UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE SEDE SANTO DOMINGO

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO	:	Abril 2023 – Octubre 2023
ASIGNATURA	:	POO
TEMA	:	Patrones de Diseño
NOMBRES	:	Ordoñez Eduardo Kelvin Quezada Riera Michael Sanchez Lander
NIVEL-PARALELO	:	Segundo “A”
DOCENTE	:	Ing. Cevallos Farias Javier Moyota
FECHA DE ENTREGA	:	06/06/2023

Grupo 4

SANTO DOMINGO - ECUADOR

2023

Contenido

1. Introducción

Según (Canelo, 2020) nos dice que 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 plantillas que 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.

En este informe hablaremos sobre los patrones de diseño, son soluciones probadas y comunes para problemas recurrentes en el desarrollo de software. Se dividen en tres categorías: creacionales, estructurales y de comportamiento.

Su objetivo es mejorar la flexibilidad, modularidad y mantenibilidad del código, evitando repeticiones y mejorando la comunicación entre los elementos del sistema.

2. Objetivos

2.1.1. Objetivos Generales

- Comprender los conceptos y principios fundamentales de los patrones de diseño en programación.

2.1.2. Objetivos Específicos

- Aplicar de manera efectiva los patrones de diseño adecuados para mejorar la calidad y la estructura del código en proyectos de desarrollo de software.
- Utilizar los tres tipos de patrones de diseño como creacional, estructural y comportamiento para establecer relaciones entre objetos y gestionar la comunicación entre ellos.

3. Desarrollo

3.1. Que es patrones de diseños design patterns

Según (Canelo, 2020) los patrones de diseño, también conocidos como design patterns, son soluciones generales y reutilizables aplicables a diversos problemas de diseño de software. Estas plantillas identifican problemas comunes en el sistema y ofrecen soluciones probadas a través de la experiencia acumulada a lo largo del tiempo.

Mediante (Sánchez, 2017) cada patrón de diseño tiene un propósito específico y aborda problemas comunes de diseño. Podemos decir que, si el modelo de solución encontrado se adapta a múltiples ámbitos, entonces nos encontramos ante un posible patrón de diseño de software. Estos modelos de solución son considerados patrones de diseño si su efectividad fue probada resolviendo problemas similares en otras ocasiones. Su reutilización también debe estar probada.

Daremos a conocer las ventajas y desventajas de los patrones de diseño:

Ventajas

- Reutilización de soluciones probadas.
- Mejora de la calidad del código.
- Facilita la comunicación y colaboración en el equipo.

Desventajas

- Puede introducir complejidad innecesaria.
- Existe el riesgo de caer en el sobre diseño.

3.2. Tipos de patrones de diseños

Hay un total de 23 patrones de diseño diferentes, y las tres categorías más populares contienen los patrones de diseño más utilizados.

- patrones en la creación.
- patrones arquitectónicos.
- patrones de comportamiento

Imagen 1. Tipos de Patrones de Diseños

Resumen: *Los patrones de diseños son herramientas que desarrolla a resolver los problemas comunes de manera probada y estructurada ya que promueve las buenas prácticas de programación.* (Canelo, 2020)

3.2.1. Patrones de 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.

Estos son los patrones de la creación.

1. **Abstract Factory:** Este patrón se utiliza interfaz para crear conjuntos o familias de objetos relacionados sin especificar el nombre de la clase.

2. **Buider Patterns:** Abstrae la construcción de objetos complejos paso a paso, permitiendo diferentes representaciones del mismo proceso de construcción.
3. **Factory Method:** Define una interfaz para crear objetos, pero permite a las subclases decidir qué clase instanciar.
4. **Prototype:** Permite copiar objetos existentes sin hacer que su código dependa de sus clases.
5. **Singleton:** Garantiza que una clase tenga solo una instancia y proporciona un punto global de acceso a ella.

3.2.2. Patrones estructurales

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.

