Universidad de las Fuerzas Armadas - ESPE

Pregrado SII OCT24 – MAR25

Programación Orientada a Objetos – Nrc: 1323

Control de Lectura 2: Creación de Objetos y UML

Nombre de los integrantes:

David Iván Granada Pachacama

Nombre del docente:

Mgtr. Luis Enrique Jaramillo Montaño

Fecha de entrega: 08 de diciembre del 2024

Contenido

Tabla de figuras

Introducción

El estilo y diseño para realizar un programa en lenguaje JAVA es muy importante ya que podemos realizarlo mediante un modelado unificado de objetos (UML) siendo una opción de buenas prácticas. Para ello podemos usar diferentes softwares libres, en este informe definiremos los diferentes tipos de relaciones que existen al momento de crear una clase y objetos. Además, se podrá visualizar el código en el entorno de trabajo (IDE Apache NetBeans)

1. Objetivos

1.1. Objetivos General

- Diseñar 5 objetos diferentes con su correspondiente diagrama UML mediante el uso de la IDE Apache NetBeans para mostrar las relaciones que existe entre las clases y objetos.

1.2. Objetivos Específicos

- Conocer la clasificación de relaciones entre clases y objetos.
- Implementar un plugin para graficar los diagramas UML.
- Crear objetos y UML a partir de diferentes clases.

2. Diagrama de clases

Los diagramas de clases es quizá el diagrama más importante de los diagramas UML, y el más usado en el paradigma de Programación Orientada a Objetos, ya que permite reconocer los atributos y operaciones que permiten abstraer el dominio del problema. Tiene como principal base a las clases, la cual cumple la función de una plantilla que permitirá describir los elementos de un objeto del sistema. (Jiménez de Parga, 2015)

Componentes

Un diagrama de clases cuenta con: el nombre, los atributos, las operaciones o comportamientos o métodos y las relaciones, elementos que se describen brevemente a continuación.

- **Nombre:** sirve de identificador para la clase, cada una de estas debe tener un nombre que no se repita
- **Atributos:** son las propiedades y características que tienen las clases, se los representa con el nombre del objeto, seguido del tipo de dato que es el atributo.
- **Operaciones:** también conocidos como métodos en Programación Orientada a Objetos, son las acciones que pueden ser realizadas por las instancias de esa clase y que representa el comportamiento que tendrá dicha clase, se representan con: el nombre de la operación o método, los parámetros que tendrá y el tipo de retorno.

Relaciones

Hace referencia a como se relacionan las clases de un programa y se clasifican en:

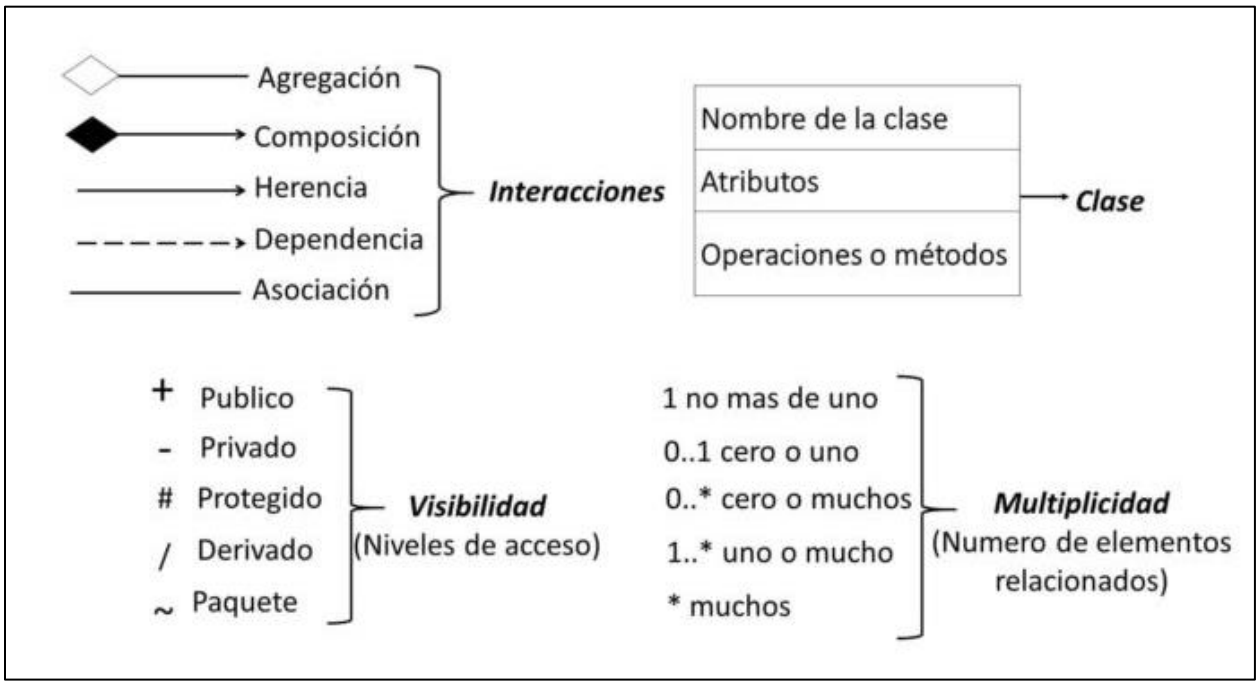
- **Dependencia:** es un tipo de relación débil, en la que una clase utiliza los objetos de otra clase en ciertos contextos, pero estas pueden seguir funcionando de manera independiente.
- **Herencia:** es un tipo de relación en el que una clase hereda atributos y métodos de una clase padre.
- **Agregación:** hace referencia a una relación en el que una clase puede contener objetos de otra clase, pero estos objetos pueden seguir existiendo en caso de que la clase de la que provengan desaparezca, como por ejemplo se considera la clase estudiante, en la que se tiene agregada la clase materia, en el caso de que la clase estudiante desaparezca, la materia aun seguiría existiendo, por lo que la clase materia, seguirá funcionando de manera independiente a estudiante.

- **Composición:** es un tipo de relación opuesta a la agregación, ya que en este caso se habla de una clase que necesita de otra para seguir existiendo y tener validez, como por ejemplo se tiene la clase Auto, en la que esta instanciada la clase Motor, por lo que, si se elimina la clase Auto, la clase motor carecería de sentido.

