

Objektové modelování, AD7B36OMO

Návrhové vzory

Tvorba objektů

TOMÁŠ MAYER

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

<https://edux.feld.cvut.cz/courses/AD7B36OMO>

Přednáška je vytvořena na základě původních přednášek autorů: Karel Richta;Pavel Strnad, 2014

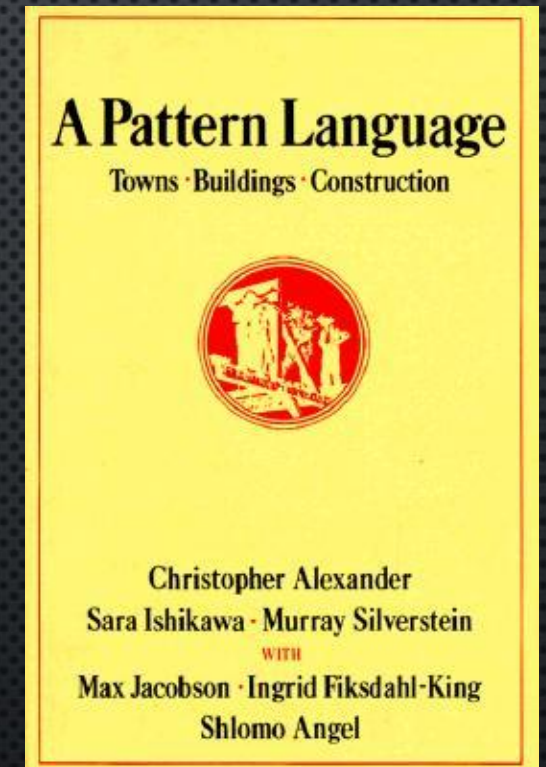
MOTIVACE

CÍLEM OBJEKTIVÉHO NÁVRHU JE STRUKTURU APLIKACE NAVRHNOUT TAK, ABY SPLŇOVALA NÁSLEDUJÍCÍ KRITÉRIA:

- SNADNÁ ROZŠÍŘITELNOST,
- ÚČELNOST,
- TESTOVATELNOST,
- DOKUMENTOVATELNOST.

CO TO JE NÁVRHOVÝ VZOR?

- CHRISTOPHER ALEXANDER: "KAŽDÝ VZOR POPISUJE ČASTO SE VYSKYTUJÍCÍ PROBLÉM A POTÉ POPISUJE JÁDRO ŘEŠENÍ TOHOTO PROBLÉMU TAK, ABY BYLO MOŽNO TOTO ŘEŠENÍ OPAKOVANĚ VYUŽÍVAT, BEZ TOHO, ŽE BYCHOM STEJNOU VĚC DĚLALI DVAKRÁT".
- PŘESTOŽE SE TATO DEFINICE TÝKÁ NÁVRHU BUDOV, TOTÉŽ PLATÍ PRO NÁVRHOVÉ VZORY PŘI TVORBĚ PROGRAMŮ. ŘEŠENÍ JE ZDE VYJÁDŘENO POMOCÍ POTŘEBNÝCH OBJEKTŮ A ZPŮSOBU JEJICH KOMUNIKACE.

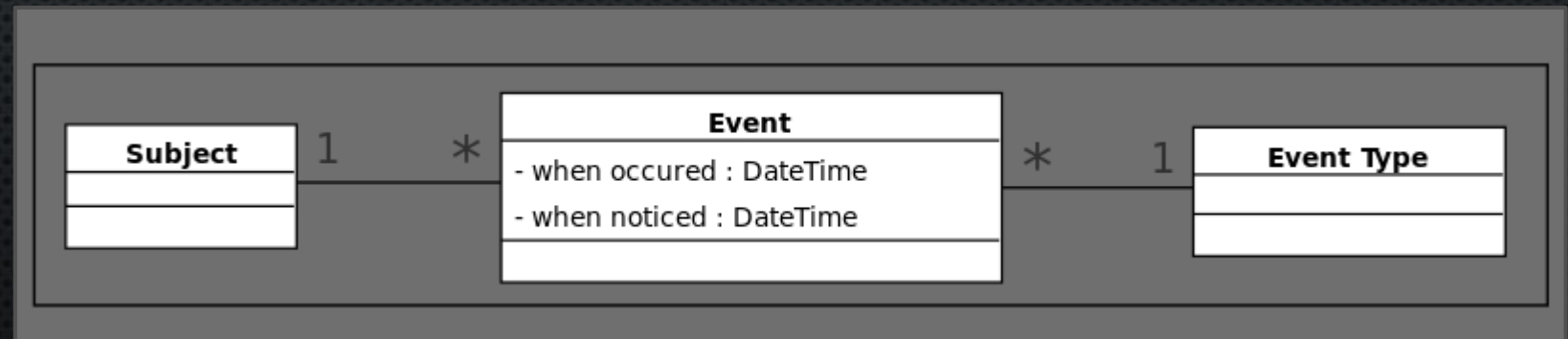


TYPY VZORŮ V SOFTWAREM INŽENÝRSTVÍ

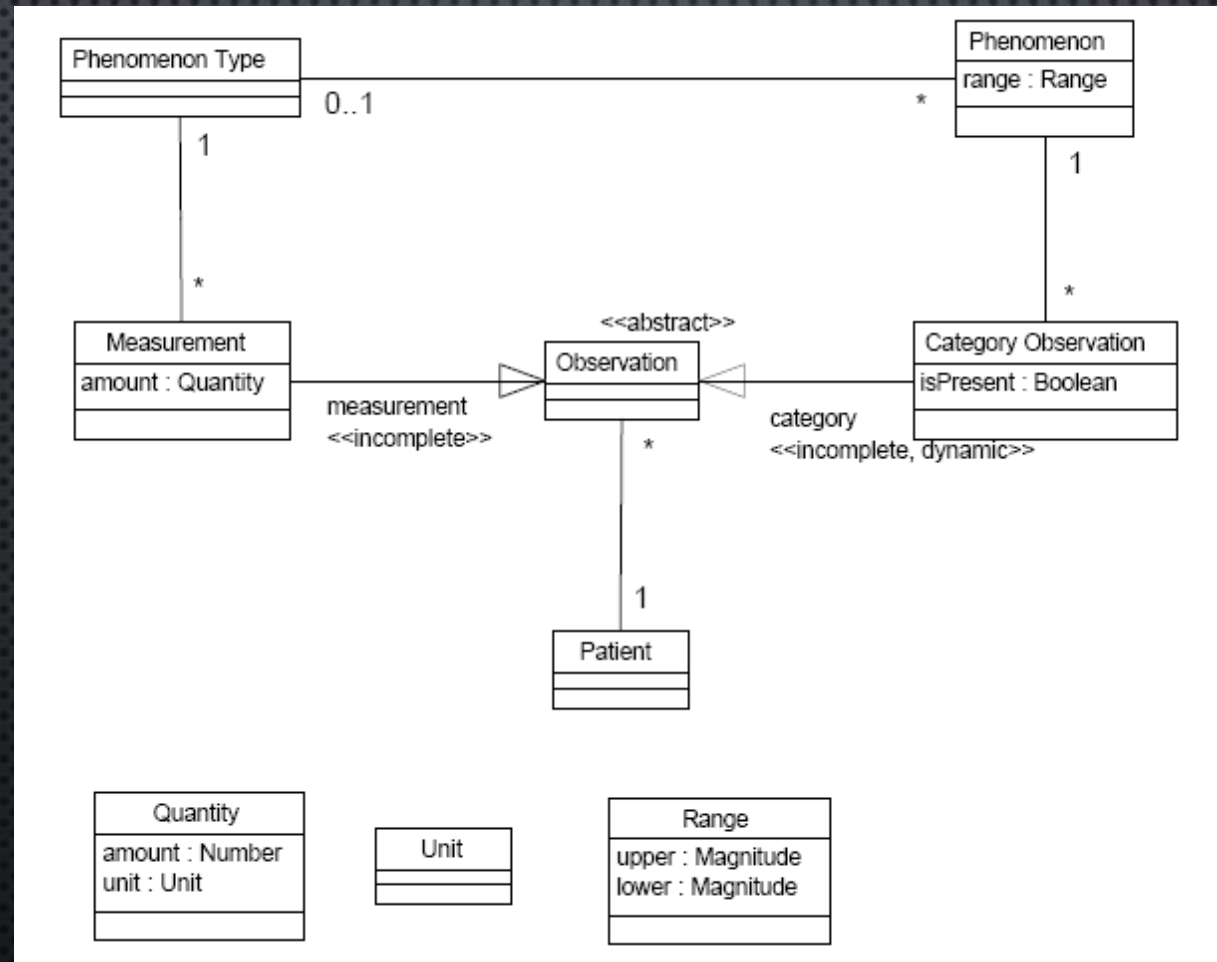
- ANALYTICKÉ – VZORY PRO ŘEŠENÍ ANALÝZY A POPISU ZADÁNÍ
- ARCHITEKTONICKÉ – VZORY PRO URČENÍ VHODNÉ ARCHITEKTURY ŘEŠENÍ
- INTEGRAČNÍ – VZORY PRO ŘEŠENÍ INTEGRACE
- NÁVRHOVÉ – VZORY PRO IMPLEMENTACI SOUČÁSTÍ ŘEŠENÍ
- KONVENCE KÓDU – KONVENCE, VZORY A STANDARDY JAKO PSÁT ZDROJOVÝ KÓD

ANALYTICKÉ VZORY (SOFTWARE ANALYSIS PATTERNS)

- CONCEPTUÁLNÍ MODEL
- ZACHYCUJE ABSTRAKCI SITUACE, KTERÁ BÝVÁ ČASTO MODELOVÁNA
- REPREZENTOVÁN JAKO SKUPINA GENERICKÝCH OBJEKTŮ, VZTAHY A INTERAKCE MEZI NIIMI
- DOMÉNOVĚ NEUTRÁLNÍ
- POPISUJE KONCEPTUÁLNÍ STRUKTURY NA BUSINESS ÚROVNI, NEPOPISUJÍ SOFTWAREOVÉ IMPLEMENTACE



ANALYTICKÝ VZOR: MĚŘENÍ A POZOROVÁNÍ



ARCHITEKTONICKÉ VZORY (SOFTWARE ARCHITECTURE PATTERNS)

- OBECNÉ OPAKOVANĚ POUŽÍVANÉ ŘEŠENÍ V SOFTWARE ARCHITEKTUŘE
- ARCHITEKTONICKÉ VZORY REAGUJÍ NA NUTNOST ŘEŠENÍ V SOFTWARE INŽENÝRSTVÍ
 - LIMITOVANÝ VÝKON HARDWARU
 - VYSOKÁ DOSTUPNOST
 - MINIMALIZACE OBCHODNÍCH RIZIK
 - ROZŠÍŘITELNOST ŘEŠENÍ
- VZOR PRO KONSTRUKCI, NAPŘ. SNAHA O KONSTRUKCI DOSTATEČNĚ PRUŽNÉ APLIKACE.

