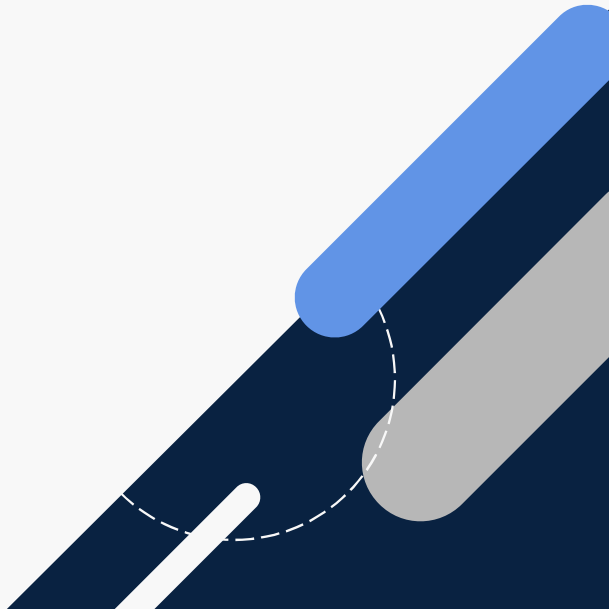
Pattern
Abstract Factory



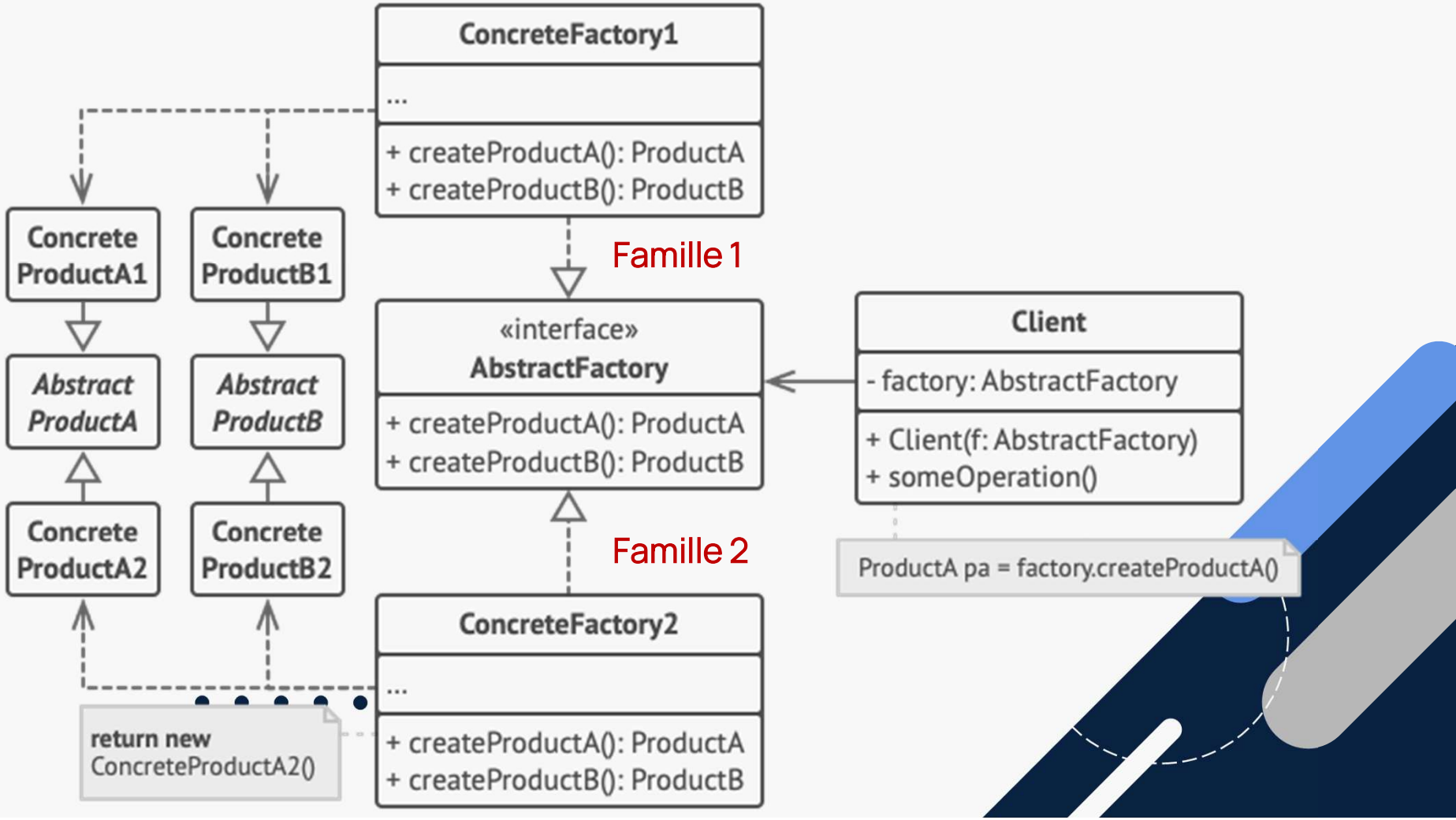
Définition

- L'Abstract Factory est un pattern de création qui fournit une interface pour créer des familles d'objets liés ou dépendants, sans spécifier leurs classes concrètes.
- Contrairement au Factory Method, qui crée un seul type d'objet, l'Abstract Factory crée plusieurs objets qui sont cohérents entre eux.
- Il est souvent utilisé lorsque le système doit être indépendant de la façon dont ses objets sont créés, composés et représentés.

Objectifs

- Encapsuler la création d'objets : Permet de créer des objets sans exposer la logique de création au client.
- Assurer la cohérence : Garantir que les objets créés appartiennent à la même famille ou respectent des contraintes communes.
- Séparer le client des classes concrètes : Le client utilise uniquement les interfaces abstraites, ce qui facilite l'extension et la maintenance.
- Faciliter l'extensibilité : Ajouter de nouvelles familles de produits ne nécessite pas de modifier le code client existant.

