Фабричный метод и абстрактная фабрика

Выполнили
Шевцов Дмитрий
Григорьева Анастасия

Фабричный метод

Определение🤓

Фабричный метод — порождающий паттерн проектирования, определяет интерфейс для создания объекта в родительском классе.
Объекты создаются внутри подклассов - конкретных создателей.

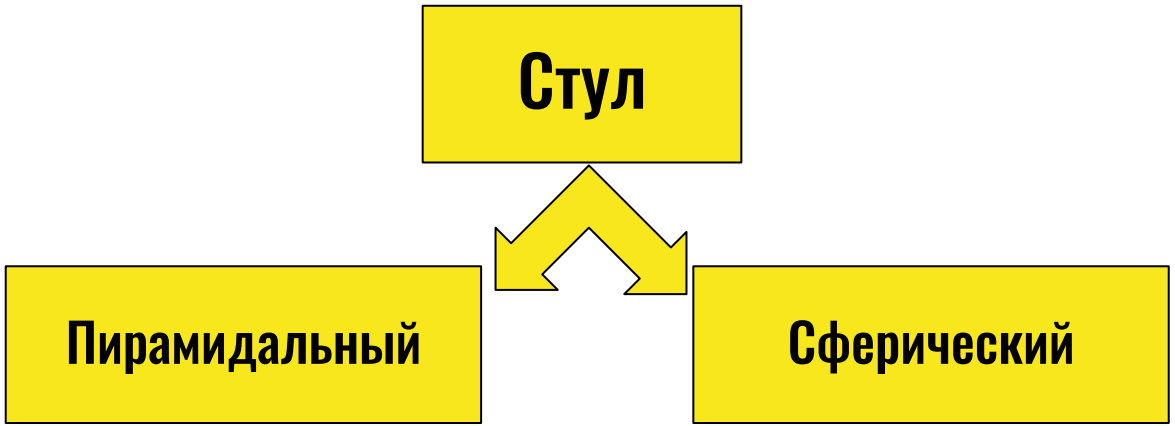
Фабричный метод

Причины использования

I. Неизвестна спецификация объекта

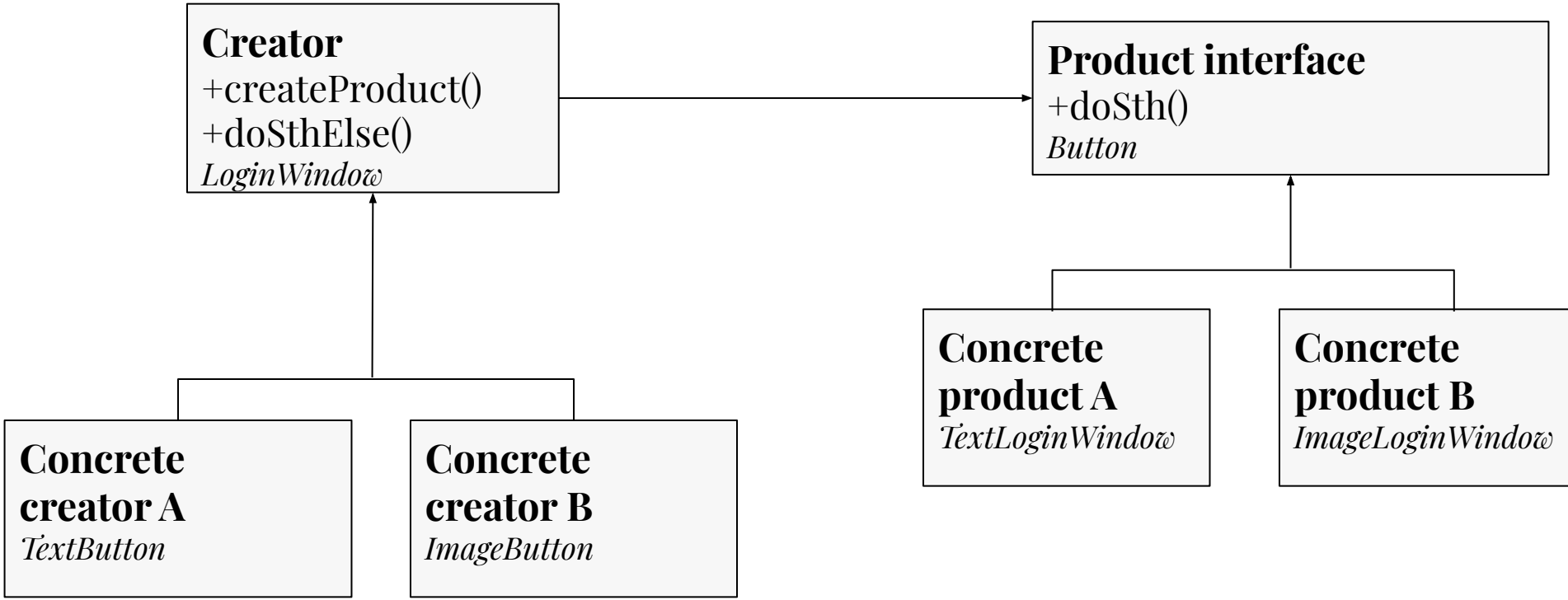
II. Отделить логику создания объектов

III. Независимость от способа создания новых объектов



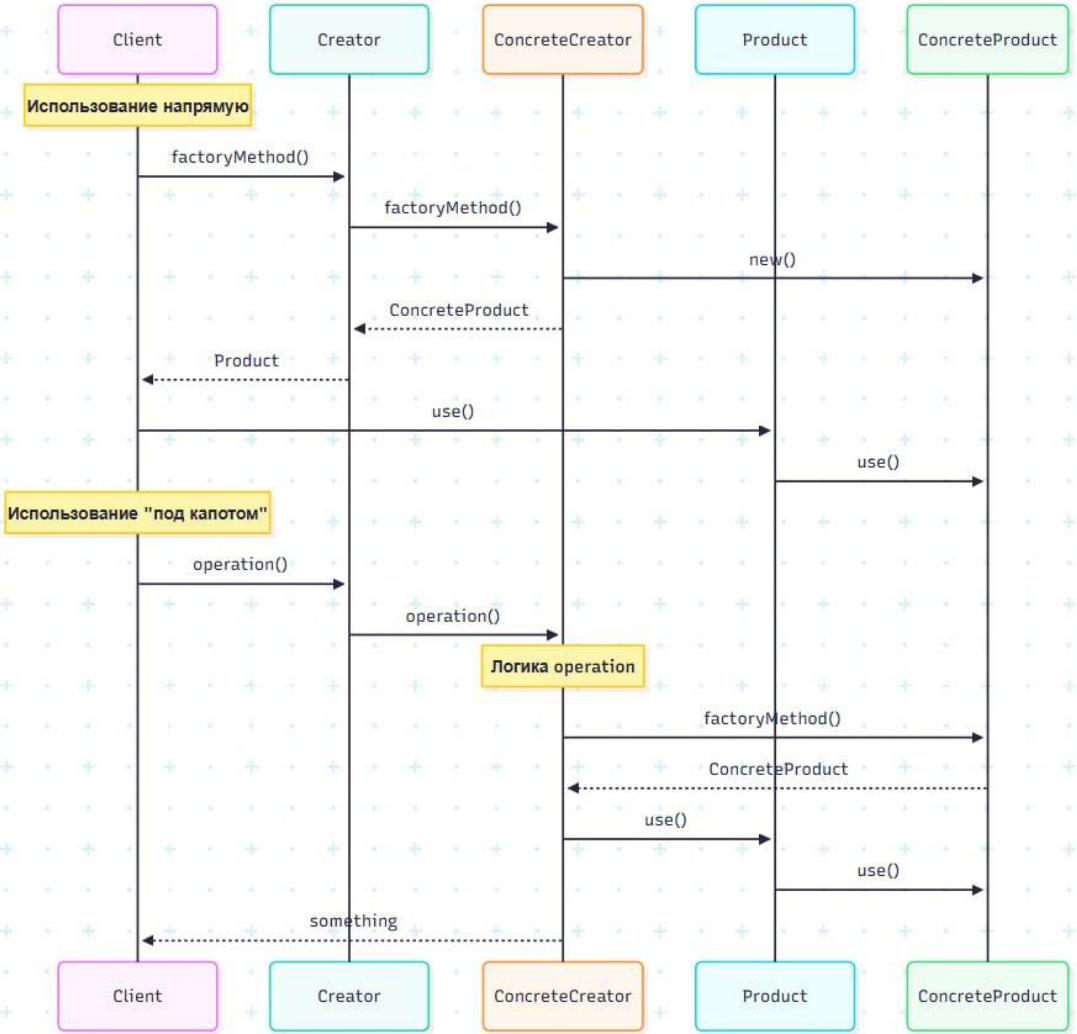
Фабричный метод

Схема



Фабричный метод

Диаграмма последовательности



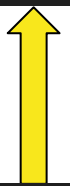
Фабричный метод

Код как на схеме

```
//Creator
abstract class LoginWindow {
    public abstract Button createButton();

    public void open() {
        System.out.println("login window");
        Button button = createButton();
        button.render();
