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 : Tarea 2

NOMBRES : Kelvin Quezada

NIVEL-PARALELO : Segundo “A”

DOCENTE : Ing. Cevallos Farias Javier Moyota

FECHA DE ENTREGA : 14/08/2023

SANTO DOMINGO - ECUADOR

2023

Contenido

Imágenes

1. Introducción

Según (Lara, 2015) una aplicación se puede dividir en partes más manejables usando módulos gracias a la programación modular. Dado que cada módulo consta de una colección de sentencias que permiten la ejecución de una acción u operación, el programa también incluye un subprograma que se creó utilizando códigos independientes.

Según (Canelo, 2020) nos dice que los patrones de diseños son soluciones generales, 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.

Mediante esta investigación pudimos comprender que la programación modular y los patrones de diseños buscan mejorar la estructura y la reutilización de código ya que la programación modular se basan en dividir un programa en módulos independiente que facilite la comprensión del programa. Por otra parte, los patrones de diseños son soluciones que son probadas para resolver problemas mediante un diseño de software.

2. Objetivos

2.1. Objetivos Generales

- Diseñar un programa que ayude al aprendizaje de la programación modularidad y realizar los patrones de diseños.

2.2. Objetivos Específico

- Comprender y diseñar por medio del software como utilizar la programación modular y conocer sus funcionalidades.

- Aplicar de manera efectiva los patrones de diseño adecuados para mejorar la calidad y la estructura del código en proyectos de desarrollo de software.

3. Marco Teórico

3.1.Programación Modularidad

Según (Lara, 2015) el modularidad consiste en dividir un programa por medio de módulos ya que se puede compilarse por separados y tendrá conexiones mediante otros módulos.

La modularidad también cuenta con tres principios que son:

Capacidad de descomponer un sistema complejo: Un sistema se puede dividir en un subprograma por módulos de problemas porque se puede dividir en problemas más pequeños.

Capacidad de descomponer un sistema complejo: Dado que se utilizan algunos módulos y el programa está destinado a ser reutilizable, sugiere que puede entenderse cuando se crea mediante software.

Imagen 1. Capacidad de descomponer un sistema Complejo por Kelvin Quezada

```
// Nos dara a conocer las categoría de los modulos
case "A": maxper=8;break;
case "B": maxper=6;break;
case "C": maxper=4;break;
case "D": maxper=2;break;
```

Se conocer la descomposición de un sistema complejo de los módulos por Kelvin Quezada.

Comprensión de sistema en parte: Se encarga de separar las partes que nos ayuda a la comprensión del código del sistema mediante la modificación.

Imagen 2. Comprension de sistema en parte

```
/*Esta funcion calcula y devuelve el numero maximo de personas permitidas en una categoria especificada.
Recibe como entrada el codigo de categoría 'cate' y devuelve el numero maximo de personas 'maxper'*/
    static int MaximoPersonas(String cate){
        // Inicializa la variable que almacenara el numero maximo de personas permitidas.
        int maxper=0;
        // Utiliza un switch para asignar el valor correcto de 'maxper' segun la categoría especificada.
        switch(cate){
            // Nos dara a conocer las categoría
            case "A": maxper=8;break;
            case "B": maxper=6;break;
            case "C": maxper=4;break;
            case "D": maxper=2;break;

        }
        // Devuelve el numero maximo de personas permitidas segun la categoria especificada.
        return maxper;
    }
/*Esta funcion calcula y devuelve el numero maximo de PagoMensual permitidas en una categoria especificada.
Recibe como entrada el codigo de categoría 'cate' y devuelve el numero maximo de pagoMenusal 'maxper'*/
    static double PagoMensual(String cate){
        double pm=0;
        switch(cate){
            case "A": pm=40.00;break;
            case "B": pm=30.00;break;
            case "C": pm=20.00;break;
            case "D": pm=10.00;break;
        }
         // Devuelve el numero maximo de personas permitidas segun la categoria especificada.
        return pm;
    }
```

Nos ayuda a separar nuestros módulos en partes diferentes para conocer las categorías de cada uno de los módulos por Kelvin Quezada

3.1.2. ¿Cuáles son las características de la programación modular?

