Sesión 5: Elementos de Agrupación

PAQUETES EN JAVA

Estos paquetes permiten **agrupar las clases**, de hecho se pueden tener clases con el mismo nombre **siempre y cuando estén en distintos paquetes**. Tienen visibilidad de paquete.

Para **crear un paquete:**

```
package nombre_paquete;
```

Además también podemos crear varios paquetes, para ello cada nombre se separa con un punto:

```
package nombre_paquete1.nombre_paquete2. ... .nombre_paqueteN;
```

En el siguiente ejemplo podemos ver un paquete:

```
package packLibros;

public class Libro {
    private String titulo;
    private int num_paginas;

    Libros(String titulo, int paginas){
        this.titulo = titulo;
        num_paginas = paginas;
    }

    void mostrar(){
        System.out.println(titulo);
        System.out.println(num_paginas);
    }
}
```

Para usar esta clase en otra, importo el fichero:

```
import modulo;
```

Ahora se muestran unos paquetes que a veces son útiles pertenecientes a Java:

Package	Descripción del contenido
`java.lang`	Contiene las clases e interfaces más empleadas en la mayoría de los programas de Java. Es importado automáticamente por todos los programa Java: no se necesita sentencia `import` para utilizar lo declarado en este paquete.
`java.io`	Contiene clases que permiten las operaciones de entrada y salida de datos de un programa.
`java.util`	Contiene clases e interfaces de utilidades: operaciones con la fecha y la hora, generación de números aleatorios...
`java.applet`	Contiene todas las clases e interfaces necesarias para la construcción de *applets* de Java
`java.net`	Contiene clases que permite a un programa comunicarse a traves de redes (Internet o intranet)
`java.text`	Contiene clases e interfaces que permiten operaciones de números, fechas, caracteres y cadenas.
`java.awt`	Es el paquete *Abstract Windowing Toolkit*. Contiene muchas clases e interfaces necesarias para trabajar con la interfaz de usuario gráfica *clásica*.
`java.beans`	Contiene clases para facilitar a los programadores la generación de componentes de software reutilizables.

PAQUETES EN RUBY

Permite agrupar una gran variedad de elementos.

La forma de usar los paquetes en Ruby es la siguiente:

- Para incluir un elemento en un módulo, se abre, se incluye la definición y se cierra dicho módulo.

```ruby
module pkg
    class A
    end
end
```

- Para utilizar un elemento de un módulo en otro módulo se pone **::**

```ruby
module Test
    class A
    end
end

a = Test::A.new # Utilizando un elemento del modulo Test
```

- Se puede copiar todo el contenido de un módulo dentro de una clase y para ello se utiliza **include**

```ruby
module Test
    def test
        puts "Hola"
    end
end

class C
    include Test #Se copia todo el contenido del módulo test
end
```

- Se puede hacer un módulo dentro de otro **encadenando ::**

```ruby
module Externo
    class A
    end

    module Interno
        class B
        end
    end
end

b = Externo::Interno::B.new #Estoy llamando a new que está dentro de dos módulos
```

Ruby es un lenguaje **interpretado**, lo que significa que:

- No sabe que un proyecto está formado por varios archivos.
- No realiza un procesamiento previo que identifique las clases.

Por lo que para indicarle a Ruby lo que debe usar:

- **require:** se suele usar para archivos del lenguaje.
- **require_relative:** se suele usar par archivos propios.

```ruby
class Cosa
    @@Maximo = 3 # Atributo de clase que indica que el máximo de objetos es 3

    attr_reader :nombre # Lectura
    def initializa (unNombre)
        @nombre = unNombre #Atributo de instancia
    end

    def self.Maximo
        @@Maximo #Metodo de clase de lectura del atributo Maximo
    end
end
```

```ruby
require_relative 'cosa'

class Persona
    @@MaximoPermitido = Cosa.Maximo #Atributo de clase
```

```
    def initialize (unNombre)
        @nombre = unNombre
        @cosas = []
    end

    def otraCosaMas(unaCosa)
        if cosas.size < @@MaximoPermitido #Si no supera el nº permitido
            @cosas << unaCosa #Añadir al array una cosa
        end
    end

    def to_s
        salida = "Me llamo #{@nombre} y tengo: \n"
        for unaCosa in @cosas do
            salida += "-- #{unaCosa.nombre} \n"
        end
        salida
    end
end
```

```
require_relative 'cosa' #Importamos la clase cosa y la clase persona
require_relative 'persona'

mochila = Cosa.new("Mochila")
juan = Persona.new("Juan")
juan.otraCosaMas(mochila)
puts juan.to_s
```