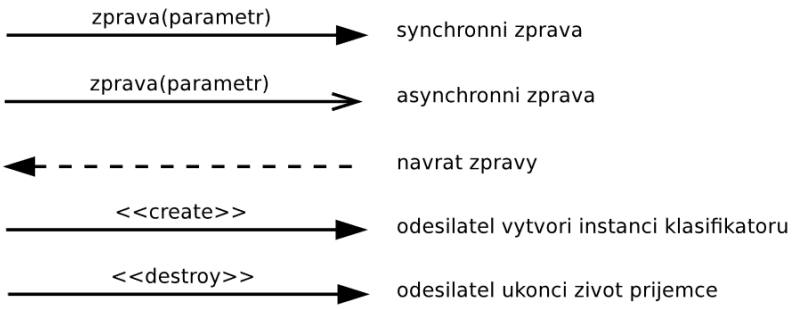
# Jazyk UML, OCL, návrhové vzory

- **Základní rozdíl mezi objektovým a strukturovaným přístupem**
  - Strukturovaný má oddělená data a nad tím se jede funkce
  - Objektový má data a funkce dohromady, objekt uchovává data a funkce objektu s nimi manipulují
- **Diagramy jazyka UML 2.0**
  - Diagramy interakce
    - **Sekvenční diagram (Sequence Diagram)**
    - **Diagram komunikace (Communication Diagram)**
    - Diagram přehledu interakcí (Interaction Overview Diagram)
    - Diagram časování (Timing Diagram)

    - Popisují spolupráci objektů
    - Typicky modelují chování jednoho případu užit´
    - **Čára života**
      - Zastupuje jednoho účastníka interakce (objekt)
      - Označení: nazev[selektor]:typ
        - Nazev – identifikátor čáry života (objektu)
        - Selektor – podmínka pro výběr určité instance
        - Typ – klasifikátor, jehož je čára života instanc´
      - honzuvUcet [ id = '15' ] : Ucet
      - **Pozn.:**
        - **Objekt** – podtržený, má dvojtečku
        - **Třída** – čistý text
    - **Zprávy**
      - Komunikace mezi účastníky interakce
      - Typy zpráv

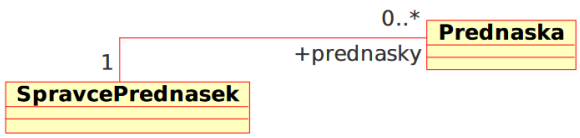


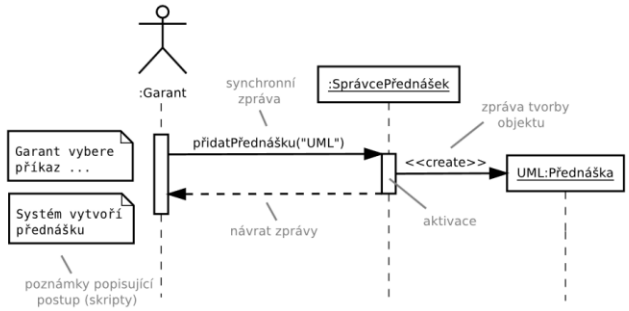

    - **Základní typy diagramů interakce**
      - **Sekvenční diagramy (Sequence Diagrams)**
        - Zdůrazňují časově orientovanou posloupnost předávání zpráv mezi objekty (chronologie zasílání zpráv)
        - Bývají přehlednější a srozumitelnější než diagramy komunikace
        - Každá čára života (objekt) je zobrazena s časovou osou
      - **Diagramy komunikace (Communication Diagrams)**
        - Zdůrazňují strukturální vztahy mezi objekty
        - Výhodné pro rychlé zobrazení komunikace mezi objekty