ARCHITEKTONICKÉ VZORY

- LAYERED PATTERN
- CLIENT – SERVER PATTERN
- MASTER – SLAVE PATTERN
- PIPE – FILTER PATTERN
- BROKER PATTERN
- PEER-TO-PEER PATTERN
- EVENT-DRIVEN PATTERN
- MICROKERNEL PATTERN
- MICROSERVICES PATTERN
- MODEL VIEW CONTROLLER PATTERN

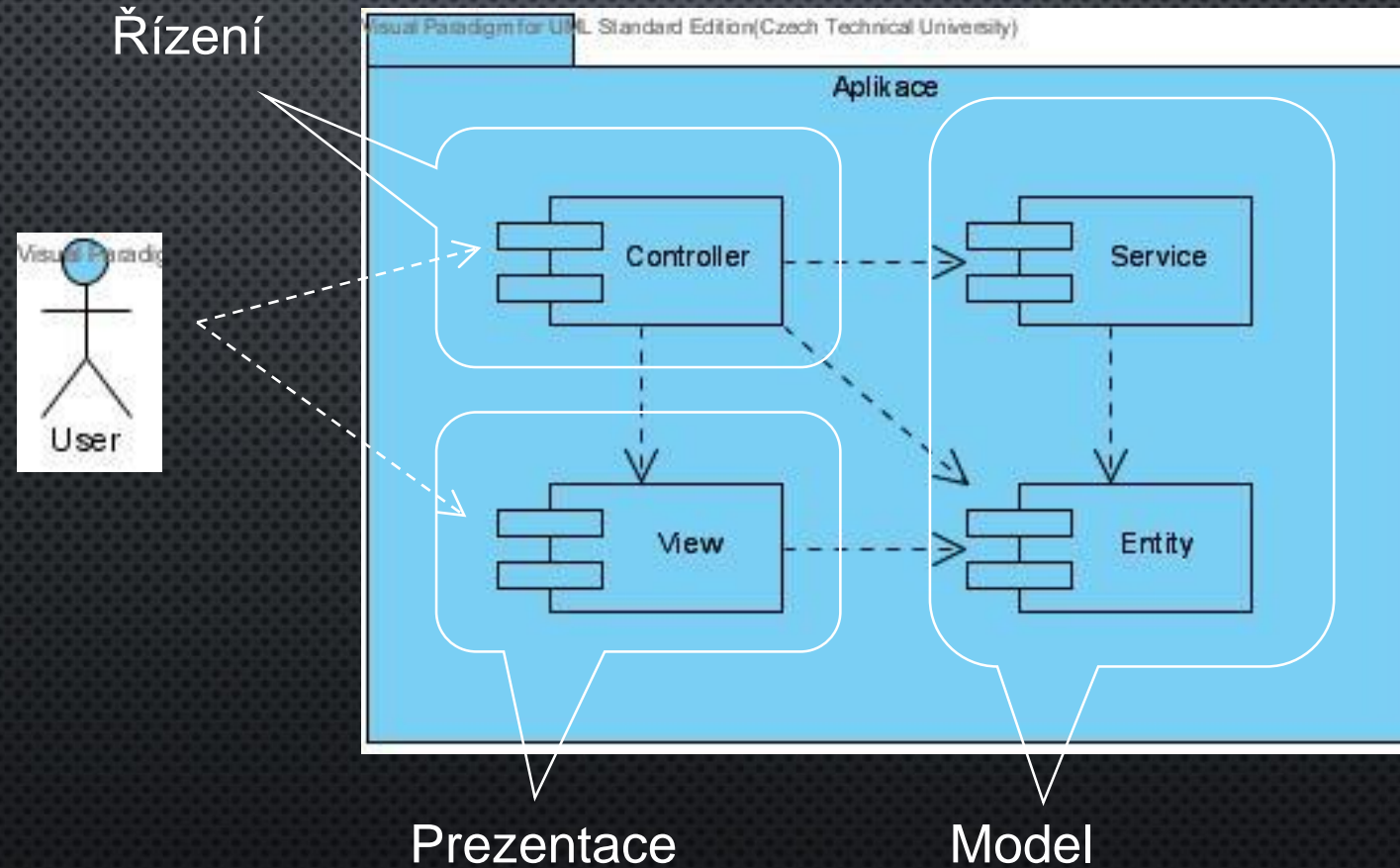
ARCHITEKTORNICKÉ VZORY – VZOR VRSTEV (LAYERED ARCHITECTURAL PATTERN)

- SYSTÉM ČLENĚNÝ DO SKUPIN, VRSTEV, KTERÉ JSOU ŘAZENY POD SEBE
- KAŽDÁ VRSTVA POSKYTUJE SLUŽBY VRSTVĚ, KTERÁ SE NACHÁZÍ NAD NÍ
- ČASTÉ ČLENĚNÍ NA NÁSLEDUJÍCÍ VRSTVY:
 - PREZENTAČNÍ VRSTVA
 - SERVISNÍ VRSTVA
 - VRSTVA BUSINESS LOGIKY
 - VRSTVA PŘÍSTUPU K DATŮM

ARCHITEKTONICKÉ VZORY PŘÍKLAD MVC

- VYPLATÍ SE ODDĚLIT SPRÁVU DAT OBHOSPODAŘOVANÝCH DANOU APLIKACÍ, JEJICH PREZENTACI A VLASTNÍ ŘÍZENÍ APLIKACE – MODEL VIEW CONTROLLER
- TOTO ROZDĚLENÍ PAK VYJÁDŘIT PŘÍMO ARCHITEKTUROU APLIKACE
- ZÁKLADNÍ KOMPONENTY APLIKACE:
- MODEL – PŘEDSTAVUJE DATA A SERVISNÍ VRSTVU PŘÍSTUPU K DATŮM
- VIEW – ZAJIŠŤUJE UI A PREZENTACI DAT
- KONTROLER – KOORDINUJE VIEW A MODEL

ARCHITEKTONICKÝ VZOR MVC



NÁVRHOVÉ VZORY

- VZOR PRO ŘEŠENÍ V SOFTWAREM NÁVRHU
- SNAHA O PODCHYCENÍ TYPICKÝCH PROBLÉMŮ, KTERÉ JE NUTNO ZOHLEDNIT PŘI NÁVRHU A NÁVOD NA JEJICH VHODNÉ ŘEŠENÍ
- KATALOG TAKOVÝCH VZORŮ, KDE MŮŽE NÁVRHÁŘ HLEDAT

“GANG OF FOUR” NÁVRHOVÉ VZORY

- ERICH GAMMA, RICHARD HELM, RALPH JOHNSON A JOHN VLISSIDES
- KATALOG VZORŮ
 - SOUHRN VZORŮ V KATALOGU
 - JEDNOTNÁ STRUKTURA POPISU VZORU
 - JMÉNO VZORU – ZAŽITÉ POJMENOVÁNÍ, KTERÝM SE ODKAZUJEME PŘI NÁVRHU
 - PROBLÉM – POPIS TYPICKÉ SITUACE, KDE JE UŽITEČNÉ POUŽITÍ VZORU
 - ŘEŠENÍ – POPIS POUŽITÍ NÁVRHOVÉHO VZORU
 - DŮSLEDKY – POPIS DOPADŮ PŘI POUŽITÍ VZORU, VÝHODY, NEVÝHODY
- CREATIONAL – VZORY PRO VYTVÁŘENÍ INSTANCÍ
- STRUCTURAL – VZORY PRO ORGANIZOVÁNÍ OBJEKTŮ DO STRUKTUR
- BEHAVIORAL – VZORY PRO PROVÁDĚNÍ PROGRAMOVÉ LOGIKY

VZORY PRO VYTVÁŘENÍ INSTANCÍ (CREATIONAL PATTERNS)

- **FACTORY METHOD**

UMOŽŇUJE JEDNOTNĚ VYTVÁŘET INSTANCE Z VÍCERO ODVOZENÝCH TŘÍD.

