Optimalizace rozhraní

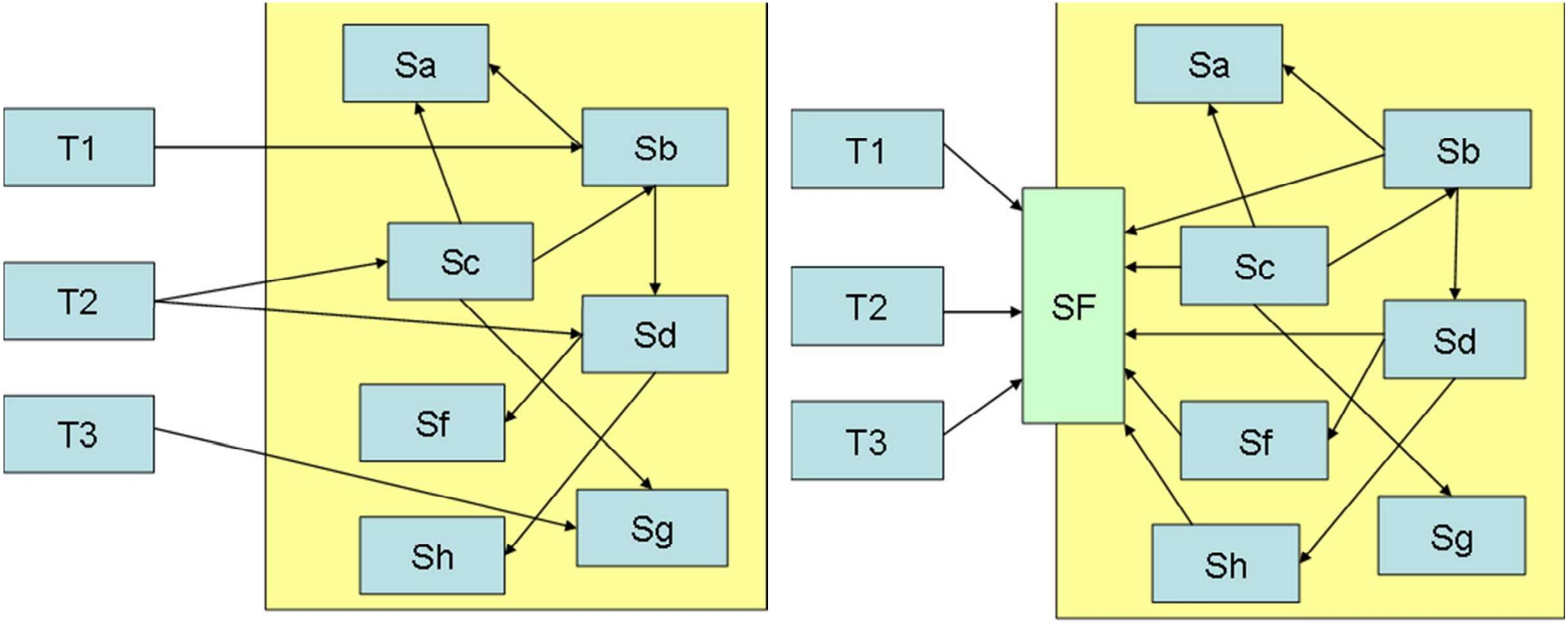
Fasáda (Facade) 041

- Ukazuje jak nahradit sadu rozhraní jednotlivých subsystémů sjednoceným rozhraním zastupujícím celý systém. Definuje tak rozhraní vyšší úrovně, které usnadní využívání podsystémů. Jejím cílem je zjednodušit rozhraní celého systému a snížit počet tříd, s nimiž musí uživatel přímo či nepřímo komunikovat.

