INF4067

DEVOIR N°1 PORTANT SUR LE PATRON DE CONSTRUCTION

Etudiant :

NOM	MATRICULE
NOUBISSIE KAMGA WILFRIED	20U2671

Master I

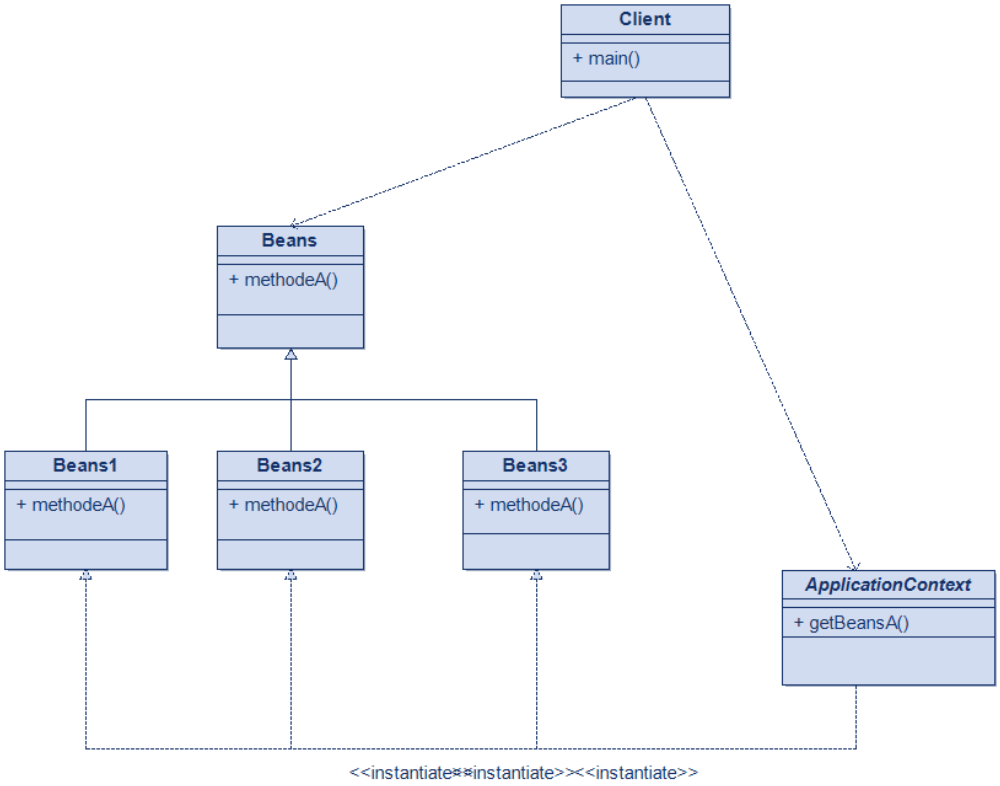
INFORMATIQUE FONDA

Table des matières

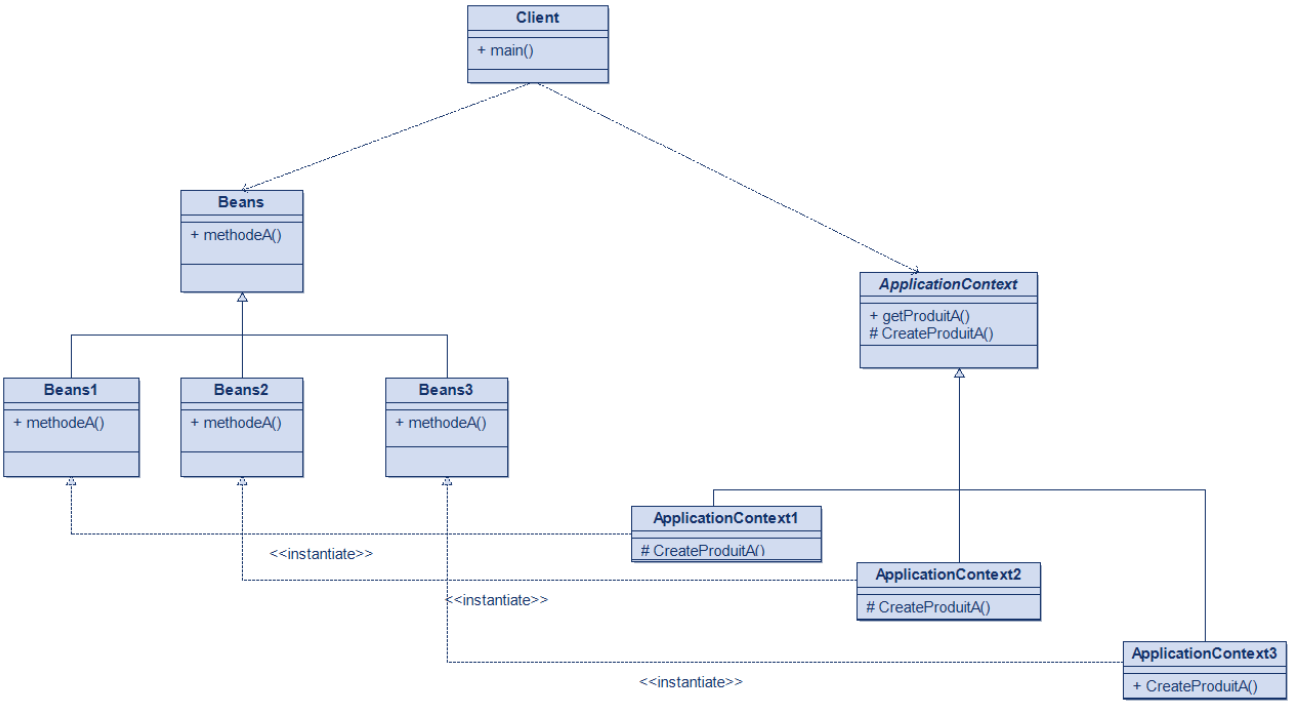
I. LE PATRON FACTORY

1) Schéma

a) *Avec une seule fabrique pour tous les types de produit*



b) *Avec une fabrique abstraite et des fabriques concrètes*



2) Bibliothèque java qui l'implémente

Une bibliothèque java courante pour l'implémentation du patron Factory est **Spring Framework,** avec l'utilisation des **@Beans** et l'**inversion de contrôle (IoC).** Ci-dessous, un exemple d'utilisation :

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

interface Personne {
    void salutation();
}

class Femme implements Personne {
    public void salutation() {
        System.out.println("Bonjour madame !");
    }
}

class Homme implements Personne {
    public void salutation() {
        System.out.println("Bonjour monsieur !");
    }
}
```

```
@Configuration
class PersonneFactory {
    @Bean
    public Personne getPersonne(String genre) {
        switch (genre.toLowerCase()) {
            case "femme":
                return new Femme();
            case "homme":
                return new Homme();
            default:
                throw new IllegalArgumentException("Genre inconnu");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext(PersonneFactory.class);

        Personne femme = context.getBean("getPersonne", "femme");
        femme.salutation(); // Resultat: Bonjour madame !

