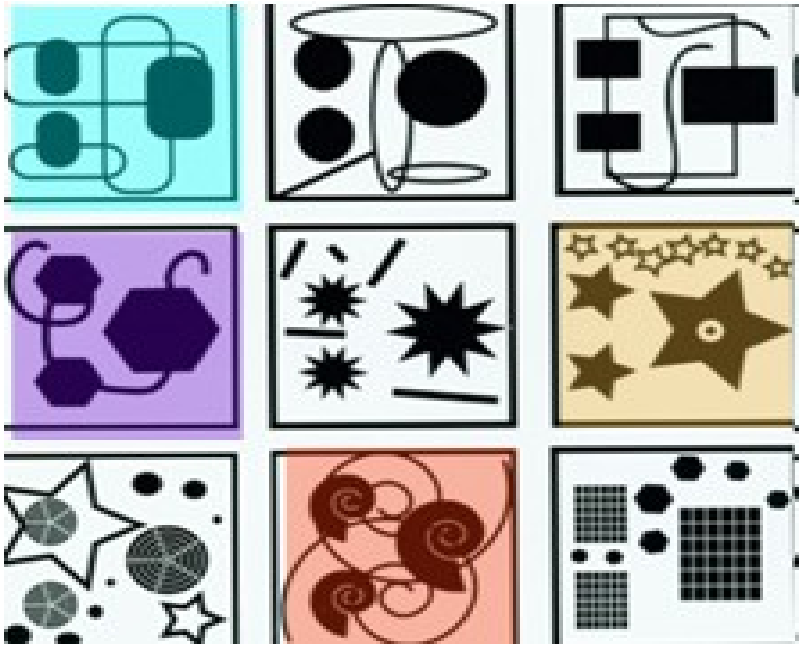
Tema 4



Clases, objetos y mensajes

Lección 4.4

Metaclases y reflexión

Objetivos de aprendizaje

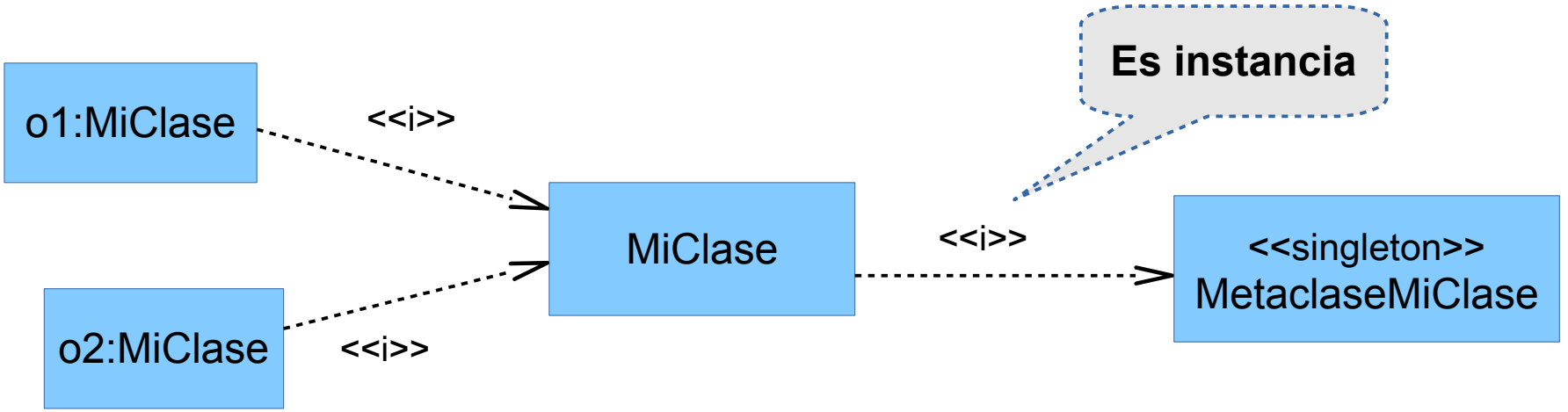
- Conocer el concepto de **metaclase**, tratando las clases como objetos.
- Entender el concepto de **reflexión** aplicado a las clases de un programa orientado a objetos.