    }
}
```

```
//Product
interface Button {
    void render();
}
```



```
//ConcreteCreatorA
class TextLoginWindow extends LoginWindow {
    @Override
    public Button createButton() {
        return new TextButton();
    }
}
```

```
//ConcreteProductA
class TextButton implements Button {
    @Override
    public void render() {
        System.out.println(x: "button with text");
    }
}
```

Фабричный метод

```java
//Product
interface Button {
    void render();
}

//ConcreteProductA
class TextButton implements Button {
    @Override
    public void render() {
        System.out.println(x: "button with text");
    }
}

//ConcreteProductB
class ImageButton implements Button {···

//ConcreteProductC
class DropdownButton implements Button {···

//Creator
abstract class LoginWindow {
    public abstract Button createButton();

    public void open() {
        System.out.println(x: "login window");
        Button button = createButton();
        button.render();
    }
}
```

Код (использование)

```java
//ConcreteCreatorA
class TextLoginWindow extends LoginWindow {
    @Override
    public Button createButton() {
        return new TextButton();
    }
}

//ConcreteCreatorB
class ImageLoginWindow extends LoginWindow {···

//ConcreteCreatorC
class DropdownLoginWindow extends LoginWindow {···

public class Main {
    Run | Debug
    public static void main(String[] args) {
        System.out.println(x: "text-based login window");
        LoginWindow window = new TextLoginWindow();
        window.open();
    }
}
```

Вывод

```
text-based login window
login window
button with text
```

Абстрактная фабрика

Определение🤓

Абстрактная фабрика — порождающий паттерн проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Абстрактная фабрика

Причины использования

I. Неизвестна спецификация объекта

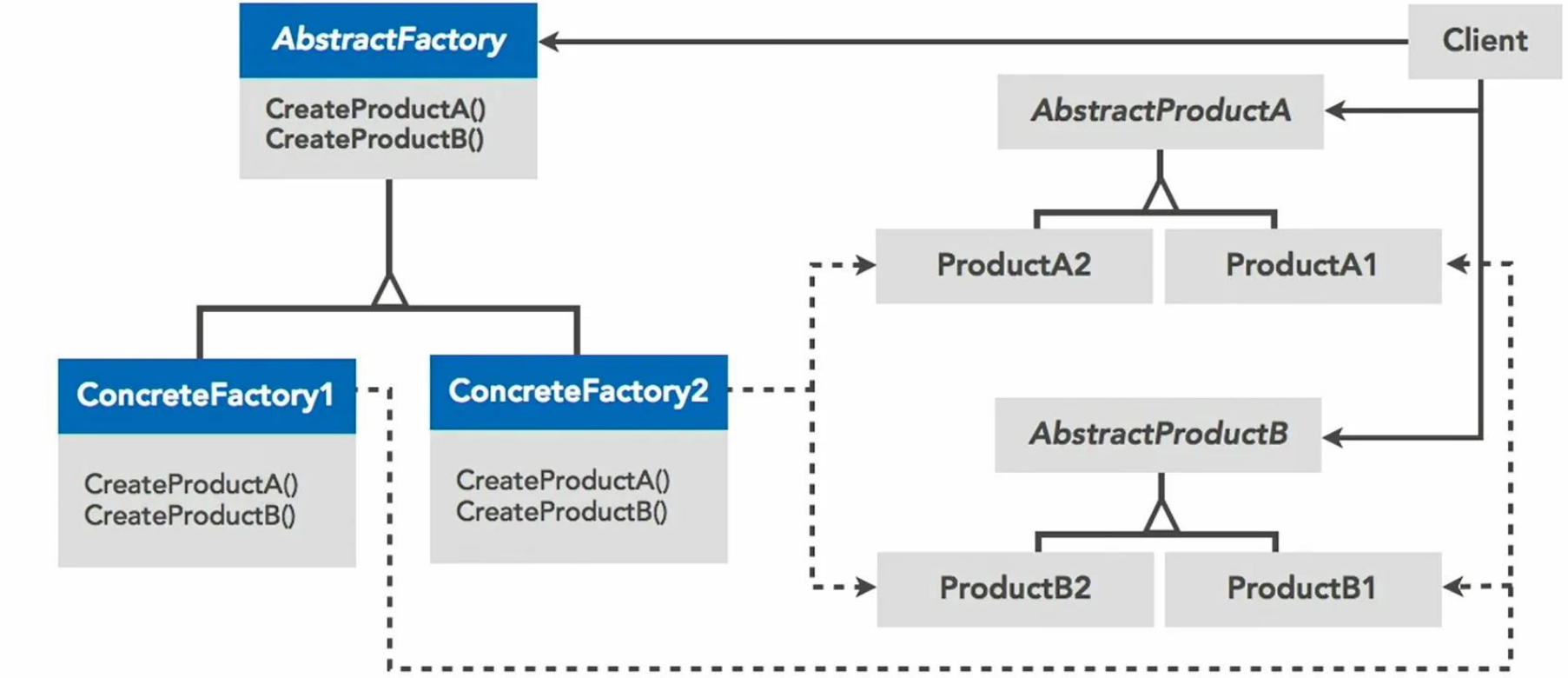
II. Отделить логику создания объектов от интерфейсов

