Quo vadis programování?

Automatizace vyhodnocování studentských úloh

Katedra informačních technologií
Kit
VŠE Praha

Rudolf PECINOVSKÝ
rudolf@pecinovsky.cz
Vladimír Oraný
vladimir.orany@gmail.com

Obsah s odkazy

- Děkuji

Současný stav

- Metody testování
- Testování studentských úloh
- „Bílé testování“ a jeho problémy

Katedra informačních technologií
Kit
VŠE Praha

Metody testování

Podle nutnosti spuštění programu

- Statické - program nemusí být spuštěn
  - Hledají se (většinou potenciální) chyby ve zdrojových souborech
- Dynamické – testování chodu programu

Podle znalosti struktury

- Černá skříňka (Black box) – testuje se funkce programu
  - Kontroluje, zda systém na zadaný vstup odpoví očekávaným výstupem
- Skleněná/Bílá skříňka (Glass/White box) – testuje se i implementace
  - Test využívá znalosti vnitřní struktury programu
- Šedá skříňka (Gray box) – kombinace předchozích dvou
  - Když jsou naše informace o struktuře programu pouze částečné

Automatizace

- Manuální testování – testy spouští a kontroluje člověk
- Testovací program kontroluje výsledky (a často se i spouští) automaticky

Testování studentských úloh

- Vzhledem k časové náročnosti jsou velké tlaky na jejich automatizaci
- Většina systémů se soustředí především na odevzdávání a archivaci odevzdaných řešení
- Mnohé systémy se snaží kontrolovat duplicitu řešení
- Testy řešení zadaných úloh
  - Statické
    - Spouští se některý z kontrolních programů (PMD, FindBug, ...)
    - Ověřuje se především dodržení konvencí
    - Neanalyzuje se správnost řešení
  - Dynamické
    - Testuje se pouze celková funkčnost (program = černá skříňka)
- Skoro nikde se studentské úlohy netestují jako bílé skříňky

„Bílé testování“ a jeho problémy

- **„Bílé testování“ částečně usnadní, pokud studenty včas naučíme implementovat rozhraní**
  - Existenci implementace požadovaných metod zkontroluje překladač
  - Automatický test pak může ověřit správnou funkci metod deklarovaných v implementovaném rozhraní
- **Problém: nelze syntakticky vyžádat definici**
  - Statické metody
  - Konstruktoru
  - Soukromé metody
  - Zadaného pole
  - ...

