Programovanie v jazyku C#

Návrhové vzory

prednáška 10

Ing. Ján Magyar, PhD.

ak. rok. 2024/2025 ZS

Vzťahy v OOP

- každý program je definovaný ako interakcia medzi objektmi
- triedy (a objekty) sú definované ako kohézna entita so zväčša sebestačnou funkcionalitou
- v niektorých prípadoch stále potrebujeme, aby objekty spolupracovali a vedieť formálne popísať túto spoluprácu
- vzťahy môžu existovať na úrovni tried a inštancií

Vzťahy na úrovni tried

- definované medzi triedami
- všetky inštancie tej istej triedy majú automaticky tie isté vzťahy
- nevzťahujú sa na konkrétne objekty
- vyššie vzťahy

Generalizácia

- nazýva sa aj dedenie
- formálny popis vzťahu, ktorý je implementovaný pomocou dedičnosti v OO jazykoch
- jedna trieda je špeciálnym príkladom druhej triedy, ktorá je zovšeobecnením prvej triedy
- vzťah typu “is-a”
- nemôžeme definovať na úrovni inštancií

Realizácia

- nazýva sa aj implementácia
- jeden prvok realizuje správanie definované druhým prvkom
- môže existovať medzi triedami, rozhraniami, komponentmi a balíkmi
- zvyčajne medzi triedou a rozhraním
- vzťah typu “is-a-kind-of”

Vzťahy na úrovni inštancií

- vytvorené počas behu, existujú iba za istú časť trvania programu
- objekty tej istej triedy môžu mať rôzne vzťahy, ale zvyčajne majú vzťahy rovnakého typu
- konkrétne inštancie sú vo vzťahu s rôznymi konkrétnymi inštanciami
- nižší vzťah

Závislosť

- sémantické prepojenie
- jednosmerný vzťah medzi dvomi prvkami
- závislý prvok (**client**, **source**) je prepojený s nezávislým prvkom (**supplier**, **target**)
- ak zmeníme suppliera, možno potrebujeme zmeniť aj klienta
- najvšeobecnejší vzťah, často definovaný presnejšie

Asociácia

- jedna inštancia môže vyžadovať, aby druhá vykonala akciu pre ňu
- štrukturálny vzťah
- príklady
  - poslanie správy
  - volanie metódy
  - volanie členskej metódy
- môže byť jednosmerná alebo obojsmerná

Definovanie asociácie

- smerovanie
- môže mať názov
- môže definovať úlohy
- môže definovať indikátory vlastníctva
- podpora kardinality

Typy asociácie

- obojsmerná
- jednosmerná
- agregácia (zoskupenie)
  - kompozícia
  - agregácia
- reflexívna

Agregácia

- typ asociácie
- vzťah typu “má”
- predstavuje vzťah typu celok-súčasť alebo súčasť-niečoho
- binárna asociácia
- implementovaná ako asociácia
- inštancie, ktoré sú obsiahnuté, nemajú silnú životnú závislosť od životného cyklu agregáta

Kompozícia

- typ asociácie
- vzťah typu “obsahuje” alebo “skladá sa z”
- binárna asociácia
- implementovaná ako asociácia
- inštancie, ktoré sú obsiahnuté, majú silnú životnú závislosť od životného cyklu kompozitu

Agregácia versus kompozícia

- reprezentuje stav softvéru alebo databázy
- ak vymažeme kontajner, časti môžu ďalej existovať (študijná skupina a študenti)

- reprezentuje súčasti
- pri vymazaní kontajnera sa vymažú aj súčasti (univerzita sa skladá z fakúlt)

