

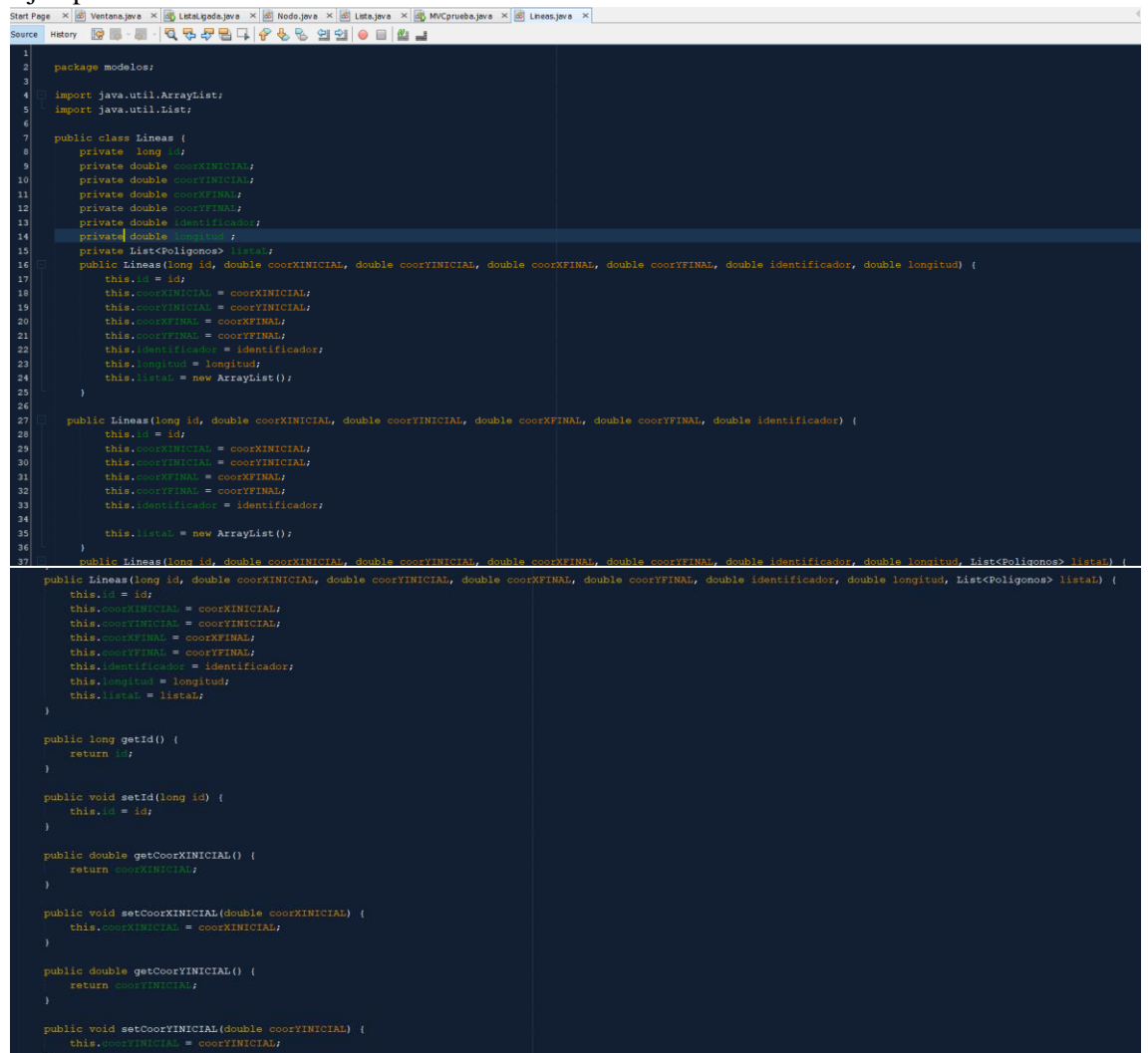
Prueba MVC

Nombre: Andrés Alba

- Modelos

Para mi código que hay en los modelos seguí una estructura en todos; cree 6 clases: Figuras, JefeProyectos, Proyectos, Planos, Polígonos, Lineas. Donde cada una se conecta de distinta forma, pero eso será más adelante, cada una de estas clases disponía de sus variables, y las cree con `private` para que no se puedan llamar desde otras clases, aparte en algunas ocasiones necesitaba crear listas cuando debía enviar datos de una clase a otra, por lo que también cree una lista para las que lo necesitaban para guardar el objeto prácticamente, después inicialice las variables en el constructor y en algunas ocasiones varias veces, pues en algunos métodos necesitaba enviar distintos valores. Después cree los getters y setters y por ultimo el `ToString` y arreglaba el texto por si alguna variable lo necesitaba.

- Ejemplo de un modelo



```
1 package modelos;
2
3
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class Lineas {
8     private long id;
9     private double coorXINICIAL;
10    private double coorYINICIAL;
11    private double coorXFINAL;
12    private double coorYFINAL;
13    private double identificador;
14    private double longitud;
15    private List<Poligonos> listal;
16    public Lineas(long id, double coorXINICIAL, double coorYINICIAL, double coorXFINAL, double coorYFINAL, double identificador, double longitud) {
17        this.id = id;
18        this.coorXINICIAL = coorXINICIAL;
19        this.coorYINICIAL = coorYINICIAL;
20        this.coorXFINAL = coorXFINAL;
21        this.coorYFINAL = coorYFINAL;
22        this.identificador = identificador;
23        this.longitud = longitud;
24        this.listal = new ArrayList();
25    }
26
27    public Lineas(long id, double coorXINICIAL, double coorYINICIAL, double coorXFINAL, double coorYFINAL, double identificador) {
28        this.id = id;
29        this.coorXINICIAL = coorXINICIAL;
30        this.coorYINICIAL = coorYINICIAL;
31        this.coorXFINAL = coorXFINAL;
32        this.coorYFINAL = coorYFINAL;
33        this.identificador = identificador;
34        this.listal = new ArrayList();
35    }
36
37    public Lineas(long id, double coorXINICIAL, double coorYINICIAL, double coorXFINAL, double coorYFINAL, double identificador, double longitud, List<Poligonos> listal) {
38        this.id = id;
39        this.coorXINICIAL = coorXINICIAL;
40        this.coorYINICIAL = coorYINICIAL;
41        this.coorXFINAL = coorXFINAL;
42        this.coorYFINAL = coorYFINAL;
