

## Content: I Presentation.

### Herramientas Avanzadas para el Desarrollo de Aplicaciones

Languages and Computer Science University of Alicante

Curso 2014-2015 , Copyleft © 2011-2015 .

Reproducción permitida bajo los términos de la licencia de documentación libre GNU.



1 / 44

## Teachers

- Garrigós Fernández, Irene - (Coordinator)
- Corbí Bellot, Antonio-Miguel
- Muñoz Terol, Rafael
- Martínez-Larraz Prats, Carlos

Offices, office hours, previous appointments, etc: [www.dlsi.ua.es](http://www.dlsi.ua.es).



3 / 44

## Content

- 1 Teachers
- 2 Teaching guide
- 3 Contents
- 4 Assessment
- 5 Assessment without passing the ongoing work
- 6 Learning plan (I)
- 7 Learning plan (II)
- 8 Programming languages used
- 9 Vala Introduction
- 10 Vala Features
- 11 Reserved words
- 12 Operators
- 13 Hello World in Vala
- 14 Compilation
- 15 Strings
- 16 Input / Output
- 17 Arrays
- 18 Classes



2 / 44

## Teaching guide

- **Virtual Campus** → **Learning resources** → **teaching guide**
- Calendar, objectives, content, learning plan, assessment, bibliography and links
- 1,2 credits for theory classes + 1,2 practical credits.



4 / 44

The subject is composed by the following units:

- **U1** - Presentation, programming language
- **U2** - Version control systems
- **U3** - Event driven programming and deferred code execution
- **U4** - Graphical user interfaces
- **U5** - DDBB access from desktop applications
- **U6** - Libraries management
- **U7** - Design and development of Web applications (I)
- **U8** - DDBB access using an object model
- **U9** - Effective presentations

- 1 **Ongoing assessment:** Individual work. Students will have to do 3 practical assignments individually. **Marks: P1: 2,5%, P2: 7,5%, P3: 10%. Total: 20%.**
- 2 **Ongoing assessment:** Test. Students will be evaluated by an individual test in the middle of the semester. Minimal mark required: 4. **Marks: 30%.**
- 3 **Ongoing assessment:** Group assignments. Students will do a project in a collaborative way which will have to be finished by the end of the semester. Moreover, the students should do an oral defense of this work **Marks: 30%.**
- 4 **Ongoing assessment:** Test. Students will be evaluated by an individual test at the end of the semester in the official date stated by the polytechnic school in june. Minimal mark required: 4. **Marks: 20%.**

## Assessment without passing the ongoing work

- **Attention!** In July, students who do not pass the subject with the ongoing work will have to do an exam, which mark can count up to 50%.
- Marks obtained during the semester are maintained for calculating the final mark in June.
- **For more details, check the document in the virtual campus "Hada assessment criteria".**

## Learning plan (I)

Week	U.	Pres. work	Non-p. work
01	1	Introduction to the subject. Introduction to the programming language.	-
02	2	Version control systems	Guided practical work to understand the programming environment.
03	2	Version control systems	individual assignment 1
04	3	Event driven programming and deferred code execution	individual assignment 1
05	4	Graphical user interfaces	individual assignment 2
06	5	DDBB access from desktop applications	individual assignment 3
07	6	Libraries.	individual assignment 3

Week	U.	Pres. work	Non-p. work
08	7	Introduction to C# and Web applications	Group assignment
09	8	Layered model	Group assignment
		<b>Evaluation (test)</b>	
10	7	Web applications: Interface layer	Group assignment
11	7	Web applications: Interface layer (II)	Group assignment
12	8	DDBB access. Connected environment	Group assignment
13	8	DDBB access. Disconnected environment. Effective presentations	Group assignment
14	9	Advanced aspects in the development of Web applications	Group assignment. Oral presentation.
15	1-9	Recap session	Correction of group assignment
<b>Total</b>		<b>60</b>	<b>90</b>

- ① Individual assignments Vala language.
- ② Group assignments C# language (within ASP.net).

## Vala Introduction

- Vala is a new programming language: Vala
- It uses the functionalities provided by Glib y GObject
- The Vala compiler generates 'C' code, which is compiled by a **C Language** compiler.
- It is a similar language to Java and C#, more similar to this last one.