III. Система полностью реализуема от одного семейства объектов

IV. Объекты взаимосвязаны и спроектированы для совместной работы

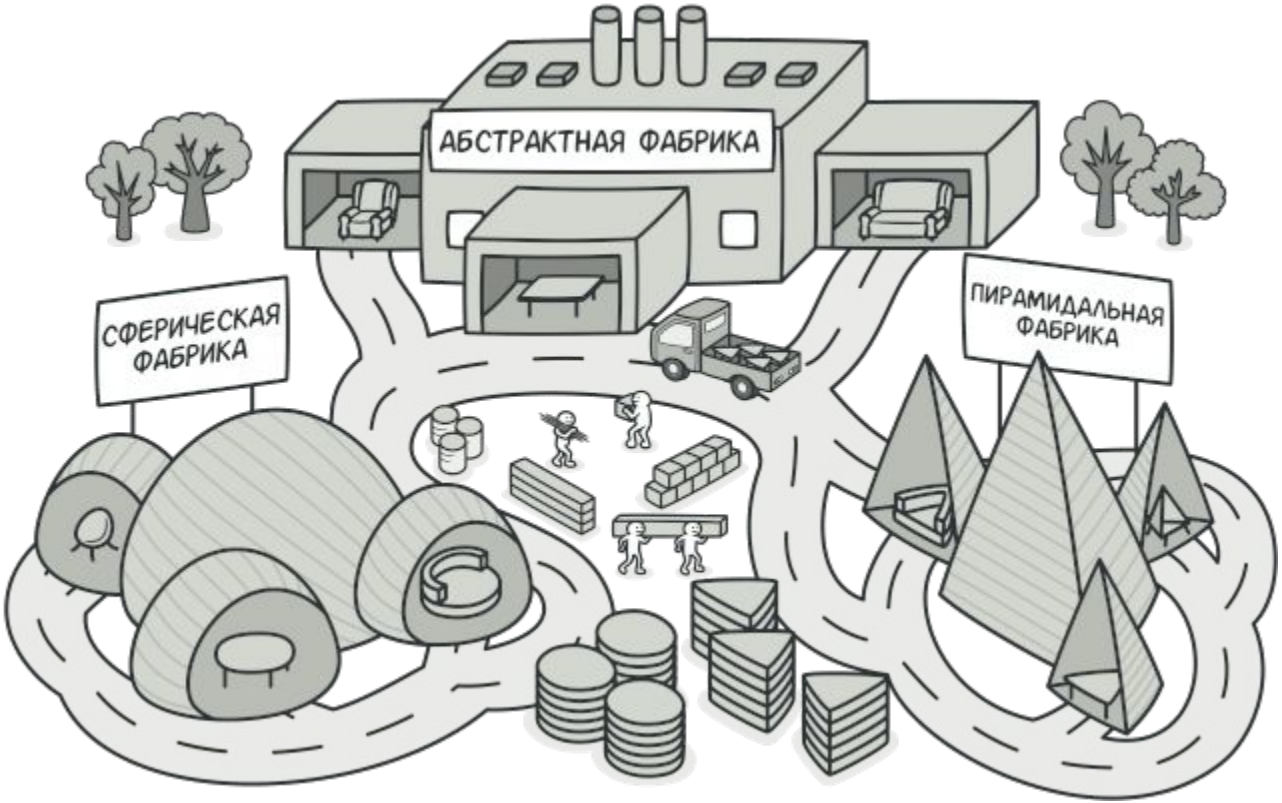
Абстрактная фабрика

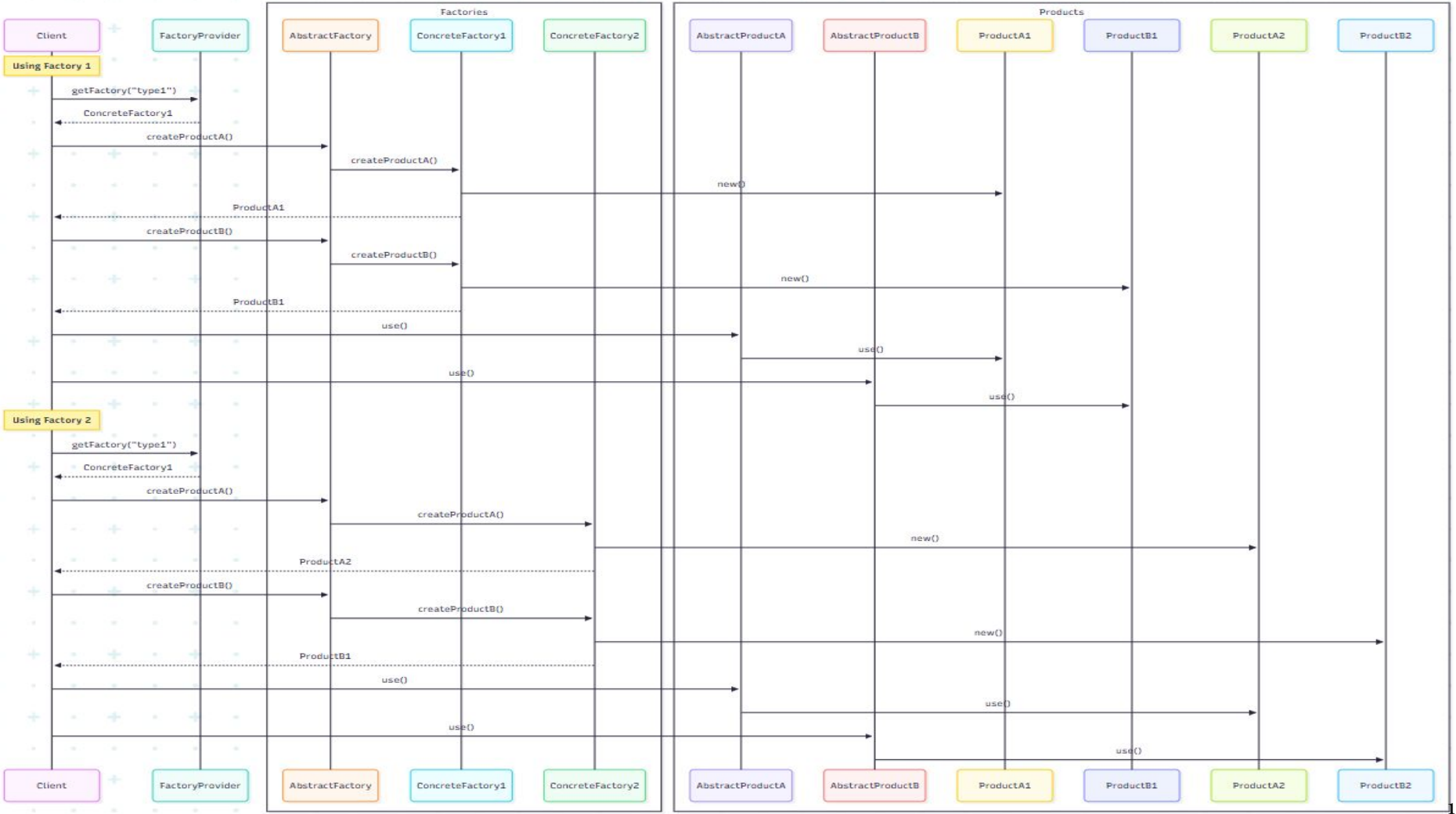
Схема

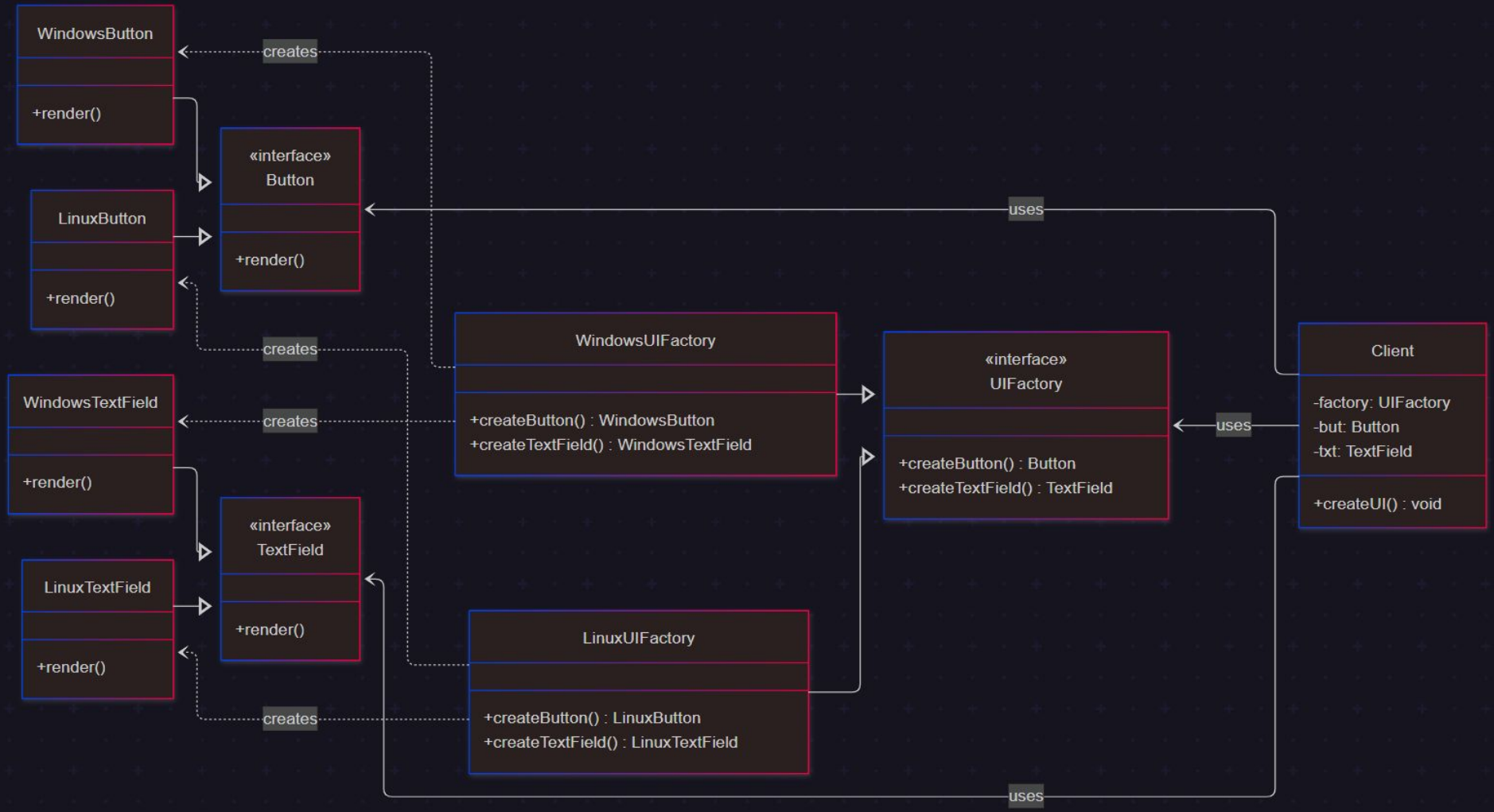


Абстрактная фабрика

Схема




Factories
Products
Client
FactoryProvider
AbstractFactory
ConcreteFactory1
ConcreteFactory2
AbstractProductA
AbstractProductB
ProductA1
ProductB1
ProductA2
ProductB2
Using Factory 1
getFactory("type1")
ConcreteFactory1
createProductA()
createProductA()
new()
ProductA1
createProductB()
createProductB()
new()
ProductB1
use()
use()
use()
use()
Using Factory 2
getFactory("type1")
ConcreteFactory1
createProductA()
createProductA()
new()
ProductA2
createProductB()
createProductB()
new()
ProductB1
use()
use()
use()
use()


WindowsButton
+render()
creates
«interface»
Button
+render()
LinuxButton
+render()
uses
creates
WindowsUIFactory
+createButton() : WindowsButton
+createTextField() : WindowsTextField
«interface»
UIFactory
+createButton() : Button
+createTextField() : TextField
uses
Client
-factory: UIFactory
-but: Button