- **ABSTRACT FACTORY**

UMOŽŇUJE JEDNOTNĚ VYTVÁŘET SKUPINY SPOLUSOUVISEJÍCÍCH OBJEKTŮ

- **BUILDER**

ODDĚLUJE KONSTRUKCI OBJEKTU OD JEHO REPREZENTACE

- **PROTOTYPE**

PLNĚ INICIALIZOVANÁ INSTANCE, KTERÁ SLOUŽÍ JAKO VZOR PRO ZKOPÍROVÁNÍ

- **SINGLETON**

TŘÍDA, KTERÁ MÁ MÍT V PROGRAMU POUZE JEDINOU INSTANCI

VZOR FACTORY METHOD

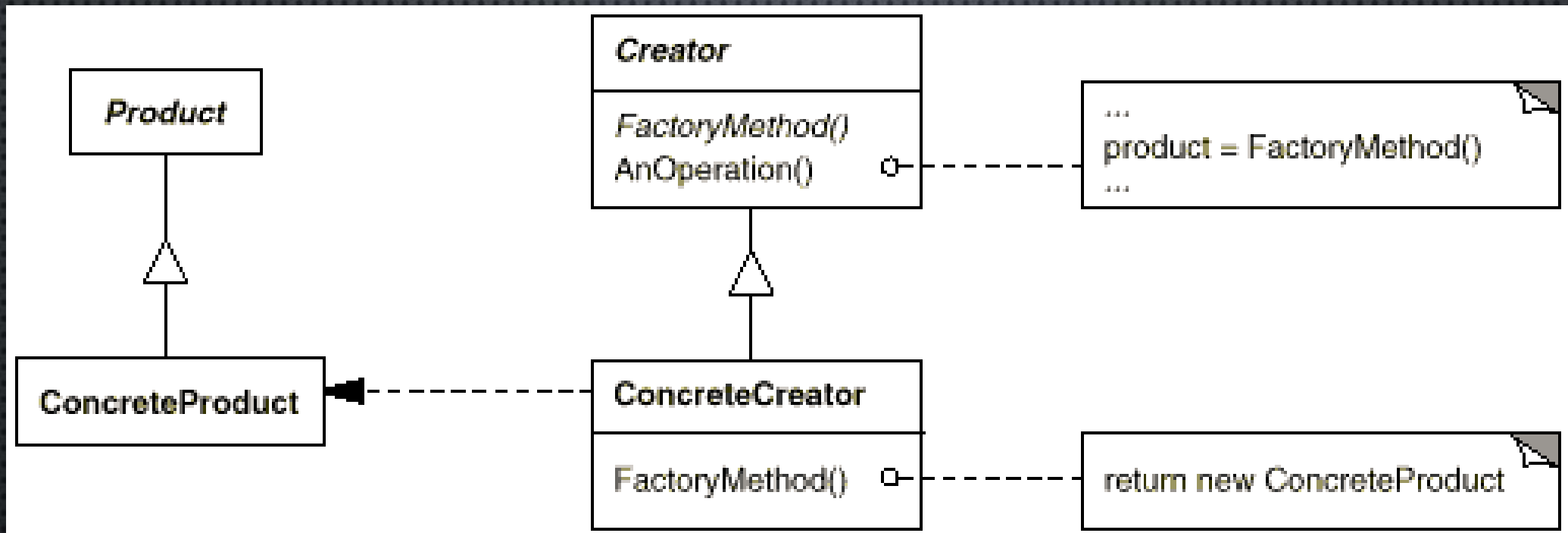
- MOTIVACE

- V APLIKACI POTŘEBUJI SKRÝT PODROBNOSTI VYTVÁŘENÍ KONKRÉTNÍHO OBJEKTU.
- MÁM SADU TŘÍD, KTERÉ MAJÍ SPOLEČNÉ METODY, ALE DETAILS VYTVOŘENÍ INSTANCE ZNÁ AŽ KONKRÉTNÍ APLIKACE.

- PŘÍKLAD

- LOGISTICKÁ APLIKACE PRACUJE S RŮZNÝMI DRUHY DOPRAVY (NÁKLADNÍ AUTO, VLAK, LOŽ...), TY SE NAVÍC V BUDOUCNU MOHOU MĚNIT.
- POTŘEBUJI ALE MÍT JEDNOTNÝ ZPŮSOB PRO VYTVOŘENÍ – METODU `CREATETRANSPORT()` , KTERÝ VYTVOŘÍ KONKRÉTNÍ OBJEKT ODPOVÍDAJÍCÍ DANÉ LOGISTICE – AUTOMOBILOVÁ, LODNÍ, ŽELEZNIČNÍ DOPRAVA

CLASS DIAGRAM



Obrázek převzat z [GoF]

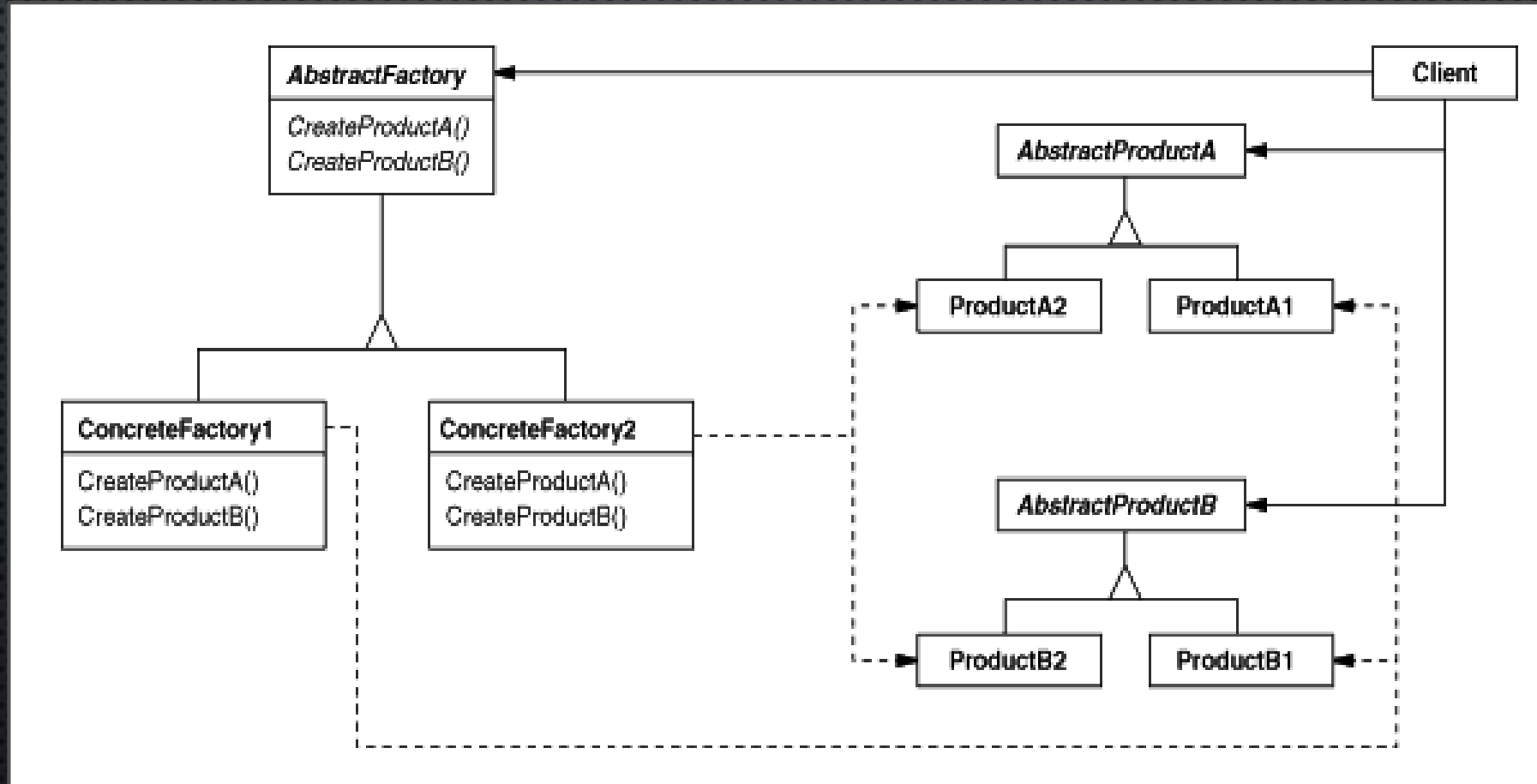
FACTORY METHOD

- KDY POUŽÍT
 - NEVÍME PŘESNÉ TYPY A VZTAHY MEZI OBJEKTY, KTERÉ BUDE KÓD POUŽÍVAT
 - CHCEME UŽIVATELŮM KNIHOVNY/FRAMEWORKU POSKYTNOUT ZPŮSOB JAK ROZŠIŘOVAT JEHO INTERNÍ KOMPONENTY
 - CHCEME ŠETŘIT SYSTÉMOVÉ ZDROJE A ZNOVUPOUŽÍT JIŽ EXSTUJÍCÍ OBJEKTY (NAPŘÍKLAD DATABÁZOVÉ PŘIPOJENÍ)
- VÝHODY
 - ODSTRAŇUJE TĚSNOU VAZBU MEZI IMPLEMENTACÍ A KÓDEM, KTERÝ JI VYUŽÍVÁ
 - ZJEDNODUŠUJE KÓD PŘESUNUTÍM VYTVÁŘENÍ OBJEKTŮ NA JEDNO MÍSTO
 - ZJEDNODUŠUJE PŘIDÁVÁNÍ NOVÝCH TŘÍD DO PROGRAMU
- NEVÝHODY
 - VYŽADUJE NAVÍC ZDĚDĚNÉ TŘÍDY