## Vala Features

- ① POO (classes, abstract classes, mixin interfaces, polymorphism)
- ② Namespaces
- ③ Delegates
- ④ Properties
- ⑤ Signals
- ⑥ Automatic notification of properties modification
- ⑦ Foreach
- ⑧ Lambda Expressions / Closures
- ⑨ Inference of local variable types
- ⑩ Generic Types
- ⑪ Non-null types
- ⑫ Automatic management of dynamic memory (automatic reference counting)
- ⑬ Deterministic destructors (RAII)
- ⑭ Exceptions (checked exceptions)
- ⑮ Asynchronous Methods (coroutines)
- ⑯ Preconditions and postconditions (programming by contract)
- ⑰ Run-time type information
- ⑱ Named Constructors
- ⑲ Verbatim Strings
- ⑳ Array and string chunking
- ㉑ Conditional compilation
- ㉒ Similar syntax to C#
- ㉓ Compatibility in the ABI level with C.

- Selection: if, else, switch, case, default
- Iteration: do, while, for, foreach, in
- Jump: break, continue, return
- Exceptions: try, catch, finally, throw
- Synchronization: lock
- Types declaration: class, interface, struct, enum, delegate, errordomain
- Types modifiers: const, weak, unowned, dynamic
- Modifiers: abstract, virtual, override, signal, extern, static, async, inline, new
- Access Modifiers: public, private, protected, internal
- Parameters of methods: out, ref
- Programming by contract: throws, requires, ensures
- Namespaces: namespace, using
- Operators: as, is, in, new, delete, sizeof, typeof
- Access: this, base
- Literals: null, true, false
- Properties: get, set, construct, default, value
- Constructor blocks: construct, static construct, class construct
- Others: void, var, yield, global, owned

- Arithmetics: +, -, \*, /, %
- Bit by bit: ~, &, |, ^, <<, >>
- Relational: <, >, <=, >=
- Equality: ==, !=
- Logical: !, &&, ||
- Assignment: =, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=
- Increment, Decrement: ++, --
- Pointers: &, \*, ->, delete
- Conditionals: ?
- Comparison with null: ??
- String concatenation: +
- Methods invocation: ()
- Member access: .
- Index: [ ]
- Chunking: [ : ]
- Lambda: =>
- Casting: (Type), (!), as
- Type checking at runtime: is
- Owing transfer: (owned)
- Namespaces alias qualifier: :: (currently only with global)
- Others: new, sizeof, typeof, in

## Hello World in Vala

## Compilation

```

1  class Demo.HelloWorld : GLib.Object {
2      public static int main(string[] args) {
3          stdout.printf("Hello, World\n");
4          return 0;
5      }
6  }

```

- \$ valac compiler.vala --pkg libvala
- \$ valac source1.vala source2.vala -o myprogram
- \$ valac hello.vala -C -H hello.h

```

1  int a = 6, b = 7;
2  string s = @"$a * $b = $(a * b)"; // => "6 * 7 = 42"

4  string greeting = "hello, world";
   string s1 = greeting[7:12]; // => "world"
6  string s2 = greeting[-4:-2]; // => "or"

8  bool b = bool.parse("false"); // => false
   int i = int.parse("-52"); // => -52
10 double d = double.parse("6.67428E-11"); // => 6.67428E-11
   string s1 = true.to_string(); // => "true"
12 string s2 = 21.to_string(); // => "21"

14 if ("ere" in "Able was I ere I saw Elba.") ...

```

```

1  stdout.printf("Hello, world\n");
   stdout.printf("%d %g %s\n", 42, 3.1415, "Vala");
3  string input = stdin.read_line();
   int number = int.parse(stdin.read_line());

```

- We also can use the standard error output represented by “stderr”.
- We can show information on it by using “printf” as follows:  
stderr.printf(“‘...’”);

```

1  int[] a = new int[10];
2  int[] b = { 2, 4, 6, 8 };
   int[] c = b[1:3]; // => { 4, 6 }
4  int al = a.length;

6  int[,] c = new int[3,4];
   int[,] d = {{2, 4, 6, 8},
7             {3, 5, 7, 9},
8             {1, 3, 5, 7}};
10 d[2,3] = 42;
   int d0l = d.length[0];
12 int[] e = {}; e += 12; e += 5; e += 37;

```

```

1  /* defining a class */
   class Track : GLib.Object { // subclassing 'GLib.Object' */
3     public double mass; // a public field */
   public double name { get; set; } // a public property */
5     private bool terminated = false; // a private field */
   public void terminate() { // a public method */
7         terminated = true;
9     }
}

