Seminář Java

III

Rekapitulace

- Deklarace tříd
- Proměnné, metody, modifikátory přístupu
- Konstruktory
- Datové typy
- Tvorba dokumentace

Dědičnost

Co už víme ...

- Třídy popisují skupiny objektů podobných vlastností
- Třídy mohou mít tyto skupiny vlastností:
  - Metody - procedury/funkce, které pracují (především) s objekty této třídy
  - Proměnné - pojmenované datové prvky (hodnoty) uchovávané v každém objektu této třídy
- Vlastnosti jsou ve třídě "schované" (zapouzdřené)

Dědičnost

- Specializace, rozšiřování funkčnosti třídy.
- Odvození nové třídy od nějaké stávající
- Odvozená (dceřinná) třída
  - má všechny vlastnosti nadtřídy
  - + vlastnosti uvedené přímo v deklaraci podtřídy.

Třída Ucet

```
public class Ucet {
    protected String majitel;
    protected double zustatek;

    public Ucet(String name);
    public void pridej(double castka);
    public void vypisZustatek();
    public void uber(double castka);
    public void prevedNa(Ucet kam, double castka);
    public void prevedNa(Ucet kam);
}
```

Třída KUcet

```
public class KUcet extends Ucet {
    protected double kkorent;

    public KUcet(String name, double kk) {
        super(name);
        kkorent = kk;
    }

    public boolean uber(double castka) {
        if ((zustatek+kkorent) >= castka) {
            super.uber(castka);
            return true;
        }
        else
            return false;
    }
}
```

Inicializace objektu

- nalezení a vyvolání konstruktoru
- vyvolání bezparametrického konstruktoru nadřazené třídy
- inicializace instančních proměnných
- provedení těla konstruktoru třídy

Možné modifikace

- lze volat jiný než bezparametrický konstruktor nadřazené třídy (musí být vždy na začátku konstruktoru potomka), např. super(name)
- lze volat i jiný konstruktor třídy, toto volání může být kdekoliv v konstruktoru, např. this(parametry)
- Pzn.: Bezparametrický (implicitní) konstruktor neexistuje, pokud existuje alespoň jeden jiný.

Inicializace objektu – II

```
class Z {
    public Z() { System.out.println("Konstruktor Z"); }
}

class A {
    public A() { System.out.println("Konstruktor A"); }
}

class B extends A {
    Z z = new Z();
    public B() { System.out.println("Konstruktor B"); }
}
```

```
B b = new B();

Konstruktor A
Konstruktor Z
Konstruktor B
```

Typy omezení přístupu k vlastnostem objektu

Pro vlastnosti tříd = proměnné/metody:

- veřejné - public
- chráněné - protected
  - přístupné jen ze tříd stejného balíku a z podtříd
- neveřejné - lokální v balíku
  - přístupné jen ze tříd stejného balíku, už ale ne z podtříd, jsou-li v jiném balíku) (nedoporučuje se)
- soukromé - private
  - přístupné jen v rámci třídy – používá se častěji pro proměnné než metody
  - zneviditelníme i případným podtřídám

Rozhraní

V Javě, na rozdíl od C++ neexistuje vícenásobná dědičnost

- to nám ušetří řadu komplikací
- ale je třeba to něčím nahradit

Pokud po třídě chceme, aby disponovala vlastnostmi z několika různých množin (skupin), můžeme ji deklarovat tak, že

- implementuje více rozhraní

Co je rozhraní

- Rozhraní je vlastně popis (specifikace) množiny vlastností, aniž bychom tyto vlastnosti ihned implementovali.
- Vlastnostmi zde rozumíme především metody.
- Říkáme, že určitá třída implementuje rozhraní, pokud implementuje (tedy má - přímo sama nebo podědí) všechny vlastnosti (tj. metody), které jsou daným rozhraním předepsány.
- Javové rozhraní je tedy množina hlaviček metod označená identifikátorem - názvem rozhraní (a celých specifikací - tj. popisem, co přesně má metoda dělat - vstupy/výstupy metody, její vedlejší efekty...)

Deklarace rozhraní

- Vypadá i umisťuje se do souborů podobně jako deklarace třídy
- Všechny metody v rozhraní musí být public a v hlavičce se to ani nemusí uvádět.
- Těla metod v deklaraci rozhraní se nepíší.

Příklad deklarace rozhraní

```
public interface Informator {
     public void vypisInfo();
}
```

Implementace rozhraní

```
public class Ucet implements Informator {
      ...
      public void vypisInfo() {
            ...
      }
}
```

- Třída implementuje všechny metody předepsané rozhraním.
- Třída může implementovat více rozhraní současně.

```
public class Name implements Interface1, Interface2

{ ...  }
```

Použití rozhraní

- Tam, kde stačí funkcionalita definovaná rozhraním.
- Proměnnou můžeme definovat jako typ rozhraní (ne třídu, která rozhraní implementuje).
- Do proměnné lze přiřadit libovolný objekt, který **implementuje** uvedené rozhraní.

```
Informator petruvUcet = new Ucet("Petr");
petruvUcet.vypisInfo();
```

- Deklarace, že třída implementuje rozhraní ji nezavazuje, poskytuje typovou informaci o třídě.
- Umožňuje větší flexibilitu kódu při zachování (statické) typové kontroly.