Según (Jose, 2023) el encapsulamiento ya que cada modulo es una unidad independiente.

La modularidad ya que permite los bloques de códigos que son pequeños y muy fáciles de comprender.

La abstracción mediante módulos se utiliza para proporcionar un aislamiento de los procesos subyacentes que ya que el código sea mas fácil de comprender.

La Reutilización ya que cada pieza de código se puede utilizar en otros proyectos lo que ayuda a reducir el tiempo y los costros del desarrollo del programa.

Desacoplamiento por medio de módulos no depende entre si y se pueden cambiar y actualizar de forma aislada si afectar a otros módulos.

3.2. ¿Que son los Patrones de diseños?

Según (Canelo, 2020) los patrones de diseño, lo podemos conocer como design patterns, son soluciones generales que son reutilizables aplicables a los distintos 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. Es posible que estemos ante un patrón de diseño de software si el modelo de solución encontrado es adaptable a varios campos. Si la eficacia de estos modelos de solución se ha demostrado al resolver problemas análogos en el pasado, se consideran patrones de diseño. 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.

3.2.1. 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. Estos son los cuatro grandes grupos:

- patrones en la creación.
- patrones arquitectónicos.

- patrones de comportamiento
- comportamiento

Imagen 3. 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)

3.2.2. 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. **El patrón Abstract Factory:** Se utiliza mediante interfaz para asi crear conjuntos de objetos que son relacionados en las diferentes clases sin especificar el nombre.

2. **Buider Patterns:** Abstrae la construcción de objetos complejos paso a paso, permitiendo diferentes representaciones del mismo proceso de construcción.
3. **Factory Method:** especifica una interfaz para crear objetos, pero permite que las subclases elijan qué clase instanciar.
4. **Prototype:** permite la duplicación de objetos sin necesidad de que su código dependa de las clases de los objetos originales.
5. **Singleton:** asegura que una clase tenga solo una instancia y ofrece un único punto de acceso global.

3.2.3. 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 se utiliza para que objetos mediante interfaces incompatibles entre sí.
2. **Bridge:** resuelve un problema habitual en la herencia ya que se divide en clases relacionando entre dos jerarquías diferentes: la implementación y abstracción.
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:** esta herramienta se usa para agregar funcionalidad a un objeto encapsulando objetos que ya tienen esa funcionalidad.
5. **Facade:** brinda a una biblioteca, marco u otro conjunto complicado de clases una interfaz de usuario optimizada.
6. **Flyweight:** una técnica para reducir el tamaño de los objetos que implica almacenar solo el estado intrínseco (también conocido como información

constante) del objeto y distribuir el resto de la información (también conocido como estado extrínseco) entre varios objetos similares.

7. **Proxy:** La interfaz se utiliza para acceder a las funcionalidades de otras clases u objetos, y se utiliza para crear objetos que puedan representar esas funcionalidades.

3.2.4. Patrones de comportamiento

Los patrones de comportamiento tienen como objetivo mejorar la comunicación entre varias áreas.

1. **Chain of responsibility:** podemos permitir que más de un objeto responda a una solicitud evitando que se adjunte a un solo receptor.
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.Ejercicio de modular y Patrones de diseños

3.3.1. Ejercicio de Programación Modular

Objetivo: Permitir desarrollar la lógica del alumno mediante la resolución del problema aplicando la programación modular con la ayuda de las funciones.

