Informatika 2

2

Udržovateľný kód

KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Pojmy zavedené v 1. prednáške(1)

- typy chýb
  - syntaktické
  - behové
  - logické
- techniky boja s chybami
  - testovanie
  - ladenie
  - písanie udržovateľného kódu

Pojmy zavedené v 1. prednáške(2)

- testovanie – rôzne pohľady
  - aplikačné testovanie a testovanie jednotiek
  - biela a čierna skrinka
  - pozitívne a negatívne
  - manuálne a automatické
- manuálne testovanie jednotiek
  - prechádzanie zdrojového kódu
  - priama komunikácia s objektom
    - BlueJ – využitie funkcie object inspector

Pojmy zavedené v 1. prednáške(3)

- automatické testovanie
  - testy regresie
  - test driven development
  - testovacie triedy – JUnit
    - správa `assertEquals`
    - prípravky – fixtures

Pojmy zavedené v 1. prednáške(4)

- ladenie
  - manuálne prechádzanie kódu
  - ladiace výpisy
  - debugger

Cieľ prednášky

- charakteristiky dobrého návrhu tried
  - nízka implementačná závislosť
  - vysoká súdržnosť
  - duplicita kódu – zlá vlastnosť kódu
- návrh tried určený zodpovednosťou
- refaktoring – zlepšenie kódu
- príklad: Hra „World of FRI“

World of FRI

Autor: Jozef Karas

Písanie udržovateľného kódu

- čitateľnosť kódu
- konvencie
- dokumentačné komentáre
- komentáre v zložitejších miestach algoritmu
- samopopisné identifikátory
- súdržnosť (cohesion) – max.
- implementačná závislosť (coupling) – min.

Implementačná závislosť(1)

- coupling
- úroveň vzájomného prepojenia tried
- zmeny v implementácii jednej triedy si vynútia zmeny v implementácii druhej, závislej triedy
- snažíme sa minimalizovať

Implementačná závislosť(2)

- vonkajší pohľad – rozhranie = „čo objekt robí“
- vnútorný pohľad – implementácia = „ako to robí“

- minimálna závislosť – používa iba dobre navrhnuté rozhranie
- implementačná závislosť – požíva „ako to robí“

Znaky minimálnej závislosti

- pochopenie triedy bez nutnosti skúmať triedy, na ktorých je závislá
- zmena implementácie jednej triedy nevyžaduje zmeny iných tried

Súdržnosť kódu

- cohesion
- počet a rôznosť úloh jednej jednotky kódu
- vysoká súdržnosť – jednotka má jedinú logickú úlohu

- metóda – práve jedna presne definovaná úloha – operácia.
- trieda – jedna presne definovaná logická entita

- cieľ – maximalizácia súdržnosti

Vysoká súdržnosť uľahčuje

- pochopenie úloh triedy a metód
- opakované použitie (reuse) triedy alebo metód v iných častiach kódu, v inom softvéri

Zodpovednosťou riadený návrh(1)

- responsibility-driven design
- objekt má len jednu úlohu
- objekt zodpovedá za svoj stav – dáta
- objekt zodpovedá za zmeny stavu – operácie s dátami
- informácie o svojom stave poskytuje objekt
- atribúty tvoria logický celok
- metódy implementujú jednu operáciu

Zodpovednosťou riadený návrh(2)

- do ktorej triedy pridať novú metódu?
- čie dáta bude metóda spracovávať?