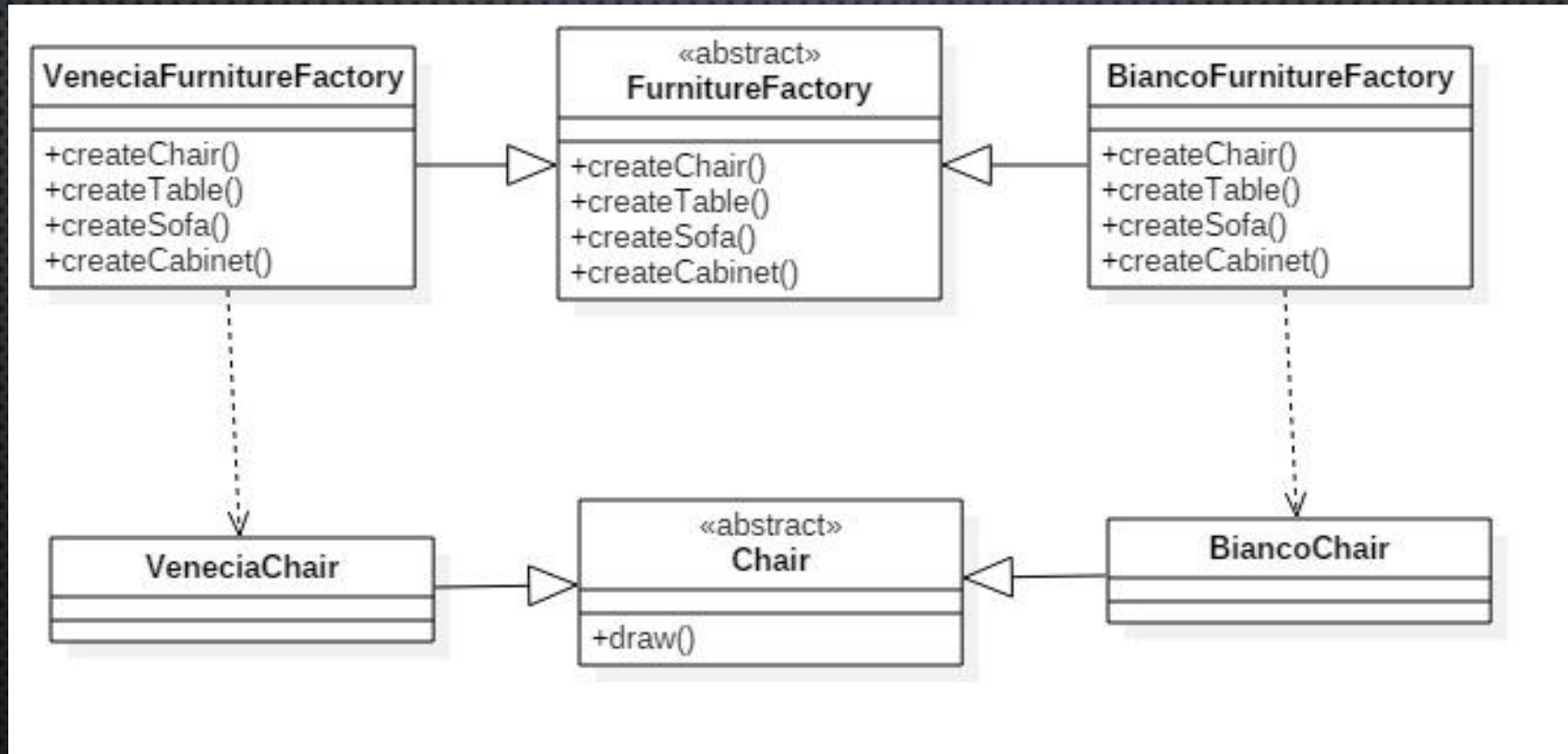
VZOR: ABSTRACT FACTORY (KIT)

- DEKLARACE ZÁMĚRU:
 - INTERFACE PRO VYTVÁŘENÍ KOLEKCÍ OBJEKTŮ BEZ PODROBNÉ SPECIFIKACE JEJICH STRUKTURY
- MOTIVAČNÍ PŘÍKLAD:
 - SADA SKUPIN SOUVISEJÍCÍCH OBJEKTŮ – PŘÍKLAD NÁBYTKOVÉ SOUPRAVY SESTÁVAJÍCÍ ZE STOLU, ŽIDLÍ, POHOVKY, SKŘÍNĚK
 - KAŽDÁ SADA JE JINÁ, NECHCEME KOMBINOVAT NÁBYTEK Z RÚZNÝCH SAD, ALE JEDNOTLIVÉ PRVKY SE STÁLE OPAKUJÍ
 - CHCEME DO BUDOUCNA MÍT MOŽNOST JEDNODUŠE PŘIDAT NOVÉ SADY

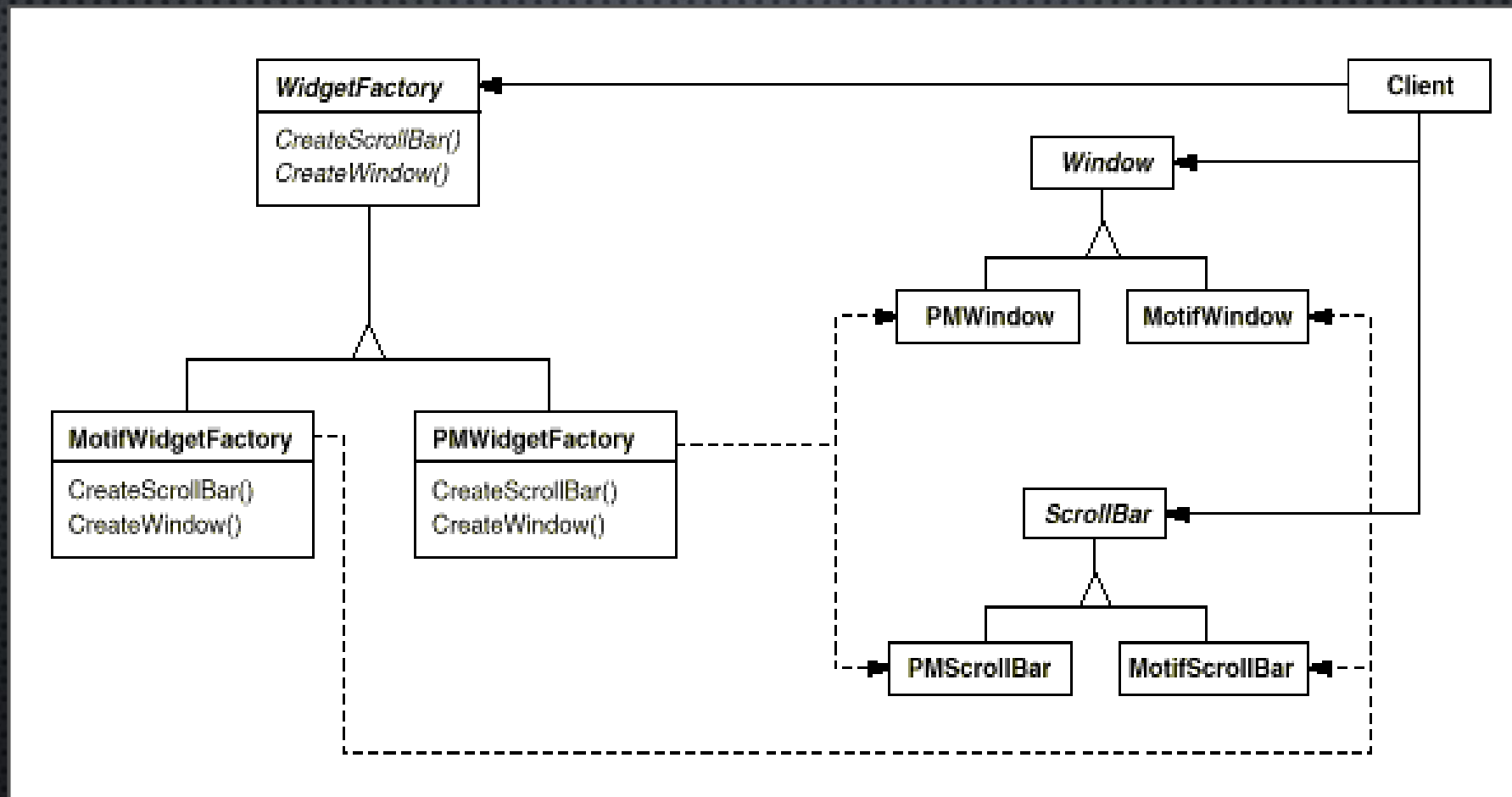
ABSTRACT FACTORY (STAVEBNICE)



ABSTRACT FACTORY - PŘÍKLAD



ABSTRACT FACTORY - PŘÍKLAD POUŽITÍ



Obrázek převzat z [GoF]

ABSTRACT FACTORY

- KDY POUŽÍT
 - KÓD MUSÍ PRACOVAT S ROZDÍLNÝMI VARIANTAMI PRODUKTŮ Z JEDNÉ SADY, ALE NECHCEME SE VÁZAT KONKRÉTNÍ TŘÍDY (NEBO TŘÍDY JEŠTĚ VŮBEC NEJSOU ZNÁMY)
 - TŘÍDA MÁ NĚKOLIK FACTORY METOD, KTERÉ NESOUVISÍ S JEJÍ HLAVNÍ ZODPOVĚDNOSTÍ
- VÝHODY
 - UMOŽNÍ VYTVÁŘET NOVÉ SADY OBJEKTŮ, KDE MÁME ZARUČENU JEJICH KOMPATIBILITU
 - ODSTRAŇUJE TĚSNOU VAZBU MEZI IMPLEMENTACÍ PRODUKTU A KÓDEM, KTERÝ JI POUŽÍVÁ
 - URČUJE ZODPOVĚDNOST TAK, ABY TŘÍDA MĚLA POKUD MOŽNO JEDNU ZODPOVĚDNOST
- NEVÝHODY
 - NEPATRNĚ ZVYŠUJE POČET TŘÍD V ŘEŠENÍ
- VZTAH K JINÝM VZORŮM
 - FACTORY METHOD, PROTOTYPE, SINGLETON

