Laboratorio S02:
Tipos referencia en Java.

Laboratorio de formación. Guía paso a paso

Objetivos:

- Trabajar con la idea de array en Java y tipo (registro ADA)

0. Empieza creando un proyecto `LabS02` y un package `packclases`.

1. Subprogramas y uso de parámetros de tipo array

Existen algunos cambios de notación de Java con respecto a ADA: la forma de declararlo es totalmente diferente, la de usarlo muy parecida pero su gestión en memoria totalmente diferente. Lo veremos con un ejemplo.

Supongamos la clase `Histogram` que representa las funcionalidades asociadas a los histogramas de una imagen en niveles de gris (tiene 256 niveles de gris); el 0 representa el negro puro, el 255 el blanco puro y entre ambos toda la gama de grises desde los muy oscuros cercanos al 0, hasta los muy claros cercanos al 255. Un vector de 256 posiciones representa un histograma y contendrá la distribución de píxeles de una imagen en niveles de gris, es decir, cuántos píxeles de cada valor hay en la imagen. Por ejemplo, si en la posición 0 hay un 10, significa que 10 píxeles de la imagen son blancos puros (con valor 0).

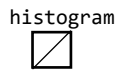
1.1. Creación de la clase Histogram: Crea la clase **`Histogram`** con **main** en el package **packclases**. Iremos completándola poco a poco.

1.2. Representación del histograma como un vector.

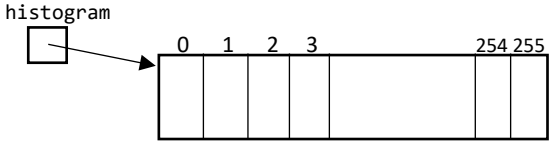
Representaremos el histograma como un array de 256 posiciones de enteros. Como sabemos las posiciones del array se numeran del 0 en adelante. Así que, si se define un array de 256 posiciones los valores de sus índices van del 0 al 255.

```
int[] histogram;

histogram= new int [256];
```

En la primera línea se indica que la variable `histogram` es un vector, pero a diferencia de ADA no hace reserva de memoria para el array de hecho, no se ha indicado todavía cuántas posiciones va a tener. Simplemente se indica que va a ser una array de enteros.



En la segunda sentencia es donde se hace la reserva de memoria. En nuestro caso de exactamente 256 posiciones en memoria para albergar valores de tipo entero.



Añade las dos líneas de código anterior dentro del método main.

En Java el contenido de una variable es un valor de un tipo primitivo, o un puntero a un objeto

1.3. Subprogramas y parámetro vector.

Vamos a definir dos métodos en los que se pasa como parámetro un histograma (un vector):

- `getNumPixels` que obtiene el número de píxeles de la imagen que se encuentra representada en el histograma que se pasa como parámetro.

```
/**
 * Obtains the number of pixels expressed in the histogram his
 * @param his express the histogram to be evaluated
 * @return the number of pixels expressed in his
 */
public static int getNumPixels(int[] his) {
   int num=0;
   for(int i=0; i<his.length; i++) {
      num= num+ his[i];
   }
   return num;
}
```

Las posiciones de los arrays están numeradas 0, 1, 2…

Acceso a una posición
Ada: Constraint_Error
Java: IndexOutOfBoundsException

Añade al main lo siguiente y ejecútalo:

```
public static void main(String[] args) {
   int[] histogram;
   histogram = new int[256];
   for (int i=0; i< histogram.length; i++) {
      histogram[i]=10*(i+1);
   }
   System.out.println("El número de pixeles es "+
                                          getNumPixels(histogram));
}
```

Modificación del valor de una posición (con valores cualesquiera).

```
El resultado que da será: número de pixeles es 328960
```

- ***Este Método déjalo para implementarlo al final del laboratorio.***

  Implementa y documenta el método average que calcula la media de los valores de los píxeles de una imagen representados en el histograma, es decir, calcula el valor medio de gris del histograma.

  El valor devuelto será de tipo double. Para hacerlo de manera eficiente piensa en la siguiente fórmula: (Σ(i=0..255) i*his[i])/num, donde his[i] expresa el valor del histograma en la posición i indicada, y num corresponde al número de píxeles de la imagen. Pruébalo desde el main. Debería dar:

  ```
  La media es 170.0
  ```

2. Subprogramas y uso de parámetros de tipo record

Vamos a modificar el enfoque que hicimos de la clase GPS en el primer laboratorio, ya que ahora vamos a diseñar un punto GPS como un estructura del estilo record de ADA.

