margin: 0;
    padding: 20px;
}

/* Contenedor del formulario */
.form-container {
    background: rgba(255, 255, 255, 0.9); /* Más opaco para un look más limpio */
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.1); /* Sombra más suave */
    display: inline-block;
    text-align: left;
}

/* Inputs y selectores */
input, select, button {
    font-size: 16px;
    padding: 10px;
    border-radius: 8px;
    border: none;
    margin: 5px;
}

/* Input con borde en tonos azules */
input {
    border: 2px solid #4e9fd1;
}

/* Selectores con fondo suave y texto blanco */
select {
    background-color: #6cc2c4;
    color: #fff;
}

/* Botones con tono verde */
button {
```

```

background-color: #4e9fd1; /* Azul vibrante */
color: white;
cursor: pointer;
transition: transform 0.2s, background-color 0.3s;
}

/* Efecto hover para el botón */
button:hover {
    background-color: #3b8bb2; /* Azul más oscuro al hacer hover */
    transform: scale(1.1);
}

/* Canvas para UML con bordes en verde */
canvas {
    background-color: white;
    border: 5px dashed #6cc2c4; /* Bordes verdes */
    border-radius: 10px;
    margin-top: 20px;
}

```

Diagrama UML de Clases - Iván X +

127.0.0.1:5000

Nombre de la Clase: Agregar Clase

Atributo: Visibilidad: Tipo: Añadir Atributo

Método: Visibilidad: Tipo: Añadir Método

Clase Origen: Clase Destino: Tipo de Relación: Multiplicidad Origen: Multiplicidad Destino: Agregar Relación

Código Java

```

// Java code for class B
public class B {
    public C;
    private HashSet<A> alist;
}

// Java code for class A
public class A {
}

```

Traducir a Python

```

class B:
    def __init__(self):
        self.C = None
        self.alist = set()

class A:
    def __init__(self):
        pass

```

Resultados de ejemplos:

Traducir a Python

```
class B:
    def __init__(self):
        self.C = None
        self.aList = set()

class A:
    def __init__(self):
        pass
```

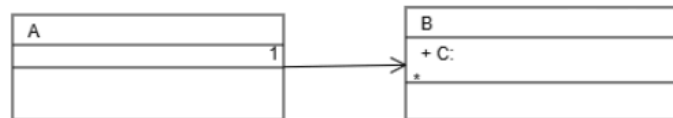


Diagrama UML de Clases - Iván

127.0.0.1:5000

Clase Origen: Clase Destino: Tipo de Relación: Multiplicidad Origen: Multiplicidad Destino:

Código Java

```
// Java code for class Alumno
public class Alumno {
    public String 2;
    private HashSet<Nombre> nombreList;
}

// Java code for class Nombre
public class Nombre {
    private HashSet<Alumno> alumnoList;
}
```

Traducir a Python

```
class Alumno:
    def __init__(self):
        self.s = ""
        self.nombreList = set()

class Nombre:
    def __init__(self):
        self.alumnoList = set()
```

```
classDiagram
    class Nombre
    class Alumno {
        + 2 string
    }
    Nombre "1" --> "1" Alumno
```

Anexo de códigos (ADICIONAL)

Nota: Se han añadido comentarios separados por sección para entender mejor el funcionamiento del código, y así poder entender mejor el funcionamiento.

app.py

```
import subprocess
import sys

# Verificar si las dependencias están instaladas
def install_requirements():
    try:
        # Intentar importar las librerías necesarias
        import flask
        import flask_cors
        import google.generativeai
    except ImportError:
        # Si alguna librería no está instalada, instalar las dependencias del archivo requirements.txt
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"])

# Llamamos a la función para instalar las dependencias si no están presentes
install_requirements()

from flask import Flask, render_template, request, jsonify, send_from_directory
from flask_cors import CORS, cross_origin
import subprocess
import os
import google.generativeai as genai

# Creación de la aplicación Flask
app = Flask(__name__)
CORS(app)

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')

# Procesar archivo XML
@app.route('/generated_files', methods=['POST'])
@cross_origin()
def process_diagram():
```

```

try:
    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}')

    if not os.path.exists(diagram_path):
        return jsonify({'error': 'Archivo diagram.xmi no encontrado'}), 400

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

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

# Mostrar código generado por CLIPS
@app.route('/mostrar_clp', methods=['GET'])
@cross_origin()
def mostrar_clp():
    try:
        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

        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

# Descargar archivos generados
@app.route('/generated_files/<path:filename>')
@cross_origin()
def download_file(filename):
    return send_from_directory(app.config['GENERATED_FOLDER'], filename)

import google.generativeai as genai

# clave API
genai.configure(api_key="AlzaSyBsrUYJXR2jN4YG4IR3-qmhsiCZ11007sE")

# Traducir Java a Python con Gemini
@app.route('/traducir_a_python', methods=['POST'])
@cross_origin()
def traducir_a_python():
    try:

```



```

java_code = request.json.get('codigo_java', "")

if not java_code.strip():
    return jsonify({'error': 'Código Java vacío'}), 400

model = genai.GenerativeModel("gemini-2.0-flash")
response = model.generate_content(
    f"Traduce este código Java a Python. Solo responde con el código Python, sin explicaciones ni comentarios:\n\n{java_code}"
)

# Limpiar respuesta de bloque markdown si es necesario
codigo_python = response.text.strip()
if codigo_python.startswith("```"):
    codigo_python = "\n".join(codigo_python.split("\n")[1:-1]).strip()

return jsonify({'codigo_python': codigo_python})