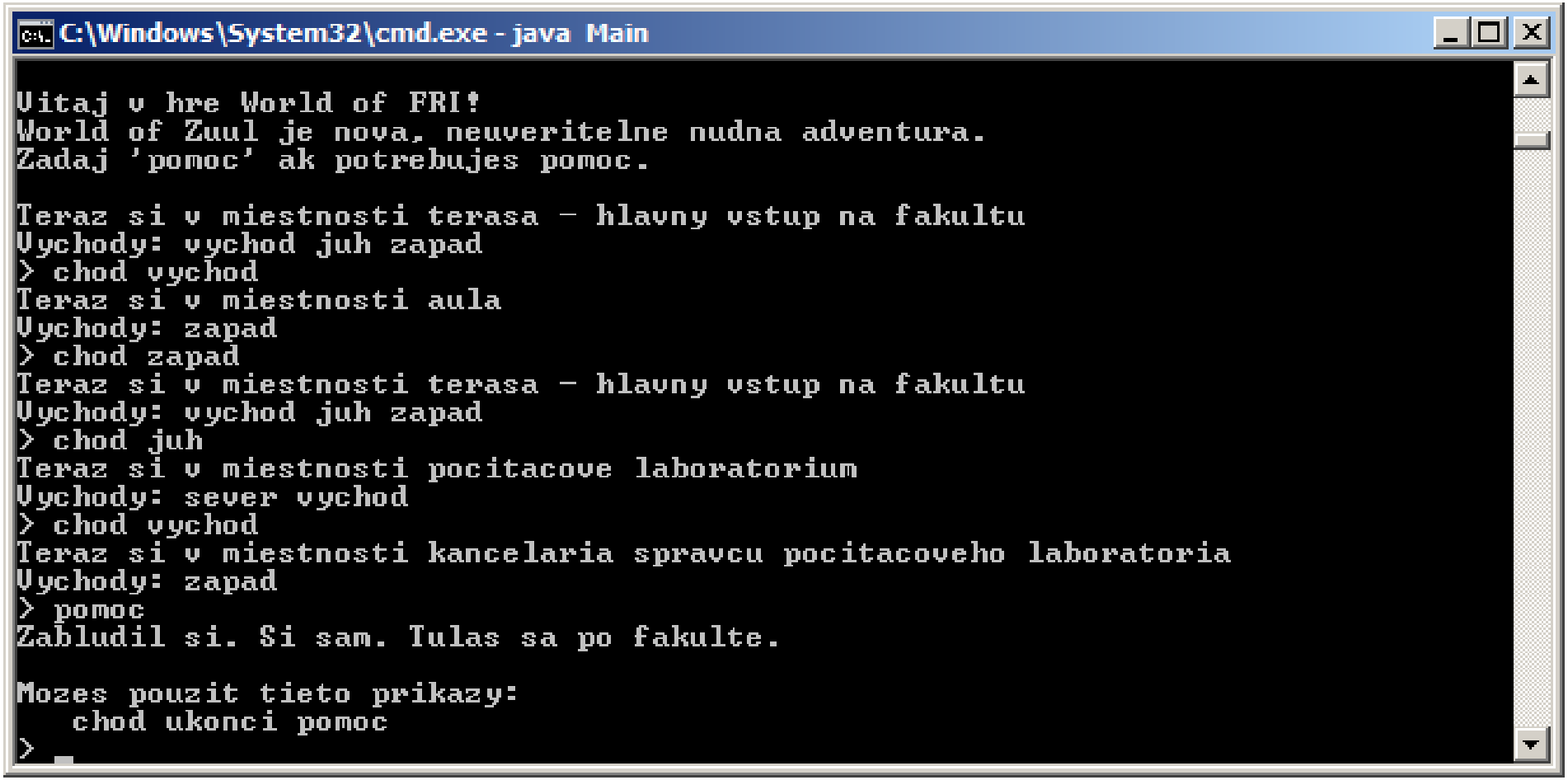
- každá trieda je zodpovedná za operácie so svojimi vlastnými dátami

Aká veľká má byť

- trieda?
- metóda?

- metóda je príliš dlhá, ak vykonáva viac ako jednu úlohu – operáciu
- trieda je príliš zložitá, ak spája viac ako jednu logickú entitu
- rešpektovanie týchto pravidiel necháva ešte stále dostatočný priestor programátorovi