Charakteristika

- **Použijeme jej ve chvíli, kdy nějaký systém začíná být pro své uživatele příliš složitý vzhledem k oblasti úloh, které chtějí s jeho pomocí řešit**
- **Z celého spektra dostupných metod vybere podmnožinu nejpoužívanějších, nebo přímo definuje vzorové verze metod pro nejčastěji používané úlohy**
- **Možné způsoby implementace**
  - Rozhraní či abstraktní třídy (příp. systém R+AT) jejichž implementaci definují až potomci
  - Konfigurovatelná třída (slabší varianta předchozího)
  - Samostatná třída (skupina tříd) poskytující nejčastěji požadované metody

Komunikace tříd před a po použití fasády



- **Příklad:**
  - javax.swing.JOptionPane

Výhody použití

- **Redukuje počet objektů, s nimiž klienti komunikují**
  - Snadnější použití subsystému
- **Zmenšuje počet závislostí mezi klienty a subsystémem**
  - Odstraňuje některé komplexní a kruhové závislosti
  - Méně závislostí při překladu i při běhu
- **Liberální fasáda: neskrývá třídy subsystému**
  - Klient si může vybrat jednoduchost nebo použití na míru
- **Přísná fasáda: nezaručuje implementaci systému**
  - Náhražka neexistující možnosti přísnějšího skrytí implementace
  - Java 7 má zavést superpackages – pak nebude potřeba

Adaptér (Adapter) 042

- Návrhový vzor Adaptér využijeme ve chvíli, kdy bychom potřebovali, aby třída měla jiné rozhraní, než to, které právě má. Pak mezi ní a potenciálního uživatele vložíme třídu adaptéru, která bude mít požadované rozhraní a konvertuje tak rozhraní naší třídy na rozhraní požadované.

Motivace

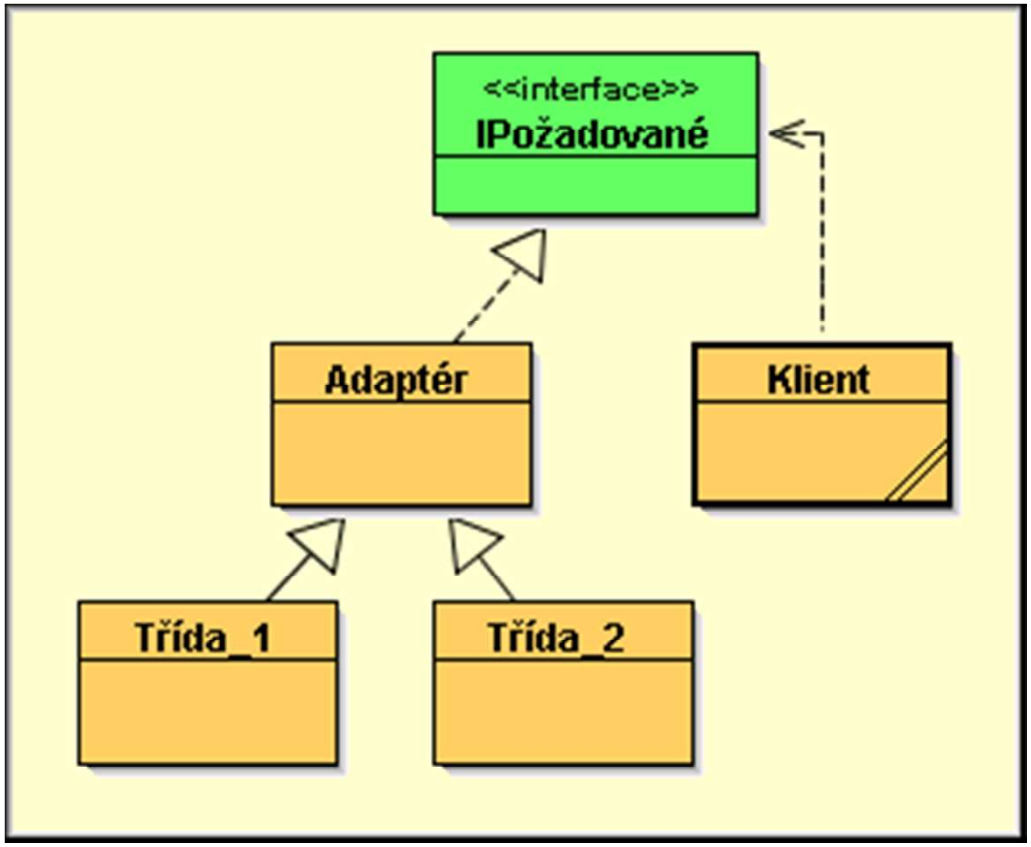
- **Občas potřebujeme, aby třída měla jiné rozhraní než to, které má**
  - Třída neimplementuje požadované rozhraní,
    nicméně poskytuje požadovanou funkčnost
  - **Příklad:**
    používáme třídu z jedné knihovny,
    jejíž instance bychom mohli použít jako parametry metod jiné knihovny,
    ale tato jiná knihovna vyžaduje parametry
    implementující nějaké specifické rozhraní, které naše třída nezná
- **Dopředu víme, že z hlediska požadované funkčnosti stačí implementovat pouze část daného rozhraní nicméně překladač vyžaduje kompletní implementaci**
  - Iterátor neumožňující odstraňovat prvky z kontejneru
  - Kontejnery s předem zadaným, nezměnitelným obsahem
  - Posluchači některých událostí při tvorbě GUI