A continuación, podemos observar en a Figura 1. la representación gráfica de las relaciones.

Figura 1.

Elementos y símbolos en los diagramas de clases UML



- **Modificadores de acceso:** hace referencia al alcance o visibilidad de una clase o su contenido, existen tres tipos que son: publico (+) es visible para cualquier clase, private(-) el contenido es visible solo para la propia clase, protegido(#) es visible para cualquier clase que sea herencia de la clase de contenido protegido. Estos modificadores de acceso se colocan antes de la declaración de atributos o métodos. (Jiménez de Parga, 2015)

A continuación, podemos observar en a Figura 2. la representación gráfica de los modificadores de acceso.

Figura 2.

Modificadores de acceso

Administración configuración del UML en NetBeans

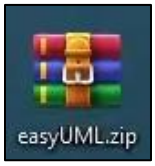
Otro concepto de UML o también llamado “Lenguaje de modelado” según Minguillón (2001) “es un lenguaje basado en diagramas para la especificación, visualización, construcción y documentación de cualquier sistema complejo.” (Minguillón, 2001)

Para poder trabajar los diagramas de clases en el NetBeans, primeramente, se debe instalar un plug-in también conocido como plugin, aplicación que permite extender las funciones de programas sin tener que alterar el código, para lo cual la ingresamos a un navegador y dentro del buscador Google ponemos “descarga de UML para el Apache NetBeans” o algo similar, muestra videos y páginas web que indican como descargarse e instalar dichos plugins. (Beltran, 2015)

Descargado el easyUML, como lo indica la siguiente Figura 3, se procede a descomprimir.

Figura 3.

Archivo .zip del easyUML

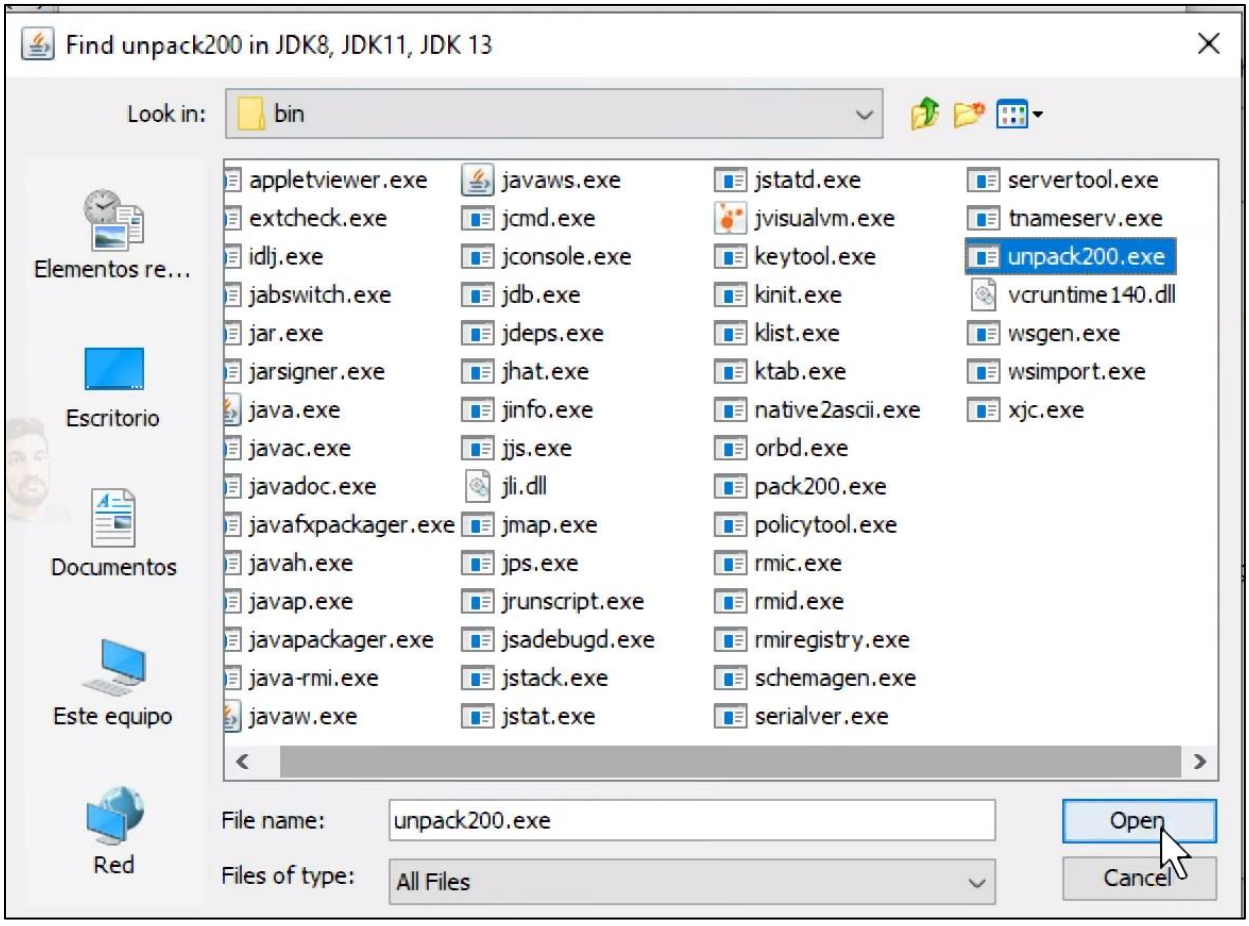


Nota. Fuente: https://drive.google.com/file/d/1_KSygrKOLPtDWT68jGkw2Ii0NmnZDrKN/view

Cabe mencionar que para que no marque error es necesario colocar el archivo unpack200.exe, dentro de la carpeta “bin”, la cual se encuentra dentro de “archivos de programa\java”, de ahí se selecciona la carpeta con el jdk instalado, como lo indica la siguiente Figura 4.

Figura 4.

