Ejercicio Puntual – Abstract Factory

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Integrantes:

Daniel Felipe Sanchez Garcia - 20211020110

Docente:

Henry Alberto Diosa

Universidad Distrital Francisco José de Caldas

Facultad de Ingeniería

DASP

Parte 1

Enunciado

Transformar el código fuente Java a Python o viceversa, según uno de los casos prácticos de recuperación de diseño que haya(n) sustentado en la primera ronda de ejercicios.

Descripción sucinta de la versión de las plataformas de IA Generativa, programación y ejecución de programas.

Versión de la plataforma de IA Generativa

Para la realización del ejercicio puntual, se utilizó la misma versión de IA para la parte 1 y 2, en este caso, se utilizó Claude Sonnet 4.5, la cual hace parte de los modelos Claude 4 de Anthropic.

Este modelo tiene las siguientes características:

- Es el modelo más inteligente de la familia Claude 4
- Eficiente para uso diario debido a su balanceo óptimo entre capacidad y rendimiento
- Fecha de conocimiento actualizado hasta finales de enero de 2025
- Fecha actual del sistema: de noviembre de 2025

Las capacidades destacadas de este modelo son:

- Razonamiento avanzado y análisis complejo
- Generación de código y artifacts interactivos
- Búsqueda web integrada para información actualizada
- Procesamiento de imágenes y documentos PDF
- Conversaciones largas con buen mantenimiento de contexto

Programación y ejecución de programas

Para la transformación del código, se trabajó con el patrón de diseño creacional “Abstract Factory”, el cual está escrito en Python y cuenta con 6 archivos los cuales son:

- AbstractFactory.py
- AutoSearchUI.py
- ButtonHandler.py
- CAR.py
- Minivan.py
- SUV.py

La ejecución de este código nos muestra una interfaz donde se puede seleccionar la categoría y el tipo de vehículo, y se muestra las características de estos, según la selección del usuario, además, de presentar la opción para salir del programa.

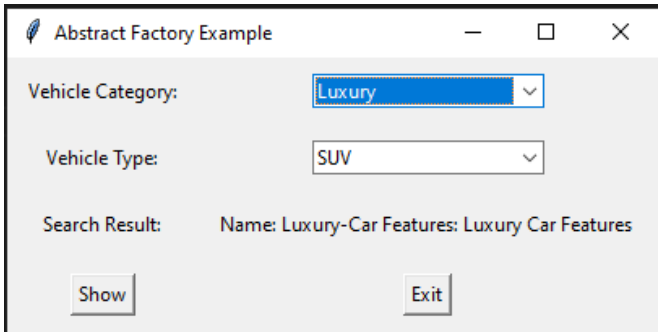

Figura 1. Interfaz generada tras ejecutar el programa.

“Prompts” utilizados para generar la transformación.

En este caso, para la utilización de este modelo de inteligencia artificial, se utilizó el siguiente “Prompt”:

> *Te pase una serie de archivos que conforman un solo programa escrito en Python y debes hacer lo siguiente:*
>
> *<instrucciones>*
>
> *1. lee y analiza detenidamente todos los archivos.*
>
> *2.Comprender el propósito y la funcionalidad de cada uno componente.*
>
> *3. Dame el mismo código, pero en el lenguaje de programación java y separado por archivos.*
>
> *4. verifica que el nuevo programa en java cumpla y satisfaga las mismas necesidades que el de Python*
>
> *</instrucciones>*

Códigos fuentes original y obtenido después de la transformación.

Código fuente Original escrito en Python

AbstractFactory.py

```
# VehicleFactory hierarchy
import abc
from SUV import *
from CAR import *
from MiniVan import *
class VehicleFactory:
    __metaclass__ = abc.ABCMeta
    LUXURY_VEHICLE="Luxury"
    NON_LUXURY_VEHICLE="Non-Luxury"
    def __init__(self):
        raise NotImplementedError()
    def getCar(self):
        raise NotImplementedError()
    def getSUV(self):
        raise NotImplementedError()
    def getMiniVan(self):
        raise NotImplementedError()
    def getVehicleFactory(tipo):
        if tipo==VehicleFactory.LUXURY_VEHICLE:
            return LuxuryVehicleFactory()
        if tipo==VehicleFactory.NON_LUXURY_VEHICLE:
            return NonLuxuryVehicleFactory()
        return LuxuryVehicleFactory()
    getVehicleFactory = staticmethod(getVehicleFactory)
```

```
#End of class
class LuxuryVehicleFactory(VehicleFactory):
    def __init__(self):
        pass
    def getCar(self):
        return LuxuryCar("Luxury-Car")
    def getSUV(self):
        return LuxurySUV("Luxury-SUV")
    def getMiniVan(self):
        return LuxuryMinivan("Luxury Mini-Van")
#End of class
class NonLuxuryVehicleFactory(VehicleFactory):
    def __init__(self):
        pass
    def getCar(self):
        return NonLuxuryCar("Non Luxury-Car")
    def getSUV(self):
        return NonLuxurySUV("Non Luxury-SUV")
    def getMiniVan(self):
        return NonLuxuryMinivan("Non Luxury-Mini-Van")
#End of class
```

AutoSearchUI.py

```
from tkinter import ttk
from tkinter import *
from ButtonHandler import *
from AbstractFactory import *
import SUV
import CAR
import MiniVan
class AutoSearchUI(Toplevel):
    SEARCH = "Show"
    EXIT = "Exit"
    CAR = "Car"
    SUV = "SUV"
    MiniVan = "Mini-Van"
    def __init__(self, master):
        Toplevel.__init__(self, master)
        self.__cmbVehicleCategory = ttk.Combobox(self, state="readonly")
        self.__cmbVehicleCategory["values"] =
[VehicleFactory.LUXURY_VEHICLE,
                                    VehicleFactory.NON_LUXURY_VEHICLE]
        self.__cmbVehicleCategory.current(0)
        self.__cmbVehicleType = ttk.Combobox(self, state="readonly")
        self.__cmbVehicleType["values"] = [self.CAR, self.SUV, self.MiniVan]
        self.__cmbVehicleType.current(0)
        self.__lblVehicleCategory = Label(self, text="Vehicle Category:")
        self.__lblVehicleType = Label(self, text="Vehicle Type:")
        self.__lblCarName = Label(self, text="Search Result:")
        self.__lblCarNameValue = Label(self, text=" Please click on Show button")
```

```
        self.__openButton = Button(self, text=self.SEARCH)
        self.__exitButton = Button(self, text=self.EXIT)
        #
**********************************************************************************************
***********#
        self.__lblVehicleCategory.grid(row=1, column=1, padx=10, pady=10)
        self.__cmbVehicleCategory.grid(row=1, column=2, padx=10, pady=10)
        self.__lblVehicleType.grid(row=2, column=1, padx=10, pady=10)
        self.__cmbVehicleType.grid(row=2, column=2, padx=10, pady=10)
        self.__lblCarName.grid(row=3, column=1, padx=10, pady=10)
        self.__lblCarNameValue.grid(row=3, column=2, padx=10, pady=10)
        self.__openButton.grid(row=4, column=1, padx=10, pady=10)
        self.__exitButton.grid(row=4, column=2, padx=10, pady=10)
    def getExitButton(self):
        return self.__exitButton
    def getOpenButton(self):
        return self.__openButton
    def getCmbVehicleCategory(self):
        return self.__cmbVehicleCategory
    def getCmbVehicleType(self):
        return self.__cmbVehicleType
    def getLblCarNameValue(self):
        return self.__lblCarNameValue
    def setText(self, s):
        self.__lblCarNameValue.config(text=s)
# End of class
# Main method
def main():
    root = Tk()
    root.withdraw()
    root.title("Abstract Factory Example")
    app = ButtonHandler(root)
    root.mainloop()
# Executing the main method
if __name__ == "__main__":
    main()
```

