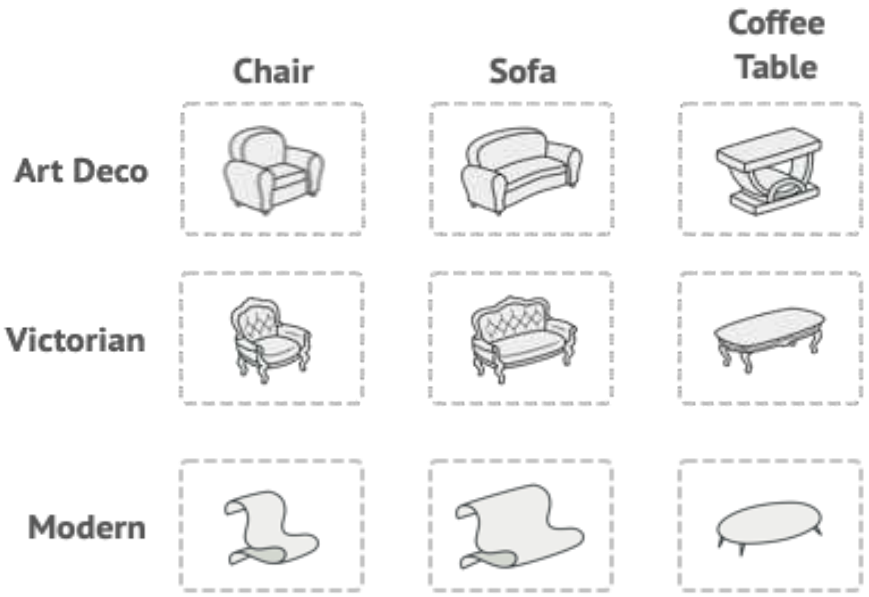
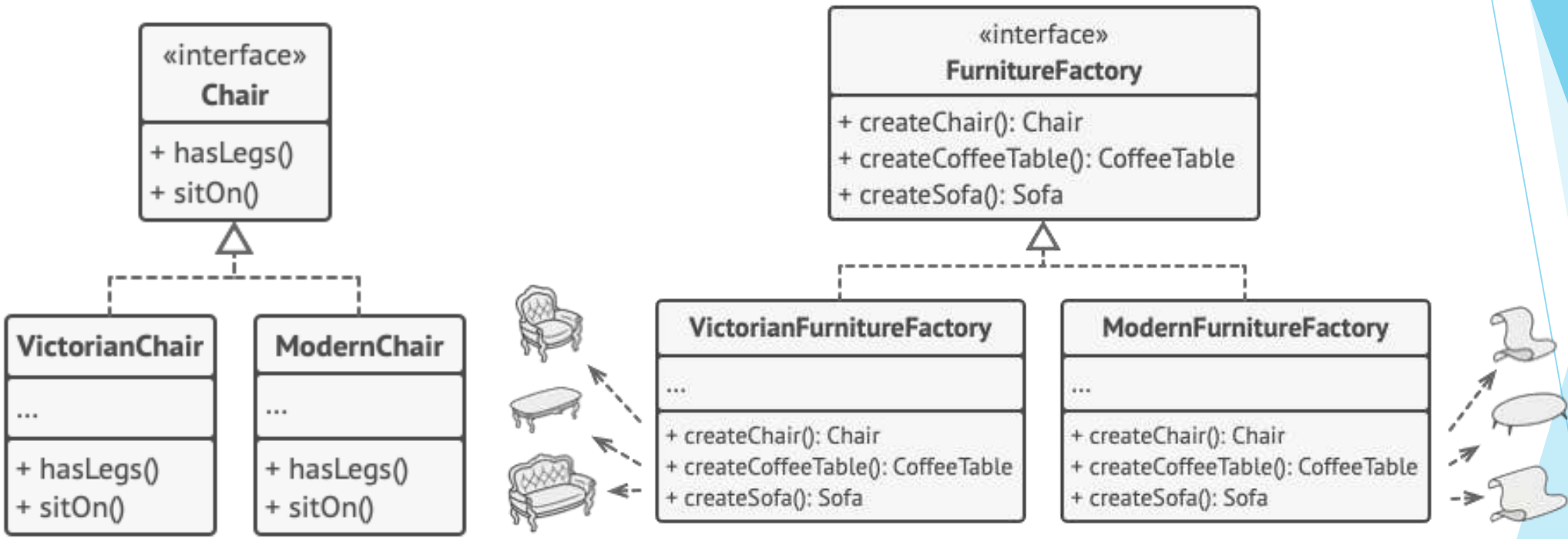
Abstract Factory