```

```
1  int i = 10;
   float j = (float) i;
3
   var p = new Person(); // same as: Person p = new Person();
   var s = "hello"; // same as: string s = "hello";
5   var l = new List<int>(); // same as: List<int> l = new List<int>();
   var i = 10; // same as: int i = 10;
7
9   MyFoo<string, MyBar<string, int>> foo = new MyFoo<string, MyBar<
   string, int>>();
   // Compare with ...
11  var foo = new MyFoo<string, MyBar<string, int>>();
```

```
1  stdout.printf("Hello, %s!\n", name ?? "unknown person");
```

## Foreach

```
1  foreach (int a in int_array) { stdout.printf("%d\n", a); }
```

## Automatic checking of null values

```
1  string? method_name(string? text, Foo? foo, Bar bar) {
   // ...
3  }
5  Object o1 = new Object(); // not nullable
   Object? o2 = new Object(); // nullable
7
   o1 = o2; // Forbidden
9  o1 = (!) o2; // Allowed with the non-null cast explicit: operator !
```

# Delegates

<https://live.gnome.org/Vala/Tutorial>

```
1  delegate void DelegateType(int a);  
  
3  void f1(int a) {  
    stdout.printf("%d\n", a);  
5  }  
  
7  void f2(DelegateType d, int a) {  
    d(a);          // Calling a delegate  
9  }  
  
11 void main() {  
    f2(f1, 5);     // Passing a method as delegate argument to another  
                  method  
13 }
```

# Closures

<https://live.gnome.org/Vala/Tutorial>

```
2  delegate void PrintIntFunc(int a);  
  
4  void main() {  
    PrintIntFunc p1 = (a) => { stdout.printf("%d\n", a); };  
6    p1(10);  
    // Curly braces are optional if the body contains only one statement  
    :  
8    PrintIntFunc p2 = (a) => stdout.printf("%d\n", a);  
    p2(20);  
10 }
```

# Namespaces

<https://live.gnome.org/Vala/Tutorial>

```
1  namespace Hada {  
2    int n;  
3  }  
4  
5  using Hada;  
6  n = 3; // Or also ...  
   Hada.n = 3;
```

# Visibility

<https://live.gnome.org/Vala/Tutorial>

<b>public</b>	With non access restrictions
<b>private</b>	Limited access from inside the structure or class definition. This is the default access if nothing is specified.
<b>protected</b>	Limited access from inside the structure or class definition or from any other class that derives from it.
<b>internal</b>	Limited access from the classes defined in the same package

```

1  public class Button : Object {
2      public Button() {
3      }
4
5      public Button.with_label(string label) {
6      }
7
8      public Button.from_stock(string stock_id) {
9      }
10 }
11
12 class Demo : Object {
13     ~Demo() {
14         stdout.printf("in destructor");
15     }
16 }

```

```

1  public class Test : GLib.Object {
2      public signal void sig_1(int a);
3
4      public static int main(string[] args) {
5          Test t1 = new Test();
6
7          t1.sig_1.connect((t, a) => { stdout.printf("%d\n", a); });
8
9          t1.sig_1(5);
10
11         return 0;
12     }
13 }

```

```

1  class Person : Object {
2      private int _age = 32; // underscore prefix to avoid name clash
3      // with property
4
5      /* Property */
6      public int age {
7          get { return _age; }
8          set { _age = value; }
9      }
10
11     // Or shorter ...
12     class Person : Object {
13         /* Property with standard getter and setter and default value */
14         public int age { get; set; default = 32; }
15         ...
16         // Read only
17         public int age2 { get; private set; default = 32; }
18     }
19
20     Person alice = new Person();
21     alice.notify["age"].connect (
22         (s, p) => { stdout.printf("age has changed\n"); }
23     );

```

```

1  public abstract class Animal : Object {
2      public void eat() {
3          stdout.printf("*chomp chomp*\n");
4      }
5
6      public abstract void say_hello();
7  }
8
9  public class Tiger : Animal {
10     public override void say_hello() {
11         stdout.printf("*roar*\n");
12     }
13 }
14
15 public class Duck : Animal {
16     public override void say_hello() {
17         stdout.printf("*quack*\n");
18     }
19 }

```



```

1  public interface ITest : GLib.Object {
    public abstract int data_1 { get; set; }
    public abstract void method_1();
2  }
3  ....
4  public class Test1 : GLib.Object, ITest {
    public int data_1 { get; set; }
    public void method_1() {
5      }
6  }
7  }

```

```

1  class SuperClass : GLib.Object {
2      public virtual void method_1() {
3          stdout.printf("SuperClass.method_1()\n");
4      }
5  }
6
7  class SubClass : SuperClass {
8      public override void method_1() {
9          stdout.printf("SubClass.method_1()\n");
10     }
11 }

