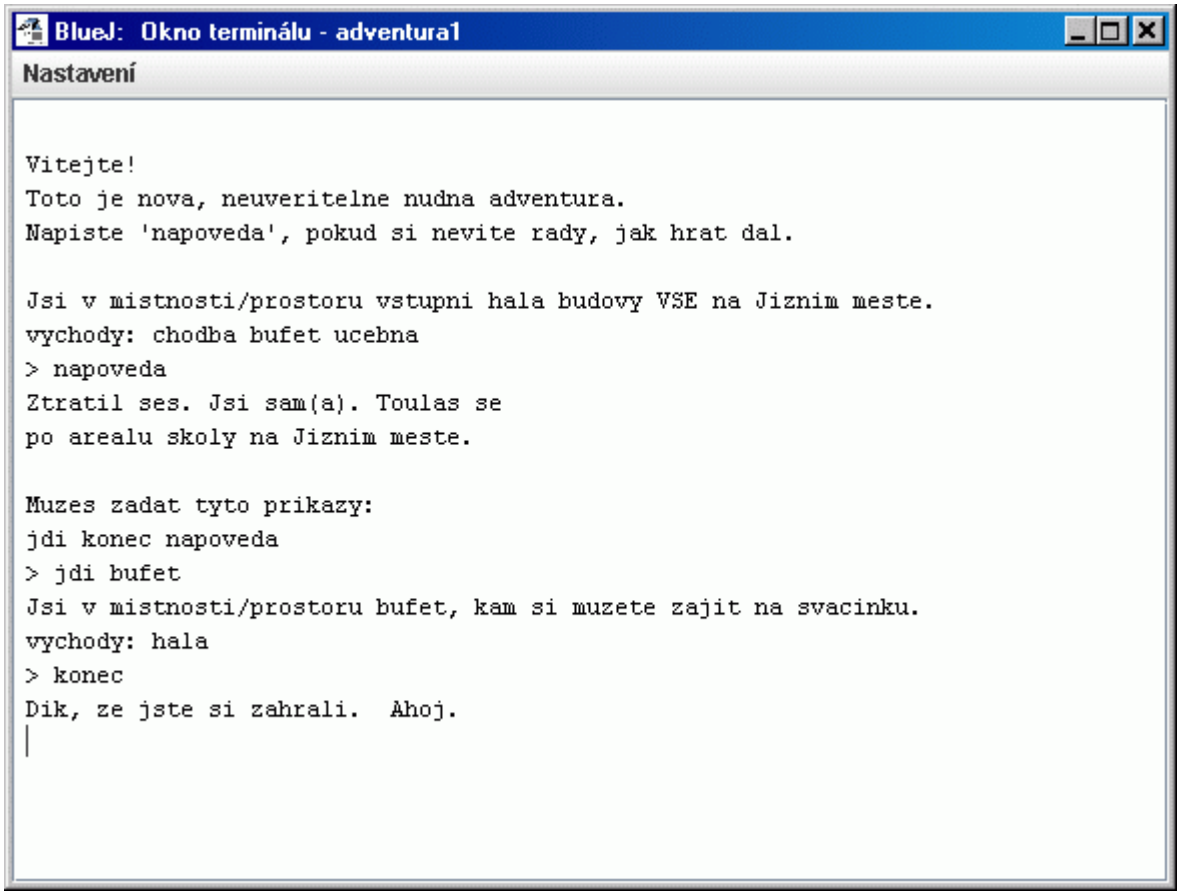
19.Projekt Adventura

19.1. Základní popis, zadání úkolu

Pracujeme na projektu Adventura, který je ke stažení na `java.vse.cz`. Po otevření v BlueJ vytvoříme instanci třídy `Hra`. Po zavolání metody `hraj()` se spustí jednoduchá adventura s textovým rozhraním, která umožní hráči procházet jednotlivými místnostmi hry. Hráč může zadávat příkazy „jdi“, „napoveda“ a „konec“. Na obrázku 19.1 je zachycen průběh „hraní“ této hry.



Obrázek 19.1 Jednoduchá ukázka průběhu komunikace s adventurou

V tomto projektu budeme řešit tyto úkoly:

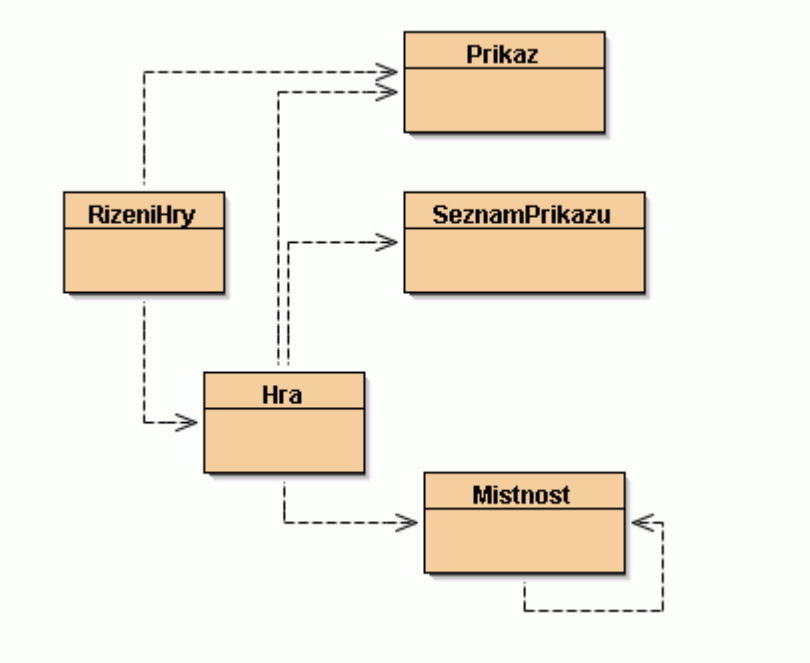
- prvním úkolem je doplnit hru o další místnost/prostor,
- dalším je změnit příkaz „napoveda“ na příkaz „pomoc“ (tj. aby uživatel místo příkazu „napoveda“ psal příkaz „pomoc“),
- dalším úkolem je doplnit „vítězství“, tj. aby po dosažení konkrétní místnosti hra sama skončila,
- posledním úkolem, který si zde ukážeme je doplnění hry o věci – tj. aby v místnostech byly jednotlivé věci, hráč je mohl sbírat do batohu či z batohu vyndávat a pokládat do místnosti.

Tento projekt má následující cíle:

- ukázat možnost, jak vytvořit síťovou strukturu instancí,
- ukázat vytváření nových tříd, doplňování metod do stávajících tříd.

