Patrones de diseño

Ingeniería de Software IIC2143

¿Que es un patrón de diseño?

Un patrón es una solución a un problema en un contexto dado.

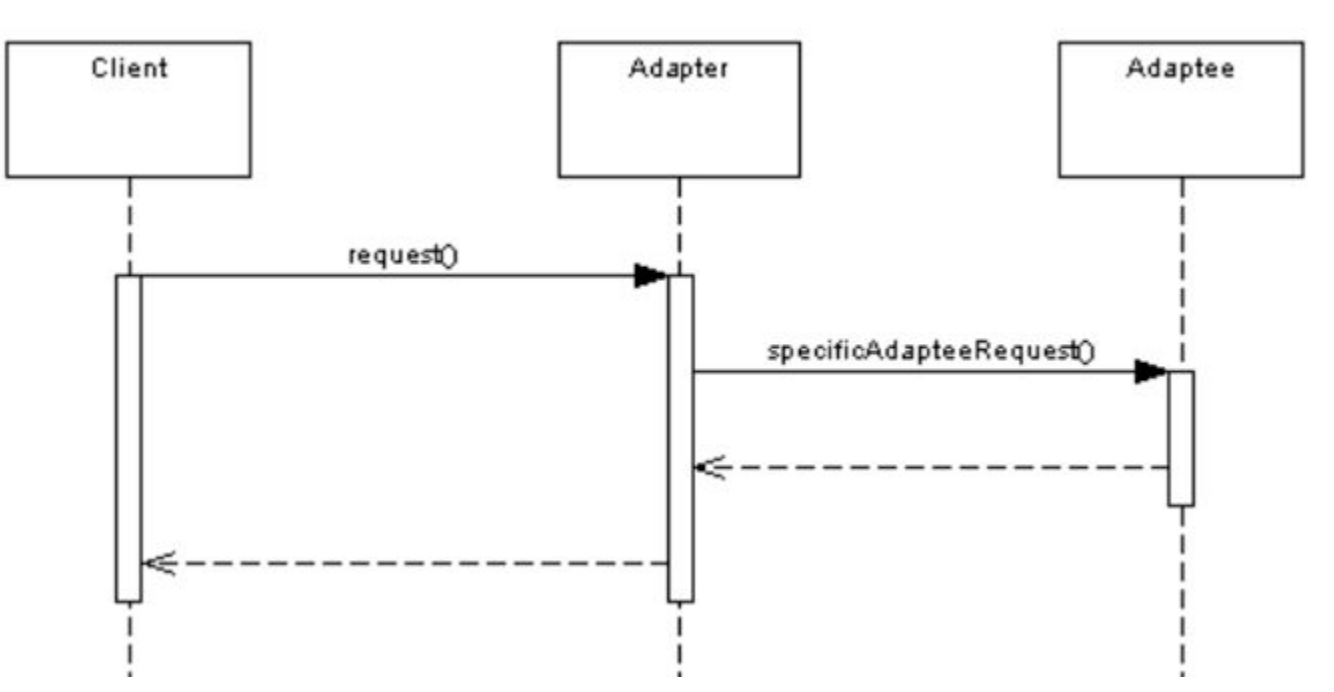
Buscan **minimizar el acoplamiento y mantener la cohesión**, para que el software se pueda extender y modificar con facilidad.

La gracia es poder reutilizar los patrones para enfrentar los distintos problemas, y así se ahorra tiempo de desarrollo.

