Viajante
- int numero - string nombre - string tipo - int millas
Constructores + Viajante (int elNumero) + Viajante (int elNumero, string elNombre, string elTipo, int lasMillas) Modificadores + setNombre (string elNombre) + setTipo(string elTipo) + setMillas(int lasMillas) Observadores + getCodigo(): int + getNombre(): string + getTipo(): string + getMillas(): int + toString(): string + equals(Viajante otroViajante): boolean Propias del tipo + aumentarTipo():boolean

```
public Viajante (int elNumero)
{
	numero=elNumero;
	tipo="";
	// o podría ser también tipo="Básico" ya que es el primer tipo
	nombre="";
	millas=0;
}

public Viajante (int elNumero, string elNombre, string elTipo, int
lasMillas)
{
	numero=elNumero;
	tipo=elTipo;
	nombre=elNombre;
	millas=lasMillas;
}
```

```
public string getNombre()
{
      return nombre;
}
public int getCodigo()
{
      return codigo;
}

public int getMillas()
{
      return millas;
}

public int getTipo()
{
      return tipo;
}

public setNombre(string elNombre)
{
      nombre=elNombre;
}

public setMillas(int lasMillas)
{
      millas=lasMillas;
}

public setTipo(string elTipo)
{
      tipo=elTipo;
}

public string toString()
{
      return "Codigo: " + codigo + " Nombre: "+ nombre + " Tipo:" + tipo
+ " Millas:" + millas;
}

public boolean equals(Viajante otroViajante)
{
      return codigo==otroViajante.getCodigo();
      // o return this.codigo==otroViajante.codigo;
```

```
    }

    public boolean aumentarTipo()
    {
        // se puede considerar también que si el viajante es “Básico” y
        // tiene 4500 millas disponibles, puede pasar a “SuperViajero”
        boolean cambioOk=false;

        if (tipo.equals("Básico") && millas>=1500)
        {
            this.setTipo ("Gold");  //o sino tipo="Gold"
            this.setMillas(this.getMillas() - 1500);
            // o this.millas -= 1500;
            cambioOk=true;
        }
        else
        {
             if (tipo.equals("Gold") && millas >=3000)
             {
                this.setTipo ("SuperViajero")   ;
                //o sino tipo="SuperViajero";
                this.setMillas(this.getMillas() - 3000);
                // o this.millas -= 3000;

                cambioOk=true;
             }
        }
        return cambioOK;
    }
```

Punto 3)

a)

```
Viajante[][] matViajantes=cargarMatriz();
```

(O también)

```
Viajante[][] matViajantes;
matViajantes=cargarMatriz();
```

b)

```
MODULO anotadoEnViaje(Viajante[][] laMatriz, Viajante unViajante) RETORNA LOGICO

ENTERO filas
ENTERO columnas
LOGICO encontrado ←false
ENTERO i, j
i ← 0
j ← 0
filas ← longitud(laMatriz)
columnas ← longitud(laMatriz[0])

MIENTRAS (NOT encontrado AND j < columnas AND laMatriz[i][j] != null) HACER
      i ← 0
      MIENTRAS (NOT encontrado AND i < filas AND laMatriz[i][j] != null) HACER
            encontrado ← laMatriz[i][j].equals(unViajante);
            i ← i + 1
      FIN MIENTRAS
      j ← j +1
FIN MIENTRAS

RETORNA encontrado
FIN MODULO
```

c)

```
public boolean inscribeViajante (Viajante[][] laMatriz, Viajante unViajante,
int unMes)
// unMes es un número del 1 al 12.
// Se podría haber considerado del 0 al 11 o incluso un String.
// Su manipulación cambiaría.
{
      boolean inscripcionOK=false;
      int asientoDisponible;

      if (anotadoenViaje(laMatriz, unViajante))
            inscripcionOK=false;
      else
      {
            asientoDisponible= posicionAsientoDisponible(laMatriz, unMes);
            if (asientoDisponible == -1)
                  inscripcionOK=false;
            else
            {
                  inscripcionOK=true;
                  laMatriz[asientoDisponible][unMes-1]=unViajante;
```

```
            }
      }
      return inscripcionOK;
}
```

```
public int posicionAsientoDisponible (Viajante[][] laMatriz, int unMes)
{
      int asientoDisponible=-1;

      int asientos;
      int i;
      i=0;

      asientos=laMatriz.length;
      // recorre la columna hasta encontrar un asiento libre
      // o hasta que llegue al último de la columna
      while (laMatriz[i][unMes-1] != null && i < asientos)
      {
            i++;
      }

      if (i != asientos)
            asientoDisponible=i;

      return asientoDisponible;
}
```

