

Elementos de Agrupación

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2023-2024)

Créditos

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
 - ▶ Emojis, <https://pixabay.com/images/id-2074153/>
- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

Objetivos

- Saber crear y usar paquetes en Java
- Saber crear y usar módulos en Ruby
- Gestionar correctamente las líneas `require_relative` en Ruby

Contenidos

- 1 Paquetes Java
- 2 Módulos Ruby
- 3 Un proyecto Ruby definido en varios archivos

Paquetes Java

- Permiten agrupar clases
- Constituyen un espacio de nombres
 - ▶ Es posible tener varias clases que se llamen igual, pero en paquetes distintos
- Existe una visibilidad (de paquete) que otros lenguajes no tienen
- Uso:
 - ▶ Para indicar que los elementos definidos en un archivo pertenecen a un paquete, en dicho archivo se añadirá el nombre del paquete (comenzando con minúscula)
 - ▶ Para usar elementos de un paquete distinto al actual, hay que indicarlo
 - ▶ En disco, un paquete aparece como una carpeta del sistema de ficheros
Los archivos de los elementos que pertenecen a un paquete estarán en su carpeta

Ejemplo: Paquetes Java

```
1 package miPrograma ;  
2 // Los elementos que se definan en este archivo pertenecerán al paquete miPrograma  
3 import modelo.Fachada; // En este fichero se va a usar la clase Fachada del paquete modelo
```

Paquetes Java

- Cada paquete Java es independiente del resto aunque a nivel de nombrado (y de almacenamiento en disco) parezca que uno es subpaquete de otro
 - ▶ En Java NO existen subpaquetes

Paquete: view

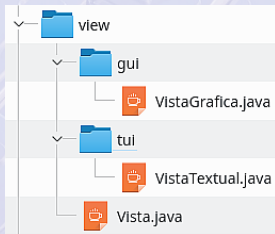
```
1 package view;
2 public interface Vista {
3     . . .
4 }
```

Paquete: view.gui

```
1 package view.gui;
2 import view.Vista;
3 class VistaGrafica
4     implements Vista {
5     . . .
6 }
```

Paquete: view.tui

```
1 package view.tui;
2 import view.Vista;
3 class VistaTextual
4     implements Vista {
5     . . .
6 }
```



← Estructura de carpetas y archivos

★ ¿Se puede quitar la palabra public?

Módulos Ruby

- **Permiten agrupar una gran variedad de elementos:** clases, constantes, funciones, otros módulos, etc.
- Constituyen un espacio de nombres
- **Uso:**
 - ▶ Para incluir un elemento en un módulo: se abre el módulo, se realiza la definición, y se cierra el módulo
 - ▶ Para utilizar un elemento de un módulo distinto al actual hay que anteponer `<NombreModulo>::`
 - ▶ Se puede **copiar** todo el contenido de un módulo dentro de una clase (`include`)
- En Ruby sí puede haber módulos dentro de módulos
 - ▶ Se accede a los símbolos encadenando nombres de módulos y `::`
Ejemplo: `objeto = ModuloExterno::ModuloInterno::Clase.new`

Módulos Ruby

Ejemplo: Ruby

```
1 module Externo
2   class A
3   end
4
5   module Interno
6     class B
7     end
8   end
9 end
10
11 module Test
12   def test
13     puts "Testeando"
14   end
15 end
16
17 class C
18   include Test # Literalmente, se copia el contenido del módulo Test
19 end
20
21 a = Externo::A.new
22 b = Externo::Interno::B.new
23 c = C.new
24 c.test
```



Un proyecto Ruby definido en varios archivos

- Normalmente, en cualquier lenguaje, cada clase que forma parte de un proyecto se define en un archivo distinto

★ Buenas prácticas de programación

- En los lenguajes compilados, se procesan todos los archivos fuentes (se compilan) antes de ejecutar el programa principal

★ ¿Habéis usado `Makefile` en C++?

★ ¿Sabéis lo que es una tabla de símbolos?

- Ruby es interpretado, **Ruby**:

No sabe que un proyecto está formado por varios archivos

No realiza un procesamiento previo que identifique las clases

- ▶ Si en un archivo mencionamos una clase definida en otro archivo, **nos dará un error si no nos hemos preocupado nosotros** de que procese las clases antes de usarlas

Referenciando archivos Ruby

- Cuando se ejecuta `ruby archivo_principal.rb` se va procesando este archivo línea a línea
- Si en este archivo se menciona la clase `A`, debemos haberle indicado a Ruby que previamente haya procesado el archivo que contiene la definición de la clase `A`
- Lo hacemos con las instrucciones:
 - ▶ `require` se suele usar para archivos del lenguaje
 - ▶ `require_relative` se suele usar para archivos propios
- **Criterio:**
 - ▶ Cuando en un archivo aparece el nombre de una clase, se añade un `require_relative` al archivo que define esa clase
- Ruby anota qué archivos ha cargado y no los carga dos veces

Ejemplo

: cosa.rb

```

1 class Cosa
2   @@Maximo = 3
3
4   attr_reader :nombre
5
6   def initialize (unNombre)
7     @nombre = unNombre
8   end
9
10  def self.Maximo
11    @@Maximo
12  end
13 end

```

: principal.rb

```

1 require_relative 'cosa' # por línea 4
2 require_relative 'persona' # por línea 5
3
4 mochila = Cosa.new("Mochila")
5 juan = Persona.new("Juan")
6 juan.otraCosaMas (mochila)
7 puts juan.to_s

```

: persona.rb

```

1 require_relative 'cosa' # por línea 4
2
3 class Persona
4   @@MaximoPermitido = Cosa.Maximo
5
6   def initialize (unNombre)
7     @nombre = unNombre
8     @cosas = []
9   end
10
11  def otraCosaMas (unaCosa)
12    if @cosas.size < @@MaximoPermitido
13      @cosas << unaCosa
14    end
15  end
16
17  def to_s
18    salida = "Me llamo #{@nombre} y
19             tengo:\n"
20    for unaCosa in @cosas do
21      salida += "- #{unaCosa.nombre}\n"
22    end
23    salida
24  end
25 end

```

Mal ejemplo

- Algunos estudiantes añaden `require_relative` de todos los archivos en todos los archivos
- Esa mala práctica, más pronto que tarde, produce errores



: cosa.rb

```
1 # Se añade un require_relative innecesario
2 # No se menciona la clase Persona en este archivo
3
4 require_relative 'persona'
5
6 class Cosa
7   # La clase se define igual que en el ejemplo anterior
8 end
9 # No cambia nada más en ningún otro archivo
```

Ejecución: Mensaje de error obtenido

```
persona.rb:4:in '<class:Persona>': uninitialized constant Persona::Cosa (NameError)
```

★ Trazémoslo y averigüemos el porqué

- ¿Cuándo debo agrupar clases en paquetes o módulos?
 - ▶ No hay una respuesta única
 - ▶ Normalmente se agrupan clases que tienen una relación entre ellas
 - ▶ En la asignatura *Fundamentos de Ingeniería del Software* profundizaréis más en cuestiones de diseño como esta

Elementos de Agrupación

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2023-2024)

Temporary page!

\LaTeX was unable to guess the total number of pages correctly. There was some unprocessed data that should have been added to the next page. This extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will be removed, because \LaTeX now knows how many pages to expect for this document.