Návrhový vzor Builder

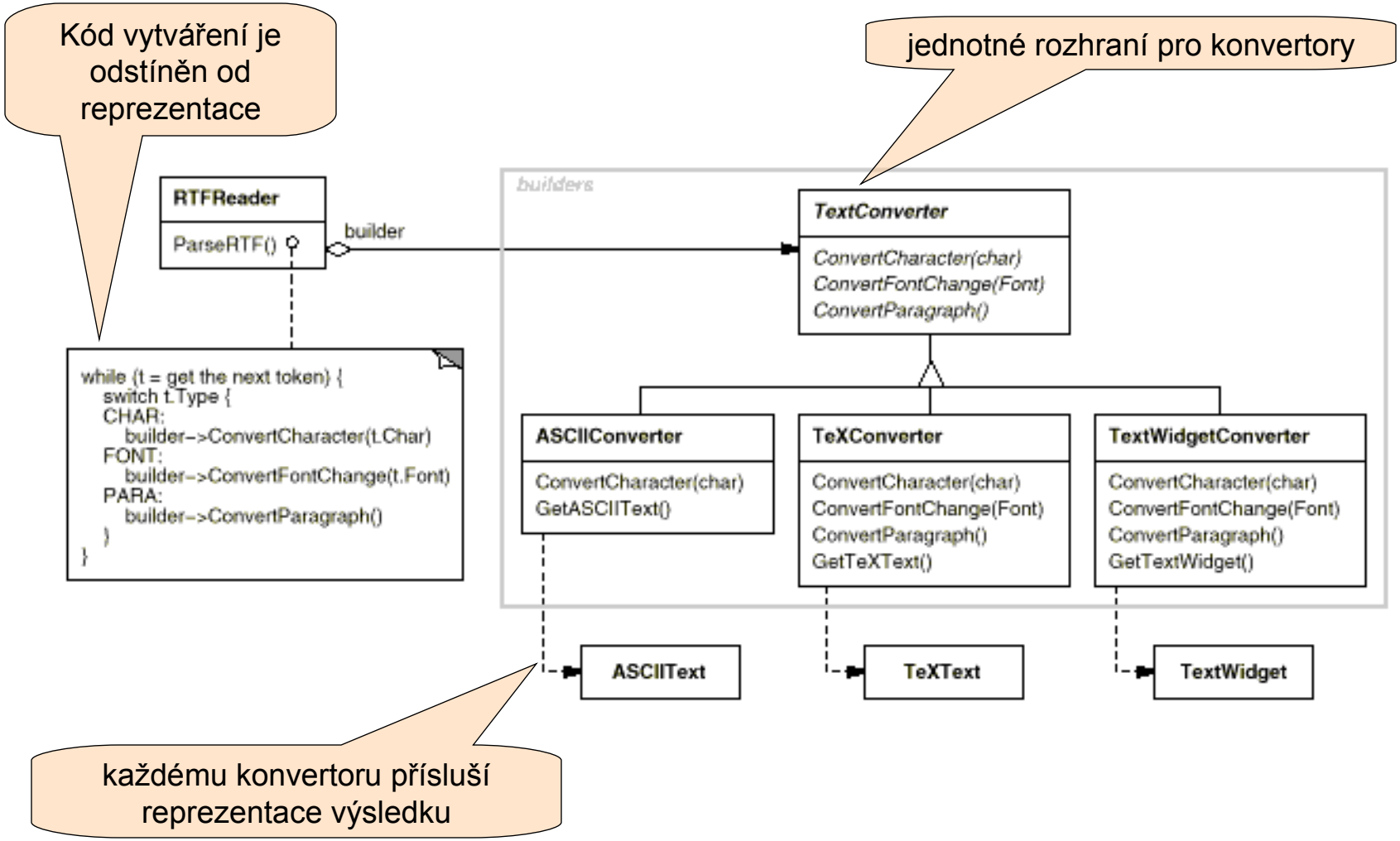
Builder

- **Účel**
  - kategorie: vytvářecí pro objekty *(object creational)*
  - Složité objekty vytvářené step-by-step
  - Oddělení konstrukce od reprezentace: stejný postup - více reprezentací

