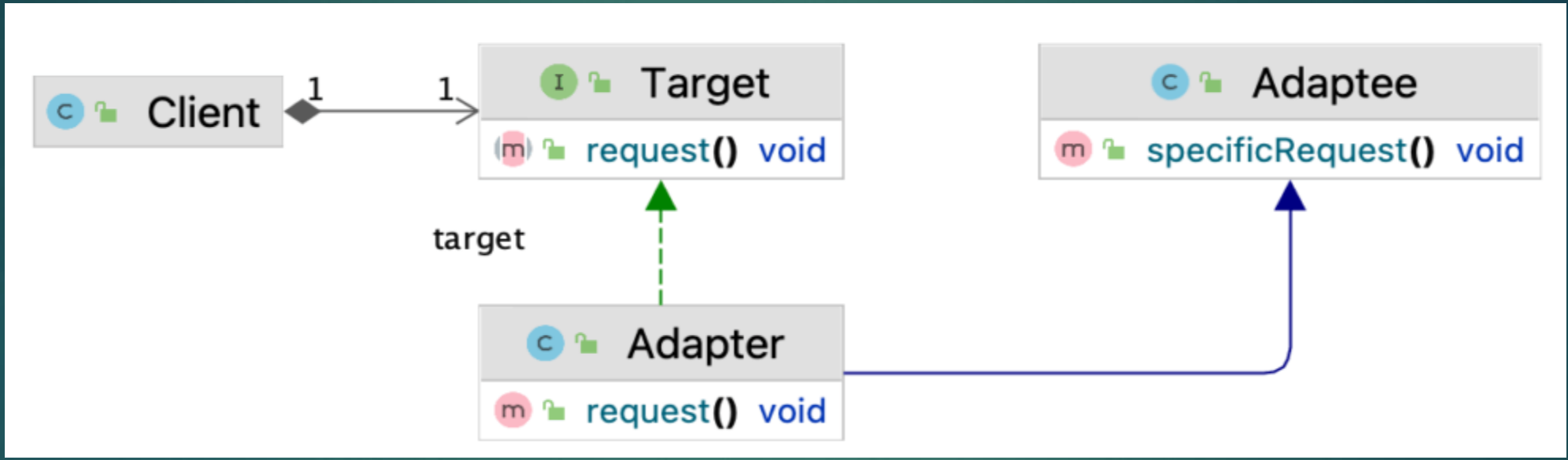
Trabajo – Patrones de diseño

INTEGRANTES:

- Augusto Coronati
- Lucas Romero
- Tomas Goicoechea
- Lautaro Casas
- Alan Mangano
- Patricio Julia
- Franco Vallejo
- Facundo Angel
- Christian Rojas
- Ghiano Gonzalo
- Rodrigo Ramato

