Programovací techniky

2007/2008

přednášky: Ing. Pavel Mautner, Ph.D. (UK415)

cvičení: Ing. Roman Mouček, Ph.D. (UK415)
Ing. Tomáš Nestorovič, (UL 410)
Ing. Richard Lipka, (Ul408)

Plán přednášek PT -2007/2008

1. **Úvod, organizace přednášek a cvičení, plán přednášek a cvičení. Základy OOP návrhu, psaní programů v Javě. Dědičnost, polymorfismus, interface v Javě.**
2. **Analýza a návrh software.**
3. **Modelování v UML.**
4. **Základní abstraktní datové typy, zásobník fronta, seznamy,řady, vektory. Tabulky s rozptýlenými položkami, vyhledávání v tabulkách.**
5. **Stromové struktury (Avl,B,Red-Black) fólie1, fólie2, fólie3 aplet redblack, AVL_aplet, Btree_aplet, Btree**
6. **Skip-list., datová struktura Trie**
7. **Grafy a grafové algoritmy.**
8. **Algoritmy zpracování textů I. - vyhledávání podřetězců (BF, KMP, Boayer-Moore algoritmus),**
9. **Algoritmy zpracování textů II. - nejmenší společný podřetězec (LCS problém), vzdálenost mezi řetězci, výpočet vzdálenosti mezi řetězci**
10. **Komprese dat I. komprese textů (RLE, Huffman,Aritmetické kódování, LZW).**
11. **Komprese dat II. (JPEG,waveletová, fraktálová komprese).**
12. **Úvod do programování mobilních zařízení Java2ME**
13. **Weka - programový systém v jazyce Java (strojové učení, předzpracování dat).**

Plán cvičení

1. úvodní informace
2. informace k zadání samostatné práce, pokyny pro vypracování, obecné principy psaní dokumentů, výběr zadání samostatné práce a termínů odevzdání
3. OOP - zapouzdření, dědičnost, polymofismus, příklady
4. konzultace k zadání samostatné práce
5. základy UML - příklady
6. stromy, tabulky
7. průběžná kontrola samostatné práce
8. průběžná kontrola samostatné práce
9. zpracování řetězců, kódování a kompres

10 - 13 (26.11. - 21.12.) termíny prezentací a odevzdání samostatných prací

Literatura:

Goodrich, Tamassia: **Data Stuctures and Algorithms in Java**,

Herout, P. : **Java – bohatství knihoven**,

Rychlík, J. : **Programovací techniky**

Wirth, N. : **Algoritmy a štruktury údajov**

Informace ke zkoušce

- Zkoušku je možné vykonávat pouze po získání zápočtu. Zápočet nemusí být zapsán v indexu, postačující je informace o získání zápočtu zveřejněná na webu cvičícími.

- Zkouška je písemná, ke zkoušce je nutné se přihlásit/odhlásit prostřednictvím studijní agendy ZČU – STAG. **Studenti, kteří se bez řádné omluvy ke zkoušce nedostaví budou klasifikováni známkou nevyhověl.**

- Zkoušková písemka obsahuje obvykle 6 příkladů z uvedených tématických okruhů, maximální počet bodů získaných ze zkoušky je 100 minimální počet bodů je 50. **Písemka s nižším počtem bodům než 50 bude klasifikována jako nevyhovující bez ohledu na bodový zisk získaný při cvičeních.**

- Doba trvání zkoušky je 2 hodiny od okamžiku zadání. V průběhu zkoušky **je možné používat libovolné tištěné/psané pomůcky, které si student přinese** (knihy, přednášky, poznámky ze cvičení), pomůcky však není možné si vyměňovat ani půjčovat.

Informace ke zkoušce

- **Notebooky, PDA ani jiná výpočetní technika (kromě kalkulaček se základními funkcemi) není v průběhu písemné zkoušky povolena.**

- **Výsledky zkouškových písemných prací** budou zveřejňovány ihned po opravení prací (do 3 pracovních dnů) na portále (popř. na předem oznámené webové adrese) pod datem, kdy zkouška byla konána. Rozbor prací, ústní dozkoušení v případech sporných či neúplných výsledků a zápis známek do indexů budou pak prováděny **v termínu zveřejněném na konci** výsledkového seznamu.