UML diagram tried

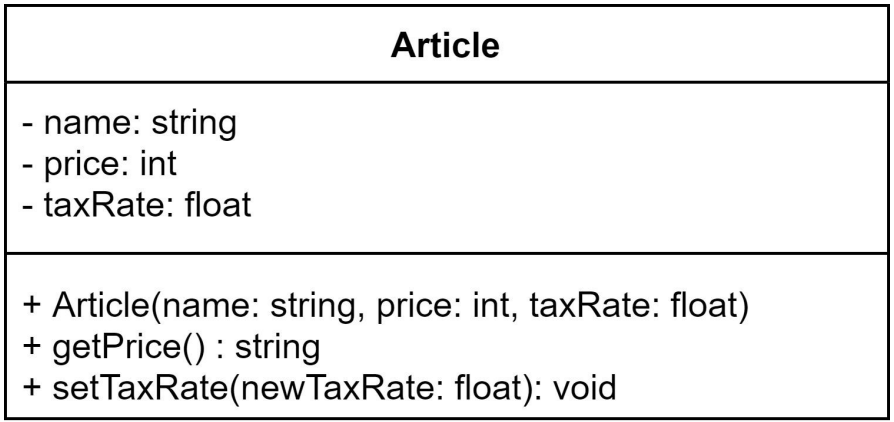
- pre grafickú reprezentáciu algoritmov používame vývojové diagramy, pre reprezentáciu vzťahov medzi triedami sa používa UML diagram tried
- každá trieda má vlastný blok
- bloky sú prepájané rôznymi šípkami, ktoré definujú vzťah
- nezávislý od jazyka a implementácie

Používanie diagramov tried

- základný stavebný blok objektovo-orientovaného modelovania
- modeluje štruktúru aplikácie
- základ pre implementáciu
- môže sa používať pre modelovanie údajov
- definuje
  - aké triedy potrebujeme implementovať
  - aká je ich štruktúra a správanie
  - aké sú vzťahy medzi triedami

Reprezentácia triedy

- každá trieda sa zobrazí v boxe
- box sa skladá z troch častí
  - názov triedy (alebo rozhrania)
  - atribúty triedy
  - zoznam operácií (metód)



Reprezentácia prístupu

\+ public

\- private

\# protected

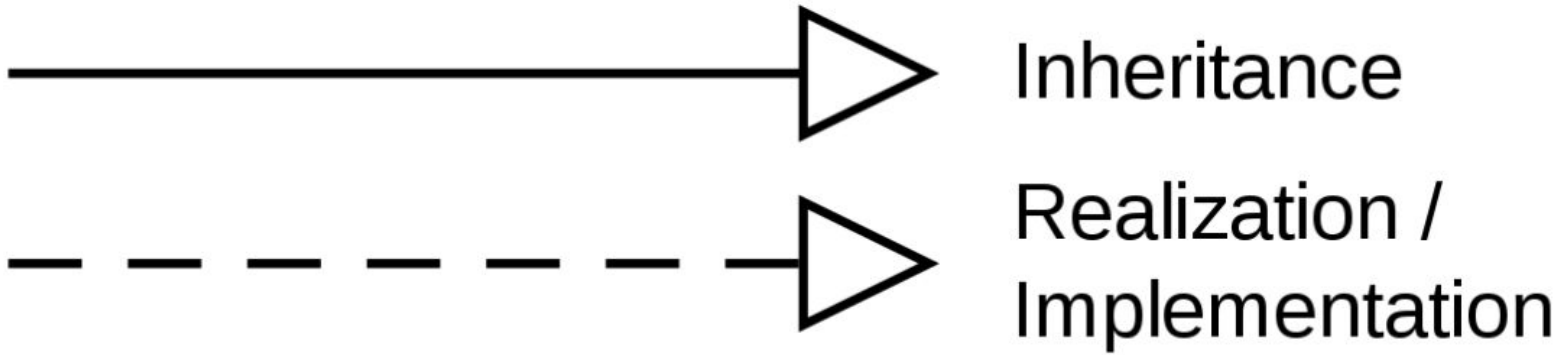
~ package/internal

Členské a statické premenné

- člen inštancie
  - každý objekt triedy má vlastnú kópiu
  - zavolanie metódy spôsobuje zmenu stavu objektu
- člen triedy (statický člen)
  - statická hodnota je rovnaká pre každú inštanciu
  - pri volaní statickej metódy nedochádza k zmene stavu
  - reprezentované podčiarknutými názvami

