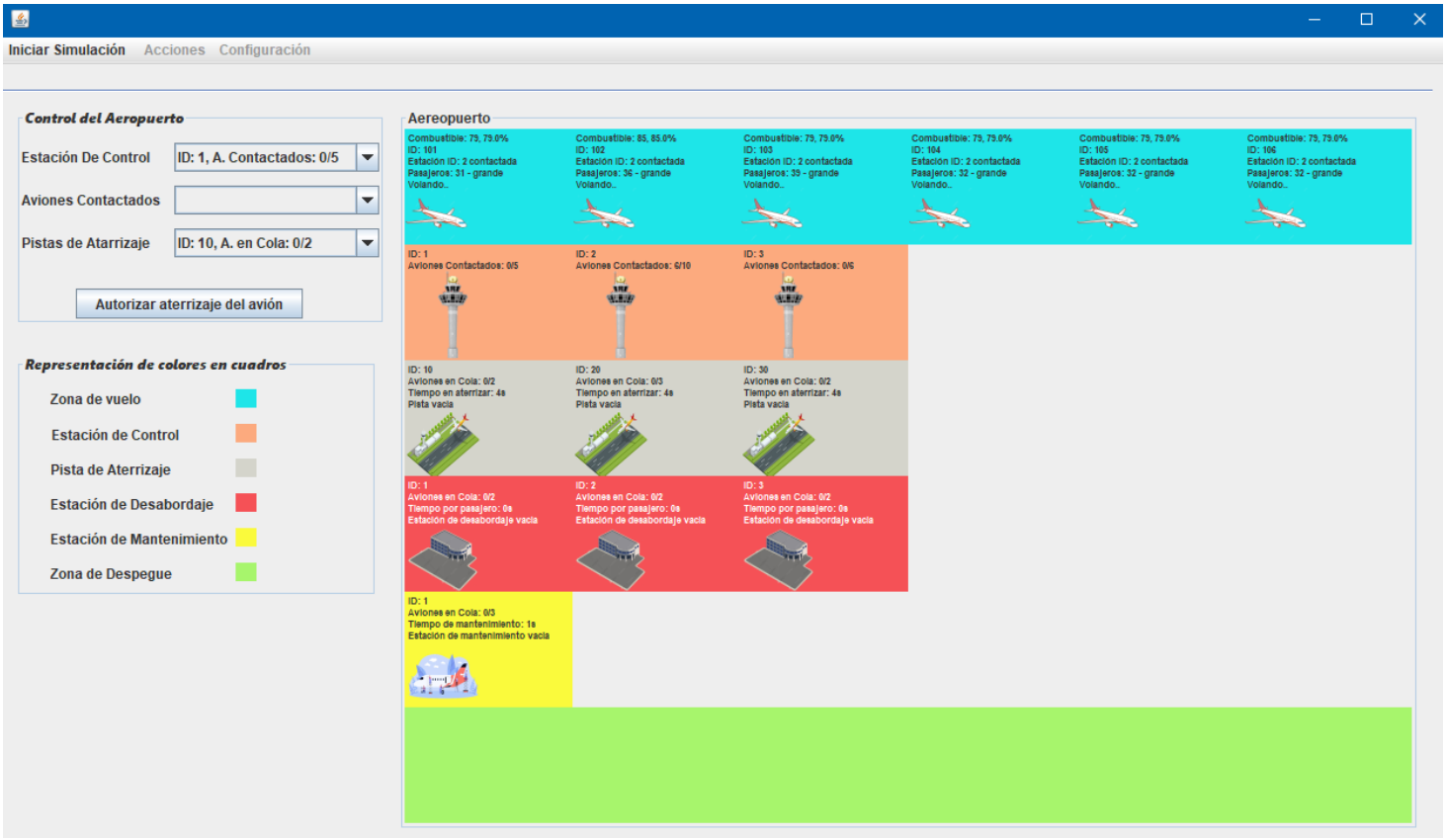
MANUAL TÉCNICO

“Simulación de Aeropuerto”

El siguiente manual está organizado y descrito de acuerdo a todos los algoritmos que se utilizaron para realizar con éxito el sistema de simulación de un aeropuerto, el cual se entrega dividido por paquetes y clases las cuales se describirán a continuación para entender el funcionamiento de las mismas.

