Skrývání implementace

Zástupce (Proxy) 031

- Zavádí zástupný objekt,
  který odstiňuje zastupovaný objekt od jeho uživatelů
  a sám řídí přístup uživatelů k zastupovanému objektu.

Druhy zástupců 1/2

- **Vzdálený zástupce (remote proxy)**
  - Lokální zástupce vzdáleného objektu (zastupuje objekt v jiném adresovém prostoru, na jiném VM, na jiném počítači, ...)
  - Java RMI, CORBA, XML/SOAP, ...
- **Virtuální zástupce (virtual proxy)**
  - Potřebujeme-li odložit okamžik skutečného vytvoření objektu (vytvoření objektu je drahé – *load on demand*)
  - Rozlišuje mezi vyžádáním objektu a jeho skutečným použitím
  - Transparentní optimalizace (např. cache)
- **Chytrý odkaz (smart reference)**
  - Doplňuje komunikaci s objektem o doprovodné operace (počítání odkazů, správa paměti, logování, ...)

Druhy zástupců 2/2

- **Ochranný zástupce (protection proxy)**
  - Potřebujeme-li skrýt pravou identitu objektu
  - Je-li třeba ověřovat přístupová práva volajících objektů
  - Je-li třeba rozšířeně validovat parametry
- **Modifikační zástupce (Copy-on-write proxy)**
  - Opožděné kopírování objektů až při jejich modifikaci
- **Synchronizační zástupce (Synchronization proxy)**
  - Transparentní synchronizace vláken při přístupu k objektu