Resumen de los patrones vistos en cátedra

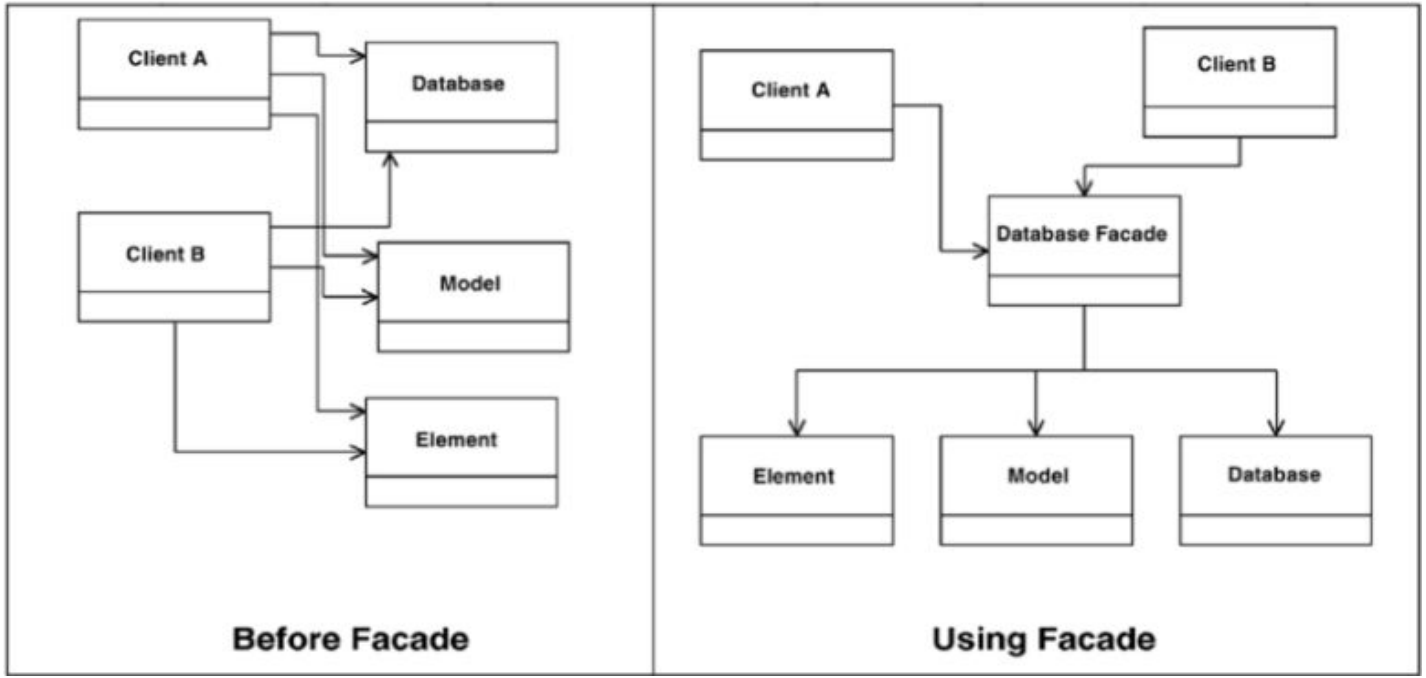
Adapter

Se crea una clase adaptador que permite hacer calzar la interfaz de una clase incompatible con nuestra interfaz



Facade

Se utiliza cuando se tiene un sistema complejo, pero solo se desea interactuar con partes específicas de este



Singleton

Asegurar la creación de una única instancia de la clase, fácilmente accesible.

```
class Tablero
    attr_accessor :status

    @@instance = Tablero.new

    def self.instance
      return @@instance
    end

    private_class_method :new
end
```

variable de clase guarda la única instancia

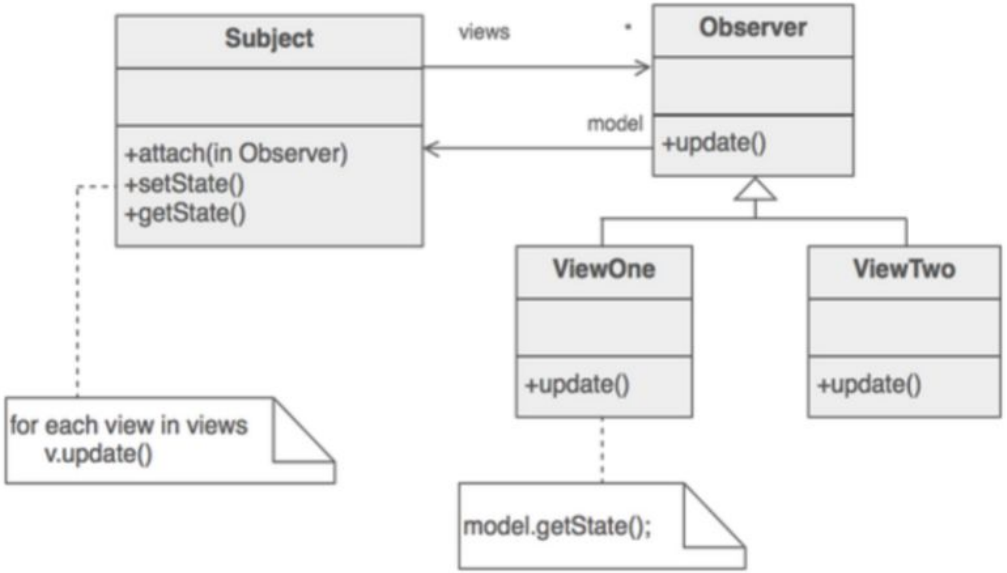
método de clase retorna la única instancia

método de clase new se hace privado

```
tablero1 = Tablero.instance
tablero1.status = "on"
puts tablero1.status
tablero2 = Tablero.instance    # devuelve el mismo tablero
puts tablero2.status
```

