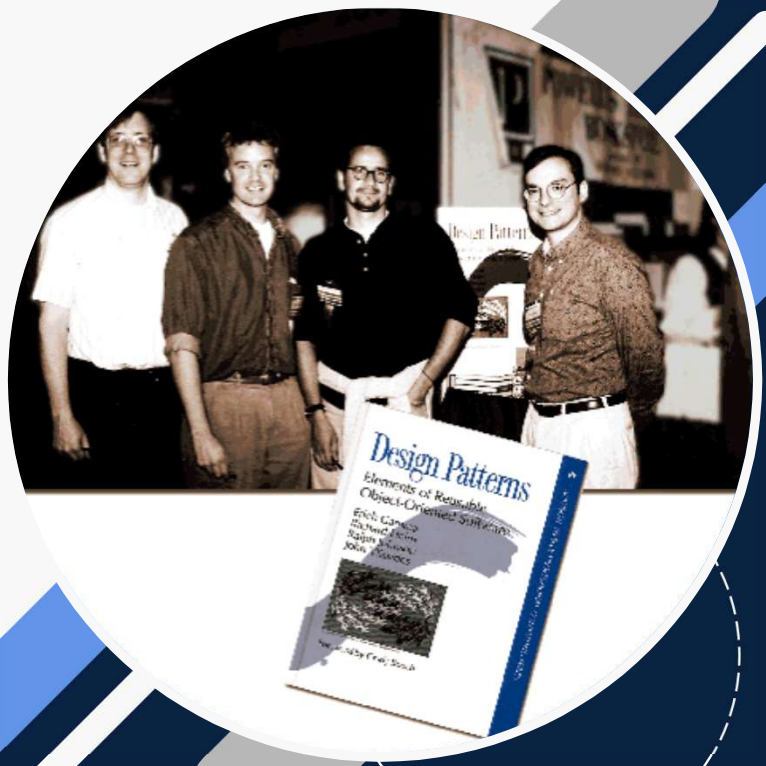
Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques
Département Informatique

Cours
Design Patterns et conception par contrats

ING 2 - Génie Logiciel

03

Patterns de création



Objectifs de ce chapitre

- Comprendre les enjeux de l'instanciation d'objets
  - Pourquoi la création d'objets est une source fréquente de problèmes de conception.
  - Différence entre instanciation directe (new) et via un pattern.
- Identifier les limites de l'instanciation classique
  - Couplage fort aux classes concrètes.
  - Difficulté à changer le type d'objets créés.
  - Complexité croissante avec la multiplicité objets.

Objectifs de ce chapitre

- Découvrir les principaux patterns de création (GoF)
- Appliquer les patterns à des cas pratiques
  - Simplifier l'instanciation d'objets complexes.
  - Découpler le code client de la logique de création.
  - Améliorer la flexibilité et la testabilité des applications.

Limites de l’instanciation classique

1. Couplage fort aux classes concrètes

Le code client dépend directement d’une implémentation précise.

Exemple :

```
Report report = new PDFReport(); // Couplé à PDF
```

→ Si on veut générer un ExcelReport, il faut modifier le code client partout.

Limites de l’instanciation classique

2. Manque de flexibilité

- Impossible de changer facilement le type d’objet créé à l’exécution (runtime).
- On est bloqué par le choix fait dans le code.

3. Difficulté à gérer des familles d’objets

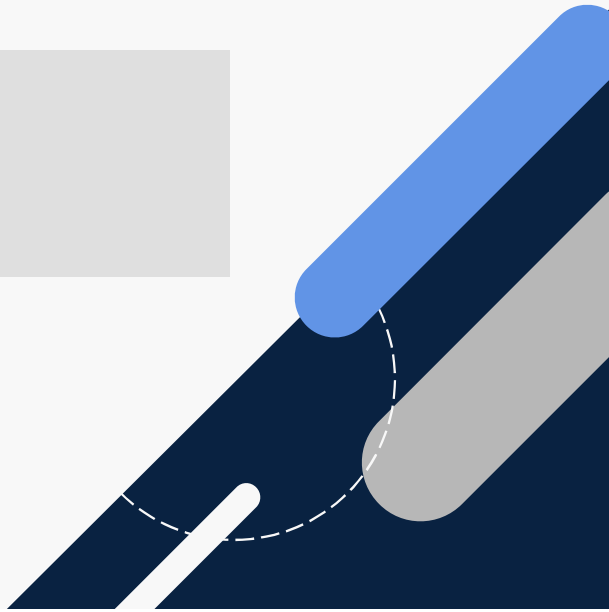
- Quand plusieurs objets liés doivent être créés ensemble, le code devient vite complexe.
- Exemple : créer une UI Windows ou UI MacOS avec plusieurs composants (boutons, menus, fenêtres…)