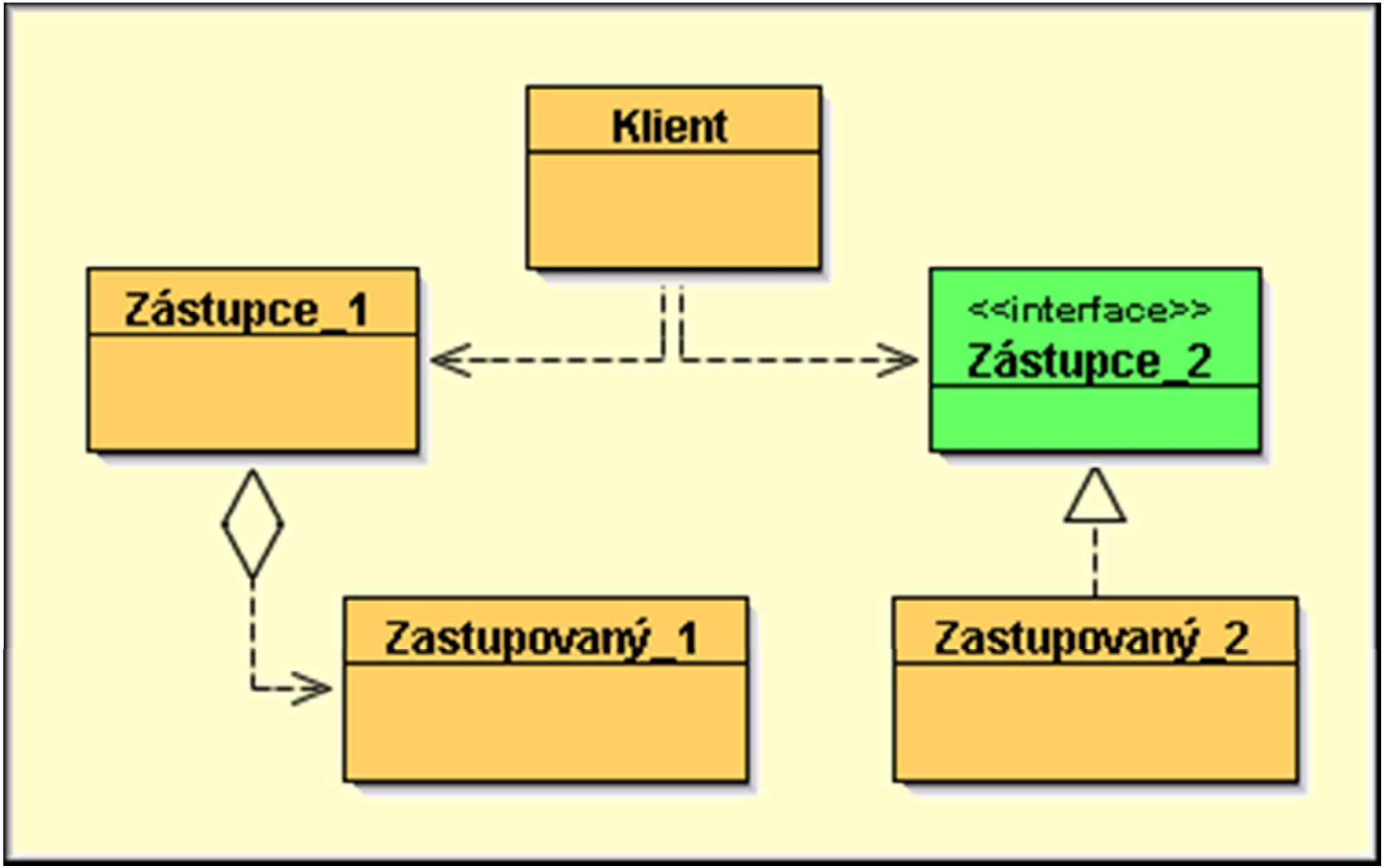
Implementace

- Zástupce obaluje zastupovaný objekt
  a zprostředkovává komunikaci okolního programu
  se zastupovaným objektem

- V převážné většině případů je odkaz
  na zastupovaný objekt atributem zástupce,
  kterému jsou po případné kontrole
  předány požadavky a naopak
  obdržené výsledky jsou předány žadateli

- V případě ochranného zástupce lze při mírnějších
  požadavcích na ochranu (nehrozí-li záměrný útok)
  „skrýt“ zastupovaný objekt za pouhé rozhraní