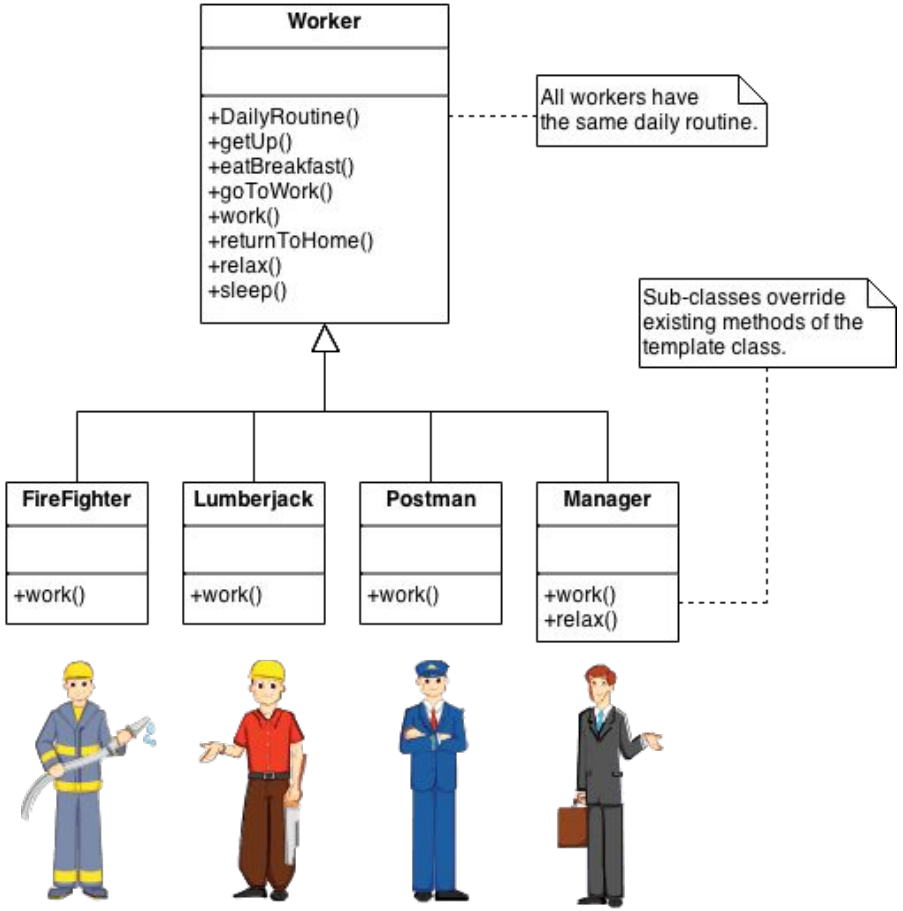
Observer

Se define un mecanismo para notificar a varios objetos (observadores) cuando suceda un evento sobre un objeto particular (sujeto)



