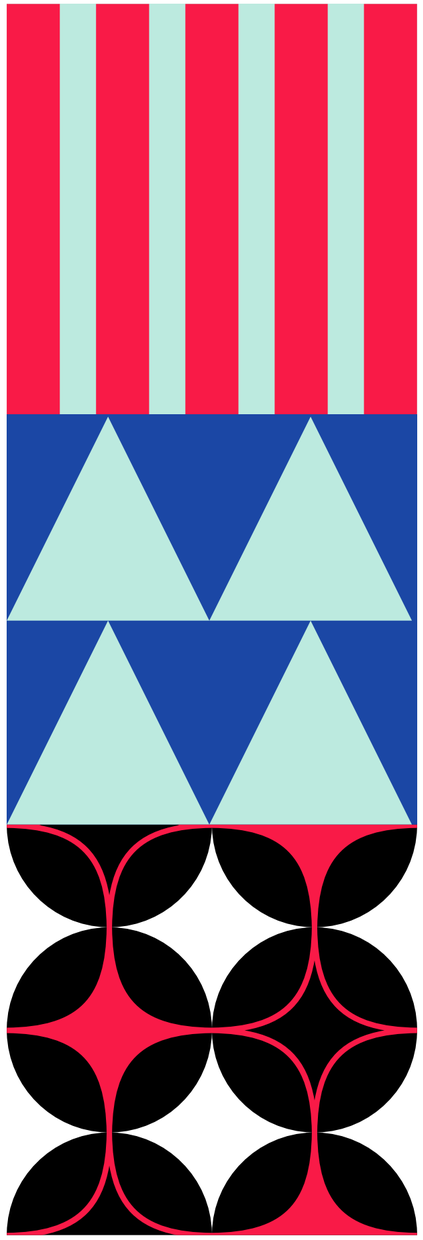
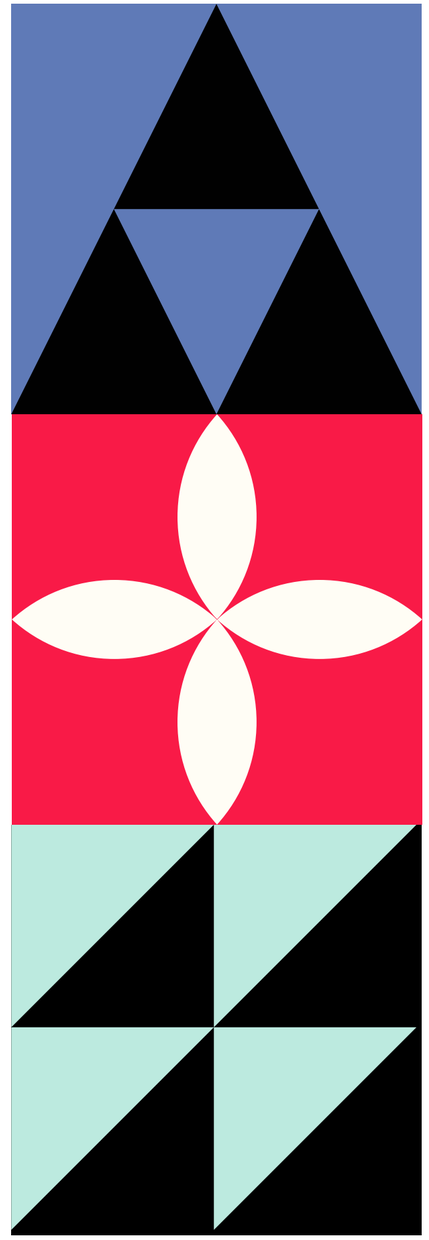