ButtonHandler.py

```
from tkinter import *
from tkinter import ttk
from AbstractFactory import *
from AutoSearchUI import *
class ButtonHandler:
    def __init__(self,root):
        self.__root=root
        self.__frame=AutoSearchUI(root)
        self.__frame.getExitButton().config(command=self.eventExitButton)
        self.__frame.getOpenButton().config(command=self.eventOpenButton)
    def eventExitButton(self):
        self.__frame.destroy()
        self.__root.destroy()
```

```
    def eventOpenButton(self):
        vhCategory = self.__frame.getCmbVehicleCategory().get()
        vhType = self.__frame.getCmbVehicleType().get()
        vf = VehicleFactory.getVehicleFactory(vhCategory)
        if vhType == self.__frame.CAR:
            c = vf.getCar()
            searchResult = "Name: " + c.getCarName() + " Features: " +
c.getCarFeatures()
        if vhType == self.__frame.SUV:
            s = vf.getSUV()
            searchResult = "Name: " + s.getSUVName() + " Features: " +
s.getSUVFeatures()
        if vhType == self.__frame.MiniVan:
            s = vf.getMiniVan()
            searchResult = "Name: " + s.getMinivanName() + " Features: " +
s.getMinivanFeatures()
        self.__frame.setText(searchResult)
#End of class
```

CAR.py

```
import abc
class Car:
    __metaclass__ = abc.ABCMeta
    def __init__(self):
        raise NotImplementedError()
    def getCarName(self):
        raise NotImplementedError()
    def getCarFeatures(self):
        raise NotImplementedError()
# End of interface (Class as interface)
class LuxuryCar(Car):
    def __init__(self, cName):
        self.__name = cName
    def getCarName(self):
        return self.__name
    def getCarFeatures(self):
        return "Luxury Car Features "
#End of class
class NonLuxuryCar(Car):
    def __init__(self, cName):
        self.__name = cName
    def getCarName(self):
        return self.__name
    def getCarFeatures(self):
        return "Non-Luxury Car Features "
#End of class
```

Minivan.py

```
import abc
class Minivan:
    __metaclass__= abc.ABCMeta
```

```
    def __init__(self):
        raise NotImplementedError()

    def  getMinivanName(self):
      raise NotImplementedError()
    def getMinivanFeatures(self):
      raise NotImplementedError()
class LuxuryMinivan(Minivan):
    def __init__(self, mName):
      self.__name = mName
    def getMinivanName(self):
       return self.__name
    def getMinivanFeatures(self):
       return "Luxury Mini-Van features"
class NonLuxuryMinivan(Minivan):
  def __init__(self, mName):
     self.__name = mName
  def getMinivanName(self):
     return self.__name
  def getMinivanFeatures(self):
     return "Non-luxury Mini-Van Fatures"
```

SUV.py

```
import abc
class SUV:
    __metaclass__ = abc.ABCMeta
    def __init__(self):
        raise NotImplementedError()
    def getSUVName(self):
        raise NotImplementedError()
    def getSUVFeatures(self):
        raise NotImplementedError()
# End of interface (Class as interface)
class LuxurySUV(SUV):
    def __init__(self, sName):
        self.__name = sName
    def getSUVName(self):
        return self.__name;
    def getSUVFeatures(self):
        return "Luxury SUV Features ";
#End of class
class NonLuxurySUV(SUV):
    def __init__(self, sName):
        self.__name = sName
    def getSUVName(self):
        return self.__name
    def getSUVFeatures(self):
        return "Non-Luxury SUV Features "
#End of class
```

Código fuente transformado en Java

AutoSearchUI.java

```
// AutoSearchUI - GUI Interface
import javax.swing.*;
import java.awt.*;
public class AutoSearchUI extends JFrame {
    public static final String SEARCH = "Show";
    public static final String EXIT = "Exit";
    public static final String CAR = "Car";
    public static final String SUV = "SUV";
    public static final String MINIVAN = "Mini-Van";
    private JComboBox<String> cmbVehicleCategory;
    private JComboBox<String> cmbVehicleType;
    private JLabel lblVehicleCategory;
    private JLabel lblVehicleType;
    private JLabel lblCarName;
    private JLabel lblCarNameValue;
    private JButton openButton;
    private JButton exitButton;

    public AutoSearchUI() {
        setTitle("Abstract Factory Example");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        // Initialize components
        cmbVehicleCategory = new JComboBox<>(new String[]{
            VehicleFactory.LUXURY_VEHICLE,
            VehicleFactory.NON_LUXURY_VEHICLE
        });

        cmbVehicleType = new JComboBox<>(new String[]{CAR, SUV,
MINIVAN});

        lblVehicleCategory = new JLabel("Vehicle Category:");
        lblVehicleType = new JLabel("Vehicle Type:");
        lblCarName = new JLabel("Search Result:");
        lblCarNameValue = new JLabel(" Please click on Show button");

        openButton = new JButton(SEARCH);
        exitButton = new JButton(EXIT);

        // Add components to grid
        gbc.gridx = 0;
        gbc.gridy = 0;
        add(lblVehicleCategory, gbc);
```

```
        gbc.gridx = 1;
        gbc.gridy = 0;
        add(cmbVehicleCategory, gbc);
        gbc.gridx = 0;
        gbc.gridy = 1;
        add(lblVehicleType, gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        add(cmbVehicleType, gbc);
        gbc.gridx = 0;
        gbc.gridy = 2;
        add(lblCarName, gbc);
        gbc.gridx = 1;
        gbc.gridy = 2;
        add(lblCarNameValue, gbc);
        gbc.gridx = 0;
        gbc.gridy = 3;
        add(openButton, gbc);
        gbc.gridx = 1;
        gbc.gridy = 3;
        add(exitButton, gbc);
        pack();
        setLocationRelativeTo(null);
    }

    public JButton getExitButton() {
        return exitButton;
    }

    public JButton getOpenButton() {
        return openButton;
    }

    public JComboBox<String> getCmbVehicleCategory() {
        return cmbVehicleCategory;
    }

    public JComboBox<String> getCmbVehicleType() {
        return cmbVehicleType;
    }

    public JLabel getLblCarNameValue() {
        return lblCarNameValue;
    }

    public void setText(String s) {
        lblCarNameValue.setText(s);
    }
}
```

ButtonHandler.java

```
// ButtonHandler - Event Handler

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ButtonHandler {
   private AutoSearchUI frame;

   public ButtonHandler(AutoSearchUI frame) {
      this.frame = frame;

      // Add action listeners
      frame.getExitButton().addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            eventExitButton();
         }
      });

      frame.getOpenButton().addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            eventOpenButton();
         }
      });
   }

   private void eventExitButton() {
      frame.dispose();
      System.exit(0);
   }

   private void eventOpenButton() {
      String vhCategory = (String)
frame.getCmbVehicleCategory().getSelectedItem();
      String vhType = (String) frame.getCmbVehicleType().getSelectedItem();
      VehicleFactory vf = VehicleFactory.getVehicleFactory(vhCategory);
      String searchResult = "";

      if (vhType.equals(AutoSearchUI.CAR)) {
         Car c = vf.getCar();
         searchResult = "Name: " + c.getCarName() + " Features: " +
c.getCarFeatures();
      }
      if (vhType.equals(AutoSearchUI.SUV)) {
         SUV s = vf.getSUV();
         searchResult = "Name: " + s.getSUVName() + " Features: " +
s.getSUVFeatures();
      }
```

```
        if (vhType.equals(AutoSearchUI.MINIVAN)) {
            Minivan m = vf.getMiniVan();
            searchResult = "Name: " + m.getMinivanName() + " Features: " +
m.getMinivanFeatures();
        }

        frame.setText(searchResult);
    }
}
```

Car.java

```
// Car hierarchy
public abstract class Car {
    public abstract String getCarName();
    public abstract String getCarFeatures();
}
class LuxuryCar extends Car {
    private String name;
    public LuxuryCar(String cName) {
        this.name = cName;
    }
    @Override
    public String getCarName() {
        return this.name;
    }
    @Override
    public String getCarFeatures() {
        return "Luxury Car Features";
    }
}
class NonLuxuryCar extends Car {
    private String name;
    public NonLuxuryCar(String cName) {
        this.name = cName;
    }
    @Override
    public String getCarName() {
        return this.name;
    }
    @Override
    public String getCarFeatures() {
        return "Non-Luxury Car Features";
    }
}
```

Main.java

```
import javax.swing.SwingUtilities;
public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
```

```
            @Override
            public void run() {
                AutoSearchUI frame = new AutoSearchUI();
                ButtonHandler handler = new ButtonHandler(frame);
                frame.setVisible(true);
            }
        });
    }
}
```