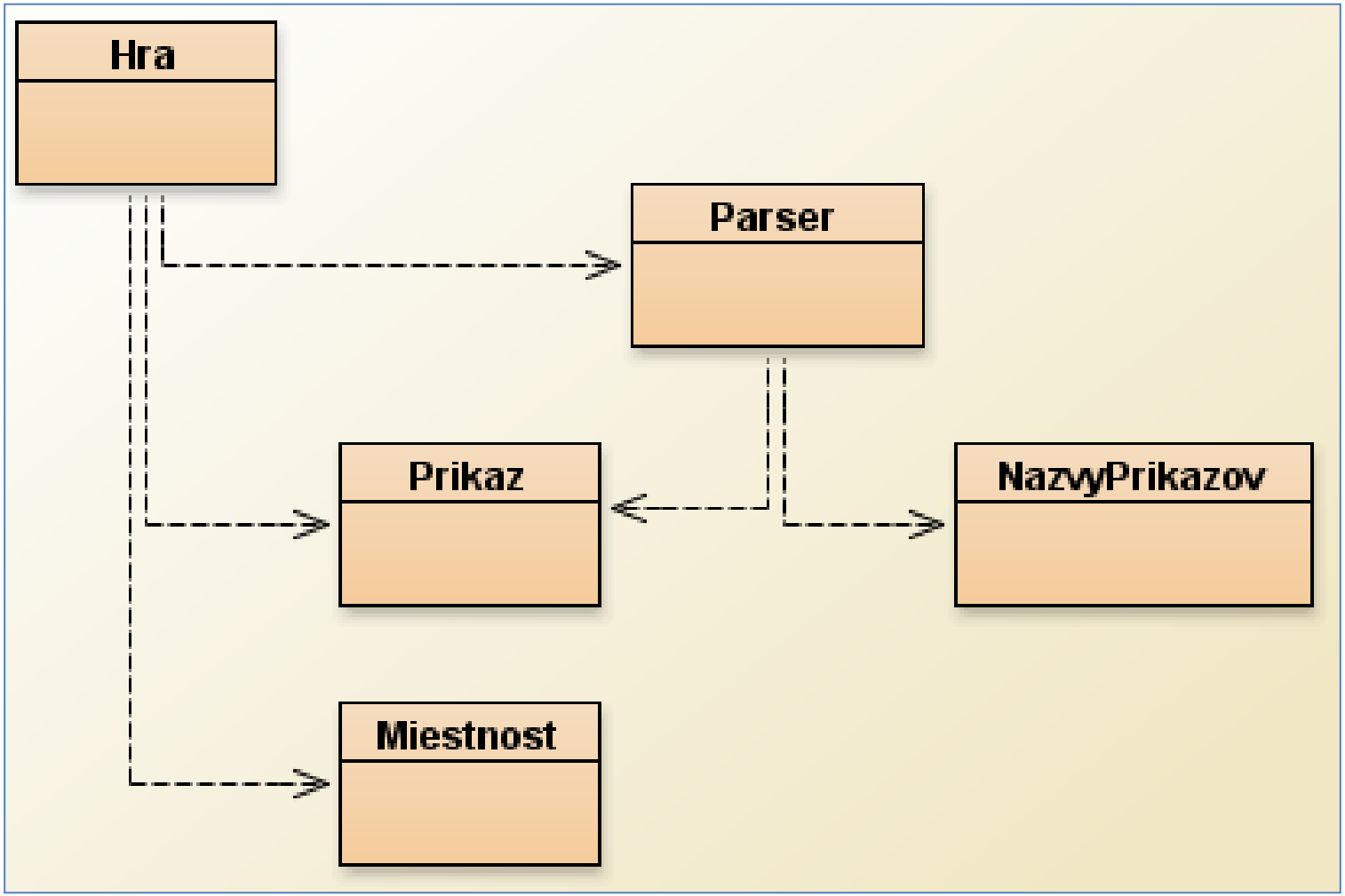
Hra World of FRI



WoF – existujúci kód

- hra – textová adventúra
- miestnosti: `terasa`, `aula`, `bufet`, `labak`, `kancelaria`
- príkazy: `chod`, `pomoc`, `ukonci`
- smery: `sever`, `vychod`, `juh`, `zapad`

WoF – diagram tried



Trieda NazvyPrikazov

- definícia všetkých platných príkazov
- názvy príkazov – pole reťazcov
- overuje platnosť príkazu

Trieda Parser

- číta vstup hráča z terminálu
- rozpoznáva vo vstupe slová príkazov
- príkazy kontroluje prostredníctvom inštancie triedy NazvyPríkazov
- vytvára inštancie triedy Prikaz

Trieda Prikaz

- inštancie sú nemeniteľné objekty
- reprezentuje zadaný príkaz
- príkaz má 2 časti: názov a parameter
- overuje platnosť príkazu (nazov != null)

Trieda Miestnost

- reprezentuje určitý priestor v hre
- má východy do iných miestností
- východy označené smerom
- neexistujúci východ = null

