# GROUND CONTROL

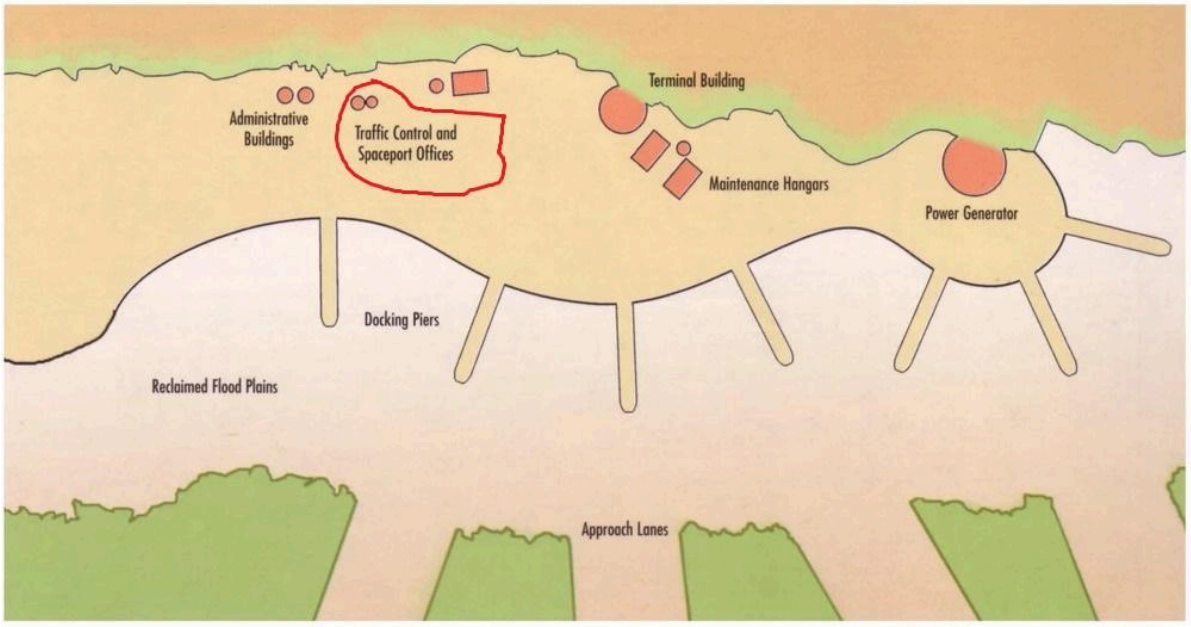
Trabajo realizado por:

- Marta Tirador Gutiérrez
- Víctor García Murillo
- Álvaro Bellón Lanz
- Irene Verdeja Díaz

# CONTENIDOS

# INTRODUCCIÓN: ORIENTACIÓN ECONÓMICA O DE SERVICIO

El grupo Alan Turing S.L.N.E se dedica al desarrollo de aplicaciones de última generación y pioneras en la innovación de la tecnología de software espacial.
Hemos sido contratados por la República Galáctica en el año 32 BBY para desarrollar una aplicación que servirá al edificio de la terminal de control del nuevo espacio-puerto Theed.



Por la venta del producto de software recibiremos 100,000 créditos galácticos.
Por el mantenimiento de dicho software recibiremos 1000 créditos galácticos mensuales.

# ANÁLISIS

Teniendo en cuenta que se prevé que la terminal del espacio-puerto Theed tendrá una concurrencia de naves espaciales tanto militares como de transporte, surge la necesidad de crear clases en herencia para crear objetos de un tipo u otro.