1. **Adapter:** Adaptador es un patrón que se utiliza para que objetos con interfaces incompatibles colaboren entre sí.
2. **Bridge:** resuelve un problema habitual en la herencia de clases dividiendo clases relacionadas en dos jerarquías diferentes: implementación y abstracción, para que estas puedan desarrollarse independientemente.
3. **Composite:** Solo se recomienda utilizar Composite cuando el modelo de código está creado a partir de un sistema ramificado en forma de árbol.
4. **Decorator:** Se utiliza para extender el comportamiento de un objeto añadiendo funcionalidades al mismo a través de objetos encapsuladores que presentan dichas funcionalidades.
5. **Facade:** Proporciona una interfaz simplificada para una biblioteca, un marco o cualquier otro conjunto complejo de clases.

6. **Flyweight:** Ayuda a reducir el tamaño de los objetos almacenando en su interior solo el estado intrínseco (información constante) del mismo y compartiendo el resto de la información (estado extrínseco) entre varios objetos similares.
7. **Proxy:** Se utiliza para crear objetos que pueden representar funciones de otras clases u objetos y la interfaz se utiliza para acceder a estas funcionalidades.

3.2.3. Patrones de comportamiento

Los patrones de comportamiento buscan resolver la comunicación entre diferentes áreas

1. **Chain of responsibility:** podremos evitar que la petición emitida por un emisor sea acoplada a un solo receptor permitiendo que más de un objeto pueda responder a dicha petición.
2. **Command:** Se utiliza cuando es necesario encapsular dentro de un objeto todos los parámetros que una acción requiere para ejecutarse.
3. **Interpreter:** Utilizando Interpreter podremos evaluar un lenguaje a través de una interfaz que indique el contexto en el cual se interpreta.
4. **Iterator:** Este patrón de comportamiento se utiliza cuando necesitamos iterar en colecciones o conjuntos de objetos sin la necesidad de intercambiar información relevante.
5. **Mediator:** Se utiliza cuando necesitamos controlar las comunicaciones directas entre objetos y disminuir sus dependencias caóticas.
6. **Memento:** Este patrón es capaz de almacenar y restaurar la información de un objeto.

7. **Observer:** A través de este patrón de comportamiento varios objetos interesados (suscriptores) en un objeto en particular (notificador) pueden recibir notificaciones de su comportamiento mientras estén suscriptos a sus notificaciones.
8. **State:** Se utiliza para modificar el comportamiento de una clase de objetos dependiendo del estado actual (comportamiento interno) de dichos objetos.
9. **Strategy:** Permite separar todos los algoritmos de una clase específica en nuevas clases separadas donde los objetos pueden intercambiarse.

3.3. Ejemplo de Abstract Factory

3.3.1. Planteamiento de Problema

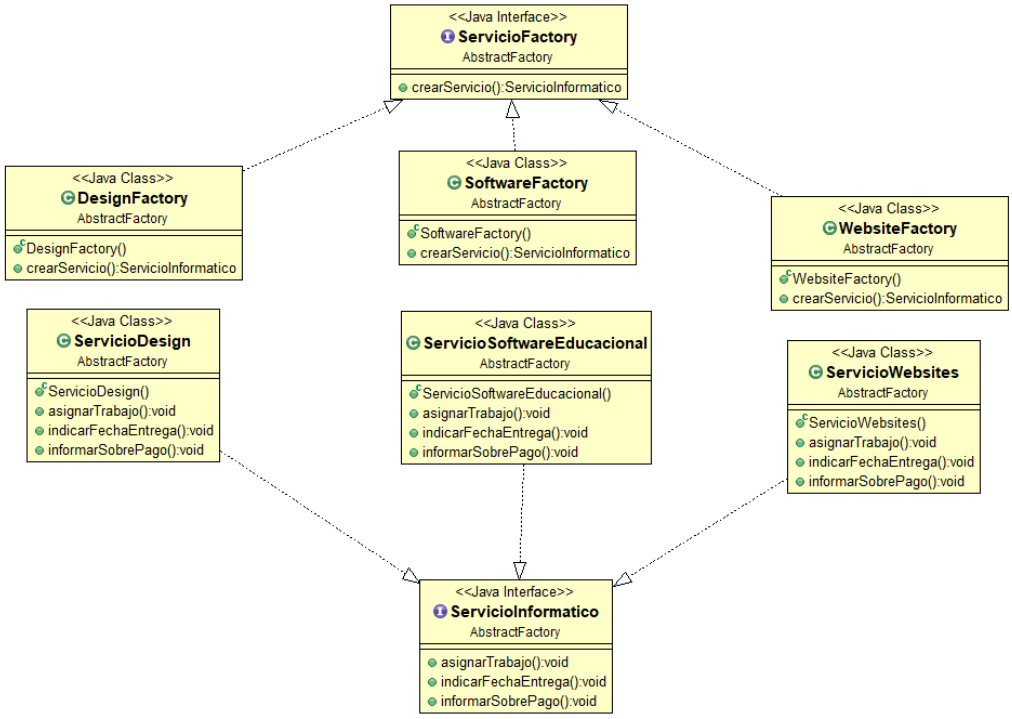
La empresa Nacional necesita realizar un servicio informático de información donde quiere saber sobre un diseño gráfico, un desarrollo web educacional y creación de sitio web indicar el trabajo, la fecha de entrega y saber sobre el pago del servicio.