Archivo unpack200.exe

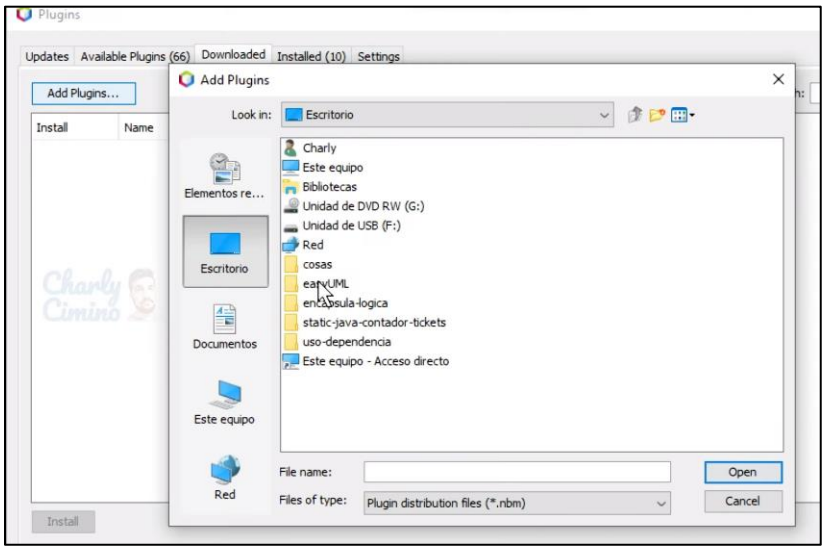


Nota. Colocación del archivo unpack200.exe dentro de la carpeta bin Fuente: https://drive.google.com/file/d/1eS-vra7ZEF74hjweGJz_StjeB12v513q/view

Finalmente, para añadir el plugin de UML al NetBeans, dentro del menú principal, seleccionamos “Tools”, luego la opción “Plugins”, posteriormente la viñeta “Downloaded”, en seguida da clic en “Add Puligins” y se debe seleccionar todos los archivos de la carpeta UML descomprimida, para posteriormente dar clic en el botón “Install” y al final un “Aceptar”.

Figura 5.

Selección de los archivos UML en NetBeans



Nota. Búsqueda de los archivos UML, para la instalación del plugin. Fuente: https://www.youtube.com/watch?v=MiysL0MC2ck

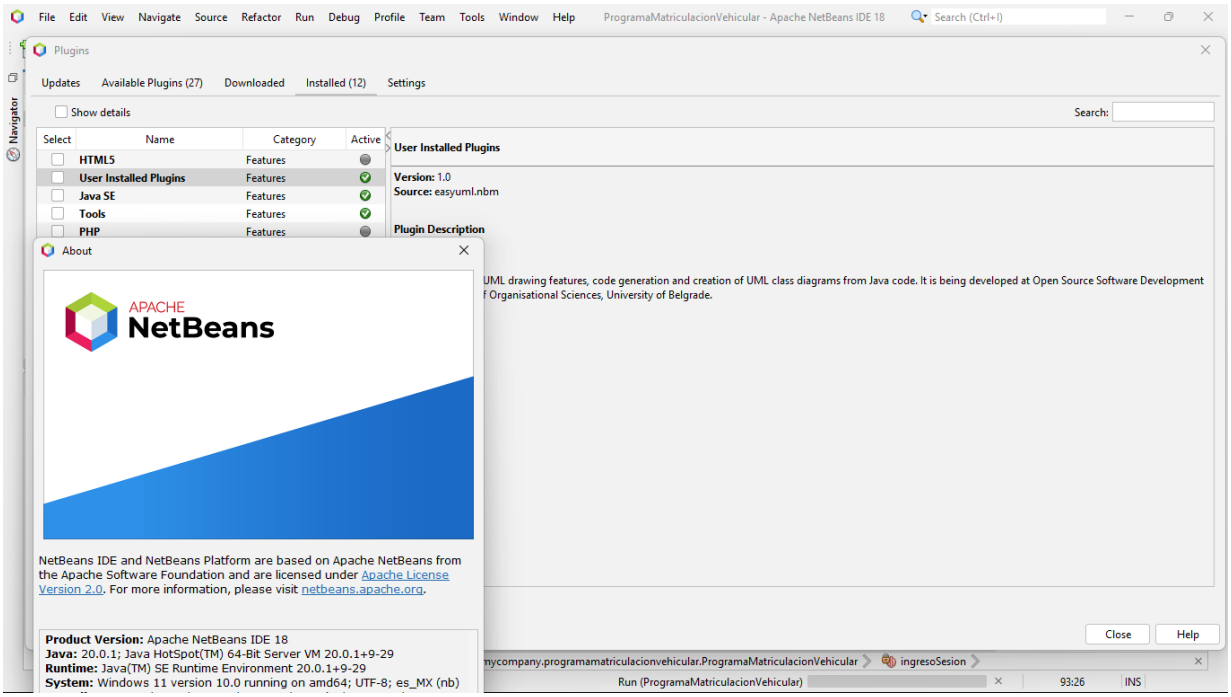
Finalizada la instalación puede seleccionar un proyecto y crear el UML, si desea ver el video de instalación y visualización de un proyecto puede visitar el siguiente link: https://www.youtube.com/watch?v=MiysL0MC2ck

3. Resultados Diagramas de clase UML

A continuación, podemos observar en la Figura 6. el plug-in instalado para crear diagramas UML en NetBeans.

Figura 6.

