

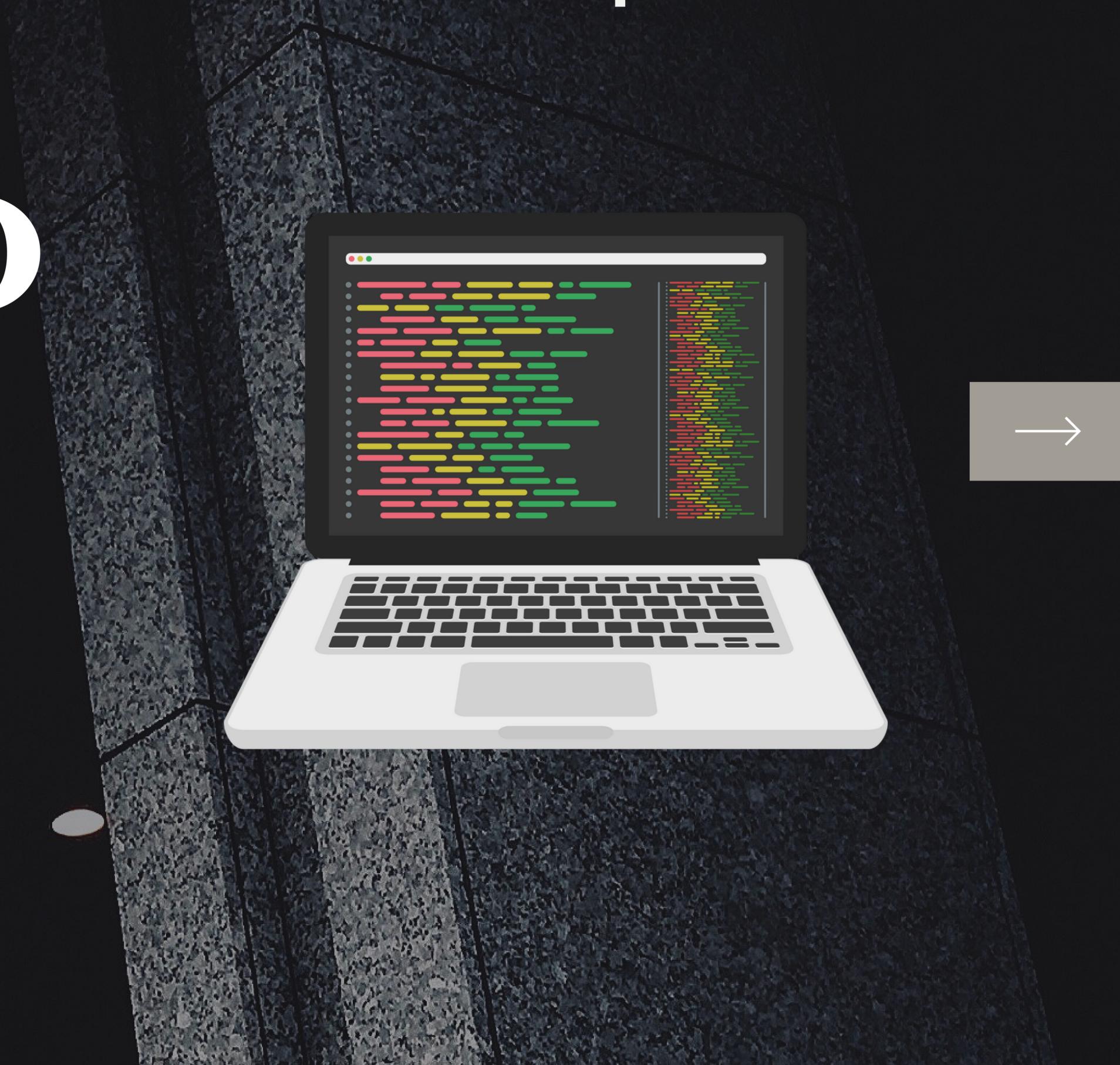
T/R

Grupo 4

Proyecto

Integrantes:

Ordoñez Eduardo
Kelvin Quezada
Riera Michael
Sanchez Lander



LOS PATRONES DE DISEÑO

los patrones de diseños son soluciones general, reutilizables y se aplica en diferentes tipos de problemas del diseño del Software ya que se trata de identifica un problema en el sistema y proporciona soluciones ya que los desarrolladores se enfrentan durante un periodo de tiempo a las pruebas y errores.



