Design Patterns : Documontation

Term	Definition or instruction…
Description	Décrit à la fois : --un problème qui se produit fréquemment --l'architecture de la solution a ce problème de telle façon que l'on puisse utiliser cette solution des milliers de fois. Permet de décrire avec succès des types de solutions récurrentes a des problèmes communs dans des types de situation Pourquoi le maitrisé ?? Il nous offre : Une documentation d'une expérience approuvée de conception. Une abstraction (élevé) au-dessus de niveau classes et instance Un vocabulaire commun, moyen de documentation…
Catégories	Création : --Description de la manière dont un objet peut être crées, initialisés, configurés --Isolation du code relatif à la création, initialisation, configuration afin de rendre l'application indépendante de ces aspects EXPL: Singleton (1 instance), Prototype, Builder, Abstract Factory
	Structuration : (couplage, diagramme de classe) --Description de la manière dont doivent être connectes des objets a fin de rendre ces connections indépendante des évolutions futures EXPL: Adapter, composite, Bridge, Decorator, Façade, Proxy
	Comportement (diagramme de séquence) --Description de comportements d'interaction entre objets --Gestion des interactions dynamiques entre des classes et des objets EXPL: Strategy, Observer, Iterator, Mediator, Visitor, State

Design Patterns : Documontation

Term	Definition or instruction…
Portée de Design Patterns	2 manières de réutiliser les classes --Portée de classe par héritage /extension -- Portée d'instance (objet) Par composition
GOF	Gang of four

		Catégorie		
		Création	**Structure**	**Comportement**
Portée	**Classe**	Factory Method	Adapter	Interpreter
				Template Method
	Objet	Abstract Factory	Adapter	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Facade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

Pattern Strategy

Code Example

```
class Context
    {
        private IStrategy _strategy;

        public Context(){ }

        public Context(IStrategy strategy)
        {
            this._strategy = strategy;
        }

        public void SetStrategy(IStrategy strategy)
        {
            this._strategy = strategy;
        }

        public void DoSomeBusinessLogic()
        {
                                var data = "hello";
            var result = this._strategy.DoAlgorithm(data);
            Console.WriteLine(result);
        }
    }
```

```
    public interface IStrategy
    {
        object DoAlgorithm(object data);
    }

    class ConcreteStrategyA : IStrategy
    {
        public object DoAlgorithm(object data)
        {
            var list = data+"world";
            return list;
        }
    }

    class ConcreteStrategyB : IStrategy
    {
        public object DoAlgorithm(object data)
        {
            var list = data+"StrategyA";
            return list;
        }
    }
class Program
    {
        static void Main(string[] args)
        {
            var context = new Context();
            context.SetStrategy(new ConcreteStrategyA());
            context.DoSomeBusinessLogic();

            Console.WriteLine();
            context.SetStrategy(new ConcreteStrategyB());
            context.DoSomeBusinessLogic();
        }
    }
```