        Personne homme = context.getBean("getPersonne", "homme");
        homme.salutation(); // Resultat: Bonjour monsieur !
    }
}
```

II. LE PATRON SINGLETON

1) Schéma

2) Bibliothèque java qui l'implémente

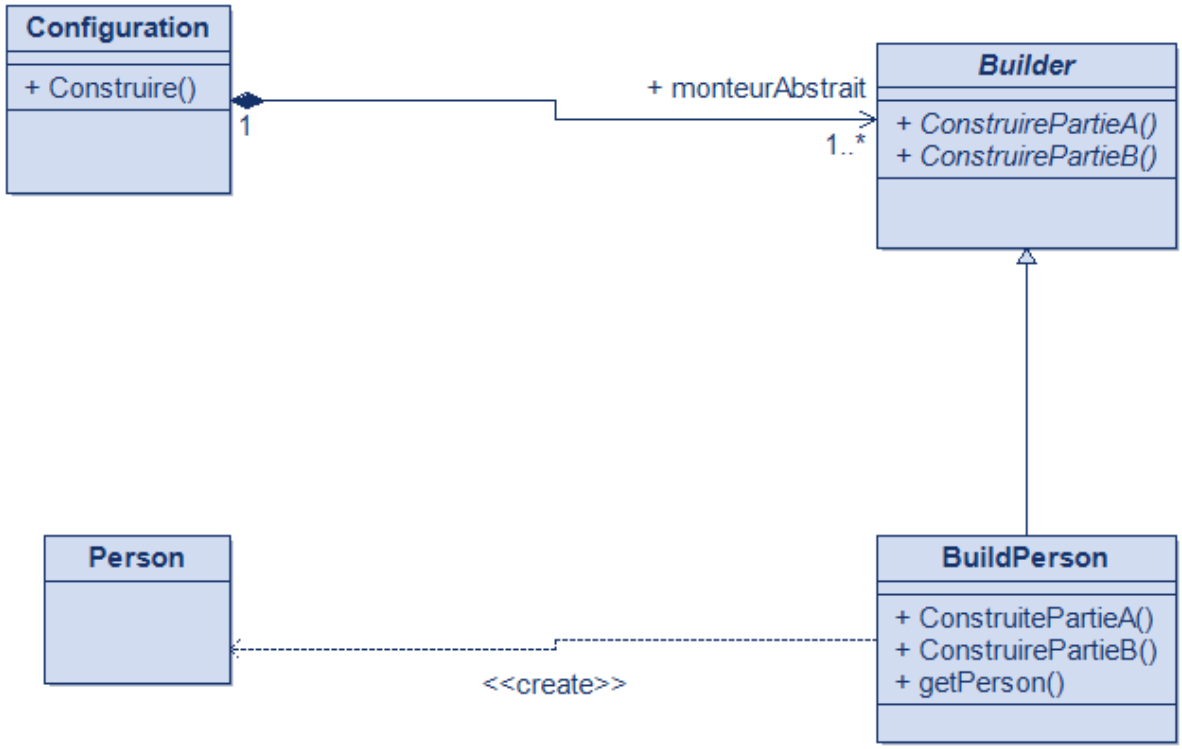
Pour l'implémentation du patron singleton en java, nous pouvons utiliser la bibliothèque **Guava**. Ci-dessous, un exemple d'utilisation :

```
import com.google.common.base.Supplier;
import com.google.common.base.Suppliers;

public class Singleton {
    // Instance unique avec Guava
    private static final Supplier<Singleton> INSTANCE = Suppliers.memoize(Singleton::new);

    // Constructeur privé pour empêcher l'instanciation
    private Singleton() {}

    // Méthode pour obtenir l'instance unique
    public static Singleton getInstance() {
        return INSTANCE.get();
    }
```

```
    // Méthode d'exemple
    public void displayMessage() {
        System.out.println("Hello from Guava Singleton!");
    }
}