43        this.identificador = identificador;
44        this.longitud = longitud;
45        this.listal = listal;
46    }
47
48    public long getId() {
49        return id;
50    }
51
52    public void setId(long id) {
53        this.id = id;
54    }
55
56    public double getCoorXINICIAL() {
57        return coorXINICIAL;
58    }
59
60    public void setCoorXINICIAL(double coorXINICIAL) {
61        this.coorXINICIAL = coorXINICIAL;
62    }
63
64    public double getCoorYINICIAL() {
65        return coorYINICIAL;
66    }
67
68    public void setCoorYINICIAL(double coorYINICIAL) {
69        this.coorYINICIAL = coorYINICIAL;
70    }
71
72    public double getCoorXFINAL() {
73        return coorXFINAL;
74    }
75
76    public void setCoorXFINAL(double coorXFINAL) {
77        this.coorXFINAL = coorXFINAL;
78    }
79
80    public double getCoorYFINAL() {
81        return coorYFINAL;
82    }
83
84    public void setCoorYFINAL(double coorYFINAL) {
85        this.coorYFINAL = coorYFINAL;
86    }
87
88    public double getIdentificador() {
89        return identificador;
90    }
91
92    public void setIdentificador(double identificador) {
93        this.identificador = identificador;
94    }
95
96    public double getLongitud() {
97        return longitud;
98    }
99
100    public void setLongitud(double longitud) {
101        this.longitud = longitud;
102    }
103
104    public List<Poligonos> getListal() {
105        return listal;
106    }
107
108    public void setListal(List<Poligonos> listal) {
109        this.listal = listal;
110    }
111
112    @Override
113    public String toString() {
114        return "Lineas{" +
115            "id=" + id +
116            ", coorXINICIAL=" + coorXINICIAL +
117            ", coorYINICIAL=" + coorYINICIAL +
118            ", coorXFINAL=" + coorXFINAL +
119            ", coorYFINAL=" + coorYFINAL +
120            ", identificador=" + identificador +
121            ", longitud=" + longitud +
122            ", listal=" + listal +
123            '}';
124    }
125}
```

```

    }

    public void setLongitud(double longitud) {
        this.longitud = longitud;
    }

    public List<Poligonos> getListal() {
        return listal;
    }

    public void setListal(List<Poligonos> listal) {
        this.listal = listal;
    }

    @Override
    public String toString() {
        return "Lineas{" + "id=" + id + ", coorXINICIAL=" + coorXINICIAL + ", coorYINICIAL=" + coorYINICIAL + ", coorXFINAL=" + coorXFINAL + ", coorYFINAL=" + coorYFINAL + ", lon
    }