Minivan.java

```
// Minivan hierarchy
public abstract class Minivan {
    public abstract String getMinivanName();
    public abstract String getMinivanFeatures();
}
class LuxuryMinivan extends Minivan {
    private String name;

    public LuxuryMinivan(String mName) {
        this.name = mName;
    }
    @Override
    public String getMinivanName() {
        return this.name;
    }
    @Override
    public String getMinivanFeatures() {
        return "Luxury Mini-Van features";
    }
}
class NonLuxuryMinivan extends Minivan {
    private String name;

    public NonLuxuryMinivan(String mName) {
        this.name = mName;
    }
    @Override
    public String getMinivanName() {
        return this.name;
    }
    @Override
    public String getMinivanFeatures() {
        return "Non-luxury Mini-Van Features";
    }
}
```

SUV.java

```
// SUV hierarchy
```

```
public abstract class SUV {
    public abstract String getSUVName();
    public abstract String getSUVFeatures();
}
class LuxurySUV extends SUV {
    private String name;
    public LuxurySUV(String sName) {
        this.name = sName;
    }
    @Override
    public String getSUVName() {
        return this.name;
    }
    @Override
    public String getSUVFeatures() {
        return "Luxury SUV Features";
    }
}
class NonLuxurySUV extends SUV {
    private String name;
    public NonLuxurySUV(String sName) {
        this.name = sName;
    }
    @Override
    public String getSUVName() {
        return this.name;
    }
    @Override
    public String getSUVFeatures() {
        return "Non-Luxury SUV Features";
    }
}
```

VehicleFactory.java

```
// VehicleFactory hierarchy
public abstract class VehicleFactory {
    public static final String LUXURY_VEHICLE = "Luxury";
    public static final String NON_LUXURY_VEHICLE = "Non-Luxury";
    public abstract Car getCar();
    public abstract SUV getSUV();
    public abstract Minivan getMiniVan();
    public static VehicleFactory getVehicleFactory(String tipo) {
        if (tipo.equals(LUXURY_VEHICLE)) {
            return new LuxuryVehicleFactory();
        }
        if (tipo.equals(NON_LUXURY_VEHICLE)) {
            return new NonLuxuryVehicleFactory();
        }
        return new LuxuryVehicleFactory();
    }
```

```
}
class LuxuryVehicleFactory extends VehicleFactory {
    @Override
    public Car getCar() {
        return new LuxuryCar("Luxury-Car");
    }
    @Override
    public SUV getSUV() {
        return new LuxurySUV("Luxury-SUV");
    }
    @Override
    public Minivan getMiniVan() {
        return new LuxuryMinivan("Luxury Mini-Van");
    }
}
class NonLuxuryVehicleFactory extends VehicleFactory {
    @Override
    public Car getCar() {
        return new NonLuxuryCar("Non Luxury-Car");
    }
    @Override
    public SUV getSUV() {
        return new NonLuxurySUV("Non Luxury-SUV");
    }
    @Override
    public Minivan getMiniVan() {
        return new NonLuxuryMinivan("Non Luxury-Mini-Van");
    }
}
```

Análisis sobre el resultado obtenido

En este caso, la IA generativa realizo la conversión del código y genero cada uno de los archivos que ya había en el programa de Python, además, agrego un archivo llamado “main”.

Sin embargo, cuando se descargaron los códigos, los nombres de los archivos no coincidían con las clases, por lo que tuvo que cambiarse el nombre de cada archivo para que funcione correctamente.

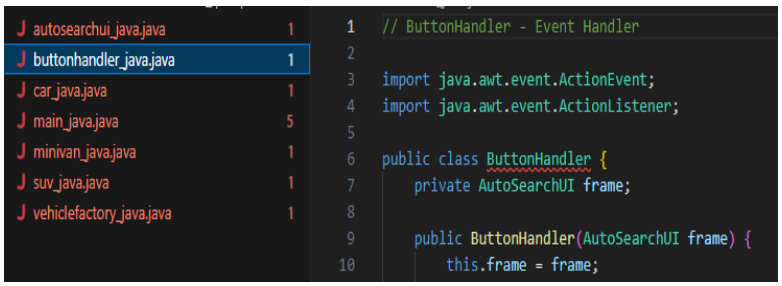

Figura 2. Archivos generados por la IA, abiertos en visual studio

Una vez hecha esa corrección, se ejecutó el programa en java y se obtuvo la siguiente interfaz:

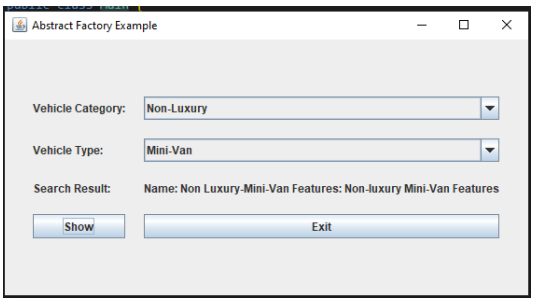

Figura 3. Interfaz generada al ejecutar el programa dado por la IA

Como se puede apreciar, tiene las mismas características que la interfaz del programa en Python y cumple con la misma funcionalidad.

Reflexiones finales.

En cuanto a las reflexiones que pueden hacerse, basándonos en el ejercicio de transformación de código de un lenguaje a otro (Python a Java), se puede decir lo siguiente:

La IA es capaz de realizar la transformación de código correctamente, siempre y cuando, se le den las instrucciones necesarias y adecuadas para la comprensión y ejecución de la tarea.

En cuanto al código generado por la IA, no se recomienda confiar plenamente en lo que se otorga, ya que como, por ejemplo, el nombre de los archivos que se generaron no coincidía con el nombre de las clases principales de cada archivo por lo que tuvo que realizarse la corrección para lograr ejecutar el programa.

La IA, tardo un aproximado de 40 segundos en realizar la conversión de código de un lenguaje a otro, lo cual es bastante más rápido que si se hace de forma manual, además, la IA genero una explicación de cómo funciona cada parte del programa, lo cual resulta bastante útil al momento de estudiar el código generado.

Parte 2

Enunciado

Usando la IA Generativa intentar recuperar el diseño con el mismo alcance de los ejercicios prácticos desarrollados en clase:

- Modelo funcional (Diagrama) basado en casos de uso.
- Modelo estructural (Diagrama de clases).
- Diagramas de secuencia.

"Prompts" utilizados para generar los modelos.

Primer prompt para generar el diagrama de clases

Te pase una serie de archivos que conforman un solo programa escrito en Python y debes hacer lo siguiente:
<instructions>
1. lee y analiza detenidamente todos los archivos.
2.Comprender el propósito y la funcionalidad de cada uno componente.
3. Realiza el modelo estructural (diagrama de clases) del programa.
4. verifica que el diagrama de clases represente de forma adecuada el programa respetando el patrón de diseño que se está usando
</instructions>

Segundo prompt para corrección del diagrama de clases

verifica:
el buttonhandler usa las clases abstractas car, suv y minivan?
también revisa que por ejemplo suv, está generalizando de minvan, lo que es erróneo

Primer prompt usado para la generación del diagrama de casos de uso

Te pase una serie de archivos que conforman un solo programa escrito en Python y debes hacer lo siguiente:
<instructions>
1. lee y analiza detenidamente todos los archivos.

2.Comprender el propósito y la funcionalidad de cada uno componente.

3. ten en cuenta el modelo estructural (diagrama de clases) del programa que me diste.

4. crea el diagrama de casos de uso que represente el programa y cada una de sus características.

5. verifica que el diagrama de casos de uso represente de forma adecuada el programa respetando el patrón de diseño que se está usando

</instructions>

Segundo prompt utilizado para corregir aspectos del diagrama de casos de uso

verifica que el usuario no debe conectarse a varios casos de uso, es mejor que se asocie con uno que será el que inicializa la aplicación

Tercer prompt utilizado para corregir aspectos del diagrama de casos de uso

verifica los extends, pues al iniciar la aplicación en la interfaz veo dos opciones que son: salir y mostrar información

Prompt utilizado para la generación de los diagramas de secuencia.

Te pase una serie de archivos que conforman un solo programa escrito en Python y debes hacer lo siguiente:

<instructions>

1. lee y analiza detenidamente todos los archivos.

2.Comprender el propósito y la funcionalidad de cada uno componente.

3. ten en cuenta el modelo estructural (diagrama de clases) del programa que me diste.

4.ten en cuenta el diagrama de casos de uso que represente el programa y cada una de sus características

5. De cada caso de uso proporcionado, generar el diagrama de secuencia a cada uno, verificándolos con los códigos correspondiente en los archivos del programa

</instructions>

Análisis comparativo de los modelos elaborados en clase respecto a los generados con la IA Generativa.