19.2. Struktura tříd

Projekt Adventura se skládá z následujících tříd (obrázek z BlueJ):



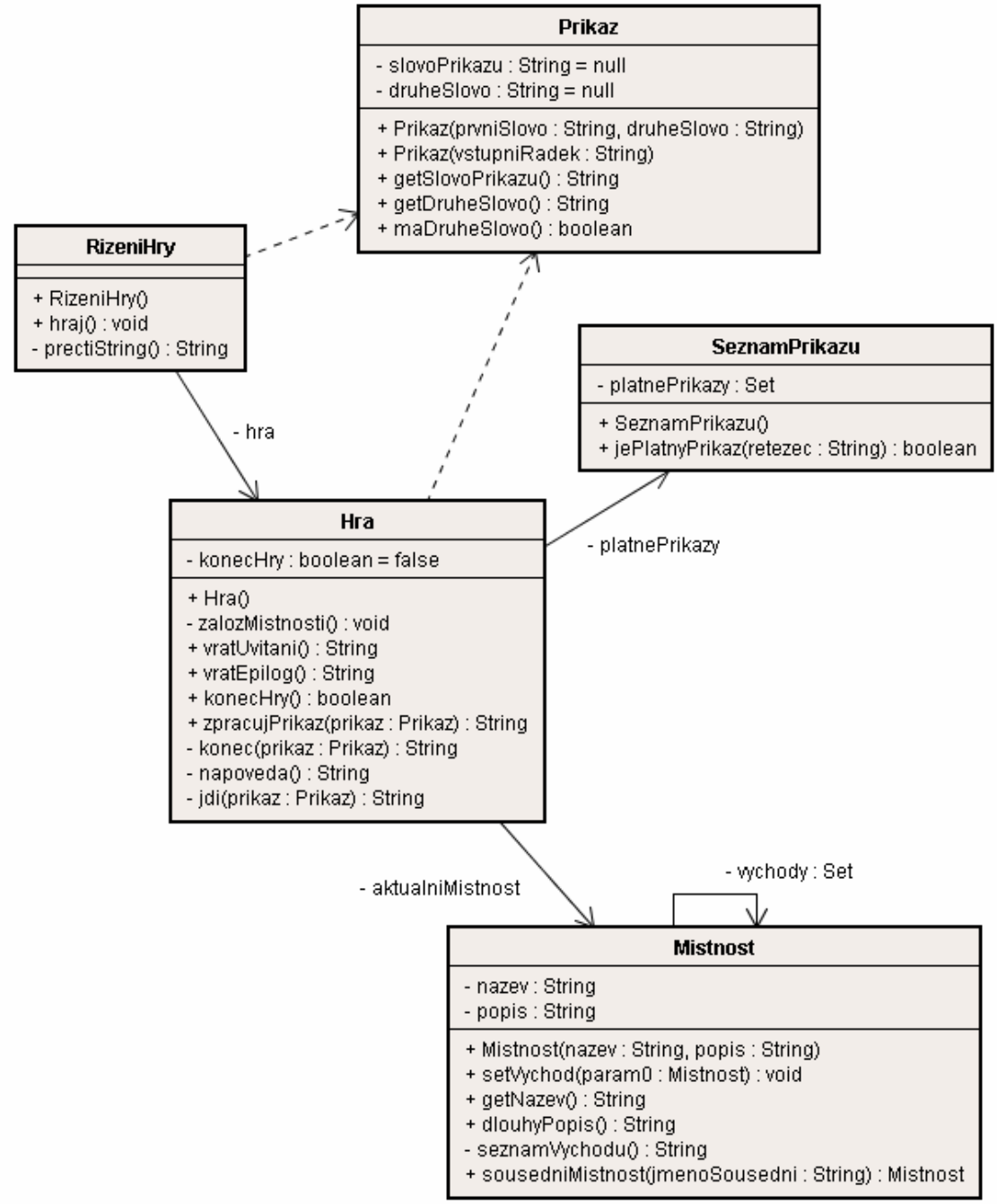
Obrázek 19.2 Struktura tříd projektu Adventura z BlueJ

19.3. Popis komunikace mezi objekty

Hra je rozdělena do 5 tříd, které mají takto rozděleny odpovědnosti:

- Třída `RizeniHry` obsahuje základ řízení hry (načtení řádku od uživatele z klávesnice, vytvoření příkazu, jeho předání ke zpracování do instance třídy `Hra` a vypsání výsledku na obrazovku). Do této třídy je soustředěno čtení z klávesnice a výpis na obrazovku.
- Na základě třídy `Prikaz` se vytváří instance obsahující povel zadaný uživatelem. Třída `Prikaz` rozdělí vstupní řádek na dvě části – vlastní slovo příkazu a případné druhé slovo příkazu.
- Instance třídy `SeznamPrikazu` obsahuje seznam všech přípustných příkazů. Instance vrací seznam dostupných příkazů a umí zjistit, zda příslušný řetězec je platný příkaz,
- Třída `Hra` představuje hlavní logiku hry – vytvářejí se v ní místnosti a seznam příkazů, obsahuje metody řešící jednotlivé příkazy uživatele. Z `RizeniHry` se volá hlavně metoda `zpracujPrikaz`, kterou se předává příkaz zadaný uživatelem. Metoda vrací řetězec, který se má zobrazit na obrazovce.
- Instance třídy `Mistnost` obsahuje jméno místnosti a seznam místností, do kterých vedou východy z aktuální místnosti. Instance pomocí metody `seznamVychodu()` vrací řetězec se seznamem východů, metoda `sousedniMistnost()` vrací instanci sousední místnosti odpovídající zadanému řetězci.

Diagram tříd používaný v BlueJ zachycuje základní vazby mezi třídami. V některých situacích je pro pochopení aplikace vhodné vytvořit si diagram tříd, který obsahuje i datové atributy a metody. Pro naši adventuru je takový diagram na obrázku 19.3.