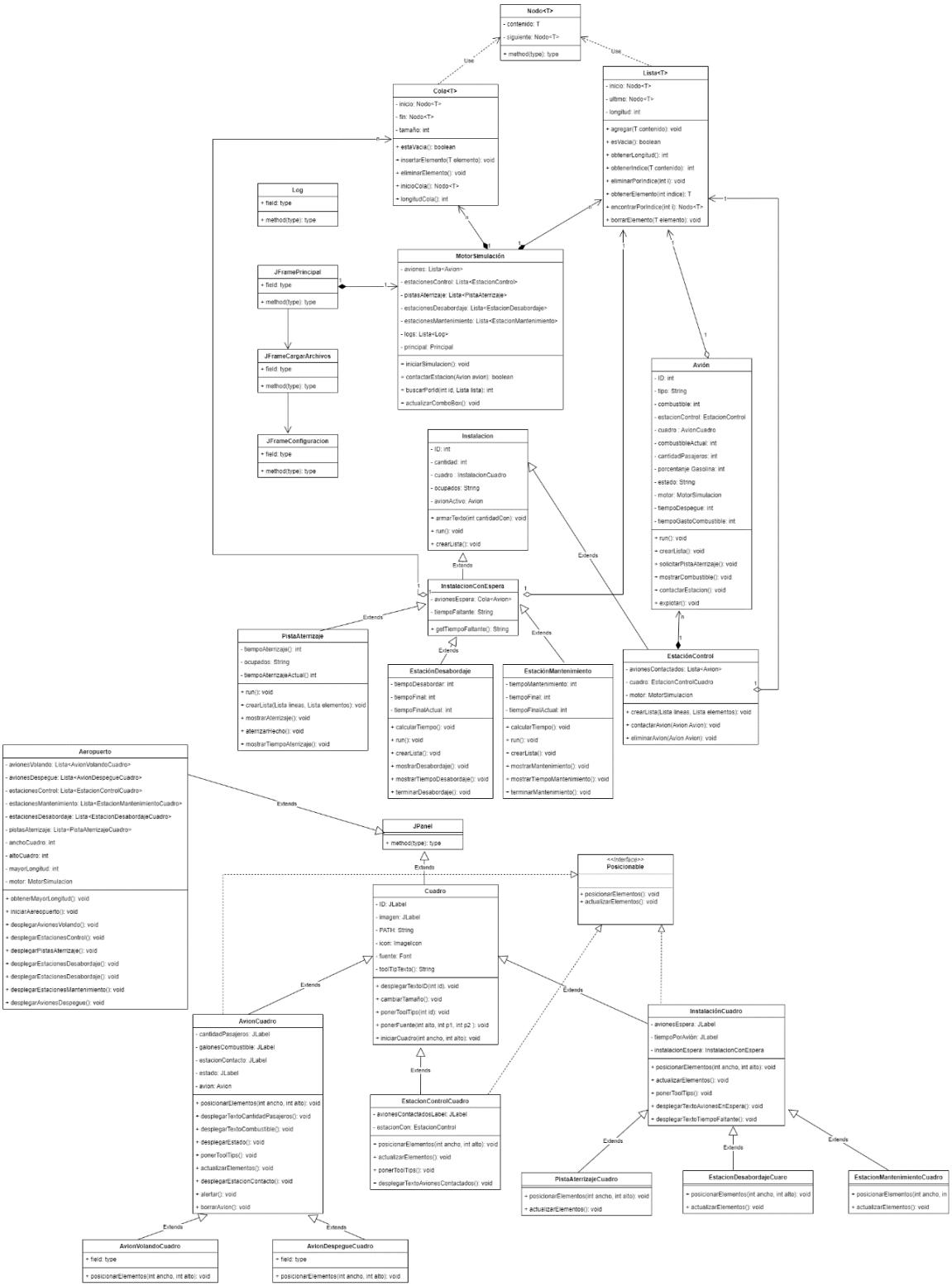
Esta será la ventana principal que se mostrará cuando se inicia aplicación.



Las Herramientas que se utilizaron para el desarrollo del programa fueron:

- Se utilizó el lenguaje de programación java en su versión 15
- El ambiente de desarrollo utilizado fue Apache Netbeans en su versión 12.6
- Para el desarrollo se utilizó maven.
- Para la elaboración del archivo .jar se utilizó maven.