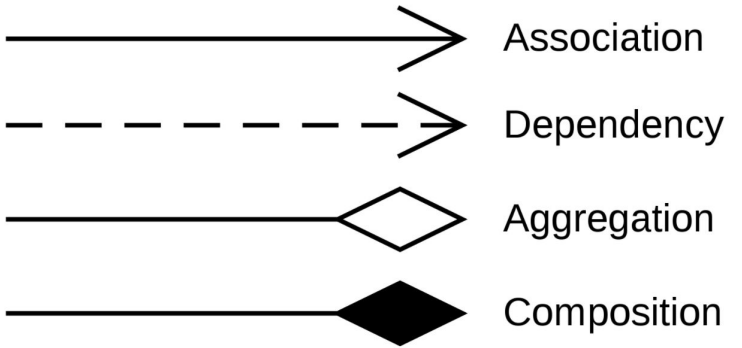
Reprezentácia vzťahov na úrovni tried

- generalizácia
  - šípka s prázdnym trojuholníkom z podtriedy do nadtriedy
- realizácia
  - šípka s čiarkovanou čiarou a prázdnym trojuholníkom



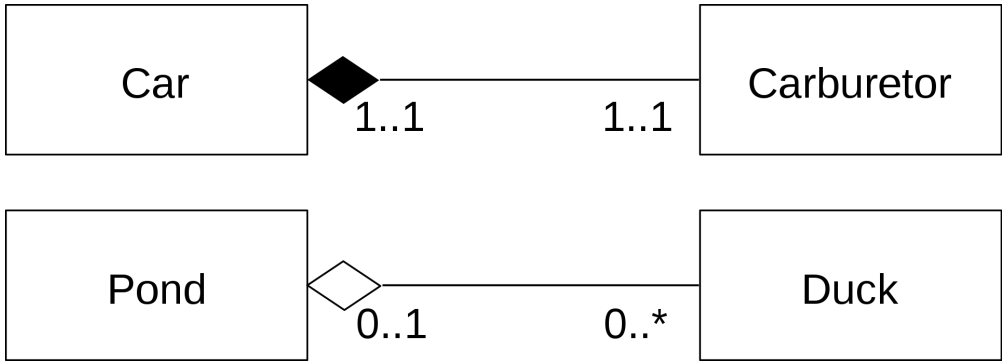
Reprezentácia vzťahov na úrovni inštancií

- asociácia
  - plná šípka
  - bez šípok v prípade obojsmernej asociácie
- závislosť
  - čiarkovaná šípka z clienta do suppliera
- agregácia
  - plná čiara s prázdnym kosoštvorcom
- kompozícia
  - plná čiara s plným kosoštvorcom



Kardinalita

- 0 - žiadna inštancia
- 0..1 - žiadna inštancia alebo presne jedna
- 1/1..1 - presne jedna inštancia
- 0..*/* - žiadna inštancia alebo viac inštancií
- 1..* - jedna alebo viac inštancií



Návrhový vzor

- všeobecné znovupoužiteľné riešenie pre opakujúce sa problémy v softvérovom inžinierstve
- popis riešenia, nie samotné riešenie
- formalizovaný best practice
- nie sú nevyhnutné, ale zjednodušujú implementáciu

Štruktúra návrhového vzoru

- definuje komponenty a ich rolu
- definuje vzťah medzi komponentmi
- nešpecifikuje funkcionalitu (závisí od prípadu použitia)
- nešpecifikuje implementáciu (je to na programátorovi)

Použitie návrhových vzorov

- pre časté problémy
- programovací jazyk môže ponúkať implicitné riešenie problému
- zvyčajne pre objektovo-orientované jazyky a programy
- osvedčený spôsob písania kvalitného softvéru

Výhody použitia návrhových vzorov

- rýchlejší vývoj
- bezpečnejšie a overené riešenia
- rieši aj skryté problémy
- lepšia čitateľnosť a štruktúra riešenia
- rozdelenie do komponentov pre znovupoužitie kódu
- zdokumentované riešenie

Kritika návrhových vzorov

- môžu naznačiť chýbajúcu podporu v programovacom jazyku
- môžu byť implementované inými prístupmi
- zvyšuje zložitosť riešenia pri nevhodnom použití

Typy návrhových vzorov

- kreačné - ako vytvoriť objekt?
- štrukturálne - ako realizovať vzťah medzi objektmi?
- behaviorálne - ako môžu komponenty komunikovať?
- konkurenčnosť - pre viacvláknové programy

