Content: I
Presentation.

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Languages and Computer Science University of Alicante

Content

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.

Teaching guide

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

Contents

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

Assessment

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

- **Atention!** 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

Learning plan (II)

Week	U.	Pres. work	Non-p. work
08	7	Introduction to C# and Web applications	Group assignment
09	8	Layered model	Group assignment
		Evaluation (test)	
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
Total		**60**	**90**

Programming languages used

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 / Clausures
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.

lsi Department of Software and Computing Systems

Reserved words

- 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 modificators: `const`, `weak`, `unowned`, `dynamic`
- Modificators: `abstract`, `virtual`, `override`, `signal`, `extern`, `static`, `async`, `inline`, `new`
- Access Modificators: `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`

Operators

- Arithmetics: +, -, *, /, %
- Bit by bit: ~ , &, |, ^, <<, >>
- Relationals: <, >, <=, >=
- 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
- Owning transfer: (owned)
- Namespaces alias qualifier: :: (currently only with global)
- Others: new, sizeof, typeof, in

Hello World in Vala

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

Compilation

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

Strings

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

```
    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.") ...
```

Input/Output

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

```
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(``...'');`.

Arrays

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

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

Classes

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

```
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   }
```

Type conversion and inference

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

```
1   int   i = 10;
    float j = (float) i;
3
    var p = new Person();       // same as: Person p = new Person();
5   var s = "hello";            // same as: string s = "hello";
    var l = new List<int>();    // same as: List<int> l = new List<int>();
7   var i = 10;                 // same as: int i = 10;

9   MyFoo<string, MyBar<string, int>> foo = new MyFoo<string, MyBar<
        string, int>>();
    // Compare with...
11  var foo = new MyFoo<string, MyBar<string, int>>();
```

Operator ??

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

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

Foreach

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

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

Automatic checking of null values

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

```
string? method_name(string? text, Foo? foo, Bar bar) {
    // ...
}

Object o1 = new Object();      // not nullable
Object? o2 = new Object();     // nullable

o1 = o2; // Forbidden
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  }
```

Clausures

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

```
namespace Hada {
    int n;
}

using Hada;
n = 3;  // Or also...
Hada.n = 3;
```

Visibility

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

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

Constructors/Destructors

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

```
1   public class Button : Object {
      public Button() {
3     }

5     public Button.with_label(string label) {
      }
7
      public Button.from_stock(string stock_id) {
9     }
    }
11
    class Demo : Object {
13    ~Demo() {
        stdout.printf("in destructor");
15    }
    }
```

Signals

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

```
public class Test : GLib.Object {
  public signal void sig_1(int a);

  public static int main(string[] args) {
    Test t1 = new Test();

    t1.sig_1.connect((t, a) => { stdout.printf("%d\n", a); });

    t1.sig_1(5);

    return 0;
  }
}
```

Properties

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

```
1   class Person : Object {
      private int _age = 32;  // underscore prefix to avoid name clash
          with property
3
      /* Property */
5     public int age {
        get { return _age; }
7       set { _age = value; }
      }
9   }

11  // Or shorter ...
    class Person : Object {
13    /* Property with standard getter and setter and default value */
      public int age { get; set; default = 32; }
15    ...
      // Read only
17    public int age2 { get; private set; default = 32; }
    }
19
    Person alice = new Person;
21  alice.notify["age"].connect (
        (s, p) => {stdout.printf("age has changed\n");}
23    );
```

Abstract Classes

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

```
1   public abstract class Animal : Object {
      public void eat() {
3       stdout.printf("*chomp chomp*\n");
      }
5
      public abstract void say_hello();
7   }

9   public class Tiger : Animal {
      public override void say_hello() {
11      stdout.printf("*roar*\n");
      }
13  }

15  public class Duck : Animal {
      public override void say_hello() {
17      stdout.printf("*quack*\n");
      }
19  }
```

Interfaces

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

```
1   public interface ITest : GLib.Object {
      public abstract int data_1 { get; set; }
3     public abstract void method_1();
    }
5   ....
    public class Test1 : GLib.Object, ITest {
7     public int data_1 { get; set; }
      public void method_1() {
9     }
    }
```

Dynamic linking of methods

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

```
class SuperClass : GLib.Object {
  public virtual void method_1() {
    stdout.printf("SuperClass.method_1()\n");
  }
}

class SubClass : SuperClass {
  public override void method_1() {
    stdout.printf("SubClass.method_1()\n");
  }
}
```

RunTime Type Information (RTTI)

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

```
1   bool b = object is SomeTypeName;
    Type type = object.get_type();
3   stdout.printf("%s\n", type.name());

5   Type type = typeof(Foo);
    Foo foo = (Foo) Object.new(type);
```

Dynamic type conversion

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

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

Generic classes

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

```
1   public class Wrapper<G> : GLib.Object {
      private G data;
3
      public void set_data(G data) {
5       this.data = data;
      }
7
      public G get_data() {
9       return this.data;
      }
11  }

13  var wrapper = new Wrapper<string>();
    wrapper.set_data("test");
15  var data = wrapper.get_data();
```

Programming by contract

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

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

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

Exceptions

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

```
errordomain IOError {
  FILE_NOT_FOUND
}

void my_method() throws IOError {
  // ...
  if (something_went_wrong) {
    throw new IOError.FILE_NOT_FOUND(
                          "Requested file could not be found.");
  }
}
...
try {
  my_method();
} catch (IOError e) {
  stdout.printf("Error: %s\n", e.message);
}
...
IOChannel channel;
try {
  channel = new IOChannel.file("/tmp/my_lock", "w");
} catch (FileError e) {
  if(e is FileError.EXIST) {
    throw e;
  }
  GLib.error("", e.message);
}
```

Parameters direction

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

```
1   void method_1(int a, out int b, ref int c) { ... }
    void method_2(Object o, out Object p, ref Object q) { ... }
3
    int a = 1;
5   int b;
    int c = 3;
7   method_1(a, out b, ref c);

9   Object o = new Object();
    Object p;
11  Object q = new Object();
    method_2(o, out p, ref q);
13
    // An implementation of method_1
15  void method_1(int a, out int b, ref int c) {
      b = a + c;
17    c = 3;
    }
```

Collections (I)

- 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>

Collections (II)

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

```
using Gee;

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

Compile and execute:

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

Multi-thread support

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

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

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

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

  return 0;
}

// This type of code should be compiled in this wayi:
> 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