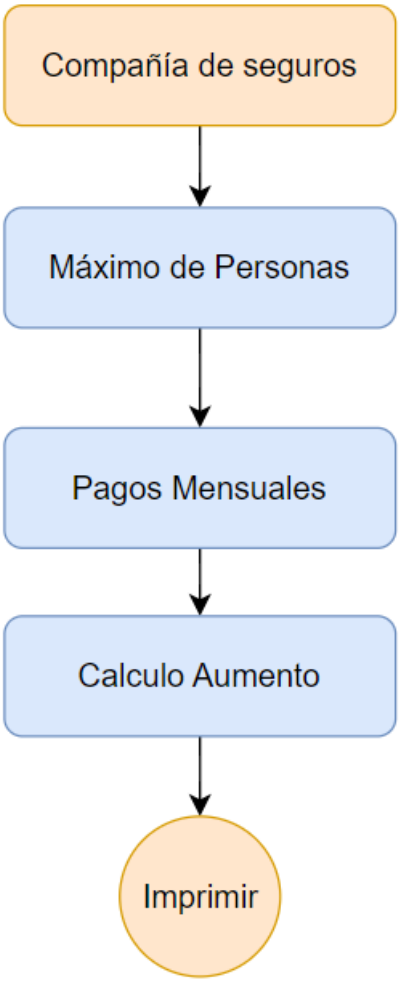
1. Una compañía de seguros ofrece a sus clientes cuatro tipos de seguros de sepilio

Tipo	Máximo número de Personas	Pagos mensuales (s/.)
A	8	40
B	6	30
C	4	20
D	2	10

Si el cliente asegura a más personas de la indicadas en el cuadro anterior tendrá que pagar S/8.00 mensuales Por cada persona Adicional si es que el seguro es de tipo A o B Será de S/5.00 mensuales por cada persona adicional si es que seguro es de tipo COD calcular el monto anual que tiene que pagar un determinado cliente.

3.3.2. Diagrama de Clases

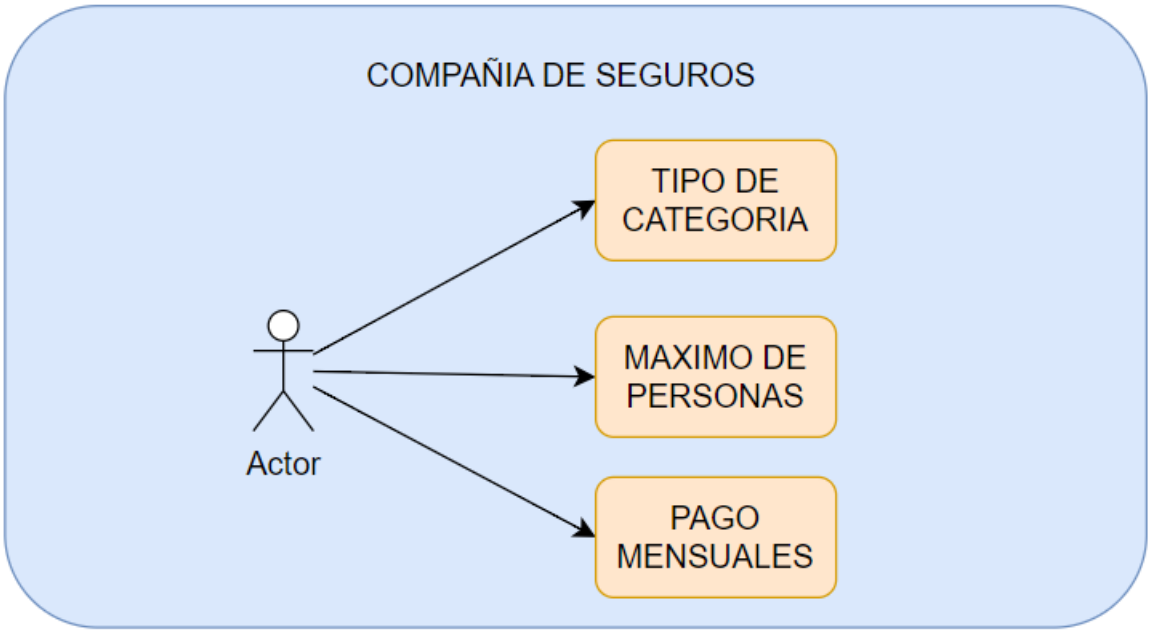
Imagen 4. Diagrama de clases de programacion modular



Mediante este diagrama de clases podemos conocer los módulos de nos ayuda mucho a entendimiento de módulos y conocer sus entradas como salidas por Kelvin Quezada.

