# Tema 10: Características de otros LOO.

## Introducción.

- De temas anteriores conocemos las características que debe tener un lenguaje de programación para ser considerado como *orientado a objetos*.
- También hemos hablado de la parte histórica de este paradigma de programación:
  - Se considera a *Simula-67* como el primer LOO.
  - Y a *Smalltalk* como el ejemplo icónico de LOO.
- En este tema vamos a ver de que manera otros LOO implementan algunos de los principios que hemos visto en la práctica sólo con **C++**.

## Lenguajes elegidos.

- SmalllTalk
- Objective-C
- Java
- C#
- Vala
- D
- Python

## SmallTalk.

- Surge en los años 70 en los laboratorios de investigación de XEROX ( XEROX PARC).
- Trataba de aunar las tareas de programación y gestión a nivel de S.O. del computador.
- Introdujo :

concepto lo han empleado luego lenguajes como Java y C#.

- Una biblioteca de clases y objetos reusables que facilitaban las labores del programador.

## El lenguaje SmallTalk.

- En SmallTalk todo es un objeto, incluso los tipos base de lenguaje y por tanto las constantes de estos.
- Tiene sólo herencia simple y con una clase base común: `Object` .
- Es débilmente tipado.
- Dispone de un *browser* de clases.
- Dispone de recolección de basura.
- Tenemos diversas implementaciones disponibles, una de ellas bastante sencilla de instalar y usar en diversos S.O.: Squeak SmallTalk.
- El equivalente a *this* de **C++** es **self**.

## Ejemplos de código SmallTalk.

- Hola mundo:

```
  "Hello world Program"
  'Hello World!' printNl !
```

- Cáculo del factorial de un número:

```
  factorial
"Answer the factorial of the receiver."

self = 0 ifTrue: [^ 1].
self > 0 ifTrue: [^ self * (self - 1) factorial].
self error: 'Not valid for negative integers'
```

- Ejemplo de llamada al factorial:

```
10 timesRepeat: [
   Transcript show:'hello'.
   Transcript cr.
].
```

## Ejemplos de código SmallTalk.

- Bucle *for*:

```
1 to: 10 do:[ :i |
   Transcript show:i.
   Transcript show:' '.
   Transcript show:i sqt.
   Transcript cr.
].
```

- Bucle sobre una coleccion:

```
#('a' 'b' 'c' ) do: [:each |
   Transcript show: each.
   Transcript cr.
].
```

## Ejemplos de código SmallTalk.

- Los **[...]** que acompañan al **do:** ¡son un objeto!:

```
|myArray myOperation|

myArray := #('a' 'b' 'c' ).
myOperation := [:each |
                  Transcript show: each.
                  Transcript cr.
               ].
```

## Otros LOO.

- En la década de los 80 comienza a imponerse el desarrollo de aplicaciones bajo el *paradigma orientado a objetos*.
- Lenguajes como SmallTalk o bien son interpretados o producen ejecutables lentos.
- Por tanto desarrolladores de compiladores de lenguajes de 3ª generación se plantean incorporar algunas de las características de la POO a este tipo de lenguajes.
- Surgieron así *extensiones* de lenguajes como Pascal y de **C**.
- En el caso de Pascal se creó Object Pascal y en el de **C** aparecieron varias extensiones para convertirlo en un LOO: **C++**, **Objective-C** y otras.

# Objective-C.

- Los creadores de Objective-C (Brad Cox y Tom Love) estaban muy influenciados por SmallTalk.
- La extensión que crearon de **C** es una mezcla de ambos lenguajes:
  - Sintáxis de **C** para la parte procedural.
  - Sintáxis de **SmallTalk** para la parte de POO.
- En los 80 y 90 *Objective-C* fue superado ampliamente en uso por **C++**. Pero en 1998 algo ocurrió...
- ... NeXT Inc. licenció el uso de Objective-C y preparó el compilador de C de GNU (gcc) para que lo soportara. Se convirtió en el lenguaje de desarrollo de NeXTStep, más tarde OpenStep y más tarde OS-X / iOS.

