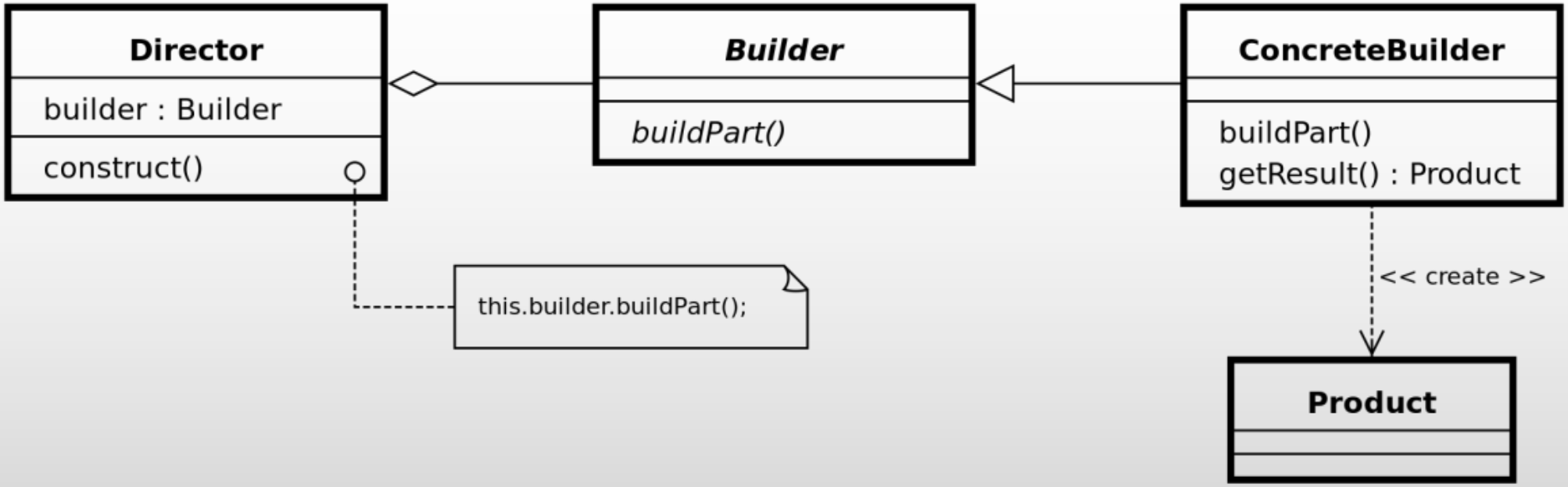

Director
builder : Builder
construct()
Builder
buildPart()
ConcreteBuilder
buildPart()
getResult() : Product
this.builder.buildPart();
<< create >>
Product

BUILDER

HLAVNÍ INFORMACE

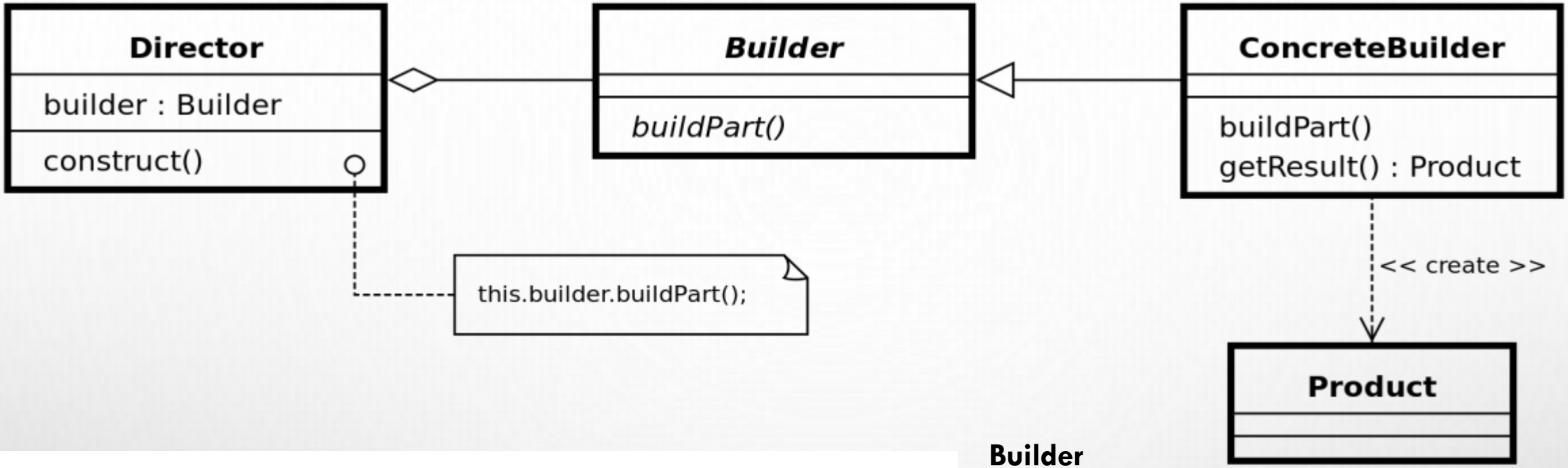
NÁVRHOVÝ VZOR ODDĚLUJÍCÍ KONSTRUKCI SLOŽITÝCH OBJEKTŮ OD JEJICH REPREZENTACE.