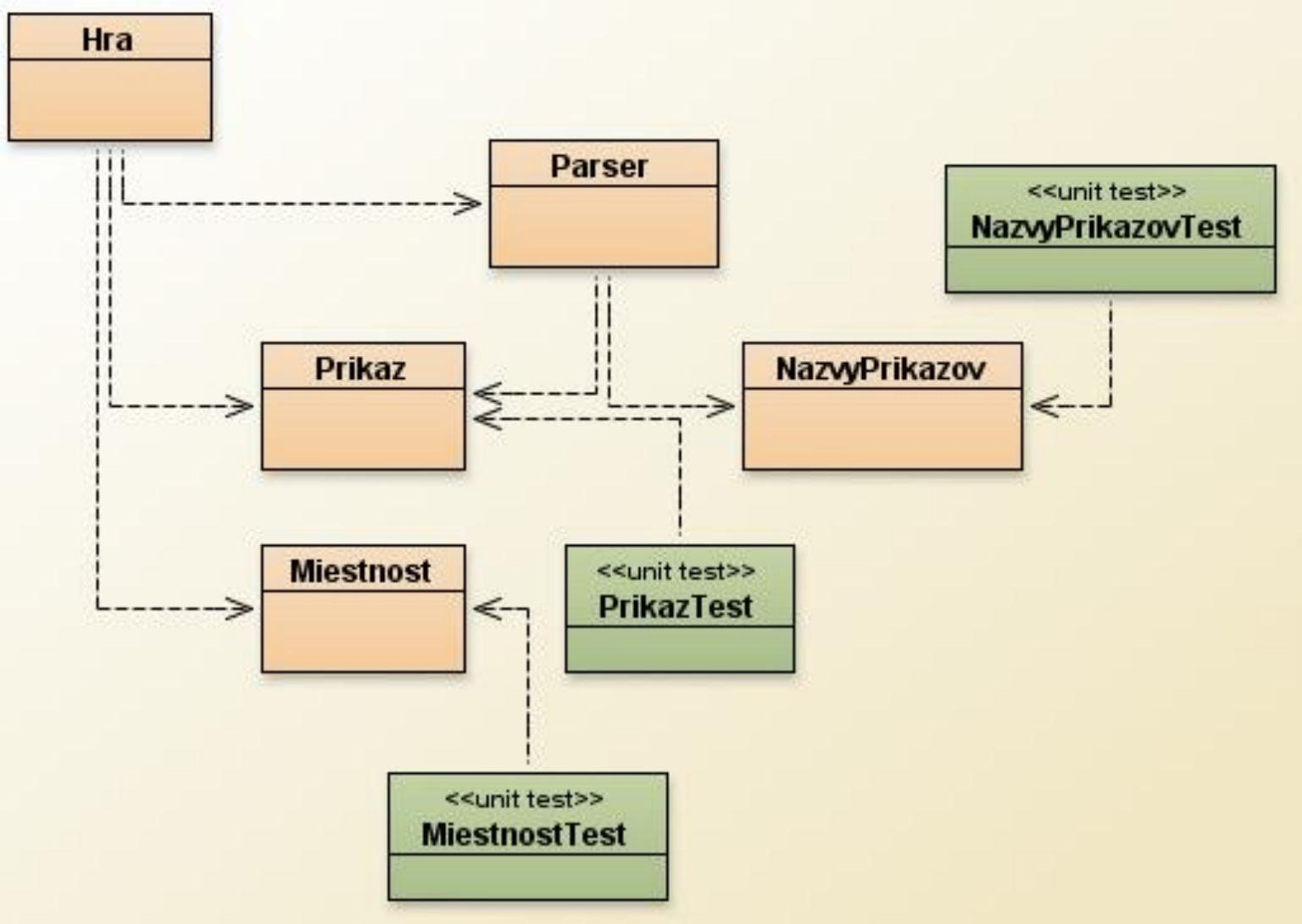
Trieda Hra

- hlavná trieda
- nastavuje hru
- vytvorí miestnosti
- nastaví miestnostiam východy
- vykonáva príkazy hráča

Návrh testov – regresné testovanie

- testy pred zmenami
- príprava kontroly korektnosti zmien
- úpravy existujúceho kódu po krokoch
- overenie správnosti každého kroku

