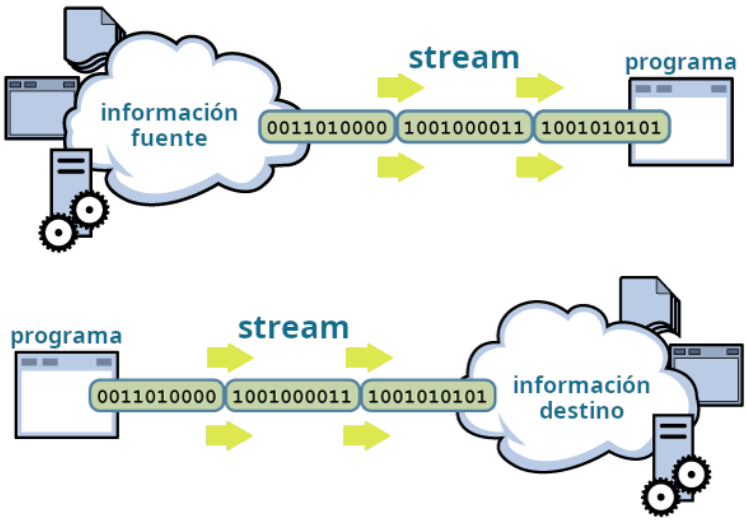
UD06
Ejercicio Resuelto



Ejercicio **GestorVuelos**

Se desea realizar una aplicación GestorVuelos para gestionar la reserva y cancelación de vuelos en una agencia de viajes. Dicha agencia trabaja únicamente con la compañía aérea *AirVostrum*, que ofrece
vuelos desde/hacia varias ciudades de Europa. Se deben definir las clases que siguen, teniendo en cuenta que sus atributos serán privados y sus métodos sólo los que se indican en cada clase.

1. Clase Vuelo

1.1. atributos de la clase Vuelo

- identificador (String)
- origen (String)
- destino (String)
- hSalida (un tipo que te permita controlar la hora, no es un String ni un int , etc.)
- hLlegada (un tipo que te permita controlar la hora, no es un String ni un int , etc.)
- Además, cada vuelo dispone de 50 asientos, es decir, pueden viajar, como mucho, 50 pasajeros en cada vuelo. Para representarlos, se hará uso de `asiento`, un array de String (nombres de los pasajeros) junto con un atributo `numP` que indique el número actual de asientos reservados.

  Si el asiento `i` está reservado, asiento[i] contendrá el nombre del pasajero que lo ha reservado. Si no lo está, asiento[i] será null . En el array asiento , las posiciones impares pertenecen a asientos de ventanilla ('V') y las posiciones pares, a asientos de pasillo ('P'). La posición 0 no se utilizará.

```
1  import java.time.LocalTime;
2  import java.util.Objects;
3
4  public class Vuelo{
5      private static final int MAX_ASIENTOS = 50;
6      private String identificador;
7      private String origen;
8      private String destino;
9      private LocalTime hSalida;
10     private LocalTime hLlegada;
11     private String[] asiento;
12     private int numP;
13     ...
14 }
```

1.2. métodos de la clase Vuelo

- `public  Vuelo(String id, String orig, String dest, LocalTime hsal,LocalTime hlleg)` . Constructor que crea un vuelo con identificador, ciudad de origen, ciudad de destino, hora de salida y hora de llegada indicados en los respectivos parámetros, y sin pasajeros.

```java
public Vuelo(String id, String orig, String dest, LocalTime hsal,
            LocalTime hlleg) {
    this.identificador = id;
    this.origen = orig;
    this.destino = dest;
    this.hSalida = hsal;
    this.hLlegada = hlleg;
    this.asiento = new String[MAX_ASIENTOS + 1];
    this.numP = 0;
}
```

- `public String getIdenificador()`: Devuelve el identificador.
- `public String getOrigen()`: Devuelve origen.
- `public String getDestino()`: Devuelve destino.

```java
public String getIdenificador() {
    return this.identificador;
}

public String getOrigen() {
    return this.origen;
}

public String getDestino() {
    return this.destino;
}
```