Použití rozhraní

Rozhraní (java.util):

```
public interface Collection ...
```

Implementující třídy:

```
AbstractCollection, AbstractList, AbstractSet, ArrayList,
BeanContextServicesSupport, BeanContextSupport, HashSet,
LinkedHashSet, LinkedList, TreeSet, Vector
```

Třída `Vector`:

```
public class Vector ... {
      ...
      public Vector(Collection c) ...
      ...
}
```

Rozšiřování rozhraní

- Podobně jako u tříd i rozhraní může být *děděno*.
- Třída dědí maximálně z jednoho předka.
- Rozhraní může dědit z více předků (*vícenásobná dědičnost*).

```
public interface DobryInformator extends Informator {
     public void vypisViceInfo();
}
```

Rozšiřování rozhraní

Třída, která implementuje rozhraní *DobryInformator* musí implementovat *obě* metody:

```
public class Ucet implements DobryInformator {
      ...
      public void vypisInfo() {
            ...
      }
      public void vypisViceInfo() {
            ...
      }
}
```

Abstraktní třídy

- Třída, která danou specifikaci implementuje jen *částečně*.

  Rozhraní = specifikace

  Abstraktní třída = částečná implementace

  Třída = úplná implementace

- Abstraktní třída *nemůže* mít instance.

```
public abstract class GraphicObject {
    int x, y;
    . . .
    void moveTo(int newX, int newY) {
        . . .
    }
    abstract void draw();
}
```

Abstraktní třídy

```
class Circle extends GraphicObject {
    void draw() {
        . . .
    }
}

class Rectangle extends GraphicObject {
    void draw() {
        . . .
    }
}
```

Rekapitulace

- Umíme deklarovat třídu a její vlastnosti.
- Umíme vytvářet instance tříd a volat její metody.
- Umíme vytvářet specializované třídy a rozhraní.
- Umíme implementovat rozhraní.

Příkazy v Javě

- Přiřazovací příkaz =
- Řízení toku programu
- Volání metody
- Návrat z metody return
- Příkaz je ukončen středníkem ;

Přiřazení

- Na levé straně musí být proměnná.
- Na pravé straně musí být *přiřaditelný* výraz.
- Primitivní typy
  - přiřazením se hodnota zduplikuje
  - konverze typů (short $\rightarrow$ int, int $\rightarrow$ short)
- Přiřazení odkazu na objekt
  - Proměnné objektového typu obsahují odkazy (reference) na objekty, ne objekty samotné!!!
  - přiřazením se duplikuje pouze reference

Pole – I

- Pole v Javě je speciálním objektem.
- Můžeme mít pole jak primitivních, tak objektových hodnot.
  - pole primitivních hodnot tyto hodnoty obsahuje
  - pole objektů obsahuje odkazy na objekty
- Kromě pole v Javě existují i jiné objekty na ukládání více hodnot - tzn. kontejnery, viz dále
- Syntaxe deklarace
  - typhodnoty [] jménopole
  - na rozdíl od C/C++ nikdy neuvádíme při deklaraci počet prvků pole - ten je podstatný až při vytvoření objektu pole
- Syntaxe přístupu k prvkům
  - jménopole [indexprvku]
  - přiřazení prvku do pole: jménopole [indexprvku] = hodnota;
  - čtení hodnoty z pole: proměnná = jménopole [indexprvku] ;

Pole – II

Syntaxe vytvoření objektu pole: jako u jiného objektu - voláním konstruktoru:

jménopole = new typhodnoty[početprvků];

nebo vzniklé pole rovnou naplníme hodnotami/odkazy

int [] pole = { 1, 2, 3 };

Před použitím je nutné pole:

- deklarovat
- vytvořit
- inicializovat (naplnit)

Pole – III

```
Ucet [] ucty;                        // deklarace pole
ucty = new Ucet[5];                  // vytvoření pole
ucty[0] = new Ucet("Franta");  // vytvoření objektu
                  // a inicializace 1. prvku pole !!!

ucty[0].vypisInfo();                 // přístup k prvku pole
```

- V poli ucty je naplněn 1. prvek odkazem na objekt
- Ostatní prvky zůstali naplněny prázdnými odkazy null.

Pole – IV

Co když vynecháme vytvoření pole?

```
Ucet [] ucty;
ucty[0] = new Ucet("Franta");  // chyba, pole neexistuje
```

Co když vynecháme inicializaci pole?

```
Ucet [] ucty;
ucty = new Ucet[5];
ucty[0].vypisInfo();          // chyba, prvek neexistuje
```

Kopírování pole

Přiřazení proměnné objektového typu (a tedy i polí) vede pouze k duplikaci odkazu, nikoli celého odkazovaného objektu.

```
Ucet [] ucty = new Ucet[5];
Ucet [] ucty2;
ucty2 = ucty;
```

- Proměnná ucty2 obsahuje odkaz na stejné pole jako ucty.

```
Ucet [] ucty2 = new Ucet[5];
System.arraycopy(ucty, 0, ucty2, 0, lidi.length);
```

- Proměnná ucty2 obsahuje kopii původního pole.
- Také arraycopy však do cílového pole zduplikuje jen odkazy na objekty, nevytvoří kopie objektů!