```

-Controladores

En todos los controladores se necesitaban de 3 cosas: de la importación de la clase de su modelo para poder crear las variables y su instancia, de crear 2 variables, una en donde se guardaran los objetos creados y será tipo List y otra llamada selección, que nos servirá para elegir que objeto queremos manejar desde otra clase, después de tener esto claro deberemos llegar a la creación de los constructores los cuales deberán estar vacíos, e inicializamos las 2 variables, después creamos los getter y setter y el toString() aunque este ultimo no sea tan importante pues es muy poco probable que se use, después de hacer los primeros pasos, toca crear los 4 métodos del CRUD.

- Create: Recibiremos todos los datos que necesitábamos desde la vista la cual veremos luego y luego si es que tiene alguna condición el enunciado, verificamos si se cumplen las condiciones si lo hacen crearemos nuestra instancia de la clase de nuestro modelo y lo guardaremos en la lista, algo muy importante es que nunca se pide el ID pues este se genera mediante un método automático y te lo asigna según corresponda, y este dato se va al modelo. Si es que no se cumplió la condición solo no se crea el objeto así que tampoco se guarda y se envía un false a la vista que se lo mostrara al usuario.
- Read: Es el método que sirve de guía para los otros 2 que faltan, pues prácticamente este método buscar lo que haces es que recibiendo un dato que le entrego el usuario buscarlo en la lista creada donde se guardan los objetos recorriendo 1 objeto por 1 y si en alguno coincide pues lo que envía es el objeto encontrado listo para usarse así que como recibiremos objetos pues el método deberá llevar el nombre de la clase antes del nombre con el que se crea. Si no lo encuentra pues simplemente se regresa un null.
- Update: el actualizar usa el método buscar de arriba para encontrar si es que no se envió información que no existe, así que si el resultado que tiene es diferente a null, recorriendo el objeto ira actualizando los datos usando los métodos Set del modelo para que las variables se establezcan con los nuevos datos recibidos. Si es que había recibido null del buscar pues simplemente regresa un false pues el método es boolean.
- Delete: Usa el método buscar y si es que existe el dato pues se hace algo llama a un método de las propias listas llamado remove y el dato se elimina si lo hace devuelve true y si no false.

Aparte cree el método de imprimir lo que hace es mostrartodos los objetos que existen en la lista del controlador. Y algunas clases necesitaban unos métodos para obtener unos datos así que también estaban en la parte de los constructores

- Ejemplo de un controlador():

```
package controladores;

import modelos.Poligonos;
import java.util.ArrayList;
import java.util.List;
import modelos.Lineas;
import controladores.controlLineas;

public class controlPoligonos {
    private long abc=1;
    private List <Poligonos> datos;
    private Poligonos selecc;
    private controlLineas controlLineas;
    private Lineas L;
    public long id1;

    public controlPoligonos(){
        controlLineas= new controlLineas();
        datos = new ArrayList();

        selecc = null;
    }
    long c=1;
    long a=1;
    long b=0;
    private double perimetro;
    public long generarId(long nroLineas){

        if(datos.size() > 0 && c==nroLineas ){
            c=1;
            return a++;
        }else{
            c++;
            return a;
        }
    }
}
```

```

    }else{
        c++;
        return a;
    }
}