Problém:
Tvorba simulátoru obchodu s nábytkem

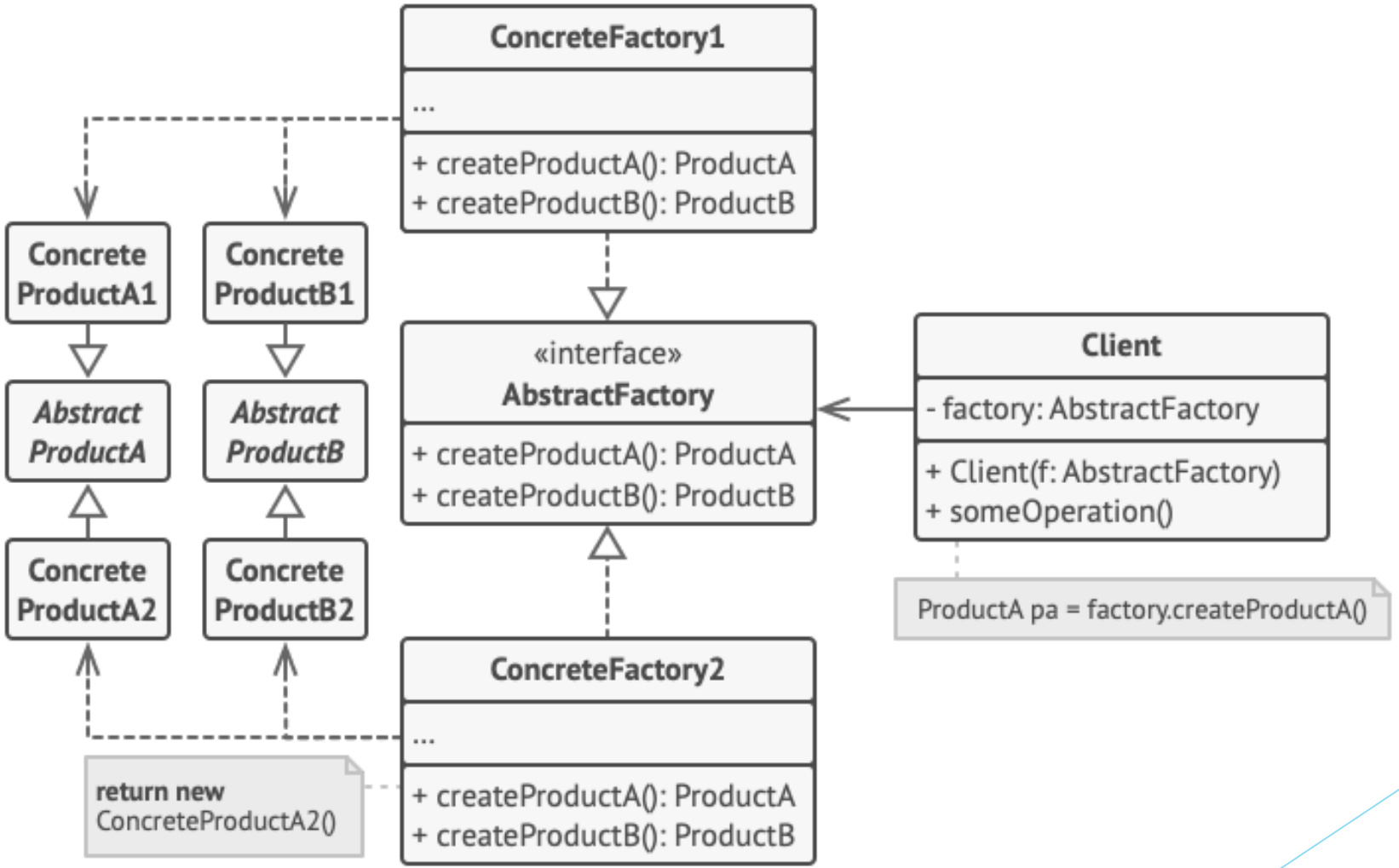
- Sady souvisejících produktů
- Několik variant této rodiny