## El lenguaje Objective-C.

- Dispone de herencia simple.

- Los ficheros de cabecera usan la extensión **.h** y los de implementación **.m**.
- Los mensajes a objetos se envían con notación de SmallTalk: `[obj method:parameter];`

## Ejemplos de código Objective-C.

Puedes encontrar más en la página de la wikipedia.

- Declaración de una clase:

class.h

```
@interface classname : superclassname { ... }
+classMethod1;
+(return_type) classMethod2;
+(return_type) classMethod3: (param1_type)parameter_varName;
-(return_type) instanceMethod: (param1_type)param1_varName
                              secondParam: (param2_type)param2_varName;
@end
```

- Implementación:

```
#import "class.h"

@implementation classname
+classMethod {
    // implementation
}
-instanceMethod {
    // implementation
}
@end
```

## Ejemplos de código Objective-C.

- Se admite dar nombre a argumentos de métodos:

- Creación de objetos

```
MyObject * o = [[MyObject alloc] init];
...
-(id) init {
  self = [super init];
  if (self) {...}
  return self;
}
```

# Java.

- Creado por Sun MicroSystems en 1995. Hoy en día es propiedad de Oracle Corp.
- Sintáxis muy parecida a la de **C++**.
- Es compilado y usa una máquina virtual (*JVM*) para respresentar y ejecutar su código objeto.
- Desde 2010 aproximadamente es uno de los lenguajes de programación más usados. El espaldarazo definitivo le vino por su uso en el S.O. Android de Google.
- Puede generar aplicaciones que funcionan tanto en la parte de escritorio como en la parte web.

## El lenguaje Java.

- Dispone de clases e interfaces.
- Fuertemente tipado con herencia simple de clases y múltiple de interfaces.
- Genericidad que permite crear una biblioteca de colecciones muy amplia y que ha marcado un estándar entre los programadores.
- Recolección de basura.

- No dispone de espacios de nombres pero si de paquetes: *package*. Estos deben ser importados donde se usan mediante la orden *import*.

## Componentes de Java.

- Disponemos de dos paquetes de software relacionados con Java:
  - OpenJDK
  - OpenJRE
- Disponemos de un compilador (*javac*) y de la máquina virtual para ejecutar el código compilado (*java*).
- Formando parte del JRE se encuentra la biblioteca de clases de Java (*JCL*).
- No es imprescindible pero si conveniente, para desarrollar en Java es bueno disponer de un IDE:
  - Eclipse
  - NetBeans

## Ejemplos de código Java.

- Creación de una clase:

```
public class Model {
  private Map<Character, FeatureVector> features;
  /// Default constructor
  public Model() {
    features = new HashMap<>();
  }
  ...
```

- Herencia de clases:

```
package gtaligner;
import gtaligner.io.Messages;
import gtaligner.io.TextReader;
```

## Ejemplos de código Java.

- Herencia de interfaz:

```
public interface Sortable {
   public bool isLessThan (Sortable b);
}
public class Line implements Sortable {...}
```

# C# (C Sharp).

- Creado por Microsoft.
- Similar en arquitectura a Java. Dispone de una máquina virtual en la que se ejecuta el código compilado.
- A la plataforma creada por Microsoft para desarrollar y ejecutar código en **C#** (y otros lenguajes) se le llama .Net.
- **C#** se parece mucho A **C++** en cuanto a sintáxis y palabras reservadas.
- Dispone de manera estándar de una biblioteca de clases muy amplia.
- **.Net** permite mezclar código objeto generado por diferentes lenguajes. Los tipos base de todos los lenguajes para **.Net** ocupan lo mismo.