Obrázek 19.3 Diagram tříd včetně datových atributů a metod

19.4. Výpis kódu třídy Místnost

```
 1 import java.util.Set;
 2 import java.util.HashSet;
 3
 4 /**
 5  *  Třída Mistnost popisuje jednotlivou místnost ve hře.
 6  *  Tato třída je součástí projektu jednoduché textové hry.
 7  *  "Mistnost" reprezentuje jedno místo (místnost, prostor, ..)
 8  *  ve scénáři hry. Místnost může mít sousední místnosti
 9  *  připojené přes východy. Pro každý východ si místnost ukládá
10  *  odkaz na sousedící místnost (instanci třídy Mistnost).
11  *
12  *@author    Michael Kolling, Lubos Pavlicek, Jarmila Pavlickova
13  *@version   3.0
14  *@created   květen 2005
15  */
16
17 class Mistnost {
18   private String nazev;
19   private String popis;
20   private Set<Mistnost> vychody;   // obsahuje sousední místnosti
21
22   /**
23    * Vytvoření místnosti (pojmenovaný prostor) se zadaným
24    * popisem, např. "kuchyň", "hala", "trávník před domem"
25    *
26    *@param  nazev   Jméno místnosti, jednoznačný
27    *                identifikátor, pokud možno jedno slovo
28    *@param  popis     Popis místnosti.
29    */
30   public Mistnost(String nazev, String popis) {
31      this.nazev = nazev;
32      this.popis = popis;
33      vychody = new HashSet<Mistnost>();
34   }
35
36   /**
37    * Definuje východ z místnosti (sousední/vedlejší místnost).
38    * Vzhledem k tomu, že je použit Set pro uložení východů,
39    * může být sousední místnost uvedena pouze jednou
40    * (tj. nelze mít dvoje dveře do stejné sousední místnosti).
41    * Druhé zadání stejné místnosti tiše přepíše předchozí zadání
42    * (neobjeví se žádné chybové hlášení).
43    * Lze zadat též cestu ze/do sebe sama.
44    *
45    * @param vedlejsi místnost, která sousedí s aktuální
46    *                místností.
47    */
48   public void setVychod(Mistnost vedlejsi) {
49      vychody.add(vedlejsi);
50   }
51
52    /**
53    * Metoda equals pro porovnání dvou místností. Překrývá se metoda
54    * equals ze třídy Object. Dvě místnosti jsou shodné, pokud mají
55    * stejné jméno.
```

```
56     * Tato metoda je důležitá z hlediska správného fungování
57     * seznamu místností (Set).
58     *
59     * Bližší popis metody equals je u třídy Object.
60     *
61     *@paramo    object, který se má porovnávat s aktuálním
62     *@return        vrací hodnotu true, pokud zadaná místnost má
63     *                 stejné jméno, jinak false
64     */
65    public boolean equals (Object o) {
66       if (o instanceof Mistnost) {
67            Mistnost druha = (Mistnost)o;
68            return nazev.equals(druha.nazev);
69       }
70       else {
71            return false;
72       }
73    }
74
75    /**
76     * Metoda hashCode vrací číselný identifikátor instance,
77     * který se používá pro optimalizaci ukládání
78     * v dynamických datových strukturách.
79     * Při překrytí metody equals je potřeba překrýt i
80     * metodu hashCode.
81     * Podrobný popis pravidel pro vytváření metody hashCode
82     *  je ve třídě Object
83     */
84    public int hashCode() {
85       return nazev.hashCode();
86    }
87
88    /**
89     * Vrací jméno místnosti (bylo zadáno při vytváření místnosti
90     * jako parametr konstruktoru)
91    *
92     *@return    Jméno místnosti
93     */
94    public String getNazev() {
95       return nazev;
96    }
97
98    /**
99     * Vrací "dlouhý" popis místnosti, který může vypadat
100    * následovně:
101    *  Jsi v mistnosti/prostoru vstupni hala budovy VSE
102    *                                          na Jiznim meste.
103    * vychody: chodba bufet ucebna
104    *
105    *@return    Dlouhý popis místnosti
106    */
107   public String dlouhyPopis() {
108   return "Jsi v mistnosti/prostoru " + popis + ".\n" +
109                                           seznamVychodu();
110   }
111
```

```
112   /**
113    * Vrací textový řetězec, který popisuje sousední místnosti
114    *                                          (východy)
115    * vychody: chodba bufet ucebna
116    *
117    *@return    Seznam sousedních místností (východů)
118    */
119   private String seznamVychodu() {
120      String vracenyText = "vychody:";
121      for (Mistnost sousedni : vychody) {
122           vracenyText += " " + sousedni.getNazev();
123      }
124      return vracenyText;
125   }
126
127   /**
128    * Vrací místnost, která sousedí s aktuální místností
129    * a jejíž jméno je zadáno jako parametr. Pokud místnost
130    * s udaným jménem nesousedí s aktuální místností,
131    * vrací se hodnota null.
132    *
133    *@param  jmenoSousedni  Jméno sousední místnosti (východu)
134    *@return    Místnost, která se nachází za příslušným východem,
135    *       nebo hodnota null, pokud místnost zadaného jména není
136    *       sousedem.
137    */
138   public Mistnost sousedniMistnost(String jmenoSousedni) {
139   if (jmenoSousedni == null) {
140      return null;
141   }
142   for ( Mistnost sousedni :vychody ){
143      if (sousedni.getNazev().equals(jmenoSousedni)) {
144           return sousedni;
145      }
146   }
147   return null;  // místnost nenalezena
148   }
149 }
```

19.5. Výpis kódu třídy Hra

```
 1 /**
 2  *  Třída Hra představující logiku adventury.
 3  *
 4  *  Tato třída inicializuje další třídy:
 5  *  vytváří všechny místnosti (třída Mistnost),
 6  *  vytváří seznam platných příkazů.
 7  *  Ve hře se též vyhodnocují jednotlivé příkazy zadané
 8  *  uživatelem.
 9  *
10  *@author Michael Kolling, Lubos Pavlicek, Jarmila Pavlickova
11  *@version    3.0
12  *@created    květen 2005
13  */
14
```

```
15 class Hra {
16   private SeznamPrikazu platnePrikazy;
17                     // obsahuje seznam přípustných slov
18   private Mistnost aktualniMistnost;
19   private boolean konecHry = false;
20
21   /**
22    *  Vytváří hru, inicializuje místnosti
23    * a seznam platných příkazů.
24    */
25   public Hra() {
26      zalozMistnosti();
27      platnePrikazy = new SeznamPrikazu();
28   }
29
30   /**
31    *  Vytváří jednotlivé místnosti a propojuje je pomocí východů.
32    */
33   private void zalozMistnosti() {
34   Mistnost hala;
35   Mistnost ucebna;
36   Mistnost bufet;
37   Mistnost chodba;
38      Mistnost kancelar;
39
40      // vytvářejí se jednotlivé místnosti
41      hala = new Mistnost("hala",
42                     "vstupni hala budovy VSE na Jiznim meste");
43      ucebna = new Mistnost("ucebna", "prednaskova ucebna 103JM");
44      bufet = new Mistnost("bufet",
45                     "bufet, kam si muzete zajit na svacinku");
46      chodba = new Mistnost("chodba","spojovaci chodba");
47      kancelar = new Mistnost("kancelar",
48                      "kancelar vaseho vyucujiciho Javy");
49   // přiřazují se východy z místností (sousedící místnosti)
50   hala.setVychod(ucebna);
51   hala.setVychod(chodba);
52   hala.setVychod(bufet);
53   ucebna.setVychod(hala);
54   bufet.setVychod(hala);
55   chodba.setVychod(hala);
56   chodba.setVychod(kancelar);
57   kancelar.setVychod(chodba);
58   aktualniMistnost = hala;  // hra začíná v místnosti hala
59   }
60   /**
61    *  Vrátí úvodní zprávu pro hráče.
62    */
63   public String vratUvitani() {
64      return "Vitejte!\n" +
65           "Toto je nova, neuveritelne nudna adventura.\n" +
66           "Napiste 'napoveda', pokud si nevite rady,
67                                        jak hrat dal.\n" +
68           "\n" +
69           aktualniMistnost.dlouhyPopis();
70   }
71
```

```
 72   /**
 73    *  Vrátí závěrečnou zprávu pro hráče.
 74    */
 75   public String vratEpilog() {
 76      return "Dik, ze jste si zahrali.  Ahoj.";
 77   }
 78
 79   /**
 80    * Vrací true, pokud hra skončila.
 81    */
 82   public boolean konecHry() {
 83      return konecHry;
 84   }
 85
 86   /**
 87    *  Metoda zpracuje příkaz uvedený jako parametr,
 88    *  tj. spustí odpovídající metodu.
 89    *  Metoda vrací řetězec, který se má vypsat na obrazovku.
 90    *
 91    *@param  prikaz  příkaz, který se má zpracovat (provést)
 92    *@return  vrací se řetězec, který se má vypsat na obrazovku
 93    */
 94   public String zpracujPrikaz(Prikaz prikaz) {
 95   String textKVypsani=" .... ";
 96   if (platnePrikazy.jePlatnyPrikaz(prikaz.getSlovoPrikazu())){
 97      String povel = prikaz.getSlovoPrikazu();
 98      if (povel.equals("napoveda")) {
 99           textKVypsani = napoveda();
100      }
101      else if (povel.equals("jdi")) {
102           textKVypsani = jdi(prikaz);
103      }
104      else if (povel.equals("konec")) {
105           textKVypsani = konec(prikaz);
106      }
107   }
108   else {
109      textKVypsani="Nevim co tim myslis, tento prikaz neznam?";
110   }
111   return textKVypsani;
112   }
113
114  // následuje implementace jednotlivých příkazů
115
116   /**
117    * V případě, že příkaz má jen jedno slovo "konec" hra končí,
118    * jinak pokračuje např. při zadání "konec a".
119    *
120    * @return Vrací true v případě, že příkaz má jen jedno slovo
121    * "konec", jinak vrací false
122    */
123
```

```
124   private String konec(Prikaz prikaz) {
125   if (prikaz.maDruheSlovo()) {
126      return "Ukoncit co? Nechapu, proc jste zadal druhe
127                                 slovo.";
128   }
129      else {
130           konecHry = true;
131           return "hra ukončena příkazem konec";
132      }
133   }
134
135   /**
136    *  Vypíše základní nápovědu po zadání příkazu "napoveda".
137    * Nyní se vypisuje vcelku primitivní zpráva
138    * a seznam dostupných příkazů.
139    */
140   private String napoveda() {
141      return "Ztratil ses. Jsi sam(a). Toulas se\n"
142           + "po arealu skoly na Jiznim meste.\n"
143           + "\n"
144           + "Muzes zadat tyto prikazy:\n"
145           + platnePrikazy.vratSeznamPrikazu();
146   }
147
148   /**
149    *  Příkaz "jdi".
150    * Zkouší se vyjít zadaným směrem/do zadané místnosti.
151    * Pokud místnost existuje, vstoupí se do nové místnosti.
152    * Pokud zadaná sousední místnosti (východ) není,
153    *    vypíše se chybové hlášení.
154    *
155    *@param  prikaz jako druhý parametr obsahuje jméno místnosti,
156    *                          do které se má jít.
157    */
158   private String jdi(Prikaz prikaz) {
159      if (!prikaz.maDruheSlovo()) {
160           // pokud chybí druhé slovo (sousední místnost), tak
161           return "Kam mám jít? Musíš zadat jméno místnosti";
162      }
163      String smer = prikaz.getDruheSlovo();
164      // zkoušíme přejít do sousední místnosti
165      Mistnost sousedniMistnost =
166                     aktualniMistnost.sousedniMistnost(smer);
167   if (sousedniMistnost == null) {
168      return "Tam se odsud jit neda!";
169   }
170   else {
171      aktualniMistnost = sousedniMistnost;
172      return aktualniMistnost.dlouhyPopis();
173   }
174   }
175 }
```

