

4A MOC:

- Ramzy KERMAD
- Nassim Morouche
- Vithursan SIVAKUMARAN

# Abstract Factory Pattern

A decorative graphic consisting of dashed white arrows on a blue grid background. The arrows form a rectangular frame around the title. Starting from the bottom-left, a dashed arrow points up and to the right. From the top-right, a dashed arrow points down and to the left. From the bottom-right, a dashed arrow points up and to the left. From the top-left, a dashed arrow points down and to the right. These four arrows are connected by solid white lines, creating a rectangular border.



# PRÉAMBULE

## La présentation

Cette présentation a été faite avec peu de difficulté. Nous avons essayé d'expliquer au mieux le concept du patron de conception et avons donné un exemple concret d'utilisation. Nous espérons qu'à la fin vous aurez une vision plus claire à son propos.

## Le sujet

Nous avons déjà utilisé ce pattern dans notre travail ou pour des projets personnels, nous n'avons donc pas eu de difficulté à le comprendre mais plutôt à vulgariser sa compréhension à ceux qui le verraient pour la première fois.






# 1

## UNE FABRIQUE, C'EST QUOI ?



Une fabrique abstraite,  
ok, mais une fabrique  
d'abord!





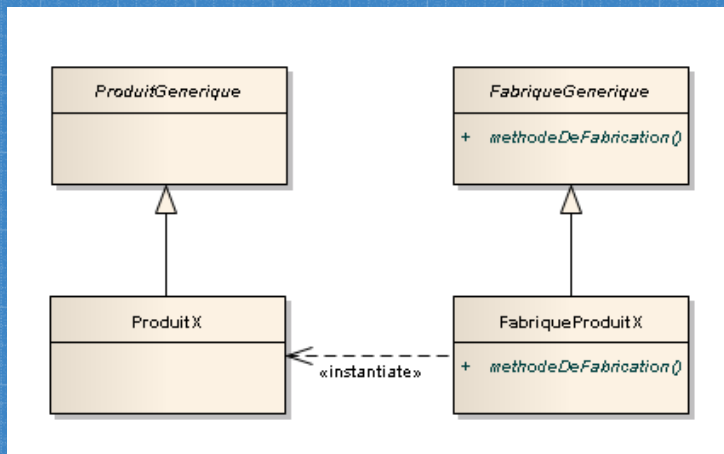


La fabrique est un patron de conception créationnel utilisé en POO. Permet d'instancier des objets dont le type est dérivé d'un type abstrait.

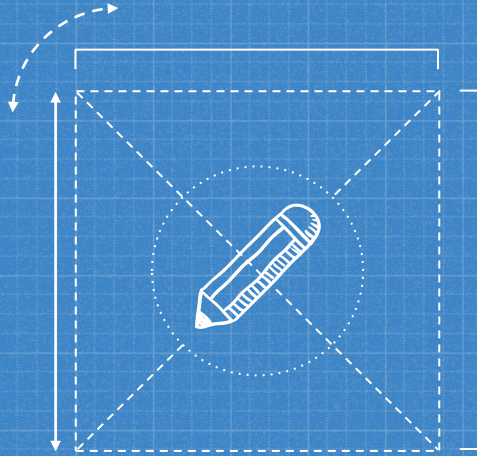


## COMMENT L'UTILISER ?

- Instancier dynamiquement des sous-classes
- Non liée à l'héritage de classes
- Créer plusieurs objets sans polymorphisme







# Fabrique... Abstraite

Fabrique simple & abstraite, quelles différences ?



## C'EST QUOI ?

- Ensemble de Fabriques à thème commun.
- Cache la responsabilité de création.
- Endroit du code où sont créés les objets.