- **Výsledná známka** - body získané v průběhu semestru se vynásobí koeficientem 0,5 (max. 60 bodů) a přičtou se k nim body za písemnou část zkoušky; celkový počet bodů bude transformován na výslednou známku podle klíče:

160 – 135	**výborně**
134 – 115	**velmi dobře**
114 – 75	**dobře**
74 a méně	**nevyhověl**

Objektově orientovaný návrh

Cíle objektově orientovaného návrhu

Cílem objektově orientovaného návrhu je vytvoření kvalitního software, který má následující vlastnosti:

- **robustnost** – cílem je vytvořit software, který je schopen správně reagovat i na neočekávané vstupy, které nejsou explicitně definovány pro danou aplikaci

- **Přizpůsobivost** (adaptability) – schopnost software běžet s minimálními úpravami i na jiné platformě (Windows, Unix, MacOs)

- **Znovupoužitelnost** (reusability) – cílem je vytvořit sw nebo jeho část tak, aby byla opětovně použitelná v jiných systémech. Vytváření programů se pak podobá stavebnici, kdy z hotových dílů vytváříme funkční celek

Principy objektově orientovaného návrhu

Mezi nejdůležitější principy objektově orientovaného návrhu, které vedou ke splnění uvedených cílů patří:

- **Abstrakce**
- **Zapouzdření (encapsulation)**
- **Modularita**

Abstrakce – cílem je rozdělit složitý problém na základní části, tyto části popsat a implementovat. Příkladem abstrakce mohou být **abstraktní datové typy** (ADT) – matematický model datových struktur, který specifikuje v jakém datovém typu budou uložena data a jaké operace lze nad těmito daty provádět. ADT v Javě = třída

Zapouzdření – cílem je ukrýt implementační detaily které nejsou pro uživatele podstatné a poskytnout uživateli určité rozhraní pomocí kterého může přistupovat k datovým strukturám. Chrání datové struktury před neoprávněnou manipulací

- **Modularita** znamená rozdělení softwarového systému do oddělených funkčních jednotek (modulů). Moduly mohou být navrženy tak aby byly využitelné v jiných systémech (princip znovupoužitelnosti).

Při návrhu složitého software obvykle skládáme celek z jednodušších komponent. Pro „lepší“ využití jednotlivých komponent využívá objektově orientovaný návrh principů **dědičnosti** a **polymorfizmu**

- **Dědičnost** - umožňuje vytvořit tzv. generické třídy, ze kterých je možné odvozovat třídy další přidávat do nich nové metody, popř. modifikovat metody existující. Dědičnost se využívá zejména proto abychom se vyhnuli nadbytečnému programovému kódu (popř. nadbytečným datovým strukturám)
- **Polymorfismus** (mnohotvarost) – umožňuje, aby stejné metody vykonávaly různou činnost v závislosti na objektu nad kterým pracují

Psaní programů v Javě (popř. jiných programovacích jazycích)

3 základní kroky :

1. **Návrh**
2. **Implementace a kódování**
3. **Testování, ladění, optimalizace**

Návrh:

nejdůležitější krok celého procesu vytváření software.

Cílem je vytvořit třídy a objekty. **ale jak ????**

- Jak identifikovat **třídy**?
- jak určit, jaké **atributy** mají mít?
- jak určit, jaké **metody** mají mít?
- jak tyto skutečnosti zachytit? - jedním ze způsobů jsou tzv. **CRC** karty

CRC-karty

CRC (Class-Responsibility-Collaborator) Card

- jednoduchá, ale silná OO modelovací technika
- CRC tým – uživatelé, analytici, vývojáři

  CRC model je množinou CRC karet, na zadní straně – podrobnější popis třídy

Class	
Responsibilities (odpovědnosti)	**Collaborators** (spolupracovníci)