Se utilizo un Diagrama UML de Clases para tener una mejor organización del desarrollo del programa.
(El diagrama completo va dentro de la carpeta de los manules)



Las clases y paquetes que se utilizaron para el desarrollo del programa fueron:

Clases en el paquete backend:

- ❖ Clase Avión: esta clase se encarga de crear un avión con todos sus atributos de acuerdo a su lista y según su tipo asigna pasajeros al avión de forma aleatoria, muestra el estado en el que un avión se encuentra actualmente.

Métodos de esta clase:

- CrearLista: esté método se encarga de ir a leer línea por línea, de ver como viene separada cada línea del archivo, y por cada línea crear un nuevo avión, agregándolos a lista de aviones.
- ContactarEstacion: esté método se encarga de verificar si alguna estación de control esta disponible, para luego contactar al avión según el ID de estación con la que se quiere comunicar para solicitar aterrizaje.
- SolicitarEstaciónControl: este método de se encarga de solicitar periódicamente comunicación con una estación de control.
- Explotar: esté método hace la validación en cuestión del combustible de avión, y si el avión llegara a cero en su combustible antes de que aterrice, se borra de la ventana, pero si llega con al menos 1% de combustible y logra aterrizar, se detiene el gasto de combustible.

```
public class Avion{

public void crearLista(Lista lineas, Lista elementos) {
    for (int i = 0; i < lineas.obtenerLongitud(); i++) {

    public void solicitarEstacionControl() {
```

```
public void contactarEstacion() {
    String idEstacion = JOptionPane.showInputDialog("El avion con id " + ID + " 
    try {
        if (idEstacion != null || idEstacion.equals("")) {
```

- ❖ Clases MotorSimulacion: esta clase de acuerdo a todas las listas crea cada uno de los elementos que se utilizaran en la simulación, y luego de haber obtenido todos los elementos para cada proceso inicia el hilo del avión.

Métodos de esta clase:

- ContactarEstacion: este método devuelve un booleano, es decir verdadero si el avión logro contactar con una estación de control, y false si aún no ha logrado comunicarse.
- EncontrarEstaciónDesabordaje: este método devuelve la estación de desabordaje a la que el avión fue enviado, si la estación de desabordaje aún tiene espacio disponible en su cola, entonces el avión será enviado hacia esa estación, pero si su cola esta llena,

el avión seguirá en zona de aterrizaje, hasta que allá algún espacio disponible, por lo tanto, devolverá null, porque ese avión aún no tiene estación de desbordaje.

- EncontrarEstaciónMantenimiento: este método devuelve la estación de mantenimiento a la que el avión fue enviado, si la estación de mantenimiento aún tiene espacio disponible en su cola, entonces el avión será enviado hacia esa estación, pero si su cola está llena, el avión seguirá en zona de desabordaje, hasta que allá algún espacio disponible, por lo tanto, devolverá null, porque ese avión aún no tiene estación de mantenimiento.
- EstacionDisponible: esté método va desencolando las estaciones, es decir, saca de su cola al avión que entro primero, y así sucesivamente para toda su cola.

```
 */
public class MotorSimulacion {


 public boolean contactarEstacion(Avion avion, int idEstacionNumero) {
     int indice = buscarPorId(idEstacionNumero, estacionesControl);
     try {
         EstacionControl estacion = estacionesControl.obtenerElemento(indice);
         estacion.setMotor(this);


public EstacionDesabordaje encontrarEstacionDesabordaje(PistaAterrizaje pista) {
    for (int i = 0; i < estacionesDesabordaje.obtenerLongitud(); i++) {
        try {
            EstacionDesabordaje estacion = estacionesDesabordaje.obtenerElemento(i);
            if (!estacion.getAvionesEnEspera().esLlena()) return estacion;
```

Clases en el paquete backend.estructuras.lista

- ❖ Clase Cola: esta clase se encarga de encolar cada elemento conforme va llegando a la estación, hace validaciones, para saber si su cola esta vacía o esta llena, y luego va desencolando a cada elemento.

```
public class Cola<T> {

 public void encolarElemento(T elemento)
     if (esLlena()) {
         throw new EstructuraException("

 public T desencolar() throws Estructur
     if (esVacia()) {
         throw new EstructuraException(
```

- ❖ Clase Lista: esta clase se encarga de agregar a cada elemento a lista de su tipo de clase, hace validaciones para saber si la lista aún esta vacía, obtiene la longitud según lo elementos que tiene esa lista, busca y elimina a un índice en específico dentro de la lista, elimina a un elemento de su lista según el índice.

```
public class Lista<T>{

public void agregar(T contenido) {
    if (esVacia()) {
        inicio = new Nodo<>(contenido)
        ultimo = inicio;
```