Investigación 2

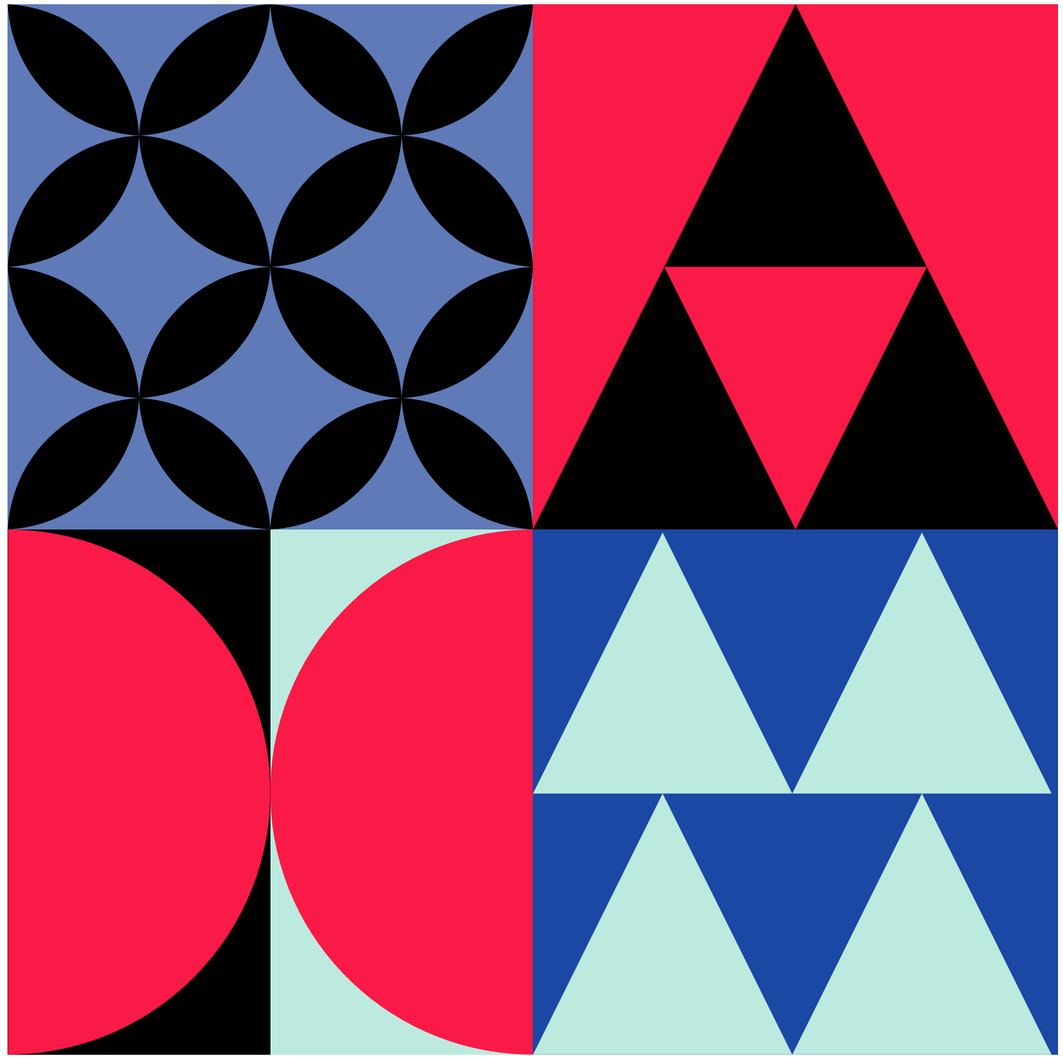
PROTOTYPE PATTERN

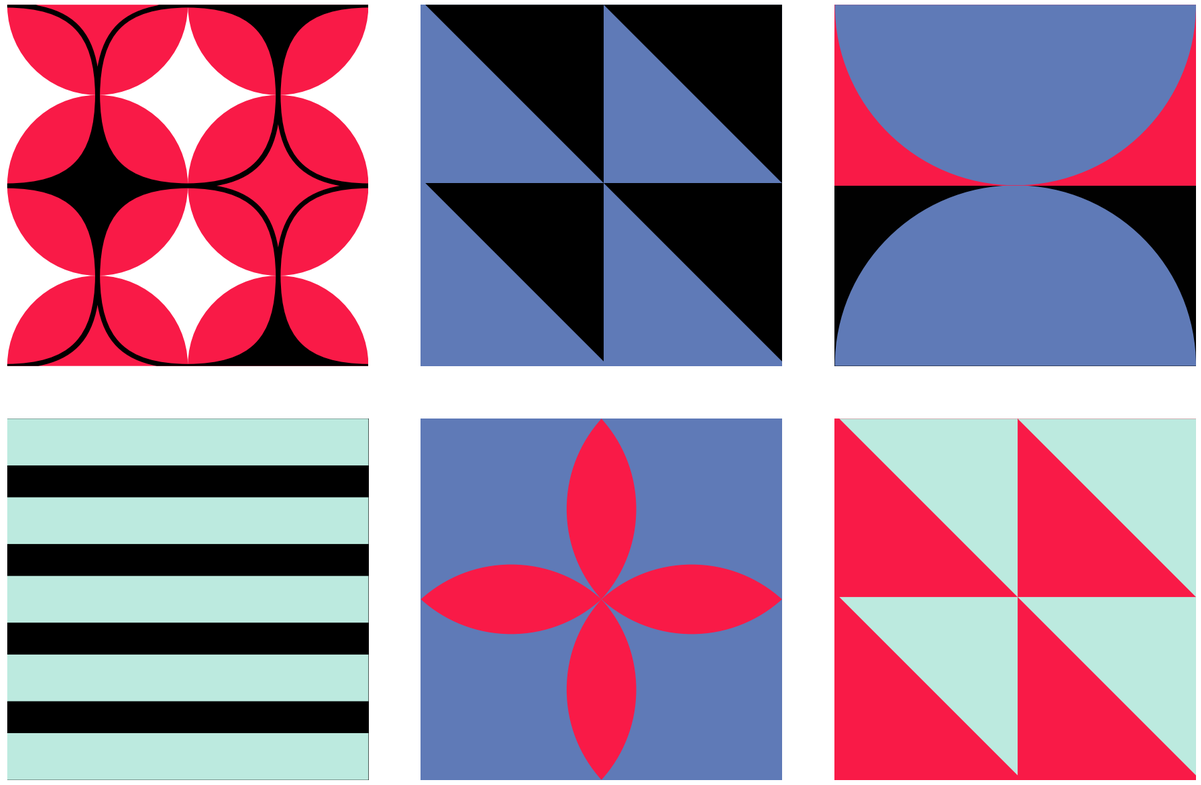
Mary Paz Álvarez Navarrete



DESIGN PATTERNS

Son una guía para resolver problemas comunes en el desarrollo de software. Al enconrarse con problemas similares, se pueden adaptar estos patrones, ya que son soluciones generales y reutilizables para problemas que suelen repetirse.