Supongamos el siguiente programa Main definido en ADA. Tenemos definido el tipo GPS, que representa un punto GPS en la corteza terrestre, con su valor de latitud y longitud. Además tenemos definida la función Distance entre dos puntos y un programa principal que llama a ese método.

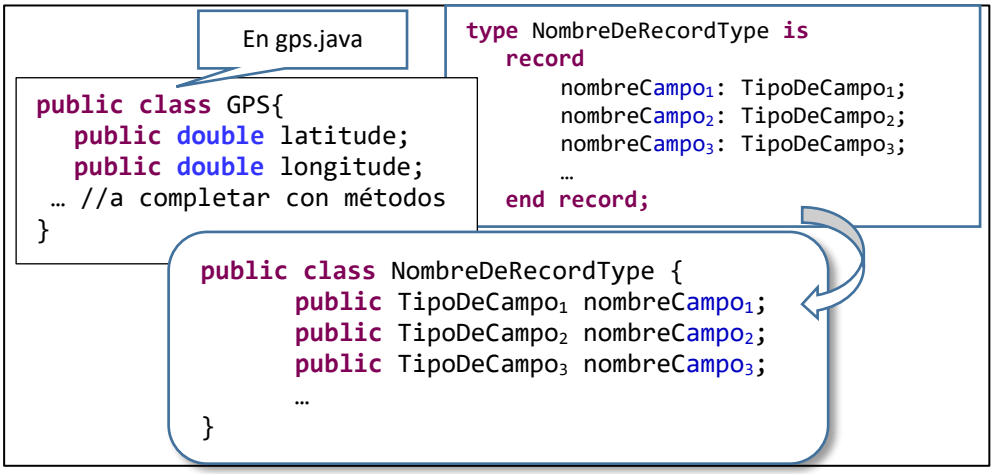
```
procedure Main is
   type GPS is record
      latitude: float;
      longitude: float;
   end record;

   function Distance(GPS1, GPS2: GPS) return float
is
   begin
      distance: float;
      …
      return distance;
   end Distance_To;

   G1, G2: GPS;
   D: float;
begin
   G1:= (43.318334, -1.9812313);
   G2:= (43.2630126, -2.9349852);
   D:= Distance(G1, G2);
   …
end Main:
```

En ADA en el momento de declarar la variable de tipo GPS (registro), se hace la reserva de memoria (G1 y G2 tienen reserva de memoria para poder guardar los valores de latitud y longitud. En java no sucede así. Lo iremos viendo en este laboratorio.

2.1. Una versión en Java.

Declaración de tipos *record*: cambia la notación y la organización del código.

Cada declaración de un tipo *record* se traduce en una *clase.*
Cada clase se guarda en un archivo.



De momento definimos los atributos (lo que eran los campos del registro) como **public** para que se pueda acceder a ellos. Esto cambiará en los próximos laboratorios para conseguir un uso más adecuado de la clase.

*Crea la clase **GPS** (seguimos en el proyecto **LabS02** y paquete **packclases**) tal y como se observa en el ejemplo.*

2.2. Reserva de memoria e inicialización de los valores de una clase en java.

Para solicitar la reserva de memoria se realiza mediante unas operaciones especiales llamadas *constructoras*.

Estas operaciones son **métodos** en los que no se indica ningún valor a devolver (ni void, ni ningún otro tipo) y tiene el mismo nombre que la clase.

```
public GPS (){
}
public GPS (double la, double lo){
   latitude= la;
   longitude =lo;
}
```

La constructora es lo primero que debe aparecer en la clase tras la declaración de los atributos. Se pueden declarar tantas constructoras como se crean necesarias. Incluso se puede definir una constructora que no tenga parámetros y no inicialice nada (en ese caso sólo hará la reserva de memoria). *Añade a la clase GPS las constructoras del margen.*

Si no se implementa ninguna constructora, java declara implícitamente una (sería como la primera que aparece en el código anterior, sin atributos y sin inicializaciones). Si se declara alguna constructora, java ya no declara ninguna otra implícitamente.

```
public static void main(String[] args) {
   GPS g1= new GPS(43.318334, -1.9812313); //GPS Donostia
   …
}
```

Añade el main del margen. En este ejemplo g1 ya tiene reserva de memoria (**new**) y se han inicializado sus valores de latitud (`43.318334`) y longitud (`-1.9812313`) mediante la llamada que se hace a la constructora.

