

Consultores y Modificadores

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2021-2022)

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 consultores y modificadores, tanto en Java como en Ruby
- Ser conscientes de la problemática de devolver o asignar referencias a objetos

Contenidos

- 1 Consultores
- 2 Modificadores
- 3 Ejemplos
 - Java
 - Ruby
- 4 Problemática de devolver (o asignar) referencias

Consultores

- Métodos encargados de **devolver el valor de un atributo**
- No tienen necesariamente que limitarse a devolver ese valor.
Pueden devolverlo modificado, o una copia del mismo, etc.
por ejemplo, trabajo con tanto por 1 pero devuelvo en tanto por ciento
- Pueden ser de clase o de instancia
- Habitualmente se nombran: `getAtributo()` en Java
- Habitualmente se nombran: `atributo` en Ruby
- Solo deben crearse los consultores que realmente sean necesarios *por ej. en Dice no hace falta, no voy a consultar del nombre en Java*
 - ▶ **Se expone el estado interno al exterior**

★ ¿Se pueden usar dentro de los constructores?

No se debe

Modificadores

- Métodos encargados de **modificar el valor de un atributo**
 - No tienen necesariamente que limitarse a fijar ese valor.
Pueden y deben controlar las restricciones sobre ese atributo
(un valor entre 0 y 100, por ej). Modularizar estas operaciones para no repetir código en < ctor modif.
 - Pueden ser de clase o de instancia
 - Habitualmente se nombran: `setAtributo(...)` en Java
 - Habitualmente se nombran: `atributo` en Ruby
 - Solo deben crearse los modificadores que realmente sean necesarios *En casos en los que sea muy claro, se penalizará ctors y modif. innecesarios.*
 - ▶ **Se expone el estado interno al exterior**
- ★ ¿Se pueden usar dentro de los constructores?

Ejemplos

Java: Consultores y modificadores

```
1 public class Persona {
2
3     private static final int MAYORIAEDAD = 18; // Atributo de clase
4     private LocalDateTime fechaNacimiento;      // Atributo de instancia
5
6     Persona (LocalDateTime fecha) {
7         fechaNacimiento = fecha;
8     }
9
10    public static int getMayoriaEdad() {
11        return MAYORIAEDAD;
12    }
13
14    public LocalDateTime getFechaNacimiento() {
15        // Se devuelve al exterior una referencia a la fecha de nacimiento
16        // Podría ser modificada desde fuera
17        return fechaNacimiento;
18    }
19
20    public void setFechaNacimiento(LocalDateTime fecha) {
21        // Añadir comprobaciones relativas a las restricciones sobre la edad
22        // Se está asignando una referencia a un objeto que ya está siendo referenciado
23        // desde fuera de la clase
24        fechaNacimiento = fecha;
25    }
26 }
```

Ejemplos

Java: Usando la clase anterior

```
1 Persona p=new Persona(LocalDate.of (2000,7,5,0,0));  
2  
3 // utilizamos el modificador  
4 p.setFechaNacimiento(LocalDate.of (1950,7,5,0,0));  
5  
6 // utilizamos el consultor  
7 System.out.println (p.getFechaNacimiento());  
8  
9 // utilizamos el consultor de clase  
10 System.out.println (Persona.getMayorEdad());
```


Ejemplos

Ruby: Consultores y modificadores

```

1 require 'date'
2
3 class Persona
4
5   @@MAYORIA_EDAD = 18 # Atributo de clase
6
7   def initialize (fecha)
8     @fecha_nacimiento = fecha
9   end
10
11   attr_reader :fecha_nacimiento # consultor
12   attr_writer :fecha_nacimiento # modificador
13   attr_accessor :fecha_nacimiento # consultor + modificador
14
15   def self.MAYORIA_EDAD=e # modificador de clase
16     @@MAYORIA_EDAD = e
17   end
18 end

```

si queremos más, ponemos ':' y más atributos. En vez de '@', ponemos '!'
esto solo para los atributos de instancia
para que sea de clase
para los de clase

También puedo hacerlos a mano si busco una funcionalidad distinta a la de por defecto.