- **Motivace – převod dokumentu z formátu RTF na jiné formáty**
  - Máme třídu **`RTFReader`** pro čtení dokumentu ve formátu RTF
  - chceme převádět obsah dokumentu do různých formátů – neznáme předem
    …výsledkem konverze budou různé (zpravidla složené) typy
  - vybavíme **`RTFReader`** objektem splňující interface **`TextConverter`**, ten se postará o konverzi tokenů a stavbu nového dokumentu v nějakém formátu.
  - **`RTFReader`** objekt čte tokeny a vysílá příkazy ke konverzi na interface
  - **`TextConverter`** objekt rozhoduje o způsobu konverze a reprezentaci výsledku

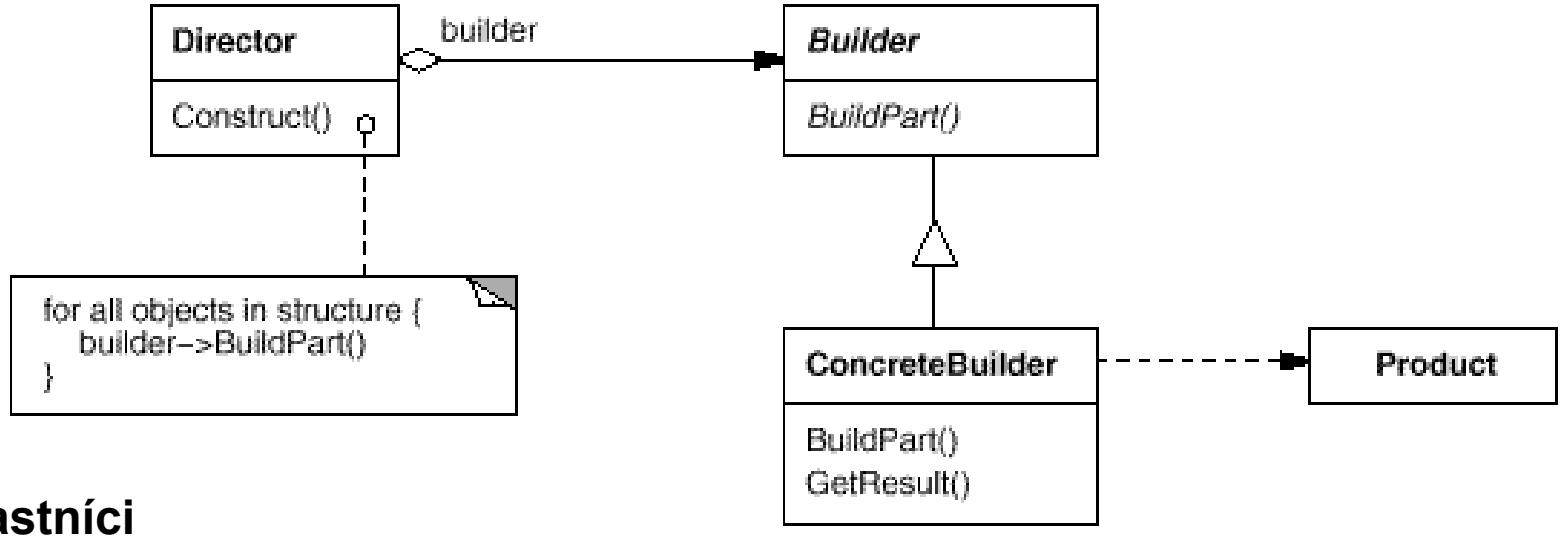
- **Použitelnost**
  - algoritmus vytvoření složitého objektu:
    typ vs. struktura
  - chceme stejným postupem vytvářet různé reprezentace

