* Hay tres categorías de Patrones de Diseño:
* Creacionales: Determinan cómo crear objetos.
* Estructurales: Cómo usar objetos entre sí mediante composición.
* De comportamiento: Cómo comunicar objetos sin componerlos directamente.

|  |  |  |
| --- | --- | --- |
| Creacionales | Estructurales | De comportamiento |
| Factory | Adapter | Intérprete |
| Builder | Bridge | Cadena de Responsabilidad |
| Singleton | Composite | Comando |
| Prototype | Decorator | Memento |
| - | Facade | Observer |
| - | Proxy | - |
| - | Abstract Factory | - |

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

Interface Dog{

Void bark();

}

Calss Poodle implements Dog{

Void bark( ){

// Cuerpo del código

}

}

Class Rottweller(){

Void bark(){

// Cuerpo del código

}

}

Class DogFactory{

Public static Dog getDog(DogType d){

If(d == DogType.BIG){

Return new Rottweiller();

}else if(d = DogType.SMALL){

Return new Pooller();

}else{

Throw new Exception(“Unknown Dog Type”);

}

}

}

enum Dogtype{

SMALL;

BIG;

}

* Interface: Una clase que no tiene atributos, tiene métodos pero este no tiene cuerpo.

Se puede implementar el método usando implements, donde se escribe el código que se ejecutará con ello.

* Enum: permite referirme a un mensaje dentro de un sistema, es como una lista de opciones.
* Main(){

…

Dog d1 = DogFactory.getDog(“Small”);

}

* Facade: Crea una clase que funcione como Fachada para un subsistema complejo. El caller interactúa con el Facadey no con el subsistema directamente.