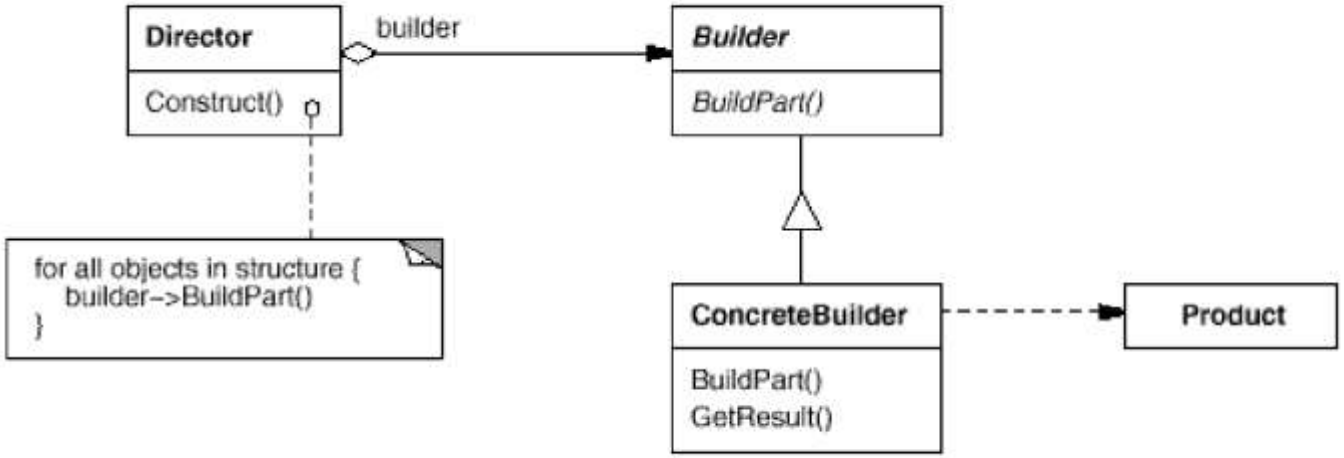
ČÍMŽ JE MOŽNÉ POUŽÍT STEJNÝ PROCES KONSTRUKCE PRO ROZDÍLNÉ REPREZENTACE.

- CO TO ZNAMENÁ?
  - POŽADOVANÝ OBJEKT A JEHO KONSTRUKTOR (BUILDER) JSOU DVĚ ROZDÍLNÉ TŘÍDY.
  - JEDEN BUILDER MŮŽE VYTVÁŘET RŮZNÉ OBJEKTY.