Builder - motivace



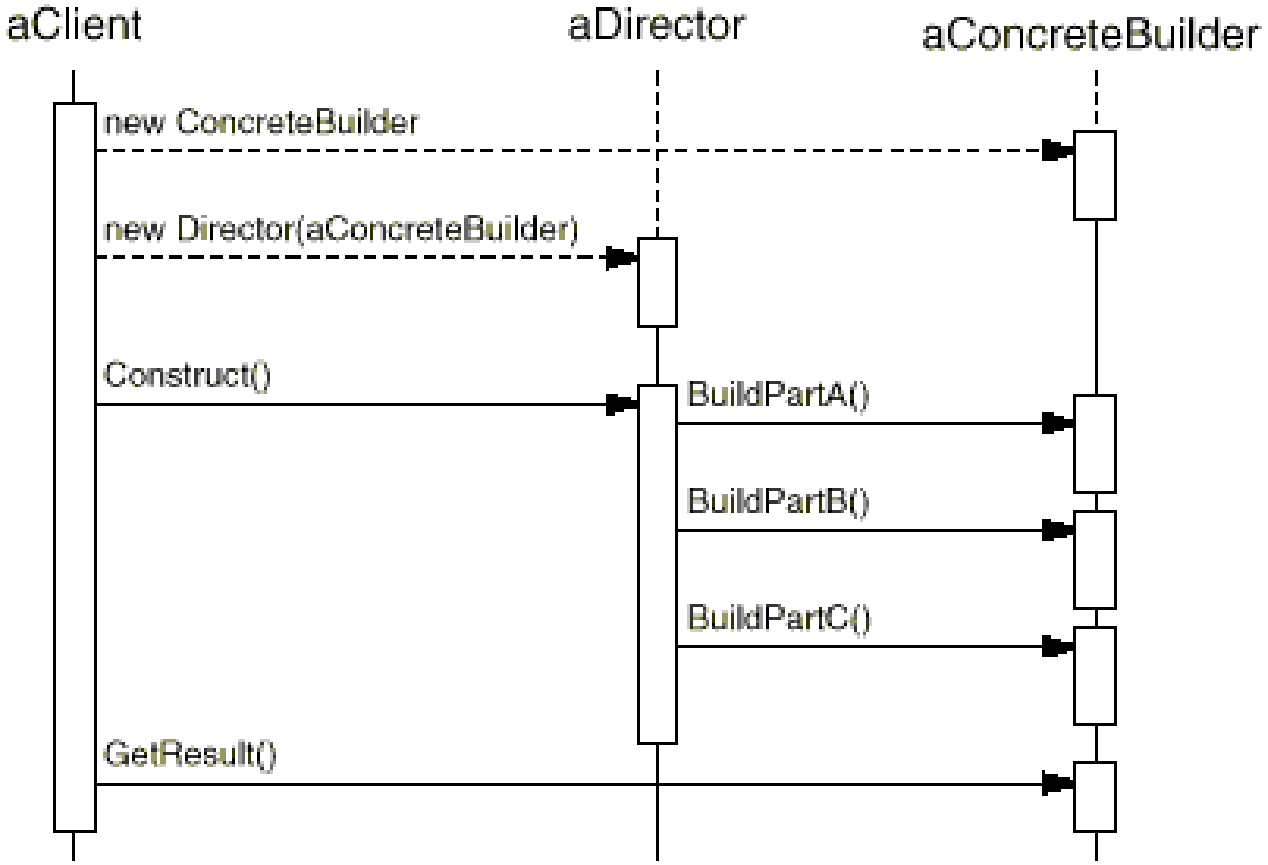
Builder - struktura

- **Struktura**



- **Účastníci**
  - Builder (TextConverter)
    - definuje rozhraní pro tvoření částí Productu
  - ConcreteBuilder (ASCIIConverter, TeXConverter, TextWidgetConverter)
    - implementuje rozhraní Builderu
    - definuje a skladuje vytvořenou reprezentaci (Product), umí ji vrátit
  - Director (RTFReader)
    - po krocích vytváří objekt pomocí funkcí z rozhraní Builderu
  - Product (ASCIIText, TeXText, TextWidget)
    - reprezentuje strukturu vytvářeného objektu

