Builder

Motivácia

- majme reprezentáciu používateľa/user
- o každom užívateľovi musíme mat aspoň informáciu o jeho mene
- ostatné údaje nemusia byť uvedené

```
public class User {
    public String name;
    public Gender gender;
    public Integer age;
```

Ne světlé písmo na tmavém pozadí !!!

Kód ne jako obrázky, ale jako text !

```
public enum Gender {
    Male,
    Female,
    Other
}
```

```
ess {
    street;
    public int streetNumber;
    public String city;
    public String zipCode;
}
```

Motivácia

- jednoduchý konštruktor

```
public User(String name){
    this.name = name;
}
```

Nevýhody

- atribúty musia byť public
- neprehľadnosť použitia kódu

```
var user = new User( name: "Alice");
user.age = 22;
user.phone = "+420 123 456 789";
user.gender = Gender.Female;
user.address = null;
```

Motivácia

- zložitý konštruktor

```
public User(String name,Gender gender,Integer age,String phone,Address address){
    this.name = name;
    this.gender = gender;
    this.age = age;
    this.phone = phone;
    this.address = address;
}
```

Nevýhody

- v poradí argumentov sa vieme pomýliť
- musíme vždy všetko uviesť
- nepovinné argumenty ako null, hodnota

```
var user = new User(
        name: "Alice",
        Gender.Female,
        age: 22,
        phone: "+420 123 456 789",
        new Address( street: "Narodni")
);
```

Motivácia

- preťaženie konštruktora

Nevýhody

- viacej konštruktorov
- napísať všetky kombinácie -> katastrofa

Motivácia

- defaultné nastavenie parametrov

Nevýhody

- nie každý jazyk podporuje defaultné hodnoty
- napr. Java nie :(

Builder

- vytvárajúci návrhový vzor
- Gang of Four
- flexibilné riešenie rôznych "vytvárajúcich" problémov

Hlavný úmysel

- **oddelenie konštrukcie od reprezentácie zložitého objektu**
- **pre rozdielne reprezentácie je možné použiť rovnaký proces konštrukcie**

Inými slovami

- dve rozdielne triedy, objekt a jeho konštruktor (builder)
- jeden builder dokáže vytvárať rôzne objekty

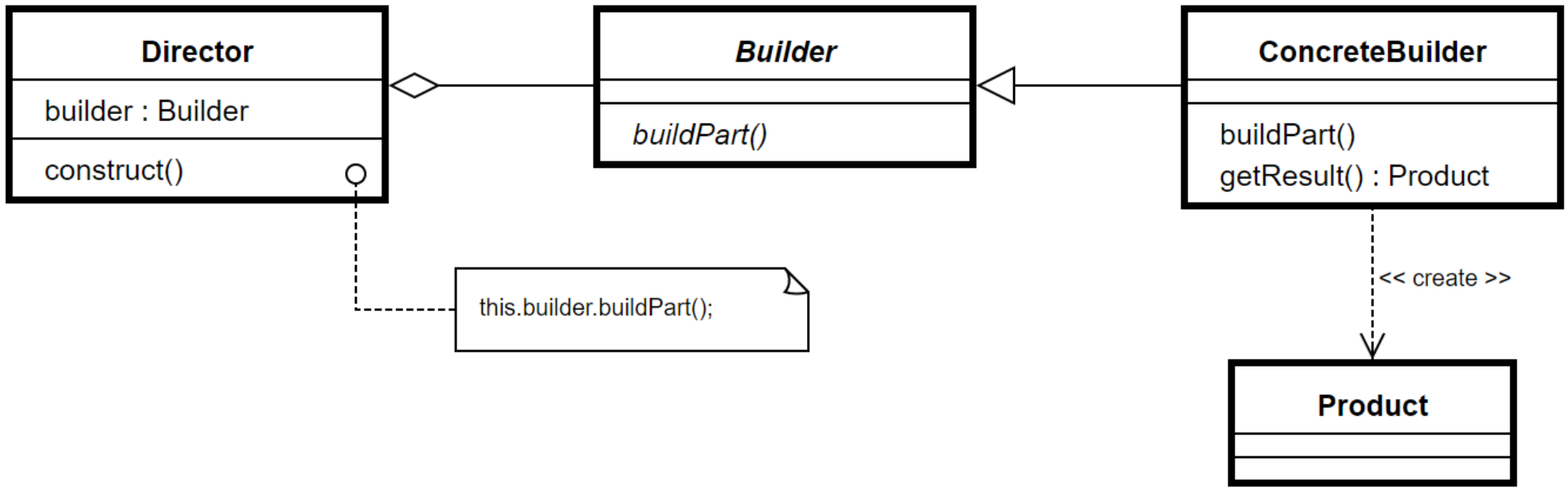
Idea, kedy a prečo?

