Návrhové vzory

OMO, LS 2013/2014

Motivace

- Cílem objektového návrhu je strukturu aplikace navrhnout tak, aby splňovala následující kritéria:
  - snadná rozšiřitelnost
  - účelnost
  - testovatelnost
  - dokumentovatelnost

Koncepty OOP

- Objekty
- Abstrakce
- Zapouzdření
- Skládání
- Delegování
- Dědičnost
- Polymorfismus

Objekty

- Jedná se o prvky modelované reality, které v sobě seskupují jak funkčnost (metody) tak data (proměnné).
- Objekty si pamatují svůj stav v podobě proměnných a poskytují rozhraní pro komunikaci s nimi.
- Většina jazyků rozlišuje třídy a instance tříd.

Abstrakce

- Pomocí abstrakce programátor může abstrahovat od některých detailů jednotlivých objektů (abstraktní metody, ...).
- Každý objekt pracuje navenek jako černá skříňka.

Zapouzdření

- Každý objekt navenek zpřístupňuje rozhraní, pomocí kterého se s objektem pracuje.
- Je vhodné, aby každá metoda tohoto rozhraní, po jejím provedení zanechala objekt v konzistentním stavu.

Skládání

- Objekt může obsahovat jiné objekty.

Delegování

- Objekt může využívat služeb jiných objektů tak, že je požádá o provedení operace.

Dědičnost

- Objekty jsou obvykle organizovány stromovým způsobem, kdy objekty nějakého druhu mohou *dědit* z jiného druhu objektů, čímž přebírají jejich schopnosti.
- K těmto schopnostem mohou zděděné objekty přidávat svá vlastní rozšíření.

Dědičnost

- Vztah mezi třídami:
  - členské proměnné a metody definované v předku mohou být použity i potomkem,
  - potomek může přidat nové členské proměnné a metody,
  - potomek může změnit definice metod předka.
- Vždy musí platit vztah is-a.
  - Jedná se o specializaci nadtypu.

Dědičnost

- Při dědění by měl být vždy dodržen substituční princip LSP:

Let q(x) be a property provable about objects x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.

Polymorfismus

- Situace, kde různé objekty:
  - rozumí stejné zprávě (mají metodu se stejnou signaturou),
  - ale na zprávu reagují různě (vyvoláním jiného kódu).
- Aby měl polymorfismus praktický význam, musí mít provedený kód stejný význam (v kontextu daného objektu).

Duplikace kódu a dat

Příklad: 10000 různých způsobů validace amerického Social Security Number v US vládních systémech

– Příčiny?

– Předejití?

Návrhové vzory

- místní architektura
- poskytují řešení opakujících se problémů
- reálné použití vzoru se může v praxi lišit
- Bible návrhových vzorů:

Design Patterns: Elements of Reusable Object-Oriented Software

Návrhové vzory

- Vzory pro tvorbu instancí
- Strukturální vzory
- Vzory chování

Vzory pro tvorbu instancí

- Factory Method
- Abstract Factory
- Prototype
- Builder
- Singleton
- ....

Strukturální vzory

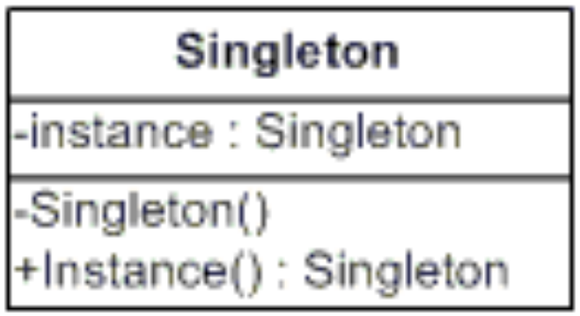
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy
- ....

Vzory chování

- Interpreter
- Template Method
- Chain of Responsibility
- Command
- Iterator
- Mediator
- Observer
- State
- Strategy
- Visitor
- ...

Singleton

Aneb: Jediná instance v systému



Singleton - Vlastnosti

- Použije se v případě, že chceme zajistit od dané třídy existenci pouze jediné instance.
- V minulosti byl používán velmi často.
- Může způsobit problémy při rozšiřování programu (produktu).

Singleton – Příklad

```
public class SingletonDemo {
   private static volatile SingletonDemo instance = null;

   private SingletonDemo() { }

   public static SingletonDemo getInstance() {
      if (instance == null) {
          synchronized (SingletonDemo.class){
              if (instance == null) {
                instance = new SingletonDemo ();
              }
          }
      }
      return instance;
   }
}
```