public class Main {
    public static void main(String[] args) {
        Singleton singleton = Singleton.getInstance();
        singleton.displayMessage(); // Affiche : Hello from Guava Singleton!
    }
}
```

III. LE PATRON BUILDER

1) Schéma



2) Bibliothèque java qui l'implémente

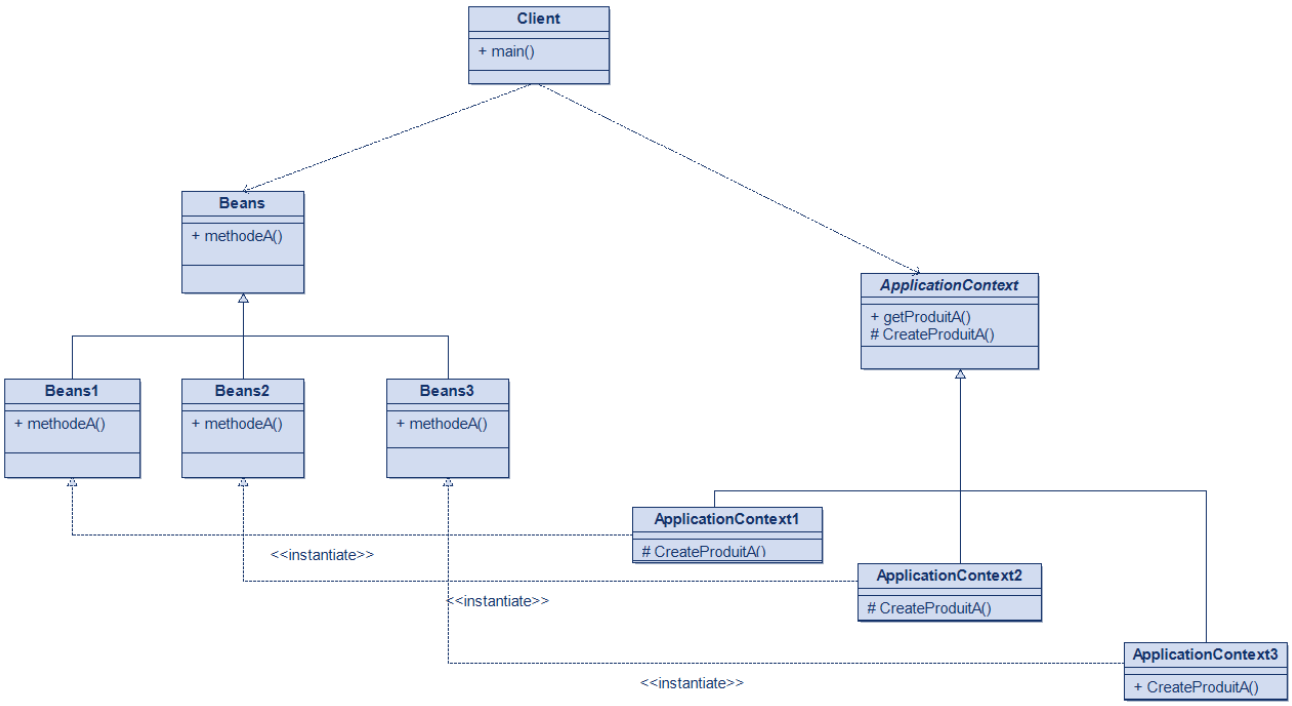
Une bibliothèque très populaire ici pour l'implémentation du patron Builder est la bibliothèque **Lombok.** Ci-dessous, un exemple d'implémentation :

```
import lombok.Builder;

import lombok.Getter;


@Getter

@Builder

public class Person {

    private final String nom;

    private final String matricule;
```

```
    private final int telephone ;
}

public class Main {
    public static void main(String[] args) {
        Person person = Person.builder()
                .nom("Noubissie Kamga Wilfried")
                .matricule("20U2671")
                .telephone(690232120)
                .build();

        System.out.println("Nom : " + person.getNom() + " --- " + person.getMatricule());
        System.out.println("Numéro : " + person.getTelephone());
    }
}
```

IV. LE PATRON ABSTRACT FACTORY

1. Schéma



2. Bibliothèque java qui l'implémente

Une bibliothèque java qui implémente le patron de construction Abstract Factory est la bibliothèque **JavaFX.** Ci-dessous, un exemple d'implémentation :

```
// Interface pour les bouton
interface Button {
    void paint();
}

// Interface pour la Factory
interface GUIFactory {
```

```
    Button createButton();
}

// Implémentation de Button pour Windows
class WindowsButton implements Button {
    public void paint() {
        System.out.println("Windows Button");
    }
}

// Implémentation de Button pour Mac
class MacButton implements Button {
    public void paint() {
        System.out.println("Mac Button");
    }
}

// Implémentation de GUIFactory pour Windows
class WindowsFactory implements GUIFactory {
    public Button createButton() {
        return new WindowsButton();
    }
}

// Implémentation de GUIFactory pour Mac
class MacFactory implements GUIFactory {
    public Button createButton() {
```

```
        return new MacButton();
    }
}

public class Main {
    private static GUIFactory getFactory(String type) {
        if (type.equals("Windows")) {
            return new WindowsFactory();
        } else if (type.equals("Mac")) {
            return new MacFactory();
        }
        return null;
    }

    public static void main(String[] args) {
        GUIFactory factory = getFactory("Windows");
        Button button = factory.createButton();
        button.paint(); // Affiche : Windows Button

        factory = getFactory("Mac");
        button = factory.createButton();
        button.paint(); // Affiche : Mac Button
    }
}
```