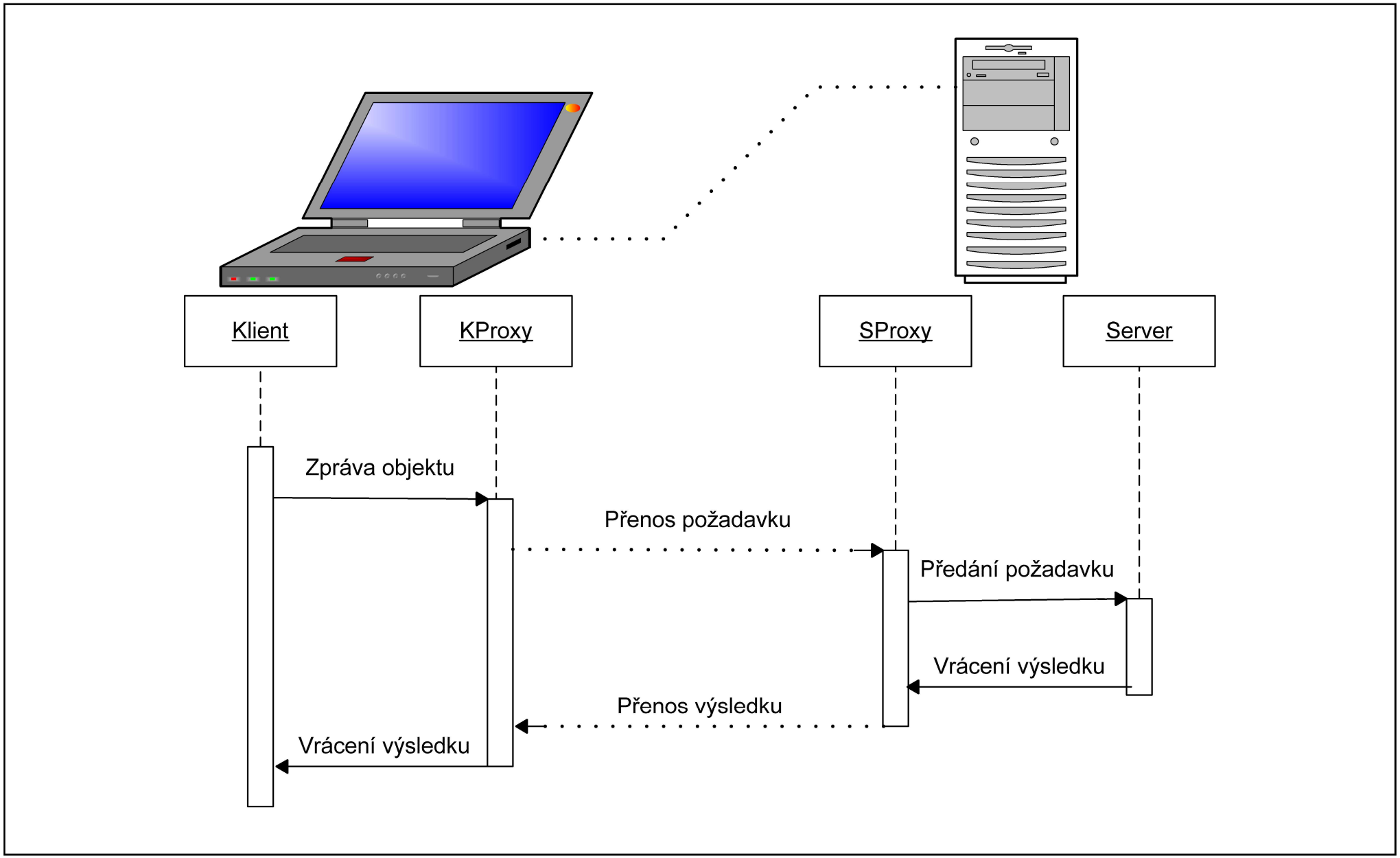
Diagram tříd zástupců



Vzdálený zástupce (Remote proxy)

- Zastupuje objekt umístěný jinde (často na jiném VM)
- Zprostředkovává komunikaci mezi zastupovaným vzdáleným objektem a objekty z okolí zástupce
- Úkol: zapouzdřit a skrýt detaily komunikace
  - Klienti komunikují se zastupovaným objektem jako by byl místní
  - Klient nemusí vůbec poznat, v kterém okamžiku je zastupován místní objekt a ve které je zastupován skutečně vzdálený objekt
- Musí být stále připraven na možnost selhání komunikace a být schopen vyhodit výjimku

Vzdálený zástupce – schéma



Ochranný zástupce (protection proxy)

- Umožňuje zatajit skutečný typ zastupovaného objektu
- Definuje pouze ty metody, které má objekt umět
- Může doplnit kontrolu přístupových práv
- Možné implementace
  - Skrýt skutečnou třídu za rozhraní – použitelné, pokud chceme pouze zjednodušit (tj. zpřehlednit) rozhraní
  - Vytvořit skutečného zástupce, tj. objekt, který obsahuje odkaz na zastupovaný objekt, jemuž předává zprávy a od nějž přebírá odpovědi – potřebujeme-li ochránit před možným útokem

Ukázka kódu ochranného zástupce

```
public class Zástupce
{
   private Zastupovaný z;

   Zástupce( Zastupovaný zz ) {
      z = zz
   }

   Zástupce( Object... parametry ) {
      z = new Zastupovaný( parametry );
   }

   public int metoda( Object... parametry ) {
      return z.metoda( parametry );
   }
}
```

Virtuální zástupce (virtual proxy)

- **Použijeme jej, když je vytváření objektu drahé a objekt ve skutečnosti není potřeba od začátku celý**
- **Zástupe vytvoří zastupovaný objekt (např. načte obrázek), až když je doopravdy potřeba**
- **Do okamžiku skutečného vytvoření může**
  - Nahrazovat volání metod zastupovaného objektu voláním vlastních náhradních metod (např. dotaz na požadovaný paměťový prostor pro budoucí obrázek)
  - Připravit a uchovávat seznam konfiguračních parametrů, které se použijí v okamžiku skutečného vytvoření objektu