Patrones Creacionales

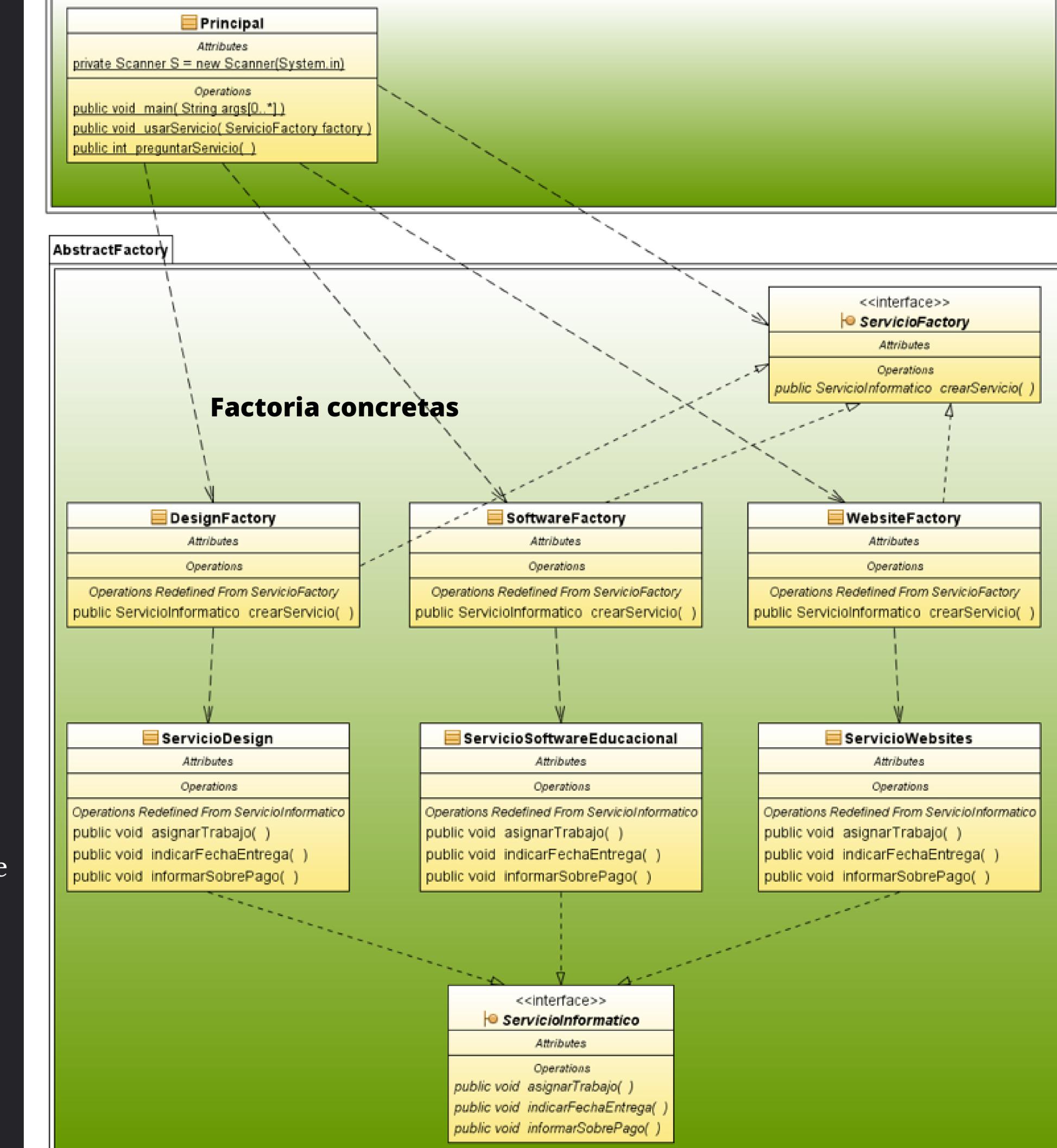
Los patrones de compilación ofrecen una variedad de mecanismos de creación de objetos que aumentan la flexibilidad y permiten la reutilización adecuada de la situación del código preexistente. Como resultado, el programa tiene más libertad para elegir qué objetos crear para un caso de uso particular.

Diagrama UML Abstract Factory

Se utiliza interfaz para crear conjuntos o familias de objetos relacionados sin especificar el nombre de la clase.