Třídy a odpovědnost tříd

Kandidáti na třídu – vybrat podstatná jména

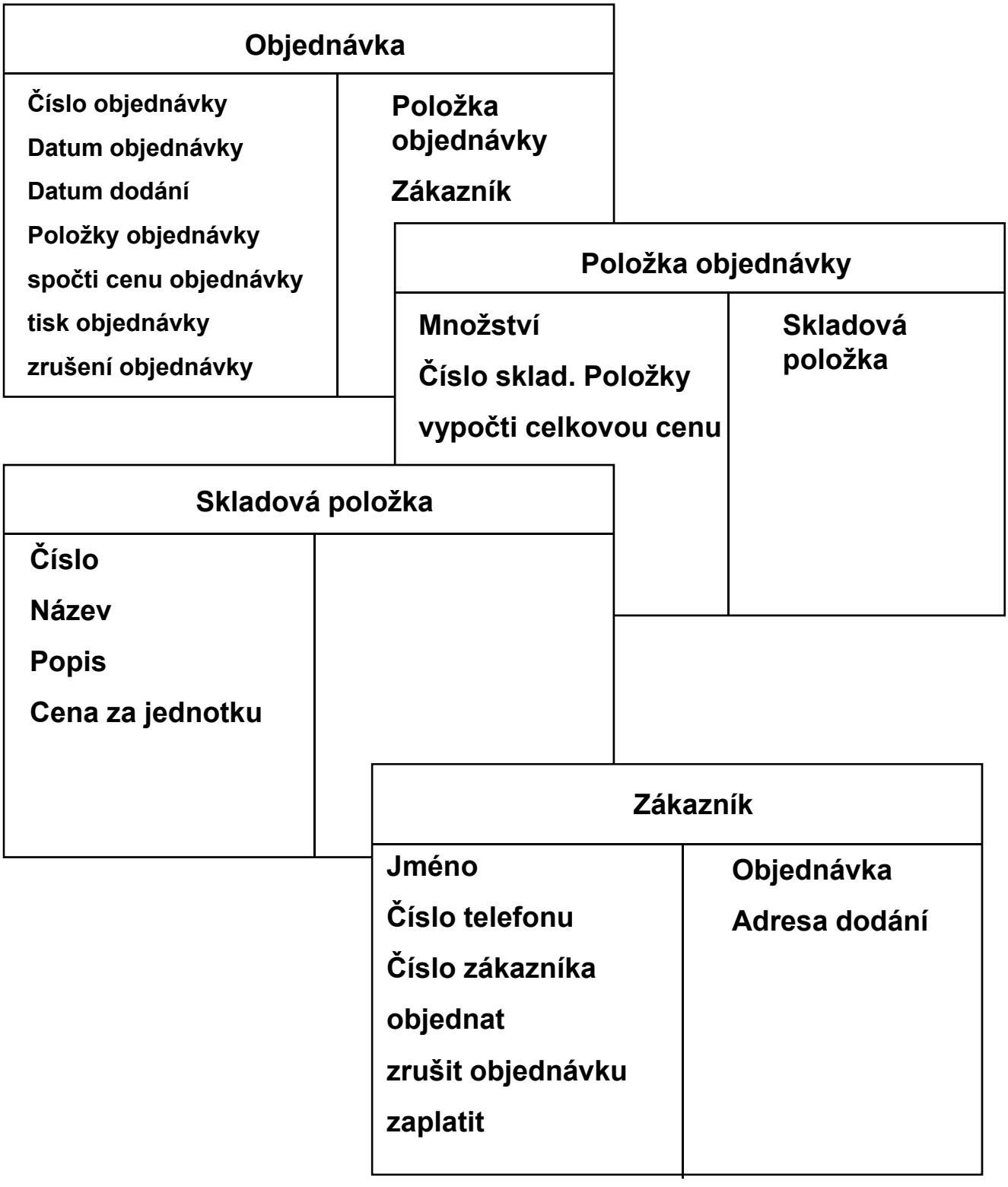
Odpovědnost

- něco, co třída „zná“ nebo „dělá“
- Například
  - třída ***Osoba*** má odpovědnost za své **jméno**, **adresu**, **rodné číslo**
  - třída ***Auto*** má odpovědnost za **velikost**, **počet dveří**, **může zastavit**, **jet**
- zjišťování odpovědností – vybrat slovesa