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

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

```

Traductor.py

```
# =====
# IMPORTACIONES Y PARSEADOR XML
# =====
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

# =====
# EXTRACCIÓN DE CLASES
# =====
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:
                class_info = {
                    'name': class_name,
                    'attributes': [],
                    'operations': []
                }
                class_dict[class_name] = class_info
            # Atributos
            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
                })
    })
```

```

# Operaciones
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

# =====
# ASOCIACIONES DIRIGIDAS
# =====
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 el atributo a la clase source
    if source in class_dict:

```

```

        if multiplicity_target != "*":
            class_dict[source]['attributes'].append({
                'name': f'{target.lower()}List',
                'visibility': 'private',
                'type': f'HashSet<{target}>'
            })

        # Añadir relación bidireccional en la clase target
        if target in class_dict:
            class_dict[target]['attributes'].append({
                'name': f'{source.lower()}List',
                'visibility': 'private',
                'type': f'HashSet<{source}>'
            })

    return directed_associations

# =====
# GENERALIZACIONES
# =====
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

# =====
# ASOCIACIONES
# =====
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

# =====
# DEPENDENCIAS
# =====
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

# =====
# COMPOSICIONES
# =====
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
    })
if whole in class_dict:
    if multiplicity_target != "":
        class_dict[whole]['attributes'].append({
            'name': f'{part.lower()}List',
            'visibility': 'private',
            'type': f'ArrayList<{part}>'
        })
    else:
        class_dict[whole]['attributes'].append({
            'name': f'{part.lower()}List',
            'visibility': 'private',
            'type': f'TreeSet<{part}>'
        })
return compositions

# =====
# AGREGACIONES
# =====
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
    })
    if whole in class_dict:
        class_dict[whole]['attributes'].append({
            'name': f'{part.lower()}List',
            'visibility': 'private',
            'type': f'LinkedList<{part}>'
        })
    return aggregations

# =====
# GENERACIÓN DE HECHOS CLIPS
# =====
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('(defacts 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

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

    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

# =====
# ESCRITURA DE ARCHIVO CLIPS
# =====
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)
  )
  "")

# =====
# EJECUCIÓN DE CLIPS
# =====
import tempfile
import os
from clips import Environment

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

# =====
# PUNTO DE ENTRADA PRINCIPAL
# =====
if __name__ == '__main__':
    xmi_path = 'generated_files\\diagram.xmi'
    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(clips_file)
```

```
except ET.ParseError as e:
```

```
    print(f"Error al parsear el archivo XML: {e}")
```

UML.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Diagrama UML de Clases - Iván Falcón Monzón</title>

    <!-- Vincula el archivo de styles.css -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>
    <!-- Contenedor del formulario de entrada -->
    <div class="form-container">

        <!-- Entrada para agregar una nueva clase -->
        <label for="classNameInput">Nombre de la Clase:</label>
        <input type="text" id="classNameInput">
        <button onclick="addClass()">Agregar Clase</button>
        <br>

        <!-- Entrada para agregar un atributo a una clase -->
        <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>

        <!-- Entrada para agregar un método a una clase -->
        <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>

<!-- Entrada para definir relaciones entre clases UML -->
<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>

<!-- Botón para mostrar el código Java generado desde el servidor -->
<br><br>
<button onclick="mostrarCLP()">Código Java</button>

<!-- Área de texto donde se muestra el código Java -->
<pre id="clpOutput"></pre>

<!-- Botón para traducir el código Java a Python -->
<br><br>
<br><br><button onclick="traducirJavaAPython()">Traducir a Python</button>

<!-- Área de texto donde se mostrará el resultado en Python -->
  <pre id="pythonOutput" style="background-color: #f4f4f4; border: 1px solid #ccc; padding:
10px;"></pre>

</div>

<!-- Lienzo donde se dibujará el diagrama UML -->
<canvas id="umlCanvas" width="1024" height="768"></canvas>

<!-- Inclusión del archivo de scripts del proyecto -->

```

```

<script src="{{ url_for('static', filename='js/script.js') }}"></script>

<!-- Script adicional para hacer la traducción de Java a Python via API -->
<script>
    async function traducirJavaAPython() {
        const javaCode = document.getElementById('clpOutput').innerText;

        if (!javaCode.trim()) {
            alert("No hay código Java para traducir.");
            return;
        }

        try {
            const response = await fetch("/traducir_a_python", {
                method: "POST",
                headers: {
                    "Content-Type": "application/json"
                },
                body: JSON.stringify({ codigo_java: javaCode })
            });

            const data = await response.json();

            if (data.error) {
                alert("Error: " + data.error);
            } else {
                document.getElementById("pythonOutput").textContent = data.codigo_python || "No se recibió código traducido.";
            }

        } catch (error) {
            console.error("Error al traducir:", error);
            alert("Hubo un problema al conectar con el servidor.");
        }
    }
</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;');
}

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}" multiplicity="${rel.fromMultiplicity}" />\n`;
        xmi += `        <ownedEnd type="${rel.toClass.name}" multiplicity2="${rel.toMultiplicity}" />\n`;
    }
    xmi += `    </packagedElement>\n`;
});

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

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

function saveXMLToServer() {
    const xmi = generateXML();
    const blob = new Blob([xmi], { type: 'application/xml' });
    const formData = new FormData();
    formData.append('xmi', blob, 'diagram.xml');
    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() {
  saveXMLToServer();
  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);
    });
}

```