## El lenguaje C#.

- Dispone de clases e interfaces.
- Fuertemente tipado con herencia simple de clases y múltiple de interfaces.
- Genericidad que permite crear una biblioteca de colecciones muy amplia.
- Recolección de basura.
- Tiene destructores pero se parecen más a los finalizadores de Java.
- No hay archivos de cabecera e implementación.
- Se recomienda desarrollar con VisualStudio o con la versión de código abierto visual studio code.

```
using System.Windows;
namespace MyCalculatorv1
{
    public partial class App : Application
    {
    }
}
```

## Ejemplo de código C#.

```
using System;
using System.Windows;
using System.Windows.Controls;
namespace MyCalculatorv1 {
 public partial class MainWindow : Window {
   public MainWindow() {
       InitializeComponent();
   }
   private void Button_Click_1(object sender, RoutedEventArgs e) {
       Button b = (Button) sender;
       tb.Text += b.Content.ToString();
   }
   private void Result_click(object sender, RoutedEventArgs e) {
       try {
           result();
       }
       catch (Exception exc) {
           tb.Text = "Error!";
       }
   }
   ...
```

## Proyecto mono.

- Mono es una implementación libre y de código abierto de *.Net*, de la máquina virtual y del compilador de C#.

# Vala.

- Vala es un lenguaje muy parecido a **C#**, aunque toma cosas de **C++** y también de **Java**.
- Está asociado a proyectos de código abierto.
- Puedes ver documentación sobre el mismo aquí.
- Dispone de una biblioteca de colecciones llamada Gee.
- Permite crear aplicaciones con interfaz gráfico de usuario de manera muy sencilla, mira este vídeo.

## El lenguaje Vala.

- Dispone de clases e interfaces.
- Fuertemente tipado con herencia simple de clases y múltiple de interfaces.
- Genericidad que permite crear una biblioteca de colecciones muy amplia (Gee).
- Liberación de memoria basada en cuenta de referencias.
- Tiene destructores como en **C++**.
- Permite añadir métodos a tipos enumerados.
- Dispone de la emisión de señales y ejecución de código diferido en el mismo lenguaje.
- Al igual que en **C#** dispone de propiedades.
- No hay archivos de cabecera e implementación.

## Ejemplo de código Vala.

- Un ejemplo sencillo:

```
using GLib;
class Droid {
    public Droid (string n) {
      name = n;
    }
    public string name {get; set;}  // propiedad: variable+set+get todo-junto
}
```

```
        stdout.printf("Nombre: %s\n", d.name);
        return 0;
    }
```

## Características adicionales a Vala.

- Dispone de una web con la documentación de todas las bibliotecas que tienen una adaptación a **Vala**.
- Realmente el compilador de **Vala** es un *traductor* a **C**.
- Se puede parar la *compilación* en el instante en el que se genera el código **C** y verlo.
- El compilador de vala se llama igual: *valac*. Lo tenéis instalado en la máquina virtual.

# D.

- Creado inicialmente por Digital Mars, fabricante de compiladores de C/C++.
- Trata de afrontar las pegas de **C++** como LOO debido a su compatibilidad hacia atrás con **C**.
- Hoy en día su desarrollo se hace por parte de una comunidad y se cuenta con una *fundación* que promueve el desarrollo y uso del mismo.

## El lenguaje D.

- Dispone de clases e interfaces.
- Fuertemente tipado con herencia simple de clases y múltiple de interfaces.
- Genericidad al estilo de **C++** (permite metaprogramación).
- Recolección de basura.
- Tiene destructores para clases parecidos a los finalizadores de Java y destructores similares a los de **C++** para *structs*.

## El lenguaje D.

- Permite tener funciones libres, incluso permite anidar funciones.

- Existen tres compiladores de D. Todos ellos libres. Los puedes descargar de aquí.

## Ejemplo de código D.

- Haciendo uso de notación funcional:

```
// Sort lines
import std.stdio, std.array, std.algorithm;

void main() {
    stdin
         .byLineCopy           // No es necesario usar () en la llamada
         .array                // si no hay argumentos
         .sort
         .each!writeln;
}
```

## Ejemplo de código D.

- Uso de clases:

```
import std.stdio;

class Base {
   protected int y = 8;
   private int n = 9;
}

class Derived : Base {
  public int get_n () { return n; } // We can access 'n' 'cause Base and
                                    // Derived are declared in the same file!
  public int get_k () { return k; }

  private int k = 0;

  ~this () {
    writeln ("~Derived.");
```

```
        Derived d = new Derived;

        writeln (d.get_k);
    }
```

