Tema: I
Presentación.

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

lsi Department of Software and Computing Systems

Contenido

Profesores

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

Despachos, Horarios de tutoría, cita previa, etc: www.dlsi.ua.es.

Guía Docente

- **Campus Virtual** $\rightarrow$ **Recursos de aprendizaje** $\rightarrow$ **Guía docente**
- Horarios, objetivos y competencias, contenidos, plan de aprendizaje, evaluación, bibliografía y enlaces
- La asignatura proporciona 1.20 créditos teóricos y 1.20 créditos prácticos.

Contenidos

El temario de la asignatura es el siguiente

- T1 - Presentación, Lenguajes de programación
- T2 - Control de versiones
- T3 - Programación dirigida por eventos y ejecución diferida de código
- T4 - Interfaces gráficas de usuario
- T5 - Acceso a BBDD desde aplicaciones de escritorio
- T6 - Reutilización del código objeto: gestión de bibliotecas
- T7 - Aspectos básicos y despliegue de aplicaciones Web
- T8 - Acceso a BBDD mediante un modelo de objetos
- T9 - Realización de presentaciones efectivas

Evaluación

1. Evaluación Contínua: Práctica individual. Se realizarán 3 prácticas individuales. Práctica 1: 2,5 %, Práctica 2: 7,5 %, Práctica 3: 10 %. **Puntuación: 20 %**.
2. Evaluación Contínua: Test escritorio. Se realizará un test para evaluar los conocimientos de los alumnos de forma individual a mitad de curso. Nota mínima necesaria: 4. **Puntuación: 30 %**.
3. Evaluación Contínua: Práctica en grupo. Se realizará una práctica en grupo sober una aplicación Web de forma colaborativa cuya entrega será a final de curso. Además se debe realizar una exposición de dicha práctica **Puntuación: 30 %**.
4. Evaluación Contínua: Test web. Se realizará un test sobre la parte web en la fecha oficial asignada por la escuela politénica en junio. Nota mínima necesaria: 4. **Puntuación: 20 %**.

lsi Department of Software and Computing Systems

Evaluación sin superar evaluación continua

- **Atención!** En Julio, los alumnos que no superen las actividades de evaluación continua tendrán que realizar un examen cuya puntuación máxima será 50 %.
- Las notas obtenidas en las prácticas durante el curso, no son recuperables. Se mantiene su nota para calcular la nota media en Julio.
- **Para más detalle, en el campus virtual ver documento "Criterios de Evaluación Hada".**

Plan de aprendizaje (I)

Sem.	Ud.	Desc. trab. pres.	Desc. trab. no pres.
01	1	Introducción a la asignatura. Introducción al lenguaje de programación.	-
02	2	Control de versiones	Autopráctica guiada para comprender el entorno de programación.
03	2	Control de versiones	Práctica 1
04	3	Programación dirigida por eventos y ejecución diferida de código	Práctica 1
05	4	Interfaces gráficas de usuario	Práctica 2
06	5	Acceso a BBDD desde aplicaciones de escritorio	Práctica 3
07	6	Bibliotecas.	Práctica 3

Plan de aprendizaje (II)

Sem.	Ud.	Desc. trab. pres.	Desc. trab. no pres.
08	7	Introducción a C# y aplicaciones Web	Práctica en grupo
09	8	Modelo de capas	Práctica en grupo
		Prueba objetiva (test)	
10	7	Capa de interfaz aplicaciones Web	Práctica en grupo
11	7	Capa de interfaz aplicaciones Web (II)	Práctica en grupo
12	8	Acceso a BBDD modo conectado	Práctica en grupo
13	8,9	Acceso a BBDD modo desconectado. Presentaciones efectivas	Práctica en grupo
14	7	Aspectos avanzados en el desarrollo de aplicaciones Web	Práctica en grupo. Exposición oral.
15	1-9	Repaso y dudas	Corrección práctica en grupo
Total		**60**	**90**

Lenguajes de prácticas

1. **Prácticas individuales** Lenguaje Vala.
2. **Práctica en grupo** Lenguaje C# (con ASP.net).

Introducción a Vala

- Vala es un nuevo lenguaje de programación: Vala
- Emplea las funcionalidades proporcionadas por Glib y GObject
- El compilador de Vala genera código '**C**', el cual es compilado por un compilador de **Lenguaje C**.
- Es un lenguaje similar a `Java` y `C#`, más parecido a este último.

Características de Vala

1. POO (clases, clases abstractas, mixin interfaces, polymorphism)
2. Espacios de nombres (namespaces)
3. Delegados
4. Propiedades
5. Señales
6. Notificaciones automaticas de modificación de propiedades
7. Foreach
8. Expresiones Lambda / Clausuras
9. Inferencia de tipos de variables locales
10. Tipos Genericos
11. Tipos No-nulos
12. Gestion automática de memoria dinámica (automatic reference counting)
13. Destructores deterministas (RAII)
14. Excepciones (checked exceptions)
15. Métodos Asíncronos (coroutines)
16. Precondiciones y postcondiciones (programación por contrato)
17. Run-time type information
18. Constructors con nombre
19. Cadenas Verbatim
20. Troceado de arrays y cadenas
21. Compilacion condicional
22. Sintaxis similar a C#
23. Compatibilidad a nivel de ABI con C.