Limites de l’instanciation classique

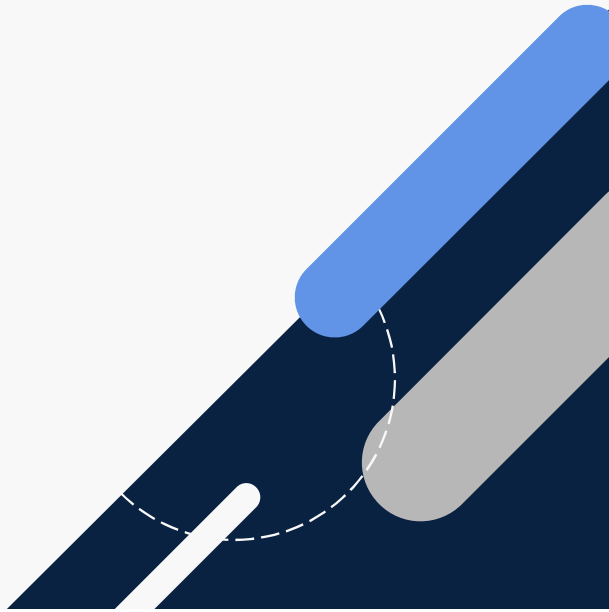
4. Maintenance coûteuse

- Chaque ajout de type d’objet impose des modifications dans plusieurs classes.
- Risque de duplication de logique de création.

new → simple mais rigide.
Patterns de création → apportent flexibilité, extensibilité et testabilité.