Builder - dynamika



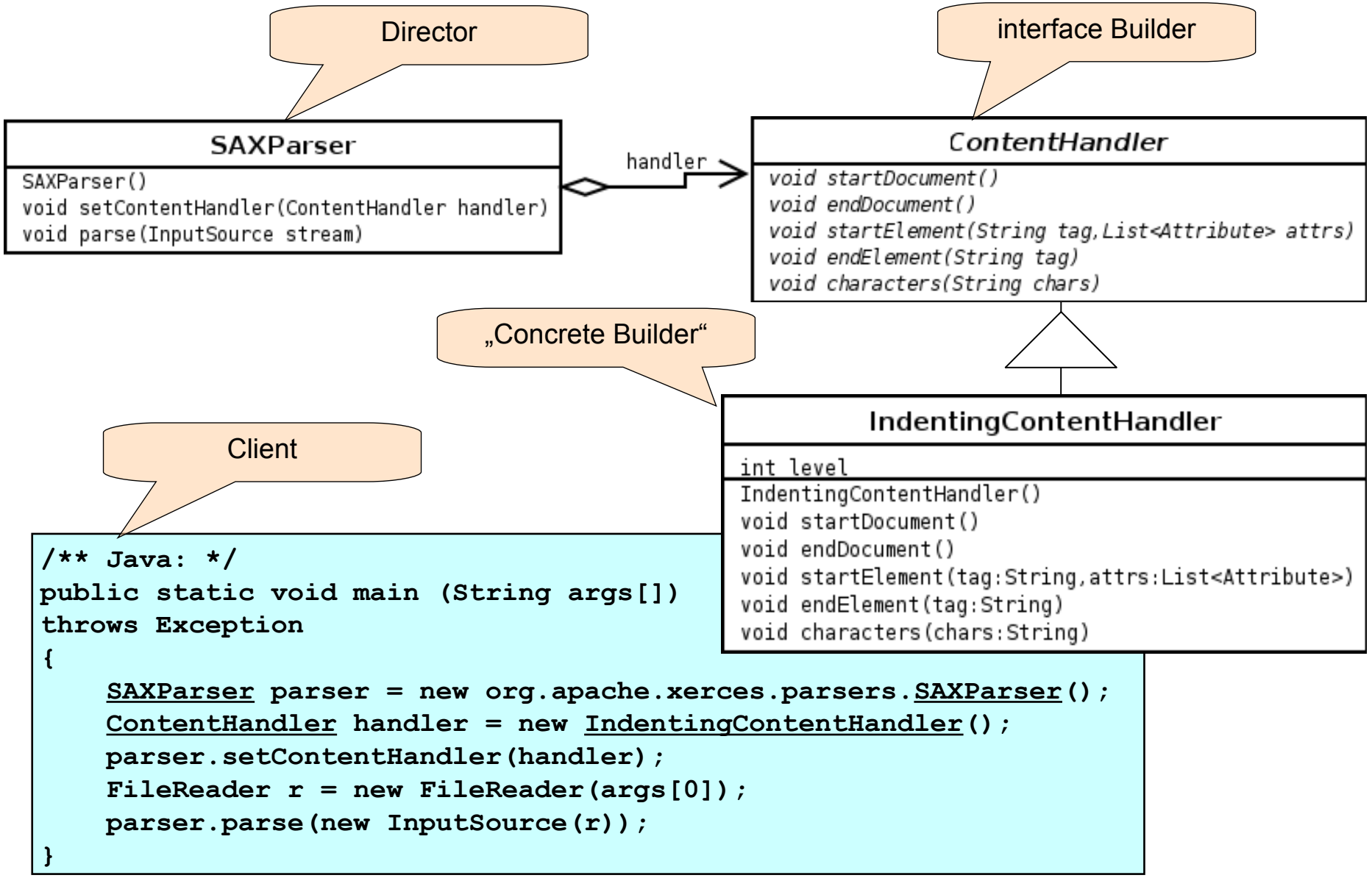
Builder - souvislosti

- **Souvislosti**
  - umožňuje měnit vnitřní reprezentaci produktu
    - stačí vytvořit nový typ Builderu
  - izolace kódu pro konstrukci a reprezentaci
    - stejný Builder lze použít pro další Director
  - umožňuje kontrolu konstrukce v jejím průběhu