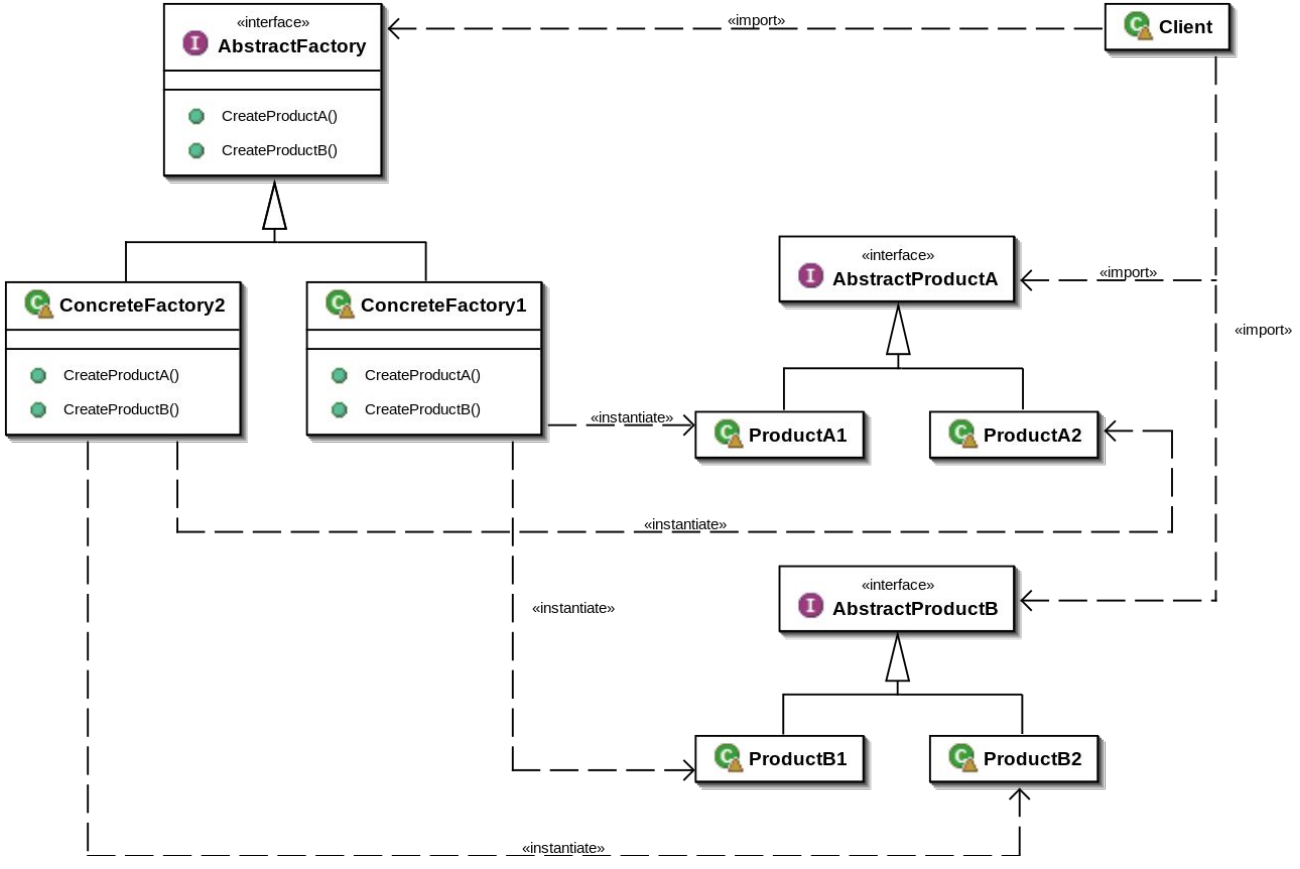
Kreačné návrhové vzory

- abstract factory
- builder
- factory method
- prototype
- singleton

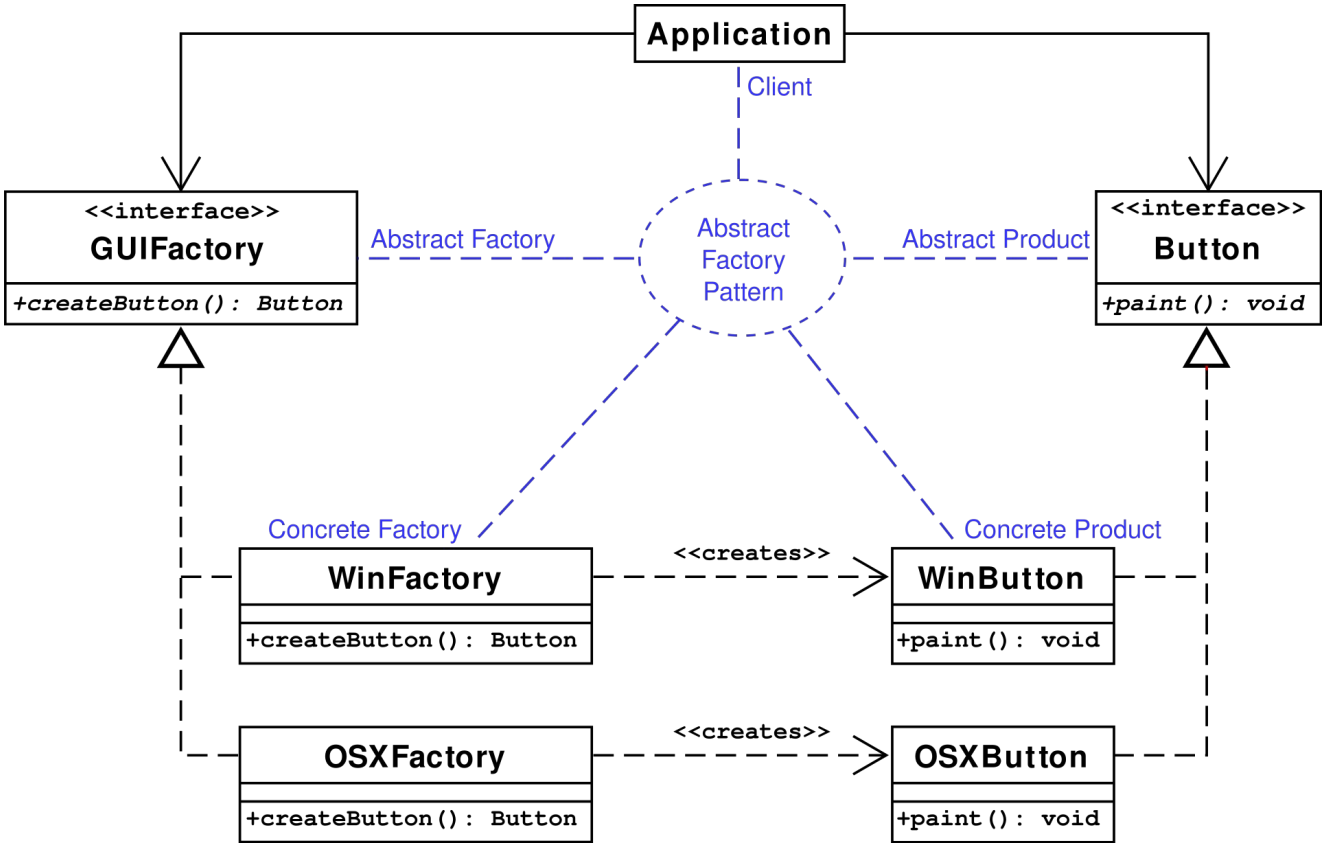
Abstract factory

- poskytnúť rozhranie pre vytvorenie skupiny závislých objektov bez špecifikácie konkrétnej triedy
- Ako môže byť aplikácia nezávislá od toho, ako sa vytvoria jej objekty, a objekty, ktoré potrebuje?
- vytvorenie objektov je skryté v osobitnom objekte
- úloha vytvorenia objektov je delegovaná factory objektu