Diagrama de clases

Original

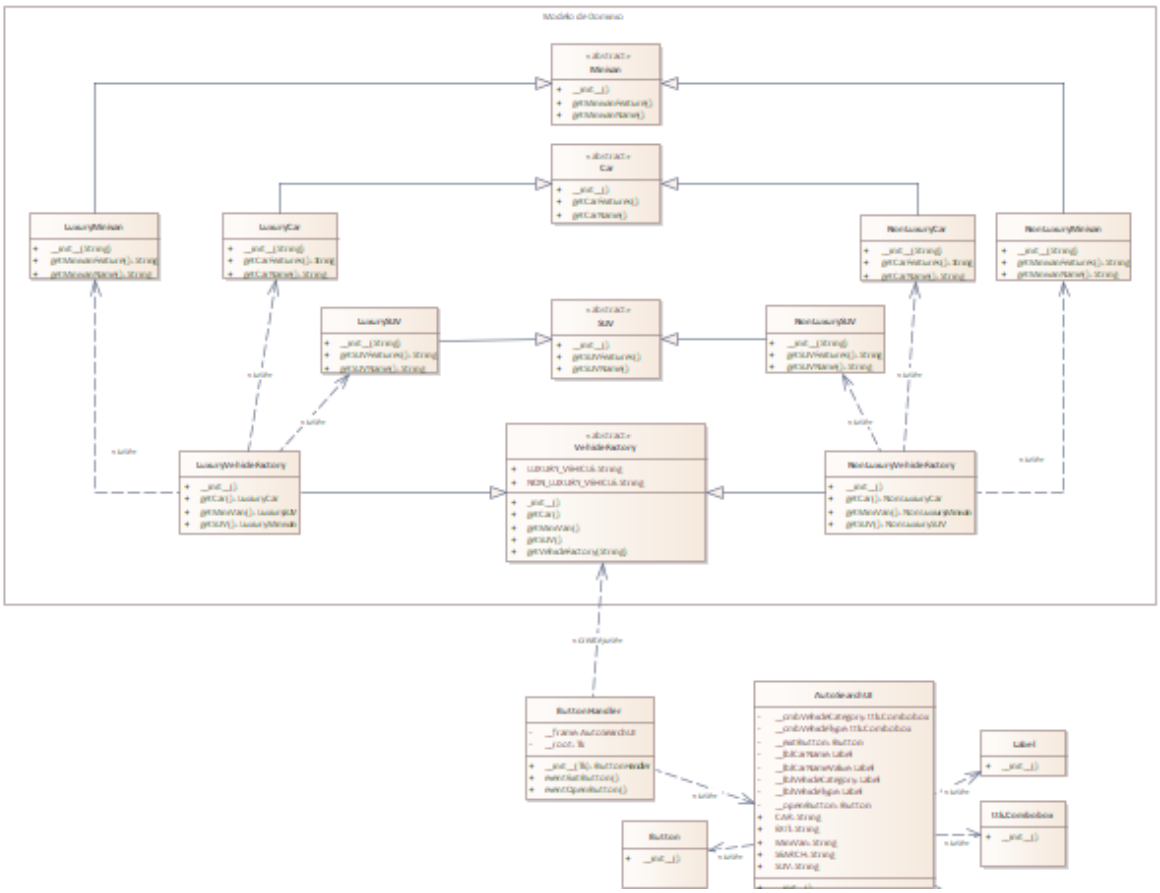

Figura 3. Diagrama de clases realizado de forma manual

Diagramas Generados por la IA

Para la primera versión generada del diagrama de clases, la inteligencia artificial creo de manera adecuada las clases que representan el programa, sin embargo, las clases que hacen parte propiamente de java, no las reflejo en el diagrama como si se hizo en el diagrama hecho manualmente.

En este caso, se realizaron erróneamente algunas relaciones entre la clase ButtonHandler y las clases abstractas de cada tipo de vehículo.

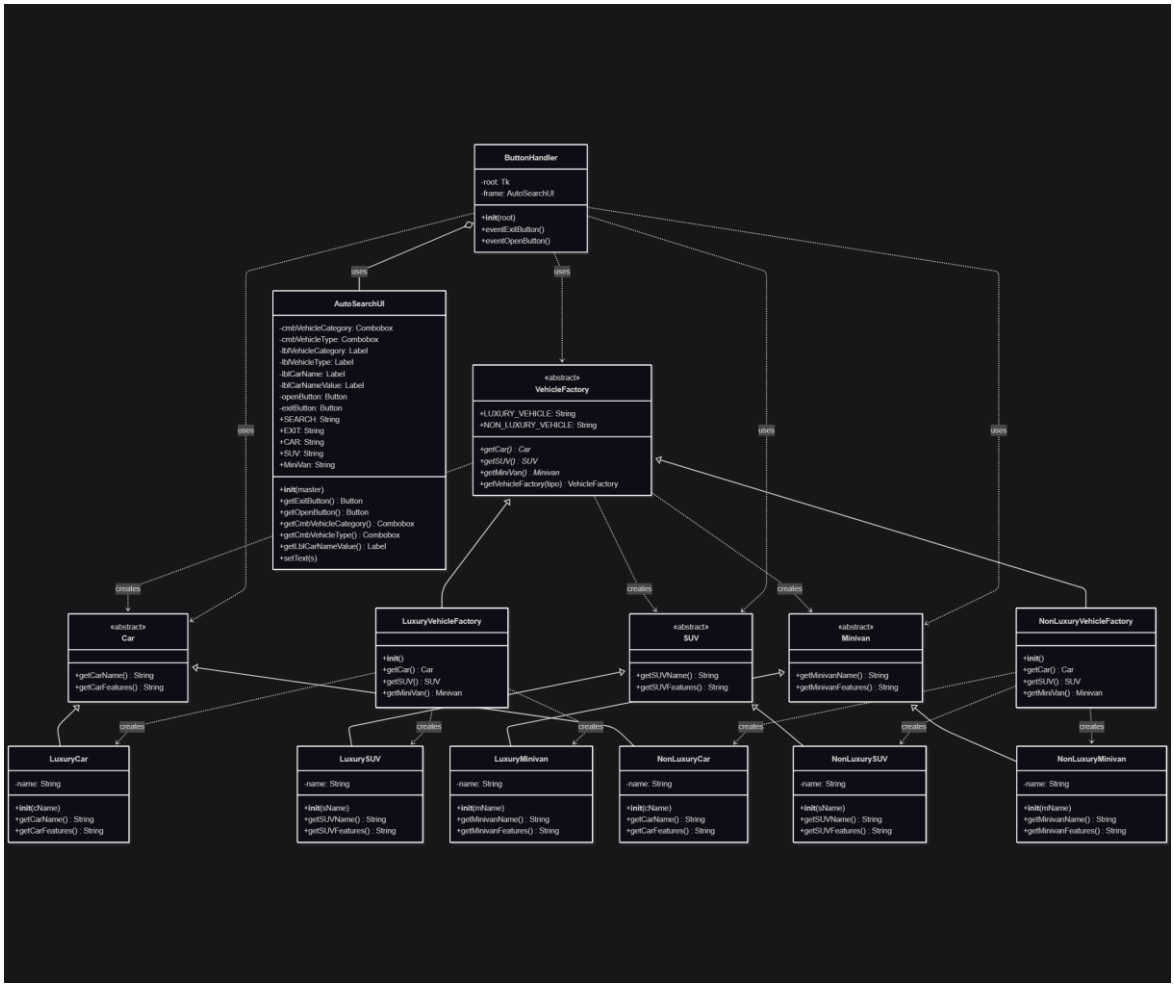

Figura 4. Version 1 del diagrama de clases realizado por la IA

Para la segunda version, se corrigieron las relaciones que se tenian mal en la versión anterior y la IA logro representar de manera adecuada el programa y prácticamente genero un diagrama igual al realizado de forma manual.