Collaborator (spolupracovník)

- Spolupracovník třídy A je jiná třída, která je využívána při realizaci určité odpovědnosti třídy A
- spolupracovník je třeba,
  - pokud třída potřebuje informace, které sama nemá
  - pokud potřebuje změnit informace, které nemá – požádá jinou třídu, aby to udělala
- při spolupráci je jedna třída iniciátorem

CRC model objednávek



Příklad spolupráce

- třída *Objednávka* má odpovědnost – *spočti cenu objednávky*
- ví sice o svých položkách, ale neví, jaké množství bylo objednáno, to ví třída *Položka objednávky*, proto musí třída *Objednávka* spolupracovat s třídou *Položka objednávky*, aby vypočítala celkovou cenu objednávky
- *Položka objednávky* zase pro výpočet ceny položky nezná cenu za jednotku, proto musí spolupracovat s třídou *Skladová položka*

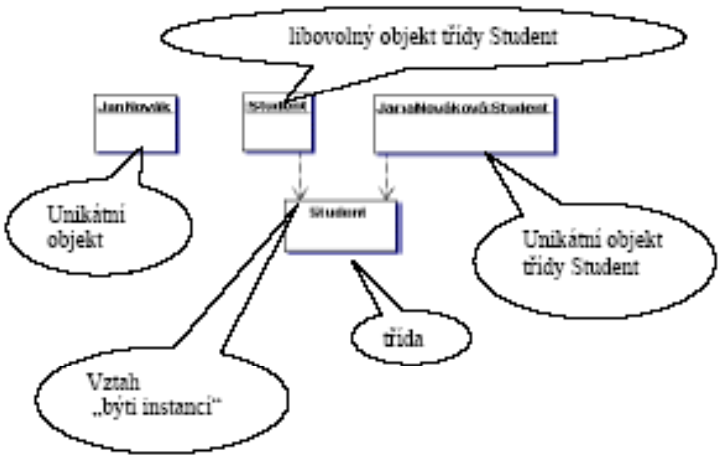
Jazyk UML, UML diagramy

- UML – unifikovaný modelovací jazyk
- Vznikl jako prostředek pro objektově-orientovanou analýzu a návrh (G. Booche, J. Rumbaugh, I. Jacobson)
- Skládá se z grafických prvků, které se dají vzájemně kombinovat do podoby diagramů. Účelem diagramů je vytvořit určité pohledy na navrhovaný systém.
  - Skupina pohledů se nazývá model
  - Model jazyka UML popisuje co má systém dělat, ale neříká jak to implementovat

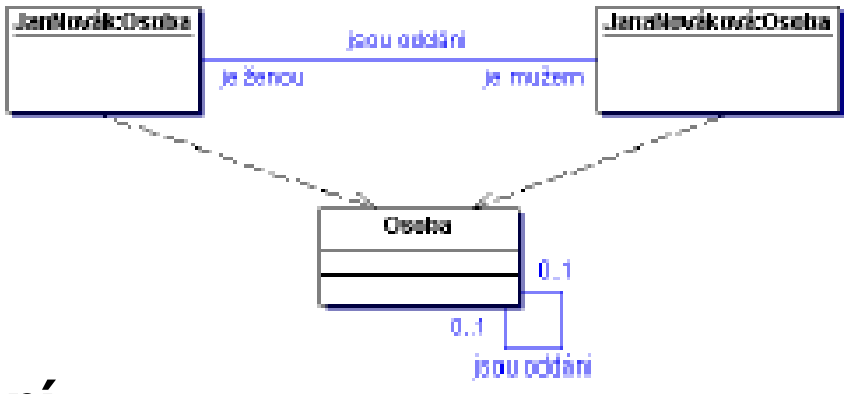
UML zahrnuje definici 8-mi typů diagramů, tj. 8 různých pohledů na model systému:

- **diagramy tříd a objektů** –popisují statickou strukturu systému, znáyorňují datový model datový model systému od konceptuální úrovně až po implementaci
- **modely jednání** (diagramy případů užití) dokumentují možné případy použití systému události, na které musí systém reagovat,
- **scénáře činností** (diagramy posloupností) – popisují scénář průběhu určité činnosti systému,
- **diagramy spolupráce** – zachycují komunikaci spolupracujících objektů,
- **stavové diagramy** – popisují dynamické chování objektu nebo systému,
- **diagramy aktivit** – popisují průběh aktivit procesu nebo činností
- **diagramy komponent** – popisují rozdělení výsledného systému na funkční celky a definují náplň jednotlivých komponent,
- **diagramy nasazení** – popisují umístění funkčních celků (komponent) na výpočetní uzly informačního systému

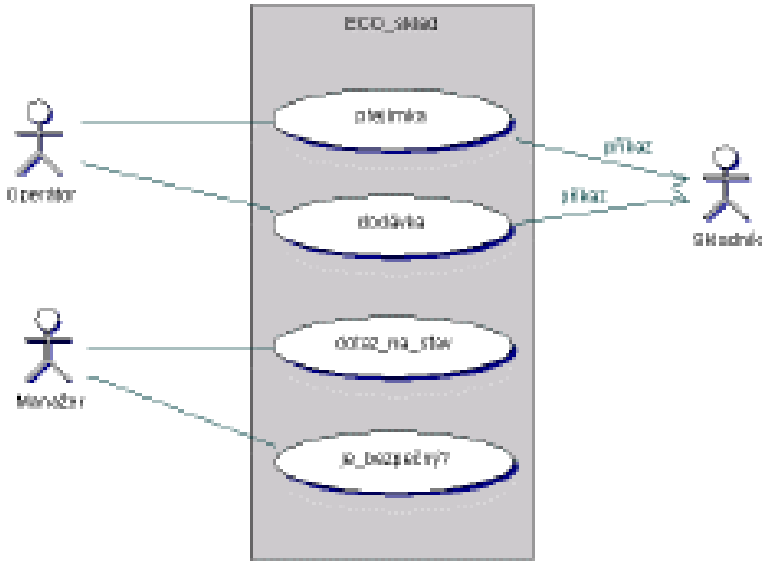
- **Diagram objektů**



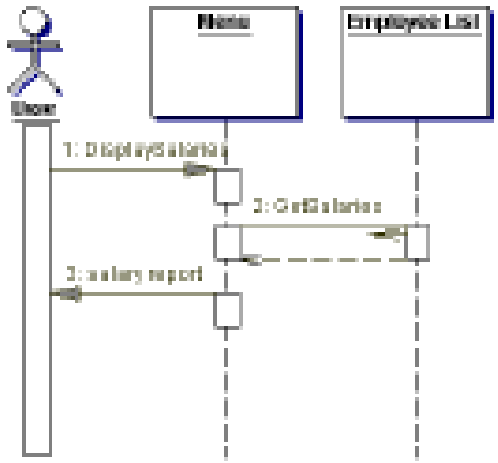
- **Diagram tříd**



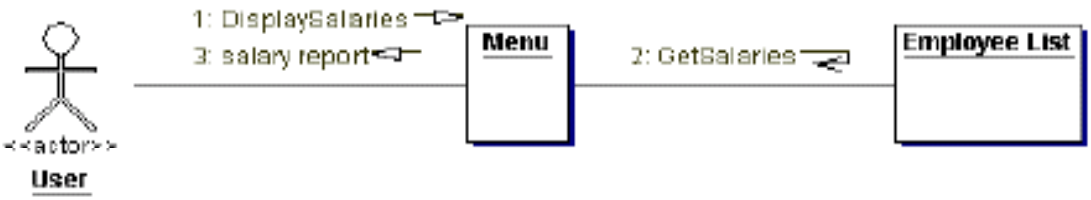
- **Model jednání**