-txt: TextField
+createUI() : void
WindowsTextField
+render()
creates
«interface»
TextField
+render()
LinuxTextField
+render()
LinuxUIFactory
+createButton() : LinuxButton
+createTextField() : LinuxTextField
creates
uses

Абстрактная фабрика

Пример

```java
// ProductA
interface Button {
    void render();
}

// ConcreteProductA1
class WindowsButton implements Button {
    @Override
    public void render() {
        System.out.println(x: "Rendering Windows button");
    }
}

// ConcreteProductA2
class LinuxButton implements Button {
    @Override
    public void render() {
        System.out.println(x: "Rendering Linux button");
    }
}
```

```java
// ProductB
interface TextField {
    void render();
}

// ConcreteProductB1
class WindowsTextField implements TextField {
    @Override
    public void render() {
        System.out.println(x: "Rendering Windows text field");
    }
}

// ConcreteProductB2
class LinuxTextField implements TextField {
    @Override
    public void render() {
        System.out.println(x: "Rendering Linux text field");
    }
}
```

Абстрактная фабрика

Пример

```java
// Creator
interface UIFactory {
    Button createButton();
    TextField createTextField();
}
```

```java
// ConcreteCreator for Windows
class WindowsUIFactory implements UIFactory {
    @Override
    public Button createButton() {
        System.out.println(x: "Creating Windows button");
        return new WindowsButton();
    }

    @Override
    public TextField createTextField() {
        System.out.println(x: "Creating Windows text field");
        return new WindowsTextField();
    }
}
```

```java
// ConcreteCreator for Linux
class LinuxUIFactory implements UIFactory {
    @Override
    public Button createButton() {
        System.out.println(x: "Creating Linux button");
        return new LinuxButton();
    }

    @Override
    public TextField createTextField() {
        System.out.println(x: "Creating Linux text field");
        return new LinuxTextField();
    }
}
```

Абстрактная фабрика

Пример

```
public static void main(String[] args) {
    String osName = System.getProperty(key: "os.name").toLowerCase();
    UIFactory factory;
    if (osName.contains(s: "win")) {
        factory = new WindowsUIFactory();
    } else if (osName.contains(s: "linux")) {
        factory = new LinuxUIFactory();
    } else {
        throw new IllegalArgumentException("Unsupported OS: " + osName);
    }
    System.out.println("Detected OS: " + osName + "\n");

    Button but = factory.createButton();
    TextField txt = factory.createTextField();

    System.out.println(x: "\nRendering Button");
    but.render();

    System.out.println(x: "\nRendering TextField");
    txt.render();
}
```

Сравнение паттернов

Критерий	Фабричный метод	Абстрактная фабрика
Что создает	Один тип продукта (стул)	Семейство продуктов (стул+стол+диван)
Сколько методов создания	Один (фабричный метод)	Несколько (createChair, createTable…)
Когда использовать	Один тип объекта с вариациями	Нужны согласованные наборы объектов

Источники

“Паттерны объектно-ориентированного программирования” Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес

https://habr.com/ru/articles/465835

https://refactoringu.ru/ru/design-patterns/abstract-factory.html

https://refactoringu.ru/ru/design-patterns/factory-method.html

https://javarush.com/groups/posts/2372--patternih-proektirovanija-factorymethod