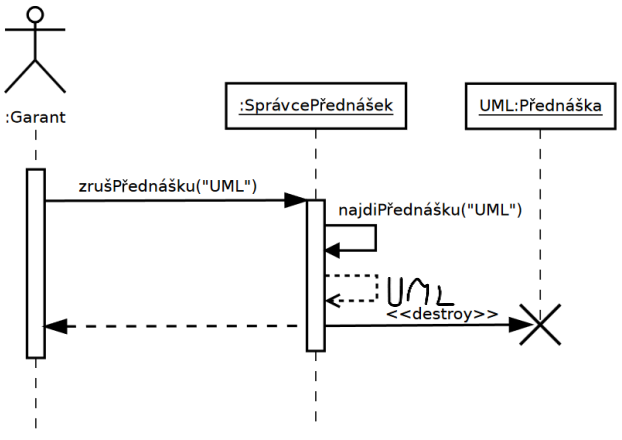
- **Pozn.:** Doménový model – Zachycuje koncepty a vztahy mezi nimi, není úplný, jenom konceptuální
- **Sekvenční diagram**
  - Vyjdeme z následující specifikace případu užití (uvedená specifikace je pouze ilustrativní, v reálném systému by se řešilo jinak):

| **Případ užití: Přidat přednášku** |
|---|
| **ID: UC11** |
| **Účastníci:**<br>Garant |
| **Vstupní podmínky:**<br>1. Garant je přihlášen do systému. |
| **Tok událostí:**<br>1. Garant zadá příkaz "přidat přednášku".<br>2. Systém přijme název nové přednášky.<br>3. Systém vytvoří novou přednášku. |
| **Následné podmínky:**<br>1. Nová přednáška byla přidána do systému. |

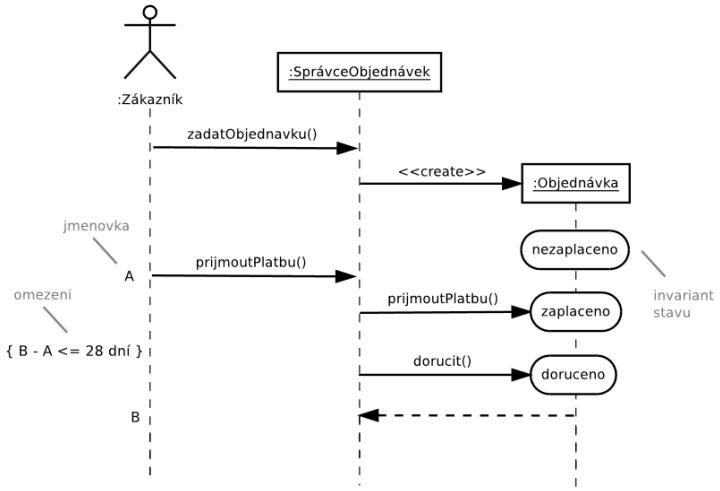
  - Na základě počáteční analýzy případu užití vytvoříme diagram tříd (doménový model)



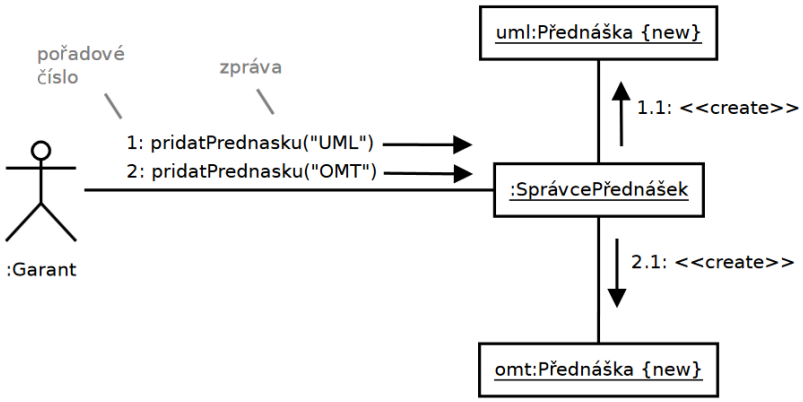

  - Příklad:
    - 

    - 

- Rozšíření sekvenčních diagramů (omezení, zobrazení stavů)