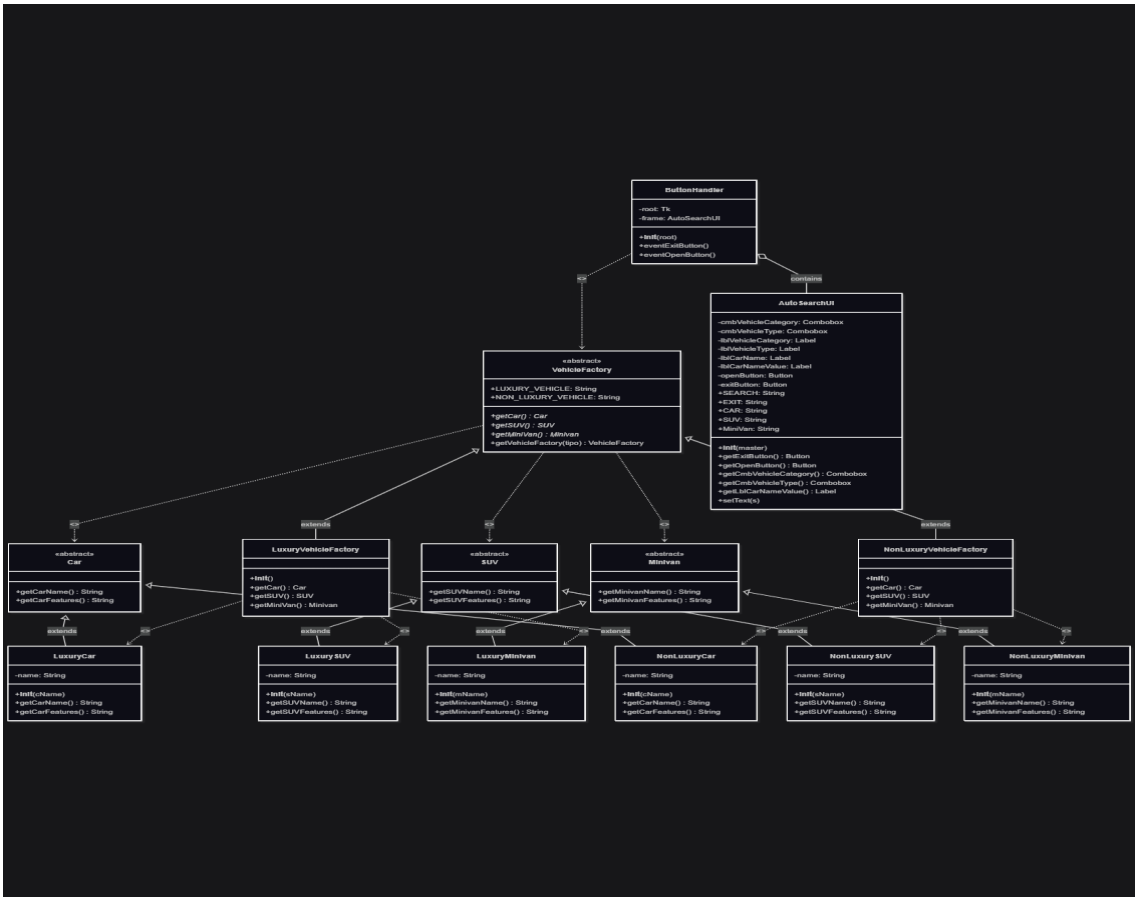

Figura 5. Versión 1 del diagrama de clases realizado por la IA

Diagrama de casos de uso

Original

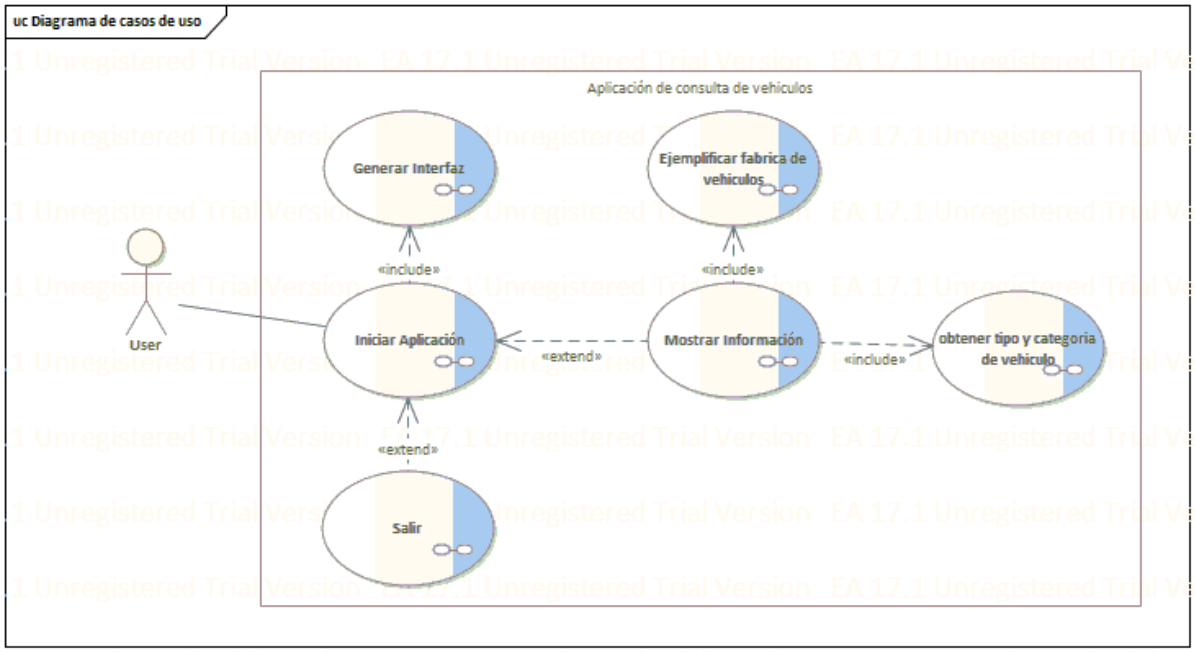

Figura 6. Diagrama de casos de uso realizado de forma manual.

Generado por la IA

En cuanto a la generación del diagrama de casos de uso, la IA entrego una primera versión en la que se presentaban varios errores como la asociación del usuario con varios casos de uso o la generación de un apartado para procesos internos, cosa que no está presente en el diagrama realizado manualmente.

Para la segunda versión, se corrigió la conexión del usuario con varios casos de uso, pero se identificaron problemas con la utilización de los “extends” ya que solo estaba usando uno para la opción “salir” y no estaba representando correctamente la otra opción correspondiente a “Mostrar la información del vehículo” y se sigue usando una sección para “procesos internos”

Por último, se le pidió a la IA que para la tercera versión corrigiera los errores encontrados en la versión anterior, y aunque represento de manera

correcta los “extends”, realizo unas relaciones “include” con el caso de uso principal, a diferencia del diagrama realizado manualmente donde este no incluye ninguna asociación de este tipo con el caso de uso principal.