```

```

    }

    public boolean crear(long nroLineas, long linForman, Lineas lineas){

        Poligonos Poligonos = new Poligonos(this.generarId(nroLineas), nroLineas);
        System.out.println(lineas);
        for (int i=0;i<nroLineas;i++) {
            long a=0;
            if (a==linForman){
                lineas=controlLineas.getListasLineas().get(i);
                Poligonos = new Poligonos(this.generarId(nroLineas), nroLineas,
            }

            }
            System.out.println(Poligonos);
            return datos.add(Poligonos);
        }
    }
}

```

```

    public Poligonos buscar(long id) {

        for (Poligonos dato : datos) {
            id1=dato.getId();
            if(id1==id){
                return dato;
            }
        }
    }
}

```

```

        id1=dato.getId();
        if(id1==id){
            return dato;
        }
    }
    return null;
}

//update
public boolean actualizar(long id, long nroLineas, long linForman){
    Poligonos Poligonos = this.buscar(id);
    if(Poligonos != null) {
        int posicion = datos.indexOf(Poligonos);
        Poligonos.setNroLineas(nroLineas);
        Poligonos.setLinForman(linForman);
        datos.set(posicion, Poligonos);
        return true;
    }
    return false;
}
}

```

```

    public boolean eliminar(long id){
        Poligonos Poligonos = this.buscar(id);
        if(Poligonos != null) {
            Poligonos.getLineas().getListasL().remove(Poligonos);
            return datos.remove(Poligonos);
        }
        return false;
    }
}

```

```

    public List<Poligonos> getDatos() {
        return datos;
    }
}

```

```

public void setDatos(List<Poligonos> datos) {
    this.datos = datos;
}

public Poligonos getSelecc() {
    return selecc;
}

public void setSelecc(Poligonos selecc) {
    this.selecc = selecc;
}

public void imprimir(){
    System.out.println("-----"+a+"-----");
    for (Poligonos poligonos : datos) {
        System.out.println(poligonos);
    }
    System.out.println("-----");
}

public double perimetro(long nroLineas,Lineas lineas){
    double sum=0;
    long a=datos.size();
    for (int i=0;i<a;i++ ) {

        sum=sum+datos.get(i).getLineas().getLongitud();
    }

    return sum;
}

@Override
public String toString() {
    return "controlPoligonos{" + "datos=" + datos + ", selecc=" + selecc + '
}

```

-Vistas

Para las vistas lo que se necesita es importar las vistas con las que conectaremos y los controladores que necesitaremos, por si necesitamos los datos de las distintas clases, pues con esto podremos usar el getSelecc() que es seleccionar el objeto que queríamos. Lo que hacemos después es crear todas nuestras variables que nos sirven para usar los métodos de otras clases, y después en el constructor las inicializamos para no tener problemas, si lo hacemos mal nuestro programa no funcionara. Creamos nuestro método llamado menú donde desde ahí llamaremos al resto de métodos de la clase, los cuales recogen los datos que ingresa el usuario y llama a los controladores, todos con los nombres correspondientes del CRUD y también una opción de imprimir y cuando no se escoja ninguna opción se saldrá. Si es que en alguna vista necesita de otra se hace una

confirmación de los datos y que se enlace al nuevo objeto que crearemos. Por ejemplo que para crear un polígono necesitamos mínimo de 3 líneas y que esas 3 líneas existan y si no lo hacen pues no se ejecutara la creación del polígono.

No es necesario crear getters ni setters pues no usaremos ninguna variable de las que tenemos en otras clases directamente de aquí si no de los controladores.

-Vista Principal

Es la primera vista que veremos, nos permite desplazarnos por todas las vistas no tiene mas ciencia que instanciar todas las vistas y llamarlas por medio de 1 menú .