Aquí se crean objetos de clase distinta pero que tienen métodos en común aquí tenemos clases de fábrica que permite la instanciación de los objetos.

Existe una Factoría abstracta en esta clase servicioFactory de esta interfaz diriba tres factoría concretas que se encarga de instanciar correctamente objetos de las clases productos la fábrica designFactory va a crear objetos de la clase servicioDesign la ultima nos va a instanciar los objetos de la clase webFactory



```

package AbstractFactory;
//factoria abstracto
public interface ServicioFactory {
    // define un método público llamado crearServicio que no acepta
    //ningún argumento y devuelve un objeto del tipo "ServicioInformatico".
    public ServicioInformatico crearServicio();
}

```

Factoría Concretas de la interface ServicioFactory

Existe una Factoría abstracta en esta clase servicioFactory de esta interfaz diriba tres factoría concretas que se encarga de instanciar correctamente objetos de las clases productos

```

package AbstractFactory;
//tendremos una clase DesignFactory que sera implementada
//a la interface ServicioFactory
public class DesignFactory implements ServicioFactory {
    //se crea un metodo para sobrescribiendo de la interfaz
    //el método crearServicio() está definido en una clase
    //que implementa la interfaz ServicioInformatico
    @Override
    public ServicioInformatico crearServicio() {
        return new ServicioTrabajo();
    }
}

```

```

public class SoftwareFactory implements ServicioFactory {
    //se crea un metodo para sobrescribiendo de la interfaz
    //el método crearServicio() está definido en una clase
    //que implementa la interfaz ServicioInformatico
    @Override
    public ServicioInformatico crearServicio() {
        return new ServicioSoftwareEducacional();
    }
}

```

```

public class WebsiteFactory implements ServicioFactory {
    //sobrescribiendo un método de la interfaz
    @Override
    //el método crearServicio() está definido en una clase
    //que implementa la interfaz ServicioInformatico
    public ServicioInformatico crearServicio() {
        return new ServicioWebsites();
    }
}

```

Factoria Concretas de la interface ServicioInformatico

```
public class ServicioDesign implements ServicioInformatico {  
    //Creamos tres metodos que estan sobrescribiendo es nuestra clase interfaz de ServicioInformatico  
    @Override  
    public void asignarTrabajo() {  
        System.out.println("El trabajado ha sido asignado a disenadores graficos disponibles.");  
    }  
    @Override  
    public void indicarFechaEntrega() {  
        System.out.println("Ellos han determinado terminar el trabajo como maximo para el dia 09/08/2023.");  
    }  
    @Override  
    public void informarSobrePago() {  
        System.out.println("Debe realizar el pago en efectivo al momento de recoger el logo completamente terminado.");  
    }  
}
```

```
public class ServicioWebsites implements ServicioInformatico {  
    //Creamos tres metodos que estan sobrescribiendo es nuestra clase interfaz de ServicioInformatico  
    @Override  
    public void asignarTrabajo() {  
        System.out.println("El personal encargado del desarrollo de sitios web ha aceptado el trabajo.");  
    }  
  
    @Override  
    public void indicarFechaEntrega() {  
        System.out.println("El sitio web con Responsabilida Designara terminado el dia 26/11/2023.");  
    }  
  
    @Override  
    public void informarSobrePago() {  
        System.out.println("El monto a pagar no incluye el pago por dominio y hosting.");  
    }  
}
```

```
public class ServicioWebsites implements ServicioInformatico {  
    //creamos tres metodos que estan sobrescribiendo es nuestra clase interfaz de ServicioInformatico  
    @Override  
    public void asignarTrabajo() {  
        System.out.println("El personal encargado del desarrollo de sitios web ha aceptado el trabajo.");  
    }  
    @Override  
    public void indicarFechaEntrega() {  
        System.out.println("El sitio web con Responsabilidad Designara terminado el dia 26/11/2023.");  
    }  
    @Override  
    public void informarSobrePago() {  
        System.out.println("El monto a pagar no incluye el pago por dominio y hosting.");  
    }  
}
```

```
package AbstractFactory;  
//los metodos aparecen a los productos abstractos  
public interface ServicioInformatico {  
    public void asignarTrabajo();  
    public void indicarFechaEntrega();  
    public void informarSobrePago();  
}
```

```
public class Principal {
public static void main(String[] args) {
    Scanner leer = new Scanner(source: System.in);
    int opcion;
    do {
        System.out.print(
            "MENU DE OPCIONES\n"
            + "---- -- ----->\n"
            + "1. Solicitar servicio de diseño gráfico.\n"
            + "2. Solicitar desarrollo de software educacional.\n"
            + "3. Solicitar creación de sitios web.\n"
            + "4. Cerrar programa.\n"
            + "Seleccione opción: "
        );
        opcion=leer.nextInt();
        switch (opcion) {
            case 1:
                usarServicio(new DesignFactory());
                break;
            case 2:
                usarServicio(new SoftwareFactory());
                break;
            case 3:
                usarServicio(new WebsiteFactory());
                break;
            case 4:
                System.out.println(x: "A finalizado el programa ");
                break;
            default:
                System.out.println(x: "A opción es incorrecta ");
        }
    } while (opcion!=4);
}
```

```
/* creamos un método pública y estática que recibe un objeto de tipo
ServicioFactory como argumento. La función se encarga de utilizar el
factory para obtener un objeto ServicioInformatico y luego realizar
algunas acciones relacionadas con ese servicio.
*/
public static void usarServicio(ServicioFactory factory) {
    // el ServicioInformatico es una clase interface que resive información
    ServicioInformatico servicio = factory.crearServicio();
    servicio.asignarTrabajo();
    servicio.indicarFechaEntrega();
    servicio.informarSobrePago();
}
```