Contenidos

1. Concepto y contenidos de metaclase
2. Niveles de implementación del concepto de metaclase
3. Reflexión: definición y mecanismos
4. Reflexión en Java
5. Reflexión en Ruby

1. Concepto y contenidos de metaclase

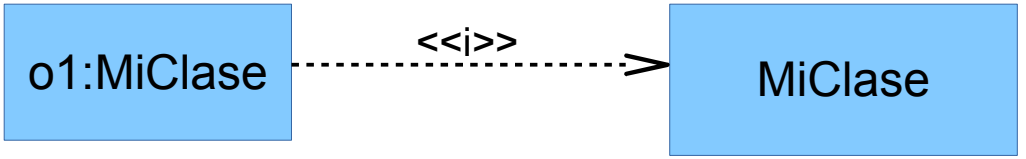
- Una **metaclase es la clase de una clase**, de la misma forma que un objeto es la instancia de una clase, una clase es instancia de una metaclase.
- Una metaclase describe las **propiedades de la clase** que es instancia suya, y puede contener:
  - **Atributos** cómo: Lista de atributos y métodos de instancia, lista de variables y métodos de clase...
  - **Métodos** para: Consultar y modificar las lista anteriores, constructores de instancia...
- Esquema general de las relaciones objeto-clase-metaclase:



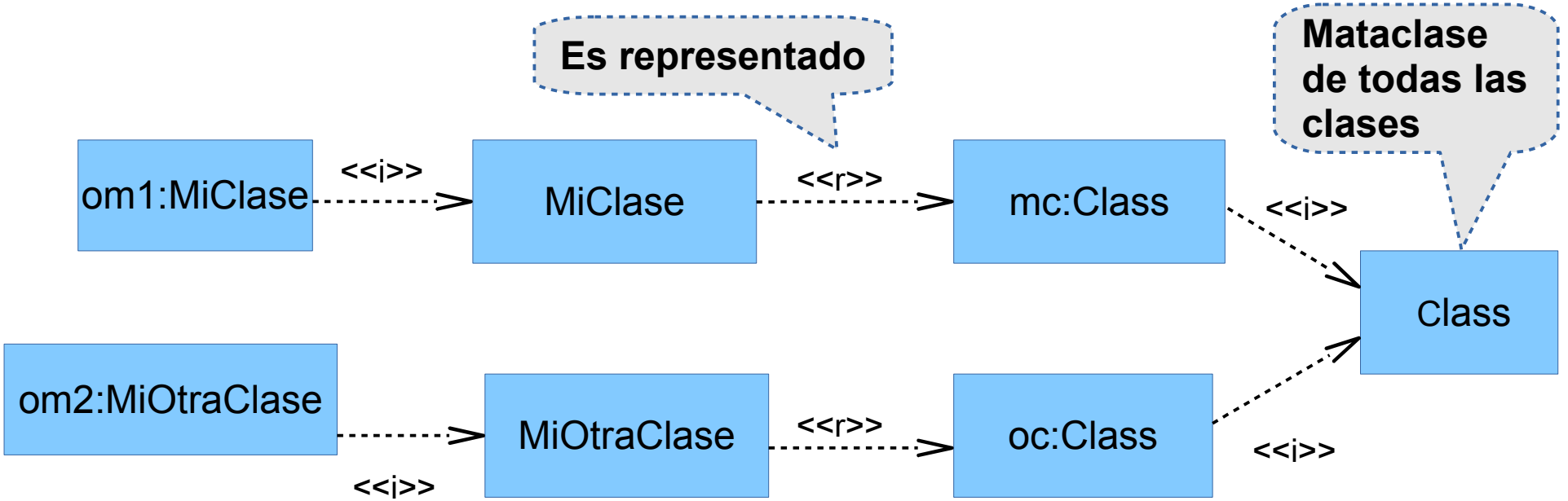
2. Niveles de implementación del concepto de metaclase