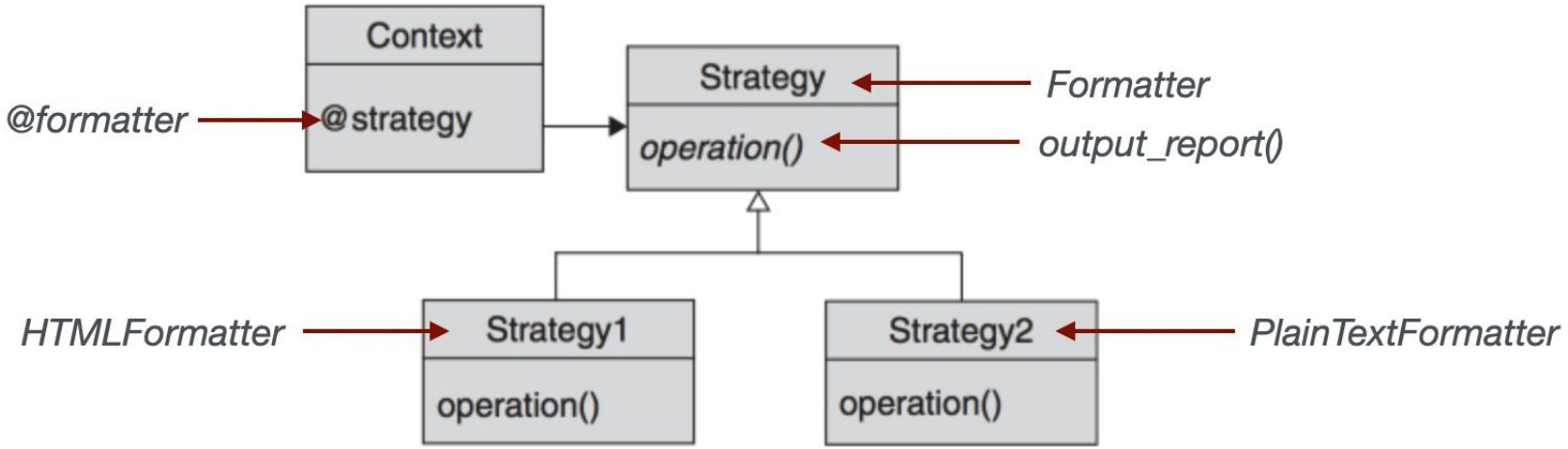
Template Method

Se encapsulan secuencias de operaciones que se repiten mucho.



Strategy

Se encapsula un algoritmo como un objeto que puede ser invocado desde la clase que lo quiere utilizar.