Řídící příkazy

- if
- while
- do – while
- for
- switch
- break, continue

Řízení toku programu v těle metody

Příkaz (neúplného) větvení if

```
if (logický výraz) příkaz
```

- platí-li logický výraz (má hodnotu true), provede se příkaz

Příkaz úplného větvení if - else

```
if (logický výraz)
    příkaz1
else
    příkaz2
```

- platí-li logický výraz (má hodnoty true), provede se příkaz1
- neplatí-li, provede se příkaz2
- větev else se nemusí uvádět
- větvení if - else můžeme vnořovat do sebe

Cyklus s podmínkou na začátku

- Tělo cyklu se provádí tak dlouho, dokud platí podmínka

v těle cyklu je jeden jednoduchý příkaz ...

```
while (podmínka)
    příkaz;
```

... nebo příkaz složený

```
while (podmínka) {
    příkaz1;
    příkaz2;
    příkaz3;
    ...
}
```

- Tělo cyklu se nemusí provést ani jednou - pokud už hned na začátku podmínka neplatí

Doporučení k psaní cyklů/větvení

- Větvení, cykly: vždy psát se složeným příkazem v těle (tj. se složenými závorkami)!!!
- jinak hrozí, že se v těle větvení/cyklu z neopatrnosti při editaci objeví něco jiného, než chceme, např.:

```
while (i < a.length)
    System.out.println(a[i]); i++;
```

Pišme proto vždy takto:

```
while (i < a.length) {
    System.out.println(a[i]); i++;
}
```

Cyklus s podmínkou na konci

- Tělo se provádí dokud platí podmínka (vždy aspoň jednou).
- Obdoba repeat v Pascalu (podmínka je ovšem interpretována opačně).
- Relativně málo používaný - je méně přehledný než while

```
do {
     příkaz1;
     příkaz2;
     příkaz3;
     ...
} while (podmínka);
```

Cyklus "for"

- obecnější než for v Pascalu, podobně jako v C/C++
- de-facto jde o rozšíření while, lze jím snadno nahradit

```
for (počáteční operace; vstupní podmínka;
                        příkaz po každém průchodu)
    příkaz;
```

anebo (obvyklejší, bezpečnější)

```
for (počáteční operace; vstupní podmínka;
                        příkaz po každém průchodu)
{
    příkaz1;
    příkaz2;
    příkaz3;
    ...
}
```

Příklad použití "for" cyklu

Provedení určité sekvence určitý počet krát

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- Vypíše na obrazovku deset řádků s čísly postupně 0 až 9

Ekvivalent s while:

```
int i=0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

Výpis argumentů programu

```
public class Pole {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++)
            System.out.println(args[i]);
    }
}
```

Vícecestné větvení "switch - case - default"

- Obdoba pascalského select - case - else.
- Větvení do více možností na základě ordinální hodnoty

```
switch(výraz) {
    case hodnota1: prikaz1a;
                   prikaz1b;
                   break;
    case hodnota2: prikaz2a;
                   ...
                   break;
    default:       prikazDa;
                   ...
}
```

- Je-li výraz roven některé z hodnot, provede se sekvence uvedená za příslušným case.
- Sekvenci obvykle ukončujeme příkazem break, který předá řízení ("skočí") na první příkaz za ukončovací závorkou příkazu switch.

Příkaz "break"

- Realizuje "násilné" ukončení průchodu cyklem nebo větvením switch
- Syntaxe použití break v cyklu:

```
for (int i = 0; i < a.length; i++) {
    if(a[i] == 0) {
        break; // skoci se za konec cyklu
    }
}
if (a[i] == 0) {
    System.out.println("Nasli jsme 0 na pozici "+i);
} else {
    System.out.println("0 v poli neni");
}
```

Příkaz "continue"

- Používá se v těle cyklu.
- Způsobí přeskočení zbylé části průchodu tělem cyklu.
- Běh pokračuje další iterací.

```
for (int i = 0; i < a.length; i++) {
    if (a[i] == 5)
        continue;
    System.out.println(i);
}
```

Cvičení

- Práce s polem a rozhraním ...
- Vytvořit rozhraní Accessible, které definuje operace
  - zda se nějaký prvek v objektu nachází
  - nalezení indexu, na kterém se prvek nachází
- Vytvořit třídu SmallArray pro manipulaci s polem
  - bude implementovat vytvořené rozhraní
  - bude obsahovat pole, které se inicializuje konstruktorem
- Vytvořit třídu Test, která pracuje s třídou SmallArray. Jako typ proměnné používá definované rozhraní Accessible.
- Vytvořit podtřídu BigArray
  - bude obsahovat pole, které se vytvoří konstruktorem
  - obsahuje metodu pro naplnění prvku pole