VZOR: BUILDER

- DEKLARACE ZÁMĚRU:
 - ODDĚLENÍ KONSTRUKCE SLOŽITÉHO OBJEKTU OD JEHO REPREZENTACE
 - POKUD MÁ BÝT ALGORITMUS PRO VYTVÁŘENÍ SLOŽITÉHO OBJEKTU NEZÁVISLÝ NA VYTVÁŘENÍ ČÁSTÍ
 - POKUD LZE OBJEKT REPREZENTOVAT RŮZNÝMI ZPŮSOBY

PŘÍKLAD: STAVBA DOMU

BUDOVA

Zdroj: <http://voho.cz/wiki/informatika/oop/navrhovy-vzor/builder/>

- ZAČNEME JEDNODUCHOU TŘÍDOU PŘEDSTAVUJÍCÍ BUDOVU. BUDE MÍT TŘI ATRIBUTY: PODLAHU (FLOOR), ZDI (WALLS) A STŘECHU (ROOF). BUDOVA BUDE NAŠÍM PRODUKTEM. RŮZNÉ BUDOVY SE OD SEBE BUDOU LIŠIT PODLAHOU, ZDMI A STŘECHOU. DŮLEŽITÉ JE, ŽE KLIENT O POSTAVENÍ BUDOVY POŽÁDÁ ŘÍDÍCÍ OBJEKT (STAVBYVEDOUČÍHO) A O PROCES JEJÍHO VYSTAVĚNÍ SE NESTARÁ.

```
public class Building {  
    private String floor;  
    private String walls;  
    private String roof;  
    public void setFloor(final String pFloor) { this.floor = pFloor; }  
    public void setWalls(final String pWall) { this.walls = pWall; }  
    public void setRoof(final String pRoof) { this.roof = pRoof; }  
    public void print() { System.out.println(String.format("%s, %s, %s",  
        this.floor, this.walls, this.roof));  
    }  
}
```


PŘÍKLAD: STAVBA DOMU (POKR.)

ROZHRANÍ STAVITELE

- STAVITELÉ V NAŠEM PŘÍKLADU BUDOU MÍT NĚKOLIK SPOLEČNÝCH METOD, A TO NA ZAHÁJENÍ NOVÉ STAVBY, POLOŽENÍ PODLAHY, STAVBU ZDÍ, STAVBU STŘECHY A ZÍSKÁNÍ VÝSLEDNÉ BUDOVY. PROTO VYTVOŘÍME ROZHRANÍ, KTERÉ BUDOU VŠICHNI STAVITELÉ IMPLEMENTOVAT.

```
public interface Builder {  
    void startNew();    /* Zahájí novou stavbu. */  
    void buildFloor(); /* Položí podlahu. */  
    void buildWalls(); /* Postaví zdi. */  
    void buildRoof();  /* Postaví střechu. */  
    Building getResult(); /* Vrábí výslednou budovu. */  
}
```


PŘÍKLAD: STAVBA DOMU (POKR.)

STAVITELÉ

- PRO NÁŠ PŘÍKLAD VYTVOŘÍME DVĚ RŮZNÉ IMPLEMENTACE STAVITELŮ: JEDEN BUDE STAVĚT BUDOVOY LEVNÉ, DRUHÝ LUXUSNÍ. KLIENT SI PAK JEDNOHO Z NICH VYBERE A TUTO VOLBU SDĚLÍ STAVBYVEDOUCÍMU. OBA STAVITELÉ BUDOU V NAŠEM PŘÍPADĚ VÝKONNÝMI OBJEKTY, PROTOŽE ZAJIŠŤUJÍ KONKRÉTNÍ ÚKONY PŘI STAVBĚ.

```
public class CheapBuilder implements Builder {
    private Building result;
    public void startNew() { this.result = new Building(); }
    public void buildFloor() { this.result.setFloor("laminated floor"); }
    public void buildWalls() { this.result.setWalls("panel walls"); }
    public void buildRoof() { this.result.setRoof("wooden roof"); }
    public Building getResult() { return this.result; }
}
```


PŘÍKLAD: STAVBA DOMU (POKR.)

STAVITELÉ - POKRAČOVÁNÍ

```
public class LuxuryBuilder implements Builder {  
    private Building result;  
    public void startNew() { this.result = new Building(); }  
    public void buildFloor() { this.result.setFloor("wooden floor"); }  
    public void buildWalls() { this.result.setWalls("brick walls"); }  
    public void buildRoof() { this.result.setRoof("shindel roof"); }  
    public Building getResult() { return this.result; }  
}
```


PŘÍKLAD: STAVBA DOMU (POKR.)

STAVBYVEDOUCÍ

- V ROLI ŘÍDÍCÍHO OBJEKTU ZDE BUDE VYSTUPOVAT STAVBYVEDOUCÍ. TEN DOSTANE K DISPOZICI STAVITELE (BUILDER), S JEHOŽ POMOCÍ BUDOVU POSTUPNĚ VYSTAVÍ. ZAČNE PODLAHOU, POKRAČUJE STĚNAMI A NAKONEC POSTAVÍ STŘECHU. PO DOKONČENÍ VŠECH OPERACÍ VRÁTÍ KLIENTOVI VÝSLEDNOU BUDOVU.

```
public class Director {  
    /* Postaví budovu s použitím zadaného stavitele (parametr builder),  
       vrátí výslednou budovu */  
    public Building build(final Builder builder) {  
        builder.startNew(); // zahájit stavbu nové budovy  
        builder.buildFloor(); // položit podlahu  
        builder.buildWalls(); // postavit zdi  
        builder.buildRoof(); // postavit střechu  
        return builder.getResult(); // vrátit výsledek  
    }  
}
```


PŘÍKLAD: STAVBA DOMU (POKR.)

TESTER

- VZOR POUŽITÍ

```
final Director director = new Director();
```

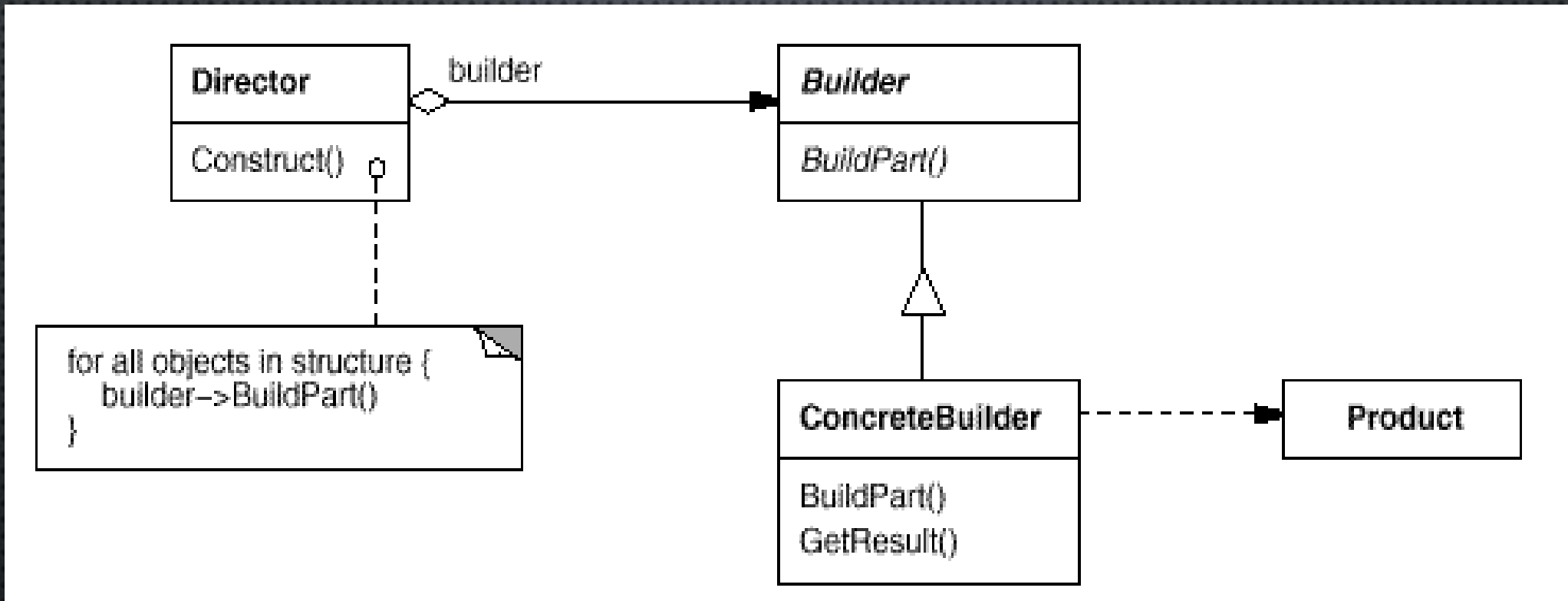
```
// levná budova
```

```
final Building cheapBuilding = director.build(new CheapBuilder());  
cheapBuilding.print();
```

```
// luxusní budova
```

```
final Building luxuryBuilding = director.build(new LuxuryBuilder());  
luxuryBuilding.print();
```

