23/8/19

* Hay tres categorías de patrones de diseño:
* Creacionales: determinar cómo crear objetos
* Estructurales: cómo usar objetos entre sí mediante composición
* De comportamiento: cómo comunicar objetos sin componerlos directamente.

|  |  |  |
| --- | --- | --- |
| **Creacionales** | **Estructurales** | **De comportamiento** |
| Factory  -Builder  -Singleton  Prototype | Adapter  -Abstract factory  Bridge  Composite  Decorator  Facade  Proxy | Interprete  Cadena de responsabilidades  Comando  Iterador  Menento  -Observer |

(Hacer ejemplos de los – )

* Patrón Factory: crea una clase factory que crea objetos concretos mediante un selector. El caller no conoce las clases concretas.

Interface Dog {

Void bark();

}

Class Rottweiller implements Dog {

Void bark() {

-------------

-------------

}

}

Class Poodle implements Dog {

Void bark() {

------------

------------

}

}

Class Dog\_Factory{

Public static Dog getDog(DogType d)

If ( d == DogType.BIG) {

return new Rottweiller();

}else if( d == DogType.SMALL) {

return new Poodle();

}else {

throw new Exception(“Unknown Dog type”);

}

}

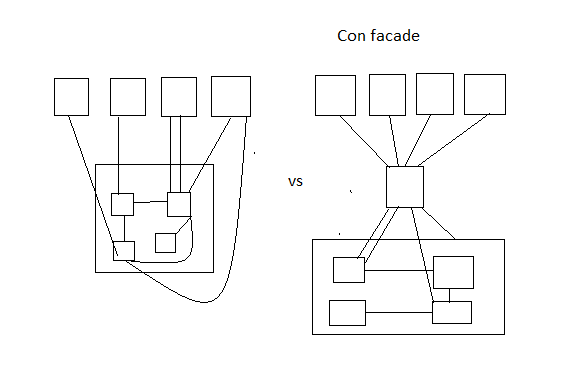
enum DogType {

SMALL;

BIG;

}

* Facade: Crea una clase que funciona como fachada para un subsistema complejo. El caller interactúa con el facade y no con el subsistema directamente.



* Patrón Singleton

Patrón que permite restringir la instanciación de una clase. Únicamente se puede instanciar una sola clase. Ejemplo:

public class Singleton {

    private  static Singleton instance = null;

    private Singleton ( ){ // Todo el código aquí

}

}

public static singleton get Instance ( ){

    if( instance == null){

instance = new Singleton ( );

}

return instance;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ main

Singleton.getInstance( ).do Something( );

Singleton var = Singleton.getInstance( );

// Synhronized → funciona para que se utilice un hilo a la vez

* Patrón Builder

Delega la construcción de una clase compleja a una clase complementaria.