Chytrý odkaz (smart reference)

- **Umožňuje doplnit komunikaci s objektem o další akce**
  - Kontrola přístupových práv k objektu
  - Evidence požadavků na služby objektu
  - Zamčení objektu při zápisu
- **Umožňuje zefektivnit práci s objektem**
  - Při první žádosti o objekt se objekt nevytváří, ale zavádí se do paměti dříve vytvořený objekt uložený v nějaké vnější paměti
  - Udržuje spojení s databází ještě chvíli po poslední žádosti
- **Virtuální zástupce je druh chytrého odkazu**
  - Musí se umět rozhodnout, na co stačí sám, kdy už je třeba zastupovaný objekt skutečně vytvořit

Příkaz (Command) 032

- Zabalí metodu do objektu, takže s ní pak lze pracovat jako s běžným objektem. To umožňuje dynamickou výměnu používaných metod za běhu programu a optimalizaci přizpůsobení programu požadavkům uživatele.

Motivace

- **Občas víme, že se v nějaké situaci má něco provést, ale při psaní programu nemáme ani vzdálenou představu o tom, co to bude**
  - Víme, že při stisku tlačítka má program nějak zareagovat, ale až při vložení konkrétního tlačítka do konkrétního GUI budeme tušit, jaká je ta správná reakce. Při definici objektu `Tlačítko` o tom však nemáme ani vzdálené tušení
- **Chceme oddělit objekt, který spouští nějakou akci, od objektu, který ví, jak ji vykonat**
- **Potřebovali bychom, aby se s akcemi mohlo pracovat stejně jako s daty**
  1. Připravíme „proměnnou“
  2. Až budeme vědě, co se má dělat, vložíme do proměnné kód
  3. Až budeme potřebovat kód provést, vytáhneme jej z proměnné a provedeme

Implementace

- Vzor doporučuje zabalit akci (skupinu akcí) do objektu a tím převést akci na data
- Definujeme rozhraní specifikující charakteristiku (signaturu) požadovaných metod
- Vytvoříme objekty implementující toto rozhraní a realizující jednotlivé typy akcí
  - Příslušné třídy mívají jedinou instanci, i když často nebývají definovány jako jedináčci
- Podle toho, jakou akci je třeba realizovat, naplní se vyhrazená proměnná odkazem na příslušný akční objekt

Příkaz × Služebník

- **Služebník:**
  Mám obsluhované objekty a hledám někoho,
  kdo by je „obsloužil“,
  tj. kdo by s nimi (nad nimi) provedl požadovanou operaci

- **Příkaz:**
  Mám připravený, resp. připravuji příkaz (akci, obsluhu)
  který vyžaduje provedení nějaké akce, resp. nějakých akcí.
  Definuji rozhraní (`interface`) s požadavky na objekt,
  který by vykonání požadované akce zabezpečil

Iterátor (Iterator) *033*

- Zprostředkuje jednoduchý a přehledný způsob sekvenčního přístupu k objektům uloženým v nějaké složité struktuře (většinou v kontejneru), přičemž implementace této struktury zůstane klientovi skryta.

Motivace

- Instance, které ukládáme do kontejneru, neodkládáme jako do popelnice – budeme je chtít v budoucnu použít
- Kontejner nás však nemůže nechat se ve svých útrobách přehrabovat – to by nám musel prozradit svou implementaci
- Potřebujeme mít možnost se kdykoliv dostat k uloženým datům, aniž by kontejner byl nucen cokoliv prozradit o své implementaci
- Problém řeší aplikace návrhového vzoru *Iterátor*

Charakteristika

- **Skryje způsob uložení objektů v kontejneru a přitom umožní procházet kontejnerem a prácovat s uloženými prvky**

- **Sekvenční (externí) iterátor**
  - Na požádání předává externímu žadateli jednotlivé uložené objekty
  - Vlastní akci iniciuje a zabezpečuje klient