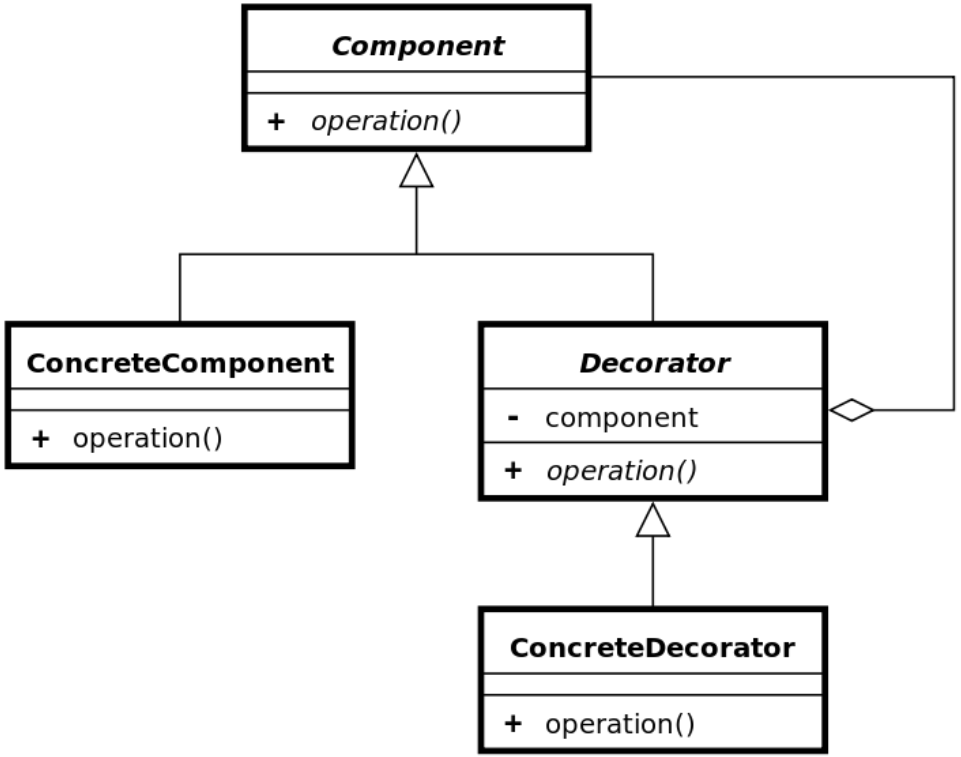


Decorator

Añadir responsabilidades a objetos individuales de forma dinámica y transparente

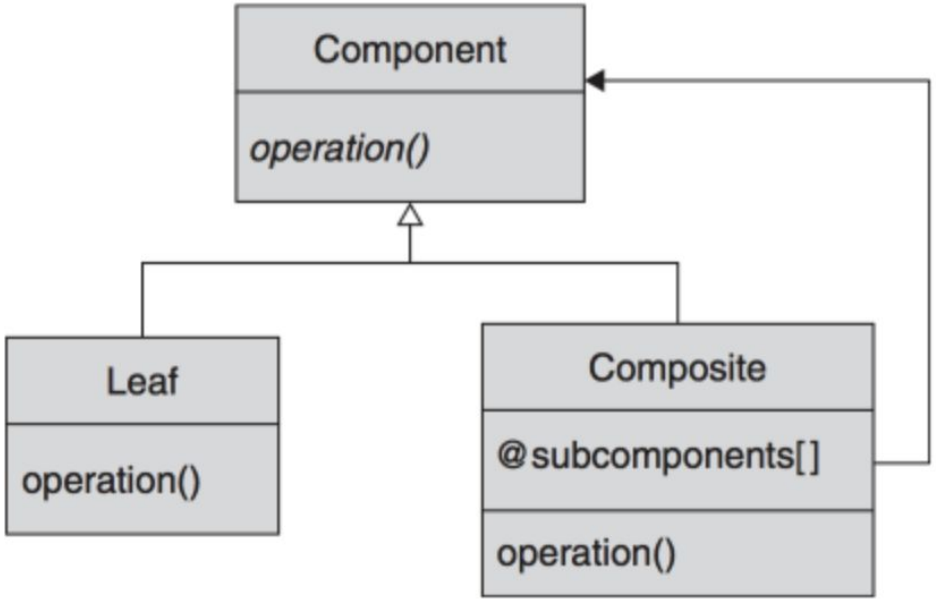
Cuando la extensión mediante la herencia no es viable

*Suele salir en las ies!



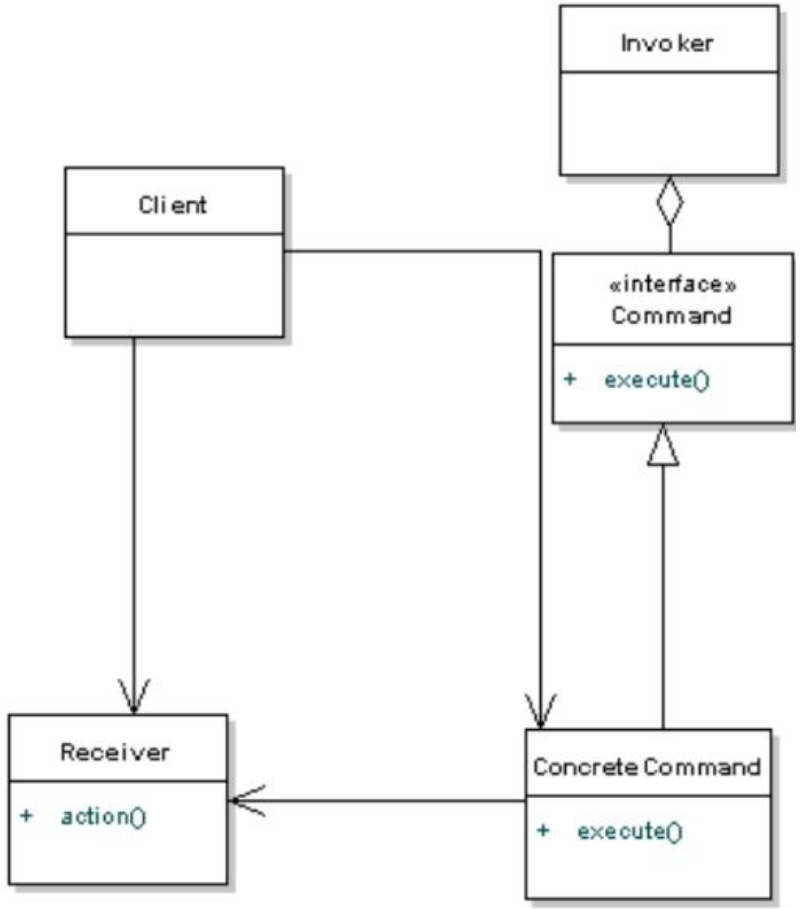
Composite

Construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol.



