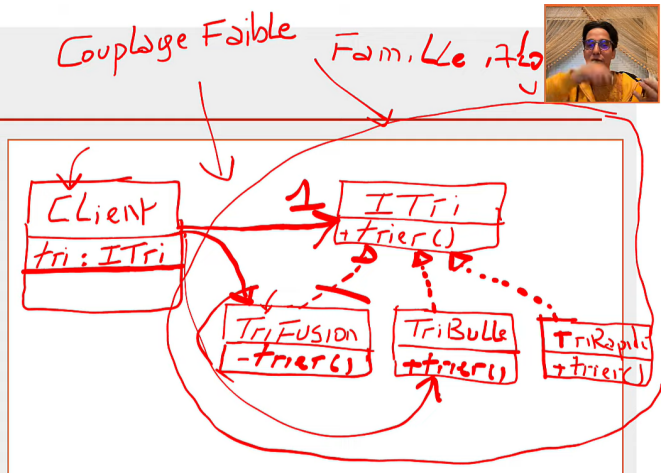
ZAKARIA ZIGHIGHI DP-Strategy

`/Notes de cours`

Pattern Strategy

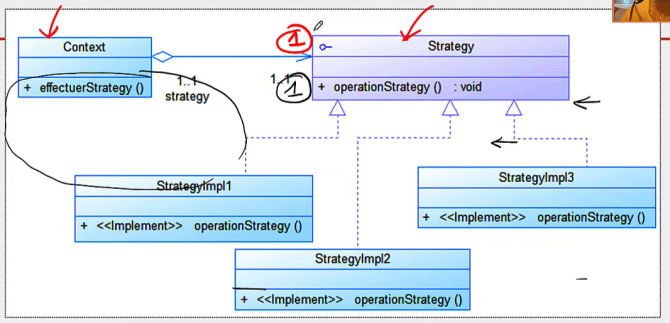
- Catégorie : Comportement
- Objectif s:
  - Définir une famille d'algorithmes, et encapsuler chacun et les rendre interchangeables tout en assurant que chaque algorithme puisse évoluer indépendamment des clients qui l'utilisent
- Raison d'utilisation :
  - Un objet doit pouvoir faire varier une partie de son algorithme dynamiquement.
- Résultat :
  - Le Design Pattern permet d'isoler les algorithmes appartenant à une même famille d'algorithmes.



🚨: REMARQUE --RELATION 1 → 1 + Une seule méthode dans l'interface

Diagramme de classes

- On crée donc une **interface** de base, appelée ici « ***Strategy »*** et on y ajoute une méthode qui sera la méthode qui applique notre stratégie.
- Il suffit alors de créer maintenant des classes concrètes qui implémentent cette interface « **StrategyImpl** » et qui donc redéfinisse la méthode de stratégie.
- A un instant donné, La classe « **Context** » qui va utiliser la stratégie compose une instance de l'une des implémentation de Strategy.



→ IMPLEMENTATION :

`class Context`

```
package org.example;

public class Context {


    private Strategy strategy=new DefaultStrategy();
    public void effectuerOperation(){
        System.out.println("***************");
        strategy.operationStrategy();
        System.out.println("---------------");
    }
    public void setStrategy(Strategy strategy) {
        this.strategy = strategy;
    }

}
```

`interface Strategy`

```
package org.example;

public interface Strategy {
    void operationStrategy();

}
```

`class StrategyImpl1`

```
package org.example;

public class StrategyImpl1 implements Strategy {
    @Override
    public void operationStrategy() {
        System.out.println("//////Strategy 1////////");
```

```
    }
}
```

`class StrategyImpl2`

```
package org.example;

public class StrategyImpl2 implements Strategy {
    @Override
    public void operationStrategy() {
        System.out.println("$$$$$$Strategy 2$$$$$$$$$$$$");

    }
}
```

`class DefaultStrategy`

```
package org.example;

public class DefaultStrategy implements Strategy {
    @Override
    public void operationStrategy() {
        System.out.println("(((DEFAULT STRATEGY)))");

    }
}
```

`class Main`

```
package org.example;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception {

        Context context =new Context();
        Scanner scanner = new Scanner(System.in);
        Map<String, Strategy> strategyMap=new HashMap<>();
        Strategy strategy;
        while (true){
            System.out.println("Quelle stratégie");
            String str = scanner.nextLine();
            strategy=strategyMap.get(str);
            if(strategy==null){
                System.out.println("Creation nv obj de StrategyImpl"+str);
                 strategy = (Strategy) Class.forName("org.example.StrategyImpl"+str).getConstructor().newInstance();
                 strategyMap.put(str,strategy);
            }

            context.setStrategy(strategy);
            context.effectuerOperation();



        }
    }
}
```

→ Exécution :


Run
Main
Quelle stratégie
1
Creation nv obj de StrategyImpl1
****
//////Strategy 1////////

Quelle stratégie
1
****
//////Strategy 1////////

Quelle stratégie
2
Creation nv obj de StrategyImpl2
****
$$$$$$Strategy 2$$$$$$$$$$$$

Quelle stratégie
2
****
$$$$$$Strategy 2$$$$$$$$$$$$