Ejemplo de código de una vista

```
1 package vistas;
2
3 import controladores.controlPoligonos;
4 import modelos.Figuras;
5 import java.util.Scanner;
6 import modelos.Poligonos;
7 import controladores.controlFigura;
8 import modelos.Lineas;
9 import vistas.vistaPoligonos;
10 public class vistaFigura {
11     private controlFigura controlFigura;
12     private controlPoligonos controlPoligonos;
13     private Figuras datos;
14     private vistaPoligonos vistaF;
15     Scanner leer = new Scanner(System.in);
16
17
18     public vistaFigura() {
19         controlFigura=new controlFigura();
20         this.vistaF= new vistaPoligonos();
21         this.controlPoligonos = new controlPoligonos();
22         datos = new Figuras();
23
24     }
25
26
27     public vistaFigura(controlPoligonos controlPoligonos) {
28         this.vistaF=new vistaPoligonos();
29         this.controlPoligonos = controlPoligonos;
30         datos = new Figuras();
31     }
32
33     public void menu() {
34         int opcion = 0;
35         do {
36             System.out.println("\n///////// Gestión de Figuras //////////");
```

```

int opcion = 0;
do {
    System.out.println("\n///////// Gestión de Figuras //////////");
    System.out.println("1. Crear");
    System.out.println("2. Actualizar");
    System.out.println("3. Buscar");
    System.out.println("4. Eliminar");
    System.out.println("5. Listar");
    System.out.println("6.Gestionar Poligonos");
    System.out.println("7. Salir");
    opcion = leer.nextInt();
    switch(opcion){
        case 1: this.crear(); break;
        case 2: this.actualizar(); break;
        case 3: this.buscar(); break;
        case 4: eliminar();break;
        case 5:
            System.out.println("Listado de figuras:");
            controlFigura.imprimir();
            break;
        case 6:vistaP.menu();break;
    }
} while (opcion < 7);
}

public void crear(){
    System.out.println("Ingrese los datos de su figura:");
    System.out.println("Nombre de la figura");
    String nombre = leer.next();
    System.out.println("Color de la figura");
    String color = leer.next();

    long id=0;
    boolean resultado=false;
    System.out.println("--Identificador de su Poligono--");

```

```

    System.out.println("Producto creado: " + resultado);
}

public void eliminar(){
    System.out.println("Eliminar Figura");
    System.out.println("ID: ");
    long id = leer.nextLong();
    boolean resultado = controlFigura.eliminar(id);
    System.out.println("Figura eliminado: " + resultado);
}

public Figuras buscar(){
    System.out.println("Buscar Linea");
    System.out.println("Ingrese ID a buscar");
    long id=leer.nextLong();
    Figuras figura = controlFigura.buscar(id);
    System.out.println(figura);
    return figura;
}

public void actualizar(){
    long id1=0;
    System.out.println("Ingrese los datos de su figura:");
    System.out.println("Nombre de la figura");
    String nombre = leer.next();
    System.out.println("Color de la figura");
    String color = leer.next();

    long id=0;

    System.out.println("--Identificador de su Poligono--");

    boolean resultado=false;

```

```

public void actualizar(){
    long idl=0;
    System.out.println("Ingrese los datos de su figura:");
    System.out.println("Nombre de la figura");
    String nombre = leer.next();
    System.out.println("Color de la figura");
    String color = leer.next();

    long id=0;

    System.out.println("---Identificador de su Poligono---");

    boolean resultado=false;
    Poligonos poli=vistaP.buscar();
    long c=vistaP.abc;
    if (poli!=null){
        controlPoligonos.setSelecc(poli);

        resultado=controlFigura.crear(id,color,nombre,0,vistaP.F,controlPoligonos);
    }else {
        System.out.println("No existe la Poligono");
    }
}

```

- **Consideraciones**
- Hubo un método que no logre hacer, que era de sacar el área de los polígonos, pues es muy difícil sacar el área de los polígonos irregulares pues tienen muchas determinantes que lo complican demasiado, pues pueden ser figuras deformes como no así que no se puede crear un método general para el calculo de polígonos irregulares.
- Ejemplo de funcionamiento
-

```

***** Gestión de vistas *****
1. vista de jefes de Proyecto
2. vista de Proyectos
3. vista de Planos
4. vista de Figuras
5. vista de Poligono
6. vista de lineas