Existen varios niveles según el soporte que proporcionan los lenguajes de programación OO para el concepto de metaclase:

Nivel 0: Objetos y clases, sin soporte para metaclases, p. ej. C++

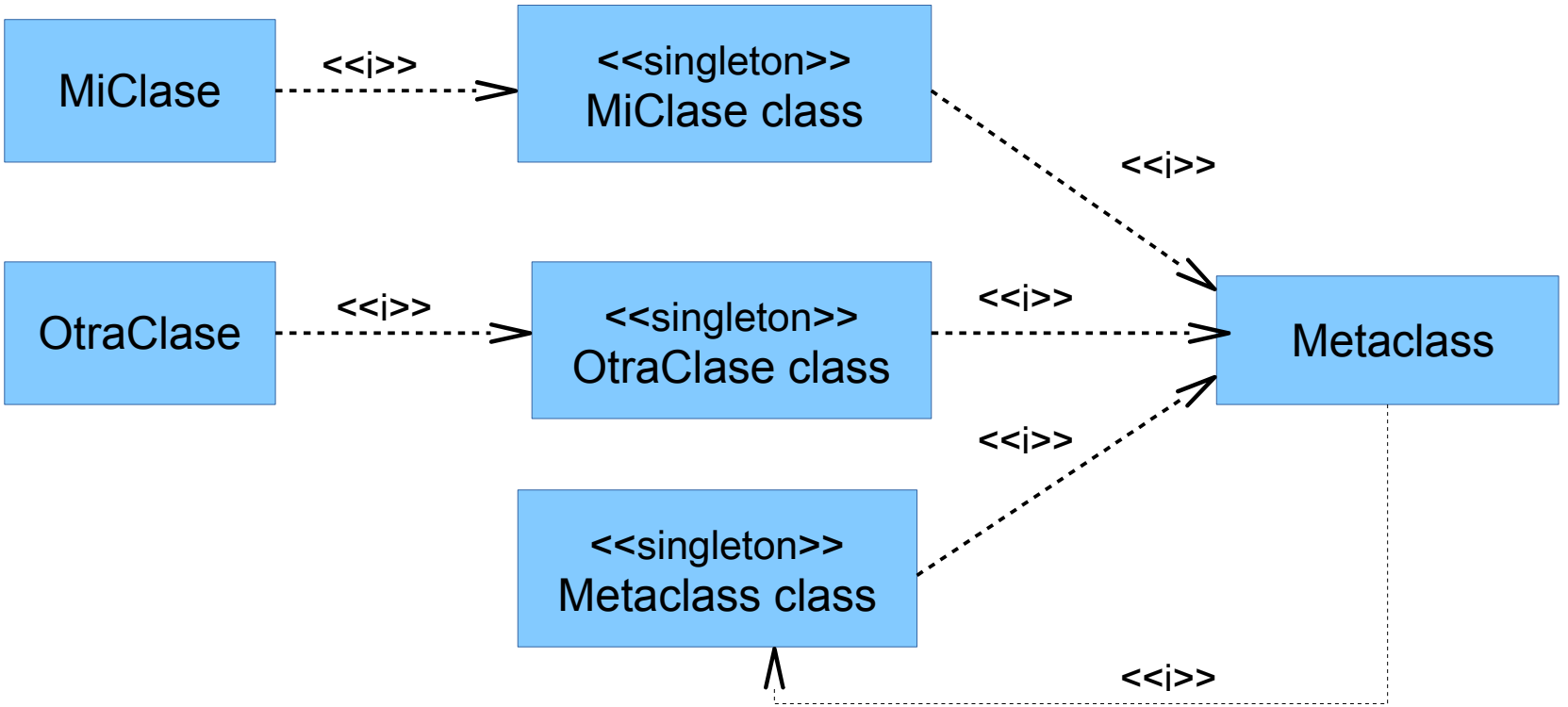


Nivel 1: Objetos, clases y una metaclase para todas las clases, p. ej. **Java**



2. Niveles de implementación del concepto de metaclase

Nivel 2: Objetos, clases, una metaclase por cada una de las clase, por. ej. Smalltalk

