PATRONES DE DISEÑO

Patrón Singleton

Asegura que una clase tiene sólo una instancia proporcionando un punto de acceso a esta instancia.

```
public class Singleton {
   private static Singleton instance = new Singleton();

   private Singleton() {
   }

   public static Singleton getInstance() {
      return instance;
   }

   public void mostrarMensaje() {
      System.out.println("Hola, ¿como estas?");
   }

}

public class Main {
   public static void main(String[] args) {
      Singleton singleton = Singleton.getInstance();
      singleton.mostrarMensaje();
   }
}
```

Patrón Fábrica

crea una interfaz la cual es implementada por diferentes subclases cuyo comportamiento varía de acuerdo a diferentes casos.

```
public interface MetodoPago {
   void pagar();
}

public class Efectivo implements MetodoPago{
   @Override
   public void pagar() {
      System.out.println("Pago en efectivo.");
   }
}
```

```
public class Pse implements MetodoPago{
  @Override
  public void pagar() {
    System.out.println("Pago por PSE.");
  }
}

public class MetodoPagoFactory {
  public static MetodoPago metodoPago(String metodoPago) {
    if(metodoPago.equals("EFECTIVO")){
      return new Efectivo();
    } else if (metodoPago.equals("PSE")) {
      return new Pse();
    } else {
      return null;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    MetodoPago efectivo = MetodoPagoFactory.metodoPago("EFECTIVO");
    efectivo.pagar();
    MetodoPago pse = MetodoPagoFactory.metodoPago("PSE");
    pse.pagar();
  }
}
```

Patrón adaptador

Funciona como un conector entre dos interfaces que no pueden comunicarse de forma directa.

```
public class Empleado {
  private int id;
  private String nombre;

  public Empleado(int id, String nombre) {
    this.id = id;
    this.nombre = nombre;
  }

  public Empleado() {
  }

  public int getId() {
    return id;
```

```
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getNombre() {
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
}

public class EmpleadoRequest {
  private String nombre;

  public EmpleadoRequest(String nombre) {
    this.nombre = nombre;
  }

  public EmpleadoRequest() {
  }

  public String getNombre() {
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

}

public interface IEmpleadoRepository {
  void guardarEmpleado(Empleado empleado);
}

public class EmpleadoRepository implements IEmpleadoRepository{
  @Override
  public void guardarEmpleado(Empleado empleado) {
    System.out.println("El empleado " + empleado.getNombre() + " con id " + empleado.getId()
+ " se guardo de forma exitosa.");
  }
}

public interface EmpleadoPort {
```

```
  void guardarEmpleado(EmpleadoRequest request);
}

public class EmpleadoService implements EmpleadoPort{
  IEmpleadoRepository repository = new EmpleadoRepository();

  @Override
  public void guardarEmpleado(EmpleadoRequest request) {
    Empleado empleado = new Empleado(1, request.getNombre());
    repository.guardarEmpleado(empleado);
  }
}

public class Main {
  public static void main(String[] args) {
    EmpleadoService service = new EmpleadoService();
    EmpleadoRequest request = new EmpleadoRequest("Juan");
    service.guardarEmpleado(request);
  }
}
```

Patrón fachada

Oculta la complejidad del sistema creando una interfaz para que un usuario pueda acceder al sistema.

```
public interface Forma {
  void dibujar();
}

public class Circulo implements Forma{
  @Override
  public void dibujar() {
    System.out.println("Dibujar circulo.");
  }
}

public class Cuadrado implements Forma{
  @Override
  public void dibujar() {
    System.out.println("Dibujar cuadrado.");
  }
}

public class Triangulo implements Forma{
  @Override
```

```
  public void dibujar() {
    System.out.println("Dibujar triangulo.");
  }
}

public class FormaFachada {
  private Forma circulo;
  private Forma cuadrado;
  private Forma triangulo;

  public FormaFachada() {
    this.circulo = new Circulo();
    this.cuadrado = new Cuadrado();
    this.triangulo = new Triangulo();
  }

  public void dibujarCirculo() {
    circulo.dibujar();
  }

  public void dibujarCuadrado() {
    cuadrado.dibujar();
  }

  public void dibujarTriangulo() {
    triangulo.dibujar();
  }

}

public class Main {
  public static void main(String[] args) {
    FormaFachada fachada = new FormaFachada();
    fachada.dibujarCirculo();
    fachada.dibujarCuadrado();
    fachada.dibujarTriangulo();
  }
}
```

Patrón Estrategia

En este patrón el comportamiento de una clase puede ser cambiado en tiempo de ejecución, el objeto en cuestión puede elegir el algoritmo que le convenga.

```
public interface Strategy {
```

```
    int operacion(int num1, int num2);
}

public class Suma implements Strategy{
    @Override
    public int operacion(int num1, int num2) {
        return num1 + num2;
    }
}

public class Resta implements Strategy{
    @Override
    public int operacion(int num1, int num2) {
        return num1 - num2;
    }
}

public class Multiplicacion implements Strategy{
    @Override
    public int operacion(int num1, int num2) {
        return num1*num2;
    }
}

public class Context {
    private Strategy strategy;

    public Context(Strategy strategy) {
        this.strategy = strategy;
    }

    public int ejecutarEstrategia(int num1, int num2) {
        return strategy.operacion(num1, num2);
    }
}

public class Main {
    public static void main(String[] args) {
        Context context = new Context(new Suma());
        System.out.println("Resultado suma: " + context.ejecutarEstrategia(3, 2));
        context = new Context(new Resta());
        System.out.println("Resultado resta: " + context.ejecutarEstrategia(3, 2));
        context = new Context(new Multiplicacion());
        System.out.println("Resultado multiplicacion: " + context.ejecutarEstrategia(3, 2));
    }
}
```