PSEUDOCÓDIGO PATRÓN DE DISEÑO "STRATEGY PATTERN"

Es un patrón de diseño de comportamiento el cual permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables

```
class EjemploStrategyPattern{
   public static void main(String[] args) {
      Context context;
       context = new Context(new primerestrategia());
       context.execute();
       context = new Context(new segundaestrategia());
       context.execute();
       context = new Context(new terceraestrategia());
       context.execute();
   }
}
interface estrategia {
  void execute();
}
class primerestrategia implements estrategia {
   public void execute() {
       System.out.println("Called primerestrategia.execute()");
   }
}
class segundaestrategia implements estrategia {
  public void execute() {
       System.out.println("Called segundaestrategia.execute()");
   }
}
class terceraestrategia implements estrategia {
```

```
    public void execute() {
        System.out.println("Called terceresstrategia.execute()");
    }
}
class Context {
    estrategia Estrategia;
    public Context(estrategia Estrategia) {
        this.Estrategia = Estrategia;
    }
   public void execute() {
        this.Estrategia.execute();
    }
 }
```

PSEUDOCÓDIGO PATRÓN DE DISEÑO "OBSERVER PATTERN"

Es un patrón de diseño de comportamiento el cual nos permite definir un mecanismo el cual nos sirva para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.

```
import java.util.Observable;
import java.util.Scanner;
class EventSource extends Observable implements Runnable {
    public void run() {
        while (true) {
            String response = new Scanner(System.in).next();
            setChanged();
            notifyObservers(response);
        }
    }
}
```

```
import java.util.Observable;
import java.util.Observer;

public class MyApp {
   public static void main(String[] args) {
      System.out.println("Enter Text: ");
      EventSource eventSource = new EventSource();
      eventSource.addObserver(new Observer() {
         public void update(Observable obj, Object arg) {
            System.out.println("Received response: " + arg);
         }
      });
     new Thread(eventSource).start();
   }
}
```