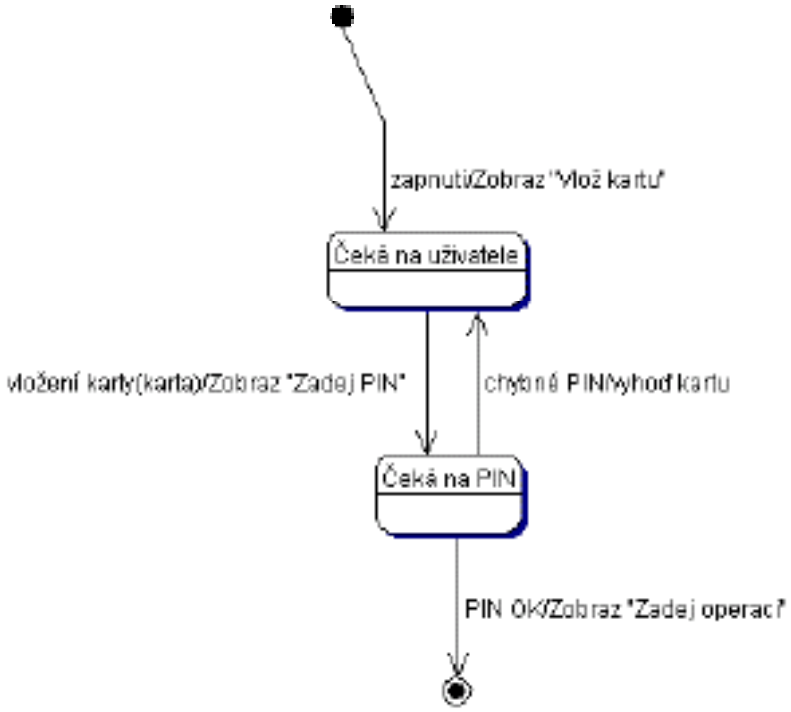
- **Scénáře činností**



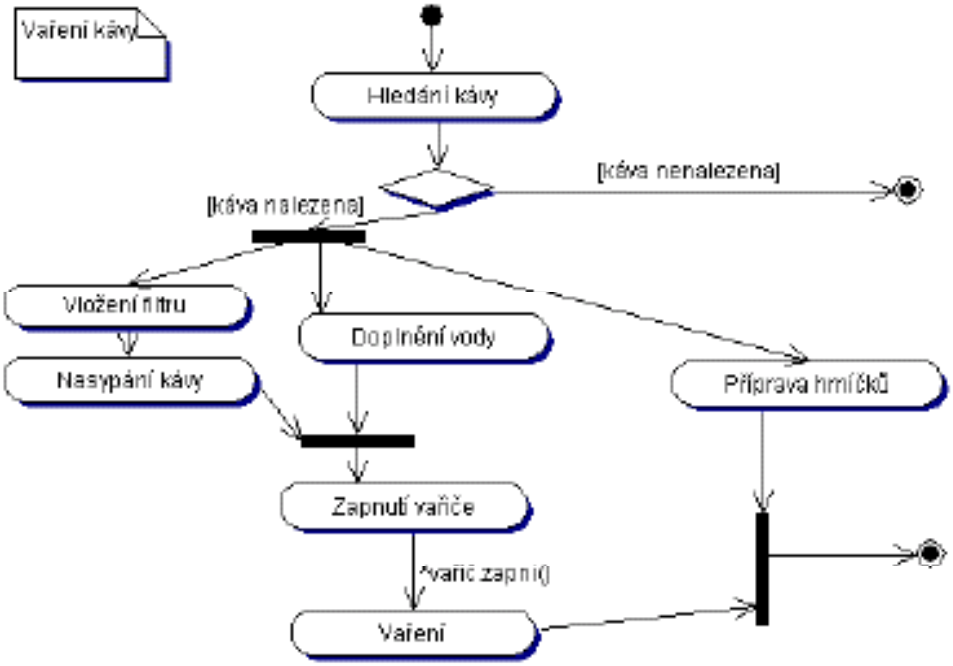
- **Diaramy spolupráce**



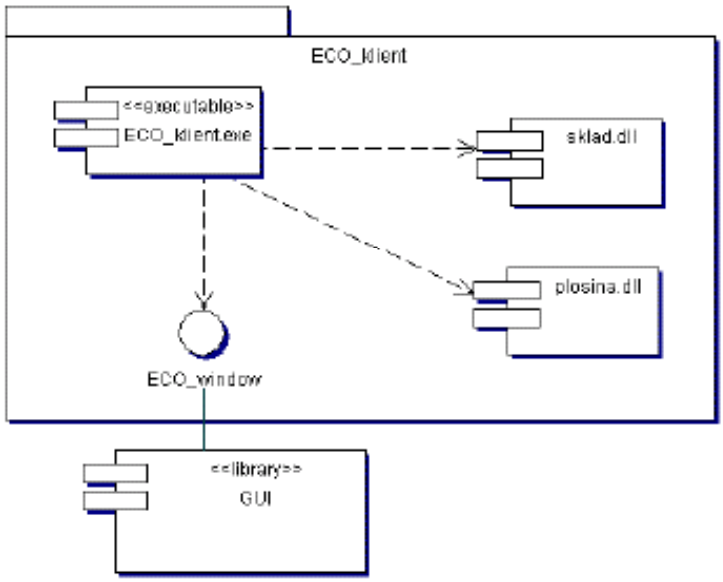
Stavový diagram



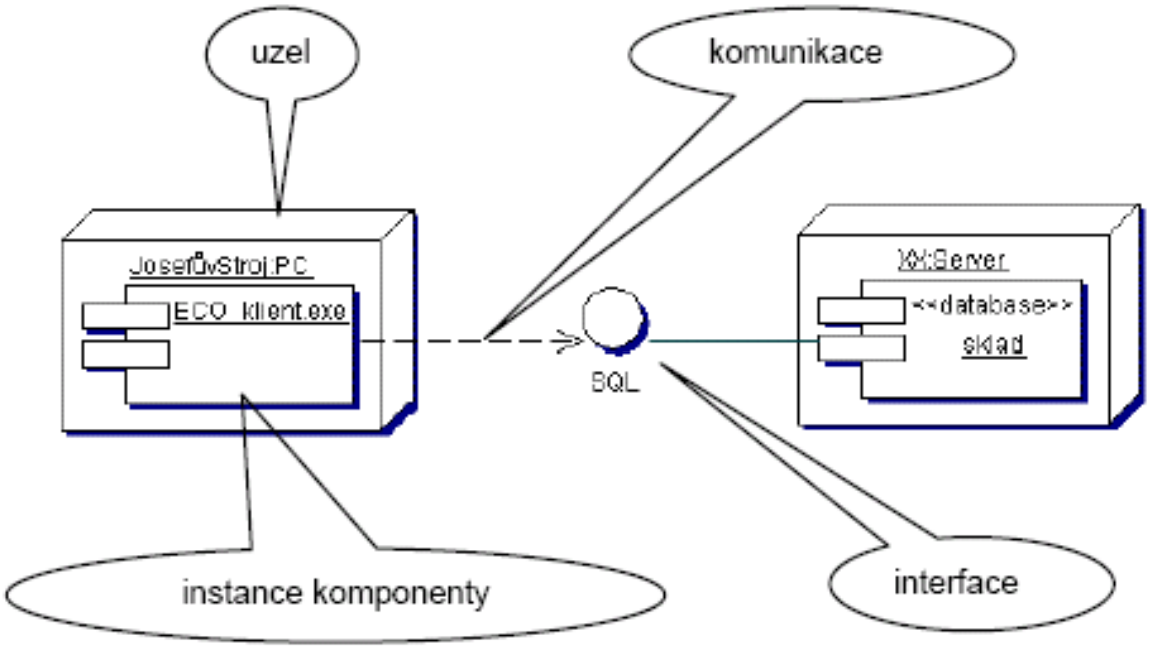
- **Diagram aktivit**



- **Diagram komponent**



- **Diagram nasazení**



Kódování:

Jakmile máme vytvořen návrh tříd je možné zapsat třídy v programovacím jazyce Java – buďto pomocí textového editoru nebo použitím integrovaného vývojového prostředí (JBuilder, JCreator, eclipse atd.)