3.3.2. Solución del Problema

Mediante un software realizaremos un ejercicio de información donde la empresa nacional podrá elegir las opciones de información que requiere saber por medio de interfaz para crear conjuntos o familias de objetos relacionados que ayudan al sitio de ayuda.

3.3.3. Diagrama de Clase

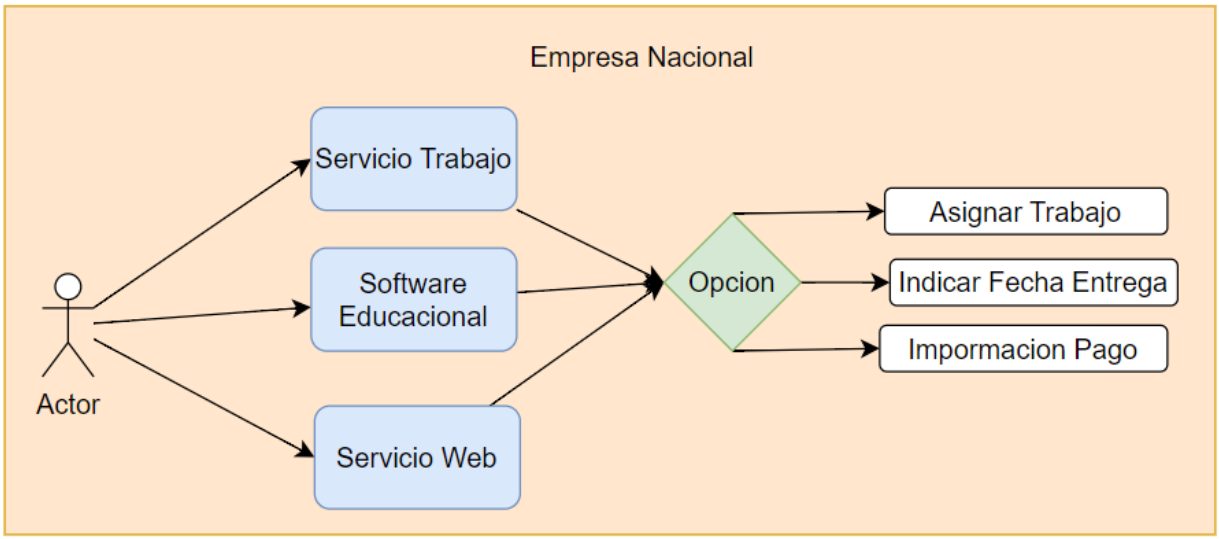
Imagen 2. Diagrama de clase por Kelvin Quezada



En este diagrama se elaboró una estructura para el servicio de información donde el usuario va a elegir tres arias en específica para saber la información que desea saber en las cuales son Servicio de trabajo, servicio de Educación, Servicio de sitio web.