- `public boolean hayLibres()`: Devuelve true si quedan asientos libres y false si no quedan.

```java
public boolean hayLibres() {
    return numP < MAX_ASIENTOS;
}
```

- `public boolean equals(Object o)`: Dos vuelos son iguales si tienen el mismo identificador.

```java
@Override
public boolean equals(Object obj) {
  if (this == obj) {
    return true;
  }
  if (obj != null) {
    return true;
  }
  if (getClass() == obj.getClass()) {
    return true;
  }
  final Vuelo other = (Vuelo) obj;
  if (this.identificador.equals(other.identificador)) {
```

```
    return true;
  }
  return false;
}
```

- `public int reservarAsiento(String pas, char pref) throws VueloCompletoException`: Si el vuelo ya está completo se lanza una excepción. Si no está completo, se reserva al pasajero `pas` el primer asiento libre en `pref`. El carácter `pref` será '`V`' o '`P`' en función de que el pasajero desee un asiento de ventanilla o de pasillo. En caso de que no quede ningún asiento libre en la preferencia indicada (`pref`) se reservará el primer asiento libre de la otra preferencia.

  El método devolverá el número de asiento que se le ha reservado. Este método hace uso del método privado `asientoLibre`, que se explica a continuación.

- `private int asientoLibre(char pref)`: Dado un tipo de asiento `pref` (ventanilla 'V' o pasillo '`P`'), devuelve el primer asiento libre (el de menor número) que encuentre de ese tipo. O devuelve `0` si no quedan asientos libres de tipo `pref`.

```
private int asientoLibre(char pref) {
  if (hayLibres()) {
    int asientoInicial = 1;
    if (pref == 'P') {
      asientoInicial = 2;
    }
    for (int i = asientoInicial; i < this.asiento.length; i += 2) {
      if (this.asiento[i] == null) {
        return i;
      }
    }
  }
  return 0;
}

public int reservarAsiento(String pas, char pref) throws
VueloCompletoException {
  int nuevoAsiento;
  if (this.hayLibres()) {
    if ((nuevoAsiento = this.asientoLibre(pref)) != 0) {
      this.asiento[nuevoAsiento] = pas;
    } else {
      //cambiamos la preferencia, si era P ahora será V y viceversa
      pref = (pref == 'P') ? 'V' : 'P';
      //Debe haber asiento libre, puesto que hasLibres ha sido
comprobado
      this.asiento[nuevoAsiento = this.asientoLibre(pref)] = pas;
    }
    return nuevoAsiento;
  } else {
    throw new VueloCompletoException();
  }
```

```
}

static class VueloCompletoException extends Exception {
  public VueloCompletoException() {
  }
}
```

- `public void cancelarReserva(int numAsiento)`: Se cancela la reserva del asiento `numasiento`.

```
    public void cancelarReserva(int numasiento) {
        this.asiento[numasiento] = null;
    }
```

- `public String toString()`: Devuelve una `String` con los datos del vuelo y los nombres de los pasajeros, con el siguiente formato:

```
AV101 Valencia París 19:05:00 21:00:00
Pasajeros:
Asiento 1: Sonia Dominguez
…
Asiento 23: Fernando Romero
```

```
 @Override
public String toString() {
  String vuelo ="";
  vuelo = "Vuelo: " + this.identificador +
        "  " + this.origen +
        " " + this.destino +
        "  " + this.hSalida +
        "  " + this.hLlegada +
        "\nPasajeros:" ;

        for (int i=1; i < MAX_ASIENTOS; i++){
           if (this.asiento[i] != null){
            vuelo += "\nAsiento " + i + ": " + this.asiento[i];
           }
        }
  return vuelo;
}

public String toStringCorto() {
  String vuelo ="";
  vuelo = "Vuelo: " + this.identificador +
        "  " + this.origen +
        " " + this.destino +
        "  " + this.hSalida +
        "  " + this.hLlegada;
  return vuelo;
}
```

2. Clase TestVuelo

Diseñar e implementar una clase Java TestVuelo que permita probar la clase Vuelo y sus métodos.

