Objektové programování a navrhování tříd

Jakým způsobem psát třídy, aby byly snadno srozumitelné, udržovatelné a znovu využitelné

Překlad originální prezentace ke Kapitole 6 z učebnice

Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling

Objektově orientované programovaní

- Je založeno na modelování reálného světa.
- Je založeno na vytváření struktury navzájem komunikujících objektů.
- Komunikace mezi objekty se uskutečňuje prostřednictvím posílání zpráv.
- Často se můžeme setkat v souvislosti s objektovým programováním s pojmy dědičnost, zapouzdření, polymorfismus.

Objekty reálného světa

- Každý objekt má řadu různých atributů.
- Hodnoty některých atributů se dají měnit, jiných nikoliv.
- Atributy mohou být reprezentovány i jiným objektem, který představuje část (komponentu) daného objektu (auto má motor, okno má rám)
  - vazba IS-PART

Objekty reálného světa

- Objekty lze seskupit do skupin objektů téhož druhu, které mají tytéž atributy.
- Objekty mohou být uskupeny do hierarchií.
  - vazba IS-A
- Objekty spolu dokáží komunikovat.

Komunikace mezi objekty

- Komunikaci mezi objekty si můžeme představit jako posílání zpráv.
- Příklad komunikace lze ilustrovat na sekání zahrady.
  - vytažení sekačky, start, pohyb, …
- De facto tzv. event based přístup
  - vazba mezi obsahem zprávy a akcí příjemce je velmi volná – reakce může být různá

Implementace objektů

- Ve převážné objektových programovacích jazyků se setkáme s pojmem třída.
- Třída je reprezentant určitého druhu objektů.
  - třída definuje společné rysy všech objektů, které do třídy náleží.
  - každý objekt náleží do nějaké třídy.
  - v Javě je i třída objektem, i když ne plnohodnotným.
- Třídy lze uspořádávat do hierarchií na základě dědičnosti.
  - v Javě je na vrcholu hierarchie třída Object.

Instance

- Třída jako šablona (továrna) na instance.
- Instance je konkrétní objekt vytvořený podle určitého předpisu (např. daného určitou třídou).
- Instance je objekt, který není třídou.
- Primitivní datové typy nejsou objekty
  - ani instance nějaké třídy
- Instance vznikají jejich vytvořením v paměti
- Instance jsou vytvářeny konstruktory
  - hlavička podobná metodě, stejné jméno jako třída

Instanční proměnné a metody

- Instanční proměnné uchovávají hodnoty atributů instance.
  - instanční proměnné jsou deklarovány ve třídě
  - hodnoty instančních proměnných jsou uloženy v objektu instance
- Stav instance je dán množinou hodnot atributů.
  - tzv. snapshot v daný čas
- Instanční metody definují chování instance.
  - instanční metody nejsou uloženy v instancích, ale jsou uloženy ve třídách.
- Dostupnost proměnných a metod může být omezena

Zprávy

- Zpráva je žádost o provedení určité akce, kterou lze předat objektu.
- Objekt typicky reaguje na zprávu zavoláním vhodné metody.
- Výběr metody, která bude provedena je plně na přijímacím objektu.
- Např. x = car.start();

V jazyce Java je třída objektem, i když ne plnohodnotným.

```
package objekty;

class PozemniDopravniProstredek {
    int maxRychlost;
    String znacka;
}

public class Auto extends PozemniDopravniProstredek {
    String barva;
    private int pocetKol = 4;

    public Auto(){};

    public Auto(int rychl, String zn, String ba, int kol) {
        maxRychlost = rychl;
        znacka = zn;
        barva = ba;
        pocetKol = kol;
    }

    public void status() {
        System.out.println(maxRychlost + "  " + barva + "  " +
                pocetKol + "  " + znacka);
    }
}
```

```
public class Objekty {
    public static void main(String[] args) {
        Auto mojeAuto = new Auto();
        mojeAuto.barva = "cervena";
        mojeAuto.maxRychlost = 200;
        mojeAuto.pocetKol = 4;
        mojeAuto.status();
        Auto mojeDruheAuto = new Auto(50,"Velorex","hneda",3);
        mojeDruheAuto.status();
    }
}
```

Ukázka

```
package faktf;
import java.util.Scanner;
public class FaktF {
  static int ctiPrirozene() {
     Scanner sc = new Scanner(System.in);
     System.out.println("zadejte přirozené
     číslo");  int n = sc.nextInt();
     if (n < 1) {
        System.out.println(n + " není přirozené číslo");
        System.exit(0);
     }
     return n;
.
.
.}
```

Navrhování tříd

- Návrh řízený zodpovědnostmi (Responsibility-driven design)
- Vazby (Coupling)
- Koheze (Cohesion)
- Refaktoring (Refactoring)

Změny softwaru

- Software není jako román, který se jednou napíše a pak zůstává nezměněn.
- Software je rozšiřován, opravován, udržován, portován, přizpůsobován…
- Tato práce je v průběhu času prováděna často různými lidmi (často desetiletí).