3.3.4. Diagrama de Uml

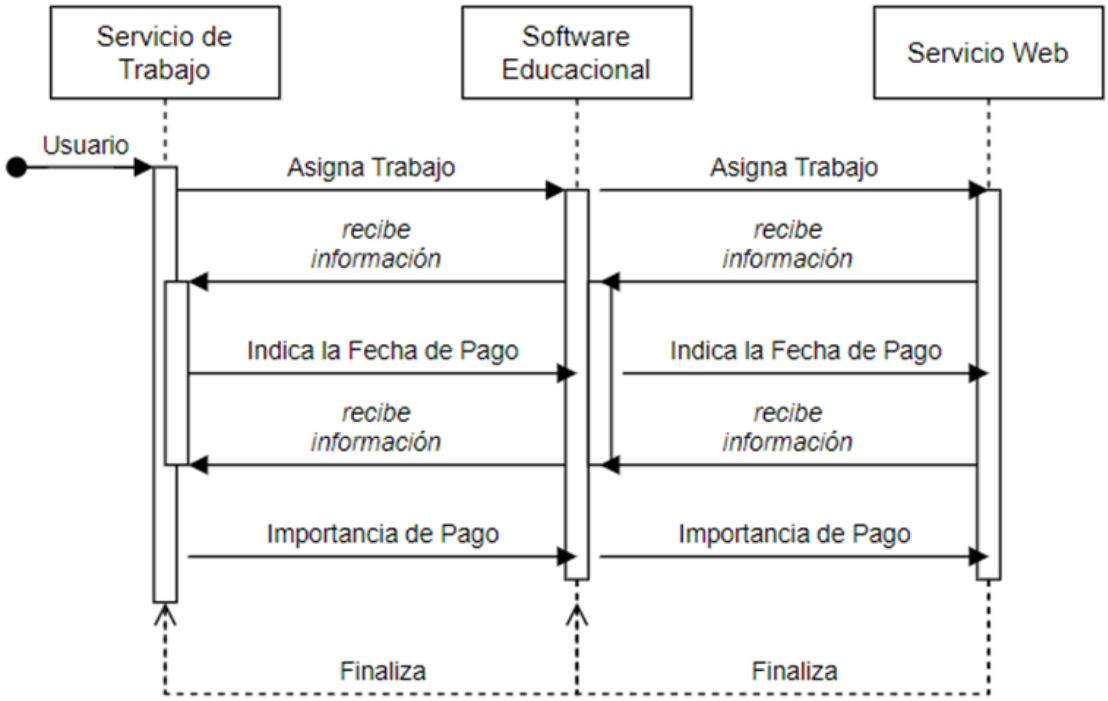
Imagen 3. Diagrama de Uml por Kelvin Quezada



Daremos a conocer un diagrama uml ya que nos ayuda en las interacciones entre el sistema y los actores mediante este diagrama uml nos indica que el usuario de una empresa necesita saber tres informaciones de tres arias en especifica mediante el autor Kelvin Quezada.

3.3.5. Diagrama de Secuencia

Imagen 4. Diagrama de secuencia por Kelvin Quezada



El usuario se va al registro de información ingresa al servicio de trabajo dará a conocer la información del trabajo que se genera, también conocerá la fecha de pago de ese trabajo y la importancia del pago y también conocerá información del software educación y servicio web, ya que nos permite producir familias de objetos relacionados sin especificar sus clases concretas según Kelvin Quezada.