19.6. Výpis kódu třídy RizeniHry

```
 1 import java.io.BufferedReader;
 2 import java.io.InputStreamReader;
 3 import java.io.IOException;
 4
 5 /**
 6  *  Class RizeniHry
 7  *
 8  *  Toto je hlavní třída aplikace.
 9  * Je to velmi jednoduchá textová hra
10  *   - uživatel se pohybuje v daném prostoru.
11  *  Je určena jako základ pro další zajímavá rozšíření.
12  *  Tato třída vytváří instanci třídy Hra,
13  *    která představuje logiku aplikace.
14  *  Čte jednotlivé příkazy zadané uživatelem
15  *     a předává je logice a vypisuje odpověď logiky na konzoli.
16  *  Pokud chcete hrát tuto hru, vytvořte instanci této třídy
17  *     a poté na ní vyvolejte metodu "hraj".
18  *
19  *
20  *@author  Michael Kolling, Lubos Pavlicek, Jarmila Pavlickova
21  *@version    3.0
22  *@created    květen 2005
23  */
24
25 class RizeniHry {
26
27   private Hra hra;
28
29   /**
30    *  Vytváří hru.
31    */
32   public RizeniHry() {
33      hra = new Hra();
34   }
35
36   /**
37    *  Hlavní metoda hry.
38    *  Cyklí se do konce hry (dokud metoda konecHry() z logiky
39    *     nevrátí hodnotu true)
40    */
41   public void hraj() {
42   System.out.println(hra.vratUvitani());
43   // základní cyklus programu - opakovaně se čtou příkazy
44   // a poté se provádějí do konce hry.
45   while (!hra.konecHry()) {
46      String radek = prectiString();
47      Prikaz prikaz = new Prikaz(radek);
48      System.out.println(hra.zpracujPrikaz(prikaz));
49   }
50   System.out.println(hra.vratEpilog());
51   }
52
53   /**
54    *  Metoda přečte příkaz z příkazového řádku
55    *
56    *@return    Vrací přečtený příkaz jako instanci třídy String
```

```
57     */
58    private String prectiString() {
59       String vstupniRadek="";
60       System.out.print("> ");        // vypíše se prompt
61       BufferedReader vstup = new BufferedReader
62                       (new InputStreamReader(System.in));
63       try {
64            vstupniRadek = vstup.readLine();
65       }
66       catch (java.io.IOException exc) {
67        System.out.println("Vyskytla se chyba během čtení příkazu:"
68                       + exc.getMessage());
69       }
70       return vstupniRadek;
71    }
72 }
```

19.7. Výpis kódu třídy SeznamPrikazu

```
 1 import java.util.Set;
 2 import java.util.TreeSet;
 3
 4 /**
 5  * Třída SeznamPrikazu - obsahuje seznam přípustných příkazů
 6  * hry. Používá se pro rozpoznávání příkazů.
 7  *
 8  * Tato třída je součástí jednoduché textové hry.
 9  *
10  *@author   Michael Kolling, Lubos Pavlicek, Jarmila Pavlickova
11  *@version  3.0
12  *@created  květen 2005
13  */
14 class SeznamPrikazu {
15      // set pro uložení přípustných příkazů
16    private  Set<String> platnePrikazy;
17
18    /**
19     * Konstruktor
20     */
21    public SeznamPrikazu() {
22       platnePrikazy = new TreeSet<String>();
23       platnePrikazy.add("jdi");
24       platnePrikazy.add("konec");
25       platnePrikazy.add("napoveda");
26    }
27    /**
28     * Kontroluje, zda zadaný řetězec je přípustný příkaz.
29     *
30     * @param retezec Řetězec, který se testuje, zda je to
31                      přípustný příkaz
32     * @return        Vrací true, pokud zadaný řetězec je
33                      přípustný příkaz
34     */
35    public boolean jePlatnyPrikaz(String retezec) {
36       return platnePrikazy.contains(retezec);
37    }
38
```

```
39    /**
40     * Vrací seznam přípustných příkazů, jednotlivé příkazy jsou
41     *  odděleny mezerou.
42     * @return   Řetězec, který obsahuje seznam přípustných příkazů
43     */
44    public String vratSeznamPrikazu() {
45    String seznam = "";
46    for (String slovoPrikazu : platnePrikazy){
47       seznam += slovoPrikazu + " ";
48    }
49    return seznam;
50    }
51  }
```