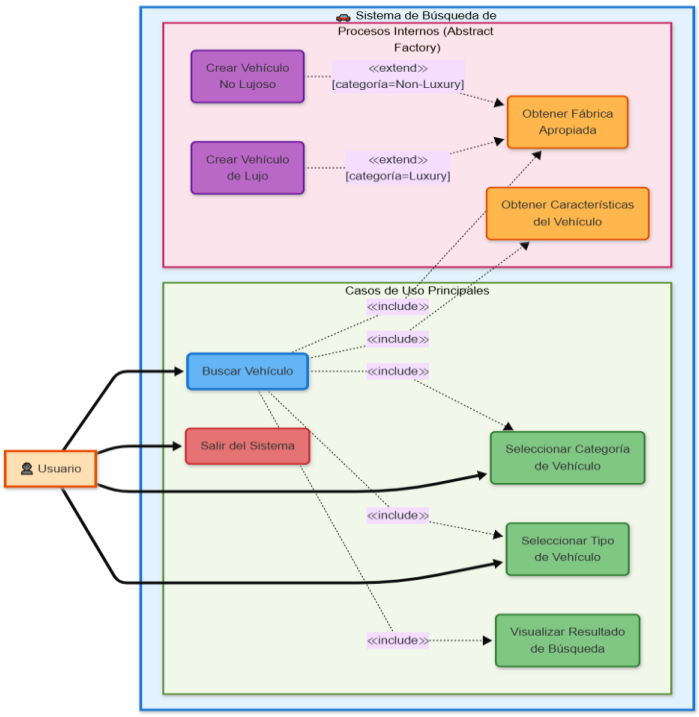

Figura 7. Version 1 del diagrama de casos de uso generado por la IA

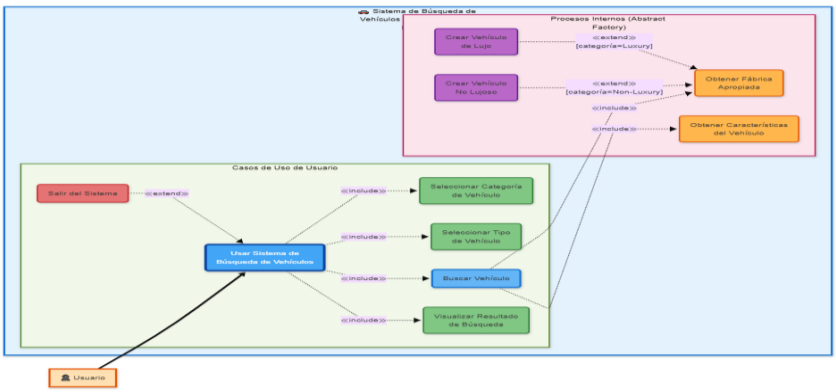

Figura 8. Version 2 del diagrama de casos de uso generado por la IA

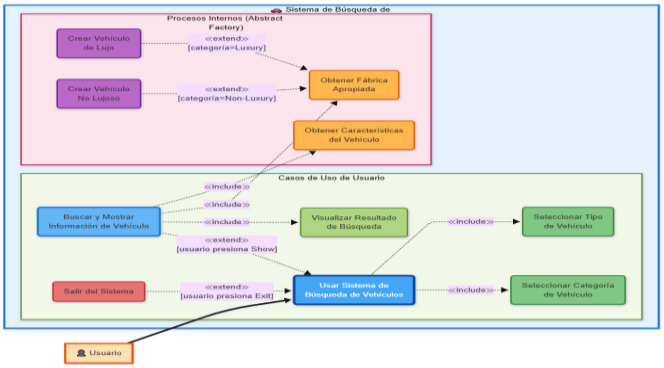

Figura 9. Version 3 del diagrama de casos de uso generado por la IA

Diagramas de secuencia

Diagramas de secuencia originales

Ejemplificar fábrica de vehículos

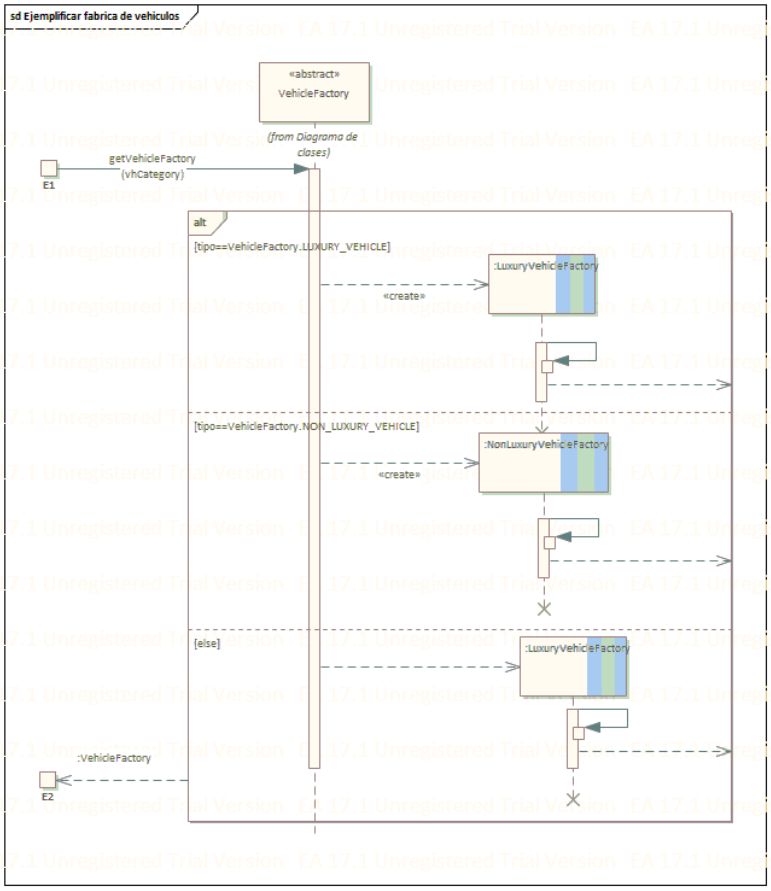

Figura 10. Caso de uso original Ejemplificar fábrica de vehículo

Iniciar aplicación.