3.3.3. Diagrama de Uml

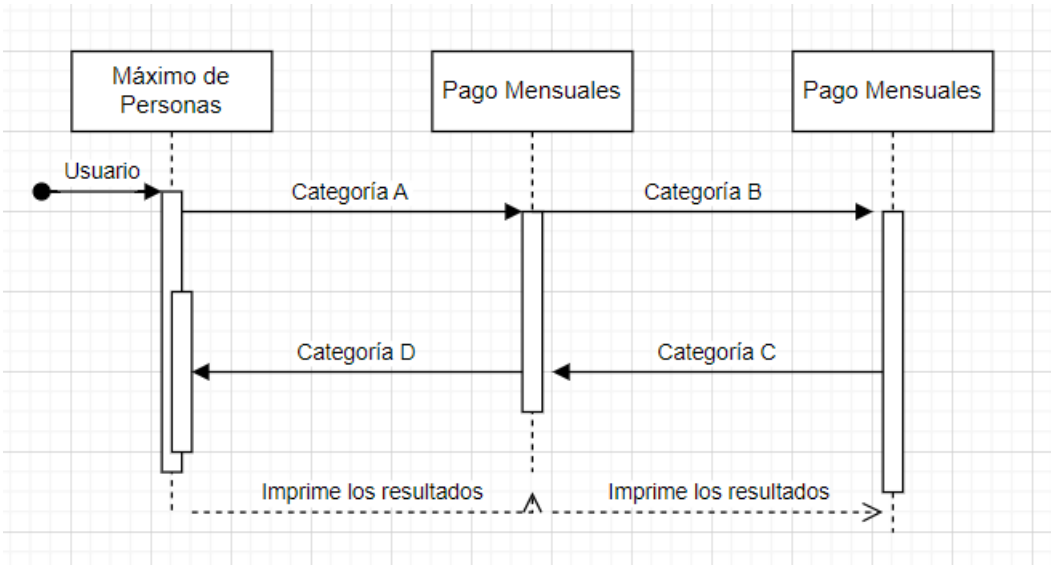
Imagen 5. Diagrama Uml de programación modular por Kelvin Quezada



Mediante el diagrama de Uso podemos ver como se aplico la programación modular ya que se divide por entrada procesos y salidas por Kelvin Quezada.

3.3.4. Diagrama de Secuencia

Imagen 6. Diagrama de Secuencia



Se desarrollará los pasos necesarios de las categorías y conocer sus procesos de cómo se desarrolló esos módulos correspondientes por Kelvin Quezada.

3.3.5. Código de Programa

```
package promodular;

import java.util.Scanner;

public class ProModular {
    public static void main(String[] args) {
        Scanner leer=new Scanner(System.in);
        //llamaremos nuestras variables
        double pmi,pmf,pa,aum;
        //creamos nuestra variable
        int cant_personas,maxper;
        String cate;
        System.out.println("Ingrese la categoria A, B, C, D");
        cate=leer.next();
        System.out.println("Ingrese la cantidad de personas: ");
        cant_personas=leer.nextInt();
        //validamos nuestros modulos
        maxper=MaximoPersonas(cate);
```

```
        pmi=PagoMensual(cate);
        aum=CalcAumento(cate);
        //creamos una condicional donde la cantidad de persona es mayor a maxper
        //entonces el pmf=pmi mas a la cantida de persona - maxper se multiplica por el aum
        if (cant_personas > maxper) {
            pmf=pmi+((cant_personas-maxper)*aum);
            //oh sino el pmf es =pmi
        } else {
            pmf=pmi;
        }
        //calculamos el pago anual
        pa=pmf*12;
        // Llamamos a la función para imprimir los resultados
        Imprimir(pmi, pmf, pa);
    }
```

/*Esta funcion calcula y devuelve el numero maximo de personas permitidas en una categoria especificada.