19.8. Výpis kódu třídy Prikaz

```
 1  /**
 2   * Class Prikaz - část jednoduché textové adventury.
 3   *
 4   * Třída udržuje informace o zadaném příkazu. V současné době
 5   * může příkaz mít až dvě slova (např. "jdi kuchyn").
 6   * Pokud má příkaz pouze jedno slovo, je jako druhé slovo
 7   * uložena hodnota null.
 8   *
 9   * Třídu je možné rozšířit na příkazy se třemi či více slovy.
10   *
11   *@author  Michael Kolling, Lubos Pavlicek, Jarmila Pavlickova
12   *@version    3.0
13   *@created    květen 2005
14   */
15
16  class Prikaz {
17    private String slovoPrikazu = null;
18    private String druheSlovo = null;
19
20    /**
21     * Vytváří instanci třídy Prikaz. Musí být zadáné první a druhé
22     * slovo příkazu. Pokud příkaz nemá druhé slovo, vloží se
23     * místo něho hodnota null.
24     *
25     *@param  prvniSlovo   První slovo příkazu
26     *@param  druheSlovo   Druhé slovo příkazu
27     */
28    public Prikaz(String prvniSlovo, String druheSlovo) {
29       slovoPrikazu = prvniSlovo;
30       this.druheSlovo = druheSlovo;
31    }
32    /**
33     * Druhý konstruktor pro vytvoření instance třídy Prikaz.
34     * Konstruktor převezme řetězec přečtený z příkazového řádku a
35     * vytvoří instanci třídy Prikaz. Pokud příkaz nemá druhé
36     * slovo, vloží se jako druhý parametr hodnota null.
37     *
38     *@param  vstupniRadek   Retezec se vstupnim radkem
39     */
```

```
40   public Prikaz(String vstupniRadek) {
41   String [] slova = vstupniRadek.split("[ \t]+");
42   if (slova.length > 0) {
43      slovoPrikazu = slova[0];          // první slovo
44   }
45   if (slova.length > 1) {
46      druheSlovo = slova[1];            // druhé slovo
47   }
48      // poznámka: ignoruje se zbytek řádky
49   }
50
51   /**
52    * Vrací se příkaz (první slovo příkazu).
53    *
54    *@return    Příkaz (první slovo příkazu), jako řetězec.
55    */
56   public String getSlovoPrikazu() {
57      return slovoPrikazu;
58   }
59
60   /**
61    * Vrací druhé slovo příkazu. Pokud neexistuje, vrací se
62    * hodnotu null.
63    *@return    Druhé slovo příkazu.
64    */
65   public String getDruheSlovo() {
66      return druheSlovo;
67   }
68
69   /**
70    * Vrací hodnotu true, pokud příkaz má druhé slovo
71    *
72    *@return    true, pokud má příkaz druhé slovo
73    */
74   public boolean maDruheSlovo() {
75      return (druheSlovo != null);
76   }
77 }
```

19.9. Postup řešení

Před řešením jednotlivých příkladů doporučujeme pro pochopení aplikace:

- spustit si základ hry a zkusit se pohybovat po místnostech,
- udělat první dva domácí úkoly (vytvoření sekvenčních diagramů).

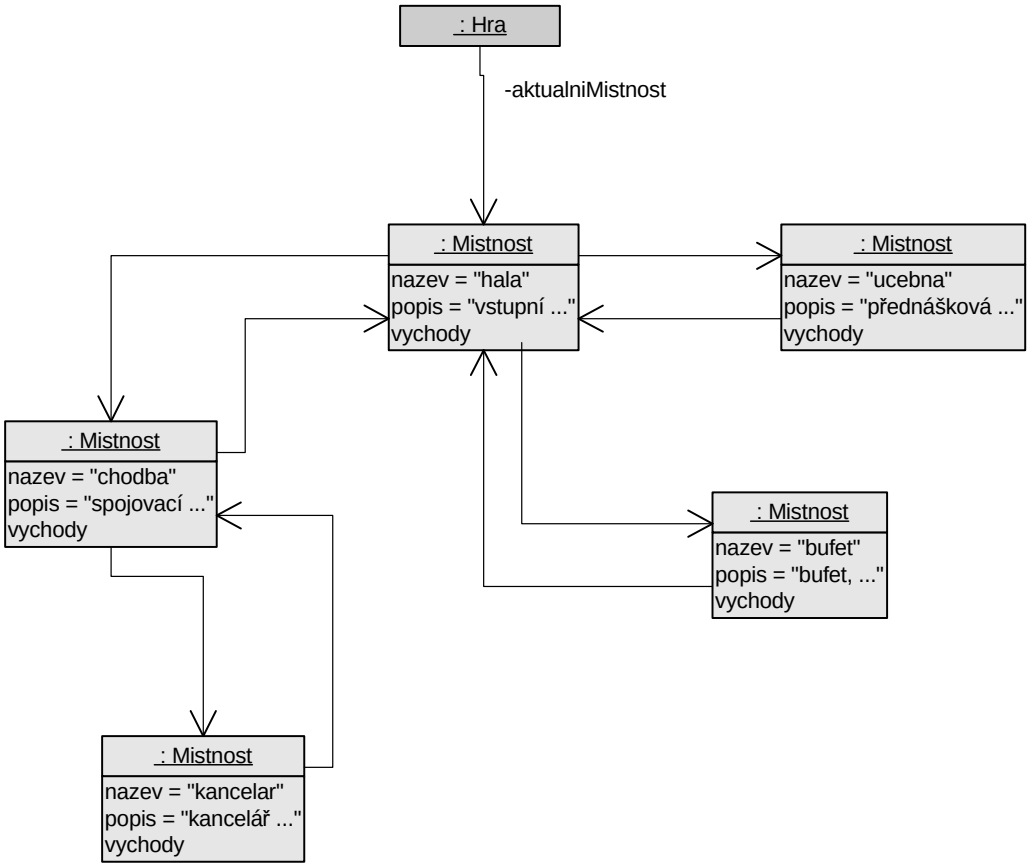
19.9.1. Úkol 1: doplnění další místnosti

Před řešením této úlohy odpovíme na otázku, kolik je ve hře místností, tj. kolik je instancí třídy *Mistnost*:

- počet instancí odpovídá počtu new s konstruktorem třídy *Mistnost*, místnosti se vytvářejí pouze ve třídě *Hra* – celkem 5 místností,
- je potřeba odlišovat počet instancí a počet odkazů – na jednu instanci lze odkazovat z různých míst, konkrétně na instanci místnosti s názvem „hala“ jsou na začátku 4 odkazy:
    * z místnosti „chodba“ je odkaz na „halu“ v seznamu východů (datový atribut *vychody*),
    * z místnosti „bufet“ je východ do haly,
    * z místnosti „ucebna“ je východ do haly,

    * poslední odkaz je z instance třídy `Hra` – odkaz je uložen v datovém atributu `aktualniMistnost` (viz řádek 60 zdrojového kódu třídy `Hra`),
- místnost, na kterou odkazuje datový atribut `aktualniMistnost` se mění při přechodu do jiné místnosti (po zadání příkazu „jdi ucebna“ bude datový atribut `aktualniMistnost` odkazovat na instanci třídy `Mistnost` s názvem „ucebna“). Při změně hodnoty datového atributu `aktualniMistnost`:
    * nemění se žádná instance třídy `Mistnost`, pouze se mění stav instance třídy `Hra`,
    * neztrácí se žádná instance třídy `Mistnost` – v Javě se ruší instance v případě, že na ně není žádný odkaz, což zde neplatí, neboť místnosti odkazují na sebe navzájem přes východy.