- **Dávkový (interní) iterátor**
  - Převezme od klienta filtr specifikující ošetřované prvky a příkaz, který se má na každý ošetřovaný prvek aplikovat
  - Sám prochází prvky a na ty, které projdou filtrem, aplikuje obdržený příkaz

Princip externího iterátoru

- Kontejner definuje speciální třídu – iterátor,
  jejíž instance ví, jak jsou svěřená data uložena
- Tomu, kdo chce pracovat s uloženými daty
  vrátí kontejner na požádání instanci iterátoru,
  jenž mu přístup k uloženým datům zprostředkuje
- Instance iterátoru na požádání
  vrátí odkaz na další z instancí uložených v kontejneru
- Až iterátor všechna data vyčerpá, oznámí, že už další nejsou
- Tazatel se tak dostane ke všem uloženým datům,
  aniž by se dozvěděl, jak jsou vlastně uložena

Implementace

- Iterátor bývá v Javě implementován jako vnitřní třída
- Rozhraní `Iterator` ze standardní knihovny vyžaduje implementaci metod:
  - `boolean hasNext()`
  - `Object next()`
  - `void remove()` (implementace může být formální)
- Seznamy umí vrátit také `ListIterator`, který přidává
  - Možnost vložení nového prvku před či za aktuální
  - Změnu směru průchodu seznamem
  - Nahrazení naposledy vráceného prvku jiným
  - Vrácení indexu předchozího, resp. následujícího prvku

interface Iterator

```
package java.util;

public interface Iterator<E>
{
    /** Je ještě nějaká instance k dispozici? */
    boolean hasNext();

    /** Vrátí odkaz na další instanci. */
    E next();

    /** Odebere z kolekce
     *  naposledy vrácenou instanci.
     *  Metoda nemusí být plně implementována. */
    void remove();
}
```

Rozhodnutí při definici iterátoru

- **Kdo řídí iteraci: klient × kontejner (externí × interní)**
  - Externí iterátor je tvárnější (lze např. porovnat 2 kolekce), interní iterátor umožňuje zabezpečit větší robustnost
- **Míru robustnosti iterátoru**
  - Jak citlivý bude na změnu struktury kontejneru v průběhu iterace
- **Jakou množinu operací bude poskytovat**
  - Viz `Enumeration` × `Iterator` × `ListIterator`

Stav (State) 034

- Řeší výrazný rozdíl mezi chováním objektu v různých stavech zavedením vnitřního stavu jako objektu reprezentovaného instancí některé ze stavových tříd. Změnu stavu objektu pak řeší záměnou objektu reprezentujícího stav.

