Bajo acoplamiento

1. Retiramos la composición de departamento y país

```java
public class Direccion implements Serializable{
    private int id;
    private Municipio municipio;
    private String calle;
    private String carrera;
    private String coordenada;
    private String descripcion;
```

2. Se hace uso de interfaces

```java
public interface IGestorDatos<T> {

    void crear(T t);
    T leer(int id);
    void listar();
    void actualizar(T t);
    void eliminar(int id);
    void guardarEnArchivoCSV(T t);
    void leerDesdeArchivoCSV();
    void guardarEnArchivoBIN(T t);
    void leerDesdeArchivoBIN();
}
```

Alto cohesión

3. El conjunto de funcionalidad para cada clase se encuentra separado

4. Se dividieron las responsabilidades de las vistas las cuales manejan tanto laentrada/salida como la lógica del programa

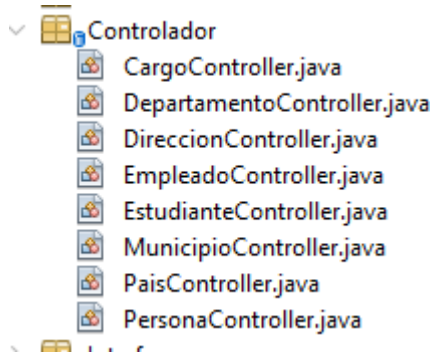
```
public void mostrarMenu() {...7 lines }

private int obtenerOpcionMenu() {...33 lines }

private void procesarOpcion(int opcion) {...36 lines }

private void agregarEmpleado() {...16 lines }

private void consultarEmpleado() {...5 lines }

private void actualizarEmpleado() {...22 lines }

private void eliminarEmpleado() {...5 lines }

private void listarEmpleados() {...3 lines }
```

5. Los controladore mantienen las responsabilidades claras

```
public class PaisController {

    private PaisDAO paisDAO;

    public PaisController() {
        this.paisDAO =  new PaisDAO();
    }

    public void agregarPais(Pais pais) {
        paisDAO.crear(pais);
    }

    public Pais obtenerPais(int id) {
        return paisDAO.leer(id);
    }

    public void listarPaises() {
        paisDAO.listar();
    }

    public void actualizarPais(Pais pais) {
        paisDAO.actualizar(pais);
    }
```

Controlador

6. delegamos la responsabilidad de manejar la lógica de la aplicación y las interacciones del usuario al controlador



Creador

7. Delegamos la responsabilidad de crear instancias de cada modelo a las clasesentidadFabrica.

```
public class PaisFabrica {

    public Pais crearPais(int id, String nombre){
        return new Pais(id, nombre);
    }

}
```

```
    public void agregarPais(int id, String nombre) {
        Pais pais = paisFabrica.crearPais(id, nombre);
        paisDAO.crear(pais);
    }
```

Experto en información

8. Agregamos un método para explicar como cada clase es responsable de suspropias operaciones

```
    public String obtenerDireccionCompleta(int id) {
        Direccion direccion = direccionDAO.leer(id);
        if (direccion != null) {
            return direccion.obtenerDireccionCompleta();
        } else {
            return "Direccion no encontrada.";
        }
    }
```

```java
public String obtenerDireccionCompleta() {
    return "Calle: " + calle + ", Carrera: " + carrera + ", Coordenada: " + coordenada +
           ", Descripción: " + descripcion + ", Municipio: " + municipio.obtenerNombreCompleto();
}
```

Fabricación pura

9. Las validaciones necesarias para la vista procedimos a encapsular sucomportamiento.

```java
public int obtenerIdValido(String mensaje) {
    while (true) {
        try {
            System.out.print(mensaje);
            int id = scanner.nextInt();
            scanner.nextLine();
            if (id >= 0) {
                return id;
            } else {
                System.out.println("Entrada inválida. El ID debe ser un número entero positivo.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Entrada inválida. El ID debe ser un número entero positivo.");
            scanner.next();
        }
    }
}

public String obtenerStringValido(String mensaje, String patron) {
    while (true) {
        System.out.print(mensaje);
        String input = scanner.nextLine();
        if (input.matches(patron)) {
            return input;
        } else {
            System.out.println("Entrada no válida. Intente nuevamente.");
        }
    }
}
```

```java
int id = validador.obtenerIdValido("Ingrese el id de la direccion a actualizar (solo números enteros positivos): ");
int municipioId = validador.obtenerIdValido("Ingrese el nuevo id del municipio (solo números enteros positivos): ");
String calle = validador.obtenerStringValido("Ingrese la nueva calle (solo caracteres alfanuméricos): ", "[a-zA-Z0-9 ]+");
String carrera = validador.obtenerStringValido("Ingrese la nueva carrera (solo caracteres alfanuméricos): ", "[a-zA-Z0-9 ]+");
String coordenada = validador.obtenerStringValido("Ingrese la nueva coordenada (solo números): ", "\\d+");
String descripcion = validador.obtenerStringValido("Ingrese la nueva descripcion (solo caracteres alfabéticos): ", "[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+");
```

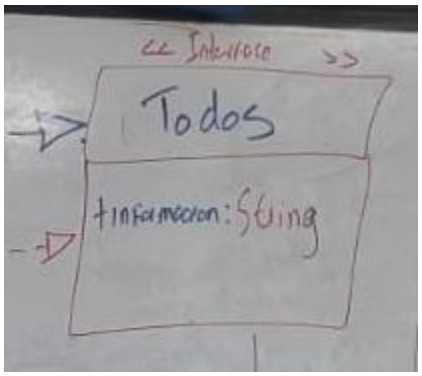
Indirección

10. Se crea una clase intermedia entre el controlador y la vista con tal de separar 2clases manteniendo un bajo acoplamiento y una alta cohesión

```
public class DireccionServicio {

    private final DireccionController direccionController;
    private final MunicipioController municipioController;

    public DireccionServicio() {...4 lines }

    public void agregarDireccion(int id, int municipioId, String calle, String carrera, Str

    public Direccion consultarDireccion(int id) {...3 lines }

    public void actualizarDireccion(int id, int municipioId, String calle, String carrera,
```

Polimorfismo

11. Definimos una interfaz que declare un método común para ser implementadosen más de una clase



Variaciones Protegidas

12. Al definir una interfaz como `IGestorDatos`, se encapsula las variaciones en el manejo de datos y permite que el sistema se adapte a futuros cambios sin afectar a otras partes del código

```java
public interface IGestorDatos<T> {

    void crear(T t);
    T leer(int id);
    void listar();
    void actualizar(T t);
    void eliminar(int id);
    void guardarEnArchivoCSV(T t);
    void leerDesdeArchivoCSV();
    void guardarEnArchivoBIN(T t);
    void leerDesdeArchivoBIN();
}
```

```java
public class CargoDAO implements IGestorDatos<Cargo>{

    private static String ARCHIVO_CSV = "data/cargo.csv";

    @Override
    public void crear(Cargo cargo) {

        String sql = "INSERT INTO Prueba.Cargo (id, nombre) VALUES (?, ?)";

        try (Connection conexion = Conexion.conectar()) {

            conexion.setAutoCommit(false);

            try (PreparedStatement consulta = conexion.prepareStatement(sql)) {

                consulta.setInt(1, cargo.getId());
                consulta.setString(2, cargo.getNombre());

                consulta.executeUpdate();

                conexion.commit();
                System.out.println("Cargo agregado con exito.");
```