- Clase Nodo: almacena el contenido, para cada cola y lista según el nodo siguiente que está vacío.

```
public class Nodo<T>{
    private T contenido;
    private Nodo<T> siguiente;
```

- Clase EstructuraException: esta clase maneja las excepciones que se puedan dar en la lista y en la cola, como están vacías o ya llegaron a su límite.

```
public class EstructuraException extends Exception {
```

Clases en el paquete backend.hilos

- Clase HiloPistaAterrizaje: esta clase inicia el hilo de la pista de aterrizaje a través de su método run (), muestra el tiempo restante para que el avión termine de aterrizar y el hilo se detiene hasta que el tiempo de aterrizaje sea >= 0 y por lo tanto termina su aterrizaje.

```
@Override
public void run() {
    mostrarAterrizaje();
}

public void mostrarAterrizaje() {
    avion.borrarCuadro();
    pista.setTiempoActual(PistaAterrizaje.get
    while (pista.getTiempoActual() >= 0) {
```

- Clase HiloEstacionDesabordaje: esta clase inicia el hilo de la estación de desabordaje a través de su método run (), muestra el tiempo restante para que el avión termine de desabordar a todos sus pasajeros y el hilo se detiene hasta que el tiempo de desabordaje sea >= 0 y por lo tanto termina su desabordaje.

```
@Override
public void run() {
    mostrarDesabordaje();
}

public void mostrarDesabordaje() {
    while (estacion.getPasajerosDesabordados() >= 0) {
```

- ❖ Clase HiloEstacionMantenimiento: esta clase inicia el hilo de la estación de mantenimiento a través de su método run (), muestra el tiempo restante para que el avión termine de desabordar a todos sus pasajeros y el hilo se detiene hasta que el tiempo de mantenimiento sea >= 0 y por lo tanto termina su mantenimiento.

```
@Override
public void run() {
    mostrarMantenimiento();
}

    public void mostrarMantenimiento() {
    estacionMantenimiento.calcularTiempo();
    while (estacionMantenimiento.getTiempoFinalActual() >= 0) {
```

- ❖ Clase HiloAvion: esta clase inicia el hilo del avión ya sea que este se encuentre volando o este en zona de despegue a través de su método run (), el hilo empieza o termina hasta que se le de la autorización de despegar o cuando inicie la simulación.

Clases en el paquete backend.instalaciones

- ❖ Clase EstacionControl: esta clase se encarga de crear una estación de control con todos sus atributos de acuerdo a su lista y contacta a un avión, y luego de enviar al avión a una pista de aterrizaje, una vez que el avión es enviado a una pista se elimina de su cola.

Métodos de esta clase

- CrearLista: esté método se encarga de ir a leer línea por línea, y de ver como viene separada cada línea del archivo, y por cada línea crear una nueva estación de control, agregándolas a una lista de estaciones de control.
- ContactarAvion: esté método se encarga de poner en contacto a una estación de control con un avión, según el ID de la estación de control, el avión se comunica con ella para solicitar una pista de aterrizaje
- EliminarAvion: este método se encarga de quitar un avión de su cola, después de haberlo enviado a una pista de aterrizaje.

```
public class EstacionControl extends Instalacion
```

```
@Override
public void crearLista(Lista lineas, Lista elementos) {
    for (int i = 0; i < lineas.obtenerLongitud(); i++) {
        try {
            String[] separador = ((String) lineas.obtenerElemento(i)).split(",");
```

```
public void contactarAvion(Avion avion) t
    if (avionesContactados.obtenerLongitu
```

```
public void eliminarAvion(Avion avion) {
    try {
        avionesContactados.borrarElemento(avion);
```

- Clase PistaAterrizaje: esta clase se encarga de crear una pista de aterrizaje con todos sus atributos de acuerdo a su lista y crear el hilo de esta pista, y luego que el avión termine su aterrizaje, lo desencola y lo borra a ese avión, y sigue el mismo procedimiento con los demás aviones de su cola,