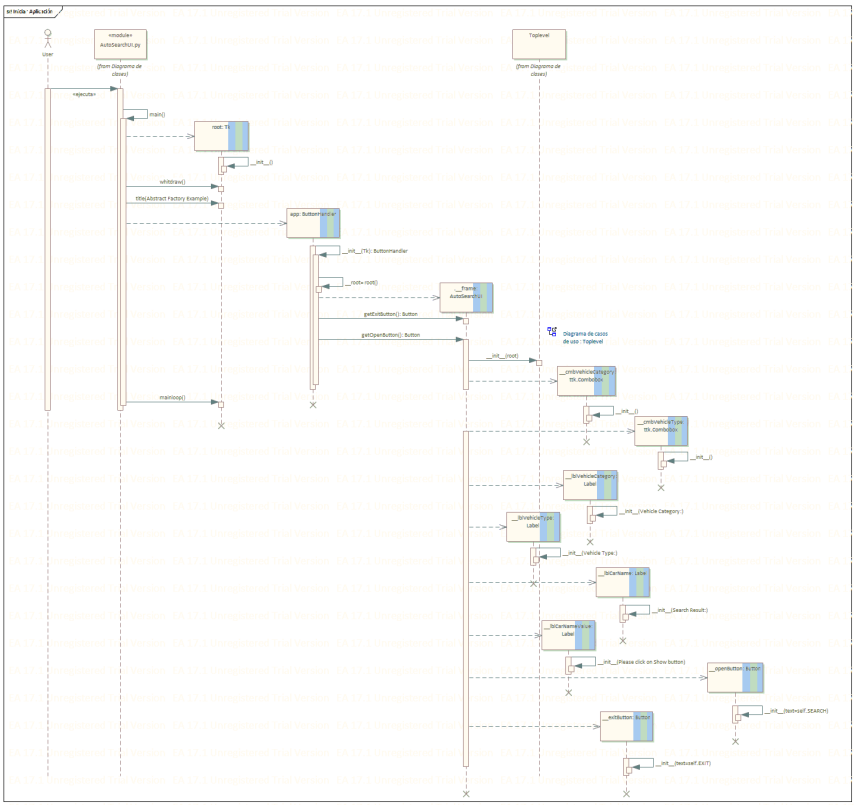

Figura 11. Caso de uso original iniciar aplicación

Obtener tipo y categoría de vehículo

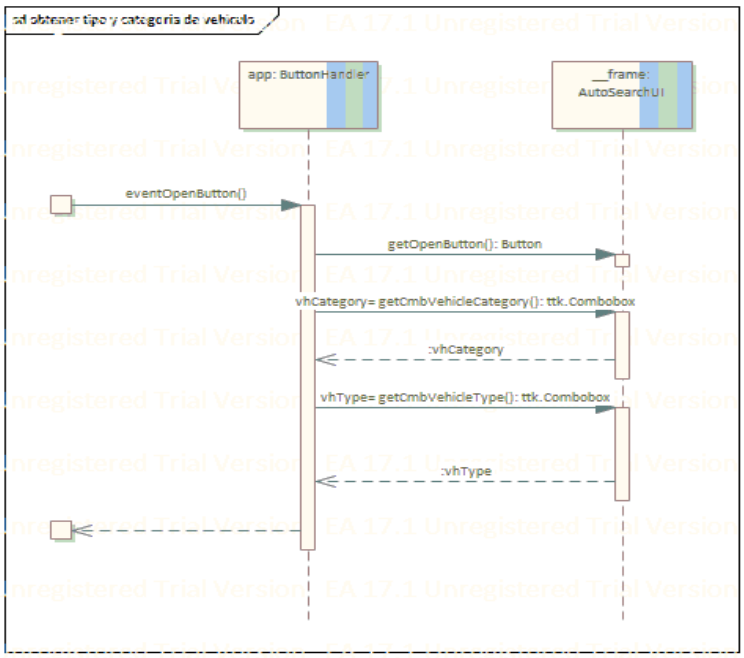

Figura 12. Caso de uso original Obtener tipo y categoría de vehículo

Salir

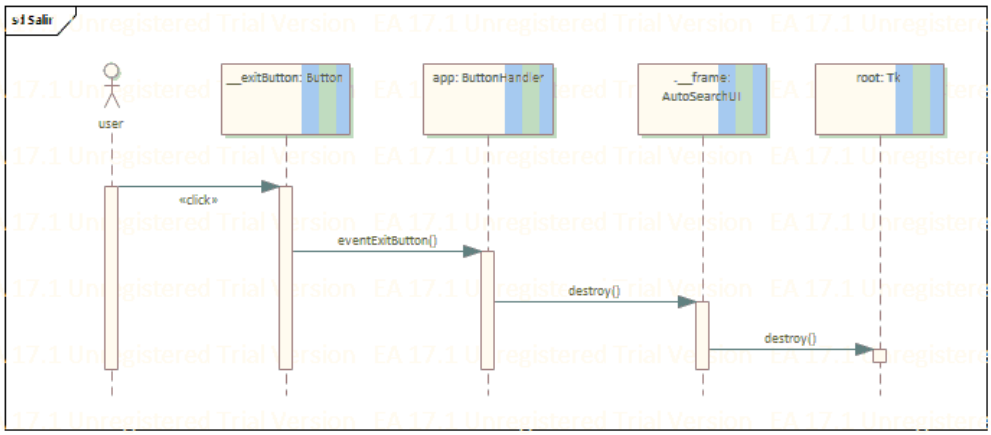

Figura 13. Caso de uso original salir

Mostrar Información

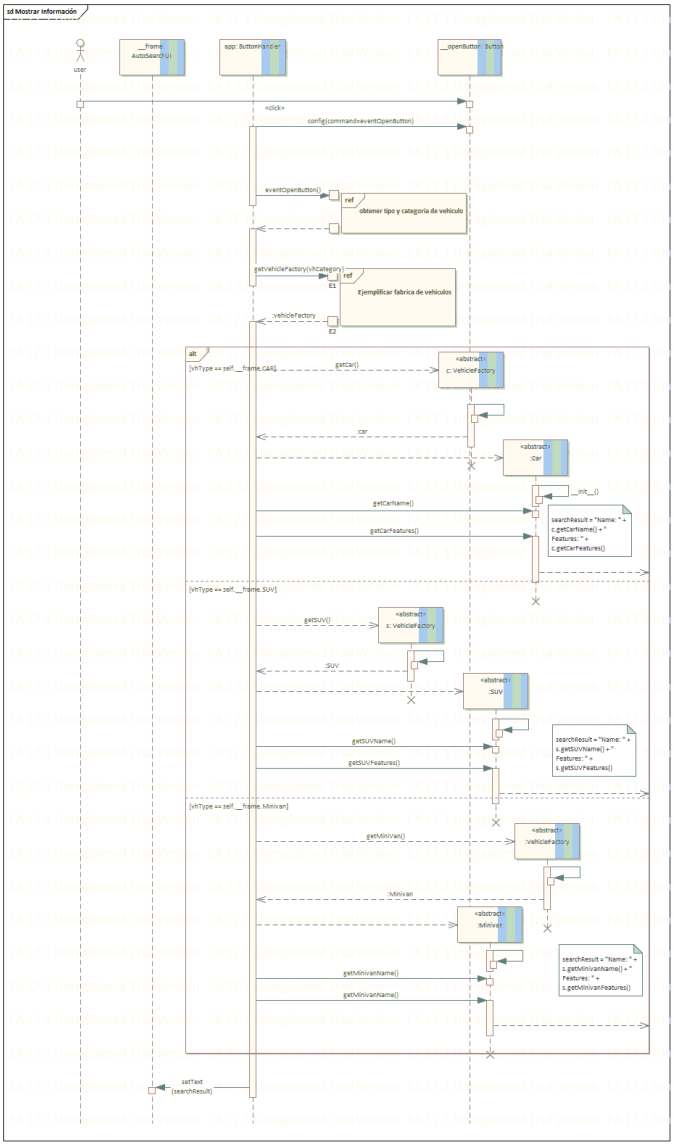

Figura 14. Caso de uso original mostrar información.

Diagramas de secuencia Generados por la IA

Buscar Luxury

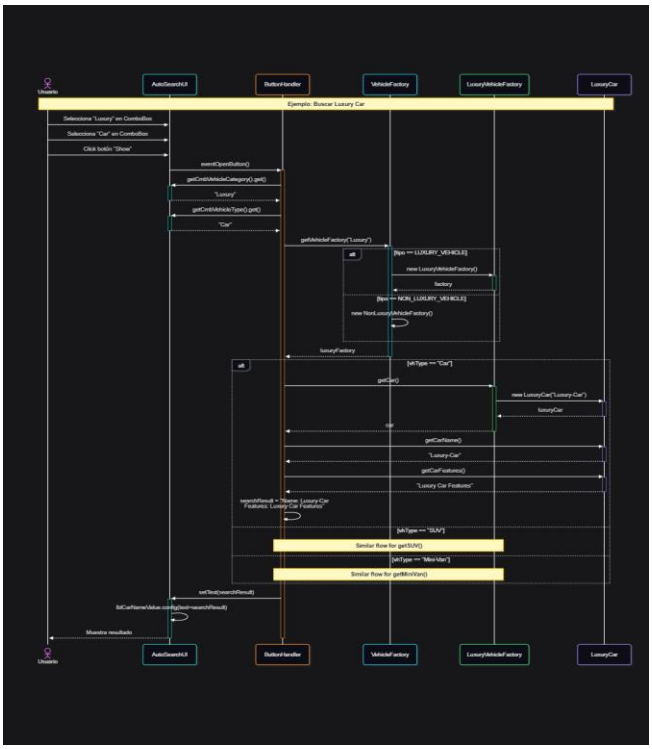

Figura 14. Caso de uso generado por la IA Buscar Luxury.

Buscar Non-Luxury

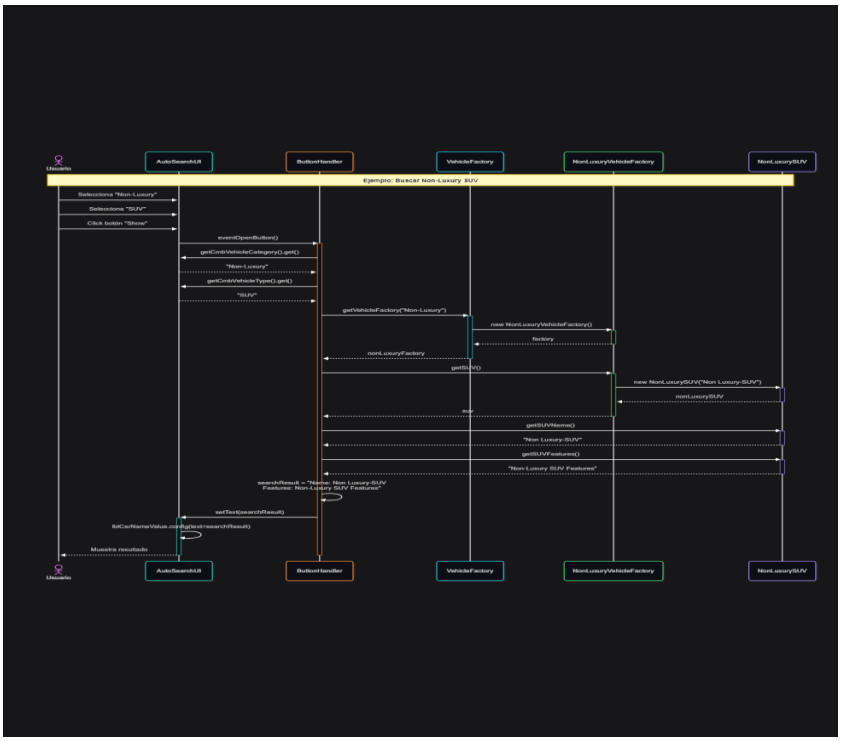

Figura 15. Caso de uso generado por la IA Buscar Non-Luxury.