run:

MENU DE OPCIONES

---- -- ----->

1. Solicitar servicio de diseño gráfico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creación de sitios web.
4. Cerrar programa.

Seleccione opción: 1

El trabajado ha sido asignado a diseñadores gráficos disponibles.

Ellos han determinado terminar el trabajo como máximo para el día 09/08/2023.

Debe realizar el pago en efectivo al momento de recoger el logo completamente terminado.

MENU DE OPCIONES

---- -- ----->

1. Solicitar servicio de diseño gráfico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creación de sitios web.
4. Cerrar programa.

Seleccione opción: 2

Nuestros programadores han sido informados del programa que deben realizar.

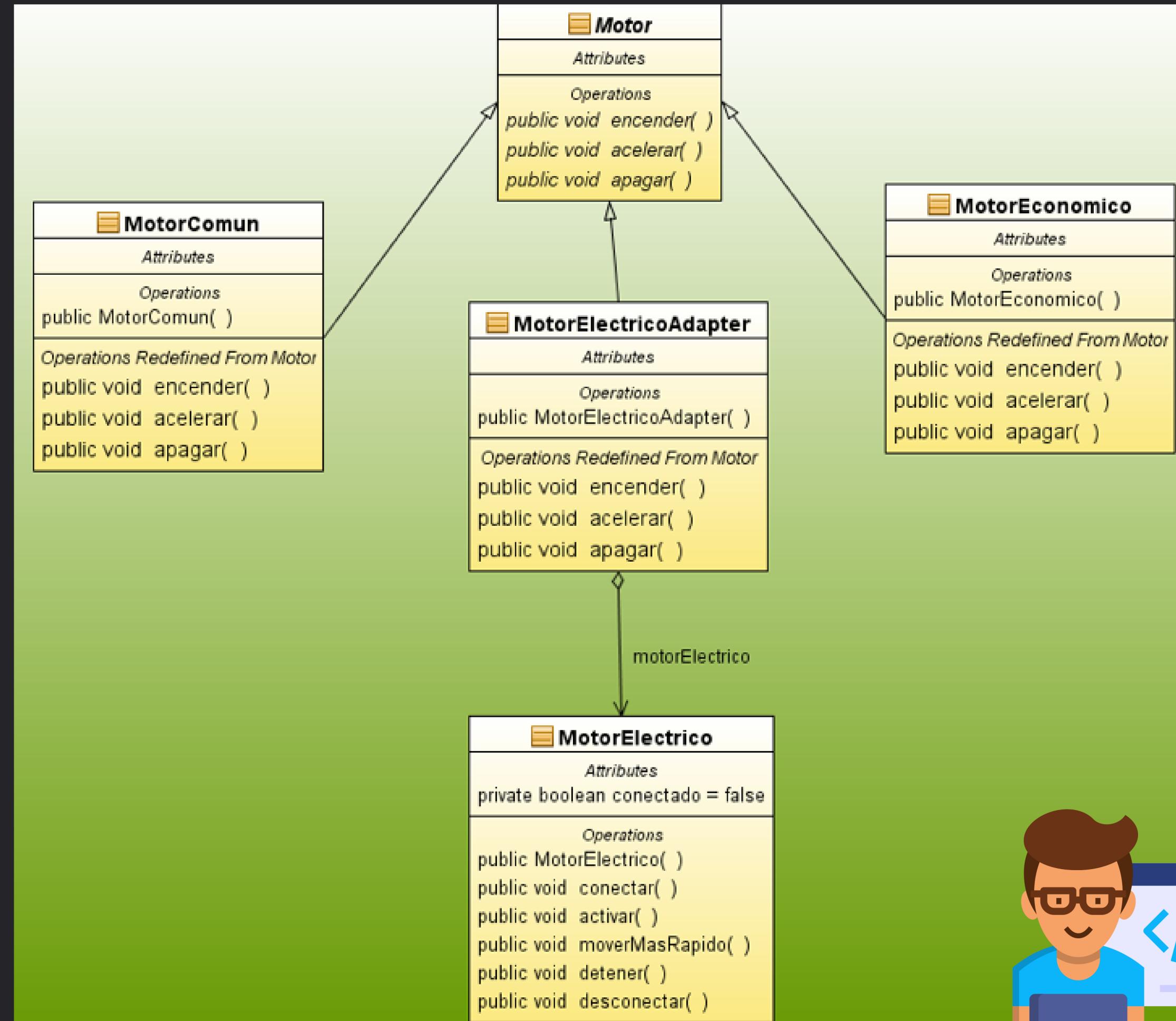
Se ha fijado como fecha de entrega el día 01/10/2023.