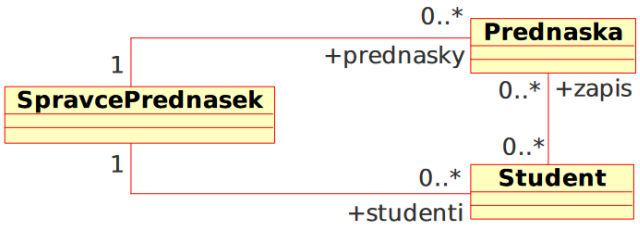

- **Diagram komunikace**
  - Objekty jsou spojeny linkami (komunikační kanály)
  - Zprávy jsou řazeny podle hierarchického číslování



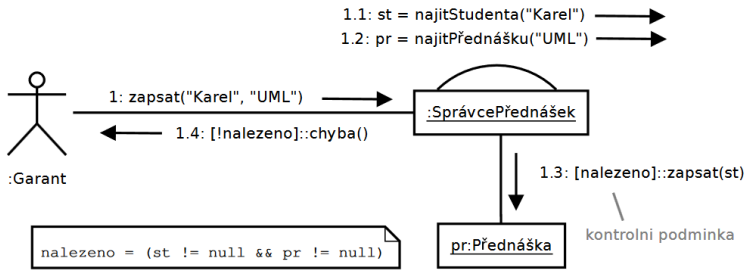

  - Pro analýzu přidáme následující specifikaci případu užití (uvedená specifikace je pouze ilustrativní, v reálném systému by se řešilo jinak):

| **Případ užití: Zapsat studenta na přednášku** |
|---|
| **ID: UC17** |
| **Účastníci:**<br>Garant, Student |
| **Vstupní podmínky:**<br>1. Garant je přihlášen do systému. |
| **Tok událostí:**<br>1. Garant zadá příkaz "zapsat studenta".<br>2. Systém vyhledá studenta **S** podle zadaného jména.<br>3. Systém vyhledá přednášku **P** podle zadaného názvu.<br>4. Systém zapíše studenta **S** na přednášku **P**. |
| **Následné podmínky:**<br>1. Student **S** je zapsán na přednášku **P**. |

