Maibeth Sofia Alferez Parrado 160004600

EXPERTO EN INFORMACIÓN

```
public void MostrarFacultadesConProgramas(){  1 usage
    FacultadDao facultadDao = new FacultadDao();
    facultadDao.mostrarFacultadesConProgramas();
}
```

Agregue un metodo para explicar como cada clase es responsable de sus propias operaciones

FABRICACIÓN PURA

```
public class PersonaFabrica {
    public Persona crearPersona(Integer Id, String nombres, String apellidos, String email) {
        return new Persona(Id, nombres, apellidos,email);

    }
}
```

antes

```
public void inscribirPersona(Integer id, String nombres, String apellidos, String email){  no usages
    Persona persona = new Persona();
        persona.setId(id);
        persona.setNombres(nombres);
        persona.setApellidos(apellidos);
        persona.setEmail(email);

        inscribir.inscribirPersona(persona);

}
```

despues

```
public void inscribirPersona(Integer id, String nombres, String apellidos, String email){  no usages

    Persona persona = personaFabrica.crearPersona(id, nombres, apellidos, email);
    inscribir.inscribirPersona(persona);
}
```

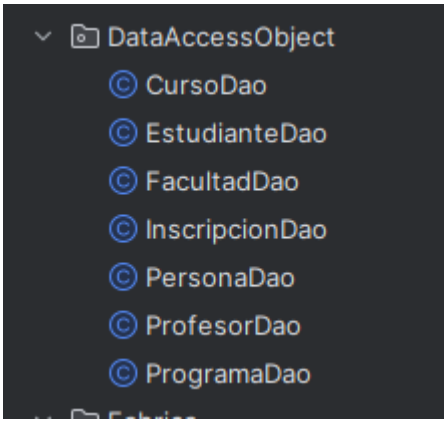
Delegue la responsabilidad de crear instancias del modelo Persona a la clase PersonaFabrica

BAJO ACOPLAMIENTO

```java
package Interfaces;

import java.util.List;

public interface Repositorio<T> {
    boolean agregar(T entidad);
    T consultar(Integer id);
    boolean actualizar(T entidad);
    boolean eliminar(Integer id);
    List<T> listar();
}
```

se hace uso de interfaces para los data Acces Objects.

ALTA COHESIÓNx



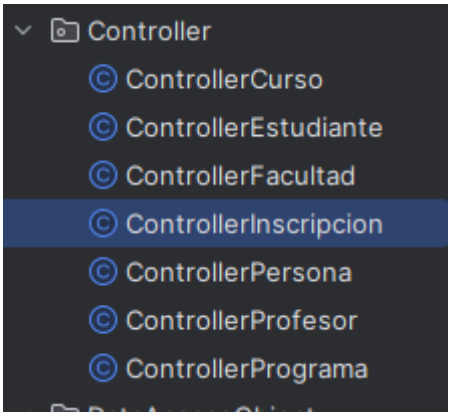
Se separo el conjunto de funcionalidades para cada clase.

```java
public CrudCurso() {...}

private void setupUI() {...}

private void actualizarTabla() {...}

private void crearCurso() {...}

private void eliminarCurso() {...}

private void editarCurso() {...}

private void limpiarCampos() {...}
```

Se divideron las responsabilidades de las vistas, las cuales manejan la entrada y salida asi como la logica del programa.

```java
public class ControllerCurso {

    private final CursoServicio cursoServicio;

    public ControllerCurso() { this.cursoServicio = new CursoServicio(); }

    public boolean crearCurso(Integer id, String nombre, Integer idPrograma, Boolean activo) {...}

    public Curso consultarCurso(Integer id) { return cursoServicio.consultarCurso(id); }

    public boolean actualizarCurso(Integer id, String nombre, Integer idPrograma, Boolean activo) {...}

    public boolean eliminarCurso(Integer id) { return cursoServicio.eliminarCurso(id); }

    public List<Curso> listarCursos() { return cursoServicio.listarCursos(); }
}
```

Los controladores mantienen sus responsabilidades claras.
CONTROLADOR



Delegue la responsabilidad de manejar la logica de la aplicacion y las interacciones del usuario al controlador

POLIMORFISMO

```java
package Interfaces;

import java.util.List;

public interface Servicios {

    String imprimirPosicion(String posicion);
    Integer cantidadActual();
    List<String> imprimirListado();
}
```

defini una interfaz que declare un metodo comun para que este pueda ser imnplementado es mas de una clase

CREADOR

```java
public boolean agregarFacultad(Integer id, String nombre, Integer idDecano) {

    Persona personaDecano = personaDao.consultar(idDecano);

    if(personaDecano != null){

        Facultad facultad = new Facultad();
        facultad.setId(id);
        facultad.setNombre(nombre);
        facultad.setDecano(personaDecano);

        if (facultad == null || facultad.getNombre().isEmpty()) {
            System.err.println("❌ Datos inválidos para agregar facultad.");
            return false;
        }

        //personaBin.agregarPersonaBin(persona);
        return facultadDao.agregar(facultad);
    }

    System.out.println("Persona no existe");
    return false;


}
```

Esta clase contiene la información necesaria para incializar el objeto, además de usarlo con frecuencia y tener una relación fuerte con la clase creada.

INDIRECCION

```java
public class CursoServicio {

    private final CursoDao cursoDao;
    private final ProgramaDao programaDao;

    public CursoServicio() {...}

    public boolean crearCurso(Integer id, String nombre, Integer idPrograma, Boolean activo) {...}

    public Curso consultarCurso(Integer id) {...}

    public boolean actualizarCurso(Integer id, String nombre, Integer idPrograma, Boolean activo) {...}

    public boolean eliminarCurso(Integer id) {...}

    public List<Curso> listarCursos() { return cursoDao.listarCursos(); }
}
```

Se crea una clase intermedia entre el controlador y las clases encargadas de manejar las operaciones con la base de datos, manteniendo asi un bajo acoplamiento.

VARIACIONES PROTEGIDAS

```java
package Interfaces;

import java.util.List;

public interface Repositorio<T> {
    boolean agregar(T entidad);
    T consultar(Integer id);
    boolean actualizar(T entidad);
    boolean eliminar(Integer id);
    List<T> listar();
}
```

encapsulamos las operaciones CRUD en una abstracción, protegiendo a las clases que la implementan de futuros cambios en la lógica de persistencia.