Palabras reservadas

- Selección: `if`, `else`, `switch`, `case`, `default`
- Iteración: `do`, `while`, `for`, `foreach`, `in`
- Salto: `break`, `continue`, `return`
- Excepciones: `try`, `catch`, `finally`, `throw`
- Sincronización: `lock`
- Declaración de tipos: `class`, `interface`, `struct`, `enum`, `delegate`, `errordomain`
- Modificadores de tipos: `const`, `weak`, `unowned`, `dynamic`
- Modificadores: `abstract`, `virtual`, `override`, `signal`, `extern`, `static`, `async`, `inline`, `new`
- Modificadores de acceso: `public`, `private`, `protected`, `internal`
- Parámetros de métodos: `out`, `ref`
- Programación por contrato: `throws`, `requires`, `ensures`
- Espacios de nombres: `namespace`, `using`
- Operadores: `as`, `is`, `in`, `new`, `delete`, `sizeof`, `typeof`
- Acceso: `this`, `base`
- Literales: `null`, `true`, `false`
- Propiedades: `get`, `set`, `construct`, `default`, `value`
- Bloques constructores: `construct`, `static construct`, `class construct`
- Otras: `void`, `var`, `yield`, `global`, `owned`

Operadores

- Aritméticos: +, -, *, /, %
- Bit a bit: ~ , &, |, ^, <<, >>
- Relacionales: <, >, <=, >=
- Igualdad: ==, !=
- Lógicos: !, &&, ||
- Asignación: =, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
- Incremento, Decremento: ++, –
- Punteros: &, *, ->, delete
- Condicionales: ?:
- Comparación con null: ??
- Concatenación de cadenas: +
- Invocación de métodos: ()
- Acceso a miembros: .
- Indice: [ ]
- Troceado: [:]
- Lambda: =>
- Casting: (Type), (!), as
- Comprobación de tipos en tiempo de ejecución: is
- Transferencia de propiedad: (owned)
- Cualificador de alias de espacios de nombres: :: (currently only with global)
- Otros: new, sizeof, typeof, in

Hola Mundo en Vala

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

Compilación

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

Cadenas

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.") ...
```

Cadenas

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());
```

- También disponemos de la salida de error estándar representada por "`stderr`".
- Podemos mostrar información en ella con "`printf`" así: `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;
```

Clases

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   }
```

Conversión e inferencia de tipos

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>>();
    // Compara con...
11  var foo = new MyFoo<string, MyBar<string, int>>();
```

Operador ??

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); }
```

Comprobación automática de valores nulos

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

```
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; // Prohibido
9   o1 = (!) o2; // Permitido con el cast non-null explicito: operador !
```

Delegados

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  }
```

Clausuras

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

```
delegate void PrintIntFunc(int a);

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

Espacios de nombres

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

```
namespace Hada {
    int n;
}

using Hada;
n = 3;   // O tambien...
Hada.n = 3;
```

Visibilidad

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

public	Sin restricciones de acceso
private	Acceso limitado desde dentro de la definicón de la clase o estructura. Este es el acceso por defecto si no se dice nada.
protected	Acceso limitado desde dentro de la definicón de la clase o estructura y desde cualquier clase que derive de ella.
internal	Acceso limitado desde clases definidas en el mismo paquete

Constructores/Destructores

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    }
    }
```

Señales

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;
  }
}
```

Propiedades

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  // O mas resumido...
    class Person : Object {
13    /* Property with standard getter and setter and default value */
      public int age { get; set; default = 32; }
15    ...
      // De solo lectura
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    );
```

Clases abstractas

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     }
    }
```

Enlace dinámico de métodos

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");
  }
}
```

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);
```

Conversiones de tipo dinámicas

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

```
    Button b = widget as Button;
2   // Lo anterior equivale a....
    Button b = (widget is Button) ? (Button) widget : null;
```

Clases genéricas

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();
```

Programación por contrato

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   }
```

Donde **result** es una variable especial que representa el resultado.

Excepciones

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);
}
```

Dirección de los parámetros

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
    // Una implementacion de method_1
15  void method_1(int a, out int b, ref int c) {
      b = a + c;
17    c = 3;
    }
```

Colecciones (I)

- Se definen fuera del núcleo del lenguaje en una biblioteca.
- Esta biblioteca se llama Gee o libgee.
- Las colecciones disponibles en Gee son:
  1. Lists: Colecciones ordenadas de items accesibles por un índice numérico.
  2. Sets: Colecciones no ordenadas.
  3. Maps: Colecciones no ordenadas de items accesibles por un índice numérico o de otro tipo.
- Algunas clases de Gee:
  - ArrayList<G>
  - HashMap<K,V>
  - HashSet<G>

Colecciones (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)
}
```

Compilar y ejecutar:

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

Soporte multi-thread

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;
}

// Este tipo de codigo se debe compilar asi:
> valac -- thread thread_sample.vala
```

Enlaces de interés

- Vala para programadores en C#
- Vala para programadores en Java
- La gestión de memoria dinámica en Vala
- Lista de bibliotecas preparadas para ser usadas desde Vala
- Preguntas frecuentes en Vala: FAQ
- Un tutorial en vídeo que muestra lo sencillo que es crear una aplicación en vala con interfaz gráfico: video-tutorial
- Ejemplos sencillos , ejemplos de nivel medio , ejemplos con cadenas , ejemplos con señales y callbacks , ejemplos con propiedades