Structure UML

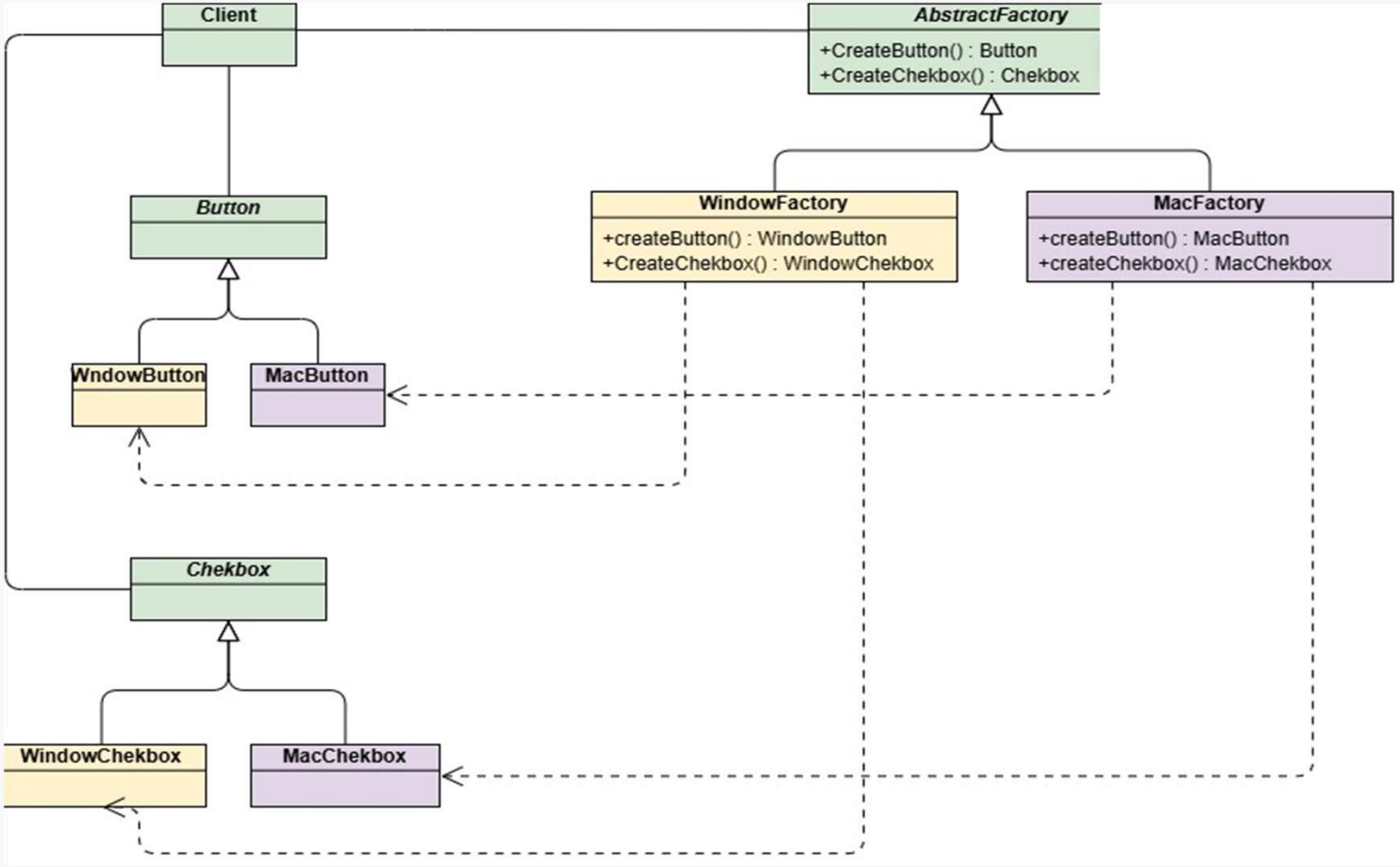


Exemple d’implémentation

Imaginons que nous développons une application qui doit pouvoir fonctionner aussi bien sur Windows que sur MacOS.

Selon le système, les boutons et les cases à cocher doivent avoir un style différent. Au lieu de mélanger du code spécifique à chaque système un peu partout dans l’application, nous allons utiliser Abstract Factory pour centraliser la création de ces composants.

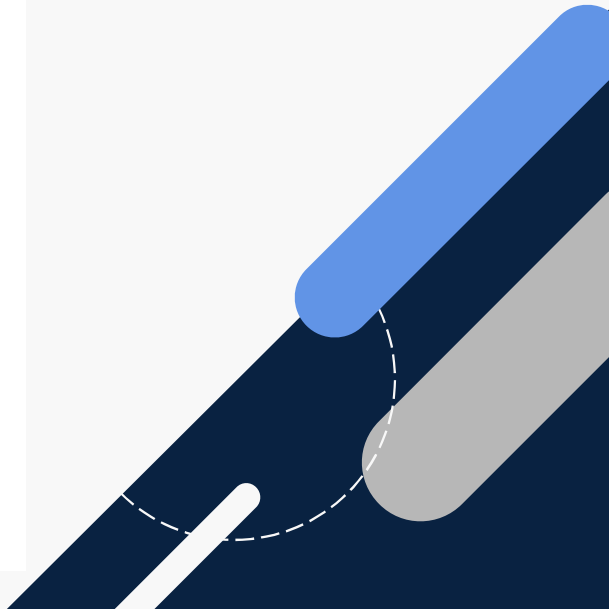
Exemple d’implémentation



Exemple d’implémentation

```
// Abstract Products
interface Button { void paint(); }
interface Checkbox { void paint(); }

// Concrete Products
class WindowsButton implements Button {
public void paint() {
System.out.println("Windows Button");
} }
class MacButton implements Button {
public void paint() {
System.out.println("Mac Button"); } }
class WindowsCheckbox implements Checkbox {
public void paint() {
System.out.println("Windows Checkbox"); } }
class MacCheckbox implements Checkbox {
public void paint() {
System.out.println("Mac Checkbox"); } }
```



Exemple d’implémentation

```
// Abstract Factory
interface GUIFactory {
    Button createButton();
    Checkbox createCheckbox();
}

// Concrete Factories
class WindowsFactory implements GUIFactory {
    public Button createButton() { return new
WindowsButton(); }
    public Checkbox createCheckbox() { return new
WindowsCheckbox(); }
}

class MacFactory implements GUIFactory {
    public Button createButton() { return new
MacButton(); }
    public Checkbox createCheckbox() { return new