# Utilisation

## Décision:

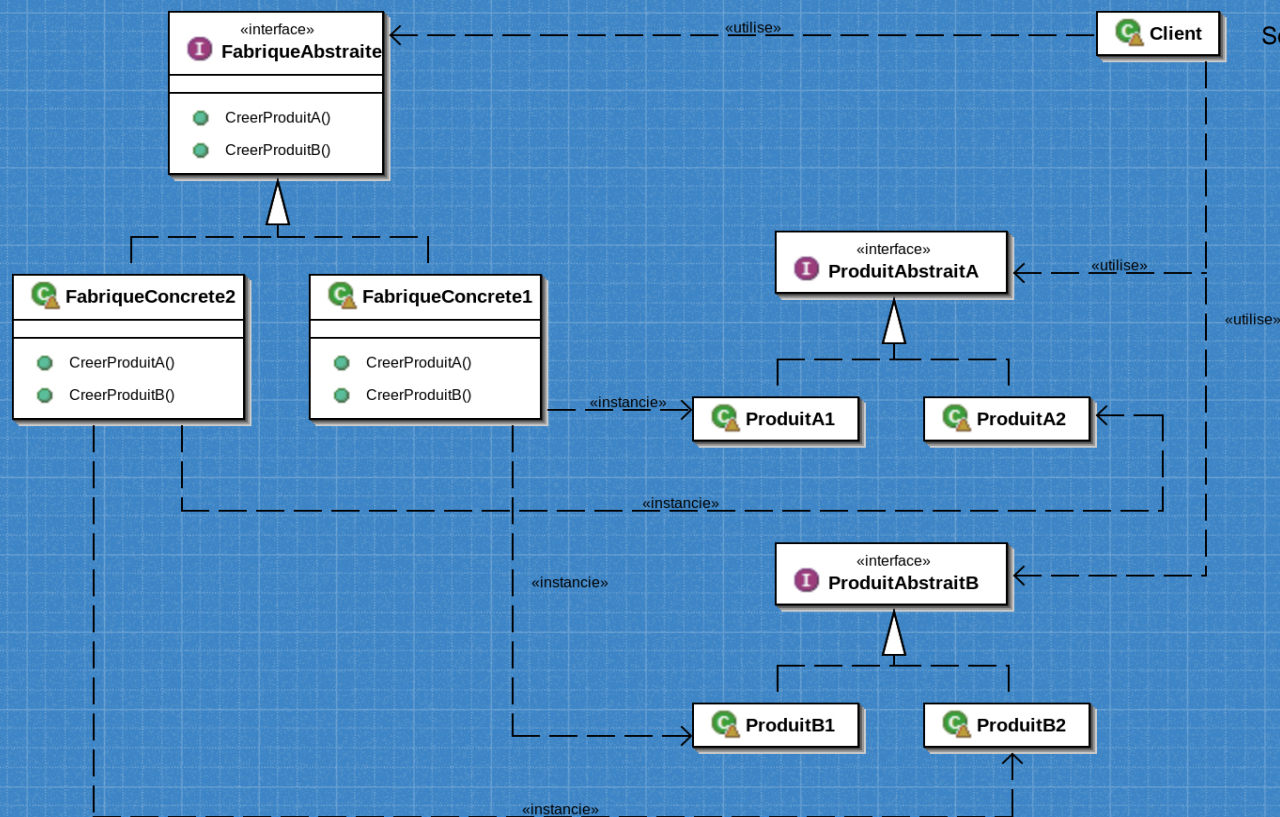
La fabrique détermine le type de l'objet concret qu'il faut créer, et c'est ici que l'objet est effectivement créé (dans le cas du C++, c'est l'instruction new).

## Abstraction:

Le code client est isolé de la création de l'objet en l'obligeant à demander à une fabrique de créer l'objet du type abstrait désiré et de lui en retourner le pointeur.



# Structure



Source: Wikipédia



## EXEMPLE D'USAGE CONCRET

Supposons que l'on veuille faire un système pour créer 2 types d'ordinateurs:

- Serveur
- PC

```
public abstract class Computer {  
  
    public abstract String getRAM();  
    public abstract String getHDD();  
    public abstract String getCPU();  
  
    @Override  
    public String toString(){  
        return "RAM= "+this.getRAM()+", HDD="+this.getHDD()+", CPU="+this.getCPU();  
    }  
}
```



# EXEMPLE D'USAGE CONCRET

```
public class PC extends Computer {  
  
    private String ram;  
    private String hdd;  
    private String cpu;  
  
    public PC(String ram, String hdd, String cpu){  
        this.ram=ram;  
        this.hdd=hdd;  
        this.cpu=cpu;  
    }  
    @Override  
    public String getRAM() {  
        return this.ram;  
    }  
  
    @Override  
    public String getHDD() {  
        return this.hdd;  
    }  
  
    @Override  
    public String getCPU() {  
        return this.cpu;  
    }  
}
```

```
public class Server extends Computer {  
  
    private String ram;  
    private String hdd;  
    private String cpu;  
  
    public Server(String ram, String hdd, String cpu){  
        this.ram=ram;  
        this.hdd=hdd;  
        this.cpu=cpu;  
    }  
    @Override  
    public String getRAM() {  
        return this.ram;  
    }  
  
    @Override  
    public String getHDD() {  
        return this.hdd;  
    }  
  
    @Override  
    public String getCPU() {  
        return this.cpu;  
    }  
}
```

```
public interface ComputerAbstractFactory {  
  
    public Computer createComputer();  
}
```

## Classes filles:

Ce sont les deux classes finales qui seront instanciées et utilisées par la classe cliente.