Builder - implementace

- **Implementace**
  - abstraktní třída Builder, definuje funkce dostupné Directoru, které nic nedělají
    - ConcreteBuilder implementuje jen ty, které hodlá zpracovávat
  - rozhraní pro konstrukci
    - nově zkonstruované části zpravidla stačí obyčejně připojit
      - v příkladu s čtením RTF – další zkonvertované tokeny se přidávají na konec
    - někdy potřebujeme identifikovat dříve vytvořené části
      - v bludišti potřebujeme při přidání dveří určit místnosti, které spojují
      - při vytváření stromů od listů chceme pro nový uzel zadat následníky
      - řešení: vracíme vytvořené objekty (nebo jejich identifikátory) Directoru, který je později použije jako parametry pro vytvoření dalších objektů
  - nedefinuje se abstraktní třída pro produkt
    - produkty se zpravidla liší tolik, že to nemá smysl
    - klient vybírá požadovaný ConcreteBuilder, proto také ví, jaký dostane produkt

Příklad Builderu: SAX



Příklad Builderu: SAX - odsazení

```
interface ContentHandler {
    protected int level = 0;

    public void startDocument();
    public void endDocument();
    public void startElement(String tag, List<Attribute> attrs);
    public void endElement(String t);
    public void characters(String chars);
    ...
}
```

- **ContentHandler**
  - Definuje „události“ při procházení XML dokumentem
  - Konkrétní instance implementuje akce provedené při těchto událostech

Příklad Builderu: SAX - odsazení

```
class IndentingContentHandler implements ContentHandler {
    protected int level = 0;
    public IndentingContentHandler() {}

    public void startDocument() {
        level = 0;
        System.out.println(„<?xml version=\"1.0\"?>“);
    }
    public void endDocument(){
    }
    public void startElement(String tag, List<Attribute> attrs){
        printIndentation(level++);
        System.out.println(„<“ + tag + „>“);
    }
    public void endElement(String t{
        printIndentation(--level);
        System.out.println(„</“ + tag + „>“);
    }
    public void characters(String chars) {
        printIndentation(level);
        System.out.printl(chars);
    }
    protected void printIndentation(int aLevel) {
        for (int i=0; i<aLevel; i++)
            System.out.print(„\t“);
    }
}
```

- **IndentingContentHandler**
  - „Concrete Builder“ (implementace rozhraní Builder)
  - pro zjednodušení vynecháváme jmenné prostory XML
  - ignorujeme atributy a další XML objekty

Příklad Builderu: SAX - odsazení

```
<?xml version="1.0"?>
<videoArchive>
    <movie lang="en">
        <title>Braveheart</title>
    </movie>
    <movie lang="cs">
        <title>Slavnosti sněženek</title>
    </movie>
    <movie lang="cs">
        <title>Kolja</title>
    </movie>
</videoArchive>
```

```
<?xml version="1.0"?>
<videoArchive>
<movie lang="en">
<title>Braveheart</title>
</movie>
<movie lang="cs">
<title>Slavnosti sněženek</title>
</movie>
<movie lang="cs">
<title>Kolja</title>
</movie>
</videoArchive>
```

- **IndentingContentHandler**
  - sjednocuje odsazení řádků
  - ignoruje atributy a další XML objekty

