PDOO temas 1 y 2

Conceptos básicos

Envío de mensajes entre objetos

Diferentes formas de ligar un mensaje al método que lo resuelve

Estática	Ocurre antes de la ejecución, más eficiente
Dinámica	Ocurre durante la ejecución, más flexible, la usan Java y Ruby

Especificadores de acceso

Java

Visible en	Mismo paquete		Otro paquete
	Clase	Otra	Otra
private	✓		
package	✓	✓	
protected	✓	✓	
public	✓	✓	✓

Se utiliza *protected* por defecto. Hay introspección, ya que se puede consultar una clase u objeto en ejecución.

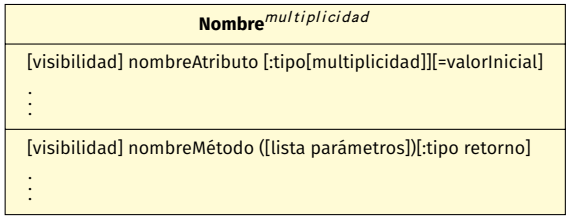
Ruby

Visible en	Desde el propio objeto	Clase	Otra
private	✓		
protected	✓	✓	
public	✓	✓	✓

Se utiliza *private* por defecto. Se pueden consultar y modificar las clases, métodos y variables en ejecución.

Diagramas UML

Diagramas de clases

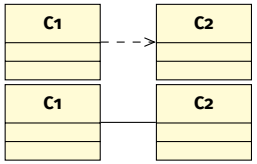


La multiplicidad de la clase indica el número de instancias que puede tener. La de un atributo, el número de elementos que tiene. La descripción de la responsabilidad de la clase puede indicarse debajo. La visibilidad se marca mediante:

+	Pública
-	Privada
~	Paquete
#	Protegida

Para indicar un atributo o método de clase, lo subrayamos.

Relaciones entre clases

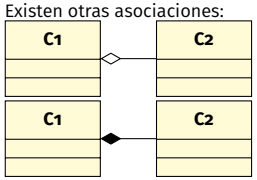


Relación de dependencia: En algún método de C1 se usa un elemento de C2.

Relación de asociación

En una relación de asociación se puede indicar el número de objetos que se asocian de cada clase escribiendo encima de la línea con la sintaxis *vMin..vMax*.

Existen otras asociaciones:



Agregación. Una clase representa el 'todo' (C1) y otra las 'partes' (C2). (C1 ⊃ C2)

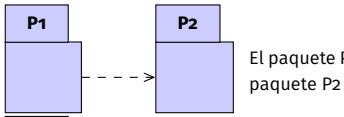
Composición. Agregación en la que las 'partes' (C2) no tienen sentido sin el 'todo' (C1).

Podemos crear una clase asociación cuando una asociación presenta atributos propios y tiene que ser modelada como una clase.

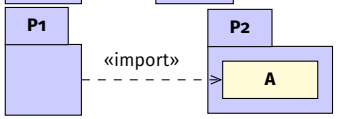
Los *cualificadores de la asociación* son los atributos de algunas de las clases que pasa a ser un atributo asociado a la clase del otro extremo. Al emplear colecciones para modelar las asociaciones cualificadas, se emplea el cualificador como identificador.

Las restricciones son extensiones de la semántica del lenguaje, añadiendo nuevas reglas o modificando las ya existentes. Se representan entre llaves.

Diagramas de paquetes



El paquete P1 usa algún elemento del paquete P2



El paquete P1 importa la clase A del paquete P2

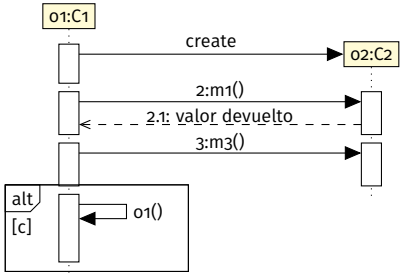
La visibilidad de los elementos de un paquete se marca con + (visible dentro y fuera) o - (visible sólo en el paquete).

Diagramas de secuencia

Muestran de forma visual el orden en el que ocurren los envíos de mensaje dentro de una interacción entre objetos. La sintaxis es la siguiente:

Participantes `nombre : nombreClase`

Mensaje `variable = mensaje(arg) : tipo devuelto`



create	crea una instancia
m1()	es un mensaje síncrono y devuelve un valor
m3()	es un mensaje asíncrono
o1()	es un mensaje al propio objeto

El recuadro con alt es un *fragmento de interacción*, una secuencia de mensajes que ocurre bajo determinadas condiciones. Se pueden combinar. Los argumentos de los operadores descritos a continuación se indican dentro, en un recuadro.

alt	Se ejecuta si se cumple la condición
break	Se ejecuta si se cumple la condición y no se continúa
loop	Se realiza arg veces el fragmento
opt	Como alt pero con un solo fragmento

Diagramas de comunicación

Tipos de enlace (objetoX envía mensaje a objetoY)

Global	G	El ámbito de objetoY es superior al de objetoX
Asociación	A	Existe una relación fuerte y duradera
Parámetro	P	objetoY es pasado como parámetro de objetoX
Local	L	objetoY es referenciado en un método de objetoX
Self	S	objetoX siempre se conoce a sí mismo

Clases en Java y Ruby

Java

```java
package paquete;
import java.util.ArrayList;

public class NombreClase {
    private static int varPrivadaClase;
    public int varPublicaInstancia;

    // Constructor
    public NombreClase(...) {...}

    public tipo metodoPublicoInstancia() {}
    public static tipo metodoPublicoClase() {}
}

NombreClase test = new NombreClase(...);
ArrayList<int> miArrayList = new ArrayList<int>();
```

Ruby

```ruby
class NombreClase
  attr_accessor :var1, :var2
  @@class_variable

  def initialize(var1, var2)
    @var1 = var1
    @var2 = var2
  end

  def metodo_instancia
    puts @var1
  end

  def self.metodo_clase
      ...
  end

   private :metodo_clase
end

test = NombreClase.new(1,2)
test.metodo_instancia
mi_array = Array.new
```