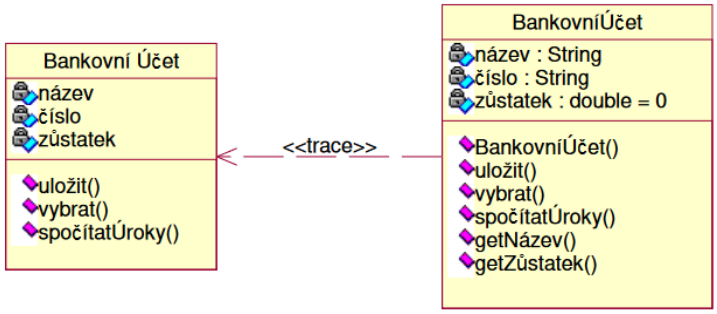
- **Diagram tříd**




- **Diagram komunikace**
  - větvení, kontrolní podmínky

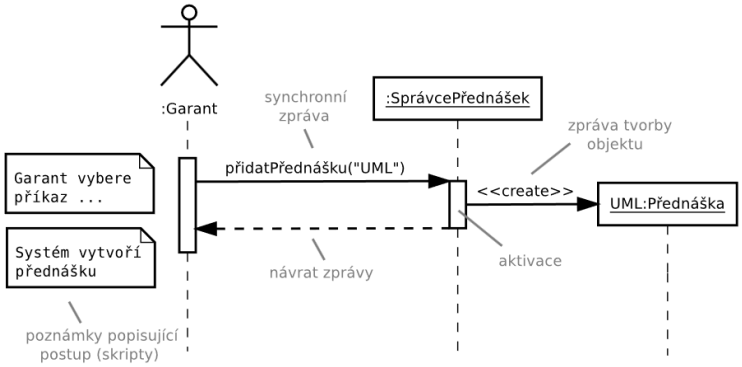



- **UML v etapách vývoje softwaru**
  - **Analytické modely**
    - Zaměřují se na otázku co, neodpovídá detailně na otázku jak
    - Zobrazují důležité koncepty (objekty, vztahy, . . . ) z problémové domény
      - Třídy Zákazník, Košík, . . .
      - Třída pro přístup k databázím patří do řešení (návrhu)
  - **Tvorba analytických modelů**
    - Doménový model (analytické třídy)
    - Diagramy případů užití
    - **Specifikace případů užití**
      - Diagramy aktivit
      - Stavové diagramy
    - **Realizace případů užití**
      - Modelují interakce (zasílání zpráv) konceptuálních objektů
      - Diagramy interakce
  - **Návrhové modely**
    - • vychází z výstupů etapy analýzy
    - • zaměřují se na otázku jak, věnuje se detailům
    - • specifikace modelů je na takové úrovni, že je lze přímo implementovat
  - **Tvorba návrhových modelů**
    - Upřesňování analytických diagramů
      - Návrhové třídy
      - Realizace případů užití
      - . . .
  - **Návrhové třídy**
    - specifikace návrhových tříd je na takovém stupni, že je lze přímo implementovat
    - upřesňování analytických tříd
    - využití tříd z doménového řešení
      - knihovny, vrstva aplikačního serveru, GUI, . . .

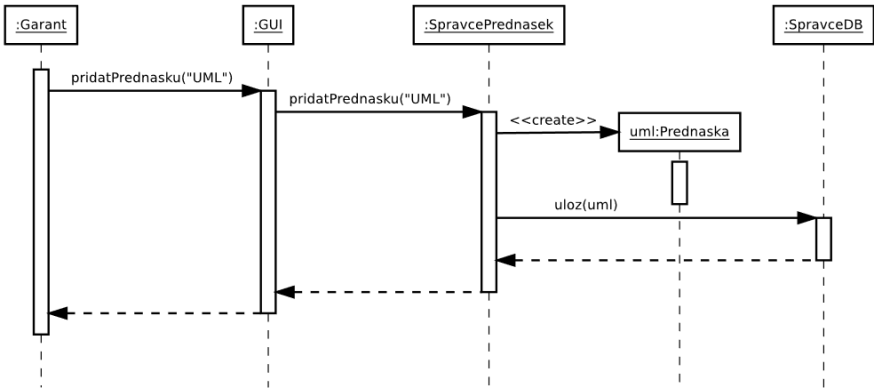



- 
- Koncept nalevo, návrhová úroveň napravo

- **Upřesnění modelu v etapě návrhu**
  - **Analytický model interakce**

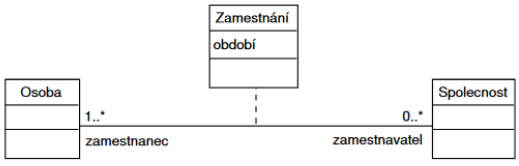


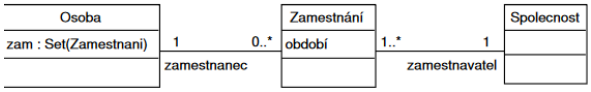

  - 
  - **Návrhový model interakce**




  - 

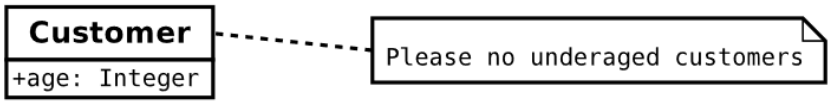
- **Upřesnění analytických relací**
  - **Asociace**
    - agregace vs. kompozice
      - upřesnění vztahu celek/část
    - asociace povýšená na třídu
      - návrh asociačních tříd
    - asociace typu 1:N
      - realizace (nejčastěji kolekce)

