



PDOO Práctica

Ruby

- ☐ **IMPORTANTE BUENA PRÁCTICA** poner **espacios** entre **=** (hay **métodos** que con **<var>=**)

Enumerados

- ☐ Se crean en un **archivo aparte** con la **cabecera module <nombre>** y su **sintaxis** es está

```
module <nombre>
  # El símbolo es con lo que se comparara el enumerado <nombre>
  # <nombre1> va en mayusculas
  <nombre1> = :<símbolo>
  ...
end
```

- ☐ Se llama así **<nombre>::<nombre1>** si está **dentro** del **mismo modulo** ("**paquete** ") sino es así **<module_enumerate>::<nombre>::<nombre1>**
- ☐ Así se ponen los **métodos de clase** y los **atributos de clase**

```

class <nombre>
  @@<NOMBRE> = <valor> # Atributo de clase
  @<NOMBRE> = <valor> # Atributo de instancia de clase
  <NOMBRE> = <valor> # Constante (NO RECOMENDABLE)

  # Método de clase, se puede utilizar para devolver atributo
  # de instancia de clase, sino no se puede acceder a él
  def sel.<nombre>
    <calculos>
    <NOMBRE> = <nuevo_valor> # ERROR no se puede cambiar
    # el resto de atributos (arriba definidos) si se pueden
  end
  ...
end

# Llamada a métodos de clase
<nombre_clase>.<metodo_clase>
# Acceso a constantes son publicas
<nombre_clase>::<constante>

```

Clase Random

- ☐ Para **crear** una `instancia` **<instancia>=Random.new**
- ☐ Veamos `posibles casos` de uso, nombrando `instancia=Random.new`

```

instancia.rand() # Num flotante [0,1)
instancia.rand(<num>) # Num int [0,num)
instancia.rand(<num>.0) # Num flotante [0,num)
instancia.rand(<num1>..<num2>) # Num entero [num1,num2]
instancia.rand(<num1>...<num2>) # Num entero [num1,num2)
instancia.rand(<num1>.0..<num2>.0) # Num flotante [num1,num2]

```

los **<num>** utilizados son `enteros`

Arrays

- ☐ Puedo **definir arrays** y **matrices** mezclando **todo tipo de datos** así

```
<array>=[ <var>, ... ]  
<matriz>= [ [ <var>, ...] [ ... ] ... ]
```

- ☐ Puedo **crear arrays** y **matrices** a partir de la **clase Array**

```
# No es necesario el valor por defecto en ambos casos  
<array>=Array.new(<tam>, <valor_defecto>)  
<matriz>= Array.new(<tam1>) { Array.new(<tam2>, <defecto>)}
```

- ☐ **Métodos útiles** de clase **Array**

- **<array>.include?(<elem>)**: **true** si se **está** dicho **elemento**
- **<array>.push(<elem>)** o **<array> << <elem>**: **añade** un **elemento** al **final**
- **<array>.unshift(<elem>)**: **añade** un **elemento** al **principio**
- **<array>.pop**: **elimina** y **devuelve** el **último elemento**
- **<matriz>.flatten**: **convierte** la **matriz** en un array
- **<array>.length** o **<array>.size**
- **<array>.empty?**: **true** si está **vacío**
- **<array>[<indice>]** o **<array>.at(<indice>)**
- **<array>.delete(<elem>)**: **elimina** **elemento** del array
- **<array>.delete_if { |<elem>| <condicion> }**: **elimina** **elemento** del array **si** la **condición** es **true**, recorre **todo el array**, así tambien

```
<array>.delete_if do |<elem>|  
  codigo # tiene que devolver true o false  
end
```

- ☐ Para **iterar** en **arrays** se puede hacer de **2** **formas**

```
# Iterar por elementos
<array>.each do |elem|
  codigo
end
# Iterar por índices
<array>.each_index do |i|
  codigo # puedes acceder con <array>[i]
end
```

Bucle FOR

- ☐ Veamos la **sintaxis** para hacer **bucles** de **2** formas

```
# Iterar como C++
for i in <num1>..<num2> do # Itera así [num1, num2]
  codigo
end
# Si quiero [num1, num2), poner <num1>...<num2>

#Iterar con .times
<num>.times do |i|
  codigo
end
# Itera [0,num)
```

Java

- ☐ **Convertir** de un **tipo** a otro (**<nuevo_tipo>**)(**<var>**), como en **C++**
- ☐ Así se ponen los **métodos de clase** y los **atributos de clase**

```
class <nombre>{
  // atributos de clase
  // "final" indica que no se puede modificar dicho valor (cor
```

```

    static private final <tipo> <NOMBRE> = <valor>;

    // método de clase
    static <tipo> <nombre>{
        <calculos>
    }
    ...
}
// Para acceder
<clase>.<método_clase>

```

Enumerados

- ☐ Se **crean** con la siguiente **sintaxis**

```

<visibilidad> enum <nombre>{
    <nombre1>,
    <nombre2>,
    ...
}

```

Clase Random

- ☐ Se debe **importar el archivo** así **import java.util.Random**
- ☐ Se usa **.nextInt(<params>)** para **enteros** y **.nextFloat(<params>)** para **flotantes**
- ☐ Veamos **posibles casos** de uso, nombrando **instancia= new Random();**

```

instancia.nextInt() // Num [Integer.MIN_VALUE, Integer.MAX_VALUE)
instancia.nextFloat() // Num [0,1)
instancia.nextInt(<num>) // Num [0,num), para num], poner (<num>:
instancia.nextFloat() * <num> // Num [0,num)

```

Arrays

- ☐ Puedo **definir arrays** y **matrices** mezclando **todo tipo de datos** así

```
// Puedo poner <tipo> en lugar de Object
Object [] <array> = { <var>, ...}
Object [] [] <matriz> = { { <var>, ...}, {...}, ...}
```

- ☐ Para **acceder** en este **caso general** de **tipo** es así **(<tipo><array>[<indice>]**

- ☐ Destacar que **definir** así los **arrays** hace que **SOLO** se puede **hacer <array>[<indice>]** (en cuanto a **"métodos"**)

- ☐ Puedo crear **arrays** y **matrices** a partir de la **clase ArrayList** (**necesario** poner **import java.util.ArrayList**

```
// Opcional poner <initial_tam>
ArrayList<tipo> <nombre> = new ArrayList<tipo>(<initial_tam>);

// Puedo dividir la definición e inicialización en 2 partes
ArrayList<tipo> <nombre>;
<nombre>=new ArrayList<>();
// Equivalente poner <tipo> y no ponerlo (mejor no ponerlo)

// Para matrices
ArrayList< ArrayList <tipo>> <nombre> = new ArrayList<>();
```

- ☐ **Métodos útiles** de la clase **ArrayList**

- **<array>.add(<elem>)**: se **añade elemento** al **final**
- **<array>.add(<indice>, <elem>)**: añade **elemento** en el **índice especificado**
- **<array>.get(<indice>)**
- **<array>.set(<indice>, <elem>)**: **cambia** el **elemento** de ese **índice** por el dado
- **<array>.remove(<elem>)**: **elimina** la **primera ocurrencia** del **elemento**

- Si **<elem>** es un **entero** se **elimina** el **índice**
- Para **arrays de enteros** para **eliminar** el **elemento** poner
<array>.remove(Integer.valueOf(<elem>)) (más **seguro**)
- **<array>.clear()**: **elimina** **todos** los **elementos**
- **<array>.isEmpty()**: **true** si está **vacía**
- **<array>.contains(<elem>)**: **true** si está el **elemento**
- **<array>.size()**: **num** **elementos**