Abstract factory



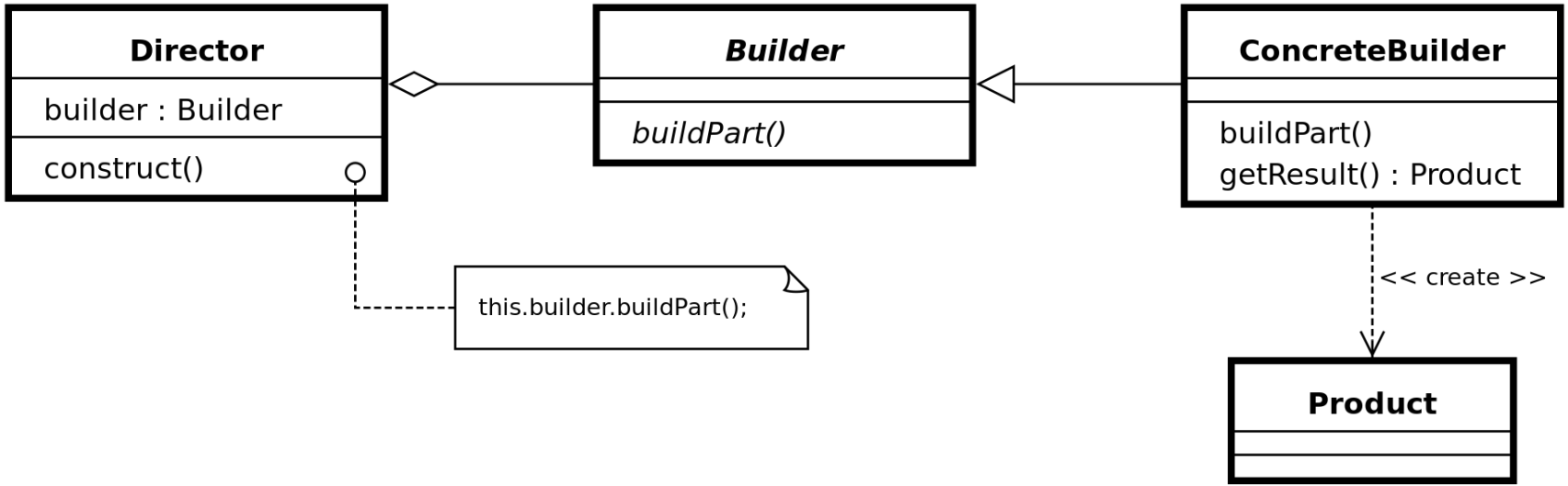
Abstract factory



Builder

- pre zložité objekty, oddelíme ich reprezentáciu od ich vytvorenia
- rovnaký proces môže vytvoriť rôzne reprezentácie
- Ako môžeme zjednodušiť triedu, ktorá obsahuje vytvorenie zložitého objektu?