Recibe como entrada el codigo de categoría 'cate' y devuelve el numero maximo de personas 'maxper'*/

```
    static int MaximoPersonas(String cate){
        // Inicializa la variable que almacenara el numero maximo de personas permitidas.
        int maxper=0;
```

```
        // Utiliza un switch para asignar el valor correcto de 'maxper' segun la categoría especificada.
        switch(cate){
            // Nos dara a conocer las categoría
            case "A": maxper=8;break;
            case "B": maxper=6;break;
            case "C": maxper=4;break;
            case "D": maxper=2;break;

        }
        // Devuelve el numero maximo de personas permitidas segun la categoria especificada.
        return maxper;
    }
    /*Esta funcion calcula y devuelve el numero maximo de PagoMensual permitidas en una categoria especificada.
    Recibe como entrada el codigo de categoría 'cate' y devuelve el numero maximo de pagoMenusal 'maxper'*/
    static double PagoMensual(String cate){
        double pm=0;
        switch(cate){
            case "A": pm=40.00;break;
            case "B": pm=30.00;break;
```

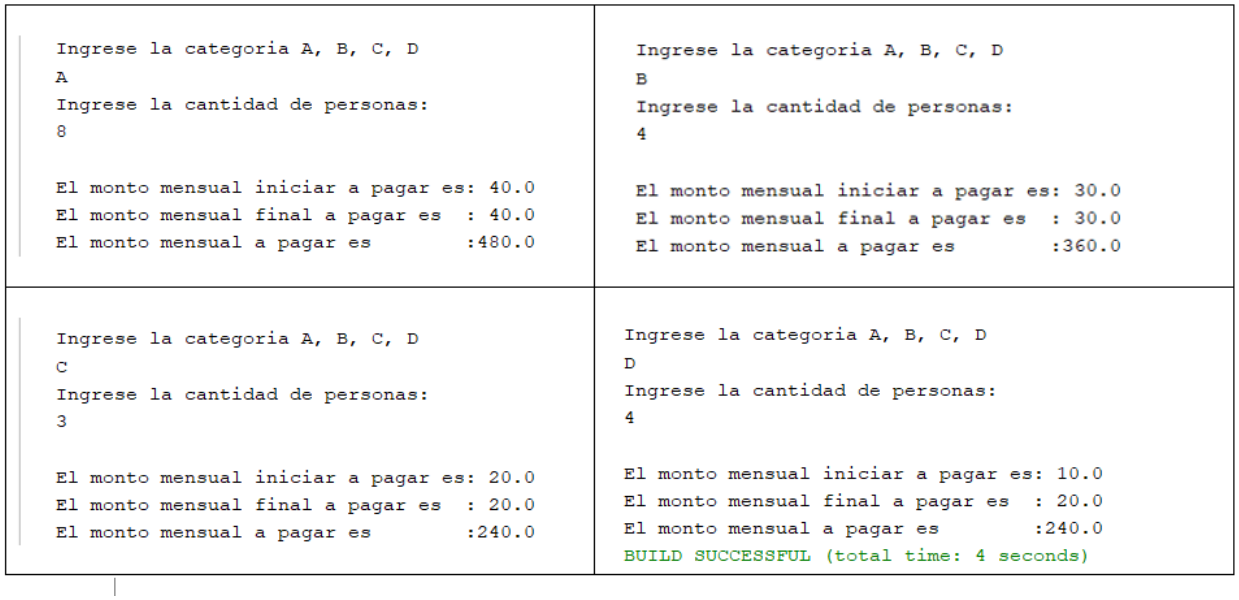
```
            case "C": pm=20.00;break;
            case "D": pm=10.00;break;
        }
        // Devuelve el numero maximo de personas permitidas segun la categoria especificada.
        return pm;
    }

    static double CalcAumento(String cate){
        double aum=0;
        switch(cate){
            case "A": aum=8.00;break;
            case "B": aum=8.00;break;
            case "C": aum=5.00;break;
            case "D": aum=5.00;break;
        }
        return aum;
    }
    //creamos un modulo para imprimir
    static void Imprimir(double pmi,double pmf,double pa){
        System.out.println("\nEl monto mensual iniciar a pagar es: "+pmi+
                           "\nEl monto mensual final a pagar es  : "+pmf+
                           "\nEl monto mensual a pagar es        :"+pa);
```

```
    }
}
```