Pattern
Factory Method

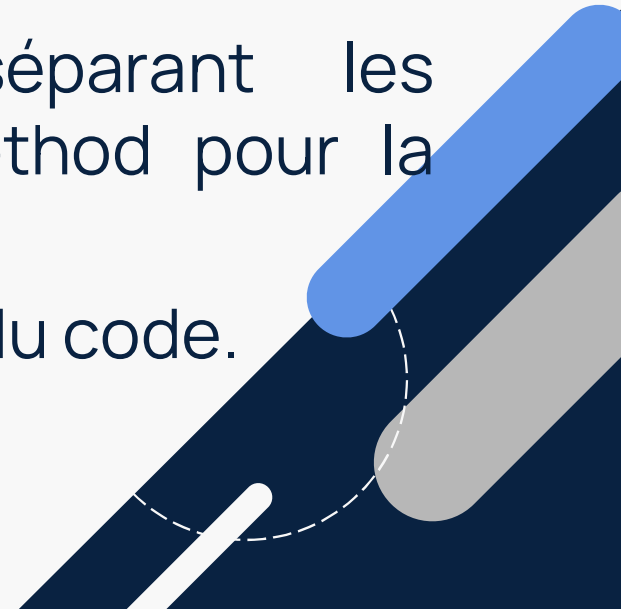


Définition

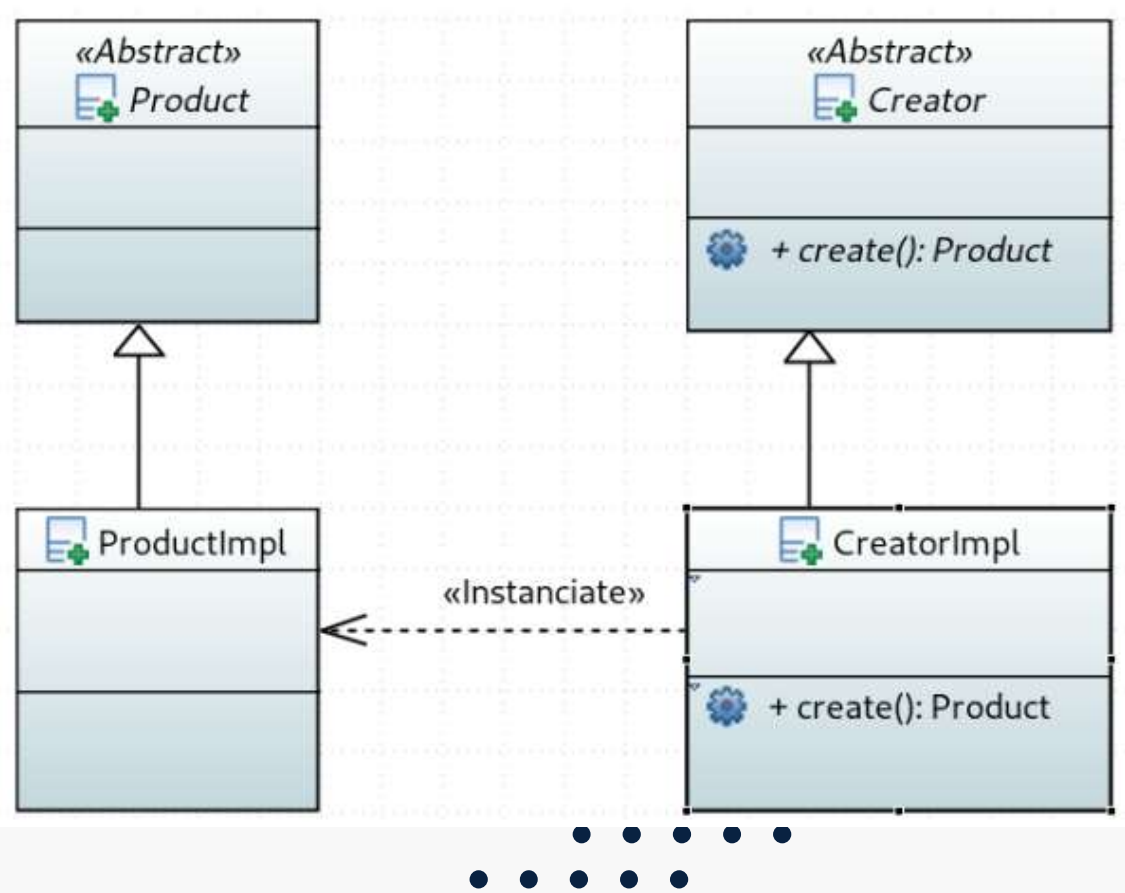
- Le Factory Method est un patron de création qui délègue l'instanciation des objets à une méthode spécifique (la "factory method") plutôt qu'à un appel direct à new.
- Il permet à une classe de laisser ses sous-classes décider du type d'objet à créer, améliorant ainsi la **flexibilité** et l'**extensibilité** du code.

Objectifs

- Encapsuler l'instanciation des objets concrets
- Libérer le code source de toute responsabilité concernant l'instanciation et la configuration des objets.
- Respecter le principe SRP en séparant les responsabilités: définir une Factory Method pour la création des objets concrets.
- Augmenter la flexibilité et la réutilisation du code.



Structure UML



Creator (abstrait) : définit la Factory Method (create()).

CreatorImpl : implémente la Factory Method pour retourner un type précis.

Product (abstrait) : définit l'interface commune des objets créés.

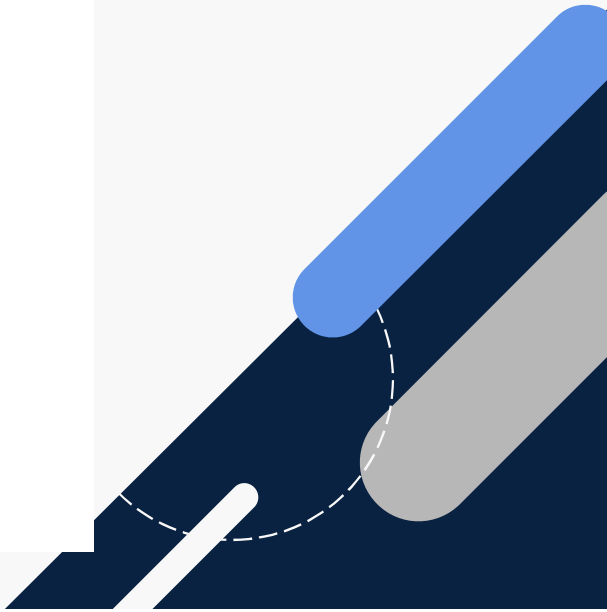
ProductImpl : implémentation concrète des objets créés.