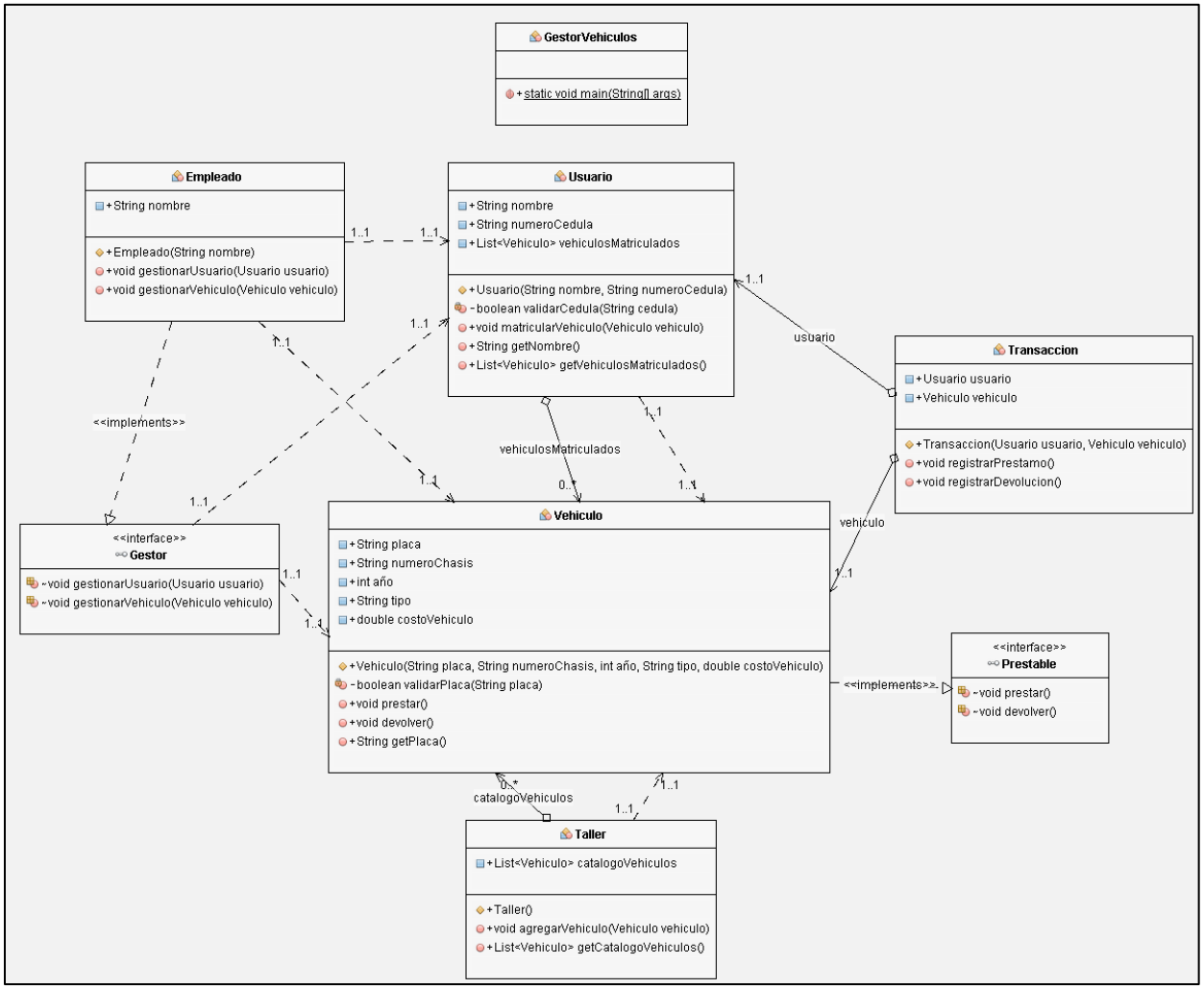
Plugin instalado easyUML instalado en Windows 11



Posteriormente, podemos observar en la Figura 7. la creación de un ejemplo de un diagrama de clases empleando el plug-in easyUML llamado GestorVehiculos.

Figura 7.

Diagrama de clase GestorVehiculos UML



4. Resumen de resolución

- **Vehículo:** Representa los vehículos disponibles en un Taller, con atributos como placa, numero de chasis, año, evaluo, tipo, costo de matrícula, y métodos como prestar() y devolver().
- **Usuario:** Los usuarios pueden tener una lista de vehículos matriculados. Se relaciona con Vehículo mediante composición.
- **Taller:** Administra un catálogo de vehículos y se relaciona con Vehículo por asociación.
- Empleado: Gestiona usuarios y Vehículos, y se relaciona con Usuario y Vehículo por asociación.
- **Transacción:** Registra préstamos y devoluciones. Se relaciona con Usuario y Vehículo por agregación.

Análisis de las relaciones

- **Composición:** Usuario tiene una lista de Vehiculo. Si se elimina un Usuario, se eliminan sus vehículos matriculados.
- **Agregación:** Transaccion tiene referencias a Usuario y Vehiculo. Las transacciones pueden existir sin eliminar los usuarios o vehículos.
- **Asociación:** Taller tiene una lista de Vehiculo. Los vehículos pueden existir independientemente del taller.

5. Código en JAVA Apache NetBeans

Interfaces

Primero, definamos algunas interfaces que representen las acciones que pueden realizar las clases como se puede observar en la Figura 9 y 10.

Figura 8.

Creación de la Interfaz Prestable

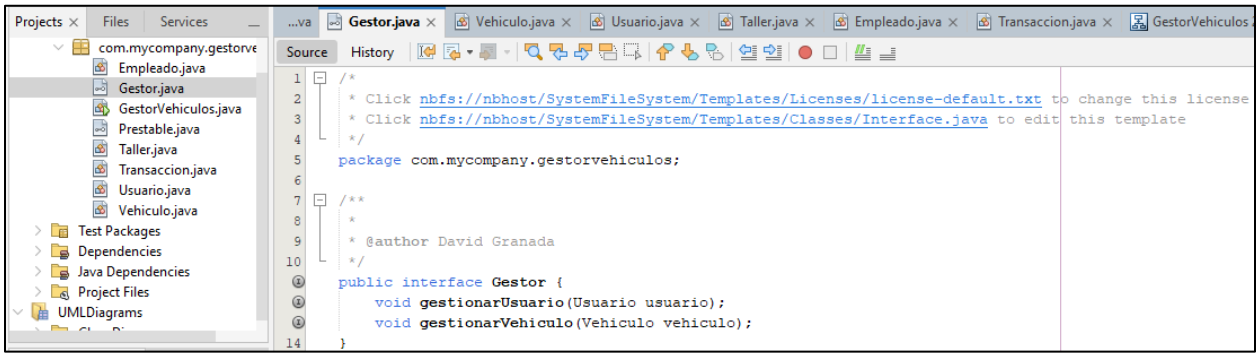
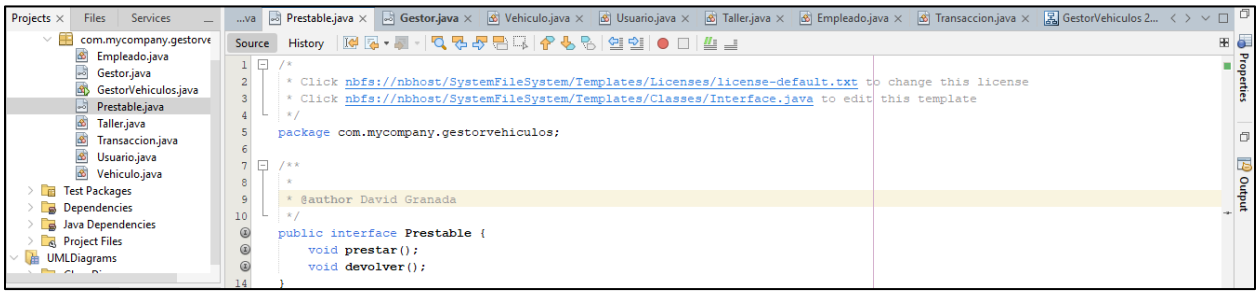


Figura 9.