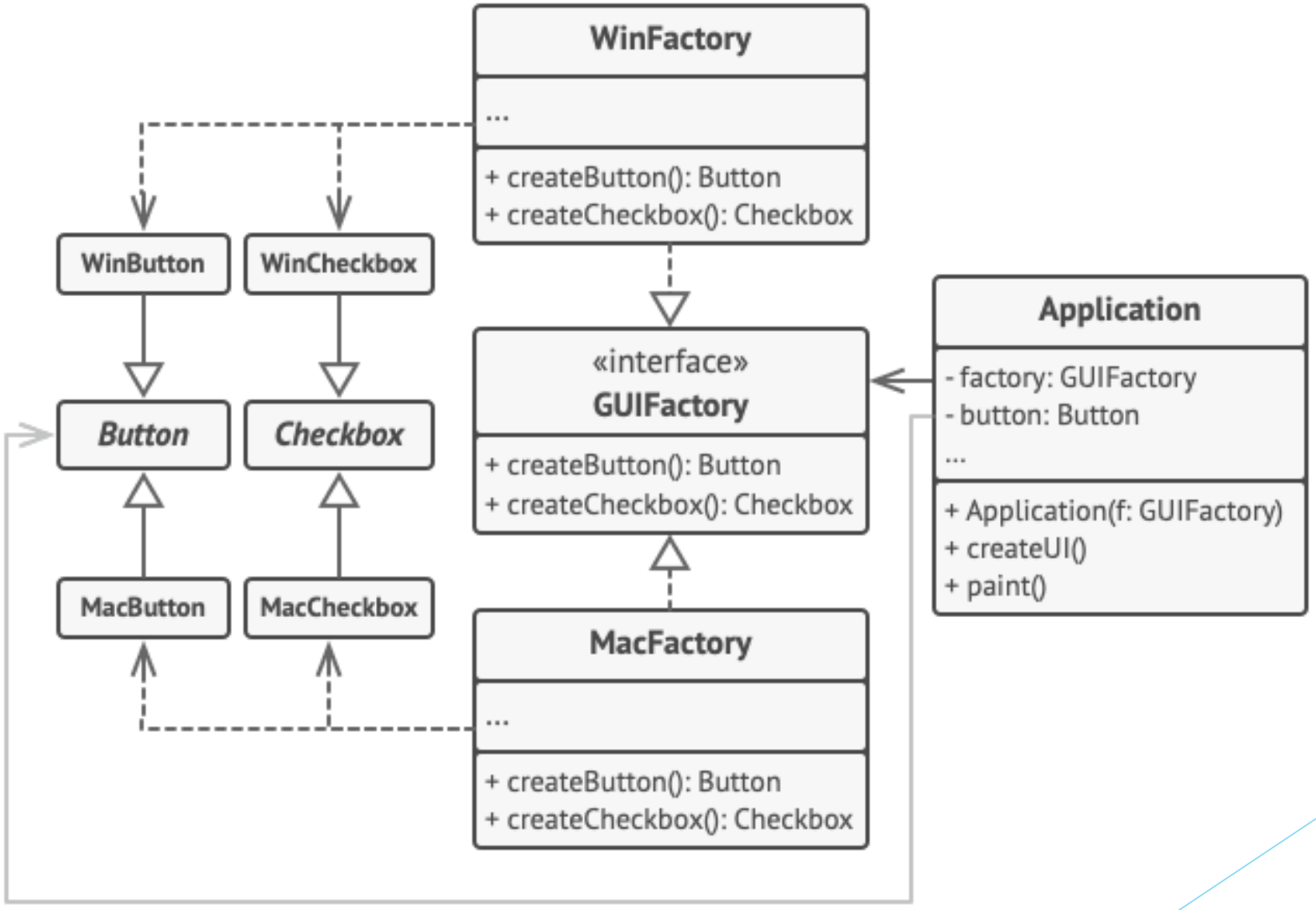
Řešení


«interface»
Chair
+ hasLegs()
+ sitOn()
VictorianChair
...
+ hasLegs()
+ sitOn()
ModernChair
...
+ hasLegs()
+ sitOn()
«interface»
FurnitureFactory
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa
VictorianFurnitureFactory
...
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa
ModernFurnitureFactory
...
+ createChair(): Chair
+ createCoffeeTable(): CoffeeTable
+ createSofa(): Sofa

Obecně


ConcreteFactory1
...
+ createProductA(): ProductA
+ createProductB(): ProductB
Concrete ProductA1
Concrete ProductB1
Abstract ProductA
Abstract ProductB
Concrete ProductA2
Concrete ProductB2
«interface»
AbstractFactory
+ createProductA(): ProductA
+ createProductB(): ProductB
Client
- factory: AbstractFactory
+ Client(f: AbstractFactory)
+ someOperation()
ProductA pa = factory.createProductA()
ConcreteFactory2
...
+ createProductA(): ProductA
+ createProductB(): ProductB
return new ConcreteProductA2()

Příklad


WinFactory
...
+ createButton(): Button
+ createCheckbox(): Checkbox
WinButton
WinCheckbox
«interface»
GUIFactory
+ createButton(): Button
+ createCheckbox(): Checkbox
Application
- factory: GUIFactory
- button: Button
...
+ Application(f: GUIFactory)
+ createUI()
+ paint()
Button
Checkbox
MacButton
MacCheckbox
MacFactory
...
+ createButton(): Button
+ createCheckbox(): Checkbox

Příklad

```
public interface IGUIFactory {
    IButton CreateButton();
    ICheckbox CreateCheckbox();
}

public interface IButton {
    void Paint();
}


public interface ICheckbox {
    void Paint();
}
```

rozhraní GUIFactory s metodami pro vytváření abstraktních produktů

Každý samostatný produkt by měl mít interface.

Příklad

```
public class WinFactory : IGUIFactory {
    public IButton CreateButton() {