Charakteristika

- **Účel**
  - Zabezpečit spolupráci již exitujících tříd,
    tj. tříd, jejichž rozhraní už nemůžeme měnit
  - Usnadnit definici nových tříd, v nichž pak nemusíme implementovat celé požadované rozhraní
- **Nasazení**
  - Použití tříd z pevných knihoven v jiných prostředích
  - Využití třídy s požadovanou funkčností, ale jiným rozhraním
  - Implementace ekvivalence mezi rozhraními
  - Doplnění funkcionality třídy na požadovanou rozhraním

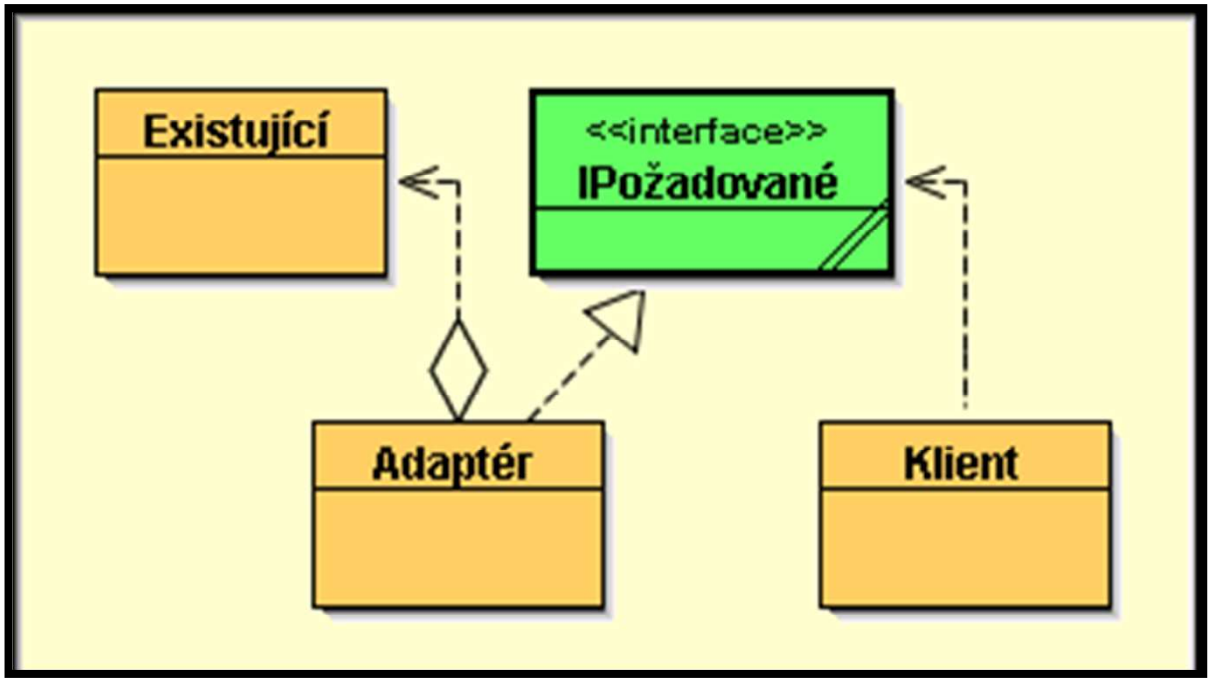
Adaptér jako rodič adaptované třídy

- Třída `Adaptér` definuje implicitní implementace všech metod požadovaných rozhraním `IPožadované` přičemž implicitní verze typicky:
  - Vyhazuje `UnsupportedOperationException`
  - Nedělá nic
- Potomci pak mohou definovat pouze ty metody, které se jim „hodí do krámu“
- Pro klienta potomek implementuje vše