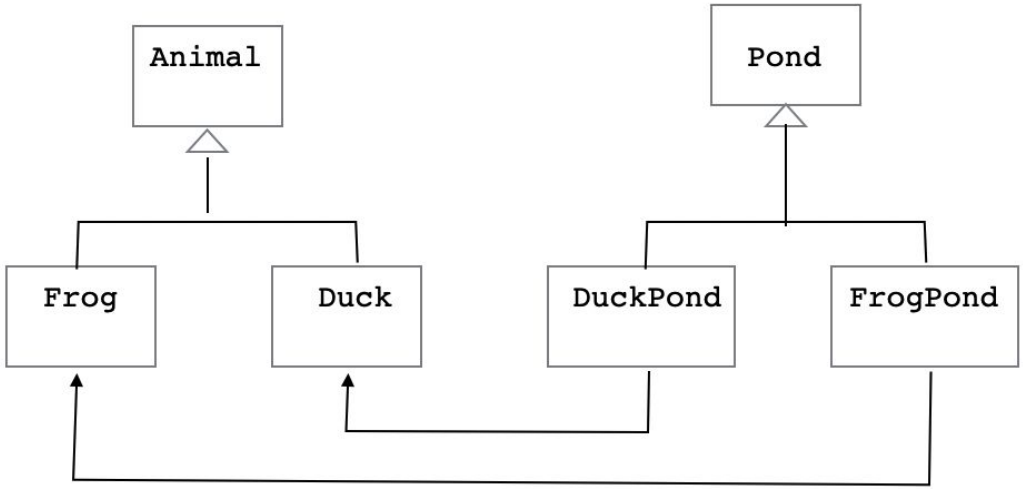
Command

El patrón se utiliza para transformar acciones en objetos. Se utiliza ya que los objetos son más fáciles de reordenar.



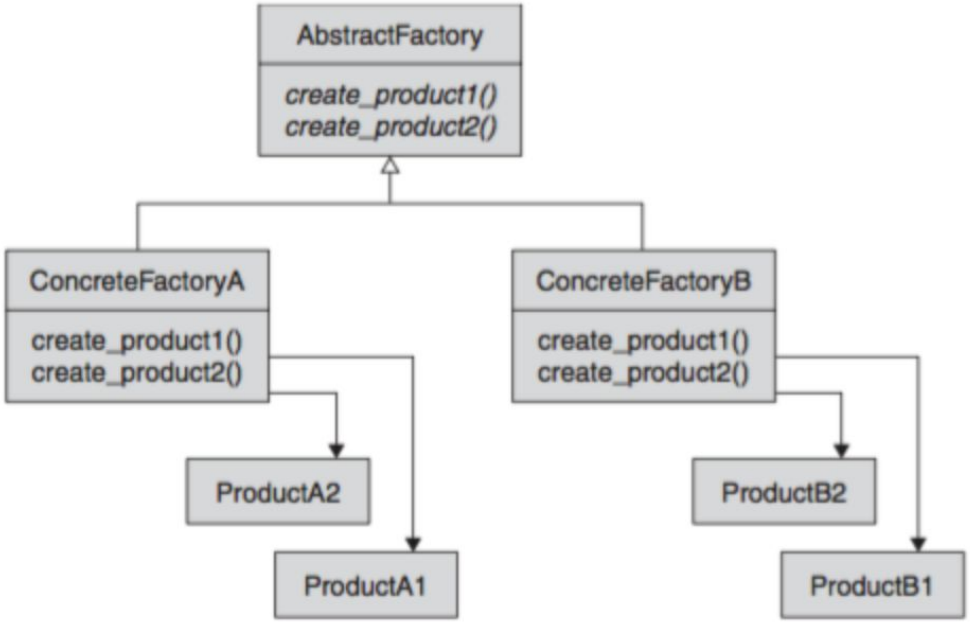
Factory Method

Permite crear objetos en una superclase. Luego, las subclases que heredan a la superclase pueden alterar el tipo de objetos que serán creados.



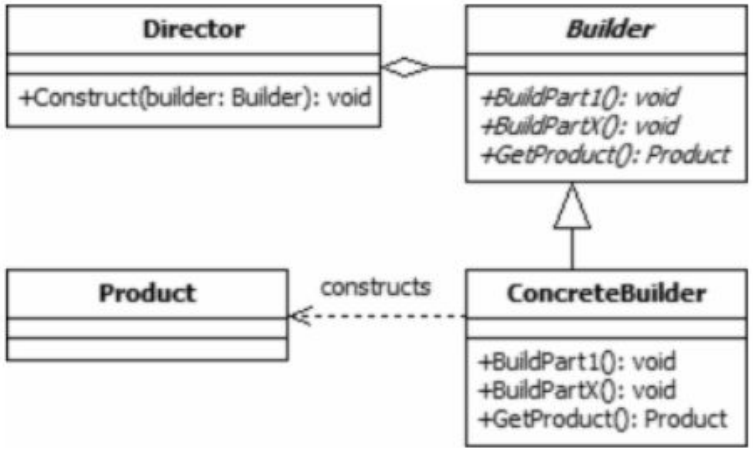
Abstract Factory

Es una fábrica de fábricas. En el diagrama la fábrica A produce objetos A1 y A2 en cambio la fábrica B produce objetos B1 y B2.