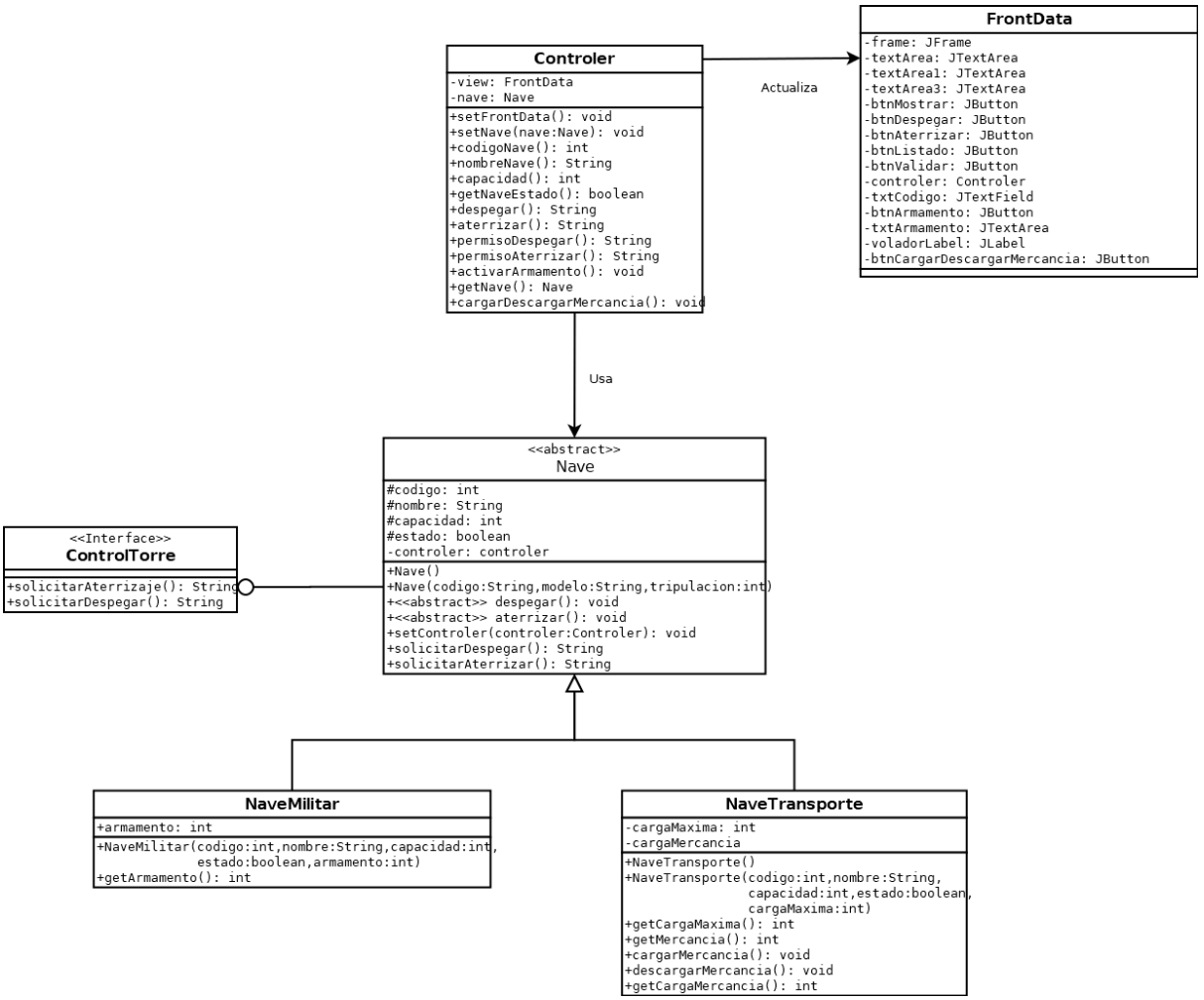
Por otro lado los empleados de la terminal necesitarán una interfaz gráfica de usuario (GUI) donde poder ver, gestionar y controlar el flujo de entradas y salidas de naves del espacio-puerto.

Se requiere una función de escaneo para carga y descarga de mercancía y/o armamento.