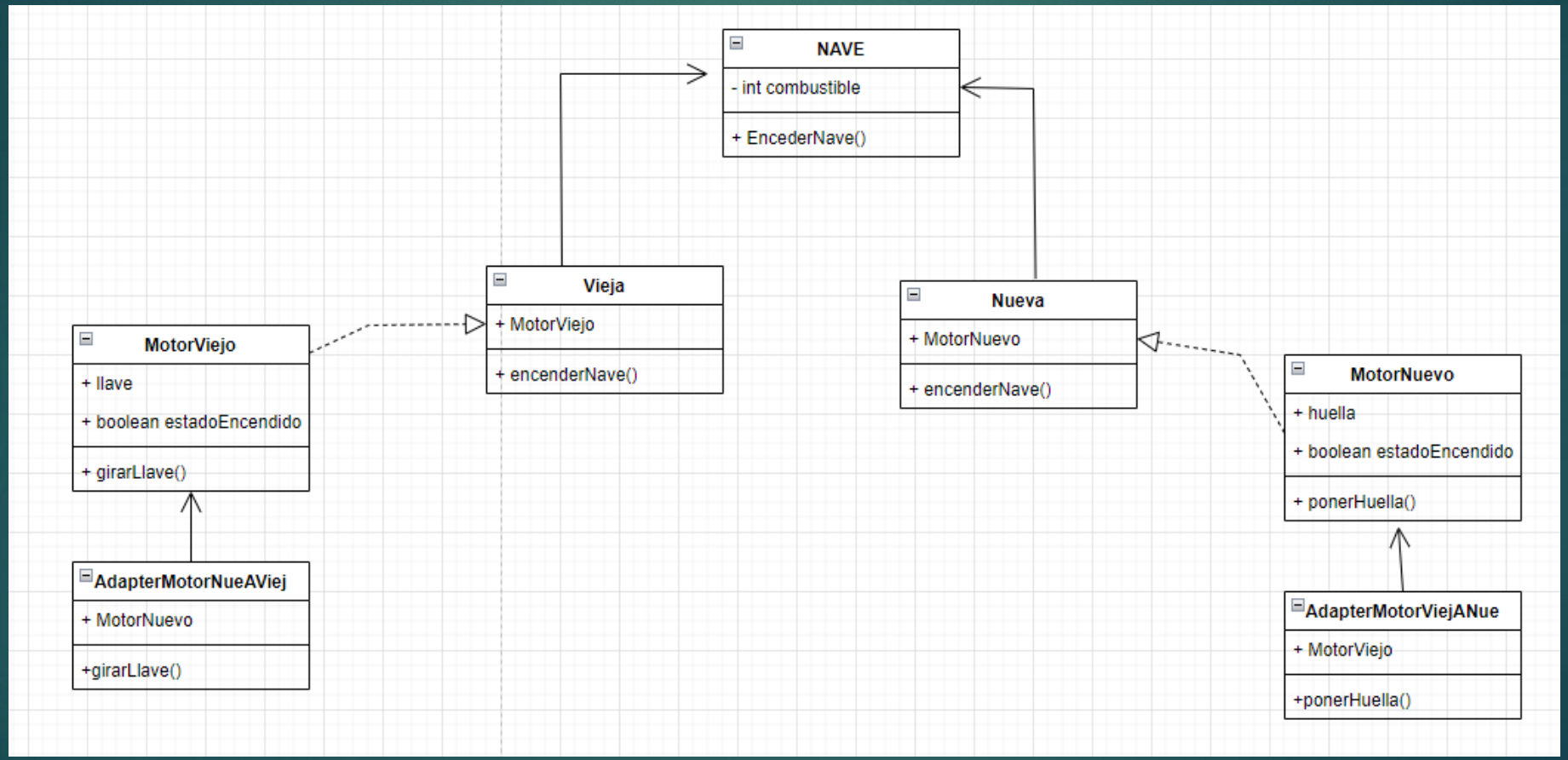
Patrón Elegido: ADAPTER

Diagrama de clase Genérico:



Intención: Compatibilizar dos o mas clases que no pueden interactuar directamente.

Diagrama de clases en contexto:



```java
public class Main {

    public static void main(String[] args) {
        MotorNuevo x8New = new MotorNuevo();
        MotorViejo Old = new MotorViejo();
        AdaptadorViejoANuevo adapterA = new AdaptadorViejoANuevo(Old);

        Nave miAltaNave = new NaveNueva(x8New);
        Nave miAltaNaveDos = new NaveNueva(adapterA);

        miAltaNave.encenderNave();
        miAltaNaveDos.encenderNave();


        AdaptadorNuevoAViejo adapterB = new AdaptadorNuevoAViejo(x8New);
        Nave naveviejita = new NaveVieja(Old);
        Nave naveviejitados = new NaveVieja(adapterB);
S
        System.out.println("\n\n\nLA NAVE VIEJA");
        naveviejita.encenderNave();
        naveviejitados.encenderNave();


    }
```

```java
package clase_22_05;

public class NaveVieja extends Nave{
    private MotorViejo motor;



    public NaveVieja(MotorViejo motor) {
        super();
        this.motor = motor;
    }

    @Override
    public boolean encenderNave() {
        motor.girarLLave();
        return true;
    }
```

```java
package clase_22_05;

public class MotorViejo {
    private boolean estado = false;

    public void girarLLave(){
        System.out.println("GIRANDO LLAVE");
        setEstado(true);
    }
```

```java
package clase_22_05;

public class AdaptadorNuevoAViejo extends MotorViejo{
    protected MotorNuevo motor;

    public AdaptadorNuevoAViejo(MotorNuevo motor) {
        super();
        this.motor = motor;
    }

    @Override
    public void girarLLave(){
        System.out.println("Adaptando... poniendo huella");
        motor.ponerHuella();
    }

}
```

Ventajas y desventajas

Ventajas:

- Evita desechar clases enteras ya que permiten que estas esten conformadas o se comuniquen con objetos nuevos o distintos a los previstos en el momento de escribir la clase en un principio.
- Da escalabilidad al proyecto ya que deja la posibilidad de que la clase vaya extendiendo su funcionalidad con el paso del tiempo, pudiendo comunicar con nuevos objetos en el proyecto.

Desventajas:

- Suele adherir mayor complejidad a la clase, con lo cual hace mas complicado el diagrama UML.
- Es necesario hacer un nuevo adaptador para cada tipo de objeto nuevo que aparezca, con lo cual provoca que se incremente el número de clases constantemente.

Referencias

- https://www.baeldung.com/java-adapter-pattern
- https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.baeldung.com%2Fjava-adapter-pattern&psig=AOvVaw1LsLpN2eoZk0-LyItkuGHX&ust=1684846687864000&source=images&cd=vfe&ved=0CA4QjRxqFwoTCKjhsZr9iP8CFQAAAAAdAAAAABAD+