- HLAVNÍ MYŠLENKY
  - VYTVÁŘÍ OBJEKTY PO ČÁSTECH
  - ÚČEL JE ŘÍDIT PROCES, NE ZÍSKAT PRODUKT

- CÍLE
  - VYTVOŘIT RŮZNÉ REPREZENTACE PODOBNÝM POSTUPEM
  - ZJEDNODUŠIT ROZŠIŘITELNOST DO BUDOUCNOSTI
  - PŘEHLEDNÝ ZDROJOVÝ KÓD





Builder

definuje abstraktní rozhraní pro tvorbu produktu

ConcreteBuilder

implementuje rozhraní Builderu

definuje proces tvorby produktu

Director

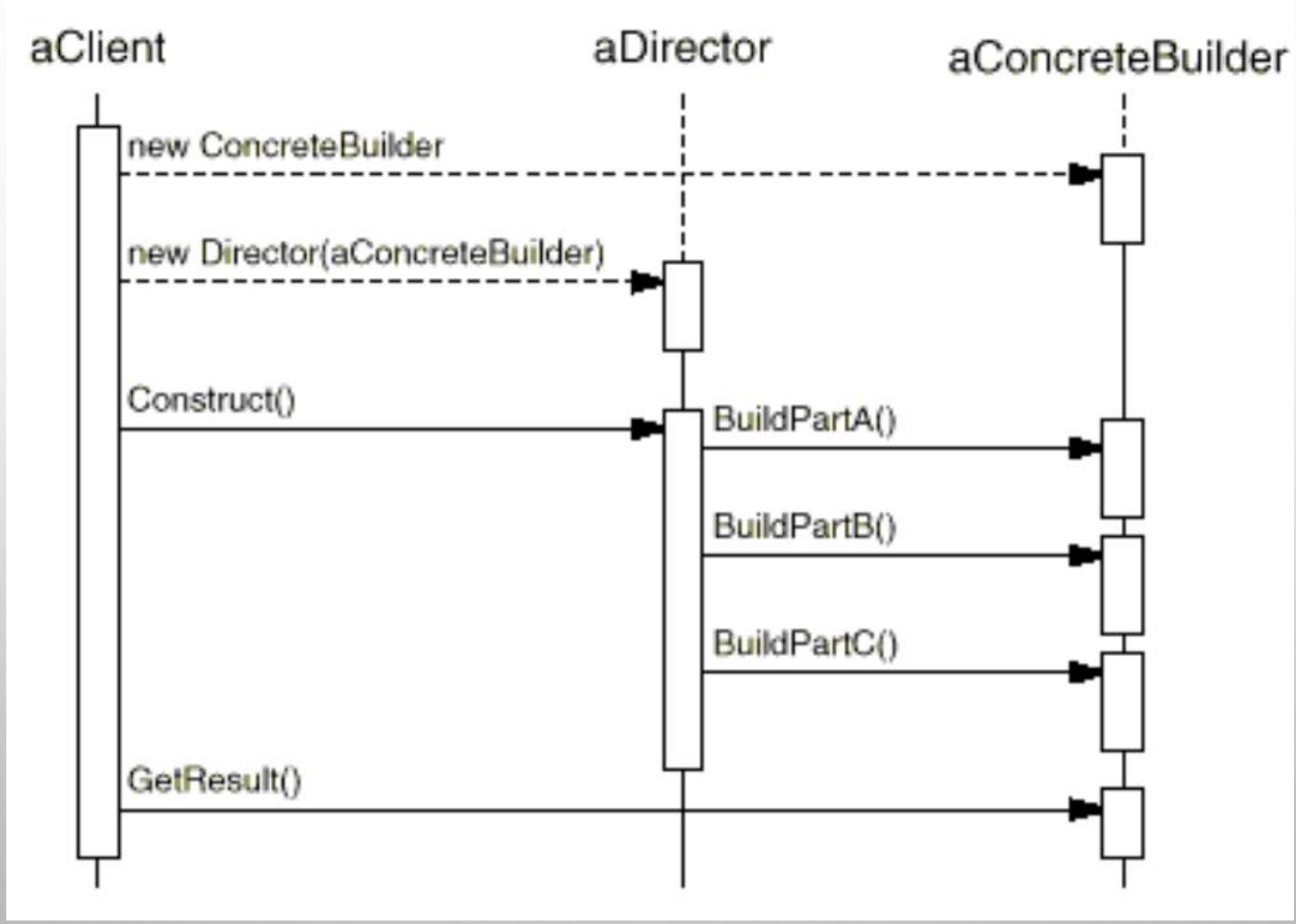
řídí proces pomocí Builderu

po krocích vytvoří objekt

Product

výsledný produkt Builderu

INTERACTION DIAGRAM



IMPLEMENTACE

- BUILDER
  - INTERFACE
  - ABSTRAKTNÍ TŘÍDA
    - PŘES VIRTUÁLNÍ A ABSTRAKTNÍ METODY DEFINUJE, CO MÁ KAŽDÝ CONCRETEBUILDER UMĚT
- PRODUCT
  - NE ABSTRAKTNÍ TŘÍDA
- DIRECTOR
  - ABSTRAKTNÍ TŘÍDA
    - PŘES VIRTUÁLNÍ A ABSTRAKTNÍ METODY DEFINUJE, CO MÁ KAŽDÝ CONCRETEDIRECTOR UMĚT