Následující obrázek zobrazuje okamžitý stav (konkrétně stav na začátku hry) mezi instancemi třídy `Mistnost` a instancí třídy `Hra`. V průběhu aplikace se vztahy mění – z instance třídy `Hra` odkazuje datový atribut `aktualniMistnost` na různé instance třídy `Mistnost`.



Obrázek 19.4 Diagram objektů zobrazuje vztahy mezi instancemi třídy Mistnost a instancí třídy Hra v projektu Adventura na začátku hry

Vlastní hra je poměrně triviální, proto ji zde nebudeme podrobněji popisovat. Při vytváření místnosti nezapomeňte na východy z místnosti/vchody do místnosti. Ověřte si dostupnost nové místnosti.

19.9.2. Úkol 2: změna příkazu „napoveda“ na „pomoc“

Cílem tohoto úkolu je uvědomit si, do kterých tříd je potřeba zasáhnout při změně názvu příkazu či při přidávání příkazů.

Vlastní úprava je triviální – jsou potřeba 2+1 úpravy:

- upravit řetězec v seznamu příkazů ve třídě `SeznamPrikazu` (řádek 25),
- upravit řetězec v metodě `zpracujPrikaz` ve třídě `Hra` (řádek 98),

- v případě příkazu nápověda je ještě potřeba upravit část úvodního textu ve třídě *Hra* (řádek 66).

Kdybychom zapomněli udělat jednu z prvních dvou úprav nebo nezapsali na obě místa v kódu stejné řetězce, nebude příkaz fungovat. Aby se tomuto zabránilo (a případně i ušetřilo pár bytů vnitřní paměti), je vhodné psát řetězec pouze jednou. Existují dvě řešení:

- Použít výčtový typ – tuto variantu necháváme na čtenářích.
- Použít pojmenované konstanty, ve třídě *SeznamPrikazu* by se na začátku nadefinovaly pojmenované konstanty:

```
private  Set<String> platnePrikazy;
public static final String PRIKAZ_JDI=”jdi”;
public static final String PRIKAZ_KONEC=”konec”;
public static final String PRIKAZ_NAPOVEDA=”napoveda”;
/**
 * Konstruktor
 */
public SeznamPrikazu() {
     platnePrikazy = new TreeSet<String>();
     platnePrikazy.add(PRIKAZ_JDI);
     platnePrikazy.add(PRIKAZ_KONEC);
     platnePrikazy.add(PRIKAZ_NAPOVEDA);
}
```

ve třídě *Hra* by příslušná část metody *zpracujPrikaz()* mohla vypadat následovně:

```
if (platnePrikazy.jePlatnyPrikaz(prikaz.getSlovoPrikazu())){
     String povel = prikaz.getSlovoPrikazu();
     if (povel.equals(SeznamPrikazu.PRIKAZ_NAPOVEDA)) {
          textKVypsani = napoveda();
     }
     else if (povel.equals(SeznamPrikazu.PRIKAZ_JDI)) {
          textKVypsani = jdi(prikaz);
     }
     else if (povel.equals(SeznamPrikazu.PRIKAZ_KONEC)) {
          textKVypsani = konec(prikaz);
     }
}
```

Po této úpravě již stačí změnit vlastní slovo příkazu pouze na jednom místě – ve třídě *SeznamPrikazu*.

Práci s příkazy lze rozšířit několika směry:

- podpora alternativních slov pro stejnou akci (např. aby uživatel mohl použít jako slovo „napoveda“, tak slovo „pomoc“),
- podpora různých jazykových verzí – zde by bylo vhodné nastudovat podporu pro internacionalizaci a lokalizaci začleněnou do Javy,
- přiřazení akcí k vlastním příkazům, tj. aby součástí definice příkazu bylo nejen vlastní slovo příkazu, ale i akce, která se má po zadání tohoto příkazu provést – tato problematika přesahuje rozsah skript.

19.9.3. Úkol 3: doplnění vítězství

Pod vítězstvím rozumíme, že po dosažení místnosti „kancelář“ by hra měla skončit a vypsat příslušný závěrečný text. Základní cyklus ve třídě `RizeniHry` vypadá následovně:

```
    while (!hra.konecHry()) {
         String radek = prectiString();
         Prikaz prikaz = new Prikaz(radek);
         System.out.println(hra.zpracujPrikaz(prikaz));
    }
         System.out.println(hra.vratEpilog());
```

tj. v rámci cyklu se čtením vstupního řádku a zpracováním tohoto příkazu se testuje, zda není konec hry (volá se metoda `konecHry()`, která při konci hry vrací hodnotu `true`, jinak hodnotu `false`). Pokud je konec hry, vypíše se závěrečné hlášení (metoda `vratEpilog()`).
My potřebujeme, aby po dosažení místnosti kancelář metoda `konecHry()` vracela hodnotu `true` a následně by metoda `vratEpilog()` měla vrátit odpovídající hlášení (odlišné od situace, kdy uživatel zadal příkaz konec). Metoda `konecHry()` ve třídě `Hra` vrací hodnotu logické proměnné `konecHry` – tj. po dosažení místnosti kancelář je potřeba nastavit tuto proměnnou na hodnotu `true`. Míst, kde to lze nastavit je více, asi nejvhodnější je umístit potřebný kód na konec metody `zpracujPrikaz()` – na řádek 114:

```
         if (aktualniMistnost == kancelar) {
              konecHry = true;
         }
         return textKVypsani;
    }
```

Aby tento kód fungoval, je potřeba si uvědomit, že nyní proměnnou `kancelar` potřebujeme ve dvou metodách – v metodě `zalozMistnosti()` a v metodě `zpracujPrikaz()` – je potřeba z této proměnné udělat datový atribut:

- na začátek třídy se doplní deklarace (včetně modifikátoru přístupu `private`):

```
         private Mistnost kancelar;
```

- v metodě `zalozMistnosti()` se musí deklarace zrušit (pokud to neuděláte, tak Vás překladač neupozorní na možnou chybu, ale aplikace nebude fungovat).