Exemple d’implémentation

```
// Produit
interface Document {
    void open();
}

// Produits concrets
class PDFDocument implements Document {
    public void open() {
        System.out.println("Ouverture d'un document PDF");
    }
}

class WordDocument implements Document {
    public void open() {
        System.out.println("Ouverture d'un document Word");
    }
}
```



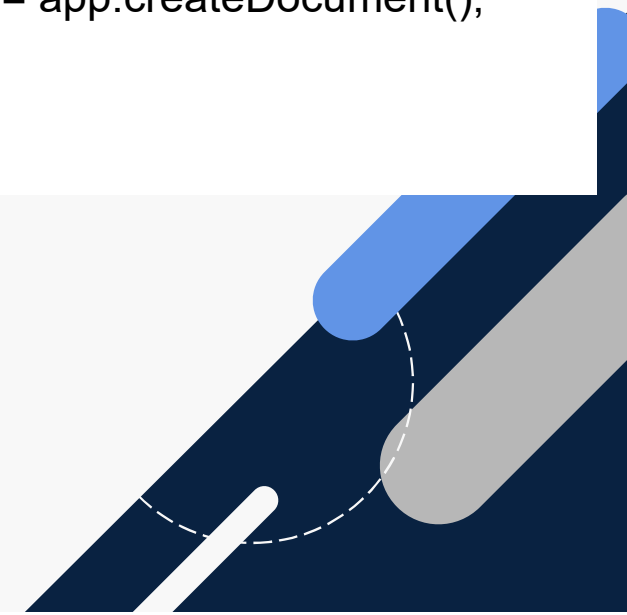
Exemple d’implémentation

```
// Créateur
abstract class Application {
    // Factory Method
    public abstract Document createDocument();
}

// Créateurs concrets
class PDFApplication extends Application {
    public Document createDocument() {
        return new PDFDocument();
    }
}

class WordApplication extends Application {
    public Document createDocument() {
        return new WordDocument();
    }
}
```

```
// Test
public class Main {
    public static void main(String[] args) {
        Application app = new PDFApplication();
// Choix flexible
        Document doc = app.createDocument();
        doc.open();
    }
}
```



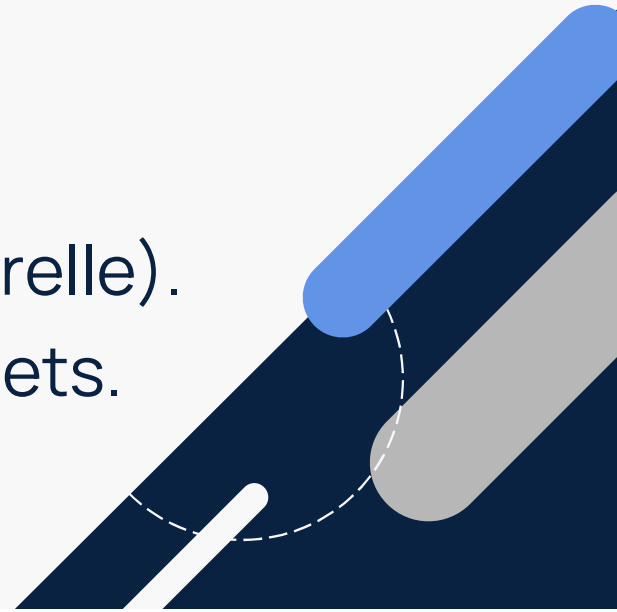
Avantages et inconvénients

Avantages

- Réduit le couplage aux classes concrètes.
- Facilite l'ajout de nouveaux produits sans modifier le client.
- Améliore la maintenabilité et la testabilité.

Inconvénients

- Plus de classes à gérer (surcharge structurelle).
- Peut sembler complexe pour de petits projets.



Exercice d’application

Problème :

Une application doit envoyer des notifications (Email, SMS, Push) à ses utilisateurs. On veut permettre d’étendre l’application par d’autres types de notifications telle que WhatsAppNotification.

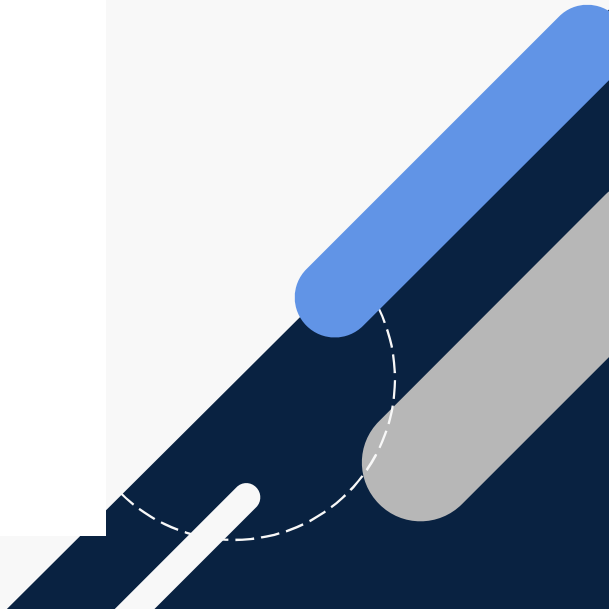
1. Proposez une solution à ce problème en appliquant le patron Factory Method.
2. Ajoutez un nouveau type WhatsAppNotification sans modifier le code existant.