MOTIVACE

ŘEŠENÍ BEZ BUILDERU

- ILUSTRACE NA PROBLÉMU BANKOVEKO KONTA

```
public class BankAccount
    {
        private string Name;
        private string? Type;
        private string? _username;
        private string _pass;
        public long _id;
        private long balance;

        public BankAccount(...)
        {
            ...
        }

        {
```

- TŘI ZPŮSOBY ŘEŠENÍ:

    1. LEHKÝ KONSTRUKTOR + TĚŽKÝ INICIALIZAČNÍ KÓD
    2. TĚŽKÝ KONSTRUKTOR + LEHKÝ INICIALIZAČNÍ KÓD
    3. VÍCE KONSTRUKTORŮ

MOTIVACE

1. LEHKÝ KONSTRUKTOR + TĚŽKÝ INICIALIZAČNÍ KÓD

- VÝHODY
  - JEDNODUCHÉ
- NEVÝHODY
  - S ROSTOUCÍ SLOŽITOSTÍ ROSTE NEPŘEHLEDNOST POUŽITÍ V KÓDU
  - **VŠECHNO MUSÍ BÝT PUBLIC**
  - POSTRÁDÁ VŠECHNY VÝHODY OOP

```
BankAccount _new = new BankAccount(123456);
_new.Name = "meno";
_new._username = "username";
_new._pass = "password";
...

…

public BankAccount(long id)
        {
            _id = id;
        }
```

MOTIVACE

2. TĚŽKÝ KONSTRUKTOR + LEHKÝ INICIALIZAČNÍ KÓD

- VÝHODY
  - JEDNOŘÁDKOVÁ INICIALIZACE
- NEVÝHODY
  - MOŽNOST SPLÉST SI POŘADÍ ARGUMENTŮ
  - DLOUHÝ KÓD TŘÍDY (HLAVNĚ KONSTRUKTORY)
  - NE VŠECHNY JAZYKY POSKYTUJÍ MOŽNOST DEFAULTNÍCH HODNOT (NAPR. JAVA)

```
BankAccount _new = new BankAccount("meno", "passs", 123456, 9999999, "JK");
…

public BankAccount(string name, string pass, long id, long balance, string? username = null, string? type  = null)
        {
            Name = name;
            Type = type;
            _username = username;
            _pass = pass;
            _id = id;
            this.balance = balance;
        }
```