Creación de la Interfaz Gestor



Clases con Implementación de Interfaces

Ahora, adaptaremos las clases para que implementen estas interfaces como se puede observar en las Figuras 10 a la 16.

Figura 10.

Creación Clase Vehiculo

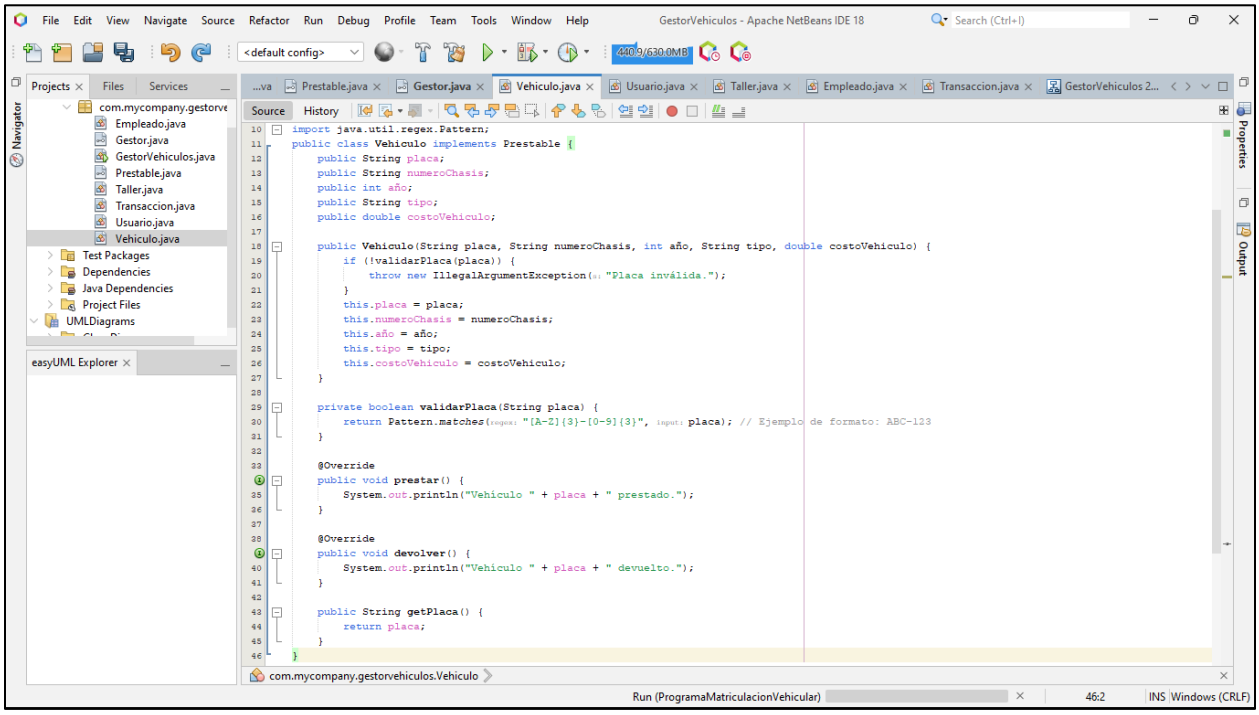
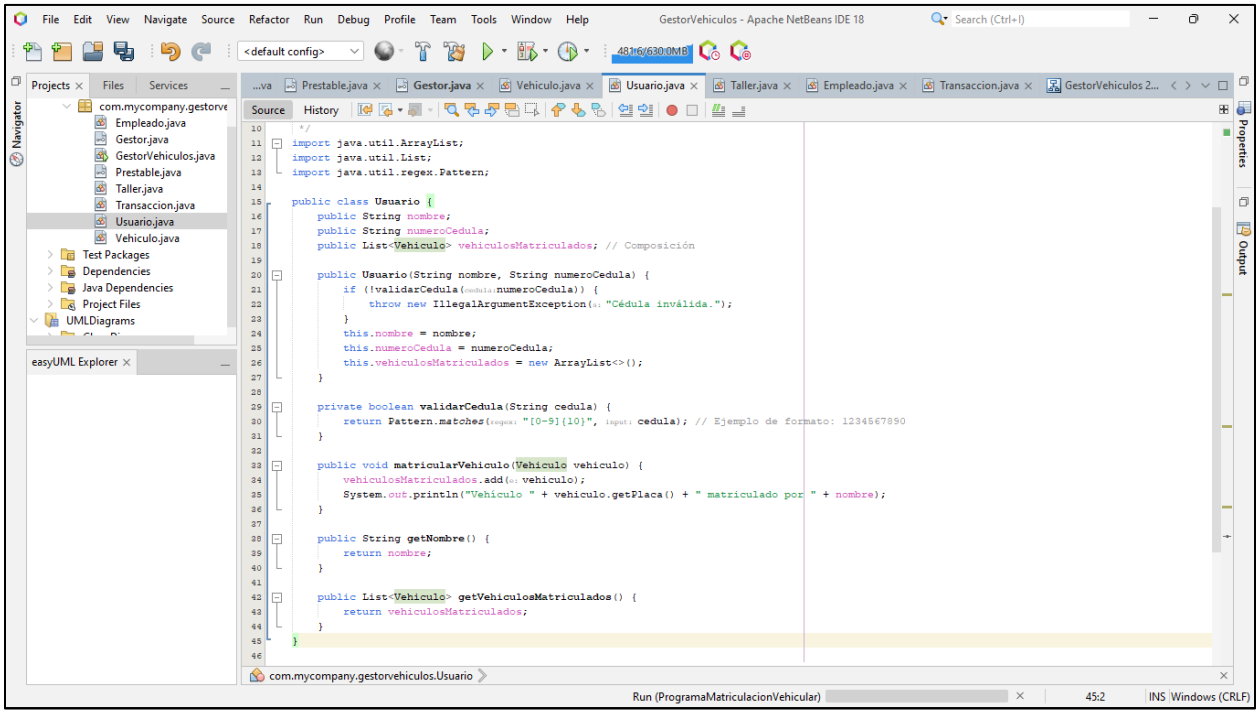


Figura 11.

