Decorator

Wrapper,
Dekoratér

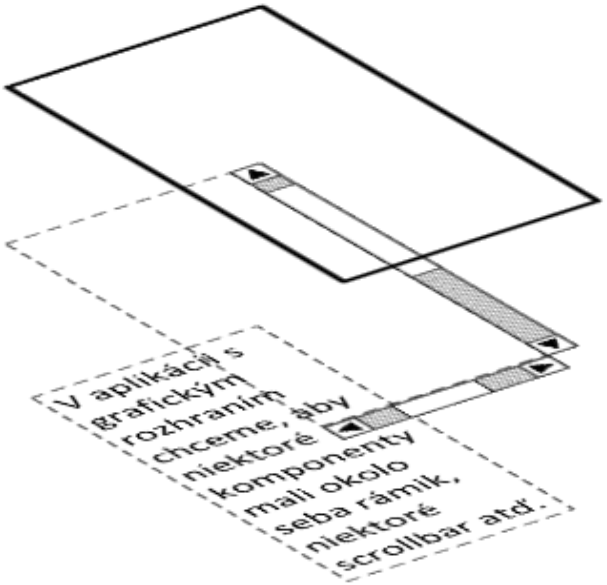
Dynamicky pridáva objektu ďalšiu funkčnosť. Dekoratér predstavuje pružnú alternatívu k dedeniu pre rozšírenie funkčnosti.

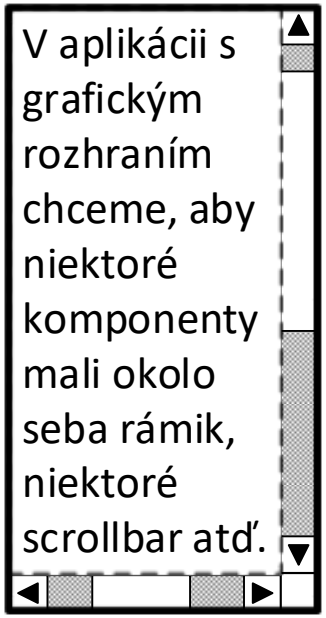
Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Motivácia

- V aplikácii s grafickým rozhraním chceme, aby niektoré komponenty mali okolo seba rámik, niektoré scrollbar, niektoré môžu mať oboje.
  - Jedno riešenie je pomocou dedičnosti – predok definuje, že má rámik. To by ale znamenalo, že všetci jeho potomkovia sú už pevne spätí s rámikom.
  - Lepšie riešenie obalí požadovaný objekt iným objektom, ktorý bude vedieť tento rámik pridať – dekoratér. Má rovnaké rozhranie ako dekorovaný objekt, preposiela mu všetky volania metód, môže však každé volanie „dekorovať“.