2.3. Subprogramas: implementación de la función `distance`

Los parámetros de tipos array/*record* se declaran como los demás. Observa que la forma de nombrar los métodos en Java (se escribe en minúscula) difiere de Ada. *Escribe el método y complétalo con la información del laboratorio anterior.*

```
public static double distance(GPS g1, GPS g2) {
   double lat1 = g1.latitude;
   double lon1 = g1.longitude;
   double lat2 = g2.latitude;
   …
}
```

Acceso al valor del atributo de la clase (public)

2.4. Programa principal, programa `main`.

Ahora vamos a crear dos puntos GPS y llamar a **`distance`** necesitamos indicar explícitamente que se haga la reserva de memoria (**new**):

```
public static void main(String[] args) {
   GPS g1= new GPS(43.318334, -1.9812313); //GPS Donostia
   GPS g2= new GPS(43.2630126, -2.9349852); //GPS Bilbo
   System.out.println("Distance between Donostia and Bilbao:  "+
                                                  distance(g1, g2));
}
```

Recuerda situar toda la información dentro de la clase GPS. Pruébalo todo (guárdalo y ejécutalo).

- **2.5. Subprogramas: parámetro array de tipos no primitivos**

  Escribiremos los valores de los puntos GPS incluidos en una lista. Revisa el código del método que se ofrece e *inclúyelo en tu implementación. Completa el main añadiendo a un array varios puntos GPS y llamando a dicho método. Comprueba con el debugger cómo se va incluyendo información en cada posición del array.*

```
public static void printArrayOfGPS(GPS[] lista){
   int index=0;
   GPS g;
   while (index< lista.length) {
      g= lista[index];
      System.out.println("("+g.latitude+", "+g.longitude+")");
      index++;
   }
}
```

Las posiciones de los arrays están numeradas 0, 1, 2…

3. Importante para estudiar y reflexionar. Cosas a recordar. Punteros en Java: tipos referencia.

En Java hay dos familias de tipos. Una la forman los tipos primitivos, y la otra incluye a todos los demás, que se denominan colectivamente como *tipos referencia* (los arrays y la clase GPS son dos ejemplos). Los tipos referencia de Java corresponden a los tipos *access* de Ada, y el análogo Ada de una clase como GPS en Ada sería un tipo *access* y no un simple tipo *record* como lo habíamos visto antes. En realidad, sería lo siguiente:

```
type GPS_Ada is record
       X: Float;
       Y: Float;
end record;
type GPS_Java is access GPS_Ada;
```

Revisa las siguientes declaraciones e inicializaciones:

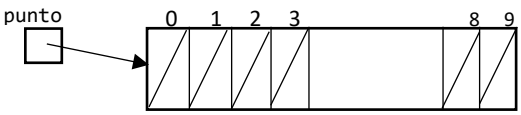
```
GPS g;            //sin reserva ni inicialización!

int[] array;      //sin reserva ni inicialización!
GPS[] puntos;     //sin reserva ni inicialización!

g = null;         //a las variables de tipo referencia
                  //se les puede asignar el puntero nulo!
```

¡los tipos array son tipos referencia!

```
array= null;
puntos = null;
```

¡ los objetos se crean!

```
g = new GPS();  // Ada: g = new GPS_Java;

puntos = new GPS[expresión entera];// la expresión índica el tamaño del array. Supongamos 10.
         // Además cada elemento de la posición
         // como se refiere a un tipo no primitivo
         // a su vez es un apuntador (nulo).
```




```
array = new int[]{1, -2, -3}; // array apunta a un array con tres posiciones
array = new int[]{};      // array apunta a un array con cero posiciones
```

4. Termina todo el Laboratorio y súbelo a eGela.

Terminadas todas las tareas debes exportar el proyecto. Para saber cómo exportar un proyecto, es importante que revises la documentación que está disponible en la pestaña **Sotfware** del curso de eGela. Es necesario que subas a eGela el proyecto exportado.

Tienes en eGela la tarea correspondiente para subir tu trabajo (**fecha límite de entrega viernes 5 de febrero de 2021**).