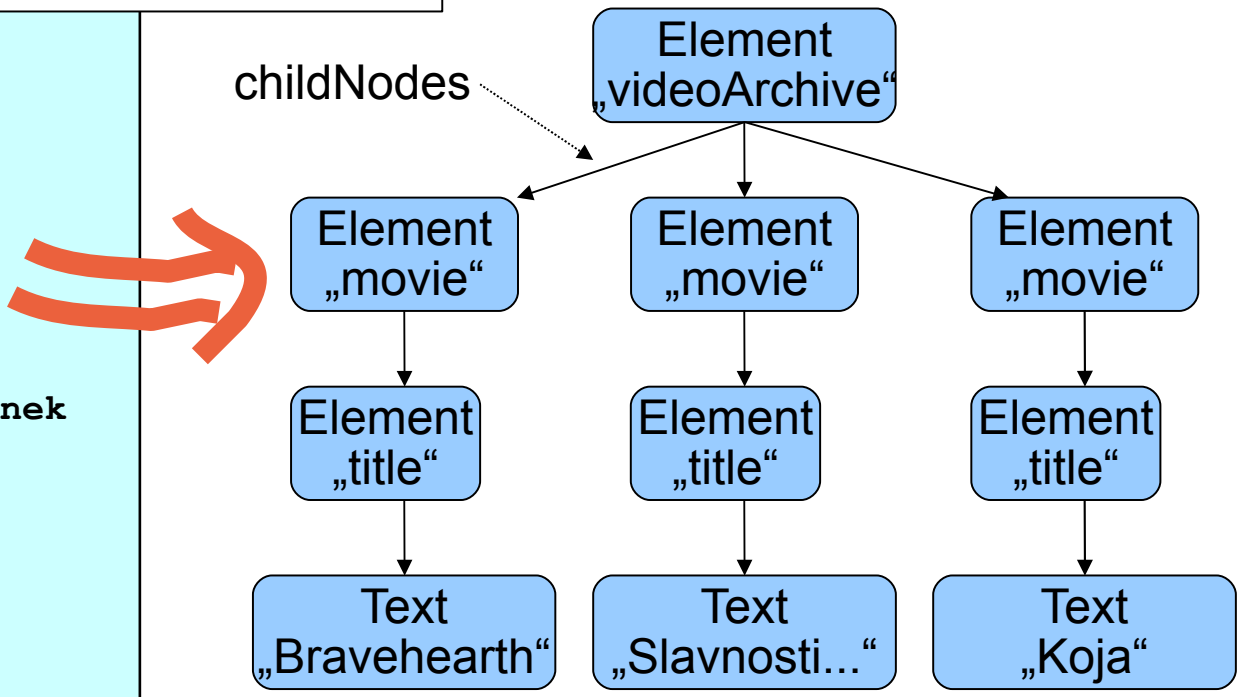
```
<?xml version="1.0"?>
<videoArchive>
    <movie>
        <title>
            Braveheart
        </title>
    </movie>
    <movie>
        <title>
            Slavnosti sněženek
        </title>
    </movie>
    <movie>
        <title>
            Kolja
        </title>
    </movie>
</videoArchive>
```

Příklad Builderu: SAX - odsazení

- **Co v příkladu chybí, aby byl příkladem pro návrhový vzor Builder?**
  - více implementací Concrete Builder: např. zápis do binárního XML (wbxml), stavba DOM modelu (stromu objektů; viz níže)
  - místo výpisu na výstup by měl stavět nějaký produkt a nakonec jej zpřístupnit pomocí getResult()

```
<?xml version="1.0"?>
<videoArchive>
    <movie>
        <title>
            Braveheart
        </title>
    </movie>
    <movie>
        <title>
            Slavnosti sněženek
        </title>
    </movie>
    <movie>
        <title>
            Kolja
        </title>
    </movie>
</videoArchive>
```

- Stavba DOM modelu:



Builder – příklad bludiště

- **Vytvoříme variantu metody CreateMaze, která použije dodaný MazeBuilder**

- **interface MazeBuilder**

```
class MazeBuilder {
public:
    virtual void BuildMaze() { }
    virtual void BuildRoom(int room) { }
    virtual void BuildDoor(int roomFrom, int roomTo) { }

    virtual Maze* GetMaze() { return 0; }
protected:
    MazeBuilder();
};
```

Výsledek

  - definuje metody pro konstrukci částí bludiště
  - GetMaze vrací vytvořené bludiště
  - funkce nejsou abstraktní, předefinují se jen ty, co mají něco dělat