consultor

```

def FN
  Date.new(@FN)
end

```

→ devuelve otra instancia en el mismo estado pero de identidad distinta

modificador

```

def atributo = 60
  @atributo = a
end

```

//main
objeto.atributo = 7

Ejemplos

Ruby: Consultores y modificadores

```
1 require 'date'
2
3 class Persona
4
5   @@MAYORIA_EDAD = 18 # Atributo de clase
6
7   def initialize (fecha)
8     @fecha_nacimiento = fecha
9   end
10
11
12   def fecha_nacimiento # consultor
13     # Se devuelve al exterior del objeto una referencia a la fecha
14     @fecha_nacimiento
15   end
16
17   def fecha_nacimiento=fecha # modificador
18     # ¿ Restricciones ?
19     # Se asigna una referencia a un objeto que ya es referenciado desde fuera
20     @fecha_nacimiento = fecha
21   end
22
23   def self.MAYORIA_EDAD=e # modificador de clase
24     @@MAYORIA_EDAD = e
25   end
26 end
```

Ejemplos

Ruby: Usando la clase anterior

```
1 p=Persona.new(Date.new(2000,7,3))
2
3 # utilizamos el modificador
4 p.fecha_nacimiento=Date.new(2000,8,3)
5
6 # utilizamos el consultor
7 puts p.fecha_nacimiento
8
9 # utilizamos el modificador de clase
10 Persona.MAYORIA_EDAD=21
```

Ejemplos

Ruby: Consultores y modificadores implícitos

```
1 class UnaClase
2
3   attr_reader    :atr1
4   attr_accessor  :atr2
5   attr_writer    :atr3
6
7   def initialize (un, dos, tres)
8     @atr1 = un
9     @atr2 = dos
10    @atr3 = tres
11  end
12
13 end
14
15 obj = UnaClase.new(1,2,3)
16 obj.atr2 = 8 modif
17 puts obj.inspect
18 obj.atr2 = 9 modif
19 puts obj.inspect
20 obj.atr3 = 7 modif
21 puts obj.inspect
22 puts obj.atr1 consultor
23 puts obj.atr2 consultor
24 #puts obj.atr3 # no existe consultor
25 #obj.atr1 = 23 # no existe modificador
```

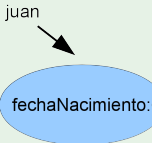
Problemática de devolver (o asignar) referencias

Java: Asignación y devolución de referencias

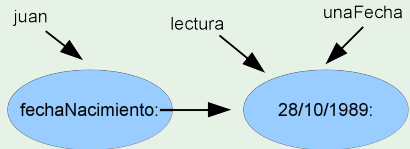
```

1 class Persona {
2     private GregorianCalendar fechaNacimiento;
3
4     Persona (GregorianCalendar nace) {
5         fechaNacimiento = nace;
6     }
7
8     GregorianCalendar getFechaNacimiento () {
9         return fechaNacimiento;
10    }
11
12    // . . .
13 }
14
15 GregorianCalendar unaFecha = new GregorianCalendar (1989,10,28);
16
17 Persona juan = new Persona (unaFecha);
18 System.out.println (juan.toString()); // Nací el 28/10/1989
19
20 GregorianCalendar lectura = juan.getFechaNacimiento();
21 lectura.set (1985,5,13);
22 System.out.println (juan.toString()); // Nací el 13/5/1985
23 unaFecha.set (2001,1,1);
24 System.out.println (juan.toString()); // Nací el 1/1/2001

```



A diagram illustrating the state of the program. A variable named 'juan' is shown with an arrow pointing to a blue oval representing a 'Persona' object. Inside the oval, the field 'fechaNacimiento' is labeled, indicating that 'juan' holds a reference to an object whose birth date is stored in this field.



★ ¿Soluciones?

Consultores y Modificadores

→ **Diseño** ←

- Crear solo los que sean realmente necesarios
- Tener en cuenta si se devuelven (o se asignan) referencias
 - ▶ En lenguajes como Java o Ruby todas las variables son referencias (punteros)
 - ★ Salvo los tipos primitivos: int, float, ...
 - Los String tampoco deben preocuparnos
- No hay una regla a aplicar en todos los casos
 - ▶ A veces interesa devolver (o asignar) una referencia
 - ▶ Otras veces interesa devolver (o asignar) una copia
 - ▶ Puede depender de *a quién* se le dé (o *de dónde*) venga
- Hay que decidirlo evaluando cada situación



Java: Asignación de referencias

```
1 objeto = new Clase();
2 otroObjeto = objeto;
3 // Un ÚNICO objeto con dos referencias
```

Ruby: Asignación de referencias

```
1 objeto = Clase.new;
2 otroObjeto = objeto;
3 # Un ÚNICO objeto con dos referencias
```

Consultores y Modificadores

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2021-2022)