3.3.6. Ejecución de Programación Modular

Imagen 7. Ejecucion del Programa Por Kelvin Quezada



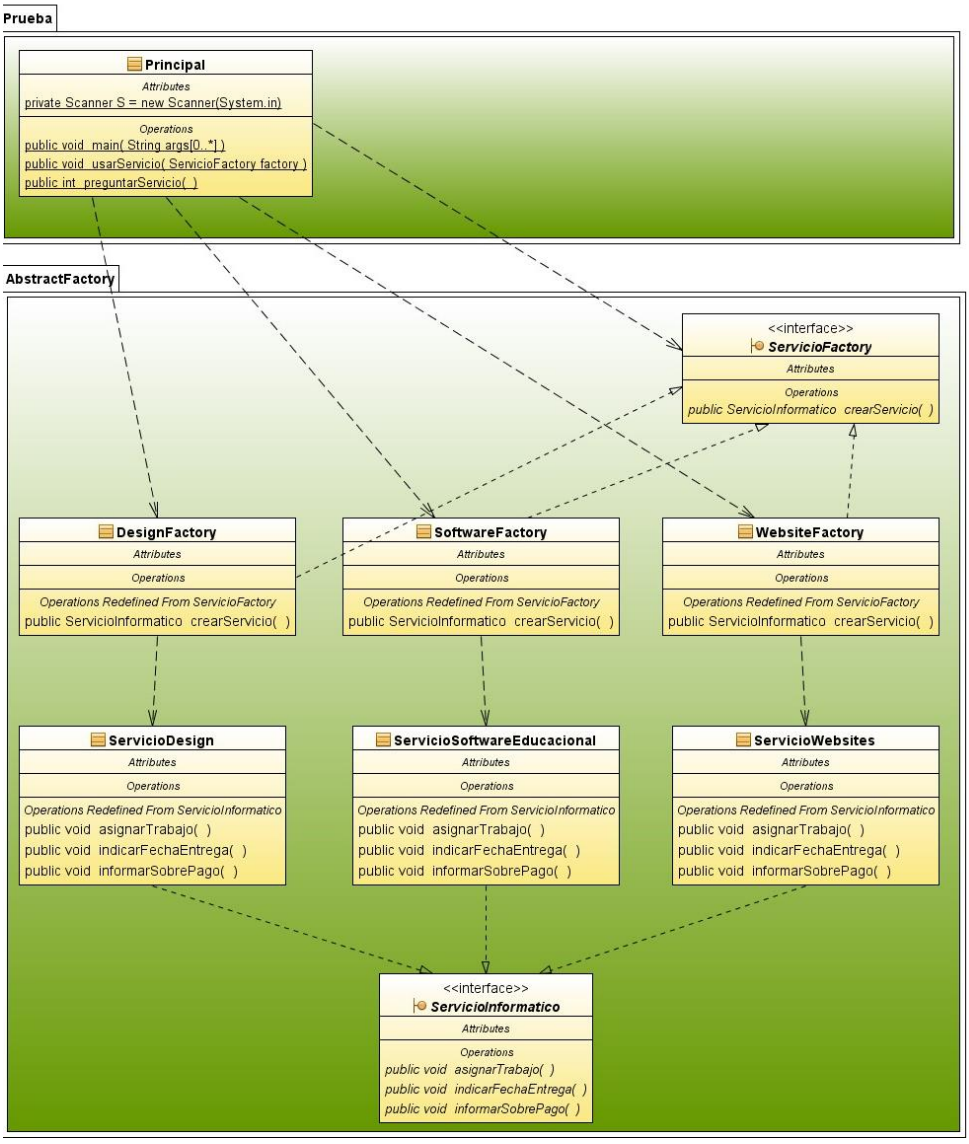
Dara a conocer los resultados de nuestro ejercicio conociendo las categorías de cada tipo por Kelvin Quezada.