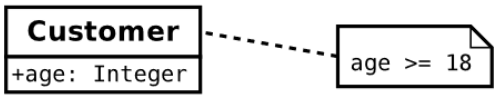



    - 

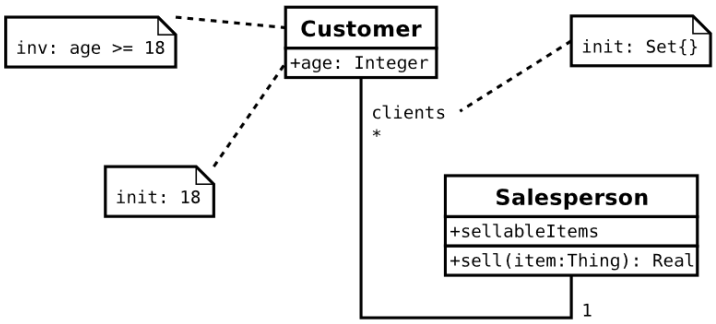
- **Mechanismy rozšiřitelnosti UML**
  - **Omezení (Constraints)**
    - definují omezující podmínky
    - rozšiřují sémantiku elementu (např. OCL)
  - **Stereotypy (Stereotypes)**
    - definuje nový element na základě stávajícího elementu
    - stereotyp má svou sémantiku
  - **Označené hodnoty (tagged values)**
    - {tag1 = hodnota1, tag2 = hodnota2}
    - většinou se přidružují k stereotypu, vyjadřují vlastnosti nových elementů

  - Jazyk UML – omezení a dotazy nad modely – Modelovací techniky UML
    - nedokáží zachytit všechny závislosti mezi elementy graficky
    - ˇreší se poznámkou se slovním popisem
    - slovní popis je nedostateˇcn´y
      - není vždy jednoznaˇcn´y, m˚uže b´yt r˚uzně pochopen
      - komplikuje automatické konverze
    - 

    - 

    - ⇒ **jednoznačný jazyk** ⇒ **OCL**
    - **Object Constraint Language (OCL)**
      - speci´aln´ı form´aln´ı jazyk pro UML
      - OCL nen´ı programovac´ı jazyk
      - definuje omezen´ı, podm´ınky a dotazy nad UML modely
        - ⇒ zpˇresˇnov´an´ı model˚u
      - je form´aln´ı deklarativn´ı jazyk navrˇzen´y pro n´avrh´aˇre
        - ⇒ nevyˇzaduje se siln´y matematick´y z´aklad
      - spojen´y s dalˇs´ımi metamodely definovan´ymi OMG
      - umoˇzˇnuje transformace model˚u
      - **Vyuˇzit´ı OCL**
        - specifikace podm´ınek pro vykon´an´ı metod
        - specifikace invariant˚u tˇr´ıd
        - specifikace poˇc´ateˇcn´ıch hodnot atribut˚u
        - specifikace tˇela operace
        - specifikace omezen´ı
        - . . .

- **Typy omezen´ı**
  - **invariant** – podm´ınka, kter´a mus´ı b´yt v˘zdy spln˘ena v˘semi instancemi
  - **precondition** – omezen´ı, kter´e mus´ı b´yt pravdiv´e p˘red proveden´ım operace
  - **postcondition** – omezen´ı, kter´e mus´ı b´yt pravdiv´e t˘esn˘e po ukon˘cen´ı operace
  - **guard** – omezen´ı, kter´e mus´ı b´yt pravdiv´e p˘red proveden´ım p˘rechodu mezi stavy
- **Z´akladn´ı knihovna**
  - typy: Boolean, Integer, Real, String
  - kolekce: Collection, Set, Ordered Set, Bag, Sequence
  - operace: and, or, <, size, includes, count, . . .
- **Příklad:**