Métodos de esta clase

- CrearLista: esté método se encarga de ir a leer línea por línea, y de ver como viene separada cada línea del archivo, y por cada línea crear una nueva pista de aterrizaje, agregándolas a una lista de pistas de aterrizaje
- AterrizarHecho: este método se encarga de que si el avión ya termino de aterrizar correctamente lo envíe a una estación de desabordaje, solo si esta estación tiene espacio en su cola, pero si la estación de desabordaje esta llena, entonces el avión se queda esperando en la pista de aterrizaje.
- AgregarAColaAvion: este método se encarga de validar si la cola de la pista de aterrizaje está vacía o tiene espacio aun en su cola, agrega a ese avión a su cola, si la cola esta llena ese avión se mantiene a la espera de que esa pista desencola a uno de sus aviones.

```
public class PistaAterrizaje extends InstalacionConEspera {
```

```
public void aterrizarHecho(boolean esperando) {
    if (!esperando) {
```

```
public void agregarAColaAvion(Avion avion, EstacionControl estacion) {
    try {
        if (avionesEnEspera.esVacia() && avionActivo == null) {
```

❖ Clase EstacionDesabordaje: esta clase se encarga de crear una estación de desabordaje con todos sus atributos de acuerdo a su lista y crear el hilo de esta estación, multiplica el tiempo que se le envíe por la cantidad de pasajeros que tiene ese avión y luego que el avión termine su desabordaje, lo desencola y lo borra ese avión, y sigue el mismo procedimiento con los demás aviones de su cola.

Métodos de esta clase

- CrearLista: esté método se encarga de ir a leer línea por línea, y de ver como viene separada cada línea del archivo, y por cada línea crear una nueva estación de desabordaje, agregándolas a una lista de estaciones de desabordaje.
- TerminarDesabordaje: este método se encarga de que si el avión ya termino su desabordaje correctamente lo envíe a una estación de mantenimiento, solo si esta estación tiene espacio en su cola, pero si la estación de mantenimiento está llena, entonces el avión se queda esperando en la estación desabordaje.
- AgregarAColaAvion: este método se encarga de validar si la cola de la estación de desabordaje está vacía o tiene espacio aun en su cola, agrega a ese avión a su cola, si la cola está llena ese avión se mantiene a la espera de que esa pista desencola a uno de sus aviones.

```
public class EstacionDesabordaje extends InstalacionConEspera {

 public void terminarDesabordaje(boolean esperando) {
     if (!esperando) {
         JOptionPane.showMessageDialog(null, "El avion con id:
     }
```

```
 public void agregarAColaAvion(Avion avion) {
     try {
         if (avionesEnEspera.esVacia() && avionActivo == null) {
```

❖ Clase EstacionMantenimiento: esta clase se encarga de crear una estación de mantenimiento con todos sus atributos de acuerdo a su lista y crear el hilo de esta estación, multiplica el tiempo que se le envíe por la cantidad de pasajeros que tiene ese avión y luego que el avión termine su mantenimiento, lo desencola y lo borra ese avión, y sigue el mismo procedimiento con los demás aviones de su cola.

Métodos de esta clase

- CrearLista: esté método se encarga de ir a leer línea por línea, y de ver como viene separada cada línea del archivo, y por cada línea crear una nueva

estación de mantenimiento, agregándolas a una lista de estaciones de mantenimiento.

- TerminarMantenimiento: este método se encarga de que si el avión ya termino su mantenimiento correctamente lo envíe a la zona de despegue, para iniciar nuevamente el recorrido en cada proceso de la simulación.
- AgregarAColaAvion: este método se encarga de validar si la cola de la estación de mantenimiento está vacía o tiene espacio aun en su cola, agrega a ese avión a su cola, si la cola está llena ese avión se mantiene a la espera de que esa pista desencola a uno de sus aviones.

```
public class EstacionMantenimiento extends InstalacionConEspera {

    public void terminarMantenimiento() {
        JOptionPane.showMessageDialog(null, "El avion con id: " + avionActivo.getID() + " a terminado su mantenimiento,
        motor.enviarAvionADespegue(avionActivo);
        try {
```

Clase Instalación e Instalación con Espera, tienes los mismos métodos que les sobrescriben sus clases hijas.

```
public abstract class Instalacion extends Thread {
public abstract class InstalacionConEspera extends Instalacion {
```

Clases en el paquete ui

- ❖ Clase Aeropuerto: esta clase se encarga de imprimir los aviones, pistas de aterrizaje, estaciones de control, desabordaje y mantenimiento según la longitud de la lista de cada uno, despliega cada elemento en un cuadro dentro de un JPanel, obtiene la mayor longitud, es decir la lista que tenga más elementos, es la que dirá el ancho y alto de los cuadros.