El monto a pagar será proporcional a la cantidad de estudiantes que harán uso del software.

Diagrama UML Patrones estructurales

Representa las clases del sistema, sus atributos y métodos, y las relaciones entre ellas, como asociaciones, herencia y dependencias.

Según (Soto, 2021) el objetivo de los patrones estructurales es facilitar el ensamblaje de elementos de clases estructurales más grandes manteniendo la flexibilidad y la eficiencia.



Ejemplo 2

```
Start Page × Motor.java × MotorComun.java × MotorEconomico.java
Source History Source History Projects Files
1 package backend;
2
3 public abstract class Motor {
4     abstract public void encender();
5     abstract public void acelerar();
6     abstract public void apagar();
7 }
8
```

```
Start Page × Motor.java × MotorComun.java × MotorEconomico.java × MotorElectrónico.java
Source History Source History Projects Files
1 package backend;
2
3 public class MotorComun extends Motor {
4
5     public MotorComun() {
6         super();
7         System.out.println("Creando motor común...");
8     }
9
10    @Override
11    public void encender() {
12        System.out.println("Encendiendo motor común.");
13    }
14
15    @Override
16    public void acelerar() {
17        System.out.println("Acelerando motor común.");
18    }
19
20    @Override
21    public void apagar() {
22        System.out.println("Apagando motor común.");
23    }
24 }
```

```
Start Page × Motor.java × MotorComun.java × MotorEconomico.java × MotorElectrónico.java
Source History Source History Projects Files
1 package backend;
2
3 public class MotorEconomico extends Motor {
4
5     public MotorEconomico() {
6         super();
7         System.out.println("Creando motor económico...");
8     }
9
10    @Override
11    public void encender() {
12        System.out.println("Encendiendo motor económico.");
13    }
14
15    @Override
16    public void acelerar() {
17        System.out.println("Acelerando el motor económico.");
18    }
19
20    @Override
21    public void apagar() {
22        System.out.println("Apagando motor económico.");
23    }
24 }
25 }
```

The image shows a Java development environment with two code editors open side-by-side.