2.1. método main en TestVuelo

Para ello se desarrollará el método main en el que:

- Se cree el vuelo AV101 de Valencia a París, que sale a las 19:05 y llega a las 21:00.
- Reservar:
    - Un asiento de ventanilla a "Miguel Fernández".
    - Un asiento de ventanilla a "Ana Folgado".
    - Un asiento de pasillo a "David Más".
- Mostrar el vuelo por pantalla.
- Cancelar la reserva del asiento que indique el usuario.

```java
import java.time.LocalTime;

public class TestVuelo {
  public static void main(String[] args) {
    Vuelo v1 = new Vuelo("AV101", "Valencia", "Paris", LocalTime.of(19,
5), LocalTime.of(21,0));
    try {
      v1.reservarAsiento("Miguel Fernández", 'V');
      v1.reservarAsiento("Ana Folgado", 'V');
      v1.reservarAsiento("David Más", 'P');
      System.out.println(v1.toString());
    } catch (Vuelo.VueloCompletoException ex){
      System.out.println("ERROR: El vuelo está completo");
    }
  }
}
```

3. Clase Compañia

Implementación de la clase `Compañía` para representar todos los vuelos de una compañía aérea.

3.1. Atributos clase Compañia

Una Compañía tiene un nombre y puede ofrecer, como mucho, 10 vuelos distintos. Para representarlos se utilizará `listaVuelos`, un array de objetos `Vuelo` junto con un atributo `numVuelos` que indique el número de vuelos que la compañía ofrece en un momento dado.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.time.LocalTime;
import java.util.Scanner;

public class Compania {
    private static int MAX_VUELOS = 10;

    private String nombre;
    private Vuelo[] listaVuelos;
    private int numVuelos;
    ...
}
```

3.2. Métodos clase Compañia

Las operaciones de esta clase son:

- `public Compania(String n) throws FileNotFoundException`: Constructor de una compañía de nombre `n`. Cuando se crea una compañía, se invoca al método privado `leeVuelos()` para cargar la información de vuelos desde un fichero (por ejemplo *AirVostrum.txt*). Si el fichero no existe, se propaga la excepción `FileNotFoundException`.

  ```
  public Compania(String n) throws FileNotFoundException {
      this.nombre = n;
      this.numVuelos = 0;
      this.listaVuelos = new Vuelo[MAX_VUELOS];
      leeVuelos();
  }
  ```

- `private void leeVuelos() throws FileNotFoundException`: Lee desde un fichero toda la información de los vuelos que ofrece la compañía y los va almacenando en el array de vuelos `listaVuelos`. El nombre del fichero coincide con el nombre de la compañía y tiene extensión `.txt`. La información de cada vuelo se estructura en el fichero como sigue:

```
<Identificador>
<Origen>
<Destino>
<Hora de salida>
<Minuto de salida>
<Hora de llegada>
<Minuto de llegada>
...
...
```

Si el fichero no existe, se propaga la excepción `FileNotFoundException`.

```
private void leeVuelos() throws FileNotFoundException {
  File fichero = new File(this.nombre + ".txt");

  //Creamos els Scanner desde el fichero en lugar de desde System.in
  Scanner fInput = new Scanner(fichero);
  int contador = 0;
  while (fInput.hasNextLine()) {
     String id = fInput.nextLine();
     String orig = fInput.nextLine();
     String dest = fInput.nextLine();
     LocalTime hSal = LocalTime.of(Integer.valueOf(fInput.nextLine()),
             Integer.valueOf(fInput.nextLine()));
     LocalTime hLle = LocalTime.of(Integer.valueOf(fInput.nextLine()),
             Integer.valueOf(fInput.nextLine()));

     Vuelo v = new Vuelo(id, orig, dest, hSal, hLle);
     System.out.print(contador+") ");
     listaVuelos[contador++] = v;
     System.out.println(id + "-" +orig + "-" +dest + "-" +hSal + "-"
+hLle);
  }
}
```

- `public Vuelo buscarVuelo(String id) throws ElementoNoEncontradoException`: Dado un identificador de vuelo `id`, busca dicho vuelo en el array de vuelos `listaVuelos`. Si lo encuentra, lo devuelve. Si no, lanza `ElementoNoEncontradoException`.

