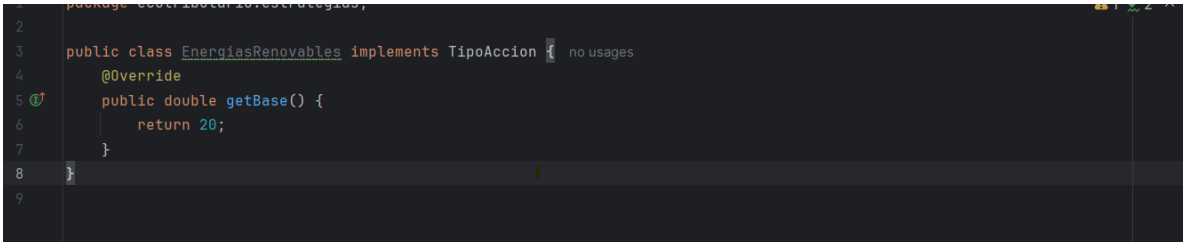
Principios

SOLID

S : SRP (responsabilidad Única)

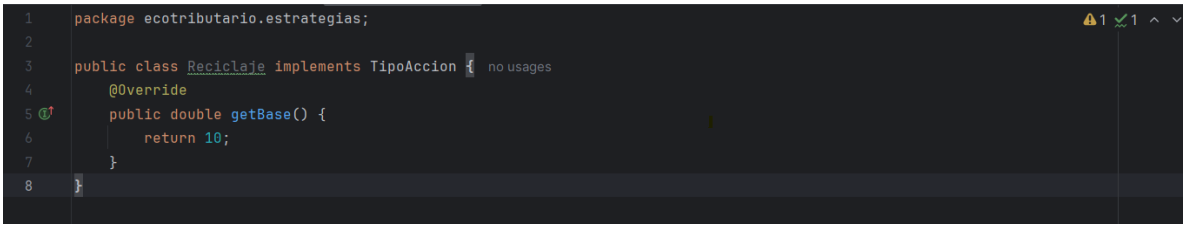
Cada clase tiene una única responsabilidad:

- ***Las clases como reciclaje , energiaRenovables, o las que desciben el tipo de actividad , encapsulan lógica de calculo***
  1.



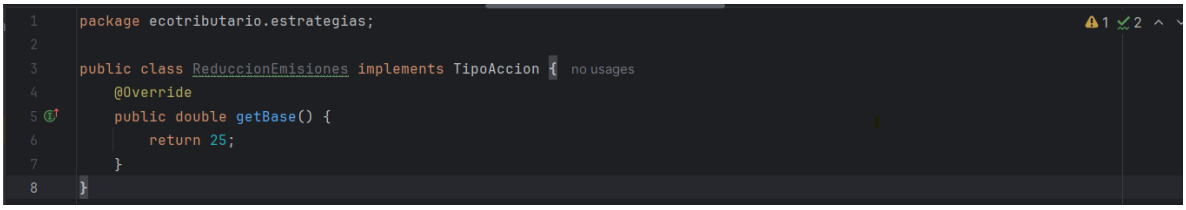
```
public class EnergiasRenovables implements TipoAccion {  no usages
    @Override
    public double getBase() {
        return 20;
    }
}
```

2.



```
package ecotributario.estrategias;

public class Reciclaje implements TipoAccion {  no usages
    @Override
    public double getBase() {
        return 10;
    }
}
```

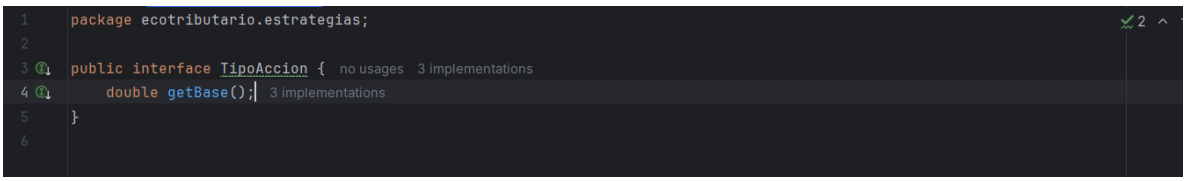
3.



```
package ecotributario.estrategias;

public class ReduccionEmisiones implements TipoAccion {  no usages
    @Override
    public double getBase() {
        return 25;
    }
}
```

O OCP: Abierto / Cerrado

A partir de la interfaz de Tipo de acción si con el tiempo se crea una nueva acción solo basta que esta nueva clase implemente TipoAccion, y no se requiere modificar las clases existentes



```
package ecotributario.estrategias;

public interface TipoAccion {  no usages  3 implementations
    double getBase();  3 implementations
}
```

D DIP Inversiones dependencias :

```java
package ecotributario.servicios;
import ecotributario.estrategias.TipoAccion;

public class incentivoServicios {
    public static double calcularIncentivo(TipoAccion accion, int frecuencia, boolean esEmpresa) {
        double base = accion.getBase();
        double factorFrecuencia = frecuencia >= 5 ? 1.5 : 1.0;
        double factorCategoria = esEmpresa ? 2.0 : 1.0;

        return base * factorFrecuencia * factorCategoria;
    }
}
```

Este principio establece que los módulos de alto nivel no deben depender de módulos de bajo nivel, ambos deben depender de abstracciones. Además, las abstracciones no deben depender de los detalles, los detalles deben depender de las abstracciones

En nuestro código Incentivo Servicios depende de la interfaz Tipo Acción, no de clases en concreto