STRUKTURA VZORU BUILDER

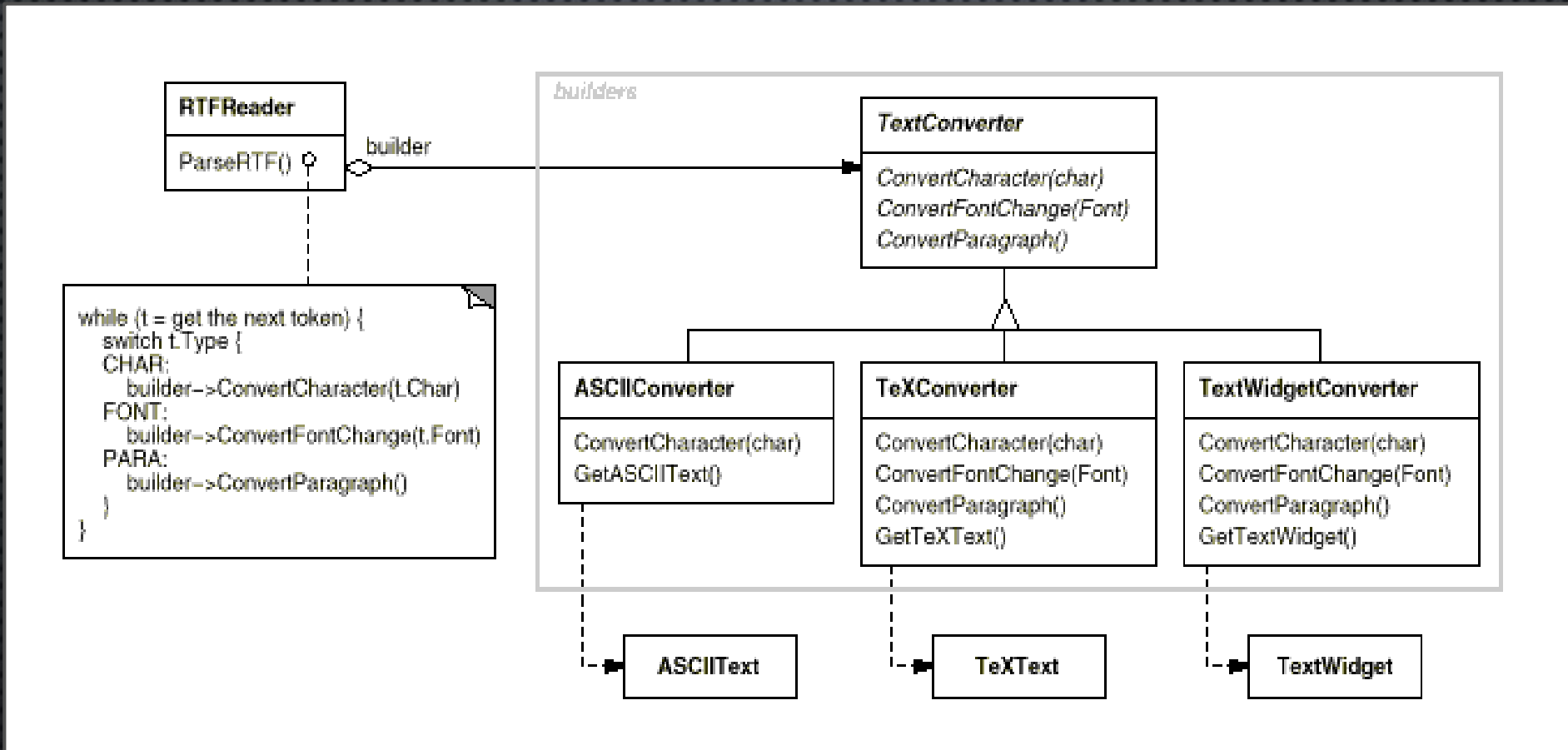


Obrázek převzat z [GoF]

VZOR BUILDER – PŘÍKLAD EXPORT Z RTF EDITORU

- EDITOR DOKUMENTU V RTF
- MĚL BY UMĚT EXPORTOVAT DO RŮZNÝCH REPREZENTACÍ TEXTU
 - ASCII
 - TEX
 - TEXTWIDGET
- POSTUPNÝM ČTENÍM RTF DOKUMENTU DÁVÁ PŘÍKAZY KONVERTERU DO VÝSTUPNÍHO FORMÁTU – PŘEČTE ZNAK – POŠLE ZNAK DO KONVERTERU, PŘEČTE ZAČÁTEK NOVÉHO Odstavce – ZADÁ PŘÍKAZ DO KONVERTERU NA ZALOŽENÍ NOVÉHO Odstavce..

PŘÍKLAD POUŽITÍ VZORU BUILDER



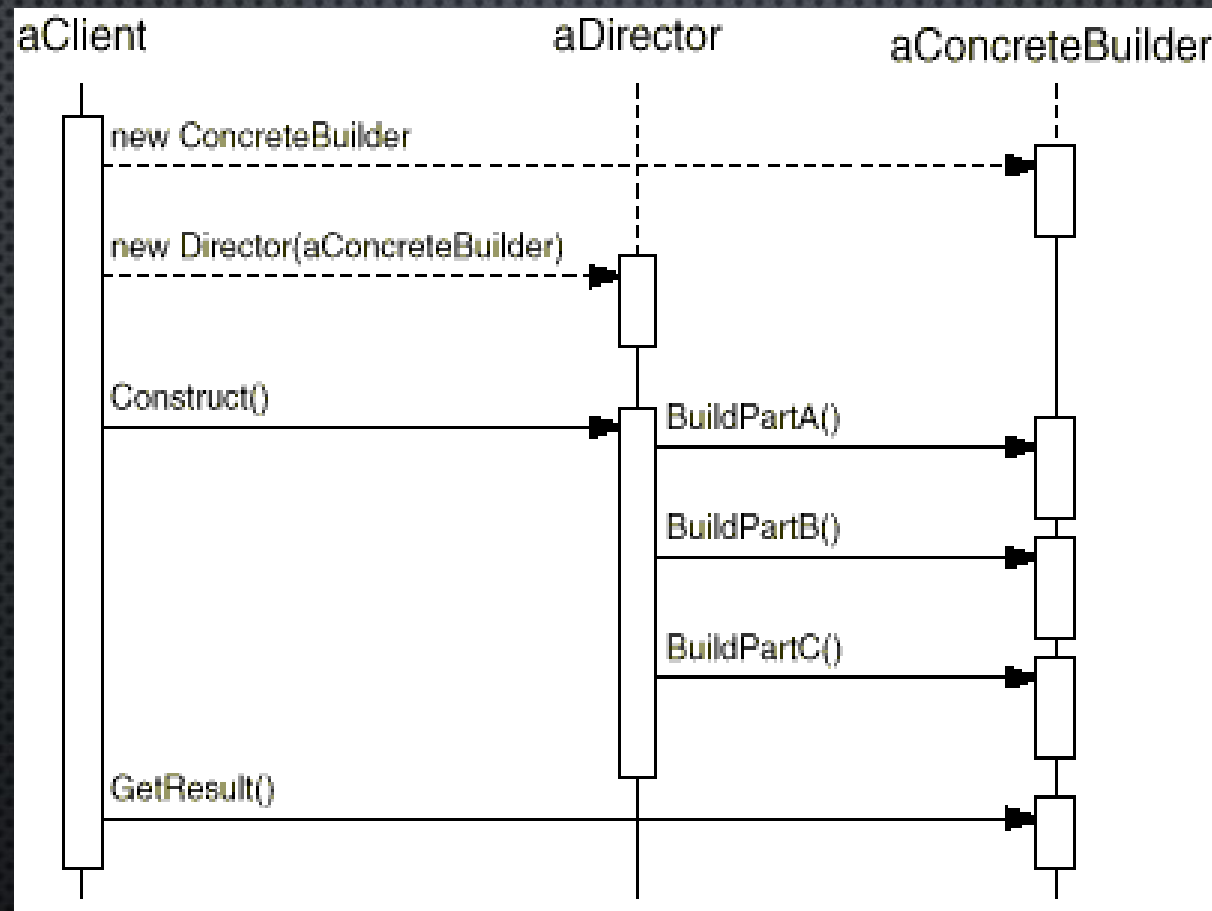
PARTICIPANTI

- BUILDER (TEXTCONVERTER)
 - SPECIFIKUJE ABSTRAKTNÍ INTERFACE PRO VYTVÁŘENÍ ČÁSTÍ OBJEKTŮ TYPU PRODUCT.
- CONCRETEBUILDER (ASCIICONVERTER, TEXCONVERTER, TEXTWIDGETCONVERTER)
 - KONTRUUJE A SESTAVUJE ČÁSTI - IMPLEMENTUJE INTERFACE TŘÍDY BUILDER
 - UDRŽUJE SI VYTVOŘENOU REPREZENTACI
 - POSKYTUJE SLUŽBY PRO VYZVEDNUTÍ PRODUKTU (GETASCIITEXT, GETTEXTWIDGET, ...)

PARTICIPANTI (POKRAČ.)

- DIRECTOR (RTFREADER)
 - KONSTRUUJE OBJEKTY POMOCÍ SLUŽEB TŘÍDY BUILDER
- PRODUCT (ASCIITEXT, TEXTTEXT, TEXTWIDGET)
 - REPREZENTUJE SLOŽITÝ OBJEKT, KTERÝ JE KONSTRUOVÁN, TŘÍDA CONCRETEBUILDER VYTVÁŘÍ KONKRÉTNÍ VNITŘNÍ REPREZENTACI
 - ZAHRNUJE TŘÍDY PRO DEFINICI ČÁSTÍ A SLUŽBY PRO KONSTRUKCI CELKU

SPOUPRÁCE OBJEKTŮ PŘI POUŽITÍ BUILDER



Obrázek převzat z [GoF]