Singleton - Příklad

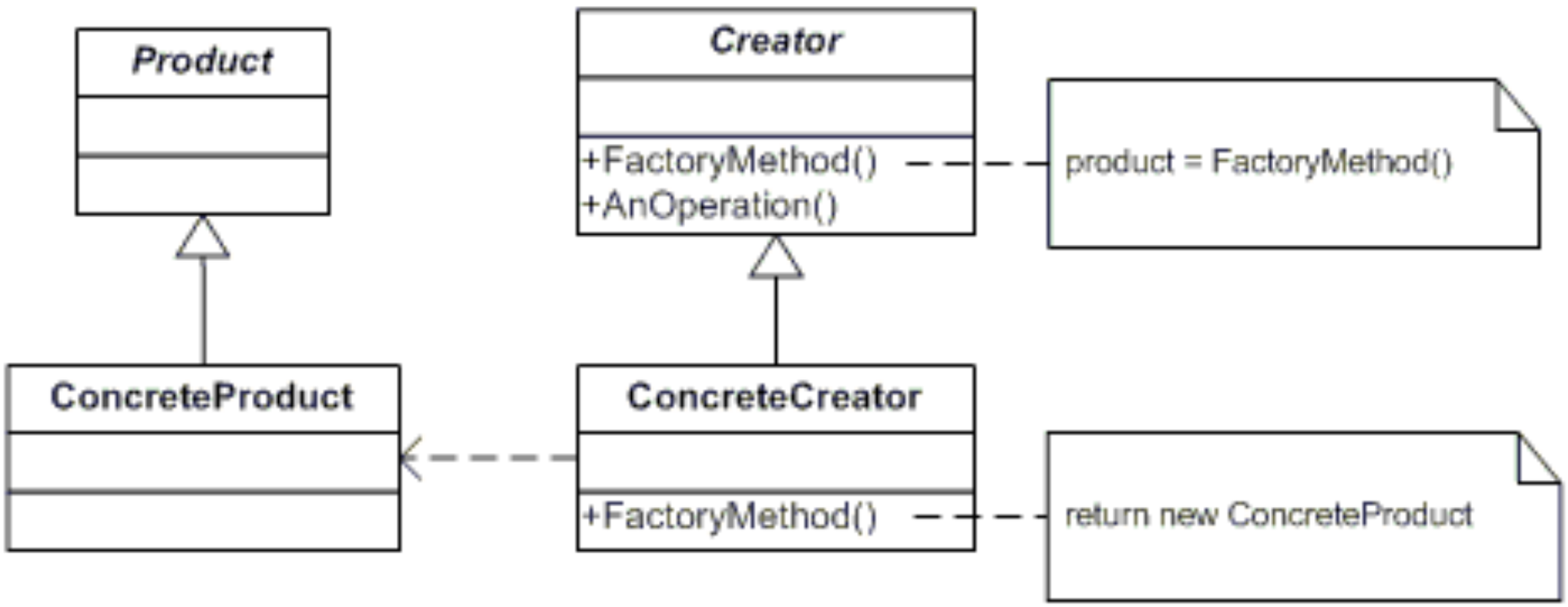
```
public class SingletonDemo {
   private static SingletonDemo instance = null;

   private SingletonDemo() { }

   public static synchronized SingletonDemo getInstance() {
      if (instance == null) {
         instance = new SingletonDemo ();
      }
      return instance;
   }
}
```

Singleton - Příklad

```
public enum Singleton {
   INSTANCE;
          public void execute (String arg) {
                   // perform operation here
          }
}
```