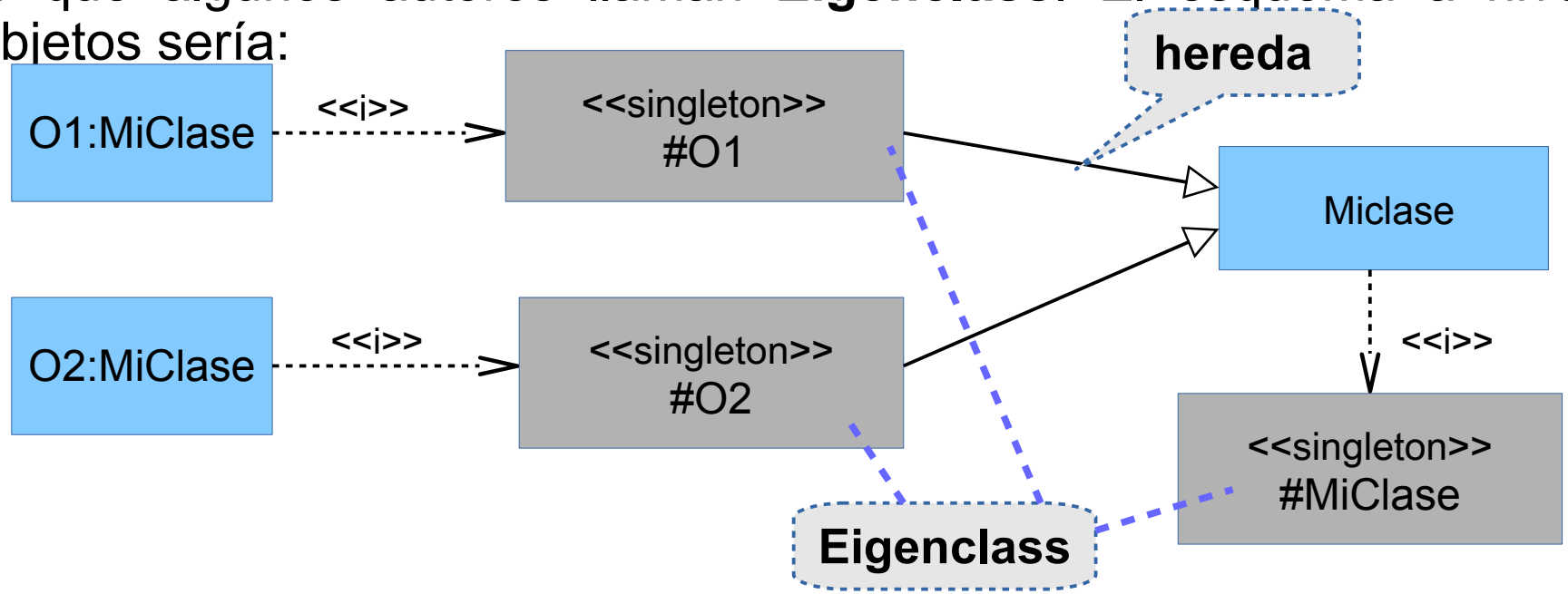


2. Niveles de implementación del concepto de metaclase

Nivel 3: Para las clases el esquema anterior, pero además los objetos de forma individual tienen su "metaclase" asociada, que hace que sean diferentes unos de otros en la misma clase, p. ej. Ruby. Por esto podemos hacer en Ruby lo siguiente:

```
mc = MiClase.new
def mc.miMetodo # método solo del objeto mc
  ...
end
```

Esta "metaclase" o "singleton", tanto a nivel de objeto como de clase, es lo que algunos autores llaman **Eigenclass**. El esquema a nivel de objetos sería:



3. Reflexión: definición y mecanismos

La reflexión es la capacidad de un programa para **manipularse a sí mismo** y comprender sus propias estructuras **en tiempo de ejecución**.