Builder

Permite construir objetos complejos paso a paso. Permite construir distintas variaciones de un objeto con el mismo código de construcción.



Ejercicios de ies anteriores

Las siguientes afirmaciones son falsas, diga porqué

1. La manera que tiene Ruby de asegurar que exista una sola instancia de un objeto es implementar una fábrica abstracta.

   Nada que ver. La manera es declarar privado el método de clase new

2. El patrón comando permite que los objetos subscritos no tengan que estar permanentemente haciendo polling para averiguar si es necesario actualizar.

   Corresponde al patrón observador.

3. El patrón fábrica abstracta se parece mucho al patrón método plantilla.

   El patrón fábrica abstracta se parece al patrón estrategia.

4. No es fácil inventar un nuevo patrón de diseño

   Los patrones de diseño no se inventan, se descubren.

Patrón Strategy P5 - Examen 2021-2

5. (20 pts) La PUC se encuentra estudiando la puesta en marcha de un nuevo sistema computacional de asignación de becas que pueda conectarse con los registros de admisión y asignar becas a los estudiantes según distintos tipos de criterios. La base de datos de admisión define objetos del tipo *Student*, donde cada estudiante tiene los siguientes atributos:

- Name (string)
- FamilyIncome (integer)
- PsuAverage (float)

Se te pide desarrollar la clase *ScholarshipAssigner*, que recibe en su constructor el atributo:

- *student_list (array)*: listado de estudiantes entregado por la base de datos de admisión

Además, esta clase define el método *assign_scholarships(number_of_scholarships, strategy) que* imprime el nombre de hasta *number_of_scholarships* alumnos del listado de estudiantes que más ameriten una beca según un determinado criterio definido según el patrón *Strategy*

Patrón Strategy P5 - Examen 2021-2

Admisión se encuentra interesado en considerar los siguientes criterios para la asignación de becas:

- Mejor puntaje PSU: se le asignarán becas a los alumnos con mejor promedio PSU
- Mayor necesidad económica: se le asignarán becas a los alumnos con menor ingreso familiar
- Híbrido: se le asignarán becas a los alumnos con menor ingreso familiar siempre y cuando hayan obtenido igual o más de 700 puntos en la PSU. Si hay más becas que alumnos que cumplan la restricción, no se asignan las becas sobrantes.

Implementa la clase *ScholarshipAssigner* junto con todas las demás clases que estimes necesarias para implementar el patrón *Strategy en una buena solución de este problema*. Puede asumir que la clase *Student* viene dada por lo que puedes instanciarla pero no puedes implementarla.

Recuerda que en Ruby existe el método *sort* definido en la clase *Array*, el cual permite retornar un listado de elementos ordenados según el resultado entregado por un bloque que se le puede pasar como parámetro. Este bloque recibe dos parámetros a y b de input y retorna un número negativo si b viene después de a, un número positivo si a viene después de b, y retorna 0 si a es igual a b.

Puedes asumir también que: $number\_of\_scholarships < student\_list.length$.

Patrón Strategy P5 - Examen 2021-2

```ruby
class ScholarshipAssigner
  def initialize(student_list)
    @student_list = student_list
  end

  def assign_scholarships(number_of_scholarships, strategy)
    strategy.assign_scholarships(@student_list, number_of_scholarships)
  end
end

class BestPsuScoreStrategy
  def assign_scholarships(student_list, number_of_scholarships)
    list = student_list.sort { |a, b| b.psu - a.psu }
    (0...number_of_scholarships).each { |i| puts(list[i].name) }
  end
end
```

Patrón Strategy P5 - Examen 2021-2

```ruby
class LessIncomeStrategy
  def assign_scholarships(student_list, number_of_scholarships)
    list = student_list.sort { |a, b| a.income - b.income }
    (0...number_of_scholarships).each { |i| puts(list[i].name) }
  end
end

class HybridStrategy
  def assign_scholarships(student_list, number_of_scholarships)
    list = student_list.sort { |a, b| a.income - b.income }
    count = 0
    list.each do |student|
      if (count < number_of_scholarships) && (student.psu >= 700)
        puts(student.name)
        count += 1
      end
    end
  end
end
```