Para la GUI se utilizará un patrón de diseño basado en el modelo-vista-controlador.

# DISEÑO

## UML


Controler
-view: FrontData
-nave: Nave
+setFrontData(): void
+setNave(nave:Nave): void
+codigoNave(): int
+nombreNave(): String
+capacidad(): int
+getNaveEstado(): boolean
+despegar(): String
+aterrizar(): String
+permisoDespegar(): String
+permisoAterrizar(): String
+activarArmamento(): void
+getNave(): Nave
+cargarDescargarMercancia(): void
Actualiza
FrontData
-frame: JFrame
-textArea: JTextArea
-textArea1: JTextArea
-textArea3: JTextArea
-btnMostrar: JButton
-btnDespegar: JButton
-btnAterrizar: JButton
-btnListado: JButton
-btnValidar: JButton
-controler: Controler
-txtCodigo: JTextField
-btnArmamento: JButton
-txtArmamento: JTextArea
-voladorLabel: JLabel
-btnCargarDescargarMercancia: JButton
Usa
<<abstract>>
Nave
#codigo: int
#nombre: String
#capacidad: int
#estado: boolean
-controler: controler
+Nave()
+Nave(codigo:String,modelo:String,tripulacion:int)
+<<abstract>> despegar(): void
+<<abstract>> aterrizar(): void
+setControler(controler:Controler): void
+solicitarDespegar(): String
+solicitarAterrizar(): String
<<Interface>>
ControlTorre
+solicitarAterrizaje(): String
+solicitarDespegar(): String
NaveMilitar
+armamento: int
+NaveMilitar(codigo:int,nombre:String,capacidad:int,
estado:boolean,armamento:int)
+getArmamento(): int
NaveTransporte
-cargaMaxima: int
-cargaMercancia
+NaveTransporte()
+NaveTransporte(codigo:int,nombre:String,
capacidad:int,estado:boolean,
cargaMaxima:int)
+getCargaMaxima(): int
+getMercancia(): int
+cargarMercancia(): void
+descargarMercancia(): void
+getCargaMercancia(): int

# AMPLIACIONES

Las ampliaciones propuestas por el Grupo Alan Turing S.L.N.E consisten en una extensión de la aplicación “Ground Control” para que sea posible el control de contrabando y defensa del hangar.

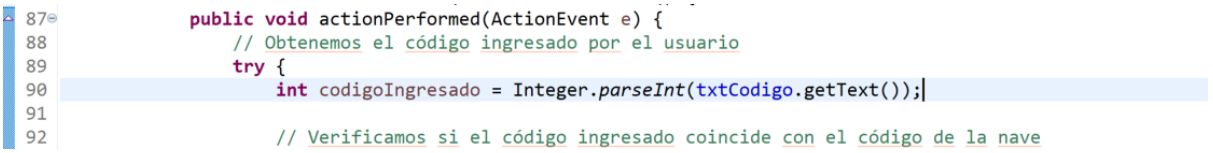
El escaneo se plantea como una barrera a la entrada del hangar que transmitirá a la torre de control la información referente a carga ilegal que pueda transportar la nave.

Para ello, en la GUI se implementaría una opción de escaneo que dirigiría la pantalla a una nueva donde revisar posibles puntos de carga ilegales.

# CONCEPTOS TEÓRICO-PRÁCTICOS

**CASTING**

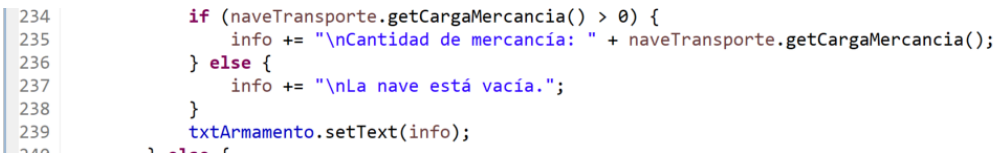
Casting explícito



```
87          public void actionPerformed(ActionEvent e) {
88              // Obtenemos el código ingresado por el usuario
89              try {
90                  int codigoIngresado = Integer.parseInt(txtCodigo.getText());
91
92                  // Verificamos si el código ingresado coincide con el código de la nave
```