Změna nebo smrt

- Existují pouze dvě možnosti pro software:
  - Buď je stále udržován,
  - nebo umírá.
- Software, který nemůže být udržován, bude odložen.
  - každé pravidlo má výjimku

Kolosální dobrodružství v jeskyních

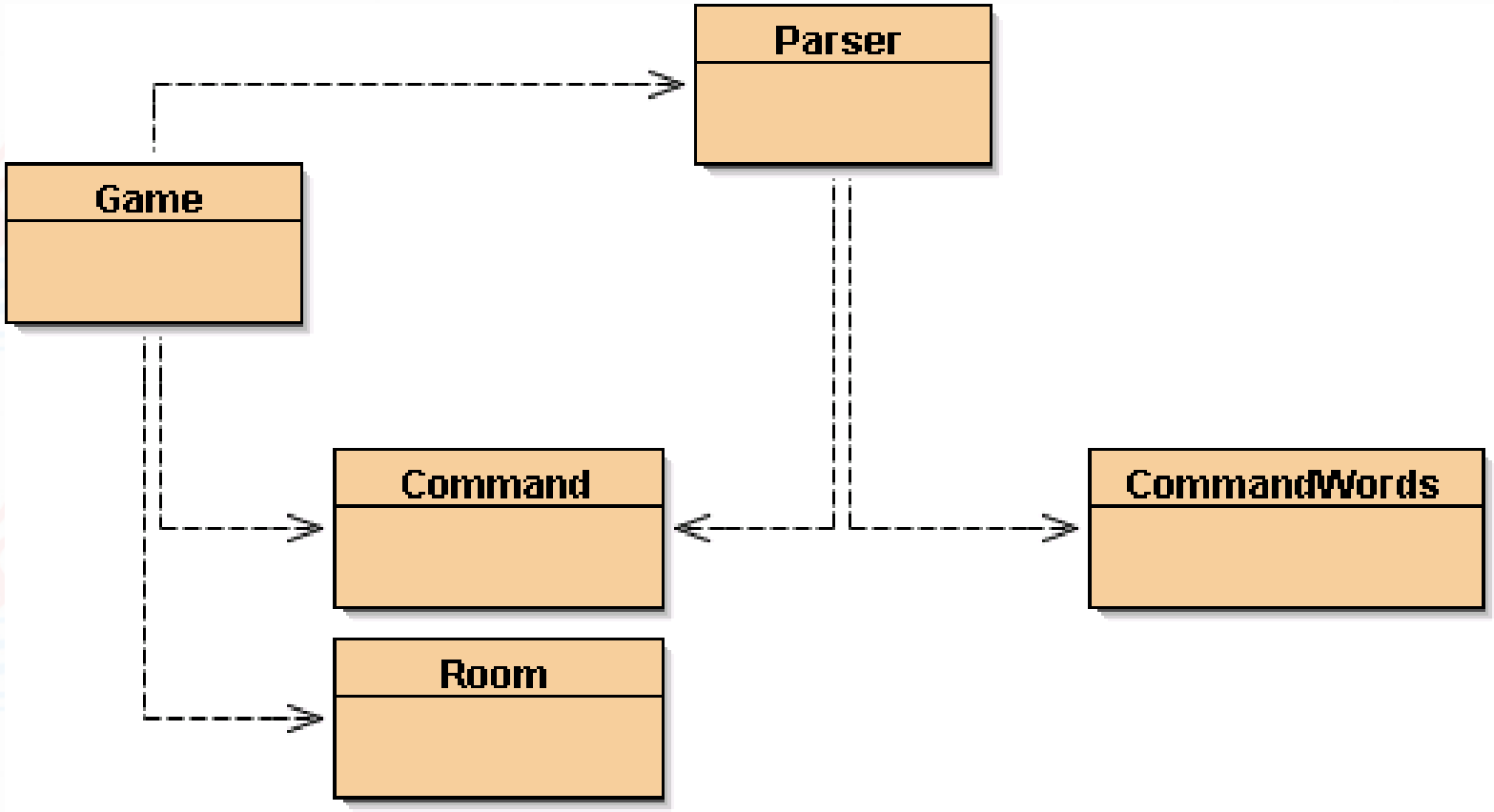
Historii této hry je možné nalézt zde:

http://rickadams.org/adventure/a_history.html

Třídy v projektu Zuul

- **Game:** Hlavní třída, které nastaví počáteční stav hry a pak opakovaně čte a provádí příkazy.
- **Room:** Instance této třídy reprezentují místa ve hře, která může hráč navštívit.
- **Parser:** Čte vstupní řádky od hráče a snaží se je interpretovat jako příkazy.
- **Command:** Instance reprezentuje celý příkaz zadaný uživatelem.
- **CommandWords:** Platná slova určující příkazy ve hře.