Factory Method

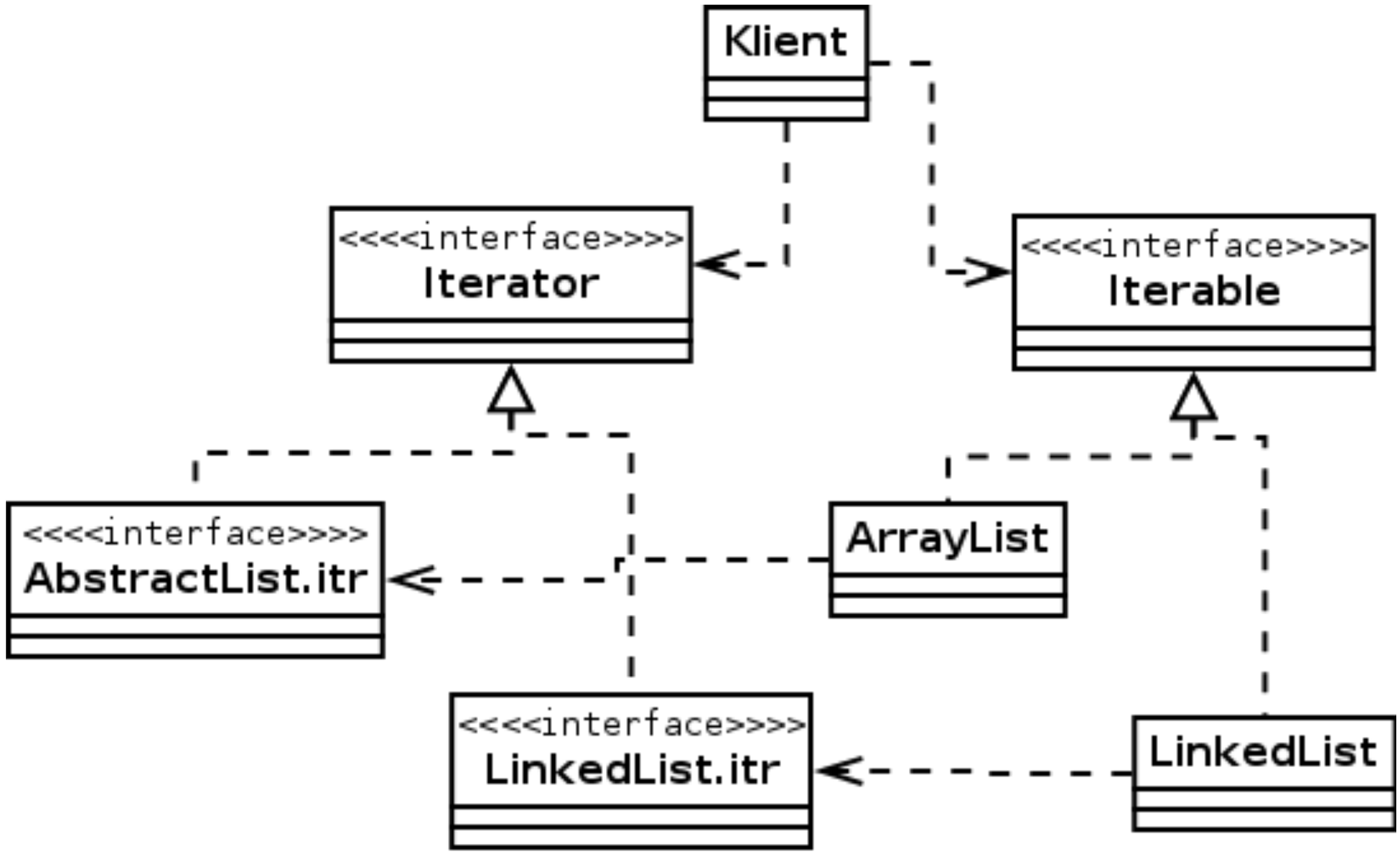
Aneb: Střihni mi to na míru



Factory Method - Vlastnosti

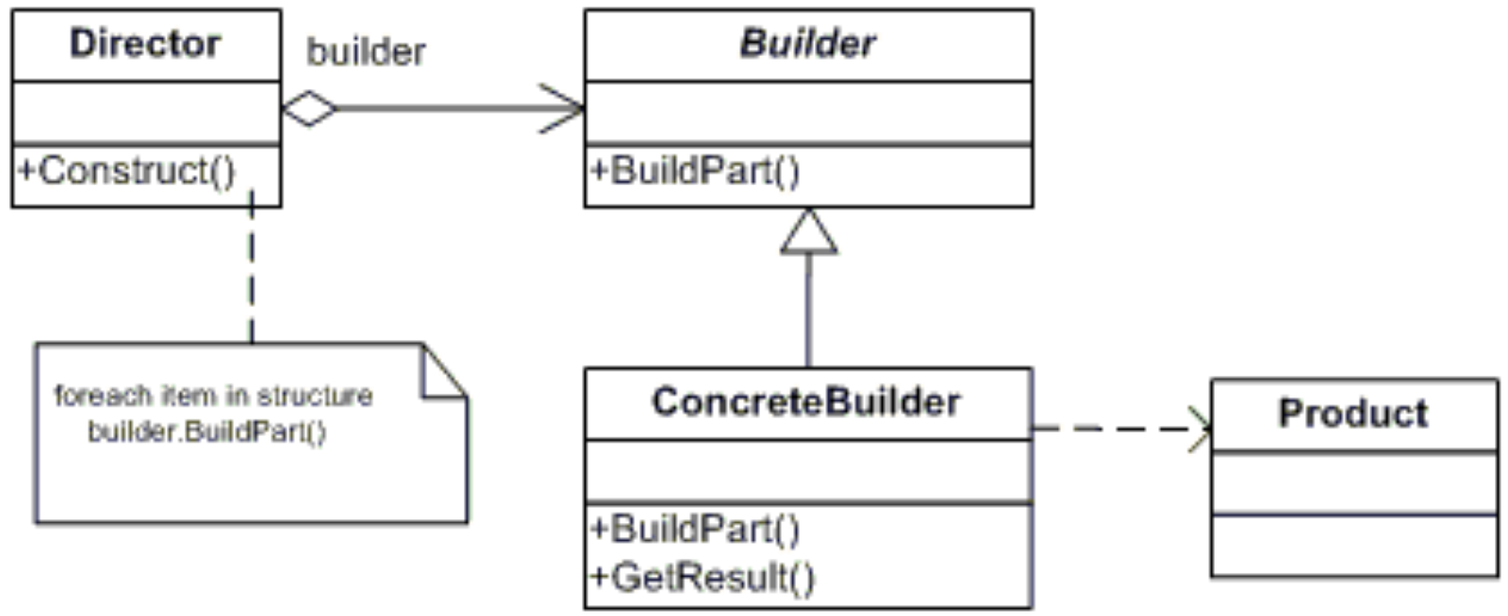
- Definuje rozhraní pro vytváření nových instancí.
- Podtřída rozhroduje o tom, která třída bude instanciována.
- Tovární metoda tedy přenechává vytváření instancí podtřídám.