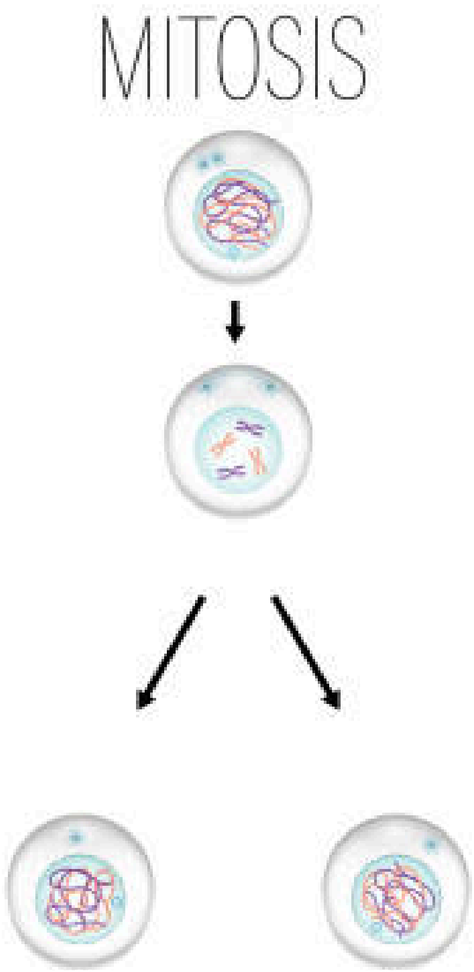
CREATIONAL PATTERNS

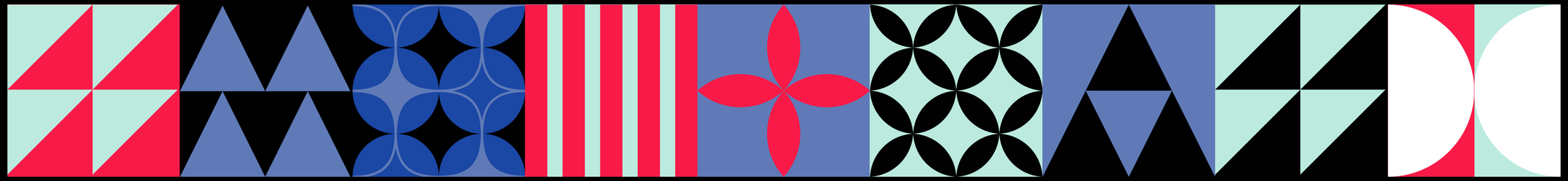
Se enfocan en lo que es la creación de objetos.
Independizan el sistema de como se crean, representa o componen esos objetos.

PATRÓN CREACIONAL PROTOTYPE

Consiste en crear nuevos objetos copiando una instancia prototípica existente, en lugar de construirlos desde cero.

Este patrón se usa cuando el costo de crear un nuevo objeto es elevado o cuando se quieren duplicar objetos con un estado específico, sin alterar el original