3.4.1. Ejercicio de Patrones de Diseños

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.

3.4.2. Diagrama de Clases

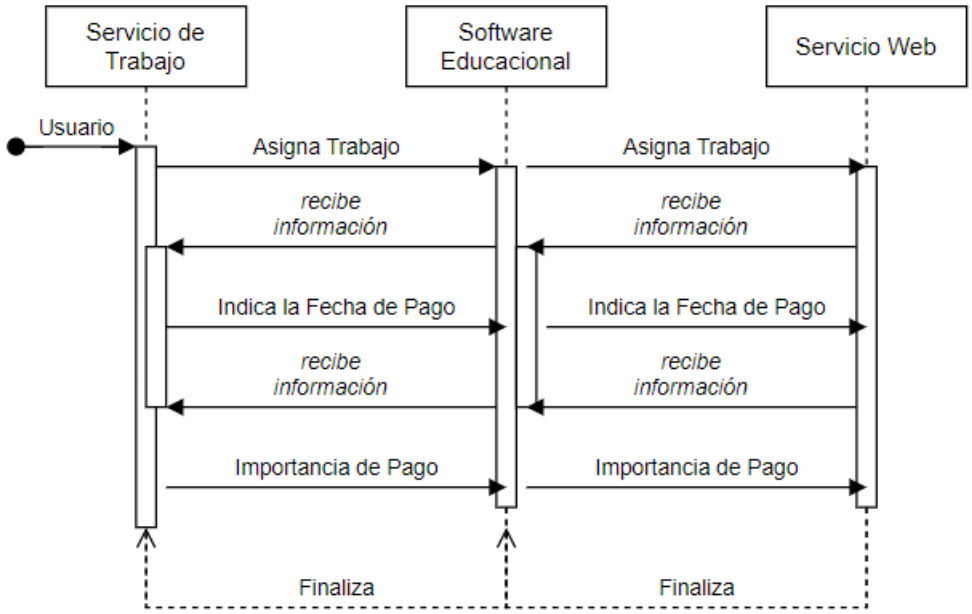
Imagen 8. Diagrama de Clases de Abstro Factory Por Kelvin Quezada



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.