Builder


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

Použitie vzoru Builder

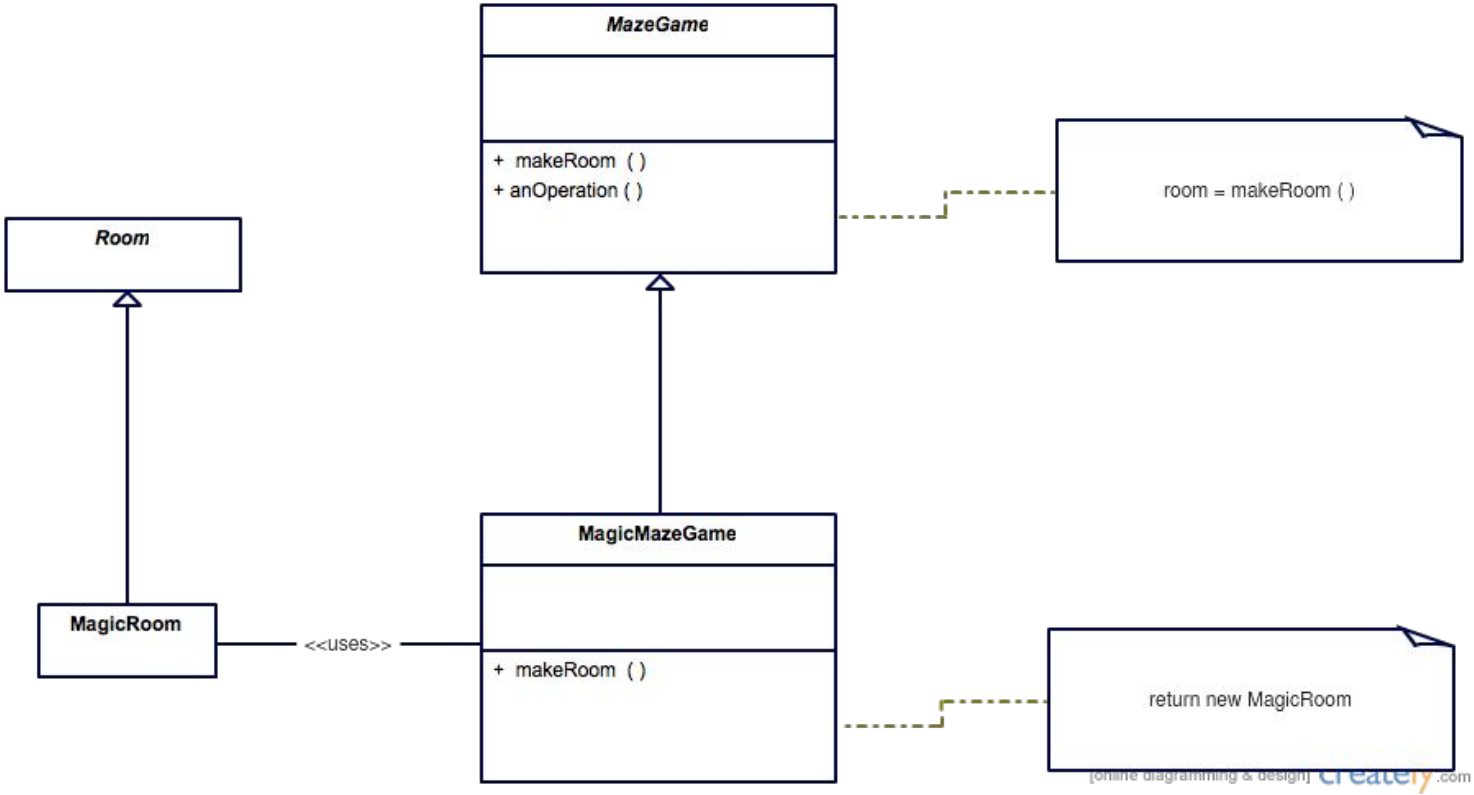
- môžeme meniť vnútornú reprezentáciu objektu
- enkapsulovaný kód pre vytvorenie a reprezentáciu
- môžeme kontrolovať proces vytvorenia

- musíme zadefinovať builder pre každý typ produktu
- triedy buildera môžeme meniť
- ťažší dependency injection

Factory method

- rozhranie pre vytvorenie jediného objektu, ale podtriedy rozhodujú, inštanciu ktorej triedy majú vytvoriť
- vytvorenie inštancií je úlohou podtried
- osobitná operácia (factory method) je zodpovedná za vytvorenie objektu, objekt vytvoríme zavolaním tejto metódy