```

## RunTime Type Information (RTTI)

```

1  bool b = object is SomeTypeName;
    Type type = object.get_type();
2  stdout.printf("%s\n", type.name());
3
4
5  Type type = typeof(Foo);
    Foo foo = (Foo) Object.new(type);

```

## Dynamic type conversion

```

1  Button b = widget as Button;
2  // The last line is equivalent to ....
    Button b = (widget is Button) ? (Button) widget : null;

```

```

1 public class Wrapper<G> : GLib.Object {
2     private G data;
3
4     public void set_data(G data) {
5         this.data = data;
6     }
7
8     public G get_data() {
9         return this.data;
10    }
11 }
12
13 var wrapper = new Wrapper<string>();
14 wrapper.set_data("test");
15 var data = wrapper.get_data();

```

```

1 double method_name(int x, double d)
2     requires (x > 0 && x < 10)
3     requires (d >= 0.0 && d <= 1.0)
4     ensures (result >= 0.0 && result <= 10.0)
5     {
6         return d * x;
7     }

```

Where **result** is a special variable that represents the result.

## Exceptions

```

1 errordomain IOError {
2     FILE_NOT_FOUND
3 }
4
5 void my_method() throws IOError {
6     // ...
7     if (something_went_wrong) {
8         throw new IOError.FILE_NOT_FOUND(
9             "Requested file could not be found.");
10    }
11 }
12
13 try {
14     my_method();
15 } catch (IOError e) {
16     stdout.printf("Error: %s\n", e.message);
17 }
18
19 IOChannel channel;
20 try {
21     channel = new IOChannel.file("/tmp/my_lock", "w");
22 } catch (FileError e) {
23     if (e is FileError.EXIST) {
24         throw e;
25     }
26     GLib.error("", e.message);
27 }

```

## Parameters direction

```

1 void method_1(int a, out int b, ref int c) { ... }
2 void method_2(Object o, out Object p, ref Object q) { ... }
3
4 int a = 1;
5 int b;
6 int c = 3;
7 method_1(a, out b, ref c);
8
9 Object o = new Object();
10 Object p;
11 Object q = new Object();
12 method_2(o, out p, ref q);
13
14 // An implementation of method_1
15 void method_1(int a, out int b, ref int c) {
16     b = a + c;
17     c = 3;
18 }

```

- They are defined outside the nutshell of the language in a library.
- This library is called `Gee` or `libgee`.
- The available collections in Gee are:
  - 1 Lists: Sorted collections of accessible items by a numeric index.
  - 2 Sets: Non-sorted collections.
  - 3 Maps: Non-sorted collections of accessible items by a numeric index or another type.
- Some Gee classes:
  - `ArrayList<G>`
  - `HashMap<K,V>`
  - `HashSet<G>`

```

using Gee;

2
void main () {
4   var list = new ArrayList<int> ();
   list.add (1);
6   list.add (2);
   list.add (5);
8   list.add (4);
   list.insert (2, 3);
10  list.remove_at (3);
   foreach (int i in list) {
12     stdout.printf ("%d\n", i);
   }
14   list[2] = 10; // same as list.set (2, 10)
   stdout.printf ("%d\n", list[2]); // same as list.get (2)
16 }

```

Compile and execute:

```

$ valac --pkg gee-1.0 gee-list.vala
2 $ ./gee-list

```

## Multi-thread support

<https://live.gnome.org/Vala/Tutorial>

```

void* thread_func () {
2   stdout.printf ("Thread running.\n");
   return null;
4 }

6 int main (string[] args) {
   if (!Thread.supported ()) {
8     stderr.printf ("Cannot run without threads.\n");
     return 1;
10  }

12  try {
     Thread.create (thread_func, false);
14  } catch (ThreadError e) {
     return 1;
16  }

18  return 0;
   }

20 // This type of code should be compiled in this way:
22 > valac --thread thread-sample.vala

```

## Interesting links

- [Vala for C# programmers](#)
- [Vala for Java programmers](#)
- [The management of dynamic memory in Vala](#)
- List of [libraries](#) ready for being used in Vala
- Frequent asked questions in Vala: [FAQ](#)
- A video tutorial that shows how easy is to create an application written in Vala with a graphic interface: [video-tutorial](#)
- [Simple examples](#) , [Medium level examples](#) , [examples with strings](#) , [examples with signals and callbacks](#) , [examples with properties](#)