Při zápisu programu dodržovat následující doporučení :

- používat smysluplné názvy identifikátorů
- jména tříd začínat s velkým písmenem (**Datum**,**Vektor**, atd.)
- jména metod a proměnných začínat s malým písmenem (**vlozPolozku(), jmenoStudenta,** atd.)
- pokud má proměnná charakter konstanty definovat jí jako konstantu (**final**) a název identifikátoru konstanty psát s velkými písmeny (**MAX_HODNOTA, MAX_POCET_PRVKU**)
- odsazovat příkazové bloky
- v každé třídě dodržovat následující pořadí deklarací
    - **Konstantní atributy třídy, tj. statické konstanty**
    - **Ostatní atributy třídy**
    - **Konstantní atributy instancí**
    - **Ostatní atributy instancí**
    - **Přístupové metody atributů třídy**
    - **Ostatní metody třídy**
    - **Konstruktory a metody vracející odkaz na instance třídy**
    - **Přístupové metody atributů instancí**
    - **Ostatní metody instancí**
    - **Testovací metody**
- používat komentáře zvlášť případech kde se vyskytují nejednoznačnosti, „podivné“ konstrukce atd. Komentáře psát nejlépe ve stylu javadoc

Testování a ladění:

- **Ladění**
  - nejjednodušší způsob – kontrolní výpisy použitím System.out.print(<String>)
  - použití speciálního programu - debuggeru - standardní jdk obsahuje základní řádkově orientovaný debugger **jdb**, vývojová prostředí (eclipse, JBuilder) obsahují debuggery s GUI (grafické uživatelské rozhraní)

- **Testování**

Testování a ladění je obvykle časově nejnáročnější činnost. Cílem je ověřit správnost programu a použitých metod popř. odstranit chyby které se během zápisu algoritmu vyskytly .

Testování provádíme na reprezentativní množině vstupů – tu je nutné zvolit tak, aby byla každá metoda třídy alespoň jednou zavolána. Totéž by mělo platit o každém příkazu v programu.

Program často „padá“ v důsledku neočekávaných vstupních hodnot -> obvykle se doporučuje nechat program proběhnou na **rozsáhlé množině náhodně vygenerovaných dat**

Příklad: pro otestování metody řazení pole celočíselných prvků volíme obvykle následující vstupy:

- pole nulové délky (neobsahující žádný vstup)
- pole s jedním prvkem
- pole s prvky stejné hodnoty
- seřazené pole
- pole seřazené v opačném pořadí

Komentáře a dokumentace

Chceme-li aby se program dožil vysokého produktivního věku musíme jej psát tak, aby bylo možno snadno upravit a doplnit o další funkce tj. psát program čitelně.

Čitelnost ovlivňují:

- **Správně volené identifikátory**
- **Komentáře vysvětlující náročnější obraty**

Druhy komentářů v Javě:

- Řádkový komentář začíná dvojicí znaků // a končí koncem řádku - poznámky ke kódu
- Obecný komentář /*…*/
- Dokumentační komentář – speciální typ obecného komentáře – /** … */ - je zpracováván programem javadoc.exe , který vytváří html dokumentaci

Dokumentační komentáře se zapisují těsně před dokumentovaný prvek (javadoc reaguje i na umístění komentářů)

- Komentáře popisující účel a použití třídy – patří před hlavičku třídy
- Komentář popisující účel atributu – patří před deklaraci tohoto atributu
- Komentáře popisující funkci nějaké metody a význam parametru – patří před hlavičku této metody

Pomocné značky pro javadoc:

@author – vkládá se do místa dokumentace k celé třídě. Zapisuje se za mí jméno autora třídy, více autorů je možné psát odděleně nebo za společnou značku

@version – vkládá se do místa dokumentace pro celou třídu a zapisuje se za ní číslo verze. Pro formát verze není žádný předpis

@param – používá se v místě dokumentace konstruktorů, za ní se uvádí přesný název parametru z hlavičky a za něj popis parametru

@returns – používá se v dokumentaci metod, které vracejí nějakou hodnotu, za ní se uvádí podrobný popis návratové hodnoty

@throw – používá se k popisu výjimky a důvodu proč jí metoda „vyhazuje“