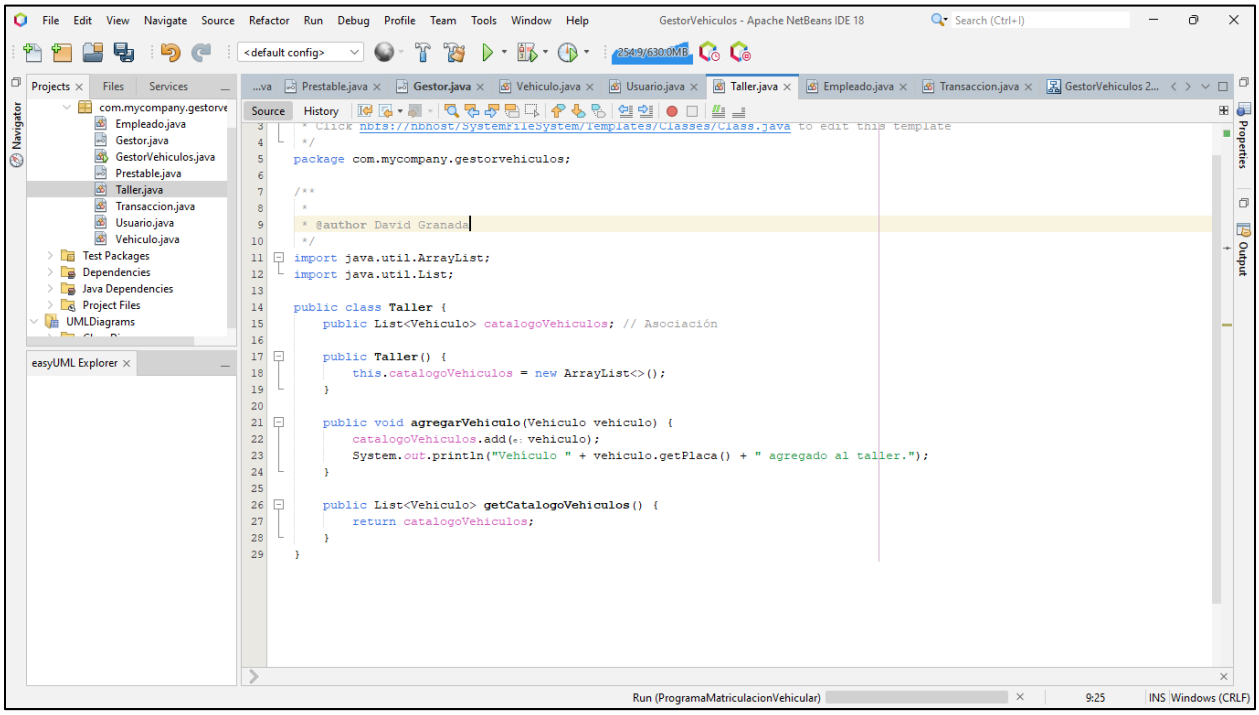
Creación Clase Usuario (Composición)



```java
*/
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Pattern;

public class Usuario {
    public String nombre;
    public String numeroCedula;
    public List<Vehiculo> vehiculosMatriculados; // Composición

    public Usuario(String nombre, String numeroCedula) {
        if (!validarCedula(numeroCedula)) {
            throw new IllegalArgumentException("Cédula inválida.");
        }
        this.nombre = nombre;
        this.numeroCedula = numeroCedula;
        this.vehiculosMatriculados = new ArrayList<>();
    }

    private boolean validarCedula(String cedula) {
        return Pattern.matches("[0-9]{10}", cedula); // Ejemplo de formato: 1234567890
    }

    public void matricularVehiculo(Vehiculo vehiculo) {
        vehiculosMatriculados.add(vehiculo);
        System.out.println("Vehículo " + vehiculo.getPlaca() + " matriculado por " + nombre);
    }

    public String getNombre() {
        return nombre;
    }

    public List<Vehiculo> getVehiculosMatriculados() {
        return vehiculosMatriculados;
    }
}
```

Figura 12.

Creación Clase Taller (Asociación)



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.gestorvehiculos;

/**
 *
 * @author David Granada
 */
import java.util.ArrayList;
import java.util.List;

public class Taller {
    public List<Vehiculo> catalogoVehiculos; // Asociación

    public Taller() {
        this.catalogoVehiculos = new ArrayList<>();
    }

    public void agregarVehiculo(Vehiculo vehiculo) {
        catalogoVehiculos.add(vehiculo);
        System.out.println("Vehículo " + vehiculo.getPlaca() + " agregado al taller.");
    }