3.4.3. Diagrama de Secuencia

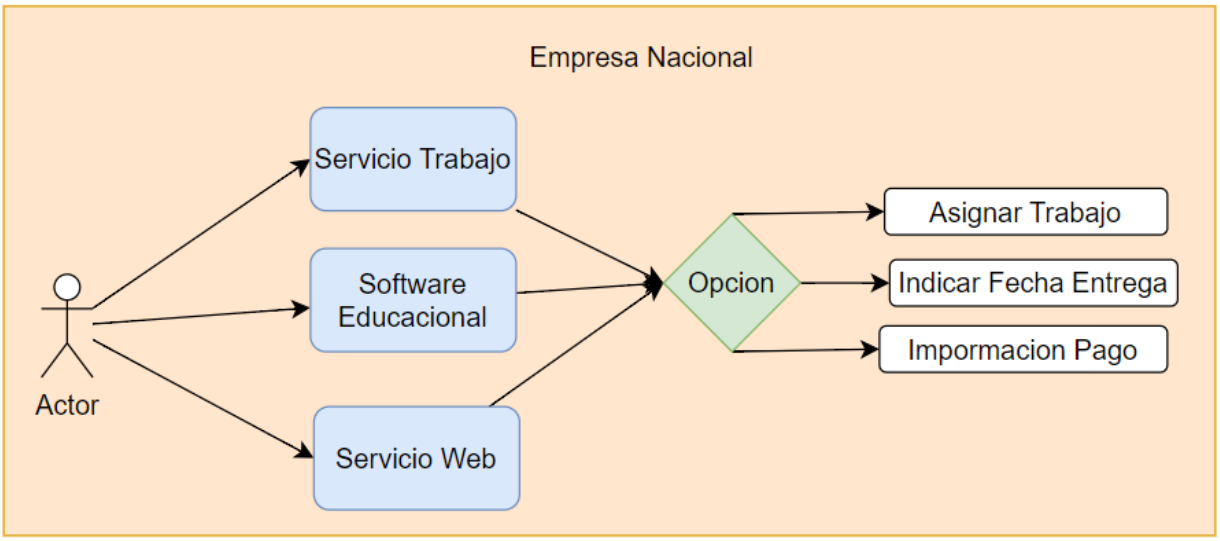
Imagen 9. Diagrama de Secuencia de Patrones de diseños Por Kelvin Quezada



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.

3.4.4. Diagrama de Uso

Imagen 10. Diagrama Uml de patrones de diseños 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.

3.4.5. Código del Programa

```
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();
    }
}
```

3.4.6. Ejecución del Programa

Imagen 11. Ejecucion de Patron de diseños 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.

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.
```

Se dará a conocer los resultados de los patrones de diseños como la información se aplico lo que es Abstract Factory por Kelvin Quezada.

4. Conclusiones

- La programación modular permite simplificar y automatizar problemas en la programación ya que el uso de los módulos permite la reutilización de código.
- La verificación de código modular es mas sencillo y rápido que hace la verificación de programas secuenciales complejas.
- Mejora la calidad del código al proporcionar soluciones probadas para problemas comunes.
- Fomenta la reutilización de soluciones exitosas, acelerando el desarrollo y reduciendo errores.

5. Recomendación

- Realizar las pruebas exhaustivas en cada modulo por separado que garantizar su correcto funcionamiento.
- Documenta cada modulo describiendo sus propósitos funcionalidades y como interactúa con otro módulos.
- 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

Jose. (05 de Febrero de 2023). *¿Qué es la programación modular? - Conoce todo sobre la descomposición de un programa en trozos más pequeños*. ¿Qué es la programación modular? - Conoce todo sobre la descomposición de un programa en trozos más pequeños: https://quees.com/programacion-modular/#%C2%BFQu%C3%A9_es_la_programaci%C3%B3n_modular_en_JavaScript?

Lara, D. (07 de Julio de 2015). *Styde*. Modularidad en la programación orientada a objetos: https://styde.net/modularidad-en-la-programacion-orientada-a-objetos/

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. (2 de Julio de 2021). *Craft Code*. Craft Code: https://craft-code.com/que-son-los-patrones-de-diseno/#:~:text=Los%20patrones%20estructurales%20buscan%20facilitar,interfaces%20incompatibles%20colaboren%20entre%20s%C3%AD.

Link de Videos de YouTube Programación Modular

https://www.youtube.com/watch?v=bZHeZKUDRMY

https://www.youtube.com/watch?v=Poi8W6GVp8o

Link de patrones de Diseños

https://youtu.be/h96NCRB5Xmc

https://www.youtube.com/watch?v=r2-k8pGbrbI

https://www.youtube.com/watch?v=yU4zZB-GfVs