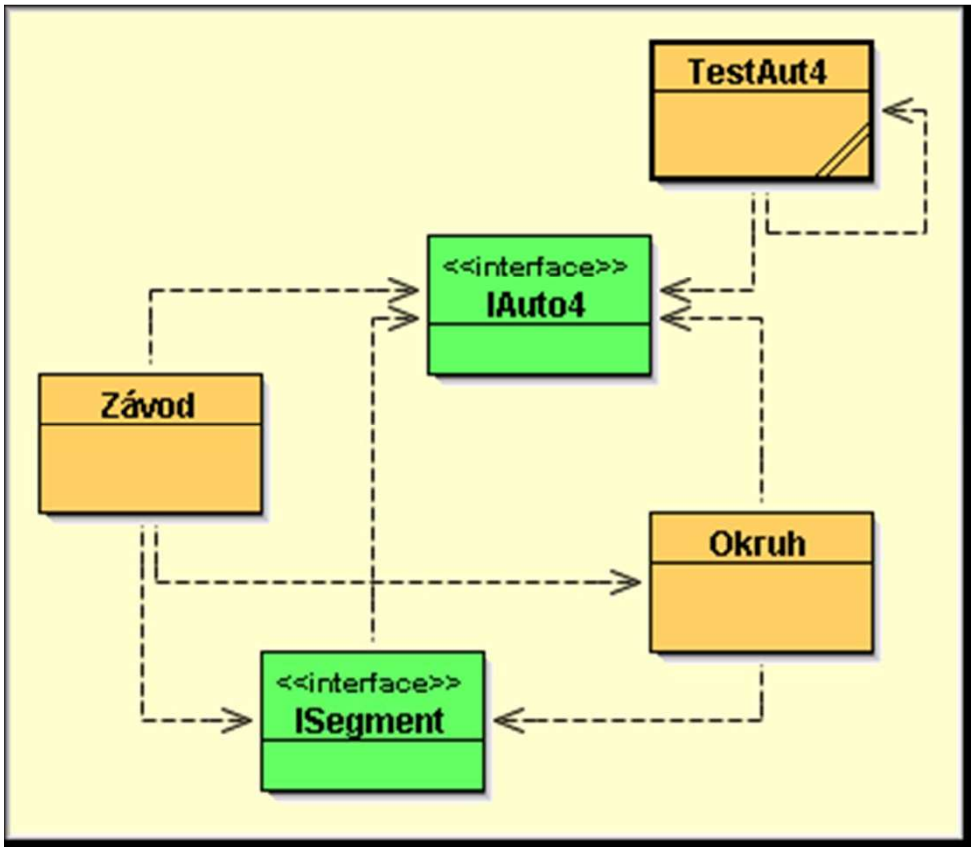
Motivace

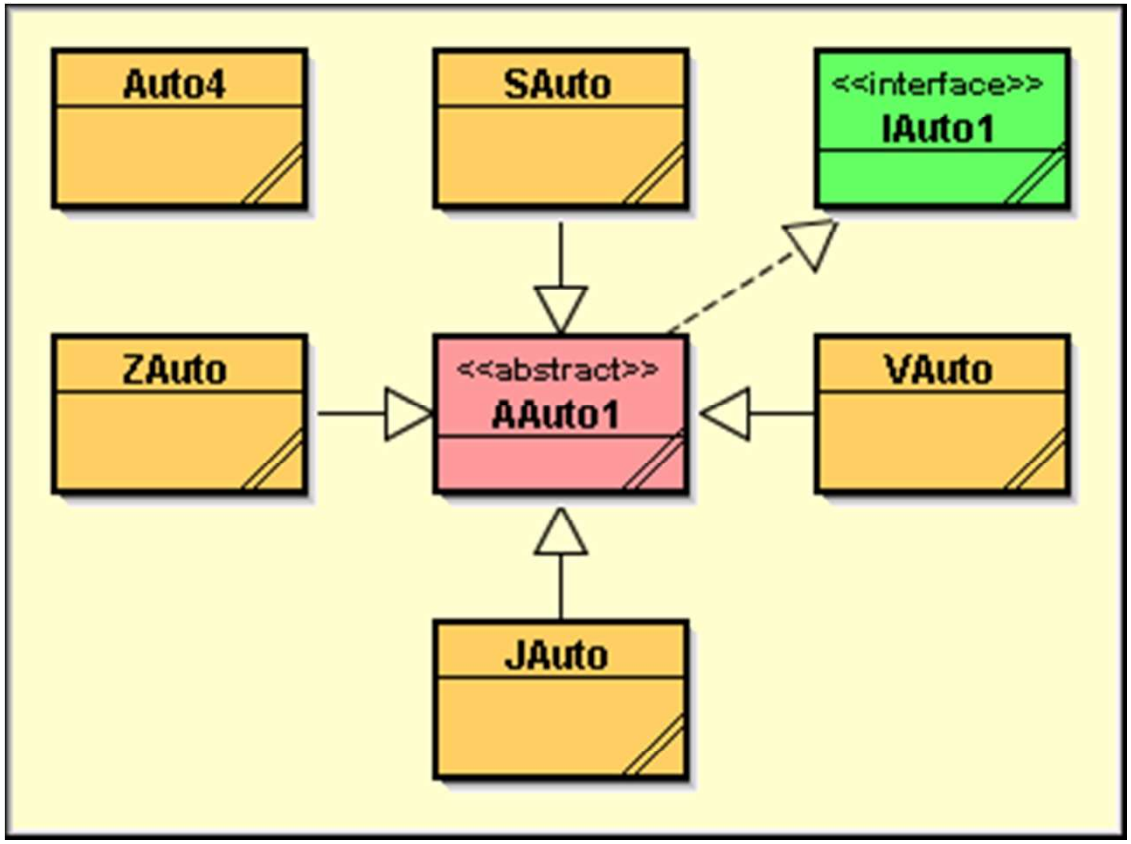
- **Chování objektu se výrazně liší v závislosti na stavu, v němž se právě nachází**
- **Při klasickém přístupu bylo třeba v každé stavově závislé metodě definovat rozhodovací sekvenci s popisy příslušné reakci v každé větvi**
- **Nevýhody:**
  - Kód je dlouhý a nepřehledný
  - Zavádění nových stavů je obtížné
  - Při modifikaci reakce v konkrétním stavu je třeba najít příslušné pasáže v záplavě okolního kódu