Vrácení vítězného textu lze řešit přes podmínku v metodě `vratEpilog()`:

```
    public String vratEpilog() {
         if (aktualniMistnost == kancelar) {
              return "Gratuluji, dosáhli jste kanceláře.  Ahoj.";
         }
         else {
              return "Dik, že jste si zahráli.  Ahoj.";
         }
    }
```

Vhodnější je využít pro závěrečný řetězec datový atribut, který by metoda `vratEpilog()` vracela. Datový atribut by byl deklarován na začátku třídy `Hra`:

```
private String epilog="Dík, že jste si zahráli.  Ahoj.";
```

V metodě `zpracujPrikaz()` by se nastavil alternativní text:

```
         if (aktualniMistnost == kancelar) {
              konecHry = true;
              epilog="Gratuluji, dosáhli jste kanceláře.  Ahoj.";
         }
         return textKVypsani;
```

A metoda *vratEpilog()* by byla jednoduchá:

```
    public String vratEpilog() {
         return epilog;
    }
```

19.9.4. Úkol 4: doplnění věcí do místností

Věci by měly být uloženy v místnostech, v jedné místnosti může být více věcí. Některé věci lze přenášet, některé ne. Uživatel má možnost věci v místnosti sbírat a ukládat je do batohu (příkaz „seber“), dále může vybrat věc v batohu a uložit ji do místnosti (příkaz „polož“). Uživatel též musí mít možnost vypsat seznam věcí v místnosti – při vstupu do místnosti se vypíše vedle popisu místnosti i seznam věcí v místnosti. Dále bude mít uživatel k dispozici příkaz „inventar“, který vypíše obsah batohu.
Doplnění věcí do místností lze rozdělit do následujících částí:

- návrh třídy *Vec*, která představuje vzor pro jednotlivé věci ve hře,
- doplnění třídy *Mistnost* o metodu, která umožní vkládat věci do místnosti, metodu, která umožní vybrat věc z místnosti a rozšíření metody *dlouhyVypis()* o vypsání seznamu věcí v místnosti,
- inicializace jednotlivých věcí ve hře a jejich umístění do místností,
- doplnění uživatelského příkazu seber, pro sebrání věci z místnosti do batohu,
- doplnění hry o batoh, batoh by měl být představován samostatnou třídou (teoreticky poté bude možno vytvořit více batohů), batoh by měl mít omezenou kapacitu,
- doplnění uživatelského příkazu poloz, který přesune věc z batohu do místnosti,
- doplnění uživatelského příkazu inventar, který vypíše obsah batohu.

My si zde ukážeme pouze první čtyři části, poslední tři necháváme na čtenáře.

Návrh třídy Vec

Třída *Vec* představuje vzor pro jednotlivé věci, které se budou používat v programu. Z našeho zadání vyplývá, že u věci potřebujeme dva datové atributy:

- název věci – použijeme typ *String*,
- informaci o tom, zda je věc přenositelná – použijeme proměnnou typu *boolean*.

Hodnoty těchto datových atributů bude vhodné nastavit při vytvoření instance třídy *Vec* – přes parametry konstruktoru.
Věci nezajišťují nějaké zvláštní činnosti ani není potřeba měnit hodnotu datových atributů, proto postačuje doplnit třídu *Vec* o metody, které budou vracet hodnoty datových atributů:

- metodu, která bude vracet název věci,
- metodu, která bude vracet informaci o tom, zda je věc přenositelná.

```
 1 /**
 2  *  Trida Vec - popisuje jednotlivou věc ve hře
 3  *
 4  *  Tato třída je součástí jednoduché textové hry.
 5  *
 6  *  "Vec" reprezentuje jednu věc ve scénáři hry. Věc může být
 7  *  přenositelná (tj. hráč některé věci může vzít a některé ne).
 8  *
 9  *@author     Lubos Pavlicek, Jarmila Pavlickova
10  *@version    3.0
11  *@created    květen 2005
12  */
13
```

```
14 class Vec {
15   private String nazev;
16   private boolean prenositelna;
17 
18   /**
19    * Vytvoření věci se zadaným názvem
20    *
21    *@param  nazev     Jméno věci, jednoznačný identifikátor,
22                       pokud možno jedno slovo
23    *@param  prenositelna     Parametr určuje, zda je věc
24                       přenositelná hráčem
25    */
26   public Vec(String nazev, boolean prenositelna) {
27      this.nazev = nazev;
28      this.prenositelna = prenositelna;
29   }
30 
31   /**
32    * Vrací název věci.
33    *
34    * @return      název věci
35    */
36   public String getNazev() {
37      return nazev;
38   }
39 
40   /**
41    * Vrací informaci o tom, zda je věc přenositelná ve hře.
42    *
43    * @return    true, pokud je věc přenositelná, jinak false
44    */
45   public boolean jePrenositelna() {
46      return prenositelna;
47   }
48 }
```

Doplnění místností o věci

Při doplnění místností o věci se rozšíří datové atributy a metody třídy `Mistnost`. Ve třídě `Mistnost` přibude datový atribut, do kterého se budou ukládat věci umístěné v místnosti. Pro uložení je vhodné použít některou z datových struktur, do kterých lze vložit více prvků (věcí). V úvahu připadají seznamy (`List`), množiny (`Set`), mapy (`Map`) i pole (`array`). Každá z těchto struktur má své výhody a nevýhody:

- seznam (`List`) – do místnosti lze uložit více stejných věcí (věcí se stejným jménem), což nemusí být žádoucí,
- množina (`Set`) – do místnosti lze vložit věc se stejným názvem pouze jednou, předpokladem použití je, že třída `Vec` bude implementovat metody `equals()` a `hashCode()`, při vkládání je potřeba též dávat pozor, aby se věci neztrácely (viz kapitola o datových strukturách),
- mapa (`Map`) – do místnosti lze vložit věc se stejným názvem pouze jednou, lze zajistit jednodušší vyhledávání věci než v případě použití množiny,
- pole (`array`) – nevýhodou pole je, že se musí předem určit maximální počet věcí, které lze uložit do nějaké místnosti, vlastní kód metod bude delší než v ostatních případech, při dobrém naprogramování bude nejrychlejší.

My použijeme pro uložení věcí seznam (`List`), konkrétně `ArrayList`. Následuje deklarace seznamu věcí na začátku třídy `Mistnost`:

```
    private List<Vec> seznamVeci;
```

Inicializace datového atributu by měla být v konstruktoru:

```
    seznamVeci = new ArrayList<Vec>();
```

Instance třídy `Mistnost` musí podporovat tyto činnosti v souvislosti s věcmi:

- vložení věci do místnosti – metoda `vlozVec()`, která bude mít jako parametr věc, která se má vložit do místnosti (do seznamu věcí v místnosti),
- zjištění, zda je v místnosti uložena nějaká věc – metoda `obsahujeVec()`, která bude mít jako parametr název věci a bude vracet hodnotu `true` či `false`,
- vybrání věci z místnosti – metoda `vyberVec()` vybere věc z místnosti a předá ji jako návratovou hodnotu (pokud věc v místnosti neexistuje či není přenositelná, vrací hodnotu `null`),
- vypsání seznamu věcí v místnosti – seznam věcí připojíme k výpisu informací o místnosti v metodě `dlouhyVypis()`, pro vytvoření textové položky obsahující jména všech věcí v místnosti vytvoříme samostatnou privátní metodu `seznamVeci()`.