Diagram tried s testovacími triedami



NazvyPrikazovTest

- testJePrikaz

- pozitívne testy na všetky platne príkazy
- negatívne testy na neplatné príkazy

PrikazTest

- testDajNazov
- testDajParameter
- testJeNeznamy
- testMaParameter

- pozitívne a negatívne testy častí príkazu

MiestnostTest

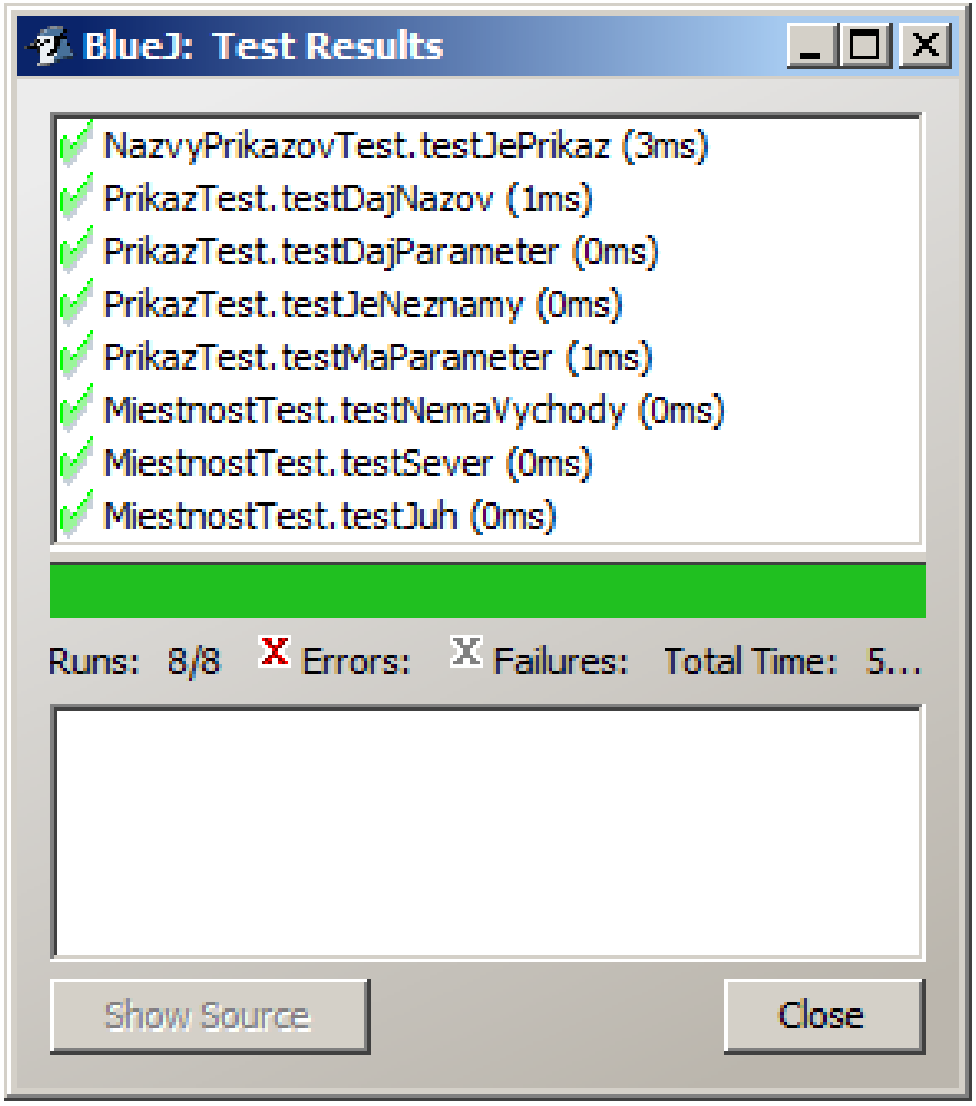
- testNemaVychody
- testSever
- testJuh

- testy nastavenia východov – priamy prístup k atribútom

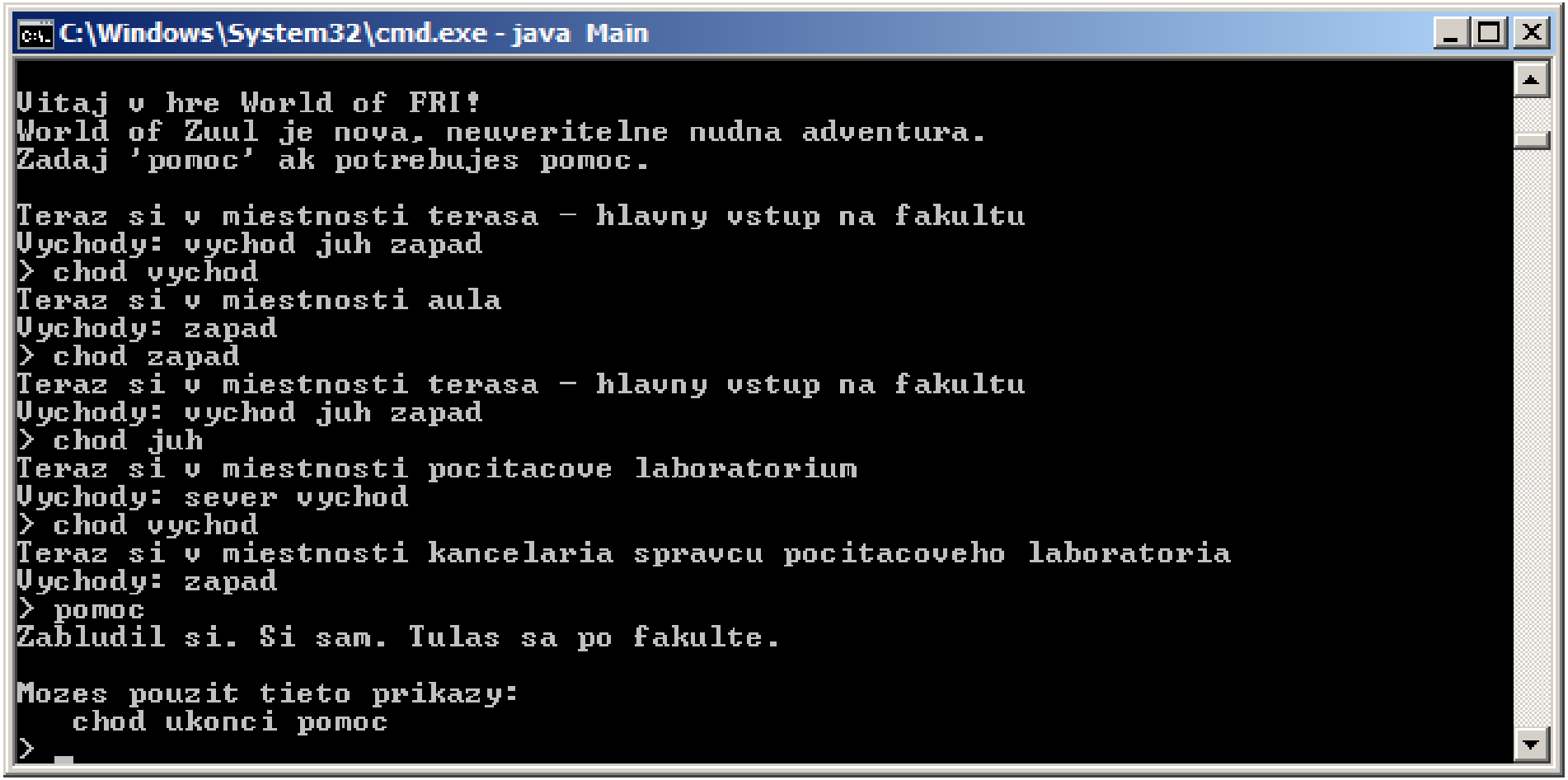
Parser, Hra

- nemajú testy
- komunikujú priamo s užívateľom cez terminál
- => nedajú sa automaticky testovať

Výsledky testov pred úpravami



Duplicita v existujúcom kóde(1)



Duplicita v existujúcom kóde(2)

- prehliadka kódu – duplicita v triede Hra

- metóda vypisPrivitanie – vypíše
  - privítanie
  - info o aktuálnej miestnosti a jej východoch

- metóda chodDoMiestnosti –
  - vyhľadá miestnosť v zadanom smere z aktuálnej
  - prejde do nej
  - vypíše info o aktuálnej miestnosti a jej východoch