Factory Method - Příklad



Builder

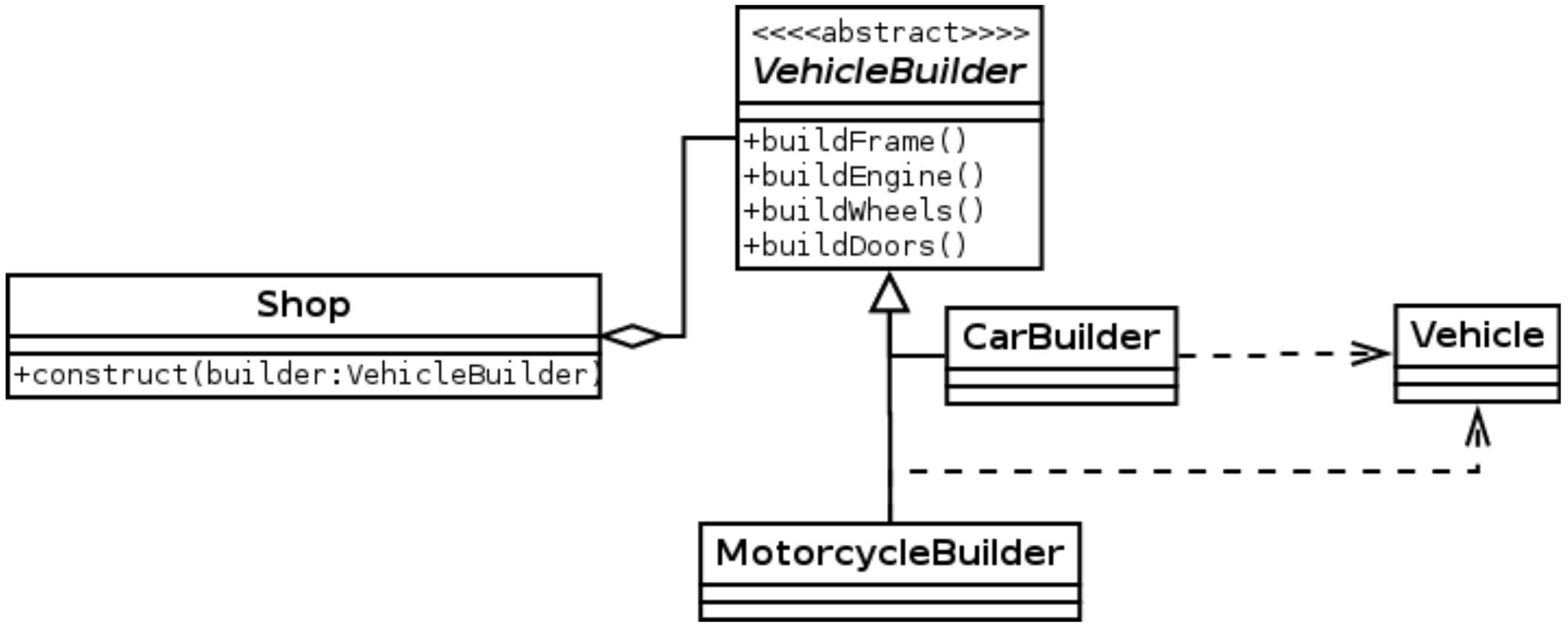
Aneb: Dosazujeme do vzorečku



Builder - Vlastnosti

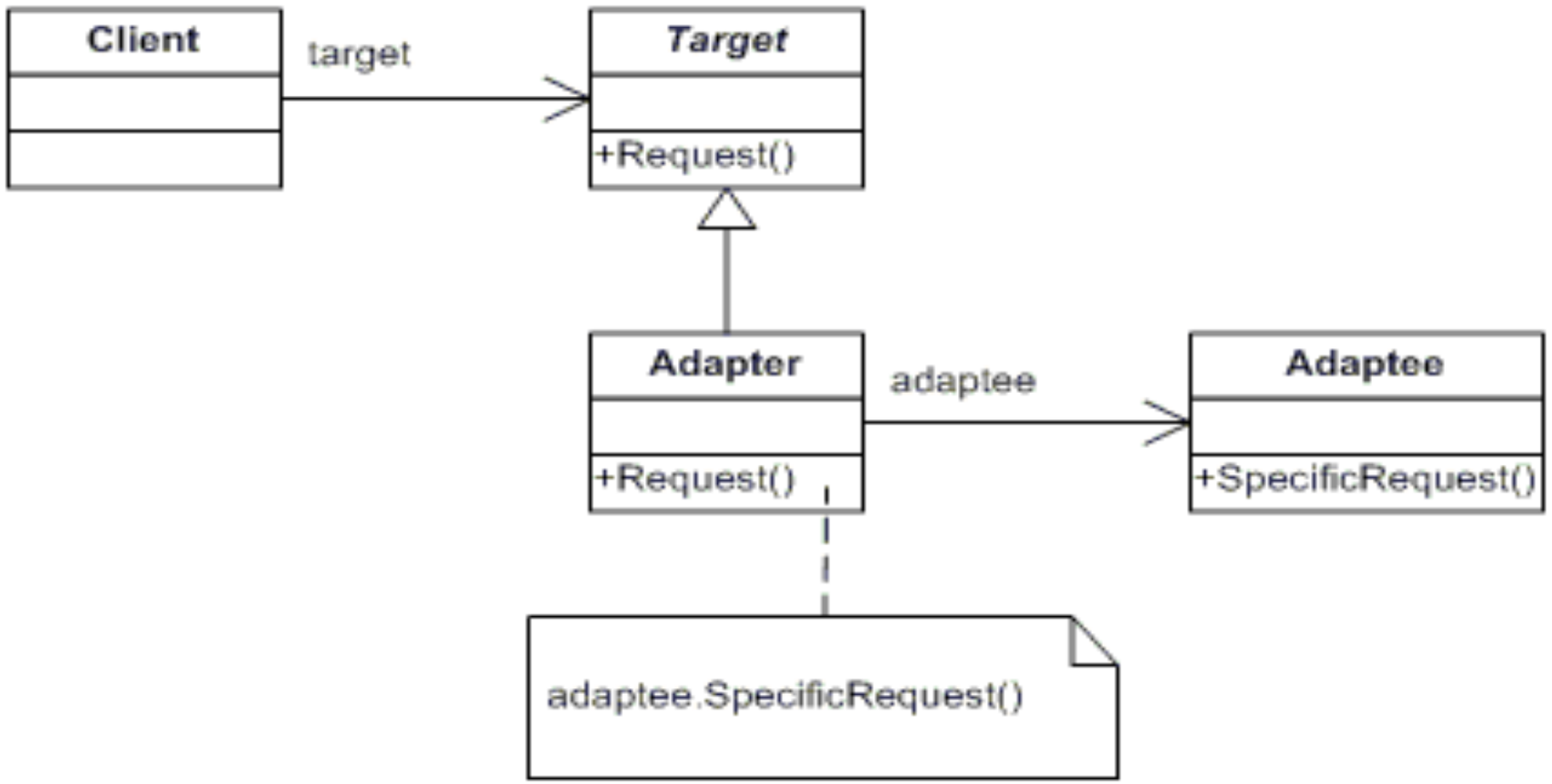
- Odděluje proces tvorby složitého objektu od jeho vnitřní reprezentace.
- Například auto se vždy bude skládat z kol, karoserie, sedaček ....., ale každé auto má jiné komponenty.