Framework Duckapter

- Principy
- Koncepce
- Diagram tříd hlavních součástí
- Funkce

Katedra informačních technologií
Kit
VŠE Praha

Principy

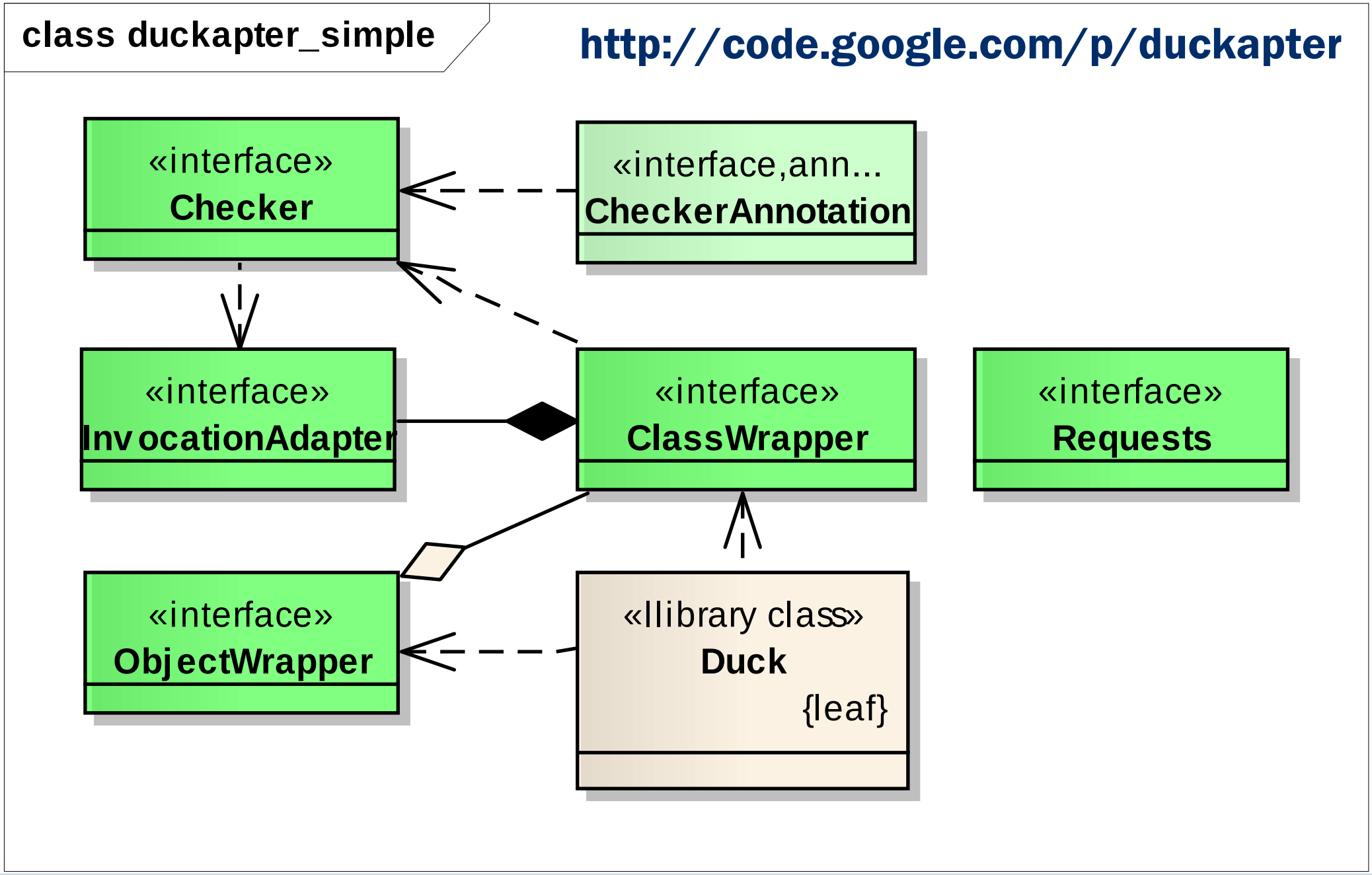
- **Kachní typování (Duck typing)**
  - Zvíře, které chodí jako kachna, plave jako kachna a kváká jako kachna, bude nejspíš kachna
  - Nevyžaduje přesnou deklaraci typu kontrolovatelnou v době překladu
  - Nevadí, jakého typu objekt ve skutečnosti je, hlavně že umí vše, co má objekt požadovaného typu umět
- **Návrhový vzor *Adaptér*, resp. *Zástupce***
  - Potřebujeme-li, aby instance měla jiné rozhraní, než to, které právě má, můžeme mezi ni a potencionálního uživatele vložit instance adaptéru s požadovaným rozhraním, která předá požadavky naší třídě.
- **Anotace**
  - Umožňují začlenit do programu požadavky, které nejsou zadatelné standardními prostředky jazyka
- **Z prvních dvou principů vyhází i název frameworku duckapter = duck adapter**

Koncepce

- Vedle rozhraní, které třída musí implementovat, definujeme tzv. požadované rozhraní, přesně definující požadované vlastnosti třídy včetně těch, které syntaxe jazyka neumožňuje zadat
- Požadavky nezadatelné prostřednictvím standardní syntaxe se zadávají prostřednictvím anotací
- Příklad:

```
//Testovaná třída má být abstraktní
@Abstract public interface PožadovanéRozhraní {
    @Private @Static void soukrStatickáMetoda();
    @Private @Final @Field int
             instančníCeločíselnáKonstanta();
    @Private @Transient @Field String
             neserializovanáStringováProměnná();
}
```

Diagram tříd hlavních součástí



Funkce

- **Ověření toho, že třída vyhovuje požadovanému rozhraní, má na starosti knihovní třída `Duck`**
  1. Zjistí, jestli má pro danou testovanou třídu již vytvořenu instanci třídy `ClassWrapper` – pokud ne, nechá ji vytvořit
  2. Požadujeme-li testování objektu, vytvoří pro tento účel instanci třídy `ObjectWrapper`
  3. Pomocí této instance vytvoří virtuálního zástupce (proxy) implementujícího požadované rozhraní
  - Následné testy spočívají ve volání metod tohoto zástupce
- **Konstrukce instance třídy `ClassWrapper`**
  1. Projdou se všechny deklarace požadovaného rozhraní a u každé se zjistí, které anotace (`CheckerAnnotation`) ji „zdobí“
  2. S každou anotací je sdružená třída implementující `Checker`, která vytvoří instanci třídy implementující `InvocationAdapter`
  3. Tyto instance si vytvářená instance třídy `ClassWrapper` zapamatuje

Děkuji za pozornost

Rudolf Pecinovský
mail: rudolf@pecinovsky.cz
ICQ: 158 156 600