GRASP

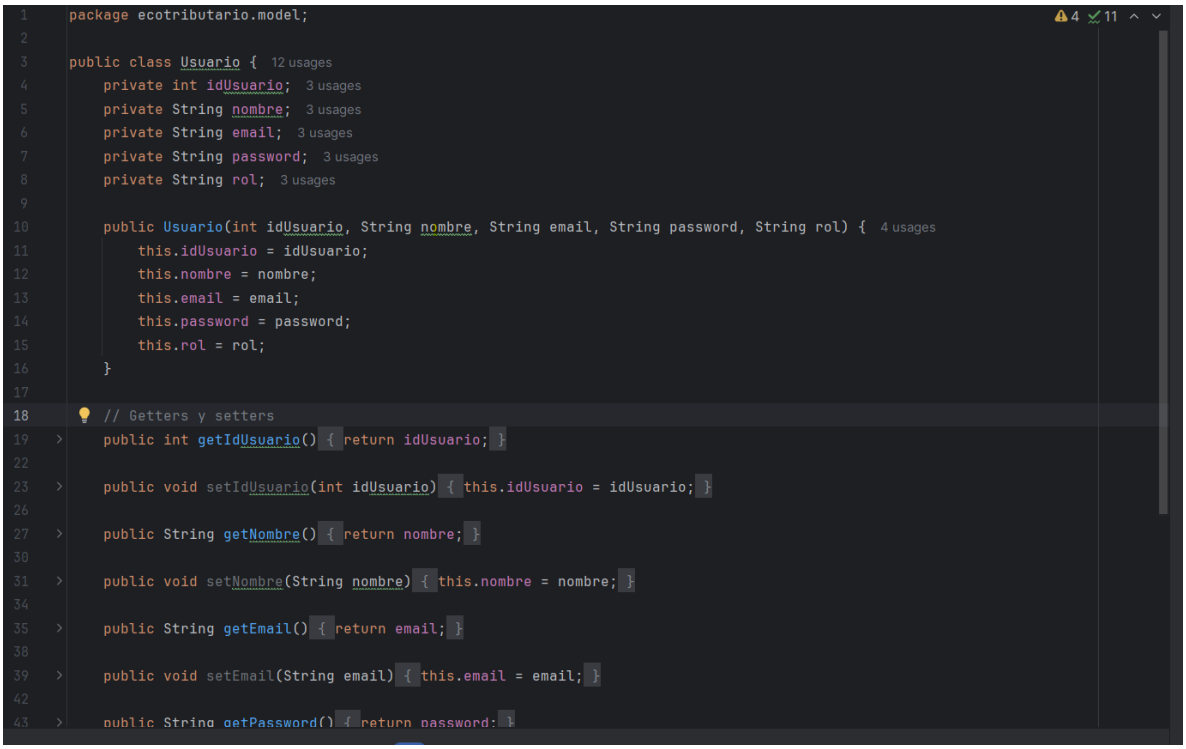
Creador

```java
    // Crear nuevo usuario (registro)
    public boolean crearUsuario(String nombre, String correo, String password, String tipoUsuario) {
        String sql = "INSERT INTO Usuarios (nombre, correo, contraseña, tipo_usuario) VALUES (?, ?, ?, ?)";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString( parameterIndex: 1, nombre);
            stmt.setString( parameterIndex: 2, correo);
            stmt.setString( parameterIndex: 3, HashUtil.encriptar(password));
            stmt.setString( parameterIndex: 4, tipoUsuario);

            return stmt.executeUpdate() > 0;

        } catch (SQLException e) {
            e.printStackTrace();
        }

        return false;
    }
```

La clase UsuarioDAo es responsable de crear instancias de Usuario al obtener datos de la base de datos , en nuestro proyecto la clase UsuarioDao cumple el patron respecto a la clase Usuario ya que:

1. ***La clase UsuarioDAO es la encargada de gestionar operaciones como crear, leer , actualizar, eliminar. Sobre los usuarios que existen en la base de datos***
2. ***UsuarioDao usa instancias de Usuarios , por ejemplo en procesos como la autenticación manipula directamente objetos de tipo Usuario***
3. ***Usuario Dao recibe los datos desde la base de datos y conoce que atrbutos necesita para construir una instancia de Usuario***

Expert



```java
package ecotributario.model;

public class Usuario {  12 usages
    private int idUsuario;  3 usages
    private String nombre;  3 usages
    private String email;  3 usages
    private String password;  3 usages
    private String rol;  3 usages

    public Usuario(int idUsuario, String nombre, String email, String password, String rol) {  4 usages
        this.idUsuario = idUsuario;
        this.nombre = nombre;
        this.email = email;
        this.password = password;
        this.rol = rol;
    }

    // Getters y setters
    public int getIdUsuario() { return idUsuario; }

    public void setIdUsuario(int idUsuario) { this.idUsuario = idUsuario; }

    public String getNombre() { return nombre; }

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

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }
```

Según el patron experto una responsabilidad debe asignarse a la clase que tiene la mayor cantidad de información necesaria para cumplirla. La clase Usuario representa el modelo de datos de un usuario del sistema, cumple el patron por las siguientes razones:

1. ***La clase encapsula todos los atributos relacionados con la identidad y perfil de un usuario***
2. ***En caso de que en un futuro se implementen más métodos relacionados con el usuario , no va a haber problemas porque no depende de otras clases***

Gof

Patrón DAO (Data Access Object) es un patrón arquitectónico que pertenece a la capa de persistencia para separar la lógica de acceso a los datos de la lógica de negocio en la clase USuarioDAO se cumple por la siguiente razon :

1. ***UsuarioDAO se encarga exclusivamente de interactuar con la base de datos:***

- ***Ejecuta consultas sql como SELECT, INSERT , abre conexiones.***
- ***Otras clases como LoginCOntroller usan a UsuarioDAo sin saber ada de SQL***