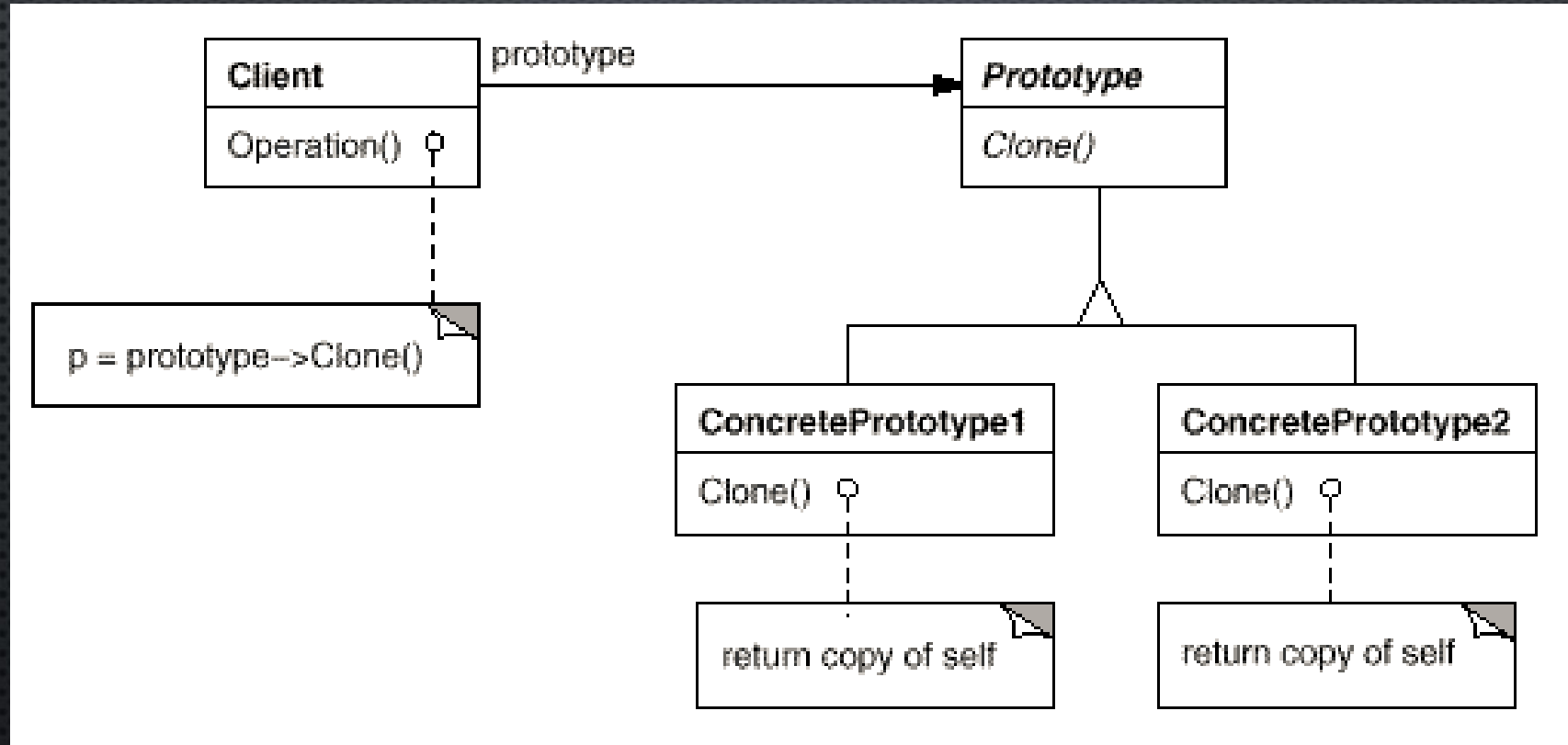
BUILDER

- KDY POUŽÍT
 - POKUD SE VYSKYTUJE KONSTRUKTOR S MNOHA VOLITENÝMI PARAMETRY
 - POKUD KÓD PRODUKUJE RŮZNÉ REPREZENTACE JEDNOHO PRODUKTU – STEJNÉ KROKY VEDOUcí K CÍLI, KTERÉ SE LIŠÍ V DETAILU
 - KDYŽ PRACUJEME SE KOMPOZITNÍMI STROMOVÝMI STRUKTURAMI
- VÝHODY
 - UMOŽŇUJE POSTAVIT PRODUKT KROK PO KROKU
 - UMOŽNÍ STEJNÉMU KÓDU POSTAVIT ROZDÍLNÉ PRODUKTY
 - IZOLUJE SLOŽITÝ KÓD PRO VYTVÁŘENÍ OD BUSINESS LOGIKY PRODUKTU
- NEVÝHODY
 - ZVYŠUJE ČLENITOST ŘEŠENÍ DALŠÍMI TŘÍDAMI
- VZTAH K OSTATNÍM VZORŮM - ABSTRACT FACTORY, COMPOSITE

VZOR PROTOTYPE (PROTOTYP)

- MOTIVACE
 - POTŘEBUJI VYROBIT KOPII OBJEKTU. OBJEKT JE ČLENITÝ, NAVÍC MÁ PRIVÁTNÍ PROMĚNNÉ, KE KTERÝM Z VNĚJŠÍHO KÓDU NEDOSTANU.
- ŘEŠENÍ
 - DELEGOVAT VYTVOŘENÍ KOPIE NA OBJECT SAMOTNÝ TAK, ABY DOKÁZAL VYROBIT KOPII SAMA SEBE.

VZOR PROTOTYPE - STRUKTURA



Obrázek převzat z [GoF]

PROTOTYPE

- KDY POUŽÍT
 - POKUD BY KÓD NEMĚL ZÁVISET NA KONTRÉTNÍCH TŘÍDÁCH OBJEKTŮ – NAPŘÍKLAD KDYŽ OBJEKTY NEJSOU ZNÁMY A KÓD PRACUJE POUZE S INTERFACE
 - CHCEME REDUKOVAT HIERARCHII TŘÍD PODOBNÝCH OBJEKTŮ JEN JINAK KONFIGUROVANÝCH
- VÝHODY
 - UMOŽNÍ KLONOVÁNÍ OBJEKTŮ BEZ VÁZÁNÍ SE NA JEJICH KONTRÉTNÍ TŘÍDY
 - UMOŽNÍ REDUKOVAT OPAKUJÍCÍ SE INICIALIZAČNÍ KÓD
 - UMOŽNÍ RYCHLEJŠÍ VÝROBU NOVÝCH INSTANCÍ
 - ALTERNATIVA K DĚDĚNÍ PRO VELKÉ OBJEKTY S MNOHA VOLBAMI KONFIGURACE
- NEVÝHODA
 - ČLENITÉ OBJEKTY S MNOHA REFERENCEMI NA JINÉ OBJEKTY MŮŽE BÝT OBTÍŽNÉ KLONOVAT

VZOR SINGLETON

- JE TŘEBA, ABY V APLIKACI EXISTOVALA POUZE JEDNA INSTANCE NĚJAKÉ TŘÍDY.
- JAK TO ZAJISTIT?

VZOR SINGLETON

```
public class PrintManager {
    private static PrintManager instance;

    private PrintManager () { /* ... */ }

    // jina jmena: getDefault(), getPrintManager,...
    public static synchronized PrintManager getInstance () {
        if (instance == null) { // "lazy" inicializace
            instance = new PrintManager ();
        }
        return instance;
    }

    // vlastni instancni metody...
}
```

VZOR SINGLETON

- VLASTNOSTI:
 - KONKRÉTNÍ TYP POSKYTNUTÉ INSTANCE LZE URČIT ZA BĚHU
 - POSKYTOVANOU INSTANCI LZE MĚNIT BEZ ZMĚN KLIENTSKÉHO KÓDU
 - SNADNÁ ZMĚNA LOGIKY TVORBY – NAPŘ. POVOLENÍ VÍCE INSTANCÍ
- PŘÍKLAD V J2SE: `JAVA.LANG.RUNTIME`
- SOUVISEJÍCÍ VZORY:
 - `NULL OBJECT/OBJEKT NULL`
 - `OBJECT POOL`

SINGLETON

- KDY POUŽÍT
 - POKUD KÓD MÁ VYUŽÍT JEDNÉ INSTANCE. NAPŘÍKLAD JEDEN DATABÁZOVÝ OBJEKT SDÍLENÝ OSTNÁMI OBJEKTY
 - POKUD JE POTŘEBA VÍCE ŘÍDIT PŘÍSTUP KE GLOBÁLNÍM PROMĚNNÝM
- VÝHODY
 - ZAJIŠŤUJE, ŽE TŘÍDA MÁ MAXIMÁLNĚ JEDNU INSTANCI
 - UMOŽNÍ ODLOŽENOU INICIALIZACI
- NEVÝHODY
 - MŮŽE SOUUISET SE ŠPATNÝM NÁVRHEM
 - POŽADUJE SPECIÁLNÍ OŠETŘENÍ VE VÍCEVLÁKNOVÉM ZPRACOVÁNÍ
 - POŽADUJE NEUSTÁLÉ MOCKOVÁNÍ V UNIT TESTECH

OBJECT POOL PATTERN

- OBJEKTY MOHOU DRŽENY V POOL A MOHOU BÝT ZNOVUPOUŽITY
- KDY POUŽÍT
 - V PŘÍPADĚ VELKÉHO MNOŽSTVÍ VYTVÁŘENÝCH OBJEKTŮ S KRÁTKOU ŽIVOTNOSTÍ
 - INICIALIZACE OBJEKTU JE ČASOVĚ NÁROČNÁ NEBO JE NÁROČNÁ NA ZDROJE
 - KDYŽ VHODNÉ ZAJISTIT PREDIKCI ČASU PRO ZÍSKÁNÍ OBJEKTU TAM, KDE NENÍ MOŽNÉ PREDIKOVAT ČAS POTŘEBNÝ PRO VYTVOŘENÍ NOVÉHO OBJEKTU
 - KDE JE POTŘEBA ŘÍDIT MAXIMÁLNÍ POČET AKTUÁLNĚ POUŽÍVANÝCH OBJEKTŮ

INVERSION OF CONTROL (IOC)