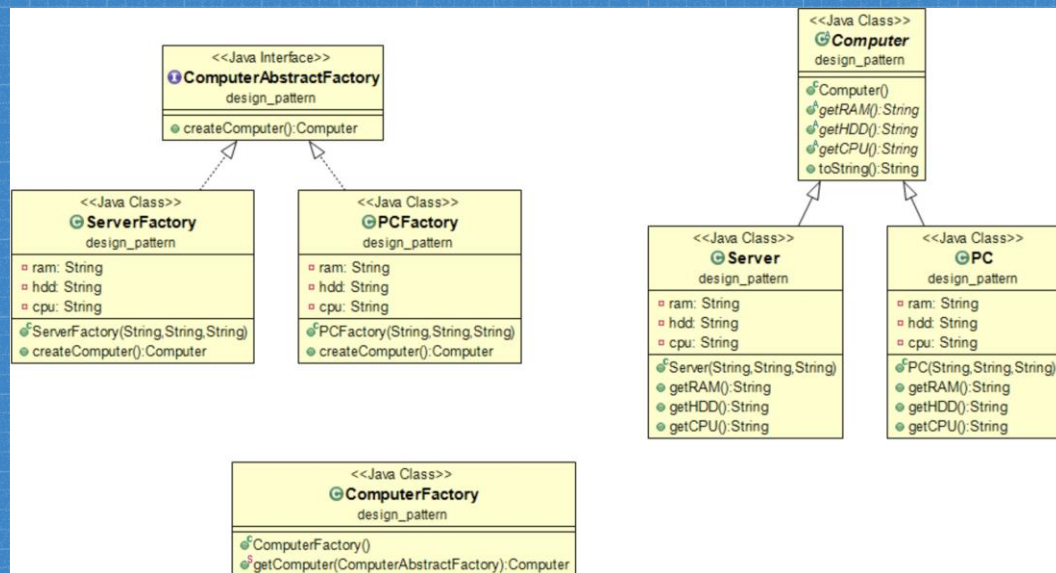


# EXEMPLE D'USAGE CONCRET

```
public class ComputerFactory {  
  
    public static Computer getComputer(ComputerAbstractFactory factory){  
        return factory.createComputer();  
    }  
}
```

## Factory:

Création de la fabrique, qui possède une fonction abstraite permettant d'instancier un objet héritant de la classe `Computer` tout en gardant une couche d'abstraction entre le client et l'objet instancié.





## EXEMPLE D'USAGE CONCRET

```
class Main {  
  
    public static void main(String[] args) {  
        Computer pc = ComputerFactory.getComputer(new PCFactory("2 GB", "500 GB", "2.4 GHz"));  
        Computer server = ComputerFactory.getComputer(new ServerFactory("16 GB", "1 TB", "2.9 GHz"));  
        System.out.println("AbstractFactory PC Config: "+pc);  
        System.out.println("AbstractFactory Server Config: "+server);  
    }  
}
```

Classe cliente

```
AbstractFactory PC Config::RAM= 2 GB, HDD=500 GB, CPU=2.4 GHz  
AbstractFactory Server Config::RAM= 16 GB, HDD=1 TB, CPU=2.9 GHz
```

Resultat



# FAISONS LE POINT!



## Abstraction

Très utile pour des projets d'envergure.



## Complexité

Il réduit toute la gestion d'instantiation des objets.



## Modulable

On peut facilement structurer ce code grâce au pattern.



## Plus

Pour creuser le sujet voici quelques patron de conception qui ont eux aussi un rôle de création d'objets plus ou moins complexes:

- Monteur
- Prototype
- Singleton



# Merci!

Que celui qui a des questions parle, ou se  
taise à jamais !



# CREDITS

Spéciale dédicace à personne parcequ'il fallait être là!

Non je rigole voici les liens où on a choppé ce contenu de type qualitative pour essayer de vous faire comprendre de nouvelles choses aujourd'hui:

Wikipédia, la base:

- [https://fr.wikipedia.org/wiki/Fabrique\\_abstraite](https://fr.wikipedia.org/wiki/Fabrique_abstraite)

Pour la présentation (avoue elle est stylée):

- Presentation template by [SlidesCarnival](#)