Builder - Příklad



Adapter

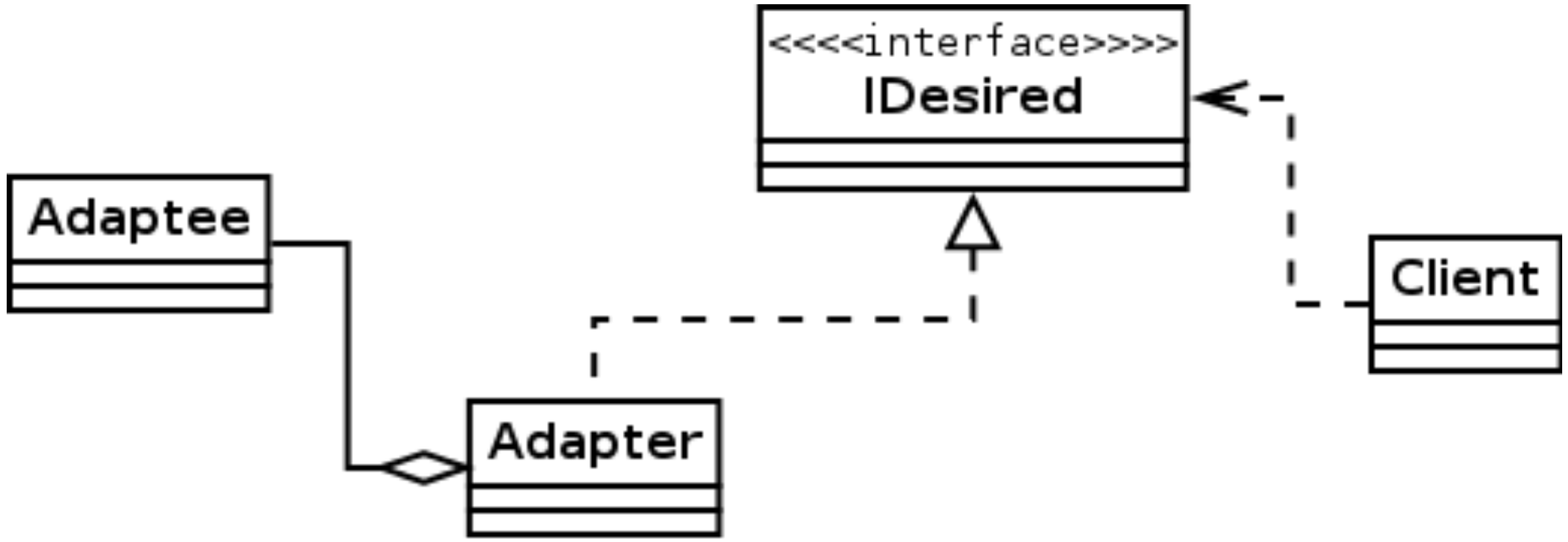
Aneb: Je to trochu jinak



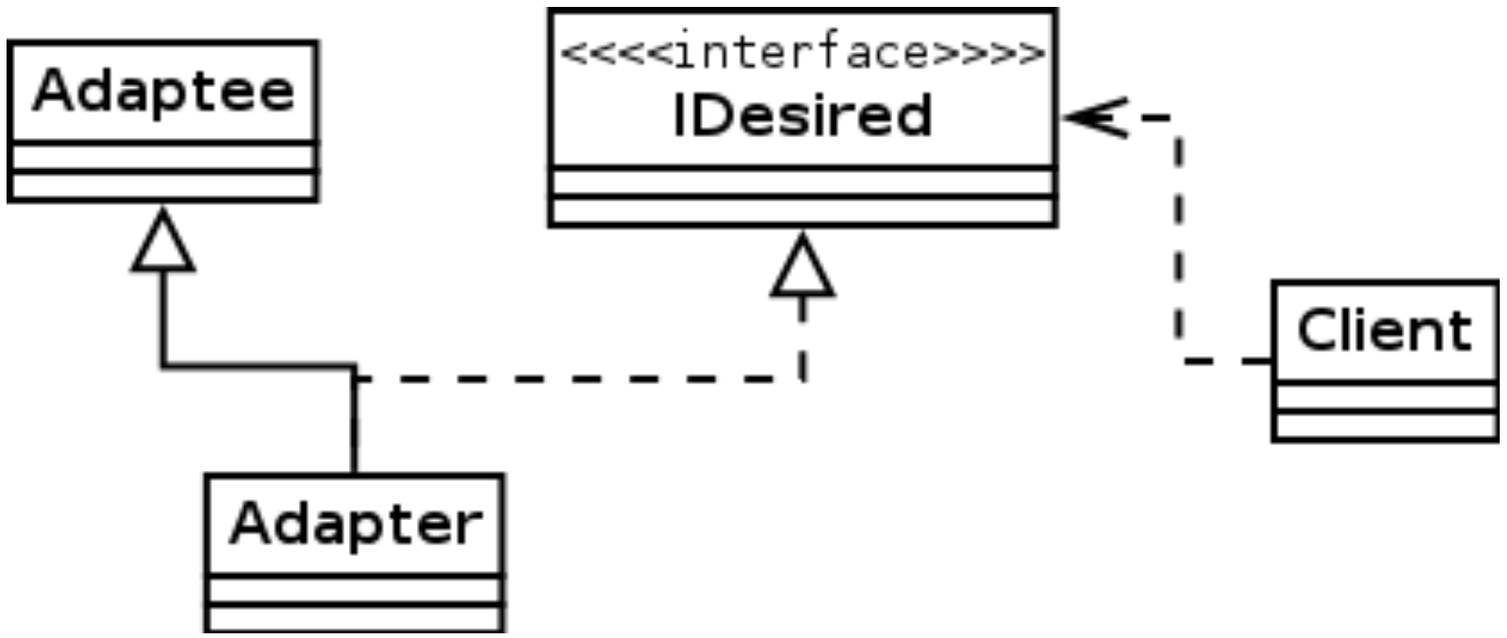
Adapter - Vlastnosti

- Slouží ke konverzi rozhraní jedné třídy na jinou.
- K použití je vhodný v případě, že rozhraní tříd se liší jen velice málo, například názvy metod.