Left Editor (MotorElectrico.java):

```
package backend;

public class MotorElectrico {
    private boolean conectado = false;

    public MotorElectrico() {
        System.out.println("Creando motor eléctrico...");
        this.conectado = false;
    }

    public void conectar() {
        System.out.println("Conectando motor eléctrico.");
        this.conectado = true;
    }

    public void activar() {
        if (!this.conectado) {
            System.out.println("No se puede activar porque no está conectado el motor eléctrico.");
        } else {
            System.out.println("Está conectado, activando motor eléctrico.");
        }
    }

    public void moverMasRapido() {
        if (!this.conectado) {
            System.out.println("No se puede mover rápido el motor eléctrico porque no está conectado.");
        } else {
            System.out.println("Moviendo más rápido, aumentando voltaje del motor eléctrico.");
        }
    }

    public void detener() {
        if (!this.conectado) {
            System.out.println("No se puede detener motor eléctrico porque no está conectado.");
        } else {
            System.out.println("Deteniendo motor eléctrico.");
        }
    }

    public void desconectar() {
        System.out.println("Desconectando motor eléctrico.");
        this.conectado = false;
    }
}
```

Right Editor (MotorElectricoAdapter.java):

```
package backend;

public class MotorElectricoAdapter extends Motor{
    private MotorElectrico motorElectrico;

    public MotorElectricoAdapter(){
        super();
        System.out.println("Creando motor eléctrico adapter...");
        this.motorElectrico = new MotorElectrico();
    }

    @Override
    public void encender() {
        System.out.println("Encendiendo motor eléctrico adapter.");
        this.motorElectrico.conectar();
        this.motorElectrico.activar();
    }

    @Override
    public void acelerar() {
        System.out.println("Acelerando motor eléctrico adapter.");
        this.motorElectrico.moverMasRapido();
    }

    @Override
    public void apagar() {
        System.out.println("Apagando motor eléctrico adapter.");
        this.motorElectrico.detener();
        this.motorElectrico.desconectar();
    }
}
```

Start Page X | Motor.java X | MotorComun.java X | MotorEconomico.java X | MotorE

Source History | Back Forward | Refresh | Save | Run | Stop | Breakpoints | Run Configuration | Files | Projects

```
1 package frontend;
2
3 import backend.Motor;
4 import backend.MotorComun;
5 import backend.MotorEconomico;
6 import backend.MotorElectricoAdapter;
7 import java.util.Scanner;
8
9 public class Aplicacion {
10
11     private static Scanner S = new Scanner( source: System.in );
12     private static Motor motor;
13
14     public static void main(String[] args) {
15         System.out.println( " " );
16         int opcion;
17         do{
18             opcion = preguntarOpcion();
19             switch(opcion) {
20                 case 1:
21                     motor = new MotorComun();
22                     usurMotor();
23                     break;
24                 case 2:
25                     motor = new MotorEconomico();
26                     usurMotor();
27                     break;
28                 case 3:
29                     motor = new MotorElectricoAdapter();
30                     usurMotor();
31                     break;
32                 case 4:
33                     System.out.println( " ;Cerrando programa!" );
34                     break;
35                 default:
36                     System.out.println( " La opción ingresada NO es válida." );
37             }
38             System.out.print( " \n\n" );
39         }while(opcion!=4);
40     }
}
```

```
42
43     private static int preguntarOpcion() {
44         System.out.print(
45             "MENÚ DE OPCIONES\n"
46             + "---- - ----\n"
47             + "1. Encender motor comun.\n"
48             + "2. Encender motor economico.\n"
49             + "3. Encender motor electrico.\n"
50             + "4. Salir.\n"
51             +"Seleccione opcion: "
52         );
53         return Integer.parseInt( S.nextLine() );
54     }
55
56     private static void usurMotor() {
57         motor.encender();
58         motor.acelerar();
59         motor.apagar();
60     }
61
```

```
MENU DE OPCIONES
-----
1. Encender motor comun.
2. Encender motor economico.
3. Encender motor electrico.
4. Salir.
Seleccione opcion: 1
Creando motor comun...
Encendiendo motor comun.
Acelerando motor comun.
Apagando motor comun.
```

```
MENU DE OPCIONES
-----
1. Encender motor comun.
2. Encender motor economico.
3. Encender motor electrico.
4. Salir.
Seleccione opcion: 3
Creando motor electrico adapter...
Creando motor electrico...
Encendiendo motor electrico adapter.
Conectando motor electrico.
Esta conectado, activando motor electrico.
Acelerando motor electrico adapter.
Moviendo mas rapido, aumentando voltaje del motor electrico.
Apagando motor electrico adapter.
Deteniendo motor electrico.
Desconectando motor electrico.
```

MENU DE OPCIONES

---- -- ----->

1. Solicitar servicio de diseño gráfico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creación de sitios web.
4. Cerrar programa.