Builder – příklad

- **nová verze CreateMaze dostane MazeBuilder jako parametr**

```
Maze* MazeGame::CreateMaze () {
  Maze* aMaze = new Maze;
  Room* r1 = new Room(1);
  Room* r2 = new Room(2);
  Door* theDoor = new Door(

  aMaze->AddRoom(r1);
  aMaze->AddRoom(r2);

  r1->SetSide(North, new Wall);
  r1->SetSide(East, theDoor);
  r1->SetSide(South, new Wall);
  r1->SetSide(West, new Wall);
  r2->SetSide(North, new Wall);
  r2->SetSide(East, new Wall);
  r2->SetSide(South, new Wall);
  r2->SetSide(West, the

  return aMaze;
}
```

původní kód

```
Maze* MazeGame::CreateMaze (MazeBuilder& builder)
{
     builder.BuildMaze();
     builder.BuildRoom(1);
     builder.BuildRoom(2);
     builder.BuildDoor(1, 2);
     return builder.GetMaze();
}
```

- nová metoda nepracuje s reprezentací bludiště
- MazeBuilder můžeme snadno použít pro vytváření jiných bludišť

```
Maze* MazeGame::CreateComplexMaze (MazeBuilder& builder)
{
     builder.BuildRoom(1);
     // ...
     builder.BuildRoom(1001);
     return builder.GetMaze();
}
```

Builder - příklad

- **StandardMazeBuilder – implementace rozhraní MazeBuilder**

```
class StandardMazeBuilder : public MazeBuilder {
public:
     StandardMazeBuilder();
     virtual void BuildMaze();
     virtual void BuildRoom(int);
     virtual void BuildDoor(int, int);
     virtual Maze* GetMaze();
private:
     Direction CommonWall(Room*, Room*);
     Maze* _currentMaze;
};

StandardMazeBuilder::StandardMazeBuilder () {
     _currentMaze = 0;
}

void StandardMazeBuilder::BuildMaze () {
     _currentMaze = new Maze;
}

Maze* StandardMazeBuilder::GetMaze () {
     return _currentMaze;
}
```

```
void StandardMazeBuilder::BuildRoom (int n) {
  if (!_currentMaze->RoomNo(n)) {
    Room* room = new Room(n);
    _currentMaze->AddRoom(room);
    room->SetSide(North, new Wall);
    room->SetSide(South, new Wall);
    room->SetSide(East, new Wall);
    room->SetSide(West, new Wall);
  }
}

void StandardMazeBuilder::BuildDoor (int n1,
int n2) {
  Room* r1 = _currentMaze->RoomNo(n1);
  Room* r2 = _currentMaze->RoomNo(n2);
  Door* d = new Door(r1, r2);
  r1->SetSide(CommonWall(r1,r2), d);
  r2->SetSide(CommonWall(r2,r1), d);
}
```

```
Maze* maze;
MazeGame game;
StandardMazeBuilder builder;
game.CreateMaze(builder);
maze = builder.GetMaze();
```

Builder - příklad

- **CountingMazeBuilder – jiná implementace rozhraní MazeBuilder**

```
class CountingMazeBuilder : public MazeBuilder {
public:
  CountingMazeBuilder();
  virtual void BuildMaze();
  virtual void BuildRoom(int);
  virtual void BuildDoor(int, int);
  virtual void AddWall(int, Direction);
  void GetCounts(int&, int&) const;
private:
  int _doors;
  int _rooms;
};

CountingMazeBuilder::CountingMazeBuilder ()
  { _rooms = _doors = 0; }

void CountingMazeBuilder::BuildRoom (int)
  { _rooms++; }

void CountingMazeBuilder::BuildDoor (int, int)
  { _doors++; }

void CountingMazeBuilder::GetCounts (
  int& rooms, int& doors
) const {
  rooms = _rooms;
  doors = _doors;
}
```

Counting

```
int rooms, doors;
MazeGame game;
CountingMazeBuilder builder;
game.CreateMaze(builder);
builder.GetCounts(rooms, doors);
cout << "The maze has "
     << rooms << " rooms and "
     << doors << " doors" << endl;
```

Builder - související návrhové vzory

- **Související NV**
  - Abstract Factory
    - také často tvoří složité objekty, vytváří je ovšem najednou
    - Builder vytváří objekty postupně a nakonec je vrací
  - Composite
    - je často výsledkem práce Builderu