Info o aktuálnej miestnosti(1)

```
System.out.println("Teraz si v miestnosti „
                                    + aAktualna.dajPopis());
System.out.print("Vychody: ");
if (aAktualna.aSevernyVychod != null) {
    System.out.print("sever ");
}
if (aAktualna.aVychodnyVychod != null) {
    System.out.print("vychod ");
}
...
```

Info o aktuálnej miestnosti

```
...

if (aAktualna.aJuznyVychod != null) {
    System.out.print("juh ");
}
if (aAktualna.aZapadnyVychod != null) {
    System.out.print("zapad ");
}
System.out.println();
```

Duplicita kódu(1)

- nežiaduci jav
- pri modifikáciách nutnosť úpravy kódu na viacerých miestach
- zväčšuje pravdepodobnosť vzniku chyby
- znižuje čitateľnosť a zrozumiteľnosť kódu

Duplicita kódu(2)

- indikuje zlý návrh – znižuje súdržnosť
  - chodDoMiestnosti
  - vypisPrivitanie
- odstraňovanie duplicít – jedna zo zásad udržovateľnosti kódu
- =>
- refaktoring: samostatná metóda pre duplicitný kód
- regresné testovanie

Refaktoring

- úprava fungujúceho kódu so zachovaním jeho funkčnosti

- nemení „čo objekt robí“
- mení „ako to objekt robí“

- zlepšenie udržovateľnosti kódu

Info o aktuálnej miestnosti

- nová metóda definovaná v triede Hra

```
private void vypisInfoMiestnosti()
{
   // pôvodne duplicitný kód
}
```

- v metódach vypisPrivitanie a chodDoMiestnosti poslanie správy

```
this.vypisInfoMiestnosti();
```

Regresné testovanie

- ! spustiť znovu všetky testy

Zadanie rozšírenia hry

- možné smery sú sever, východ, juh a západ
- rozšíriť hru o smery dole a hore

Postup práce

- nájsť časti kódu, kde sa smery priamo používajú
- redukcia týchto častí na nutné minimum
- refaktoring
- doplnenie nových smerov

Smery – trieda Miestnost

- atribút pre každý východ
- inicializácia v konštruktore
- metóda nastavVychody – nastavenie všetkých východov

Smery – trieda Hra

- metóda vytvorMiestnosti – vytvára miestnosti a nastavuje im východy (v smeroch)
- metóda chodDoMiestnosti – vyhľadá miestnosť v zadanom smere z aktuálnej a prejde do nej
- metóda vypisInfoMiestnosti – vypíše info o aktuálnej miestnosti aj s jej východmi

Metoda chodDoMiestnosti

```
Miestnost novaMiestnost = null;
if (smer.equals("sever")) {
    novaMiestnost = aAktualna.aSevernyVychod;
}
if (smer.equals("vychod")) {
    novaMiestnost = aAktualna.aVychodnyVychod;
}
if (smer.equals("juh")) {
    novaMiestnost = aAktualna.aJuznyVychod;
}
if (smer.equals("zapad")) {
    novaMiestnost = aAktualna.aZapadnyVychod;
}
```

Priamy prístup k atribútom

- priamy prístup hry k východom – atribútom miestnosti

- porušenie zapuzdrenia – základný princíp
- zvyšuje implementačnú závislosť

Závislosť Hra na Miestnost

- cieľ: odstrániť

- riešenie – posielanie správ miestnosti

Úpravy triedy Miestnost

- atribúty budú neverejné

- pridanie prístupovej metódy

```
public Miestnost dajVychod(String paSmer)
```

Telo metódy dajVychod(paSmer)

```
if (paSmer.equals("sever")) {
    return aSevernyVychod;
}
if (paSmer.equals("vychod")) {
    return aVychodnyVychod;
}
if (paSmer.equals("juh")) {
    return aJuznyVychod;
}
if (paSmer.equals("zapad")) {
    return aZapadnyVychod;
}
return null;
```

Výsledok refaktoringu

- ! nezabudnúť pustiť testy
- znížila sa implementačná závislosť triedy Hra na triede Miestnost
- pre zvýšenie počtu smerov, treba stále rozširovať triedy Hra a Miestnost

Trieda Miestnost – problémy

- východy sú definované ako atribúty
- => každá metóda v triede Miestnost musí vedieť zoznam smerov