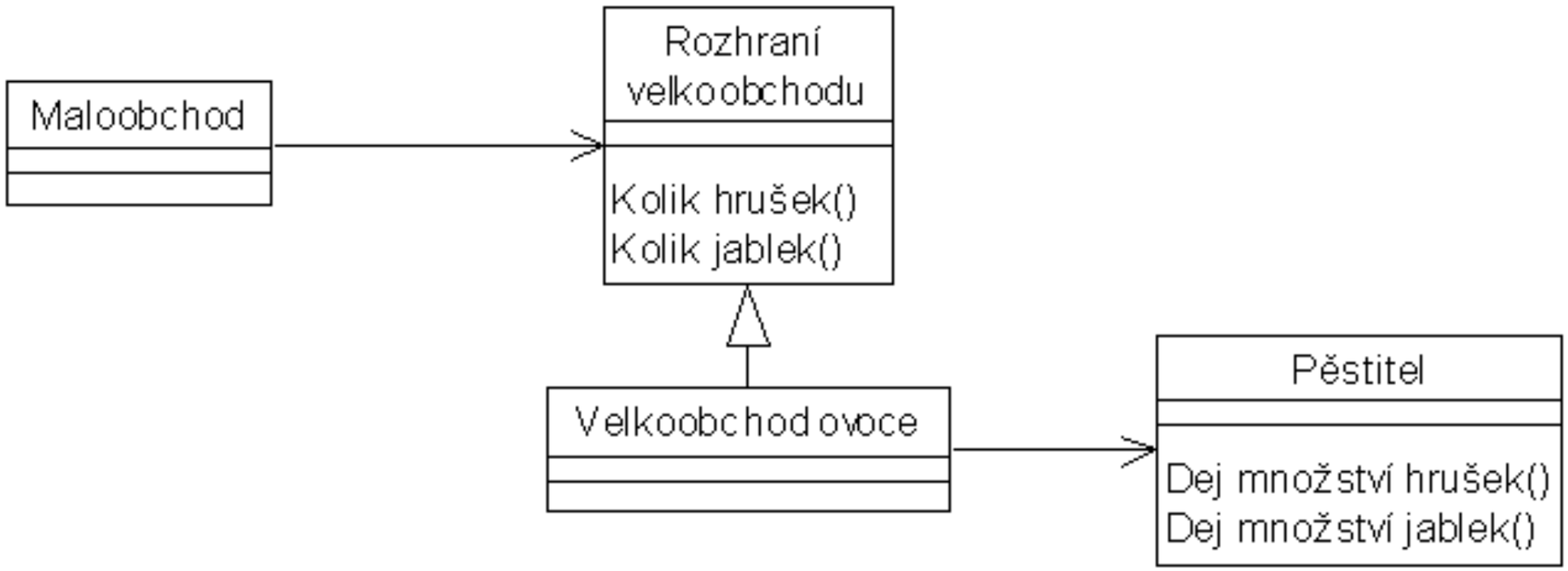
Adapter – Příklad 1



Adapter – Příklad 2



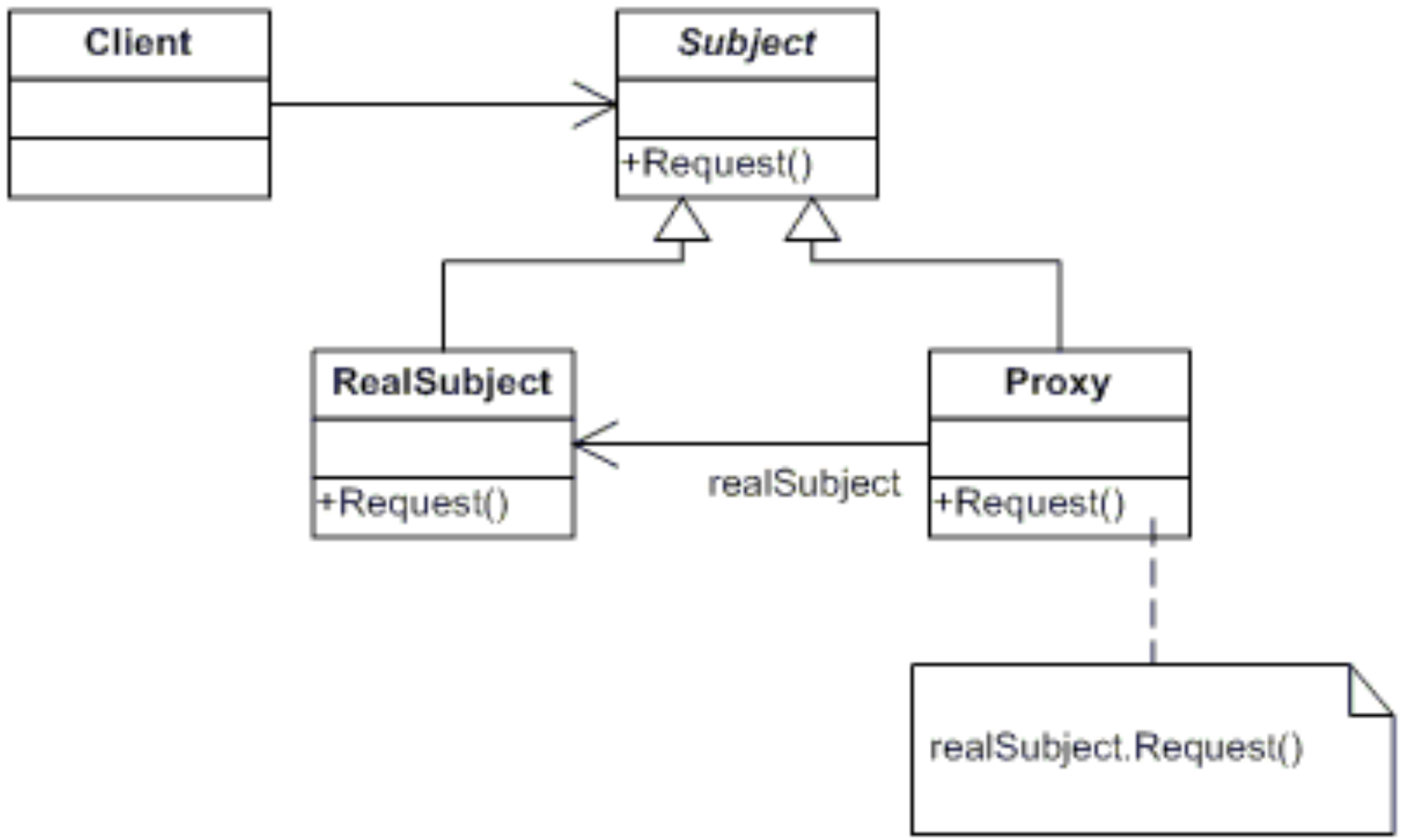
Adapter - Příklad 3



Obr 14 Object Adapter příklad

Proxy

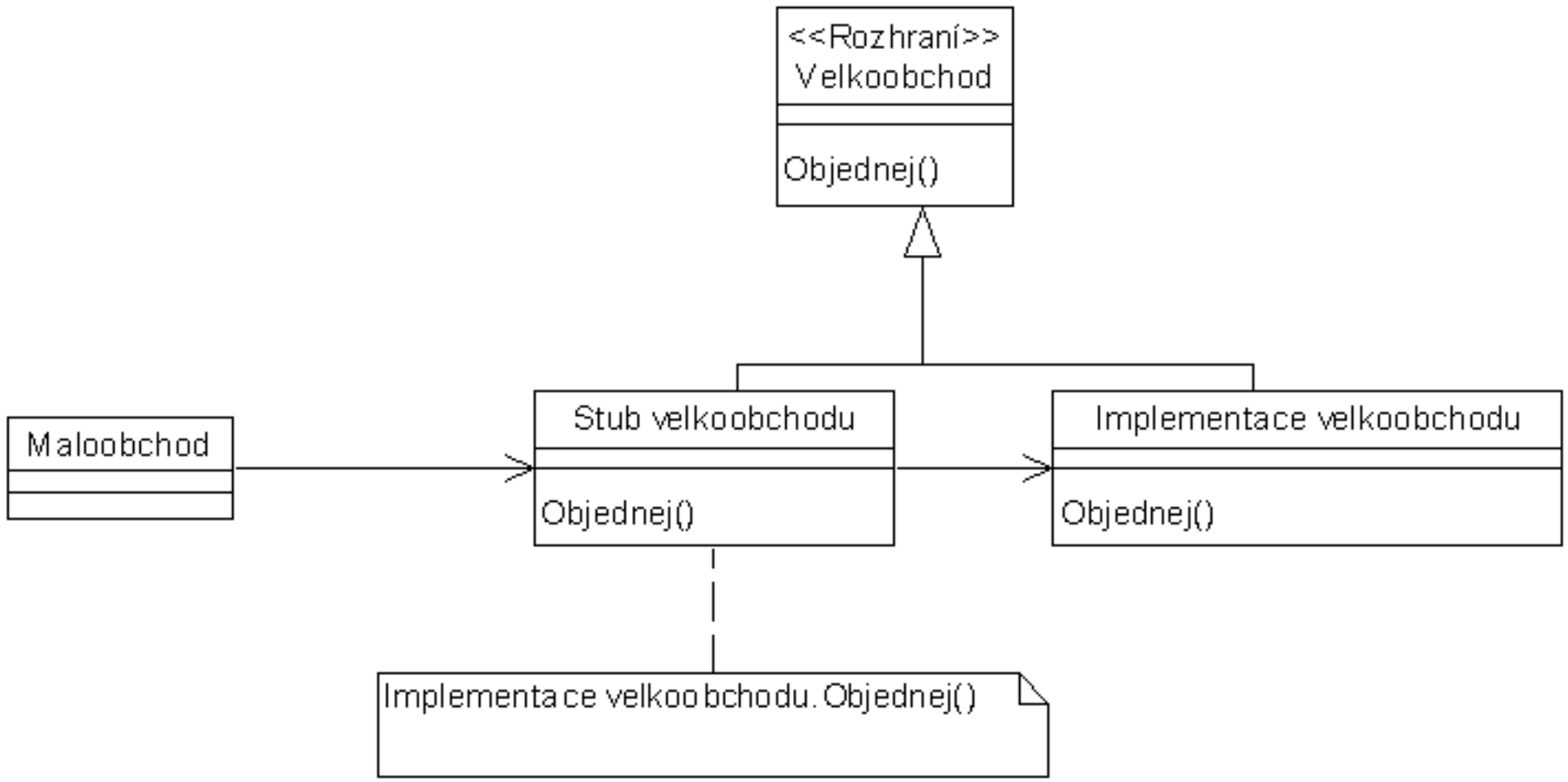
Aneb: Pod ruce mi neuvidíš



Proxy - Vlastnosti