Seleccione opción: 3

El personal encargado del desarrollo de sitios web ha aceptado el trabajo.

El sitio web con Responsabilidad Designará terminado el día 26/11/2023.

El monto a pagar no incluye el pago por dominio y hosting.

Output - EjemploAbstractFactory (run) X



run:



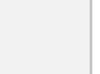
MENU DE OPCIONES
---- -- ----->



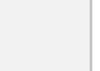
1. Solicitar servicio de diseño gráfico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creación de sitios web.
4. Cerrar programa.



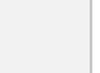
Seleccione opción: 7



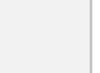
A opción es incorrecta



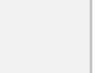
MENU DE OPCIONES
---- -- ----->



1. Solicitar servicio de diseño gráfico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creación de sitios web.
4. Cerrar programa.



Seleccione opción: 4

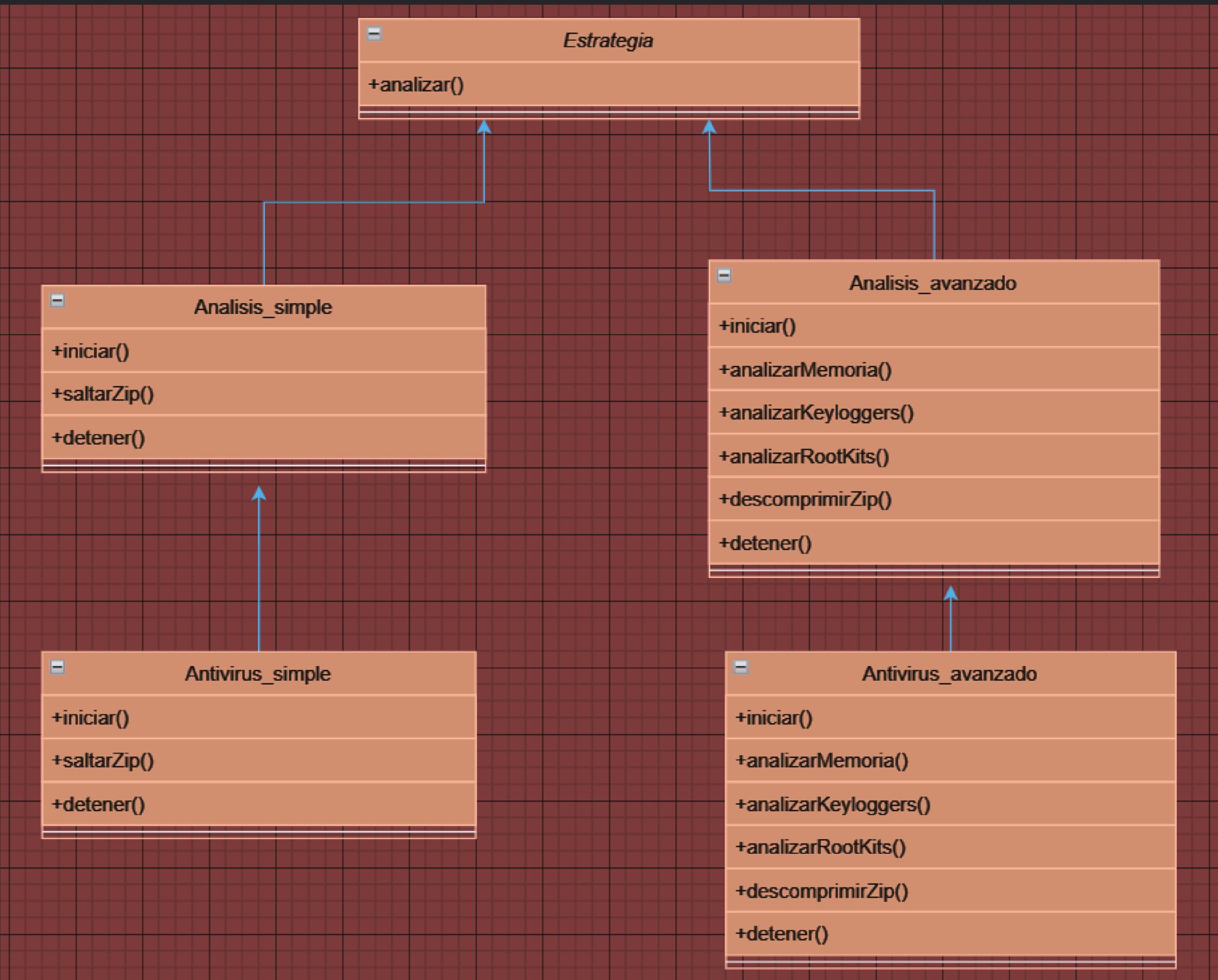


A finalizado el programa

BUILD SUCCESSFUL (total time: 5 seconds)