MOTIVACE

3. VÍCE KONSTRUKTORŮ

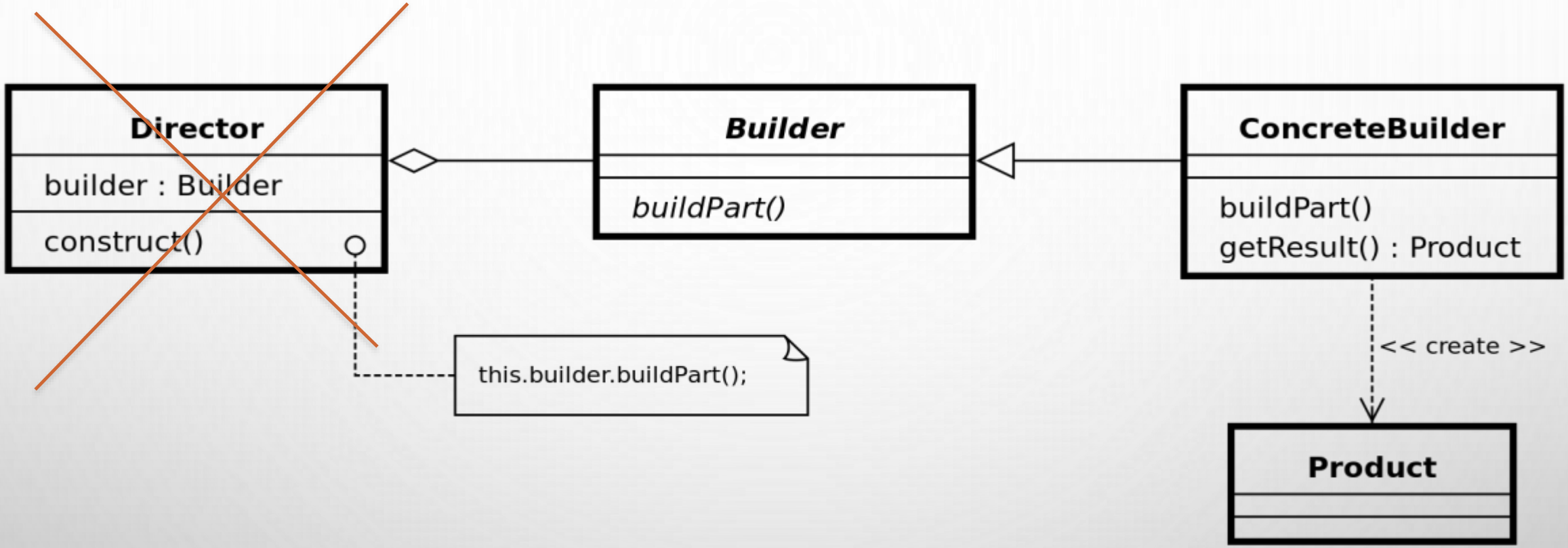
- NEVÝHODY
  - VÍCE KONSTRUKTORŮ
  - NAPSAT VŠECHNY KOMBINACE -> KATASTROFA

```
public BankAccount(string name, string pass, long id, long balance)
…
public BankAccount(string name, string pass, long id, long balance, string username)
…
public BankAccount(string name, string pass, long id, long balance, string type)
…
public BankAccount(string name, string pass, long id, long balance, string username, string type)
…
```

PROČ BUILDER

- BUILDER UMÍ ELEGANTNĚ VYŘEŠIT SLOŽITÉ VNOŘENÉ OBJEKTY, KTERÉ JE TAKÉ NUTNO KONSTRUOVAT
- BUILDER SKRYJE VNITŘNÍ FUNKCIONALITU
- POUZE VARIANTA S MNOHA KONSTRUKTORY (PŘOSTŘEDNÍ) PODPORUJE IMMUTABILITU
- BUILDER PODPORUJE DEFAULTNÍ HODNOTY
  - NĚKTERÉ JAZYKY NEMUSÍ PODPOROVAT VLOŽENÍ KOMPLIKOVANĚJŠÍCH DEFAULTNÍCH HODNOT DO KONSTRUKTORU
- BUILDER ZAVÁDÍ ČITELNOST
  - FUNKČNOST KÓDU PLYNE Z NÁZVŮ POUŽITÝCH METOD MÍSTO POŘADÍ V KONSTRUKTORU

RŘEŠENÍ S BUILDEREM (DIRECTORLESS)



- UČINÍME METODY BUILDERU PUBLIC
- LIGHTWEIGHT VARIANTA POKUD NÁM JDE VÍCE O STAVBU PRODUKTU

RŘEŠENÍ S BUILDEREM (DIRECTORLESS)

```
BankAccount _new = new BankAccount().WithId(123456)
                   .WithBalance(9999999)
                   .WithType("premium").WithName("meno")
                   .WithPass("passs").Build();
```

```
public BankAccount WithId(long id)
{
    _id = id;
    return this;
}
public BankAccount WithName(string Name)
{
    _Name= Name;
    return this;
}
public BankAccount WithType(string Type)
{
    _Type = Type;
    return this;
}
public BankAccount WithPass(string pass)
{
    _pass = pass;
    return this;
}
public BankAccount WithBalance(long balance)
{
    _balance = balance;
    return this;
}
public BankAccountBuilder Build()
{
    return this;
}
```

RŘEŠENÍ S BUILDEREM (DIRECTORLESS)

```
BankAccount _new = new BankAccountBuilder().WithId(123456)
                   .WithBalance(9999999)
                   .WithType("premium").WithName("meno")
                   .WithPass("passs").Build();
```