Motivácia

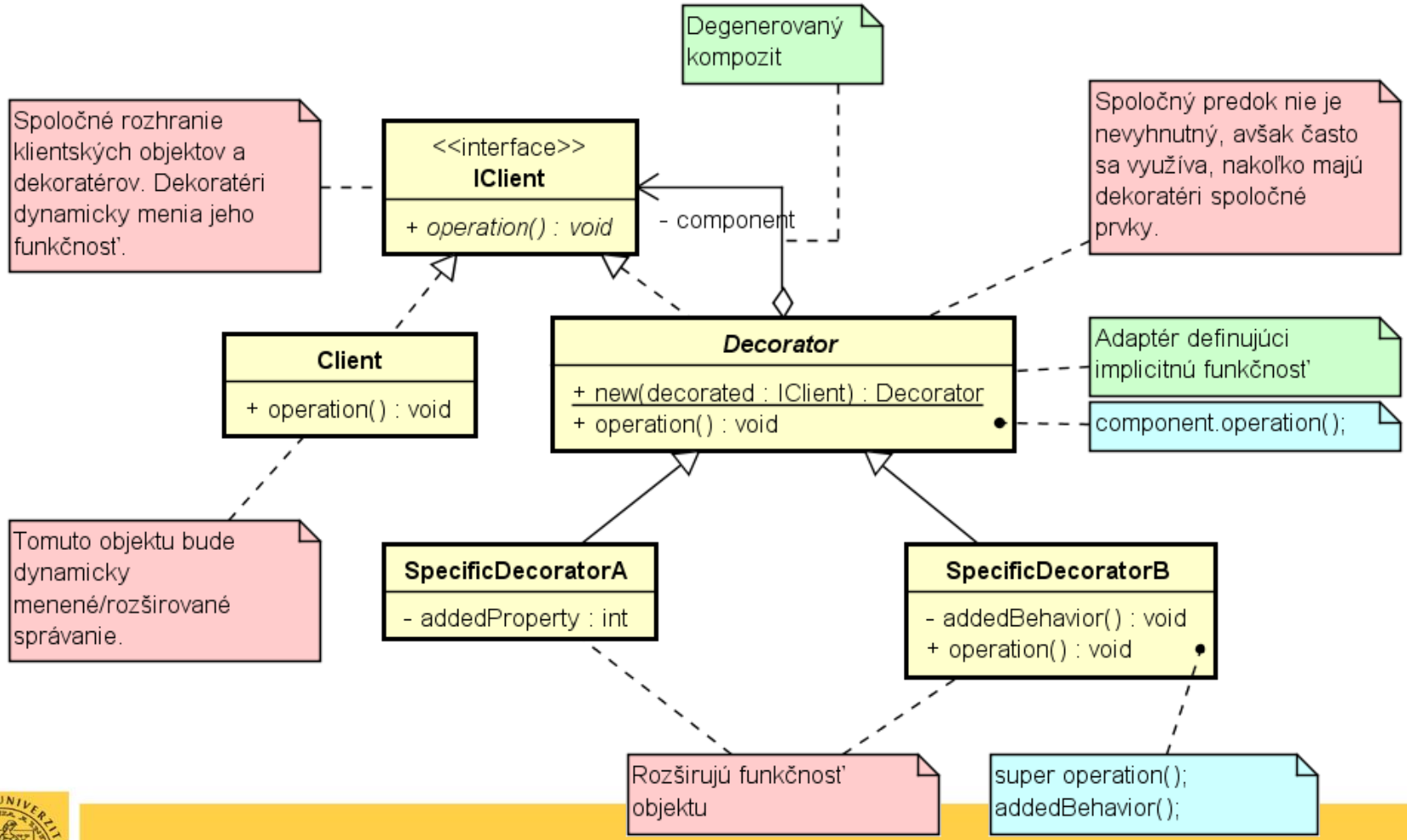

V aplikácii s grafickým rozhraním chceme, aby niektoré komponenty mali okolo seba rámik, niektoré scrollbar atď.


V aplikácii s grafickým rozhraním chceme, aby niektoré komponenty mali okolo seba rámik, niektoré scrollbar atď.

Aplikovateľnosť

- Dekoratér je vhodné použiť, keď:
  - chcete pridať funkčnosť jednotlivým objektom dynamicky a prehľadne bez ovplyvnenia iných objektov.
  - je potrebné reprezentovať činnosti objektu, ktoré je možné potlačiť.
  - nie je možné použiť dedičnosť:
    - kvôli veľkému množstvu možných rozšírení by došlo k explózii tried.
    - definícia predka (alebo inej triedy) nemusí byť dostupná.

Implementácia



Dôsledky

- Pružnejší ako dedičnosť. Každá ďalšia funkčnosť je definovaná samostatným dekoratérom, je ich možné pridávať/odoberať za behu programu, navyše je možné dodať nejakú vlastnosť viackrát (napr. dvojité orámovanie).
- Pri návrhu nemusíte ihneď myslieť na všetku funkčnosť, ale je ju možné dorábať po častiach. Pozor – takýto prístup môže viesť k zámeru vytvoriť triedu „superdekoratér“, ktorý je schopný rozšíriť funkčnosť akémukoľvek objektu, čo si však žiada odhalenie implementačných detailov dekorovanej triedy!
- Pozor na identitu objektov! Dekoratér a dekorovaná trieda majú rovnaké rozhranie, ale nie sú to rovnaké objekty.
- S využitím dekoratérov bude často celkový systém pozostávať z veľkého množstva malých objektov. Systém síce bude flexibilný, ale o to ťažší na pochopenie a ladenie.

Príklad

- Typické využitie v streamoch v jazyku Java:
  - *BufferedInputStream* - **adds functionality to another input stream** - namely, the ability to buffer the input and to support the mark and reset methods.
  - *FilterInputStream* **contains some other input stream**, which it uses as its basic source of data, possibly transforming the data along the way or providing **additional functionality**.
  - *PrintStream* **adds functionality to another output stream**, namely the ability to print representations of various data values conveniently.
  - *PushbackInputStream* **adds functionality to another input stream**, namely the ability to "push back" or "unread" one byte.

Príbuzné vzory

- *Adapter* – Dekoratér mení iba správanie objektu, Adaptér mení jeho rozhranie.
- *Composite* – Dekoratér je degenerovaným kompozitom (má iba jeden komponent), ale jeho účelom je meniť správanie tohto komponentu, naproti tomu Kompozit agreguje objekty.
- *Strategy* – Stratégia predstavuje alternatívu k Dekorátoru. Stratégia však ide „do hĺbky“ (mení správanie metódy), Dekoratér „po povrchu“ (obaľuje existujúcu metódu).

Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu 5II132 Návrhové vzory (Design Patterns) na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline.
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené.

Ing. Michal Varga, PhD.
Katedra informatiky
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline
Michal.Varga@fri.uniza.sk