World of Zuul



Kvalita návrhu

- Jestliže máme posuzovat kvalitu návrhu aplikace, potřebujeme evaluační kritéria.
- Pro posuzování kvality návrhu existují dvě důležitá hlediska:
  - vazby (coupling)
  - koheze (cohesion)

Vazby

- Vazby poukazují na souvislosti mezi oddělenými jednotkami či komponentami programu.
- Jestliže dvě třídy těsně závisí na mnoha detailech druhé z nich, říkáme, že mají *těsnou vazbu*.
- Třídní diagram poskytuje (omezený) náhled na stupeň vazeb.

Volné vazby

- Naším cílem jsou volné vazby.
- Volné vazby umožňují:
  - porozumět jedné třídě bez nutnosti číst ostatní;
  - měnit jednu třídu bez malého nebo žádného vlivu na ostatní třídy.
- Tedy: volné vazby zlepšují udržovatelnost

Těsné vazby

- Snažíme se vyhnout těsným vazbám.
- Změny v jedné třídě přináší kaskádu změn v ostatních třídách.
- Samostatné třídy jsou obtížněji srozumitelné.
- Tok řízení (a jeho sledování) mezi objekty různých tříd je komplikovaný.

Koheze

- Koheze se týká počtu a různosti úkolů, za které je jedna jednotka (komponenta) zodpovědná.
- Jestliže je každá jednotka zodpovědná za jeden logický úkol, říkáme, že má *vysokou kohezi*.
- Koheze se aplikuje na třídy, metody a moduly (packages).

Vysoká koheze

- Naším cílem je vysoká koheze.
- Vysoká koheze usnadňuje:
  - porozumět tomu, co třída nebo metoda dělá;
  - použít popisná jména pro proměnné, metody a třídy;
  - znovu použít třídy a metody.

Nízká koheze

- Naším cílem je vyhnout se třídám a metodám s nízkou kohezí.
- Metody provádí více úkolů.
- Třídy nemají jasnou identitu (ohraničení).

Aplikace koheze na různých úrovních

- Úroveň třídy:
  - Třídy by měly reprezentovat jednu dobře definovanou entitu.
- Úroveň metody:
  - Metoda by měla být zodpovědná za jeden a jen jeden dobře definovaný úkol.

Příklad na test kvality

- Přidejte dva nové směry do projektu „World of Zuul“:
  - nahoru
  - dolů
- Co potřebujete změnit, aby bylo možné toto udělat?
- Jak je snadné provést bezchybně tyto změny?

Duplicita kódu

- Duplicita kódu
  - je indikátor špatného návrhu,
  - ztěžuje udržování,
  - během udržování může vést k zavedení chyb,
  - často důsledek programování metodou „Ctrl-C Ctrl-V“.
- Naší snahou je takový návrh, který duplicitu eliminuje.
  - často za cenu „zkomplikování“ kódu jeho rozdělením na více metod, tříd, …

Návrh řízený zodpovědnostmi

- Otázka: kam bychom měli přidat novou metodu (do které třídy)?
- Každá třída by měla být zodpovědná za manipulaci s vlastními daty (daty svých instancí).
- Třída, která vlastní data, by měla zodpovídat za jejich zpracování.
- RDD vede k volné vazbě.

Lokalizace změny

- Jedním cílem redukce vazeb a návrhu řízeného zodpovědnostmi je omezit změnu pokud možno na jedno místo.
- Když je potřeba změna, mělo by tím být zasaženo pokud možno co nejméně tříd.

Přemýšlení dopředu

- Když navrhujeme třídu, pokoušíme se přemýšlet, které změny budou pravděpodobně v budoucnu prováděny.
- Usilujeme o to, aby se změny prováděly snadno.

Refaktoring

- Když jsou třídy udržovány, je často přidáván kód.
- Třídy a metody mají sklon se prodlužovat.
- Třídy a metody by měly být čas od času *refaktorovány*, aby se udržela vysoká koheze a volná vazba.
- Při refaktoringu nedochází ke změně vnější funkcionality třídy

Refaktoring a testování

- Když provádíte refaktoring kódu, oddělte refaktoring od provádění dalších funkčních změn.
- Nejprve vždy pouze refaktoring beze změny funkcionality.
- Testujte před a po provední refaktoringu, aby jste se ujistili, že se nic neporušilo.