        return new WinButton();
    }

    public ICheckbox CreateCheckbox() {
        return new WinCheckbox();
    }
}

public class MacFactory : IGUIFactory {
...
}
```

Konkrétní továrna pre Windows a MAC Gui prvky

```
public class WinButton : IButton {
    public WinButton() {
        ...
    }

    public void Paint() {
        ...
    }
}

public class MacButton : IButton {
...
}

// Konkrétní produkt pro Windows Checkbox
// Konkrétní produkt pro Mac Checkbox
```

Příklad

```
public class Application {
    private IGUIFactory _factory;
    private IButton _button;

    public Application(IGUIFactory factory) {
        _factory = factory;
    }

    public void CreateUI() {
        _button = _factory.CreateButton();
    }

    public void Paint() {
        _button.Paint();
    }
}
```

```
class ApplicationConfigurator {
    static void Main(string[] args) {
        IGUIFactory factory;
        if (OS == "Windows") {
            factory = new WinFactory();
        } else if (OS == "Mac") {
            ...
        }

        Application app = new(factory);
        app.CreateUI();
        app.Paint();
    }
}
```

Klientský kód pracuje s továrnami a produkty prostřednictvím abstraktních typů

Výhody a nevýhody

Výhody

- Klient je izolován od konkrétních tříd
- Konzistence mezi produkty
- Změna nebo rozšíření rodin produktů je snadná
- Snadná práce s továrnou

Nevýhody

- Složitost kódu - hodně tříd a rozhraní
- Náročné přidání nových typů produktů
- Omezená rozširitelnost
- Není ideální pro jednoduché systémy

Kdy použít

- Chceme zajistit konzistenci mezi produkty
- Potřebujeme oddělit kód pro vytváření produktů od jejich použití
- Plánujeme přidávat nové rodiny produktů
- Systém musí být rozšiřitelný a konfigurovatelný
- Chceme usnadnit testování