```

```
>>>>>> Gestión de Lineas <<<<<<<
```

1. Crear
2. Actualizar
3. Buscar
4. Eliminar
5. Listar
6. Salir
- 5

Listado de Lineas:

```

Lineas{id=1, coorXINICIAL=10.0, coorYINICIAL=20.0, coorXFINAL=30.0, coorYFINAL=40.0, longitud=28.2842
Lineas{id=2, coorXINICIAL=10.0, coorYINICIAL=3.0, coorXFINAL=4.0, coorYFINAL=5.0, longitud=6.32455532
Lineas{id=3, coorXINICIAL=2.0, coorYINICIAL=0.0, coorXFINAL=4.0, coorYFINAL=6.0, longitud=6.324555320

```


Cuántas líneas forman su polígono:

3

--Ingrese el Id de sus líneas--

--Ingrese el Id de sus líneas--

Buscar Línea

ID:

1

```
controlLineas{listaLineas=[Lineas{id=1, coorXINICIAL=10.0, coorYINICIAL=20.0,
Lineas{id=1, coorXINICIAL=10.0, coorYINICIAL=20.0, coorXFINAL=30.0, coorYFINAL=
Poligonos{id=1, nroLineas=3, linForman=1, lineas=Lineas{id=1, coorXINICIAL=10.
```

Buscar Línea

ID:

2

```
controlLineas{listaLineas=[Lineas{id=1, coorXINICIAL=10.0, coorYINICIAL=20.0,
Lineas{id=2, coorXINICIAL=10.0, coorYINICIAL=3.0, coorXFINAL=4.0, coorYFINAL=5
Poligonos{id=1, nroLineas=3, linForman=2, lineas=Lineas{id=2, coorXINICIAL=10.
```

Buscar Línea

ID:

3

```
controlLineas{listaLineas=[Lineas{id=1, coorXINICIAL=10.0, coorYINICIAL=20.0,
Lineas{id=3, coorXINICIAL=2.0, coorYINICIAL=0.0, coorXFINAL=4.0, coorYFINAL=6.
Poligonos{id=1, nroLineas=3, linForman=3, lineas=Lineas{id=3, coorXINICIAL=2.0
```

Perimetro:40.933381888135415

Poligono creado: true

-----Gestión de Proyectos-----

1. Crear
 2. Buscar
 3. Eliminar
 4. Listar
 5. Actualizar
 6. Gestionar Jefes de Proyecto
 7. Gestionar Planos
 8. Salir
- 4

Listado de Proyectos:

Proyecto{id=1, codigo=12, nombre=PROYECT}

-----Gestión de Proyectos-----

1. Crear
 2. Buscar
 3. Eliminar
 4. Listar
 5. Actualizar
 6. Gestionar Jefes de Proyecto
 7. Gestionar Planos
 8. Salir
- 6

+++Gestión de Jefes de Proyecto+++

Ingrese los siguientes datos:

Nombre de su jefe de proyeccto:

ANDRES

Ingrese el codigo del trabajador

212

Ingrese la dirreccion:

OCT24

Ingrese el telefono

01020304

JefeProyecto{id=1, codigo=212, nombre=ANDRES, direccion=OCT24, telefono=01020304}

Jefe de proyecto creado: true

Ingrese los datos de su figura:

Nombre de la figura

A1

Color de la figura

AZUL

--Identificador de su Poligono--

Ingrese ID deL poligono:

ID:

1

Poligonos{id=1, nroLineas=3, linForman=1, lineas=Lineas{id=1, coorXINICIAL=12

Producto creado: true

///////// Gestión de Figuras //////////

1. Crear

2. Actualizar

3. Buscar

4. Eliminar

5. Listar

6.Gestionar Poligonos

7. Salir

5

Listado de figuras:

Figuras{id=1, nombre=A1, color=AZUL, area=0.0, perimetro=93.87850748990265, poligor