  - 
    - Clients – role, vztah 1:N,
  - Ukázka textového vyjádření:
    - invarianty atribut˚u
      context Customer inv:
      age >= 18
    - kolekce objekt˚u (Salesperson 1→N Customer / role clients)
      context Salesperson inv:
      clients->size() <= 100 and
      clients->forAll(c: Customer | c.age >= 40)
    - po˘c´ate˘cn´ı hodnoty
      context Customer::age : Integer
      init: 18

      context Salesperson::clients : Set(Customer)
      init: Set
    - precondition, postcondition
      context Salesperson::sell( item: Thing ): Real
      pre: self.sellableItems->includes( item )
      post: not self.sellableItems->includes( item ) and
      result = item.price
    - podm´ınky
      self.clients.select(c : Customer | c.age > 50)

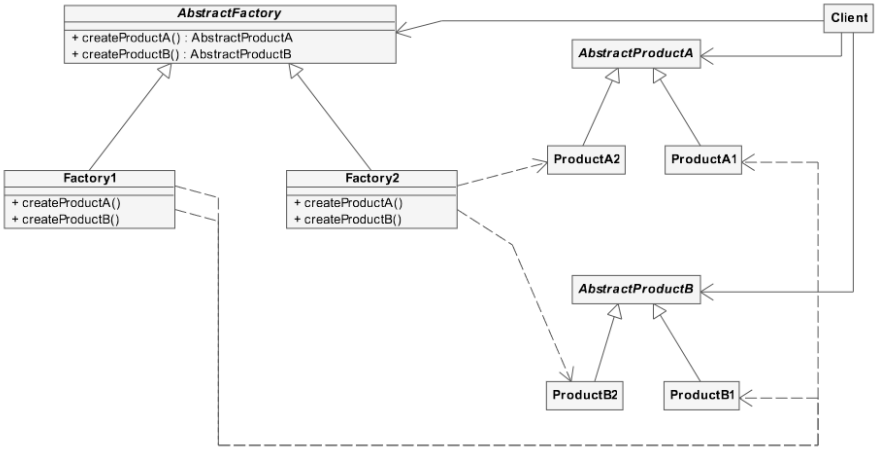
- **Objektov˘e orientovan´y n´avrh a programov´an´ı**
  - **znovupou˘zitelnost?**
    - zaji˘st˘en´ı znovupou˘zitelnosti ⇒ obecn´y n´avrh
    - zaji˘st˘en´ı aplikovatelnosti na ˘re˘sen´y probl´em ⇒ specifick´y n´avrh
    - spor