- objekt vytvárať postupne po častiach
- zjednodušenie konštruovania objektu, napr. vnútorné objekty
- skryť vnútornú funkcionalitu konštruovania
- tvorenie rôznych reprezentácií podobným postupom
- podpora defaultnej hodnoty, parametre môžu byt voliteľné
- rozšíriteľnosť v budúcnosti
- prehľadnejší zdrojový kód/čitateľnosť
- immutabilita
- viac ako 4 parametre -> builder

Použiteľnosť

- proces vytvárania zložitého objektu je nezávislý od samotného objektu
- musíme dovoliť rôzne reprezentácie konštruovaného objektu

Štruktúra


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

Účastníci

Builder

- definuje abstraktné rozhranie pre tvorbu produktu

ConcreteBuilder

- implementuje Builder
- definuje proces tvorby produktu
- poskytuje rozhranie pre získanie produktu

Director

- skonštruuje objekt pomocou Builder rozhrania

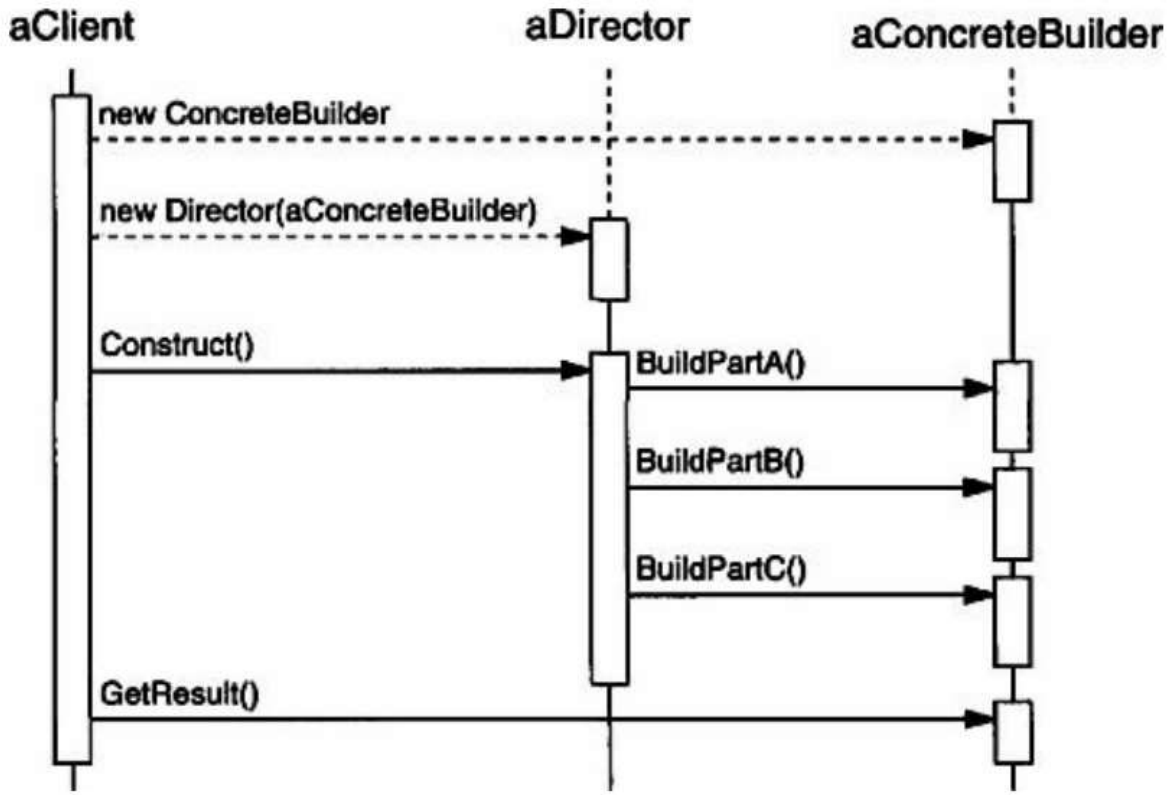
Product

- reprezentuje zložitý objekt, ktorý konštruujeme

Spolupráca pri behu

- klient vytvorí Director a nakonfiguruje ho požadovanou inštanciou Builder
- klient požiada Director o produkt
- Director upozorní Builder, ak sa má postaviť časť produktu
- Builder spracuje požiadavky a pridá časti do produktu
- klient dostane produkt od Builder

