23 agosto 2019

* Hay 3 categorías de patrones de diseño:
* Creacionales: determina como crear objetos
* Estructurales: como usar objetos entre sí mediante composición
* De comportamiento: como comunicar objetos sin componentes directamente

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

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

Interfce Dog {

void bark. ();

}

Class Rottwailler implements Dog {

void bark () {

codigo

}

}

Class Poodle implements Dog {

void bark () {

codigo}

}

Class Dog\_Factory{

public static Dog getDog(DogType d){

if (d==DogType.big) {

return new Rottweiler ();

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

return new Poodle ();

} else {

throw new Exception (“Unknown Dog,type 4);

}

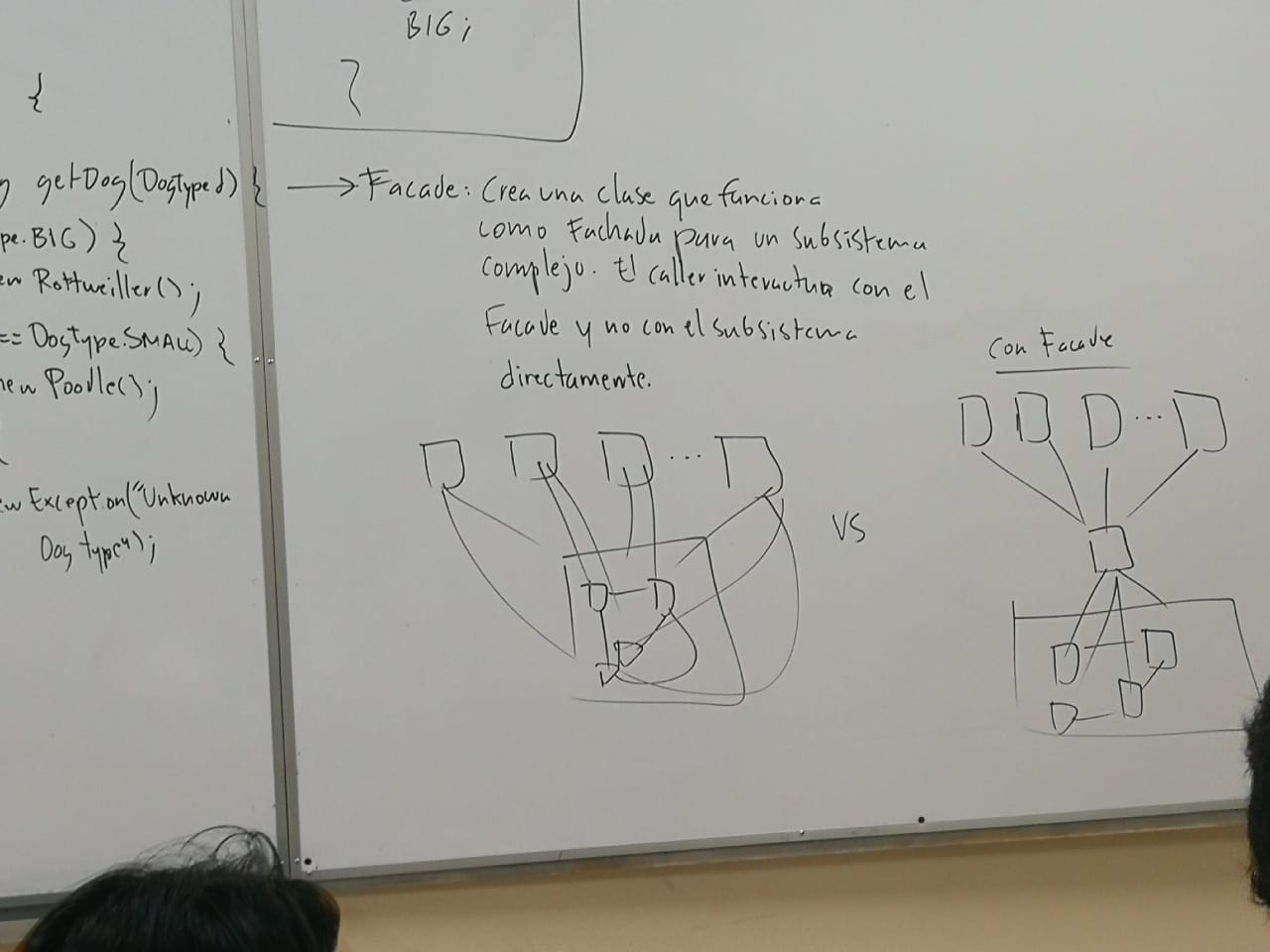
}

Enum DogType {

SMALL,

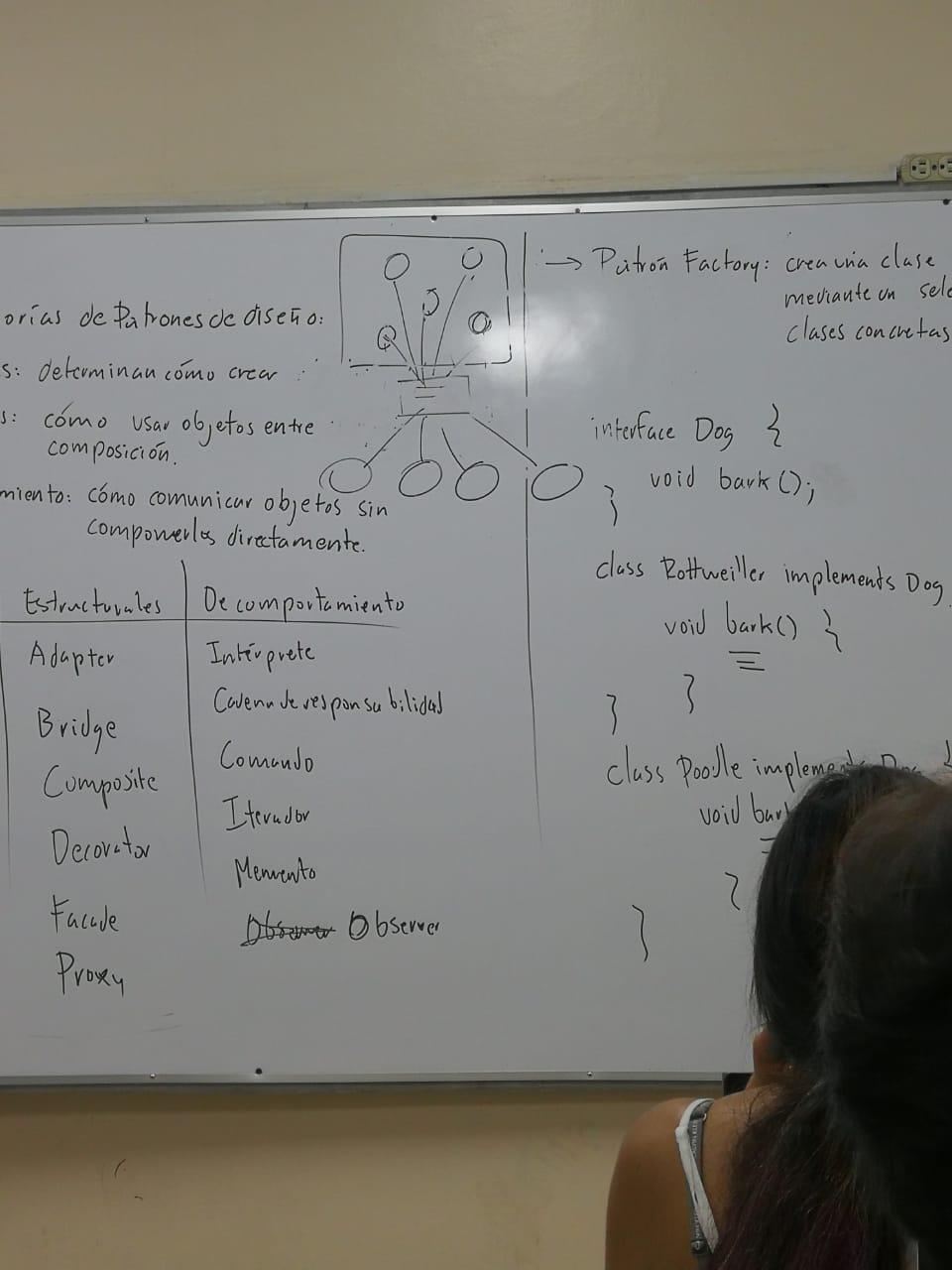
BIG;}

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



Notas:

Facade Soluciona el no tener que modificar todas las clases del sistema principal, cuando sucede algún cambio es mejor modificar el facade para no tener que lidiar con “n” cantidad de clientes.



Sistema principal

Clientes

Facade