EJEMPLO: CLASE PERSONA

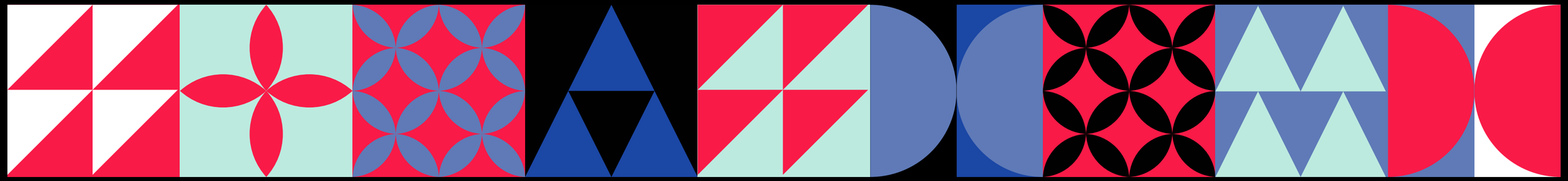
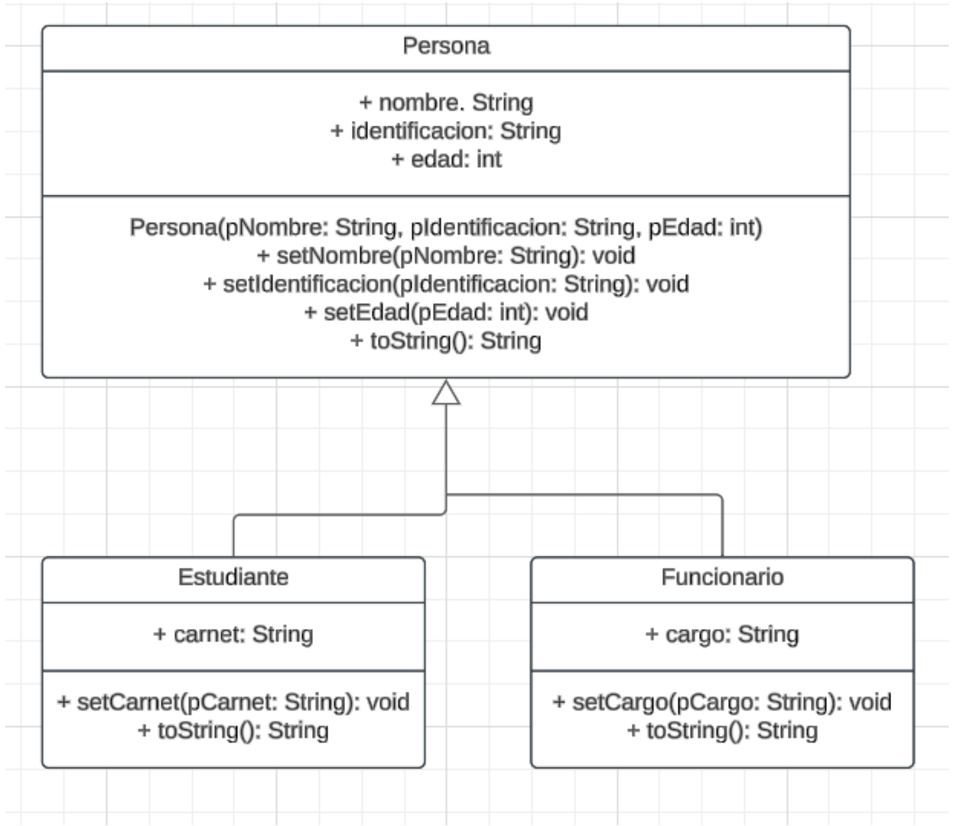
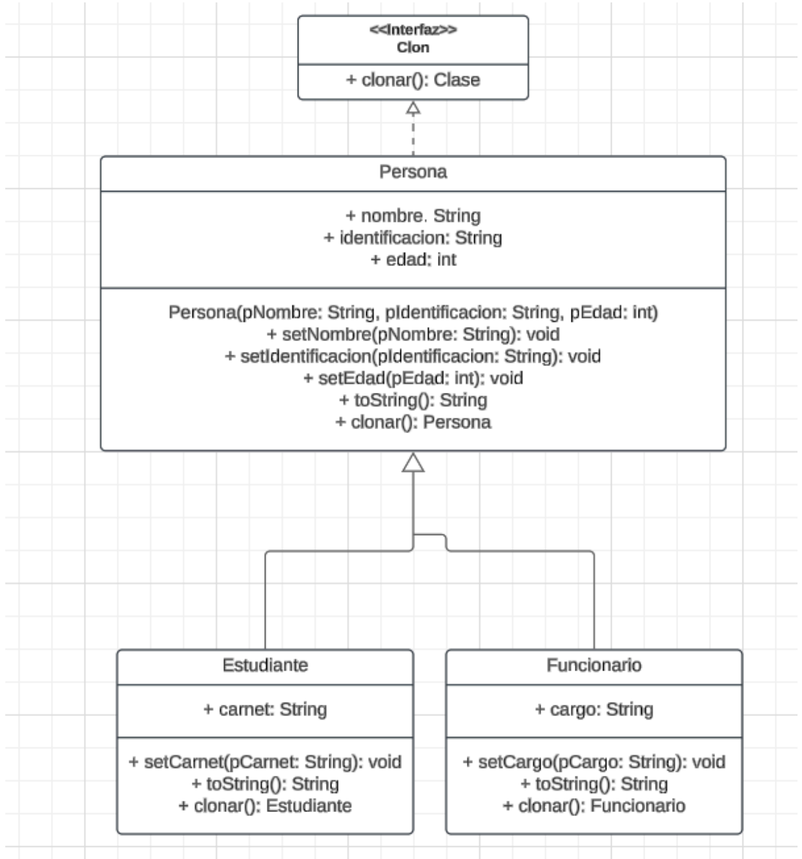


DIAGRAMA UML

SIN APLICAR EL PATRÓN



APLICANDO EL PATRÓN



CÓDIGO EN JAVA

INTERFAZ CLON

```java
// Creación de la interface Clon

public interface Clon {
    // Método para clonar un objeto.
    Clon clonar();
}
```