Adaptovaný objekt jako atribut 1/2

- Pracuje stejně jako ochranný zástupce, pouze s jinou motivací
- Adaptér definuje atribut s odkazem na adaptovaný objekt




```
public class Adaptér implements IPožadované {
  Existující adaptovaný;

  public Adaptér(Existující exist) {
    adaptovaný = exist;
  }
```

Adaptovaný objekt jako atribut 2/2

- Všechna volání metod „přehrává“ na volání ekvivalentních metod adaptovaného objektu

```
public class Adaptér implements IPožadované {
  Existující adaptovaný;

  public Adaptér(Existující exist) {
    adaptovaný = exist;
  }

  public void metoda(Parametr parametr) {
    Požadovaný požadovaný = uprav(parametr);
    adaptovaný.jehoMetoda(požadovaný);
  }
}
```

Příklady

- **Adaptace prostřednictvím předka**
  - Všechna rozhraní posluchačů `java.awt.event.XyzListener` deklarující více než jednu metodu mají sdružené třídy `java.awt.event.XyzAdapter`
- **Adaptace prostřednictvím atributu**
  - Instance třídy `Barva` z knihovny Tvary jsou jemně upravenými počeštěnými obálkami kolem instance typu `java.awt.Color`
  - Instance třídy `Kreslítko` z knihovny Tvary jsou zestručněnými počeštěnými obálkami kolem instance typu `java.awt.Graphics2D`

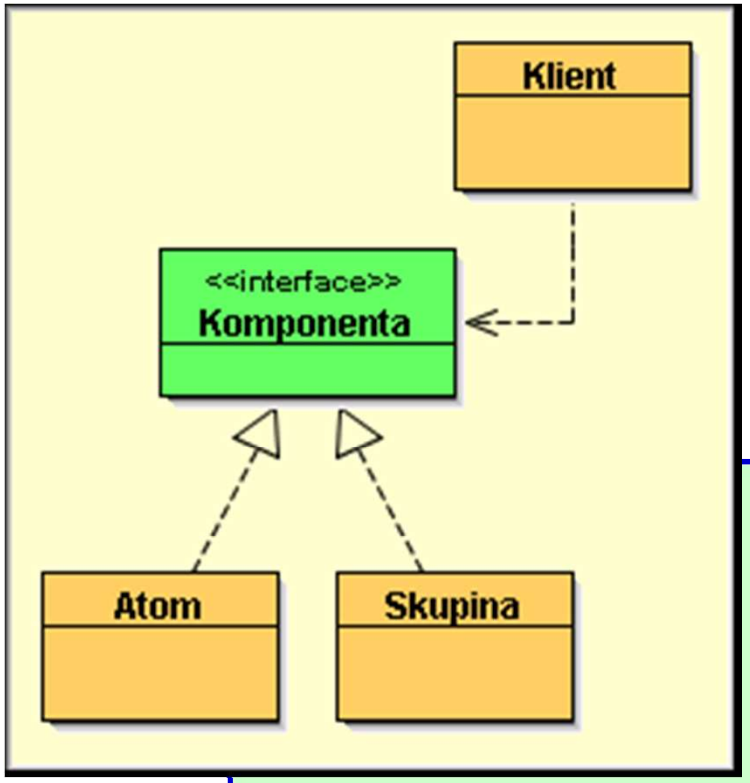
Strom (Composite) 043

- Sjednocuje typy používaných objektů a umožňuje tak jednotné zpracování každého z nich nezávisle na tom, jedná-li se o atomický (tj. dále nedělitelný) objekt nebo o objekt složený z jiných objektů.