    public List<Vehiculo> getCatalogoVehiculos() {
        return catalogoVehiculos;
    }
}
```

Figura 13.

Creación Clase Empleado (Asociación)

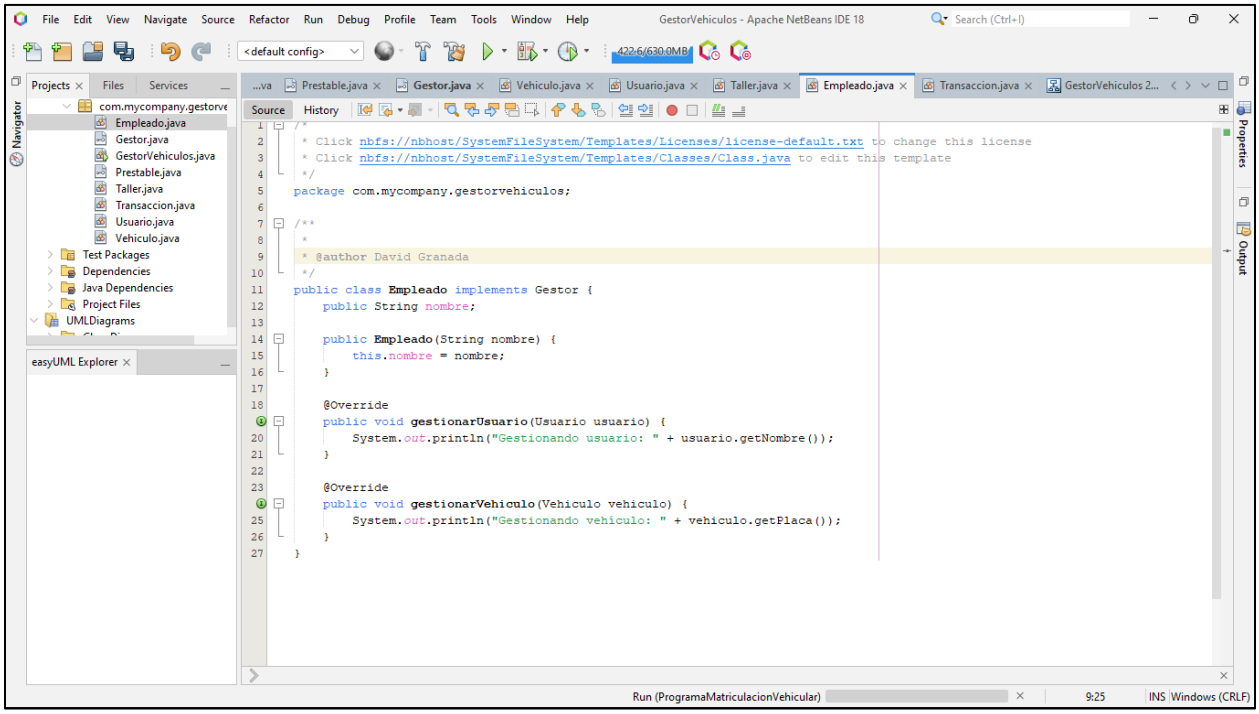


Figura 14.

Clase Transaccion (Agregación)

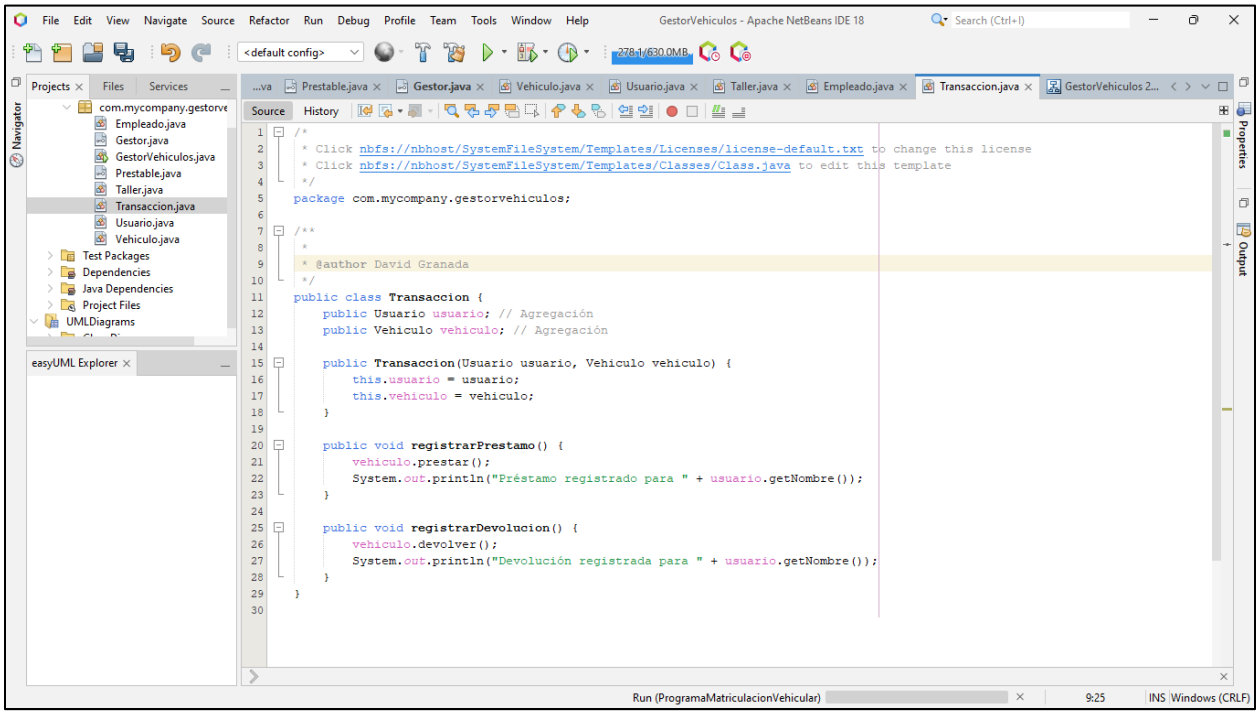


Figura 15.

Creación Clase Principal GestorVehicular

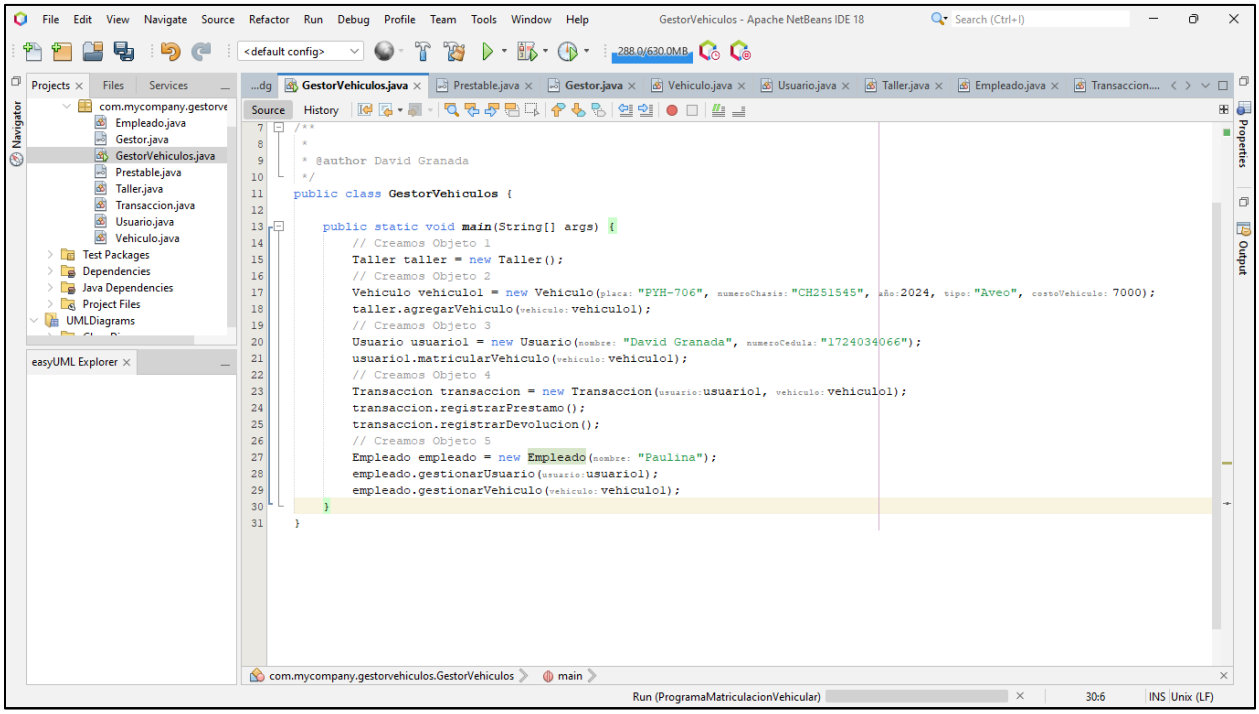
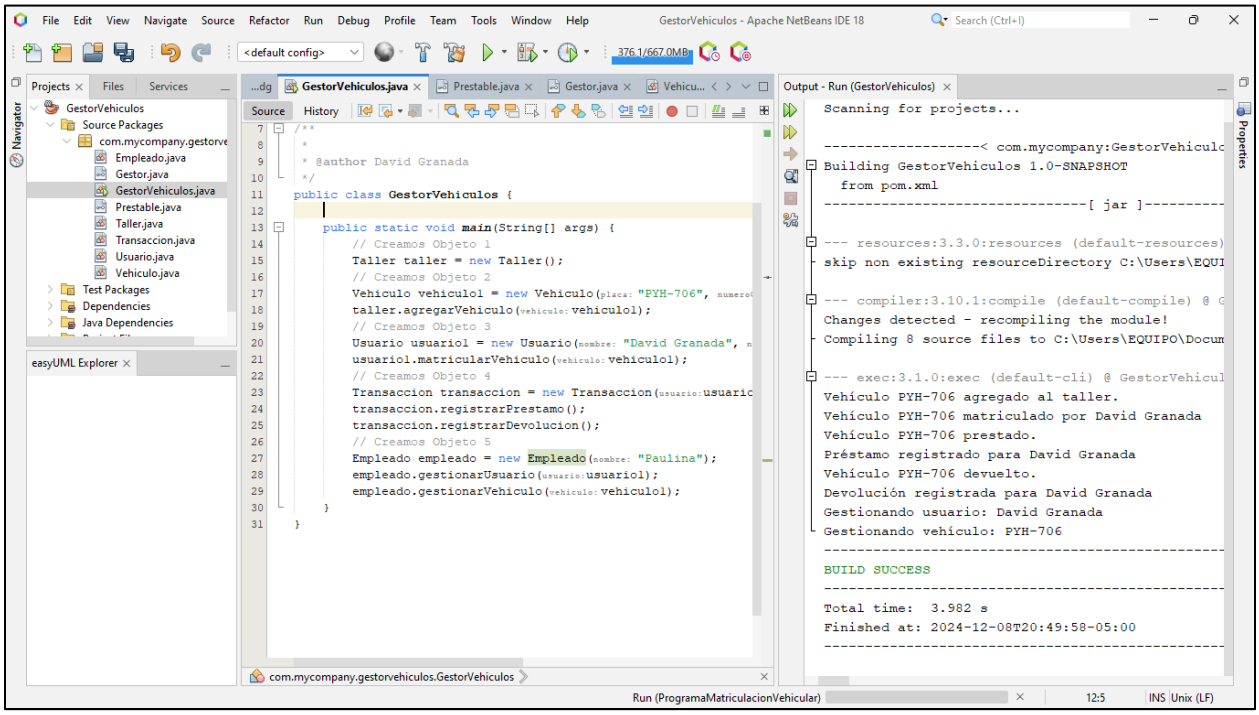


Figura 16.

Salida del programa por consola



6. Conclusiones

- Para crear las relaciones de clases y objetos se debe considerar la multiplicidad que existe entre los mismos.
- Los Diagramas UML son importantes y se debe seguir los pasos para configurar el plug-in en NetBeans.
- Una vez creado el código se debe generar el UML a partir del mismo.

7. Recomendaciones

- Recordar los conceptos de relaciones entre clases.
- Al instalar plug-in easyUML se debe tener instalado JRE 1.8
- Un objeto de un clase puede ser instanciado si está bien estructurada y modelada.

8. Referencias Bibliográficas

Jimenez de Parga, C. (8 de enero de 2015). UML Aplicaciones en java y C++. Madrid., RA-MA. https://bibliotecadigital.utn.edu.ec/download/files/original/0f5b0bcd0e28df16a5ac44ac5cbee29438f2ae46.pdf

Minguillón, J. (2001). Introducción al lenguaje de modelado unificado (UML). Universitat Oberta de Catalunya. https://openaccess.uoc.edu/handle/10609/9121

Beltran, J. (19 de junio de 2015). Construcción de diagramas UML con herramienta día. Modelando el sistema con UML. https://mcjabe.blogspot.com/2015/06/construccion-de-diagramas-uml-con.html

9. Apéndice

A continuación, adjunto link GitHub de mi repositorio donde se encuentra la actividad Con los archivos completos: https://github.com/digranada/202451_1323.git