Patron de Comportamiento

Los patrones de comportamiento buscan resolver la comunicación entre diferentes áreas



Ejemplo 3

```
package packagepoo_programa3;

import Strategy.Contexto;
import Strategy.Analisis_avanzado;
import Strategy.Antivirus_avanzado;
import Strategy.Analisis_simple;
import Strategy.Antivirus_simple;

public class Código_strategy {
    public static void main(String[] args) {
        Contexto contexto = new Contexto(new Antivirus_simple());
        contexto.ejecutar();
    }
}
```

```
package Strategy;

public interface Estrategia {
    void analizar();
}
```

```
package Strategy;

public class Contexto {
    private Estrategia estrategia;
    public Contexto(Estrategia estrategia) {
        this.estrategia = estrategia;
    }
    public void ejecutar() {
        this.estrategia.analizar();
    }
}
```

```
package Strategy;

public abstract class Analisis_simple implements Estrategia {
    public void analizar() {
        iniciar();
        saltarZip();
        detener();
    }

    abstract void iniciar();
    abstract void saltarZip();
    abstract void detener();
}
```

```
package Strategy;

public class Antivirus_simple extends Analisis_simple {
    @Override
    void iniciar() {
        System.out.println("Antivirus simple - Análisis simple iniciando....");
    }
    @Override
    void saltarZip() {
        try {
            System.out.println("Analizando....");
            Thread.sleep(5000);
            System.out.println("No se pudo analizar los archivos con extensión '.zip'.....Realice un escaneo avanzado");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    void detener() {
        System.out.println("Antivirus simple - Análisis simple a finalizado");
    }
}
```

```
package Strategy;

public abstract class Analisis_avanzado implements Estrategia {
    public void analizar() {
        iniciar();
        analizarMemoria();
        analizarKeyloggers();
        analizarRootKits();
        descomprimirZip();
        detener();
    }
    abstract void iniciar();
    abstract void analizarMemoria();
    abstract void analizarKeyloggers();
    abstract void analizarRootKits();
    abstract void descomprimirZip();
    abstract void detener();
}
```

```
package Strategy;

public class Antivirus_avanzado extends Analisis_avanzado {
    @Override
    void iniciar() {
        System.out.println("Antivirus avanzado - Análisis avanzado iniciando....");
    }
    @Override
    void analizarMemoria() {
        try {
            System.out.println("Analizando Memoria RAM....");
            Thread.sleep(7000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    void analizarKeyloggers() {
        try {
            System.out.println("Analizando en busca de KeyLoggers....");
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    void analizarRootKits() {
        try {
            System.out.println("Analizando en busca de RootKits....");
            Thread.sleep(4000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    void descomprimirZip() {
        try {
            System.out.println("Analizando los archivos '.zip'....");
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    void detener() {
        System.out.println("Antivirus avanzado - Análisis avanzado a finalizado");
    }
}
```

```
Antivirus simple - Análisis simple iniciando....  
Analizando....  
No se pudo analizar los archivos con extensión '.zip'....Realice un escaneo avanzado  
Antivirus simple - Análisis simple a finalizado  
-----  
BUILD SUCCESS
```

```
Antivirus avanzado - Análisis avanzado iniciando....  
Analizando Memoria RAM....  
Analizando en busca de KeyLoggers....  
Analizando en busca de RootKits....  
Analizando los archivos '.zip'....  
Antivirus avanzado - Análisis avanzado a finalizado  
-----  
BUILD SUCCESS
```

Conclusiones

- Mejora la calidad del código al proporcionar soluciones probadas para problemas comunes.
- Fomenta la reutilización de soluciones exitosas, acelerando el desarrollo y reduciendo errores.
- Facilita la comunicación entre desarrolladores al establecer un lenguaje común.

Recomendaciones

- Aplica los patrones de manera adecuada según el contexto y requisitos del proyecto.
- Familiarízate con patrones específicos de dominio para abordar problemas particulares.