- **... p˘resto**
  - pro˘c nevyu˘z´ıt ˘re˘sen´ı, kter´e ji˘z fungovalo
  - takov´a ˘re˘sen´ı jsou v´ysledkem mnoha pokus˚u a pou˘z´ıv´an´ı
  - ⇒ vzory pro ˘re˘sen´ı stejn´ych typ˚u probl´em˚
- **Návrhový vzor**
  - **N´avrhov´e vzory**
    - z´akladn´ı sada ˘re˘sen´ı d˚ule˘zit´ych a st´ale se opakuj´ıc´ıch n´avrh˚u
    - usnad˘nuj´ı znovupou˘zitelnost
    - umo˘z˘nuj´ı efektivn´ı n´avrh (v´yb˘er vhodn´ych alternativ, dokumentace, . . . )
  - **N´avrhov´y vzor**
    - vzor je ˘sablona pro ˘re˘sen´ı, nikoli implementace probl´emu!
    - ka˘zd´y vzor popisuje probl´em, kter´y se neust´ale vyskytuje, a j´adro ˘re˘sen´ı dan´eho probl´emu
    - umo˘z˘nuje jedno ˘re˘sen´ı pou˘z´ıvat mnohokr´at, ani˘z bychom to d˘elali dvakr´at stejn´ym zp˚usobem
  - **Prvky n´avrhov´eho vzoru**
    - **n´azev**
      - kr´atk´y popis (identifikace) n´avrhov´eho probl´emu
      - probl´em
      - popis, kdy se m´a vzor pou˘z´ıvat (vysv˘etlen´ı probl´emu, podm´ınky pro smyslupl´e pou˘zit´ı vzoru, . . . )
    - **˘re˘sen´ı**
      - popis prvk˚u n´avrhu, vztah˚u, povinnost´ı a spolupr´ace
      - nepopisuje konkr´etn´ı n´avrh, obsahuje abstraktn´ı popis probl´emu a obecn´e uspo˘r´ad´an´ı prvk˚u pro jeho ˘re˘sen´ı
    - **d˚usledky**
      - v´ysledky a kompromisy (vliv na roz˘si˘ritelnost, p˘renositelnost, . . . )
      - d˚ule˘zit´e pro hodnocen´ı n´avrhov´ych alternativ – n´aklady a v´yhody pou˘zit´ı vzoru
  - **Typy vzorů**
    - **Vzory se mohou t´ykat**
      - **t˘r´ıd**
        - zab´yvaj´ı se vztahy mezi t˘r´ıdami a podt˘r´ıdami (vztah je fixov´an)
      - **objekt˚u**
        - zab´yvan´ı se vztahy mezi objekty, jsou dynami˘ct˘ej˘s´ı
    - **Z´akladn´ı rozd˘elen´ı vzor˚u**
      - **tvo˘riv´y**
        - zab´yv´a se procesem tvorby objekt˚u
      - **struktur´aln´ı**
        - zab´yv´a se skladbou t˘r´ıd ˘ci objekt˚u
      - **chov´an´ı**
        - zab´yv´a se zp˚usoby vz´ajemn´e interakce mezi objekty nebo t˘r´ıdami
        - zab´yv´a se zp˚usoby rozd˘elen´ı povinnost´ı mezi objekty nebo t˘r´ıdy

- **Základní návrhové vzory**

| Tvořivý | Strukturální | Chování |
|---|---|---|
| Factory method | Adapter (class) | Interpreter |
| Abstract Factory | Adapter (object) | Iterator |
| Singleton | Decorator | Visitor |
| Prototype | Facade | Memento |
| Builder | Bridge | Observer |
| | Flyweight | Mediator |
| | Composite | Command |
| | Proxy | Chain of Responsibility |
| | | State |
| | | Strategy |

- **Abstraktn´ı tov´arna (Abstract Factory)**
  - **´U˘cel**
    - vytv´a˘ren´ı p˘r´ıbuzn´ych nebo z´avisl´ych objekt˚u bez specifikace konkr´etn´ı t˘r´ıdy
    - tvo˘riv´y vzor – objekty
  - **Motivace**
    - nap˘r. zm˘ena vzhledu sady grafick´ych n´astroj˚u
  - **D˚usledky**
    - izoluje konkr´etn´ı t˘r´ıdy – klient pracuje pouze s rozhran´ım
    - usnad˘nuje v´ym˘enu produktov´ych ˘rad (nap˘r. zm˘ena vzhledu, . . . )
    - podpora zcela nov´ych produktov´ych ˘rad je obt´ı˘zn˘ej˘s´ı
    - . . .
  - **Struktura**




## Abstraktní továrna: Příklad

Příklad bludiště, pracuje s objekty zeď a brána:

```
public class MazeGame {
    public Maze createNewMaze() {
        StdWall wall = new StdWall();
        StdGate gate = new StdGate();
        ...
    }
    private doSomething(StdWall wall) { .
}
```

Použití

```
MazeGame game = new MazeGame();
game.createNewMaze();
```

Úprava na novou sadu objektů:

```
public class MazeGame {
    public Maze createNewMaze() {
        SpecWall wall = new SpecWall();
        SpecGate gate = new SpecGate();
        ...
    }
    private doSomething(SpecWall wall) { ... }
}
```

Řešení

- přepis stávajícího kódu ⇒ ztrácíme původní verzi
- kopie stávajícího kódu ⇒ musíme udržovat více verzí
  ⇒ *nemožnost dynamické změny*
- vytvořit flexibilní kód ⇒ návrhový vzor *Abstraktní továrna*