Otázky návrhu

- Obecné otázky:
  - Jak dlouhá by měla být třída?
  - Jak dlouhá by měla být metoda?

- Mohou být nyní zodpovězeny pomocí pojmů koheze a vazba.

Pokyny pro návrh

- Metoda je příliš dlouhá, jestliže provádí více než jeden logický úkol.
- Třída je příliš složitá, jestliže reprezentuje více než jednu logickou entitu.
- Poznámka: toto jsou doporučení - stále ještě ponechávají návrháři mnoho volnosti.

Přehled

- Programy se stále mění.
- Je důležité, aby bylo možné tyto změny provádět.
- Kvalita kódu vyžaduje více než správné provádění v daném čase.
- Kód musí být srozumitelný a udržovatelný.

Přehled

- Kvalitní kód se vyhýbá duplicitě a projevuje se vysokou kohezí a volnou vazbou.
- Styl kódování (komentování, výběr jmen, rozvržení, atd.) je rovněž důležitý.
- Je velký rozdíl v množství práce potřebné ke změně špatně a dobře strukturovaného kódu.

Výčtové typy

Výčtové typy

- Rys programovacího jazyka.
- K zavedení jména typu se používá `enum` místo `class`.
- Nejjednodušší použití spočívá v definici množiny platných jmen.
  - Alternativa k statickým konstantám typu int.
  - Když jsou hodnoty konstant libovolného typu.
- Použitelné při znalosti konečné množiny jmen již při překladu.

Základní výčtový typ

```
public enum CommandWord
{
    // Hodnota pro každé příkazové slovo,
    // plus jedna pro nerozpoznatelné příkazy.
    GO, QUIT, HELP, UNKNOWN;
}
```

- Každé jméno reprezentuje jeden objekt výčtového typu, např. CommandWord.HELP.
- Objekty výčtového typu nejsou přímo vytvářeny programátorem.

Základní výčtový typ

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY,
  WEDNESDAY, THURSDAY, FRIDAY,
  SATURDAY
}
```

Výčtové typy

```
public class EnumTest {
    private Day day;
    public EnumTest(Day day) {
        this.day = day;
    }

    public void tellItLikeItIs() {
        switch (day) {
                case MONDAY: System.out.println("Mondays are bad.");
                             break;

                case FRIDAY: System.out.println("Fridays are better.");
                             break;

                case SATURDAY:
                case SUNDAY: System.out.println("Weekends are best.");
                             break;

                default: System.out.println("Midweek days are so-so.");
                             break;
        }
    }
```

Výčtové typy

```
public static void main(String[] args) {
        EnumTest firstDay = new EnumTest(Day.MONDAY);
        firstDay.tellItLikeItIs();
        EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);
        thirdDay.tellItLikeItIs();
        EnumTest fifthDay = new EnumTest(Day.FRIDAY);
        fifthDay.tellItLikeItIs();
        EnumTest sixthDay = new EnumTest(Day.SATURDAY);
        sixthDay.tellItLikeItIs();
        EnumTest seventhDay = new EnumTest(Day.SUNDAY);
        seventhDay.tellItLikeItIs();
}
```

Výstup

Mondays are bad.

Midweek days are so-so.

Fridays are better.

Weekends are best.

Weekends are best.

Třídy výčtového typu

- Java definuje výčtové typy jako třídy
  - mohou mít konstruktory, položky, metody
  - vytváří se jedna instance třídy pro každý prvek výčtového typu

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);
}
```

Příklad třídy výčtového typu

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);
-------------------------------------------------
    private final double mass;   // in kilograms
    private final double radius; // in meters

    // universal gravitational constant  (m3kg-1s-2)
    public static final double G = 6.67300E-11;

    private Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }

    private double getMass()   { return mass; }

    private double getRadius() { return radius; }

    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }

    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Příklad třídy výčtového typu

Poměrná váha na různých planetách

```
public class PlanetTest
{
    public static void main(String[] args)
    {
        double earthWeight = Double.parseDouble(args[0]);

        double mass = earthWeight/Planet.EARTH.surfaceGravity();

        for (Planet p : Planet.values())

                System.out.printf("Your weight on %s is %f%n",
                                  p, p.surfaceWeight(mass));
     }

}
```

Spuštění mimo vývojové prostředí a výstup programu

java PlanetTest 175 – spuštění z příkazové řádky

```
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007
Your weight on JUPITER is 442.847567
Your weight on SATURN is 186.552719
Your weight on URANUS is 158.397260
Your weight on NEPTUNE is 199.207413
```