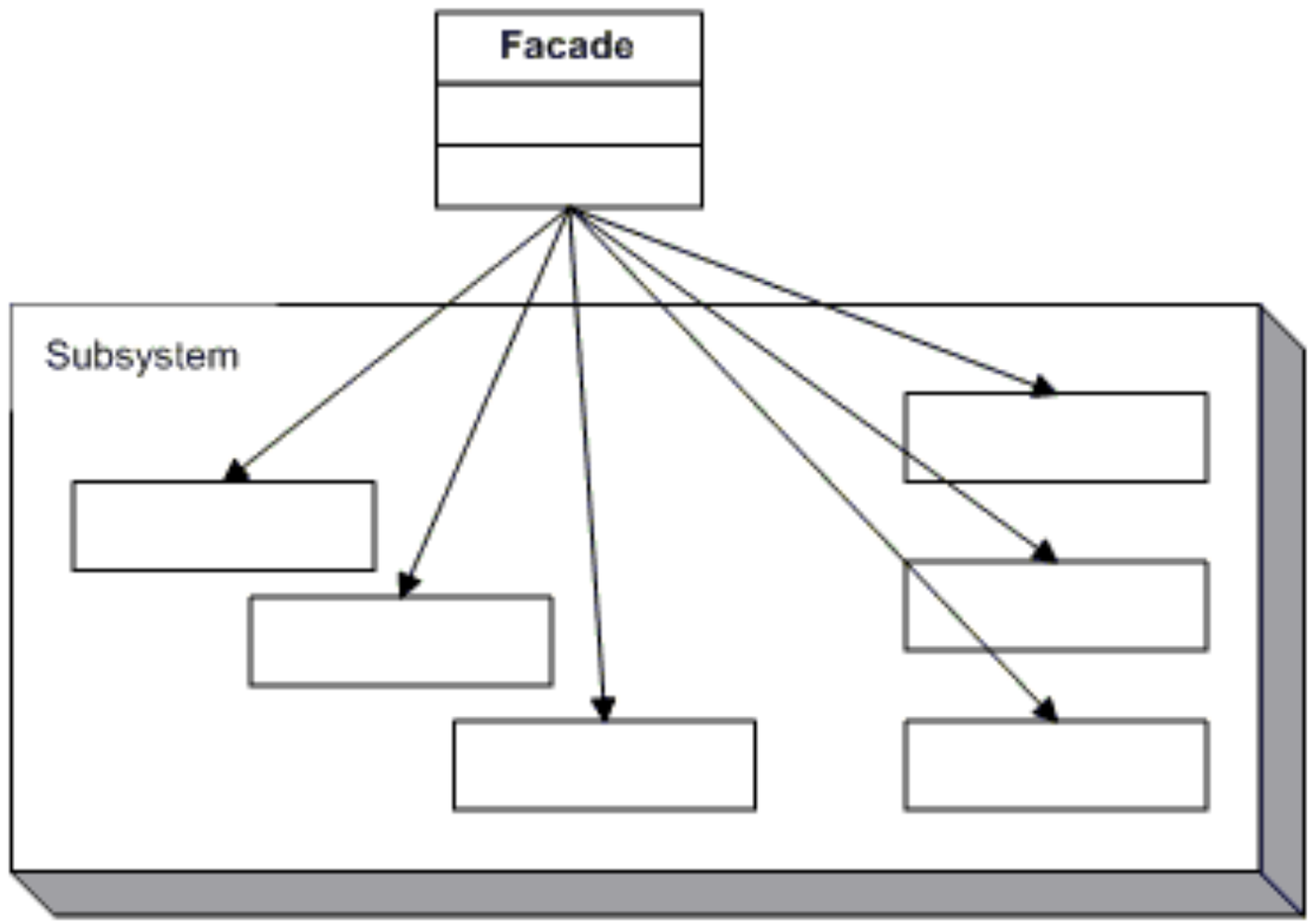
- Zastupuje cílový objekt, aniž by to uživatel poznal.
- Pomocí proxy lze například kontrolovat volání a parametry metod.
- Používá se například v RMI, Corba apod. pro zastoupení objektů, které existují fyzicky jinde.
- Lze využít například k logování nebo řízení přístupu.