# Python.

## Historia.

- Creado a finales de los años 80.
- Su desarrollador inicial fue Guido van Rossum.
- Toda la documentación sobre el lenguaje la puedes encontrar en la web del mismo: python.
- Python admite diversos paradigmas de programación, entre ellos el Orientado a Objetos. También soporta el concepto de módulos como mecanismo de división del código fuente en varios archivos.
- Actualmente puedes encontrar dos versiones de python empleadas en producción: La versión ptyhon2 y la versión python3. Ten en cuenta que son incompatibles a nivel de

## Python: Clases.

- Se declaran y definen en el mismo archivo.
- Los métodos de instancia deben declarar explícitamente un primer parámetro que representa el objeto al que se le envia el mensaje. El convenio es llamarlo `self` .
- Los objetos se crean con la notacion que ya conocemos de `C++` :

```
myObject = myClass()
```

- El constructor invoca el método especial `__init__` :

```
def __init__(self):
    self.data = []  # Obligatorio el uso de self
```

```
class Bag:
def __init__(self):
    self.data = []

def add(self, x):
    self.data.append(x)

def addtwice(self, x):
    self.add(x)
    self.add(x)
```

- Podemos declarar variables y métodos de clase :

```
class Complex:
    count = 0

    def onemore():
        Complex.count += 1

    def __init__(self, realpart = 0.0, imagpart = 0.0):
        self.r = realpart
        self.i = imagpart
        Complex.onemore()

    def rpart(this):
        return this.r

x = Complex(3.0, -4.5)
x2 = Complex()
print ("x.r = ", x.rpart())
print ("x2.r = ", x2.rpart())
print ("Complex numbers created: ", Complex.count)
# -------------- Output: --------------------
```

```
x.r =  3.0
x2.r =  0.0
```

- Y también múltiple:

```
class DerivedClassName(Base1, Base2,...,BaseN):...
```

- Técnicamente no existe la visibilidad privada de identificadores definidos en una clase, pero se puede obtener un resultado parecido empleando como prefijos/sufijos símbolos de subrayado.

## Python: Módulos.

- Los módulos son la manera que Python tiene de permitirnos separar el código que escribimos en diversos archivos.
- Cada fichero se convierte en un módulo llamado como el archivo pero sin la extensión `.py` .
- Los módulos se *importan* con la sentencia *import*:

```
import name-of-module
```

- Los símbolos importados de un módulo pertenecen al espacio de nombres de ese módulo:

fibmod.py

```
def fib(n):    # write Fibonacci series up to n
  a, b = 0, 1
  while a < n:
    print(a, end=' ')
    a, b = b, a+b
  print()
  ....

# Main Program, file: main.py
import fibmod
fibmod.fib(200)
```

```
from fibmod import fib
fib(120)
```

- Se admite el uso del carácter '*' como comodín:

```
from fibmod import *
fib(120)
```

- Se permite el renombrado:

```
from fibo import fib as fibonacci
fibonacci(120)
```

- Los módulos se *importan* desde el directorio actual o desde los directorios definidos en la variable: *sys.path*.
- Esta variable contiene una lista de directorios donde residen los módulos estándar que vienen con la instalación de python, p.e. el módulo *sys*.
- Para más información sobre módulos consulta el tutorial.

## Python: Depuración.

- Para depurar un programa en Python podemos hacer uso del módulo `pdb` .
- Este módulo define un depurador interactivo a nivel de código fuente. Podemos poner puntos de parada, ejecutar sentencias paso paso, movernos por la pila de llamadas, etc...
- Para más información consulta la página web de pdb.

# Aclaraciones.

- **Este contenido no es la bibliografía completa de la asignatura**, por lo tanto debes estudiar, aclarar y ampliar los conceptos que en ellas encuentres empleando los enlaces

Página anterior
← Tema 9: Excepciones. Patrón RAII.

Siguiente página
Tema 11: POO y lenguajes no OO. →