- NÁVRHOVÝ PRINCIP, VE KTERÉM KÓD PROGRAMU JE ŘÍZENÝ Z OBECNÉHO FRAMEWORKU
- JEDNÁ SE INVERZNÍ PŘÍSTUP OPROTI PŮVODNÍMU PŘÍSTUPU
- PŮVODNÍ KLASICKÝ PŘÍSTUP - PROGRAMOVÝ KÓD VOLÁ ZNOVUPOUŽITELNÉ KNIHOVNY STARAJÍCÍ SE O OBECNÉ ÚLOHY
- IOC – OBECNÝ FRAMEWORK ŘÍDÍ CHOD PROGRAMU A VOLÁ SPECIFICKY VYTVOŘENÝ KÓD PRO ŘEŠENÍ KONKRETNÍCH ÚLOH

DEPENDENCY INJECTION (VKLÁDÁNÍ ZÁVISLOSTÍ)

- **SPECIÁLNÍ PŘÍPAD IOC**
- **DEFINICE WIKIPEDIA: VKLÁDÁNÍ ZÁVISLOSTÍ** (ANGLICKY **DEPENDENCY INJECTION (DI)**) JE V OBJEKTOVĚ ORIENTOVANÉM PROGRAMOVÁNÍ TECHNIKA PRO VKLÁDÁNÍ ZÁVISLOSTÍ MEZI JEDNOTLIVÝMI KOMPONENTAMI PROGRAMU TAK, ABY JEDNA KOMPONENTA MOHLA POUŽÍVAT DRUHOU, ANIŽ BY NA NI MĚLA V DOBĚ SESTAVOVÁNÍ PROGRAMU REFERENCI. DEPENDENCY INJECTION LZE CHÁPAT JAKO NOVĚJŠÍ NÁZEV PRO INVERSION OF CONTROL (IOC), ALE V UŽŠÍM VÝZNAMU. JEDNÁ SE O KONKRÉTNÍ TECHNIKU IOC.
- MARTIN FOWLER, [HTTP://WWW.MARTINFOWLER.COM/ARTICLES/INJECTION.HTML](http://www.martinfowler.com/articles/injection.html)

PRINCIP DEPENDENCE INJECTION

- POKUD CHCE NÁŠ OBJEKT POUŽÍVAT SLUŽBY JINÉHO OBJEKTU BEZ POUŽITÍ DI, PAK ZODPOVÍDÁ ZA CELÝ JEHO ŽIVOTNÍ CYKLUS, TEDY INICIALIZACI APOD.
- PŘI APLIKACI DI JE ALE NÁŠ OBJEKT ODSTÍNĚN OD TÉTO SPRÁVY, TU ZAŘIZUJE DEPENDENCY PROVIDER. NÁŠ OBJEKT TEDY POTŘEBUJE UŽ JEN REFERENCI NA DEPENDENCE PROVIDERA.
- DI ZAHRNUJE NEJMÉNĚ TŘI OBJEKTY, KTERÉ SPOLU MUSÍ SPOLUPRACOVAT. **KONZUMENT**, TEDY OBJEKT POŽADUJÍCÍ SLUŽBY. **POSKYTOVATEL** SLUŽBY - POSKYTOVATELE TĚCHTO SLUŽEBMU DODÁ **DI PROVIDER**
- DI PROVIDER ZODPOVÍDÁ ZA CELÝ ŽIVOTNÍ CYKLUS POSKYTOVATELE.
- DI PROVIDER MŮŽE BÝT IMPLEMENTOVÁN NĚKOLIKA ZPŮSOBY, NAPŘ. JAKO LOKÁTOR SLUŽEB, ABSTRAKTNÍ TOVÁRNA, TOVÁRNÍ METODA, NEBO POMOCÍ NĚJAKÉHO Z ŘADY Z FRAMEWORKŮ, NAPŘÍKLAD SPRING NEBO CDI.

ZPŮSOBY VKLÁDÁNÍ

- MARTIN FOWLER POUKAZUJE NA TŘI RŮZNÉ VZORY, JAK LZE OBJEKTU PŘIDAT EXTERNÍ REFERENCI JINÉHO OBJEKTU.
 - VKLÁDÁNÍ ROZHRAŇÍM - EXTERNÍ MODUL, KTERÝ JE DO OBJEKTU PŘIDÁN, IMPLEMENTUJE ROZHRAŇÍ, JEŽ OBJEKT OČEKÁVÁ V DOBĚ SESTAVENÍ PROGRAMU.
 - VKLÁDÁNÍ SETTER METODOU - OBJEKT MÁ SETTER METODU, POMOCÍ NÍŽ LZE ZÁVISLOST INJEKTOVAT.
 - VKLÁDÁNÍ KONSTRUKTOREM - ZÁVISLOST JE DO OBJEKTU INJEKTOVÁNA V PARAMETRU KONSTRUKTORU JIŽ PŘI JEHO ZRODU.

VÝHODY A NEVÝHODY DI

- **VÝHODY**

- PROVIDER MŮŽE POSKYTOVAT ŘADU KOMPONENT K POUŽITÍ BEZ ZÁSAHU DO KÓDU PROGRAMU
- O PŘETÝPOVÁVÁNÍ SE STARÁ IOC KONTEJNER
- ZÁVISLOSTI KOMPONENT JSOU DEFINOVÁNY EXPLICITNĚ, TEDY JEDNODUŠŠÍ ORIENTACE V ZÁVISLOSTECH
- IOC KONTEJNER LZE VE VĚTŠINĚ PŘÍPADŮ KONFIGUROVAT MIMO KÓD, JEDNOTLIVÉ KOMPONENTY JE MOŽNÉ VYUŽÍT V JINÝCH PROGRAMECH BEZ PŘEPISOVÁNÍ KÓDU

- **NEVÝHODY**

- MENŠÍ PŘEHLEDNOST KÓDU
- NÁROČNĚJŠÍ SPRÁVA KÓDU

DI FRAMEWORKY

- SPRING FRAMEWORK
- JAVA EE WEBBEANS(CONTEXTS AND DEPENDENCY INJECTION - CDI)
- GOOGLE GUICE
- PICOCONTAINER
- JBOSS MICROCONTAINER

SPRING FRAMEWORK

- **POPULÁRNÍ OPEN-SOURCE FRAMEWORK (OZNAČOVÁN TAKÉ JAKO KONTEJNER) PRO VÝVOJ J2EE APLIKACÍ**
- PRVNÍ VERZE BYLA NAPSÁNA RODEM JOHNSONEM, KTERÝ JI VYDAL V RÁMCI PUBLIKACE SVÉ KNIHY EXPERT ONE-ON-ONE J2EE DESIGN AND DEVELOPMENT V ŘÍJNU 2002. ROD JOHNSON SE VE SVÉ KNIZE ZABÝVÁ VÝVOJEM J2EE APLIKACÍ A VĚNUJE POZORNOST PROBLÉMŮM, SE KTERÝMI SE PROGRAMÁTOŘI SETKÁVAJÍ. V KNIZE JE PRŮBĚŽNĚ PREZENTOVÁN KÓD FRAMEWORKU, KTERÝ SE NAZÝVÁ INTERFACE21, A MĚL BY VÝVOJ J2EE APLIKACÍ USNADNIT. ZA POMOCI JUERGENA HOELLERA JE POZDĚJI FRAMEWORK ROZŠÍŘEN A POD NÁZVEM SPRING FRAMEWORK UVOLNĚN JAKO OPEN-SOURCE.

SPRING

- ZÁKLADNÍM DŮVODEM VZNIKU SPRING FRAMEWORKU JE USNADNĚNÍ VÝVOJE ENTERPRISE APLIKACÍ:
 - ODSTRANĚNÍ TĚSNÝCH PROGRAMOVÝCH VAZEB JEDNOTLIVÝCH POJO OBJEKTŮ A VRSTEV POMOCÍ NÁVRHOVÉHO VZORU INVERSION OF CONTROL.
 - MOŽNOST VOLBY IMPLEMENTACE (EJB, POJO) BUSINESS VRSTVY PRO APLIKAČNÍ ARCHITEKTURU A NE NAOPAK (ABY ARCHITEKTURA PŘEDEPISOVALA IMPLEMENTACI)
 - ŘEŠENÍ RŮZNÝCH APLIKAČNÍCH DOMÉN BEZ NUTNOSTI POUŽITÍ EJB, NAPŘÍKLAD TRANSAKČNÍ ZPRACOVÁNÍ, PODPORA PRO REMOTING BUSINESS VRSTVY FORMOU WEBOVÝCH SLUŽEB ČI RMI.
 - PODPORA IMPLEMENTACE KOMPONENT PRO PŘÍSTUP K DATŮM, AŽ JIŽ FORMOU PŘÍMÉHO JDBC ČI ORM (OBJECT-RELATION MAPPING) TECHNOLOGIÍ A NÁSTROJŮ JAKO JE HIBERNATE, TOPLINK, IBATIS NEBO JDO.
 - ODSTRANĚNÍ ZÁVISLOSTI KONFIGURACÍCH A PRACNÉHO DOHLEDÁVÁNÍ JEJICH VÝZNAMU.
 - ABSTRAKCE VEDOUcí KE ZJEDNODUŠENÉMU POUŽÍVÁNÍ DALŠÍCH ČÁSTI J2EE, JAKO NAPŘÍKLAD JMS, JMX, JAVAMAIL, JDBC, JCA NEBO JNDI.
 - USNADNĚNÍ POUŽÍVÁNÍ A PSANÍ UNIT TESTŮ.
 - SPRÁVA A KONFIGURAČNÍ MANAGEMENT BUSINESS KOMPONENT.

SPRING 3 PŘÍKLAD

- ANOTACE JAVAX.INJECT.NAMED – TAKTO OZNAČENOU BEAN SE SPRING BUDE SNAŽIT INJECTOVAT
- ANOTACE JAVAX.INJECT.INJECT – OZNAČUJE MÍSTO, KDE MÁ SPRING INJECTOVAT BEAN

```
@Named  
public class VolvoCarFactory implements CarFactory {
```

```
@Inject CarFactory carFactory;
```