Métodos de esta clase:

- IniciarAeropuerto: este método por medio de un GridLayout divide el JPanel en forma de cuadricula, se le asigna un alto y un ancho para cada cuadro y se despliegan los aviones volando, pistas de aterrizaje, estaciones de control, desabordaje y mantenimiento y los aviones en zona de despegue.
- Métodos deplegarAvionesVolando, desplegarEstacionesControl, desplegarPistasAterrizaje, desplegarEstacionesDesabordaje, desplegarEstacionesMantenimiento, deplegarAvionesDespegue: estos métodos muestran los cuadros de cada elemento dentro de la simulación de Aeropuerto, según la cantidad de elementos en la lista.

```
public class Aeropuerto extends JPanel {


   public void iniciarAereopuerto() {
       obtenerMayorLongitud();
       setLayout(new GridLayout(6, mayorLongitud));
       anchoCuadro = Math.round(getWidth() / mayorLongitud);
       altoCuadro = Math.round(getHeight() / 6);
```

```
public void desplegarAvionesVolando() {
    for (int i = 0; i < mayorLongitud; i++) {
        try {
            AvionVolandoCuadro avionVolandoCuadro = new AvionVolandoCuadro(motor.getAviones().obtenerElemento(i));
```

Clases en el paquete ui.cuadro

- Clase Cuadro: esta clase se encarga de desplegar textos de los atributos de cada elemento de las listas, cambiar el tamaño a las imágenes, poner fuentes en los textos, poner toolTips en cada cuadro donde se situé el mousse e inicia el cuadro con su alto y su ancho.

```
public class Cuadro extends JPanel {

  public void desplegarTextoID(int id) {
      ID.setFont(fuente);
      ID.setText("ID: " + id);
  }

  public void cambiarTamaño(){
```

- Clase EstacionControlCuadro: esta clase se encarga de ponerle una imagen al cuadro, a través de un path, posiciona sus elementos respecto al tamaño del cuadro, despliega textos de los atributos de la estación de control, actualiza el cuadro cuando contacta con un avión, y despliega el texto de los aviones con los que ha contactado.

```
public class EstacionControlCuadro extends Cuadro implements Posicionable {


  @Override
  public void posicionarElementos(int ancho, int alto) {
      ponerFuente(alto, 12, 150);
      iniciarCuadro(ancho, alto);

      setBackground(new Color(252, 170, 126));
      desplegarTextoID(estacionCon.getID());
```

```
@Override
public void actualizarElementos() {
    estacionCon.armarTexto(estacionCon.ge
    desplegarTextoAvionesContactados();
```

Clases dentro del paquete ui.cuadro.avion

- Clase AvionCuadro: esta clase se encarga de ponerle una imagen al cuadro, a través de un path, posiciona sus elementos respecto al tamaño del cuadro, despliega textos de los atributos del avión, actualiza el cuadro cuando contacta con una estación de control, y despliega el texto de la estación de control con la que contacto, borra al avión temporalmente cuando este es enviado a una pista de aterrizaje.

```
public class AvionCuadro extends Cuadro implements Posicionable {
```

```
@Override
public void actualizarElementos() {
    galonesCombustible.setText("Combustible: " +
    desplegarEstado();
```

- Clase AvionVolandoCuadro y AvionDespegueCuadro, son clases hijas de la clase CuadroAvion, por lo tanto, usan los mismos métodos de su clase padre.

```
public class AvionVolandoCuadro extends AvionCuadro {

public class AvionDespegueCuadro extends AvionCuadro {
```

Clases en el paquete ui.cuadro.instalacion

- Clase InstalacionCuadro: esta clase se encarga de ponerle una imagen al cuadro, a través de un path, posiciona sus elementos respecto al tamaño del cuadro, despliega textos de los atributos de las estaciones con espera, actualiza el cuadro cuando un avión llega a una pista de aterrizaje, estación de desabordaje o mantenimiento, despliega el texto del avión que está activo en ese proceso y cuando un avión termina correctamente ese proceso.
- Clase PistaAterrizajeCuadro, EstacionDesabordajeCuadro, EstacionMantenimientoCuaro, son hijas de la clase InstalacionCuadro, por lo tanto, usan los mismo métods de su clase padre.

```
public class PistaAterrizajeCuadro extends InstalacionCuadro {

public class EstacionDesbordajeCuadro extends InstalacionCuadro {

public class EstacionMantenimientoCuadro extends InstalacionCuadro implements Posicionable {
```