3.3.6. Código del Ejercicio

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

```
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();
}
```

```
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();
   }

}
```

```
package AbstractFactory;
//creamos una clase ServicioSoftwareEducacional implements ServicioInformatico ya que son
//interfaces que definen un conjunto de métodos que una clase que las implementa debe proporcionar.
public class ServicioSoftwareEducacional implements ServicioInformatico {
   @Override
   public void asignarTrabajo() {
        System.out.println("Nuestros programadores han sido informados del programa que deben realizar.");
   }

   @Override
   public void indicarFechaEntrega() {
      System.out.println("Se ha fijado como fecha de entrega el dia 01/10/2023.");
   }

   @Override
   public void informarSobrePago() {
         System.out.println("El monto a pagar sera proporcional a la cantidad de estudiantes que haran uso del software.");
   }

}
```

```
package AbstractFactory;

public class ServicioTrabajo 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.");
    }

}
```

```
package AbstractFactory;

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 día 26/11/2023.");
    }

    @Override
    public void informarSobrePago() {
        System.out.println("El monto a pagar no incluye el pago por dominio y hosting.");
    }

}
```

```
package AbstractFactory;

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();
   }

}
```

```
package AbstractFactory;

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();
   }

}
```

```
package Prueba;

import AbstractFactory.DesignFactory;
import AbstractFactory.ServicioFactory;
import AbstractFactory.ServicioInformatico;
import AbstractFactory.SoftwareFactory;
import AbstractFactory.WebsiteFactory;
import java.util.Scanner;

public class Principal {
public static void main(String[] args) {
    Scanner leer = new Scanner(System.in);
    int opcion;
    do {
        System.out.print(
             "MENU DE OPCIONES\n"
           + "---- -- -------->\n"
           + "1. Solicitar servicio de diseno grafico.\n"
           + "2. Solicitar desarrollo de software educacional.\n"
           + "3. Solicitar creacion de sitios web.\n"
           + "4. Cerrar programa.\n"
           + "Seleccione opcion: "
       );
        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("A finalizado el programa ");
               break;
            default:
               System.out.println("A opcion es incorrecta ");

         }

   } while (opcion!=4);
   }
//creamos un metodo 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.
//https://www.youtube.com/watch?v=xNsPGA7zrVQ

public static void usarServicio(ServicioFactory factory) {
   //el ServicioInformatico es una clase interface que resive informacion
   ServicioInformatico servicio = factory.crearServicio();
      servicio.asignarTrabajo();
      servicio.indicarFechaEntrega();
      servicio.informarSobrePago();
   }
}
```

3.3.7. Anexos de Codigo

Imagen 5. Interface ServicioFactory por Kelvin Quezada

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

Imagen 6. ServicioFactory por Kelvin Quezada

```java
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();
}
```

Imagen 7. Clase DesignFactory por Kelvin Quezada

```java
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();
    }

}
```

Imagen 8. SercicioftwareEducacional por Kelvin Quezada

```java
package AbstractFactory;
//creamos una clase ServicioSoftwareEducacional implements ServicioInformatico ya que son
//interfaces que definen un conjunto de métodos que una clase que las implementa debe proporcionar.
public class ServicioSoftwareEducacional implements ServicioInformatico {
    @Override
    public void asignarTrabajo() {
        System.out.println(x: "Nuestros programadores han sido informados del programa que deben realizar.");
    }

    @Override
    public void indicarFechaEntrega() {
        System.out.println(x: "Se ha fijado como fecha de entrega el dia 01/10/2023.");
    }

    @Override
    public void informarSobrePago() {
        System.out.println(x: "El monto a pagar sera proporcional a la cantidad de estudiantes que haran uso del software.");
    }

}
```

Imagen 9. Clase ServicioTrabajo por Kelvin Quezada

```java
package AbstractFactory;

public class ServicioTrabajo implements ServicioInformatico {
//creamos tres metodos que estan sobrescribiendo es nuestra clase interfaz de ServicioInformatico
    @Override
    public void asignarTrabajo() {
        System.out.println(x: "El trabajado ha sido asignado a disenadores graficos disponibles.");
    }

    @Override
    public void indicarFechaEntrega() {
        System.out.println(x: "Ellos han determinado terminar el trabajo como maximo para el dia 09/08/2023.");
    }

    @Override
    public void informarSobrePago() {
        System.out.println(x: "Debe realizar el pago en efectivo al momento de recoger el logo completamente terminado.");
    }

}
```