Salir

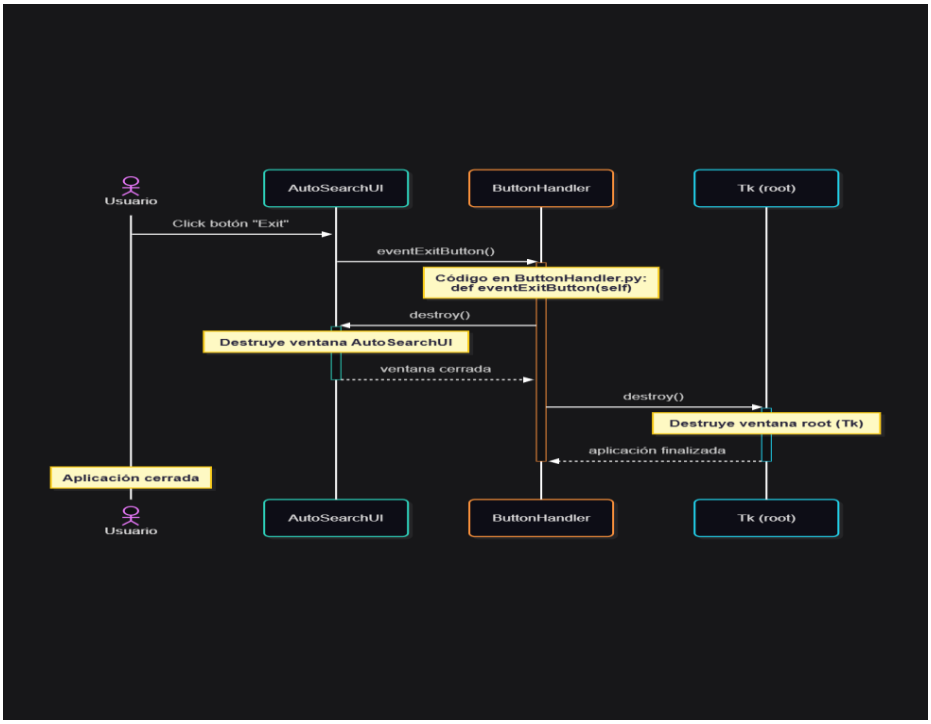

Figura 16. Caso de uso generado por la IA Salir.

Seleccionar Categoría

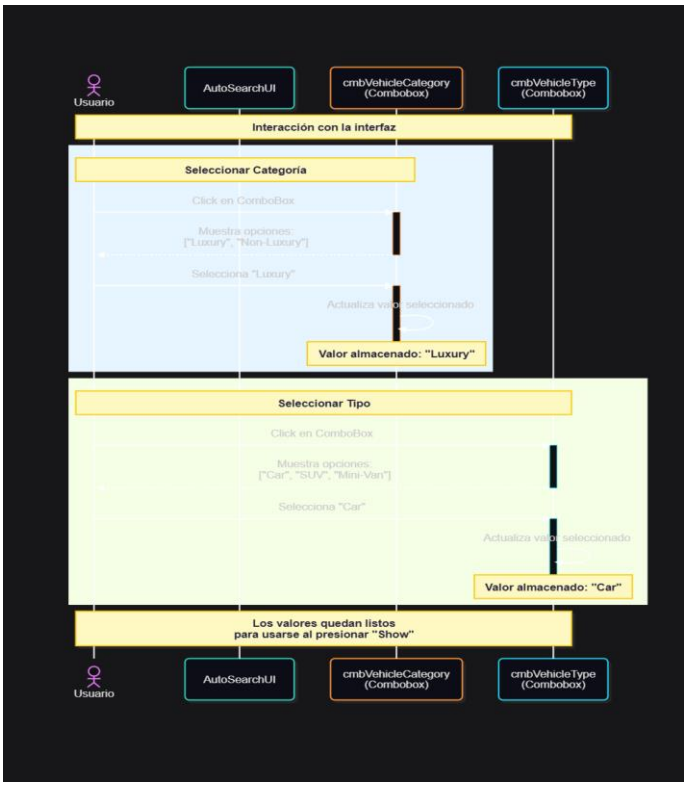

Figura 17. Caso de uso generado por la seleccionar categoría

Inicializar

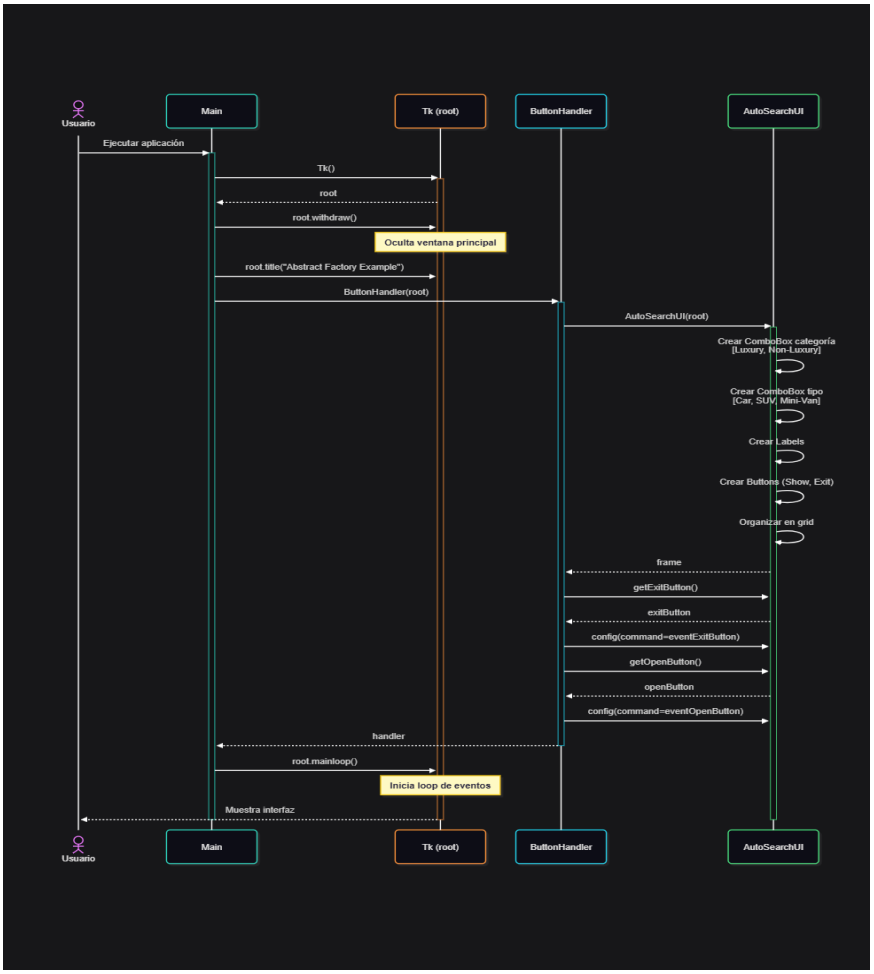

Figura 18. Caso de uso generado por la IA Inicializar.

En cuanto a la generación de los diagramas de secuencia, la inteligencia artificial realizo una representación de cada caso de uso incluyendo algunos elementos y formas de representación distintas a los modelos originales, tales elementos son:

- Las líneas de vida de cada clase representadas desde el inicio hasta el final del diagrama de secuencia con el nombre de cada clase.
- La implementación de notas de manera más frecuente que en los modelos originales.

Además de eso, aunque la inteligencia represento los modelos de caso de uso de manera distinta, y por ende los diagramas de secuencia se

realizaron de manera diferente, aunque manteniendo un nivel de similitud muy acercado al de los diagramas de secuencia originales y logrando representar de manera aceptable el programa.

También se pudo apreciar que en el caso de uso “buscar luxury” la IA opto por agregar notas como “Similar a lo anterior”, con el fin de evitar hacer el mismo procedimiento varias veces.

Reflexiones finales.

Respecto al ejercicio realizado para la parte 2 de la actividad, se puede reconocer que la IA, tuvo muchas más complicaciones para realizar el modelado del modelo estructural, funcional y los diagramas de clases. Aun así, con algunas instrucciones dadas, logro acercarse bastante a los modelos realizados de manera manual.

En cuanto a la generación de estos modelos usando IA, es necesario aclarar que la IA, aun no reflejo una capacidad de modelado fiel a la teoría y que se alinee a los estándares de UML en su totalidad, por lo que realmente, estos modelos no son de fiar, al contrario, podrían servir como una guía o como herramienta para consultas e inquietudes, para que alguien con el conocimiento necesario haga un proceso adecuado de modelado.