- riešenie: uloženie smerov do zoznamu

- poznámka: s pôvodnými verejnými atribútmi by to bolo zložitejšie

Zoznam smerov

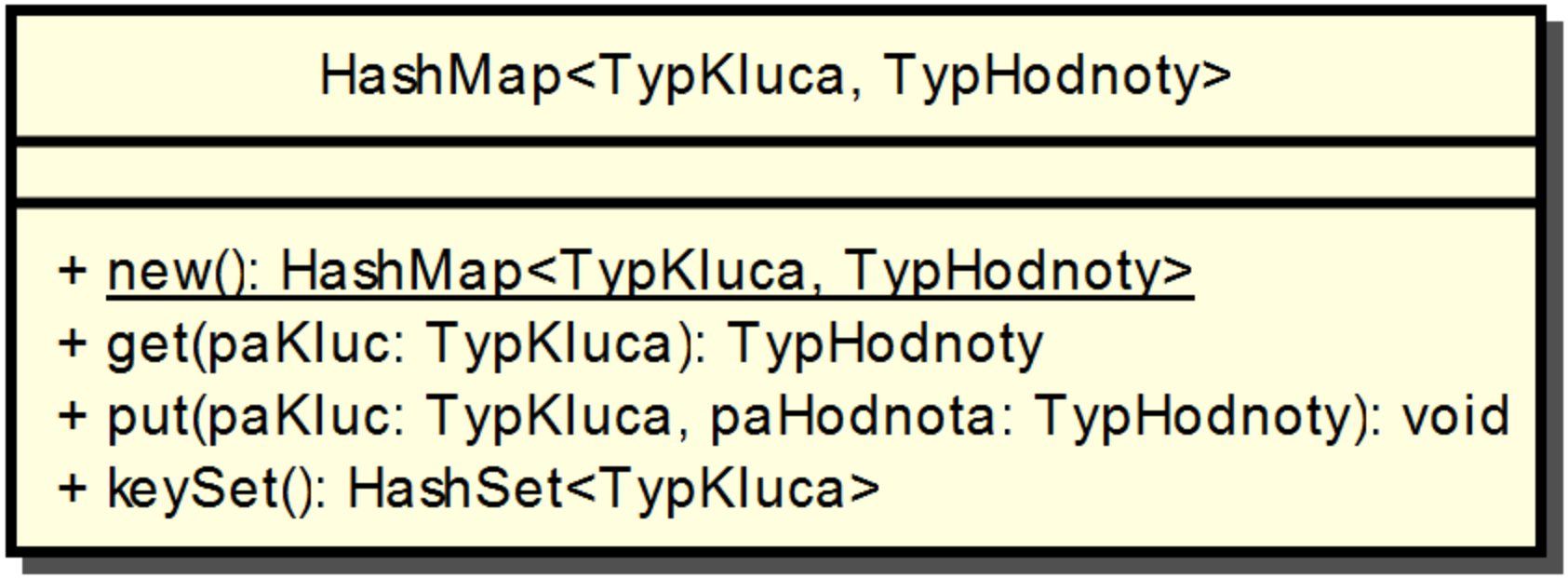
- kontajner
- treba vyhľadávať
  - ArrayList, pole – komplikované; nutnosť vyhľadávania v cykle

- riešenie kontajner HashMap

Kontajner HashMap(1)

- neusporiadaná množina dvojíc
- dvojica = (kľúč; hodnota)
- možnosť vyhľadávania podľa kľúča
- hodnota kľúča musí byť unikátna
- generická trieda
  - možnosť meniť typ kľúča
  - možnosť meniť typ hodnoty

Kontajner HashMap$_{(2)}$



Správa keySet objektom HashMap

- keySet vytvára množinu všetkých kľúčov
- zoznam je neusporiadaný

- množina – nedá sa pristupovať pomocou indexu
- dá sa prechádzať pomocou cyklu foreach

Trieda Miestnost

- zmena implementácie
- rozhranie bezo zmien
  - trieda Hra – žiadne zmeny

Atribúty v triede Miestnost

```
private Miestnost aSevernyVychod;
private Miestnost aJuznyVychod;
private Miestnost aVychodnyVychod;
private Miestnost aZapadnyVychod;
```

```
private HashMap<String, Miestnost> aVychody;
```

názov smeru

miestnosť v smere

Konštruktor v triede Miestnost

```
aSevernyVychod = null;
aJuznyVychod = null;
aVychodnyVychod = null;
aZapadnyVychod = null;
```

```
aVychody = new HashMap<String, Miestnost>();
```

Metóda dajVychod(paSmer)

```
if (paSmer.equals("sever")) {
    return aSevernyVychod;
}
...
return null;
```

```
return aVychody.get(paSmer);
```

Smery – priame použitie

- trieda Hra
  - vytvorMiestnosti – vytvára miestnosti a nastavuje im východy
  - vypisInfoMiestnosti – vypíše info o aktuálnej miestnosti aj s jej východmi
- trieda Miestnost
  - nastavVychody – nastavenie jednotlivých východov
  - rapídne sa zmenšil počet výskytov priameho použitia smerov = 1

Ďalšie zmeny

- metóda nastavVychody v triede Miestnost
  - parametre = zoznam všetkých východov
  - nastavovanie každého východu osobitne

- komplikácia pri pridávaní nového smeru
- zmena na metódu nastavVychod(paSmer, paMiestnost)
- zmena rozhrania

nastavVychod v triede Miestnost

```
public void nastavVychod(String paSmer,
                         Miestnost paMiestnost)
{
    aVychody.put(paSmer, paMiestnost);
}
```

Dôsledky pre triedu Hra

- dôsledok zmeny rozhrania triedy Miestnost
- metóda vytvorMiestnosti
  - vytvára jednotlivé miestnosti
  - nastavuje východy
- náhrada správy nastavVychody miestnosti na postupnosťou správ nastavVychod miestnosti