Exercice d’application

```
// Produit
interface Notification {
    void notifyUser();
}

// Produits concrets
class EmailNotification implements Notification {
    public void notifyUser() {
        System.out.println("Envoi d'un email...");
    }
}

class SMSNotification implements Notification {
    public void notifyUser() {
        System.out.println("Envoi d'un SMS...");
    }
}
```



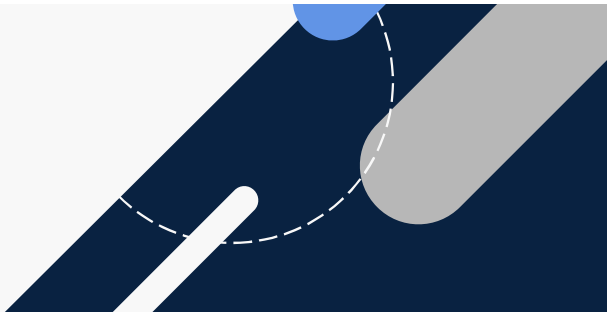
Exercice d’application

```
// Créateur
abstract class NotificationFactory {
    public abstract Notification createNotification();
}

// Créateurs concrets
class EmailFactory extends NotificationFactory {
    public Notification createNotification() {
        return new EmailNotification();
    }
}

class SMSFactory extends NotificationFactory {
    public Notification createNotification() {
        return new SMSNotification();
    }
}
```

```
// Utilisation
public class Main {
    public static void main(String[] args) {
        NotificationFactory factory = new EmailFactory();
        Notification notification = factory.createNotification();
        notification.notifyUser();
    }
}
```



Travail à domicile

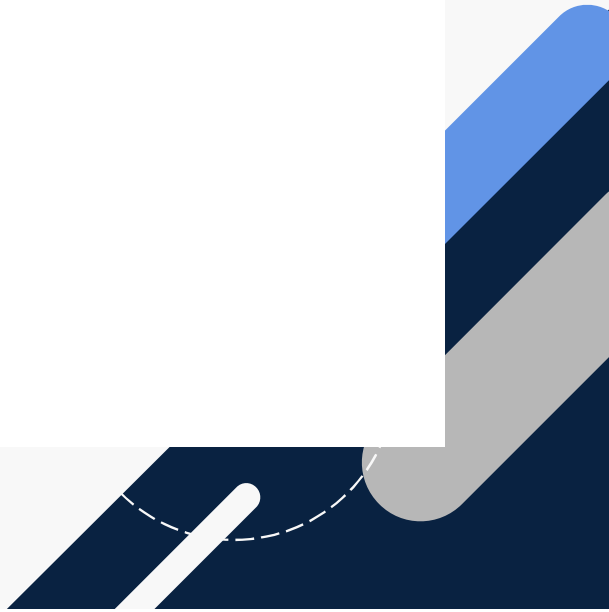
On souhaite créer une classe Logger permettant à des utilisateurs d'afficher des informations sur un programme lors de son exécution.

Un Logger est un objet possédant trois méthodes :

- logInfo(String message)
- logDebug(String message)
- logError(String message)

Un Logger est construit avec un état qui peut être :

- LogLevel.INFO
- LogLevel.DEBUG
- LogLevel.ERROR



Travail à domicile

Implémentez le système suivant :

1. Définir une énumération LogLevel.
2. Créer une interface Logger avec les trois méthodes ci-dessus.
3. Implémenter trois classes concrètes :

InfoLogger : n'affiche que les messages INFO.
DebugLogger : affiche DEBUG et INFO.
ErrorLogger : affiche seulement ERROR.

4. Créer une factory (LoggerFactory) avec une méthode createLogger(LogLevel level).Cette méthode retourne le logger correspondant au niveau demandé.
5. Écrire un programme de test qui crée un Logger via la factory et affiche des messages avec différents niveaux.