Imagen 10. Clase ServicioWebsites por Kelvin Quezada

```java
package AbstractFactory;

public class ServicioWebsites implements ServicioInformatico {
//creamos tres metodos que estan sobrescribiendo es nuestra clase interfaz de ServicioInformatico
    @Override
    public void asignarTrabajo() {
        System.out.println(x: "El personal encargado del desarrollo de sitios web ha aceptado el trabajo.");
    }

    @Override
    public void indicarFechaEntrega() {
        System.out.println(x: "El sitio web con Responsabilida Designara terminado el día 26/11/2023.");
    }

    @Override
    public void informarSobrePago() {
        System.out.println(x: "El monto a pagar no incluye el pago por dominio y hosting.");
    }

}
```

Imagen 11. SoftWareFactory por Kelvin Quezada

```java
package AbstractFactory;

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();
    }

}
```

Imagen 12. WebsiFactory Por Kelvin Quezada

```java
package AbstractFactory;

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();
    }

}
```

Imagen 13. Clase Principal por Kelvin Quezada

```
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 diseno grafico.\n"
          + "2. Solicitar desarrollo de software educacional.\n"
          + "3. Solicitar creacion de sitios web.\n"
          + "4. Cerrar programa.\n"
          + "Seleccione opcion: "
      );
        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 opcion es incorrecta ");

        }

public static void usarServicio(ServicioFactory factory) {
    //el ServicioInformatico es una clase interface que resive informacion
    ServicioInformatico servicio = factory.crearServicio();
        servicio.asignarTrabajo();
        servicio.indicarFechaEntrega();
        servicio.informarSobrePago();
    }
}
```

3.3.8. Anexos de Ejecución del Programa

Imagen 14. Ejecución 1 Diseño Grafico por Kelvin Quezada

```
run:
MENU DE OPCIONES
---- -- -------->
1. Solicitar servicio de diseno grafico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creacion de sitios web.
4. Cerrar programa.
Seleccione opcion: 1
El trabajado ha sido asignado a disenadores graficos disponibles.
Ellos han determinado terminar el trabajo como maximo para el dia 09/08/2023.
Debe realizar el pago en efectivo al momento de recoger el logo completamente terminado.
```

Imagen 15. Ejecución 2 Software educacional por Kelvin Quezada

```
MENU DE OPCIONES
---- -- -------->
1. Solicitar servicio de diseno grafico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creacion de sitios web.
4. Cerrar programa.
Seleccione opcion: 2
Nuestros programadores han sido informados del programa que deben realizar.
Se ha fijado como fecha de entrega el dia 01/10/2023.
El monto a pagar sera proporcional a la cantidad de estudiantes que haran uso del software.
```

Imagen 16. Ejecución 3 Sitio Web por Kelvin Quezada

```
MENU DE OPCIONES
---- -- -------->
1. Solicitar servicio de diseno grafico.
2. Solicitar desarrollo de software educacional.
3. Solicitar creacion de sitios web.
4. Cerrar programa.
Seleccione opcion: 3
El personal encargado del desarrollo de sitios web ha aceptado el trabajo.
El sitio web con Responsabilida Designara terminado el d?a 26/11/2023.
El monto a pagar no incluye el pago por dominio y hosting.
```

4. 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.

5. Recomendaciones

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

6. Bibliografía

Canelo, M. M. (24 de Junio de 2020). *profile*. profile: https://profile.es/blog/patrones-de-diseno-de-software/#%C2%BFQue_son_los_patrones_de_diseno_design_patterns

Sánchez, M. Á. (22 de Noviembre de 2017). *medium*. Patrones de Diseño de Software: https://medium.com/all-you-need-is-clean-code/patrones-de-dise%C3%B1o-b7a99b8525e

Soto, N. (2021, Julio 2). *¿QUÉ SON LOS PATRONES DE DISEÑO?* Craft Code. https://craft-code.com/que-son-los-patrones-de-diseno/#:~:text=Los%20patrones%20estr

ucturales%20buscan%20facilitar,interfaces%20incompatibles%20colaboren%20entre%20s%C3%AD