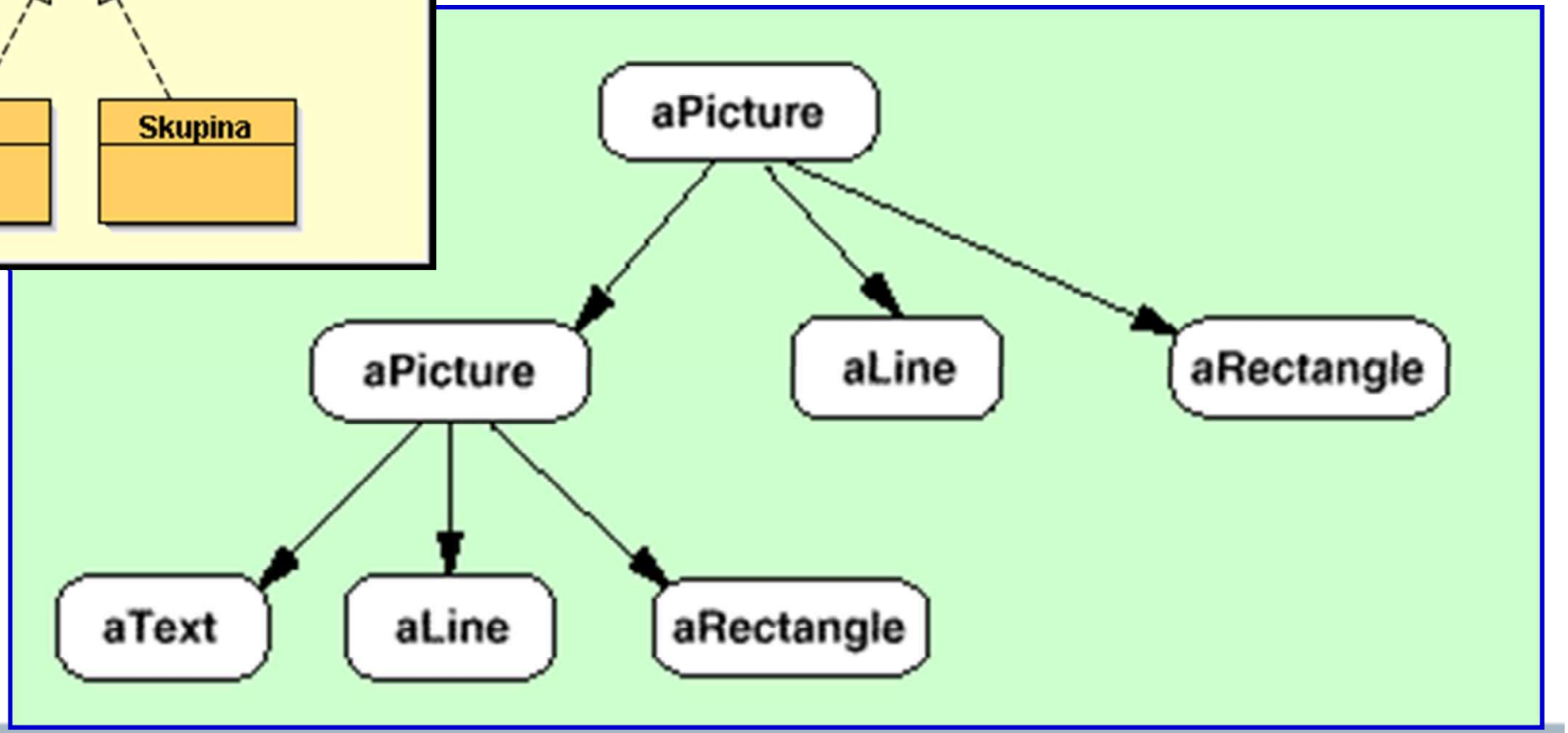
Charakteristika

- **Ukazuje, jak vytvořit hierarchii složenou ze dvou druhů objektů:**
  - Atomických (primitivních)
  - Složených (z atomických či dalších složených)
- **Většinou tvoří struktura strom, ale není to nutné**
- **Použití**
  - Grafické editory vytvářející složité objekty z jednodušších
  - Reprezentace tahů a protitahů ve hrách
  - Adresářová struktura
  - Struktura různých orgranizací
  - Implementace nejrůznějších dalších stromových struktur

Diagram tříd v návrhovém vzoru *Strom*



- Doporučuje definovat pro atomické i složené objekty společného rodiče



Důsledky použití vzoru

- **Zjednodušuje klienta, protože může s atomickými i složenými objekty zacházet jednotně**
- **Jednodušeji se přidávají nové komponenty**
  - Díky společnému rodiči je klient automaticky zná
  - Je-li rodičem abstraktní třída, může v ní být již řada potřebných metod definována
- **Nebezpečí**
  - Jednoduché přidávání komponent komplikuje verifikaci, že přidávaná komponenta smí být přidána – je třeba použít kontroly za běhu
  - Metody ve společném rodiči zvyšují obecnost, ale současně snižují robustnost (nemusí na konkrétního potomka sedět), metody v potomcích naopak