Metóda dajVychod(paSmer)

```
terasa.nastavVychody(null, aula, labak, bufet);
```

```
terasa.nastavVychod("vychod", aula);
terasa.nastavVychod("juh", labak);
terasa.nastavVychod("sever", bufet);
```

Čo zostáva

- smery sú priamo využité na dvoch miestach
  - metóda vytvorMiestnosti
    - musí obsahovať smery
  - metóda vypisInfoMiestnosti

- odstrániť z vypisInfoMiestnosti
- presun vytvárania popisu východov do triedy Miestnost
  - zmena rozhrania

Návrh tried určený zodpovednosťou

- atribút pre východy – miestnosť
- výpis informácie o miestnosti – hra

- porušenie RDD
- informácie o sebe poskytuje miestnosť

Trieda Miestnost – dajPopisVychodov

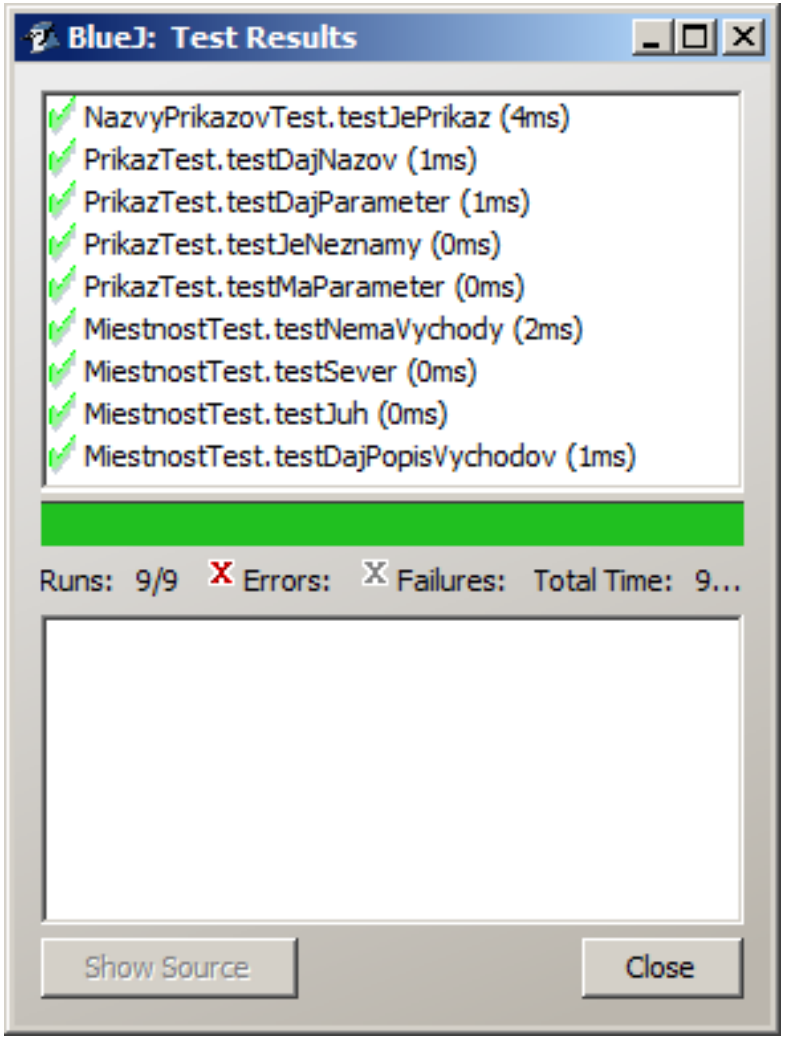
```
public String dajPopisVychodov()
{
    String vychody = "Vychody:";
    for (String smer : aVychody.keySet()) {
        vychody = vychody + " " + smer;
    }
    return vychody;
}
```

Dôsledok pre triedu Hra

```
private void vypisInfoMiestnosti()
{
   System.out.println("Teraz si v miestnosti " +
                                 aAktualna.dajPopis());
   System.out.println(aAktualna.dajPopisVychodov());
   System.out.println();
}
```

Nová verejná metóda = nový test

- testDajPopisVychodov do testovacej triedy MiestnostTest



Smery – priame použitie

- trieda Hra
  - vytvorMiestnosti – vytvára miestnosti a nastavuje im východy

- smery priamo použité už len na jednom mieste

Pridanie nových smerov

```
...
aula.nastavVychod("dole", jedalen);
jedalen.nastavVychod("hore", aula);
...
```

Zhodnotenie riešenia

- smery hore, dole boli pridané

- jednoduchá možnosť pridať iné nové smery (napr. teleport)
- jediné miesto – metóda vytvorMiestnosti

Zmeny softvéru

- Softvér nie je román
  - román sa napíše len raz
  - Krstný otec, tretie aktualizované vydanie

- úspešný softvér sa stále opravuje, rozširuje.
- neudržovaný softvér „umiera“.

Kvalita kódu

- implementačná závislosť
- súdržnosť
- jasne definovaná zodpovednosť

- kvalitný kód
  - neobsahuje duplicitu
  - dodržiava zapuzdrenie

Vďaka za pozornosť