CÓDIGO EN JAVA

CLASE PERSONA

```java
// Creación de la superclase Persona que implementa la interfaz Clon
// Esta clase representa a una persona con atributos básicos como nombre, apellidos, identificación y edad.

public class Persona implements Clon {
    // Atributos de la clase
    String nombre;
    String identificacion;
    int edad;

    // Constructor de la clase que inicializa todos los atributos
    public Persona(String pNombre, String pIdentificacion, int pEdad) {
        this.nombre = pNombre;
        this.identificacion = pIdentificacion;
        this.edad = pEdad;
    }

    // Métodos set para todos los atributos
    // Método para establecer el nombre de la persona
    public void setNombre(String pNombre) {
        this.nombre = pNombre;
    }

    // Método para establecer la identificación de la persona
    public void setIdentificacion(String pIdentificacion) {
        this.identificacion = pIdentificacion;
    }
```

```java
    // Método para establecer la edad de la persona
    public void setEdad(int pEdad) {
        this.edad = pEdad;
    }

    // Método toString
    // Muestra la información del objeto
    public String toString() {
        String info = "Nombre Completo: " + this.nombre + " " + "\n";
        info += "Identificación: " + this.identificacion + "\n";
        info += "Edad: " + this.edad + "\n";

        return info;
    }

    // Método clonar
    // Crea y devuelve una copia del objeto actual
    @Override
    public Clon clonar() {
        // Se utiliza el operador new para crear una nueva instancia de Persona
        // Se pasan los atributos actuales al constructor para inicializar la copia
        return new Persona(this.nombre, this.identificacion, this.edad);
    }
}
```

CÓDIGO EN JAVA

CLASE ESTUDIANTE

```java
// Creacion de la clase hija Estudiante que herada Persona

public class Estudiante extends Persona {
    // Método de estudiante
    String carnet;

    // Constructor
    // Se inicializan los atrbutos generales + el carnet
    public Estudiante(String pNombre, String pIdentificacion, int pEdad, String pCarnet) {
        super(pNombre, pIdentificacion, pEdad);
        this.carnet = pCarnet;
    }

    // Método set para carnet
    public void setCarnet(String pCarnet) {
        this.carnet = pCarnet;
    }

    // Se sobreescribe el método toString para añadir la información del carnet
    @Override
    public String toString() {
        String info = "Nombre Completo: " + this.nombre + " "  + "\n";
        info += "Identificación: " + this.identificacion + "\n";
        info += "Edad: " + this.edad + "\n";
        info += "Carnét: " + this.carnet + "\n";

        return info;
    }

    // Se sobreescribe el método clonar para añadr el atributo de carnet
    @Override
    public Clon clonar() {
        return new Estudiante(this.nombre, this.identificacion, this.edad, this.carnet);
    }
}
```

CLASE FUNCIONARIO

```java
// Creacion de la clase hija Funcionario que herada Persona

public class Funcionario extends Persona {
    // Método de estudiante
    String cargo;

    // Constructor
    // Se inicializan los atrbutos generales + el cargo
    public Funcionario(String pNombre, String pIdentificacion, int pEdad, String pCargo) {
        super(pNombre, pIdentificacion, pEdad);
        this.cargo = pCargo;
    }

    // Método set para cargo
    public void setCargo(String pCargo) {
        this.cargo = pCargo;
    }

    // Se sobreescribe el método toString para añadir la información del cargo
    @Override
    public String toString() {
        String info = "Nombre Completo: " + this.nombre + " " + "\n";
        info += "Identificación: " + this.identificacion + "\n";
        info += "Edad: " + this.edad + "\n";
        info += "Cargo: " + this.cargo + "\n";

        return info;
    }

    // Se sobreescribe el método clonar para añadr el atributo de carnet
    @Override
    public Clon clonar() {
        return new Funcionario(nombre, this.identificacion, this.edad, this.cargo);
    }

}
```

CÓDIGO EN JAVA

MAIN

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Creación del un estudiante (original)
        Estudiante estudianteOriginal = new Estudiante(pNombre:"Mary Paz Alavrez Navarrete", pIdentificacion:"12345678", pEdad:19, pCarnet:"2023138604");

        // Creación de un funcionario (original)
        Funcionario funcionarioOriginal = new Funcionario(pNombre:"Katherine Rodriguez Amador", pIdentificacion:"23456789", pEdad:26, pCargo:"Profesora");