Kdy nepoužít

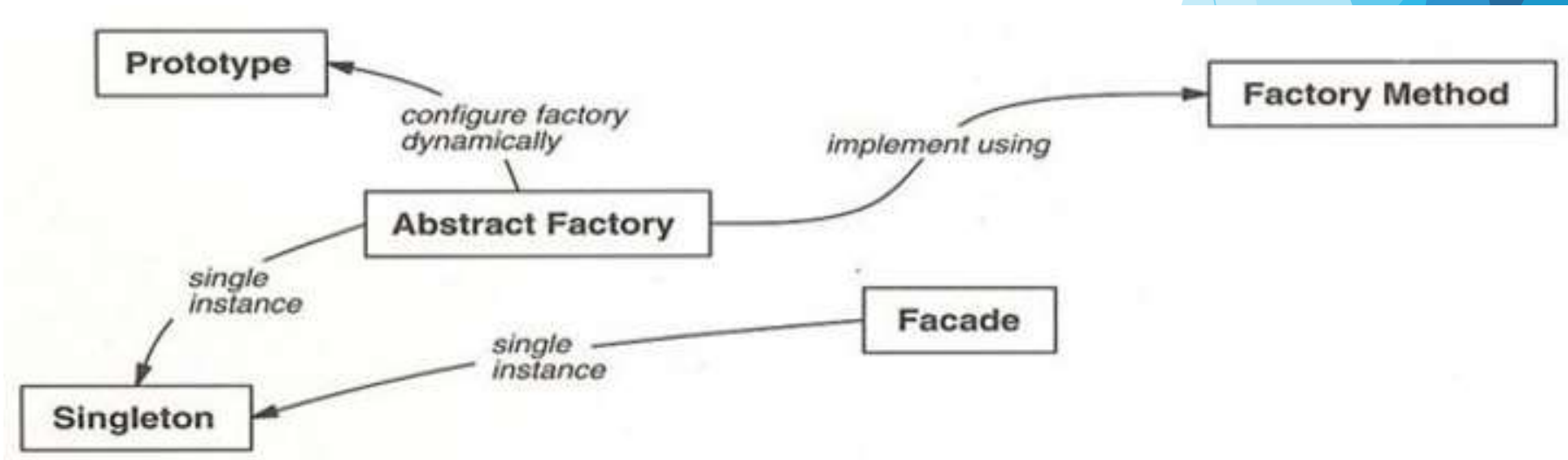
- Rodiny produktů se pravděpodobně nezmění
- Nezabývá se více rodinami objektů
- Není potřeba rozsáhlá konfigurovatelnost