NEVÝHODA:

- POKUD CHCEME ABY NÁM TO VŠECHNO FUNGOVALO MUSÍME SI HRÁT S PŘÍSTUPNOUSTÍ NEBO UDĚLAT VŠECHNO PUBLIC

```
class BankAccountBuilder
    {
        BankAccount _newBA;
        public BankAccountBuilder WithId(long id)
        {
            _newBA._id = id;
            return this;
        }
        public BankAccountBuilder WithName(string Name)
        {
            _newBA._Name = Name;
            return this;
        }
        public BankAccountBuilder WithType(string Type)
        {
            _newBA._Type = Type;
            return this;
        }
        public BankAccountBuilder WithPass(string pass)
        {
            _newBA._pass = pass;
            return this;
        }
        public BankAccountBuilder WithBalance(long balance)
        {
            _newBA._balance = balance;
            return this;
        }
        public BankAccount Build()
        {
            return _newBA;
        }
    }
```

RŘEŠENÍ S BUILDEREM S DIRECTOREM

- JAKO MEZIKROK MEZI BUILDERY A KLIENTEM MŮŽU IMPLEMENTOVAT DIRECTOR
- DIRECTOR UMOŽNÍ IMPLEMENTOVAT PŘEDVOLBY
- JE TAKÉ MOŽNÉ POUŽÍVAT JEDEN DIRECTOR NA SPRÁVU VÍCE BUILDERŮ

```
abstract class AbDirector
    {
        public BankAccountBuilder _newBAB;
        public AbDirector(BankAccountBuilder newBAB)
        {
            _newBAB = newBAB;
        }
        public BankAccount GetBankAccount() { return _newBAB.Build();
    }


class Director : AbDirector
    {
        public Director(BankAccountBuilder newBAB) : base(newBAB)
        {
            _newBAB = newBAB;
        }
        public void Skeleton_premium()
        {
            _newBAB = new BankAccountBuilder()
                .WithBalance(10000)
                .WithType("premium")
                .WithPass("defpass");
        }
        public void Skeleton_default()
        {
            _newBAB = new BankAccountBuilder()
                .WithBalance(1000)
                .WithType("default")
                .WithPass("defpass");
        }
    }
```

VOLANÍ

POSTUPNÝ

```
BankAccountBuilder _new = new BankAccountBuilder();
_new.WithId(123456);
_new.WithBalance(9999999);
_new.WithType("premium");
_new.WithName("meno");
_new.WithPass("passs");
BankAccount _newBA = _new.Build();
```

- JEDEN BUILDER JE DOPLŇOVÁN NOVÝMI ARGUMENTY
- VZHLEDOVĚ SE MOC NELIŠÍ OD POUŽITÍ SETERŮ
  - FUNKČNĚ SE ALE CHOVÁ JAKO BUILDER
- VE VELKÝCH MNOŽSTVÍCH NEPŘEHLADNÍ

FLUENT

```
BankAccount _new = new
BankAccountBuilder().WithId(123456)
                    .WithBalance(9999999).WithType("premium")
                    .WithName("meno").WithPass("passs").Build();
```

- NA JEDEN ŘÁDEK

IMMUTABILITA

- KDYBYCHOM CHTĚLI ZAVÉST IMMUTABILITU, TAK BY KÓD VYPADAL TAKTO
  1. **CONSTRUCT BUILDER** – VYTVOŘÍ BUILDER A PŘIDÁ POVINNÉ PARAMETRY
  2. **ADD PARAMETERS** – PŘIDÁ OSTATNÍ VOLITELNÉ PARAMETRY
  3. **BUILD** – SESTAVÍ PRODUKT (NASTAVÍ DEFAULTNÍ HODNOTY + ZAVOLÁNÍ KONSTRUKTORU PRODUKTU)
  4. **RETURN PRODUCT** – VRÁTÍ HOTOVÝ PRODUKT

IN OBJECT-ORIENTED AND FUNCTIONAL PROGRAMMING, AN **IMMUTABLE OBJECT** IS AN OBJECT WHOSE STATE CANNOT BE MODIFIED AFTER IT IS CREATED.