d)

```
public Viajante[] devolverSuperViajeros (Viajante[][] laMatriz)
// retorna un arreglo sobredimensionado con los viajantes solicitados.
// a partir de la posición cantViajeros, el arreglo está  vacío

// una opción es crear un nuevo arreglo de tamaño cantViajeros y copiar los
// elementos del arreglo sobredimensionado
{
    //obtengo la cantidad máxima de viajantes para crear el arreglo
    int cantidad= laMatriz.length  * laMatriz[0].length;

    // creo el arreglo de viajantes sobredimensionado
    Viajante[] arrViajantes=new Viajante[cantidad];

    Viajante unViajante; //una variable auxiliar
    int filas;
    int columnas;
```

```
    int cantViajeros=0;   // para saber en qué posición del arreglo voy
    int i, j;
    i=0;
    j=0;

    filas= laMatriz.length;
    columnas=laMatriz[0].length

      while (laMatriz[i][j] != null && j < columnas)
      {
            i=0;
            while (laMatriz[i][j] != null && i < filas)
            {
                  unViajante=laMatriz[i][j];
                  if (unViajante.getTipo().equals(”SuperViajero”) &&
                     tiene3Vocales(unViajante.getNombre()))
                  {
                        arrViajantes[cantViajeros]=unViajante;
                        cantViajeros++;
                  }
                  i++;
            }
            j++;
      }
   return arrViajantes;
}
```

```
public boolean tiene3Vocales(String cadena)
{

      int tieneA=0;
      int tieneE=0;
      int tieneI=0;
      int tieneO=0;
      int tieneU=0;

      cadena=cadena.toUpper();
      if (cadena.indexOf('A') != -1)
           tieneA = 1;
      if (cadena.indexOf('E') != -1)
           tieneE = 1;
      if (cadena.indexOf('I') != -1)
           tieneI = 1;
```

```
      if (cadena.indexOf('O') != -1)
           tieneO = 1;
      if (cadena.indexOf('U') != -1)
           tieneU = 1;

      return (tieneA + tieneE + tieneI + tieneO + tieneU ) >=3;
}


O tambien podria ser:
public boolean tiene3Vocales(String cadena)
{

      int tieneA=0;
      int tieneE=0;
      int tieneI=0;
      int tieneO=0;
      int tieneU=0;

      cadena=cadena.toUpper();
      if (cadena.contains(‘A’)) tieneA = 1;
      if (cadena.contains('E')) tieneE = 1;
      if (cadena.contains('I')) tieneI = 1;
      if (cadena.contains('O')) tieneO = 1;
      if (cadena.contains('U')) tieneU = 1;

      return (tieneA + tieneE + tieneI + tieneO + tieneU ) >=3;
}

O tambien podria ser:
public boolean tiene3Vocales(String cadena){

      boolean tieneA = false;
      boolean tieneE = false;
      boolean tieneI = false;
      boolean tieneO = false;
      boolean tieneU = false;
      int vocalesDif = 0;

      cadena=cadena.toUpper();
      for(int i = 0; i < cadena.length; i++){
           tieneA = tieneA || cadena.charAt(i)=‘A’;
           tieneE = tieneE || cadena.charAt(i)=‘E’;
           tieneI = tieneI || cadena.charAt(i)=‘I’;
           tieneO = tieneO || cadena.charAt(i)=‘O’;
           tieneU = tieneU || cadena.charAt(i)=‘U’;
```

```
        }
        if(tienaA) vocalesDif++;
        if(tienaE) vocalesDif++;
        if(tienaI) vocalesDif++;
        if(tienaO) vocalesDif++;
        if(tienaU) vocalesDif++;

        return (vocalesDif >=3);
}
```