Použití

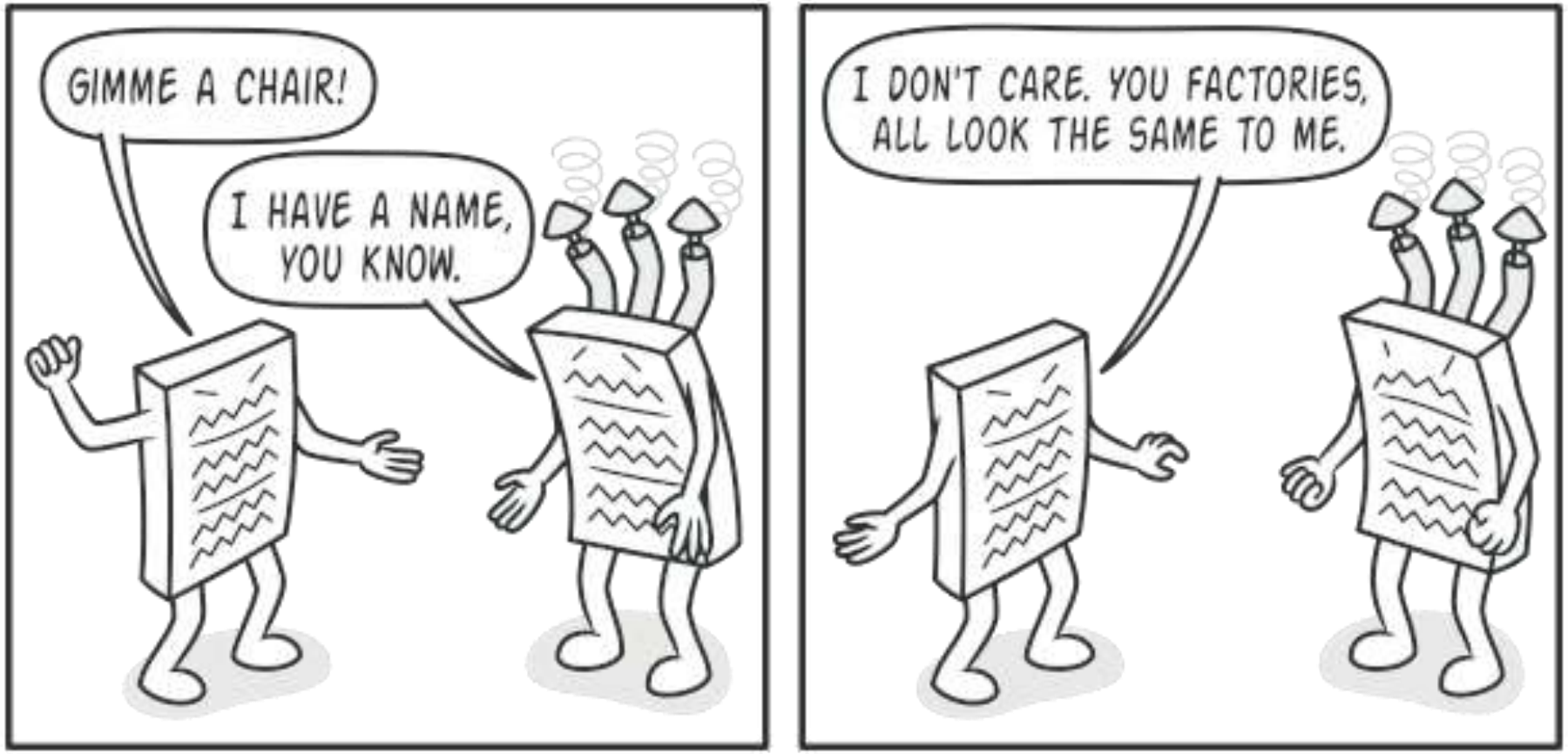
- Vývoj multiplatformních aplikací
  - UI Komponenty: rozhraní pro vytváření oken, tlačítek, textových polí ...
  - Databázová připojení: rozhraní pro vytváření připojení k databázi
- Hry a softwarové frameworky
  - Herní Assety: rozhraní pro vytváření herních assetů (postavy, vozidla, budovy ...)
  - Frameworky pro testování
- Softwarová konfigurace a personalizace
- Cloudové a distribuované systémy
- Pluginové systémy a rozšiřitelnost

Související návrhové vzory

- Factory Method:
  - méně komplikovaný
  - může být vyvinut směrem k Abstract Factory
- Builder:
  - Builder vytváří složité objekty krok za krokem
  - Možnost použití spolu s Bridge
- Facade:
  - Abstract Factory jako alternativa k Facade, ne vždy
- Singleton:
  - jediná instance třídy



Dotazy?


GIMME A CHAIR!
I HAVE A NAME, YOU KNOW.
I DON'T CARE. YOU FACTORIES, ALL LOOK THE SAME TO ME.

Zdroje

- The Gang of Four (GoF)
- Minuloroční verze prezentácie
- Refactoring Guru https://refactoring.guru/design-patterns/abstract-factory
- Medium https://medium.com/@aainajain/abstract-factory-pattern-501273665ec1
- Wikipedia https://en.wikipedia.org/wiki/Abstract_factory_pattern
- Codeburst https://codeburst.io/design-patterns-learning-abstract-factory-method-through-real-life-examples-9d0cc99ef0e8
- Geeksforgeeks https://www.geeksforgeeks.org/abstract-factory-pattern/