~~

readonly

+/-

+ DEFAULTNÍ HODNOTY PRO NEPOVINNÉ PARAMETRY
+ ROZŠIŘITELNOST O NOVÉ POLOŽKY
+ ZAPOUZDŘENÍ
+ ZBAVENÍ SE KONSTRUKTORŮ
+ IMUTABILITA - BEZPEČNÉ PRO VÍCE VLÁKEN
+ FLUENT INTERFACE

- KÓD JE REPETETIVNÍ A OBČAS ZBYTEČNĚ DLOUHÝ
- V ZÁVISLOSTI NA IMPLEMENTACI IMUTABILITY DOCHÁZÍ K URČITÉMU MNOŽSTÍ KOPÍROVÁNÍ
- PRO KAŽDÝ PRODUCT MUSÍME VYTVOŘIT ODLIŠNÝ CONCRETEBUILDER
- BUILDER TŘÍDY MUSÍ BÝT MUTABLE

KDY POUŽÍT BUILDER PATTERN

- VELKÁ ČÁST PARAMETRŮ JE VOLITELNÁ
- VÍCE KONSTRUKTORŮ
- RULE OF THUMB JE POUŽÍT BUILDER PŘI 4+ PARAMETRECH
- CÍLOVÝ OBJEKT SE SKLÁDÁ Z MNOHA SLOŽITÝCH PODOBJEKTŮ
- POTŘEBUJEME ROZŠIŘITELNOST OBJEKTU
- POTŘEBUJEME IMMUTABILITU

> The ideal number of arguments for a function is zero (niladic). Next comes one (monadic), followed closely by two (dyadic). Three arguments (triadic) should be avoided where possible. More than three (polyadic) requires very special justification - and then shouldn't be used anyway.
>
> -Robert Martin

REAL WORLD POUŽITÍ

- LARAVEL QUERY BUILDER

```
$user = User::query()
->where('is_active', 1)
->orderBy('updated_at', 'desc')
->first();
```

- PATTERN BUILDER SE POUŽÍVÁ V JAVAX.JSON.JSON A JAVAX.JSON.JSONBUILDER CLASSES PŘI BUDOVÁNÍ JSON OBJEKTŮ

```
JsonObjectBuilder b = Json.createObjectBuilder()
        . add( "report", Json.createObjectBuilder()
                . add( "reportId", reportId )
                . add( "title", title )
                . add( "subtitle", subTitle == null ? "" :
        subTitle )
                . add( "created", created.toString() )
                . add( "description", description == null ? "" :
        description )
                . add( "data", report )
                    );
return b.build();
```

- UNIT TESTS

```
Given(db => /* Setup here */)
.When().Query<SearchQuery>().WithDate(_date)
.And(result => bookingKey = result.First().BookingKey).Command<ReserveCommand>().WithBookingKey(bookingKey)
.And().Command<ConfirmCommand>().WithBookingKey(bookingKey)
.And().Command<CancelCommand>().WithBookingKey(bookingKey)
.Repeat()
.Twice()

.Then(db => db.Set<Booking>().SingleOrDefaultAsync(b => b.BookingKey == bookingKey));
```

SOUVISEJÍCÍ VZORY

ABSTRACT FACTORY

- MŮŽE TAKÉ KONSTRUOVAT SLOŽITÉ OBJEKTY
- ROZDÍL JE, ŽE BUILDER STAVÍ KROK PO KROKU A NA KONCI VRÁTÍ PRODUKT,
- ABSTRACT FACTORY KLADE DORAZ NA "RODINY" OBJEKTŮ A VRÁTÍ PRODUKT IHNED

SINGLETON

- BUILDER UMÍME IMPLEMENTOVAT JAKO SINGLETON

COMPOSITE

- BUILDER ČASTO VYTVÁŘÍ OBJEKT PODLE VZORU COMPOSITE

ZÁVĚR

- BUILDER ODDĚLUJE KONSTRUKCI OD REPREZENTACE
- BUILDER HRAJE ROLI KONSTRUKTORU
- BUILDER UMOŽŇUJE PRODUKTŮM BÝT IMMUTABLE
- BUILDER KROK PO KROKU DEFINUJE PRODUKT
- DIRECTOR OVLÁDÁ VÍCE BUILDERŮ A UMOŽŇUJE IMPLEMENTOVAT PŘEDVOLBY