MacCheckbox(); }
}
```

```
// Client
public class Application {
    private Button button;
    private Checkbox checkbox;

    public Application(GUIFactory factory) {
        button = factory.createButton();
        checkbox = factory.createCheckbox();
    }

    public void render() {
        button.paint();
        checkbox.paint();
    }

    public static void main(String[] args) {
        GUIFactory factory = new WindowsFactory(); //
ou MacFactory
        Application app = new Application(factory);
        app.render();
    }
}
```

Avantages et inconvénients

Avantages

- Créer des familles cohérentes d'objets.
- Le client est indépendant des classes concrètes.
- Facilite l'extension du code à de nouvelles familles de produits.

Inconvénients

- Ajouter un nouveau type de produit (nouvelle interface) demande de modifier toutes les factories concrètes.
- Peut devenir complexe si le nombre de produits et familles est élevé.

Exercice d’application

Problème :

Une entreprise de jeux vidéo veut développer un système pour gérer différents thèmes graphiques de son interface utilisateur.

Chaque thème définit un style pour deux éléments :

Fenêtre (**Window**) et Bouton (**Button**)

L’entreprise souhaite pouvoir changer facilement le thème graphique (ex. **Dark** ou **Light**) sans modifier le code principal du jeu.

Exercice d’application

Travail demandé :

1. Définir une modélisation de ce système en appliquant le pattern Abstract Factory
2. Implémenter les interfaces/classes définies
3. Tester en exécutant le jeu avec les deux thèmes (Dark et Light).