Proxy - příklad



Obr 33 Příklad Proxy

Facade



Facade

- Fasáda je třída, která poskytuje jednotné rozhraní k celému souboru tříd.
- Velice užitečné řešení v případě, že rozhraní jednotlivých tříd jsou velice heterogenní a chceme je sjednotit.

Facade - Příklad

- Každý den ráno, když vyjíždím na kolo provádím na svém telefonu následující akce:
  - vypnout WiFi
  - zapnout mobilní data
  - povolit GPS
  - zapnout Bluetooth
  - zapnout aplikaci na měření, např. Runkeeper

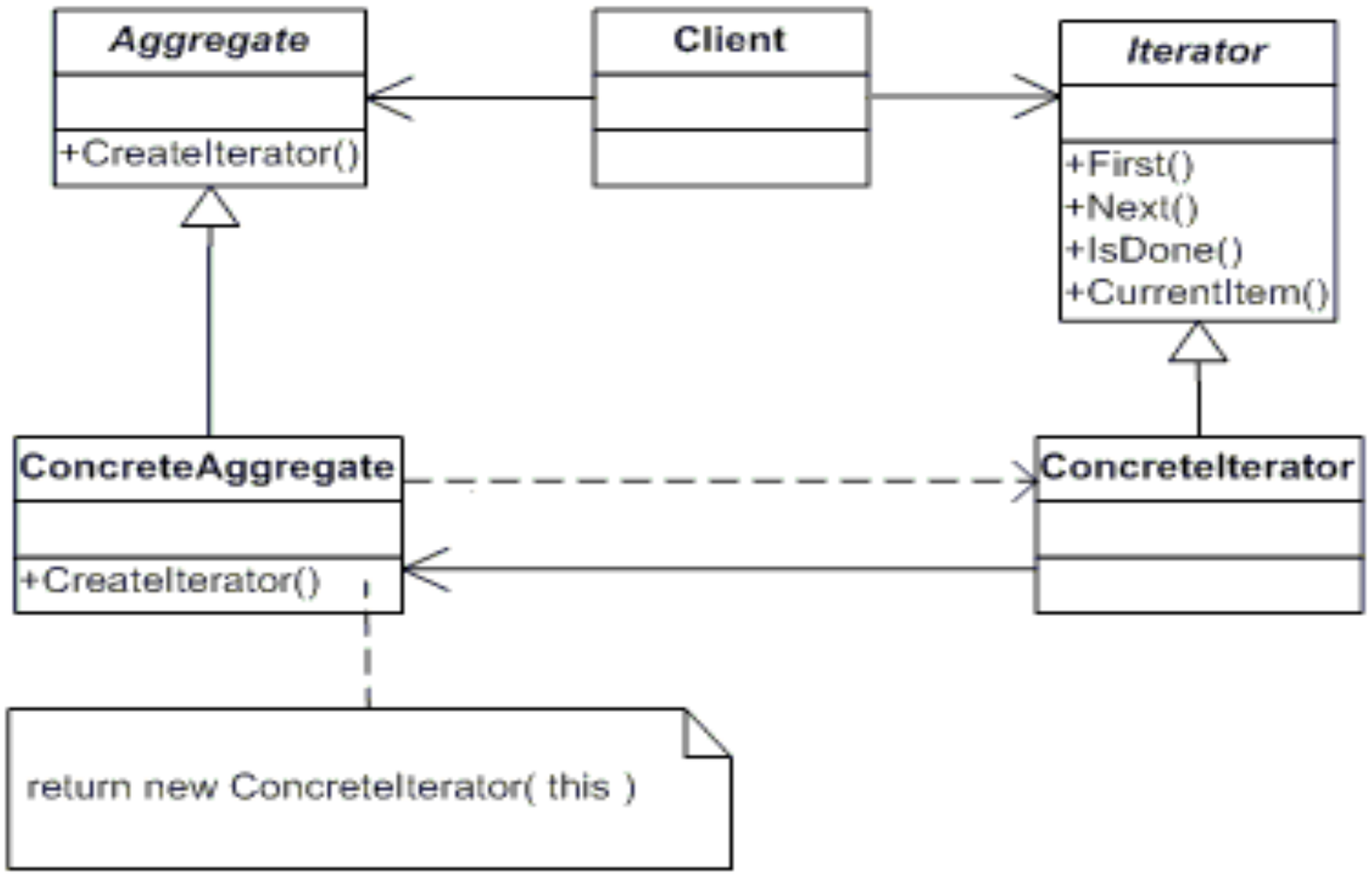
Facade - Příklad

- Poté, co přijedu z kola provádím následující akce:
  - nasdílím naměřenou vzdálenost na Facebook
  - vypnu Runkeeper
  - vypnu GPS
  - vypnu Bluetooth
  - vypnu mobilní data
  - zapnu WiFi

Facade - Příklad

- Jak bude vypadat třída, která mi usnadní život?
- Bude to právě fasáda, např. CyclingFacade s operacemi startCycling a stopCycling.

Iterator


Aggregate
+CreateIterator()
Client
Iterator
+First()
+Next()
+IsDone()
+CurrentItem()
ConcreteAggregate
+CreateIterator()
ConcreteIterator
return new ConcreteIterator(this)

Iterator

- Poskytuje rozhraní pro sekvenční procházení dané struktury.
- Uživatel iterátoru nemusí znát skutečnou reprezentaci procházené struktury.
- Příkladem může být foreach (C#, Java) nebo iterátory v STL v C++.

Literatura

- http://objekty.vse.cz/Objekty/Vzory
- http://dofactory.com/Patterns/