**CARACTERES DE ESCAPE**

Salto de línea



```
234             if (naveTransporte.getCargaMercancia() > 0) {
235                 info += "\nCantidad de mercancía: " + naveTransporte.getCargaMercancia();
236             } else {
237                 info += "\nLa nave está vacía.";
238             }
239             txtArmamento.setText(info);
```

## SOBRECARGA DE MÉTODOS

```
10
11    public Nave() {
12          this.codigo=0;
13          this.nombre="";
14          this.capacidad = 0;
15          this.estado=true;
16    }
17    public Nave(int codigo, String nombre,int capacidad, boolean estado) {
18          this.codigo=codigo;
19          this.nombre=nombre;
20          this.capacidad = capacidad;
21          this.estado=estado;
22    }
23
```

## REDEFINICIÓN DE MÉTODOS

```
20    @Override
21    public String despegar() {
22        return "La nave militar "+super.nombre+" está despegando.";
23
24    }
25
26    @Override
27    public String aterrizar() {
28        return "La nave militar "+super.nombre+" está aterrizando.";
29    }
30
31    @Override
32    public String solicitarAterrizar() {
33        String mensaje="Nave militar "+super.nombre+ " solicitando permiso para aterrizar...";
34        return mensaje;
35
36    }
37
38    @Override
39    public String solicitarDespegar() {
40       String mensaje="Nave militar "+super.nombre+" solicitando permiso para despegar...";
41        return mensaje;
42    }
43    }
44
```

## HERENCIA

```
4  public class NaveMilitar extends Nave{
5      private int armamento;
6
```

## HERENCIA Y CONSTRUCTORES (VACÍO Y PARAMETRIZADO)

```
4  public class NaveMilitar extends Nave{
5      private int armamento;
6
7      public NaveMilitar() {
8          super();
9          this.armamento=0;
10     }
11     public NaveMilitar(int codigo, String nombre,int capacidad, boolean estado, int armamento) {
12         super(codigo, nombre,capacidad, estado);
13         this.armamento=armamento;
14     }
15
```

## CLASES ABSTRACTAS

```java
public abstract class Nave implements ControlTorre{
    protected int codigo;
    protected String nombre;
    protected int capacidad;
    protected boolean estado;
    private  Controler controler;


    public Nave() {
          this.codigo=0;
          this.nombre="";
          this.capacidad = 0;
          this.estado=true;
    }
    public Nave(int codigo, String nombre,int capacidad, boolean estado) {
          this.codigo=codigo;
          this.nombre=nombre;
          this.capacidad = capacidad;
          this.estado=estado;
    }

    public abstract String despegar();
    public abstract String aterrizar();


```

## POLIMORFISMO

```java
        Nave Nave = new NaveTransporte(1112,"Transport Shuttle",690, true,1098);

```

## INTERFACES

```java
public interface ControlTorre {
    String solicitarAterrizar();
    String solicitarDespegar();


}