Patrón Composite P2 - I2 2021-2

La discografía de un artista incluye varios álbums. Cada álbum incluye varias canciones. Queremos modelar las clases Discografía, Album y Song de la forma en que se indica a continuación

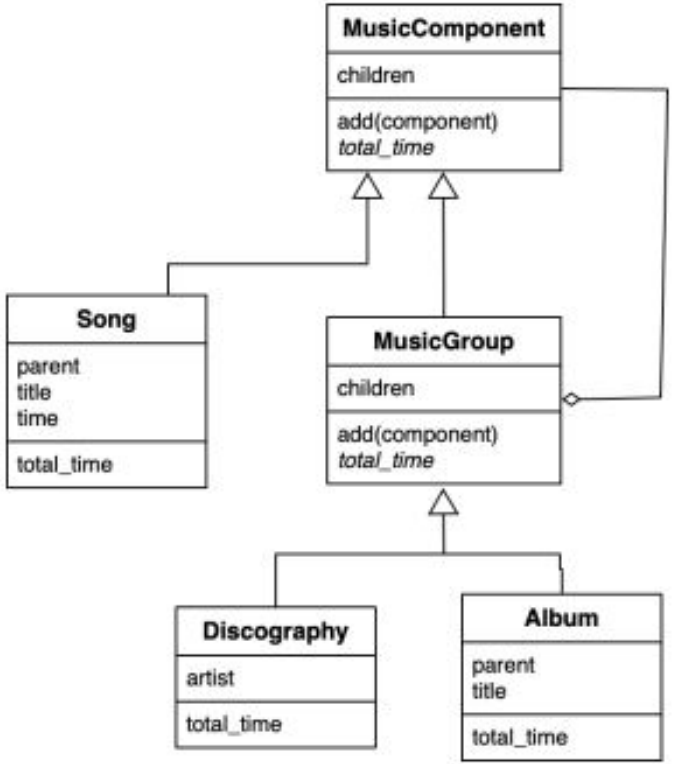
- una discografía tiene un único atributo artista y un método tiempo_total que entrega el total de minutos correspondiente a toda la discografía del artista
- un álbum tiene un único atributo nombre y un método tiempo_total que entrega el total de minutos correspondiente a las canciones del álbum
- una canción tiene dos atributos: título y duración y un método duración que entrega la duración en minutos de la canción
- queremos sacar partido del patrón composite de modo que el método duración entregue el total de minutos de una canción, álbum o de toda la discografía del artista dependiendo del objeto de que se trate

a) (10 pts) Dibuje un diagrama de clases UML que muestre la solución

b) (10 pts)Escriba el código Ruby que lo implemente

Patrón Composite P2 - I2 2021-2

a)



Patrón Composite P2 - I2 2021-2

b)

ejemplo.rb

```ruby
class MusicComponent
    def total_time
        raise('abstract method')
    end
end

class MusicGroup < MusicComponent
    def initialize
        @children = Array.new
    end
    def add_child(component)
        @children << component
    end
    def total_time
        total_time = 0
        @children.each {|item| total_time += item.total_time}
        total_time
    end
end
```

```ruby
class Discography < MusicGroup
    def initialize(artist)
        super()
        @artist = artist
    end
end
class Album < MusicGroup
    attr_accessor :parent, :title
    def initialize(title)
        super()
        @title = title
    end
end

class Song < MusicGroup
    attr_accessor :parent, :title
    def initialize(title, time)
        @title = title
        @time = time
    end
    def total_time
        @time
    end
end
```

Patrón Composite P2 - I2 2021-2

b)

```
46
47  beatles_discography = Discography.new('The Beatles')
48  rubber_soul = Album.new('Rubber Soul')
49  revolver = Album.new('Revolver')
50  rubber_soul.add_child Song.new('Norwegian Wood', 1.5)
51  rubber_soul.add_child Song.new('Nowhere Man', 2.0)
52  revolver.add_child Song.new('Eleanor Rigby', 1.5)
53  revolver.add_child Song.new('Tomorrow Never Knows', 2.5)
54  beatles_discography.add_child rubber_soul
55  beatles_discography.add_child revolver
56  puts rubber_soul.total_time
57  puts revolver.total_time
58  puts beatles_discography.total_time
```