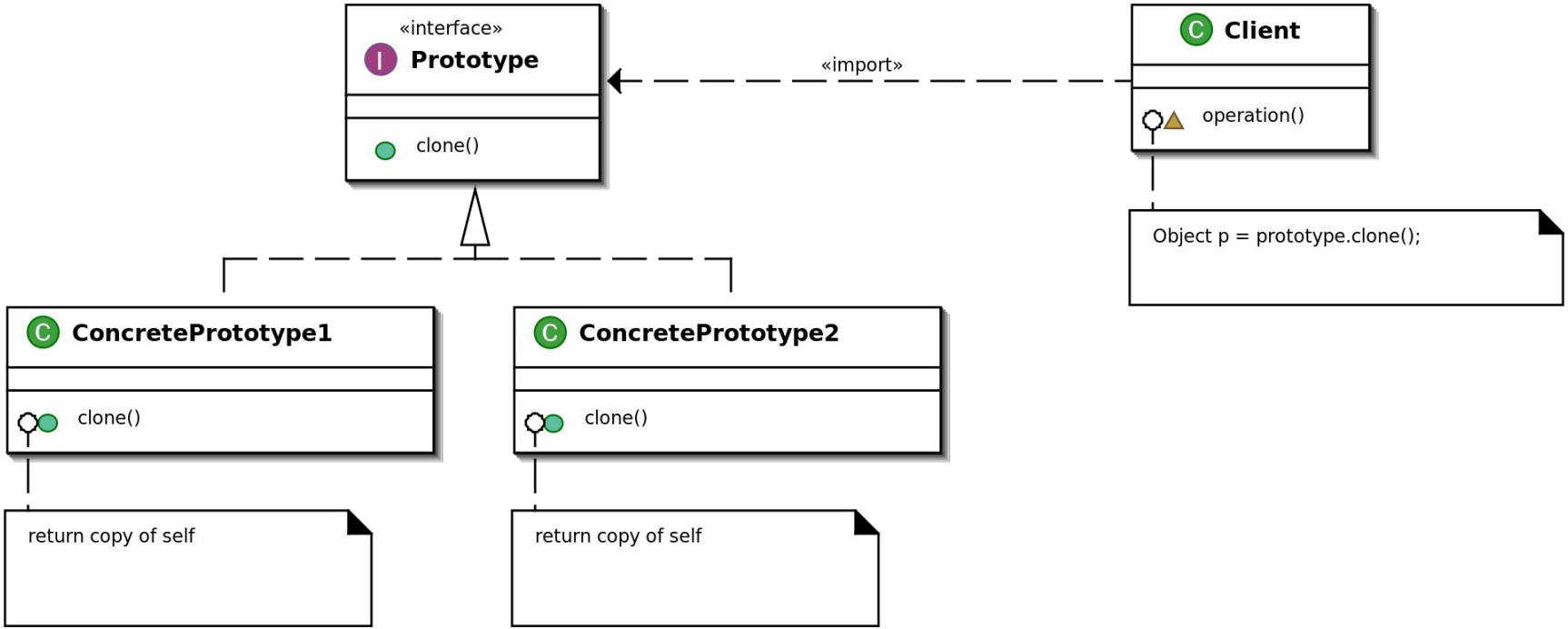
Factory method



Prototype

- objekty vytvárame na základe prototypovej inštancie
- vytváranie inštancií použitím už existujúceho objektu
- lepší výkon, menšia záťaž na pamäť
- môžeme špecifikovať počas behu, ktorý objekt sa má vytvoriť - dynamicky načítané triedy
- definujeme abstraktnú triedu s metódou `clone()`, ktorú implementujú konkrétne podtriedy

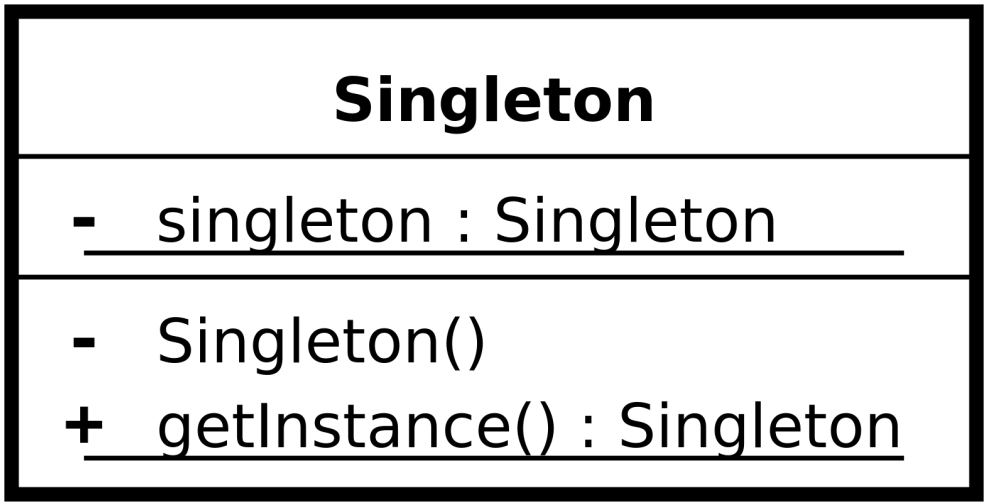
Prototype


«interface»
Prototype
clone()
«import»
Client
operation()
Object p = prototype.clone();
ConcretePrototype1
clone()
return copy of self
ConcretePrototype2
clone()
return copy of self

Singleton

- zabezpečuje, že trieda má iba jednu inštanciu (alebo žiadnu)
- poskytuje prístup k jedinej inštancii
- môžeme kontrolovať vytvorenie inštancie
- možná lazy initialization
- globálny stav

Singleton


Singleton
- singleton : Singleton
- Singleton()
+ getInstance() : Singleton

Ďalšie kreačné návrhové vzory

- dependency injection
- lazy initialization
- multiton
- object pool
- resource acquisition is initialization

otázky?