Interakčný diagram



Dôsledky návrhového vzoru

- vieme meniť internú reprezentáciu produktu
- oddelí kód pre konštrukciu a reprezentáciu
- lepšia kontrola nad procesom konštruovania

Implementácia

Builder

- interface/abstraktná trieda ako "predloha" pre ConcreteBuilder
- metódy build a get

Director

- abstraktná trieda, definujeme vlastnosti, ktoré má každý ConcreteDirector vedieť

Product

- nie je abstraktná trieda, produkty vytvorené konkrétnymi builder-mi sú veľmi odlišné v reprezentácii

Zjednodušená implementácia (Directorless Builder)

- niekedy vynecháme Director, ale Builder metódy musia byt public
- postupný variant

```
var userBuilder = new UserBuilder();
userBuilder.setName("Alice");
userBuilder.setGender(Gender.Female);
userBuilder.setAge(22);
userBuilder.setPhone( "+420 123 456 789");
userBuilder.setAdress(null);

var user :User = userBuilder.getUser();
```

- fluent variant

```
var user :User = new UserBuilder()
        .setName("Alice")
        .setGender(Gender.Female)
        .setAge(22)
        .setPhone( "+420 123 456 789")
        .setAdress(null)
        .getUser();
```

```
public class UserBuilder {
    private User user;

    public UserBuilder setName(String name){
        user.name = name;
        return this;
    }
    public UserBuilder setGender(Gender gender){
        user.gender = gender;
        return this;
    }
    public UserBuilder setAge(Integer age){
        user.age = age;
        return this;
    }
    public UserBuilder setPhone(String phone){
        user.phone = phone;
        return this;
    }
    public UserBuilder setAdress(Address adress){
        user.address = adress;
        return this;
    }
    public User getUser() {
        return user;
    }
}
```

Implementácia

- medzi Builder a klientom zavedieme medzikrok – Director (implementácia predvolieb)

```
abstract public class AbstractDirector {
    protected UserBuilder userBuilder;
    public AbstractDirector(UserBuilder builder){
        userBuilder = builder;
    }
    public User GetUser(){
        return userBuilder.getUser();
    }
}
```

```
public class Director extends AbstractDirector{
    public Director(UserBuilder builder) {
        super(builder);
    }
    public void AliceUser(){
        userBuilder = new UserBuilder()
                .setName("Alice")
                .setGender(Gender.Female)
                .setAge(22)
                .setPhone( "+420 123 456 789")
                .setAdress(null);
    }
    public void BobUser(){
        userBuilder = new UserBuilder()
                .setName("Bob")
                .setGender(Gender.Male)
                .setAge(23)
                .setPhone( "+420 987 654 321")
                .setAdress(null);
    }

}
```

Výhody

- dovolí odlišnosť vnútornej reprezentácie produktu
- zapuzdrí kód pre konštrukciu a reprezentáciu
- ľahko rozšíriteľný
- immutabilita
- defaultné hodnoty parametru
- kontrola nad konštrukciou produktu
- fluent variant

Nevýhody

- pre každý Product musíme vytvoriť odlišný ConcreteBuilder
- niekedy je kód repetetívny
- Builder triedy musia byť mutable

Súvisiace vzory

Abstract Factory

- môže tiež konštruovať zložité objekty
- rozdiel je, že Builder stavia krok po kroku a na konci vráti produkt,

  Abstract Factory kladie doraz na "rodiny" objektov a vráti produkt ihneď

Composite

- často Builder vytvára objekt podľa vzoru Composite

Singleton

- Builder vieme implementovať ako singleton

Záver

- Builder nám oddelí reprezentáciu objektu od konštrukcie
- plní úlohu konštruktoru
- definuje krok po kroku produkt
- produkt vie byť immutable
- Director ovláda viac Builder-ov a umožňuje predvoľby