```

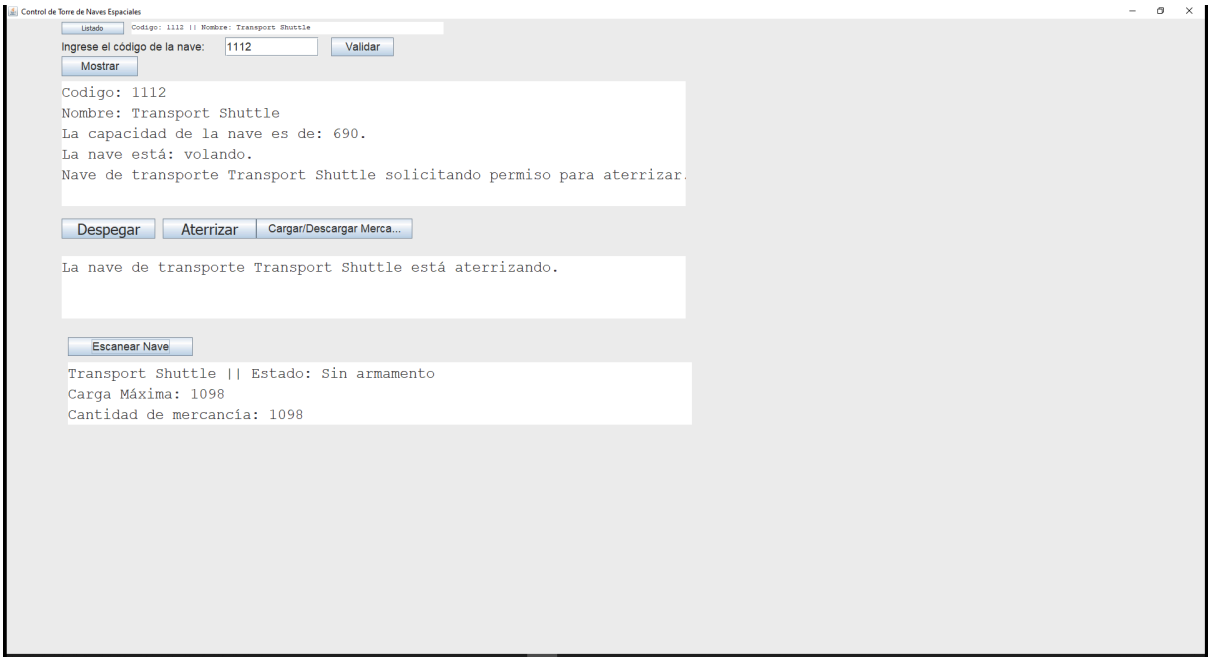
```java
public abstract class Nave implements ControlTorre{
    protected int codigo;
    protected String nombre;
    protected int capacidad;
    protected boolean estado;
    private  Controler controler;

```

## EXCEPCIONES

```
// Método para validar el código
private void validarCodigo() {
    try {
        int codigoIngresado = Integer.parseInt(txtCodigo.getText());
        if (codigoIngresado == controler.codigoNave()) {
            // Si el código es válido, habilita el botón Mostrar
            btnMostrar.setEnabled(true);
        } else {
            // Si el código no es válido, muestra un mensaje de error
            JOptionPane.showMessageDialog(frame, "Código de nave incorrecto", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } catch (NumberFormatException ex) {
        // Si ocurre un error al convertir a entero, muestra un mensaje de error
        JOptionPane.showMessageDialog(frame, "Por favor, ingrese un valor numérico válido en el campo de código", "Er
    }
}
```

## INTERFACES DE USUARIO




## Componentes

```
private JFrame frame;
private JTextArea textArea;
private JTextArea textArea1;
private JTextArea textArea3;
private JButton btnMostrar;
private JButton btnDespegar;
private JButton btnAterrizar;
private JButton btnListado;
private JButton btnValidar; // Botón de validación
private Controler controler;
private JTextField txtCodigo;
private JButton btnArmamento;
private JTextArea txtArmamento;
private JLabel voladorLabel;

private JButton btnCargarDescargarMercancia;
```

## Contenedores

```
    private JFrame frame;
```

```
176         JPanel panel = new JPanel();
177         panel.setLayout(null);
178         panel.setBounds(88, 500, 1500, 600);
179         frame.getContentPane().add(panel);
```

## Contenedores de eventos

```
194         btnArmamento.addActionListener(new ActionListener() {
195             public void actionPerformed(ActionEvent e) {
196                 activarArmamento();
197             }
198         });
```