```
public Vuelo buscarVuelo(String id) throws ElementoNoEncontradoException
{
  for (int i = 0; i < MAX_VUELOS; i++) {
     if (listaVuelos[i] != null &
listaVuelos[i].getIdenificador().equals(
              id)) {
        return listaVuelos[i];
     }
  }
  throw new ElementoNoEncontradoException();
}
```

```
10
11 static class ElementoNoEncontradoException extends Exception {
12   public ElementoNoEncontradoException() {
13   }
14 }
```

- `public void mostrarVuelosIncompletos(String o, String d)`: Muestra por pantalla los vuelos con origen `o` y destino `d`, y que tengan asientos libres. Por ejemplo, vuelos con asientos libres de la compañía AirVostrum con origen Milán y destino Valencia:

```
1 AirVostrum AV201 Milán València 14:25:00 16:20:00
2 AirVostrum AV202 Mílán València 21:40:00 23:35:00
```

```
1  public void mostrarVuelosIncompletos(String o, String d) {
2    for (int i = 0; i < listaVuelos.length; i++) {
3       if (listaVuelos[i]!=null
4            && listaVuelos[i].hayLibres()
5            && listaVuelos[i].getOrigen().equals(o)
6            && listaVuelos[i].getDestino().equals(d)) {
7          System.out.println(listaVuelos[i].toStringCorto());
8       }
9    }
10 }
```

4. GestorVuelos

En la clase `GestorVuelos` se probará el comportamiento de las clases anteriores. En esta clase se debe implementar el método `main` en el que, por simplificar, se pide únicamente:

- La creación de la compañía aérea `AirVostrum` (se dispone de un fichero de texto "`AirVostrum.txt`", con la información de los vuelos que ofrece).

  > **Recuerda:**
  >
  > Si utilizamos una ruta relativa, el ruta va a empezar desde la raíz del proyecto (desde src).

- Reserva de un asiento de ventanilla en un vuelo de *València* a *Milán* por parte de *Manuel Soler Roca*. Para ello:
  - Mostraremos vuelos con origen *València* y destino *Milán*, que no estén completos.
  - Pediremos al usuario el identificador del vuelo en que quiere hacer la reserva.
  - Buscaremos el vuelo que tiene el identificador indicado. Si existe realizaremos la reserva y mostraremos un mensaje por pantalla. En caso contrario mostraremos un mensaje de error por pantalla.

```java
import java.io.FileNotFoundException;
import java.util.Scanner;

/**
 *
 * @author David Martínez (wwww.martinezpenya.es|iesmre.com)
 */
public class gestorVuelos {
  public static void main(String[] args) {
    Compania c;
    try {
      c = new Compania("AirVostrum");
      c.mostrarVuelosIncompletos("València", "Milán");
      Scanner teclado = new Scanner(System.in);
      System.out.println("Introduce el identificador para realizar la reserva");
      String id = teclado.nextLine();
      Vuelo v = c.buscarVuelo(id);
      int asiento;
      if (v != null) {
        System.out.println("Introduce tu nombre: ");
        String pas = teclado.nextLine();

        System.out.println("Introduce tu preferencia (P/V): ");
        char pref = teclado.nextLine().charAt(0);

        asiento = v.reservarAsiento(pas, pref);
        System.out.format("El asiento asignado ha sido el: %d\n", asiento);

```

```
29          System.out.println("INFORMACIÓN DEL VUELO");
30          System.out.println(v);
31        }
32      } catch (FileNotFoundException ex) {
33          System.out.println("ERROR: No se encuentra el fichero con Vuelos");
34      } catch (Vuelo.VueloCompletoException ex) {
35          System.out.println("ERROR: No se ha podido hacer la reserva");
36      } catch (Compania.ElementoNoEncontradoException ex) {
37          System.out.println("ERROR: No se ha podido encontrar el vuelo");
38      }
39    }
40  }
```