Implementace

- **Definice objektu se rozdělí na dvě části:**
  - Stavově nezávislou obsahující metody nezávisející na stavu
  - Stavově závislou s metodami obsahujícími rozhodovací sekvence definující reakce se v závislosti na stavu
- **Definuje se rozhraní (interface nebo abstraktní třída) deklarující metody s reakcemi závisejícími na stavu**
- **Pro každý stav se zavede samostatná třída implementující dané jednostavové rozhraní a definující chování objektu nacházejícího se v daném stavu**
- **Multistavový objekt definuje atribut odkazující na jednostavový objekt, na nějž pak deleguje reakce na zprávy závisející na aktuálním stavu**

Příklad: rup.cesky.vzory._17_stav.autoa





- **Použití abstraktní rodičovské třídy umožňuje:**
  - „Vytknout“ společné atributy
  - Definovat na jednom místě metody pro přechody mezi stavy
  - Definovat tovární metodu vracející instanci požadovaného stavu

Šablonová metoda (Template Method) 035

- Definuje metodu obsahující kostru nějakého algoritmu. Ne všechny kroky tohoto algoritmu jsou však v době vzniku šablony známy – jejich konkrétní náplň definují až potomci třídy se šablonovou metodou prostřednictvím překrytí metod, které šablonová metoda volá.

Charakteristika

- Návrhový vzor používaný i v učebnicích a kurzech, které se o návrhových vzorech vůbec nezmiňují
- Umožňuje podtřídám měnit části algoritmu bez změny samotného algoritmu
- Používá se při řešení typických úloh, jejichž přesné parametry budou známy až za běhu
- Umožňuje definovat metody, jejichž chování je definováno jen částečně; tyto části chování definují až potomci
- Jeden ze způsobů odstranění duplicit v kódu
- Takto bývají definovány klíčové objekty knihoven a rámců
  - Aplety
  - MIDlety

Implementace

- **Definuje metodu obsahující kostru nějakého algoritmu, u nějž však v době konstrukce ještě nejsou všechny jeho kroky známy**
- **Konkrétní náplň neznámých kroků definují až potomci na základě svých speciálních dodatečných znalostí**
- **Třída se šablonovou metodou definuje příslušnou kostru a pro dosud neznámé postupy definuje virtuální metody, které potomci překryjí svými vlastními**
  - Je-li jedno zmožných řešení např. „nedělat nic“, je možno definovat virtuální metodu jako prázdnou
  - Neexistuje-li žádné přijatelné implicitní řešení, definuje se metoda jako abstraktní
- **Při aplikaci vzoru bývá často použit také návrhový vzor *Příkaz***

Template method – Velká a malá ryba

- **Scénář: „Velké“ a „Malé“ ryby plavou oceánem**
  - ryby se pohybují náhodně
  - velká ryba může plavat tam kde je malá (a sníst ji)
  - malá ryba nemůže plavat tam kde je velká

```
public void move() {
   vyber náhodný směr;
   najdi místo vtom směru;
   //odlišné pro druh ryby
   ověř lze-li tam plout;
     jestli ano, pluj;
}
```