Vytvoříme abstraktní prvky podle vzoru:

```
// abstraktní produkty
public interface Wall { ... }
public interface Gate { ... }

// abstraktní továrna
public abstract class MazeFactory {
    public abstract Wall createWall();
    public abstract Gate createGate();
}
```

Upravíme původní kód podle vzoru:

```
// aplikace vzoru v původním kódu
public class MazeGame() {
    public Maze createNewMaze(MazeFactory factory) {
        // StdWall wall = new StdWall();
        Wall wall = factory.createWall();

        // StdGate gate = new StdGate();
        Gate gate = factory.createGate();
        ...
    }
    private doSomething(Wall wall) { ... }
}
```

Vytvoříme konkrétní prvky podle vzoru:

```
// konkrétní produkty
public class StdWall implements Wall { ... }
public class StdGate implements Gate { ... }

// konkrétní továrna
public class StdMazeFactory {
    public Wall createWall() { return new StdWall(); }
    public Gate createGate() { return new StdGate(); }
}
```

Použijeme konkrétní prvky:

```
MazeGame game = new MazeGame();
MazeFactory factory = new StdMazeFactory();
game.createNewMaze(factory);
```

Vytvoříme jinou sadu prvků:

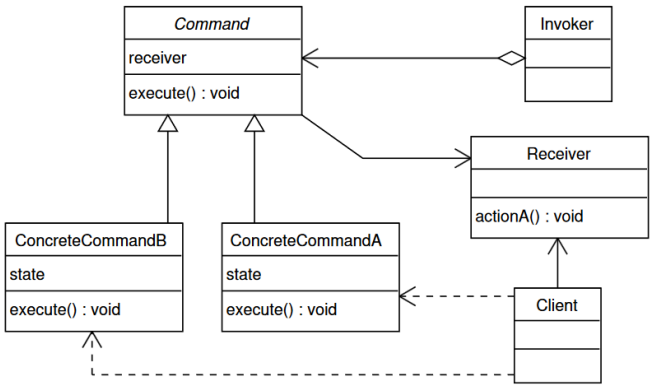
```
// konkrétní produkty
public class SpecWall implements Wall { ... }
public class SpecGate implements Gate { ... }

// konkrétní továrna
public class SpecMazeFactory {
    public Wall createWall() { return new SpecWall(); }
    public Gate createGate() { return new SpecGate(); }
}
```

Použijeme konkrétní prvky *bez modifikace kódu bludiště*:

```
MazeGame game = new MazeGame();
MazeFactory factory = new SpecMazeFactory();
game.createNewMaze(factory);
```

- **Command**
  - **´U˘cel**
    - zapouzd˘ren´ı po˘zadavk˚u nebo operac´ı
    - vzor chov´an´ı
  - **Motivace**
    - zasl´an´ı po˘zadavku na obecn´e ´urovni, ani˘z zn´ame konkr´etn´ı protokol
    - podpora undo operac´ı
  - **D˚usledky**
    - reprezentuje jeden proveden´y p˘r´ıkaz
    - umo˘z˘nuje uchov´avat stav klienta p˘red proveden´ım p˘r´ıkazu
    - . . .

- **Struktura**




## Command – Příklad

**Původní operace:**

```
t1.remove(ch);
t2.put(ch);
```

⇒

**Aplikace vzoru:**

```
cmd = new CommandA(t1,t2,ch);
invoker.putAndExec(cmd);
...
invoker.removeAndUndo();
```

**Invoker:**

```
putAndExec(cmd) {
  stack.push(cmd);
  cmd.execute();
}

removeAndUndo() {
  cmd = stack.pop();
  cmd.undo();
}
```

**CommandA:**

```
execute() {
  t1.remove(ch);
  t2.put(ch);
}

undo() {
  t2.remove(ch);
  t1.put(ch);
}
```

- Ukázka sekvenčního diagramu pro popis chování.