        // Clonar al estudiante (Prototype)
        // Se hace un casting porque clonar devuleve un tipo Clon pero tiene que devolver un tipo Estudiante
        Estudiante clonEstudiante = (Estudiante) estudianteOriginal.clonar();

        // Clonar al funcionar (Prototype)
        Funcionario clonFuncionario = (Funcionario) funcionarioOriginal.clonar();

        // Cambiar datos de nombre, identificacion y carnet
        clonEstudiante.setNombre(pNombre:"Kristel Barrantes Garcia");
        clonEstudiante.setIdentificacion(pIdentificacion:"45678927");
        clonEstudiante.setCarnet(pCarnet:"2023458974");

        // Cambiar datos de nombre, identificacion y cargo
        clonFuncionario.setNombre(pNombre:"Michael Valladarez Hidalgo");
        clonFuncionario.setIdentificacion(pIdentificacion:"3125723892");
        clonFuncionario.setCargo(pCargo:"Secretario");

        // Mostrar estudiante original
        System.out.println(x:"Estudiante Original");
        System.out.println(estudianteOriginal.toString());
        System.out.println(x:"\n");

        // Mostrar estudiante clonado
        System.out.println(x:"Estudiante Clonado");
        System.out.println(clonEstudiante.toString());
        System.out.println(x:"\n");

        // Mostrar funcionario original
        System.out.println(x:"Funcionario Original");
        System.out.println(funcionarioOriginal.toString());
        System.out.println(x:"\n");

        // Mostrar funcionario clonado
        System.out.println(x:"Funcionario Clonado");
        System.out.println(clonFuncionario.toString());
        System.out.println(x:"\n");
    }
```

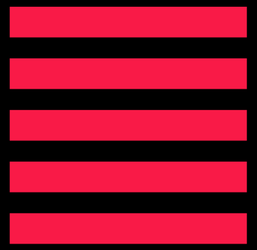
RESULTADOS

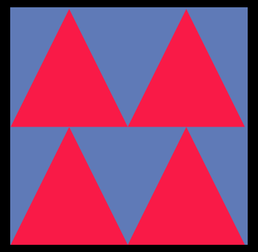
```
Estudiante Original
Nombre Completo: Mary Paz Alavrez Navarrete
Identificación: 12345678
Edad: 19
Carnét: 2023138604



Estudiante Clonado
Nombre Completo: Kristel Barrantes Garcia
Identificación: 45678927
Edad: 19
Carnét: 2023458974



Funcionario Original
Nombre Completo: Katherine Rodriguez Amador
Identificación: 23456789
Edad: 26
Cargo: Profesora



Funcionario Clonado
Nombre Completo: Michael Valladarez Hidalgo
Identificación: 3125723892
Edad: 26
Cargo: Secretario
```

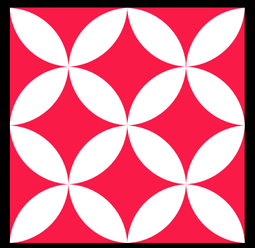
¿EN QUE CASO SE IMPLEMENTARIA?

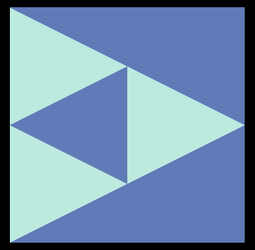
Para un sistema universitario que recibe un gran número de estudiantes y funcionarios, es común que estas personas compartan características similares. Por ejemplo, los estudiantes de primer ingreso suelen estar matriculados en los mismos cursos y, en muchos casos, tienen la misma edad. De igual manera, existen funcionarios que son contratados para posiciones similares.

Al implementar este patrón, podemos evitar la creación de múltiples objetos individuales. En su lugar, podemos utilizar un objeto base que contenga las propiedades comunes y crear copias de este objeto para representar a cada nuevo estudiante o funcionario. Esto no solo optimiza el uso de memoria, sino que también simplifica la gestión y el mantenimiento del sistema.









REFERENCIAS

https://refactoring.guru/es/design-patterns/prototype

https://www.javier8a.com/itc/bd1/articulo.pdf

https://devexpert.io/prototype-patrones-diseno/

¡GRACIAS POR SU ATENCIÓN!