Následuje kód prvních tří metod – `vlozVec()`, `obsahujeVec()` a `vyberVec()`:

```
    public void vlozVec(Vec neco) {
        seznamVeci.add(neco);
    }

    public boolean obsahujeVec(String nazevVeci) {
        for ( Vec neco : seznamVeci ) {
            if (neco.getNazev().equals(nazevVeci)) {
                return true;
            }
        }
        return false;
    }

    public Vec vyberVec(String nazevVeci) {
        Vec vybranaVec = null;
        for ( Vec neco : seznamVeci ) {
            if (neco.getNazev().equals(nazevVeci)) {
                vybranaVec=neco;
            }
        }
        if (vybranaVec != null) {
            if (vybranaVec.jePrenositelna()) {
                seznamVeci.remove(vybranaVec);
            }
            else {
                vybranaVec=null;
            }
        }
        return vybranaVec;
    }
```

Všimněte si, že metoda `vyberVec()` vrací hodnotu `null` ve dvou situacích:

- v místnosti příslušná věc neexistuje,
- věc v místnosti existuje, ale není přenositelná. Testování přenositelnosti přímo v této metodě je znakem dobrého zapouzdření objektu.

Metoda `vyberVec()` při úspěšném vybrání věci z místnosti tuto věc v seznamu věcí v místnosti zruší.
Vypsání věci v místnosti znamená rozšíření metody `dlouhyVypis()` a vytvoření privátní metody `seznamVeci()`, která vytvoří řetězec obsahující jména věcí v místnosti:

```
    private String seznamVeci() {
         String seznam = "";
         for ( Vec neco : seznamVeci ) {
              seznam = seznam + neco.getNazev()+" ";
         }
         return seznam;
    }

    public String dlouhyPopis() {
         return "Jsi v mistnosti/prostoru " + popis + ".\n"
                   + seznamVychodu() + "\n"
                   + "Veci: " + seznamVeci();
    }
```

Inicializace věcí v místnostech

Inicializaci věcí je vhodné umístit do stejného místa, kde se inicializují místnosti – do metody `zalozMistnosti()` ve třídě `Hra`. V následujícím kódu se vytvářejí tři věci a vkládají se do místností:

```
         // vkládání věcí do místností
         Vec hamburger = new Vec("hamburger",true);
         bufet.vlozVec(hamburger);
         bufet.vlozVec(new Vec("stul",false));
         kancelar.vlozVec(new Vec("sponka",true));
```

V případě hamburgeru je deklarována proměnná `hamburger`, vytvořena instance a poté vložena do místnosti, na kterou odkazuje proměnná `bufet`. V případě ostatních věcí je vše v jednom řádku. Druhý způsob se používá častěji, první varianta (s hamburgerem) se používá, pokud se dále potřebujeme odkazovat na tuto instanci. V situaci, kdy k věci potřebujeme přistupovat i z dalších metod, by se použil datový atribut s modifikátorem `private`.
Pozor! Uvedený kód musí být zařazen až za vytvoření instancí místnosti – nejdříve je potřeba vytvořit místnosti a teprve poté do nich umísťovat věci.

Doplnění příkazu seber

Doplnění příkazu se skládá ze tří částí:

- Doplnění příkazu do seznamu platných příkazů ve třídě `SeznamPrikazu`. Toto je jednoduché, v konstruktoru třídy `SeznamPrikazu` stačí doplnit další příkaz.
- Doplnění rozskoku v metodě `zpracujPrikaz()` ve třídě Hra. V metodě `zpracujPrikaz()` se na základě zadaného příkazu volá příslušná metoda. Vlastní doplnění je také triviální.
- Doplnění metody `seber()` do třídy `Hra`. Struktura metody je podobná, jako u metody `jdi()`:
    * metoda vrací textový řetězec, který se má zobrazit uživateli na obrazovce,
    * na začátku se zjišťuje, zda uživatel zadal jméno věci, která se má sebrat v místnosti,
    * pokud je věc v místnosti (volání `aktualniMistnost.obsahujeVec()`), zkusí se sebrat věc z místnosti. V případě, že věc není přenositelná, nepodaří se ji sebrat a metoda vrátí příslušné chybové hlášení. Pokud se podaří sebrat věc, měla by se uložit do batohu – tato část kódu chybí, doplní se při programování batohu.

```
    /**
     *  Příkaz "seber". Zkouší se sebrat věc z místnosti a uložit do
     *  batohu. Pokud věc v místnosti je a je přenositelná, uloží se
     *  do batohu.
     *@param  prikaz  příkaz, jako druhý parametr obsahuje jméno
     *                        věci, která se má sebrat.
     */
    private String seber(Prikaz prikaz) {
        if (!prikaz.maDruheSlovo()) {
            return "Co mám sebrat? Musíš zadat jméno věci";
        }

        String nazevVeci = prikaz.getDruheSlovo();
        if (aktualniMistnost.obsahujeVec(nazevVeci)) {
            Vec pozadovanaVec =
                        aktualniMistnost.vyberVec(nazevVeci);
            if (pozadovanaVec == null) {
                return nazevVeci+" se nedá přenášet";
            }
            else {
                // DOPROGRAMOVAT ULOZENI VECI DO BATOHU
                return nazevVeci+" jsi vzal z místnosti a "
                        + "uložil do batohu ";
            }
        }
        else {
            return nazevVeci + " není v místnosti";
        }
    }
```

19.10. Domácí úkol

1. Nakreslete sekvenční diagram komunikace mezi instancemi tříd *Hra*, *Prikaz* a *SeznamPrikazu* při volání metod *hraj()* a *zpracujPrikaz()* v instanci třídy *Hra*.

2. Nakreslete sekvenční diagram komunikace mezi instancemi při provádění metody *jdi()* ve třídě *Hra*.

3. Upravte hru tak, aby vracela nápovědu pro jednotlivé příkazy, tj. aby se po napsání příkazu
   napoveda jdi
   vypsala nápověda k příkazu jdi. Texty, které se budou vypisovat k jednotlivým příkazům, by měly být ve třídě *SeznamPrikazu* – místo množiny *Set* pro uložení seznamu příkazů použijte implementaci rozhraní *Map* – jako klíč uložte příkaz, jako hodnotu vlastní text nápovědy. Bude potřeba též do této třídy doplnit metodu pro získání nápovědy.

4. Zkuste upravit hru tak, aby bylo možné používat skloňování v příkazech, např. „jdi do chodby“, „seber sponku“. Názvy místností a věcí by se měly vypisovat v prvním pádě. Pro řešení tohoto zadání je potřeba ukládat do instancí tříd *Mistnost* a *Vec* více názvů – v 1. a ve 4. pádě.

5. Napište třídu, která bude simulovat vstupy od uživatele. Tato třída nahradí třídu `RizeniHry`, po spuštění metody by se měla vypsat komunikace hry, jako kdyby vstup vkládal uživatel. Ve třídě bude uložena např. následující posloupnost příkazů:
   jdi chodba
   napoveda
   jdi kancelar
   konec