Hay 2 mecanismos principales:

Introspección: Habilidad del programa para observar y razonar sobre su mismo estado (objetos y clases) en tiempo de ejecución.

Modificación: Habilidad del programa para cambiar su estado (objetos y clases) durante la ejecución.

	Introspección	Modificación
C++	NO	No
Java	Sí	No
Python	Sí	Sí
PHP	Sí	Sí
Smalltalk	Sí	Sí
Ruby	Sí	Sí

4. Reflexión en Java

Debido a la estructura de metaclases desarrollada por Java, el nivel de reflexión que se permite es de introspección. Toda la funcionalidad para ello está definida en la clase ***Class*** de Java, ejemplos:

Si tenemos:

MiClase obj = new MiClase();

Class class = obj.getClass() //método definido Object

Conocida la clase de un objeto podemos consultar:

Field[] varInstancia = clase.getFields();

Constructor[] construct = clase.getConstructors();

Method[] metodosInstancia = clase.getMethods();

String nombreClase = clase.getSimpleName();

...

Para más información : https://docs.oracle.com/javase/7/docs/api/java/lang/Class.html

5. Reflexión en Ruby

Debido a la estructura de metaclases desarrollada por Ruby, el nivel de reflexión que se permite es de introspección y de modificación. Pudiéndose hacer en ejecución:

- Consultar y modificar una clase.
- Consultar y modificar la estructura y funcionalidad de un objeto haciéndolo distinto de los demás de la misma clase.

- **Modificación de una clase** (**class_eval**), ejemplo:

```
class Libro
      def initialize(titulo)
      @titulo = titulo
      end
end

libro1 = Libro.new("El señor de los anillos")

Libro.class_eval do def publicacion(añopublicacion)
              @añoPublicacion = añopublicacion
            end
end

libro1.publicacion(1997)
puts libro1.inspect
```

Resultado:

#<Libro:0x266d2a46 @titulo="El señor de los anillos", @añoPublicacion=1997>

5. Reflexión en Ruby

- **Modificación de una instancia** (**instance_eval**), (siguiendo con el mismo ejemplo):

```
libro1.instance_eval do def autor(autor)
                                    @autor = autor
                                end
end
libro1.autor("J. R. R. Tolkien") # hay que enviar este mensaje para
                                    # que el atributo @autor esté en libro1
puts libro1.inspect
```

Resultado:
#<Libro:0x2a66b0df @autor="J. R. R. Tolkien", @titulo="El señor de los anillos", @añoPublicacion=1997>

- Para otra instancia:

```
libro2 = Libro.new("Cien años de soledad")
libro2.autor("G. García Márquez")
```

Resultado:
undefined method `autor' for #<Libro:0x42d38409 @titulo="Cien años de soledad">

5. Reflexión en Ruby

- Introspección en Ruby, ejemplos:

```
puts  Libro.instance_methods(false) # publicacion
puts libro1.instance_variables          # @autor
                                        # @titulo
                                        # @añoPublicacion
puts libro2.instance_variables          # @titulo
puts libro1.instance_of?(Libro)    # true
puts libro1.singleton_methods(false)  # autor
```

-false: Sólo métodos en la clase
-true o nada: métodos en las superclases también

- true o nada: también métodos de módulos incluidos en el objeto
- false: no incluye los de los módulos

Ver todos los métodos de consulta en : http://ruby-doc.org/core-2.1.5/Object.html