Informatika 2

Polymorfizmus

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

Pojmy zavedené v 2. prednáške(1)

- implementačná závislosť – minimálna
- súdržnosť – maximálna
- zodpovednosť – jedna jasne definovaná
  - miery dobrého návrhu tried
- duplicita kódu – súdržnosť
- priamy prístup k atribútom – závislosť, zodpovednosť

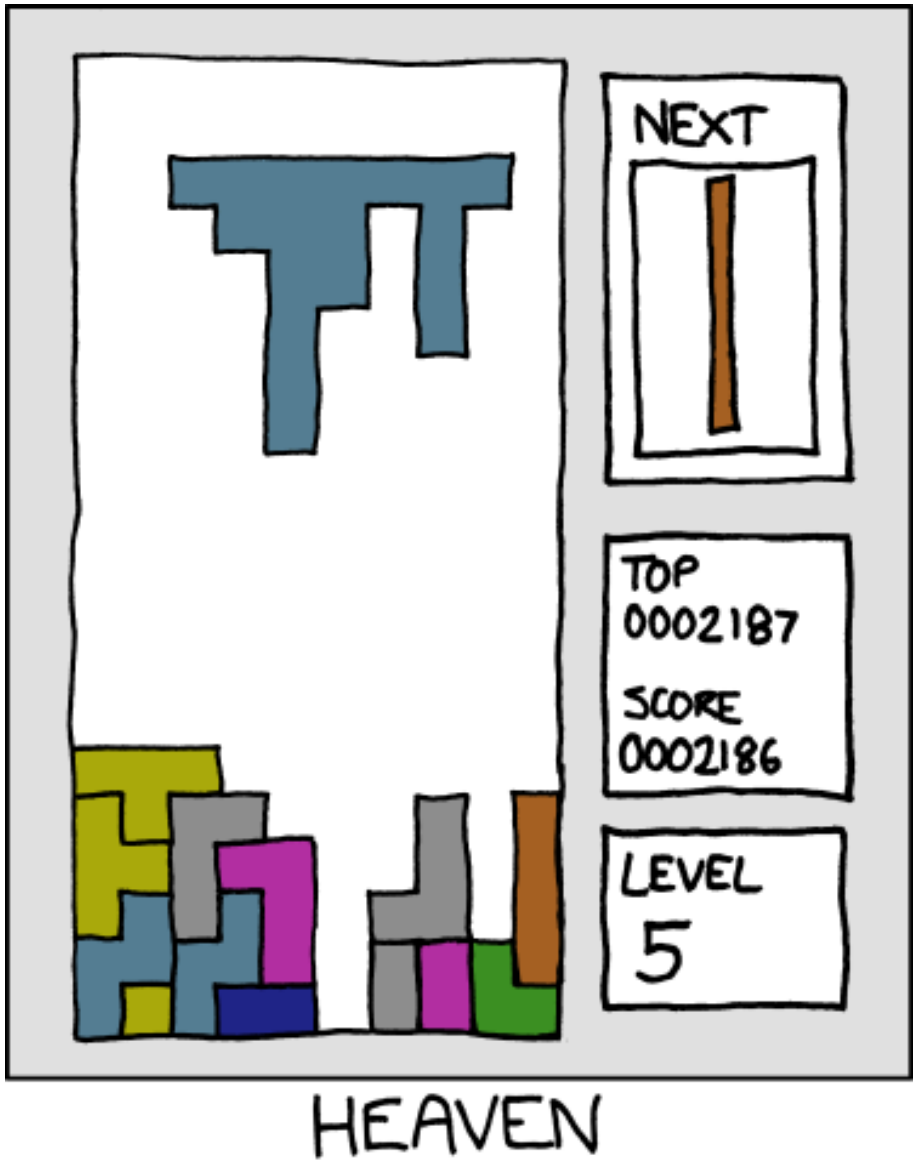
Pojmy zavedené v 2. prednáške(2)

- refaktoring
- zmena kódu bez zmeny jeho funkcie
- úpravy pre zlepšenie miery závislosti, súdržnosti a zodpovednosti
- overenie korektnosti refaktoringu – regresné testovanie

Pojmy zavedené v 2. prednáške(3)

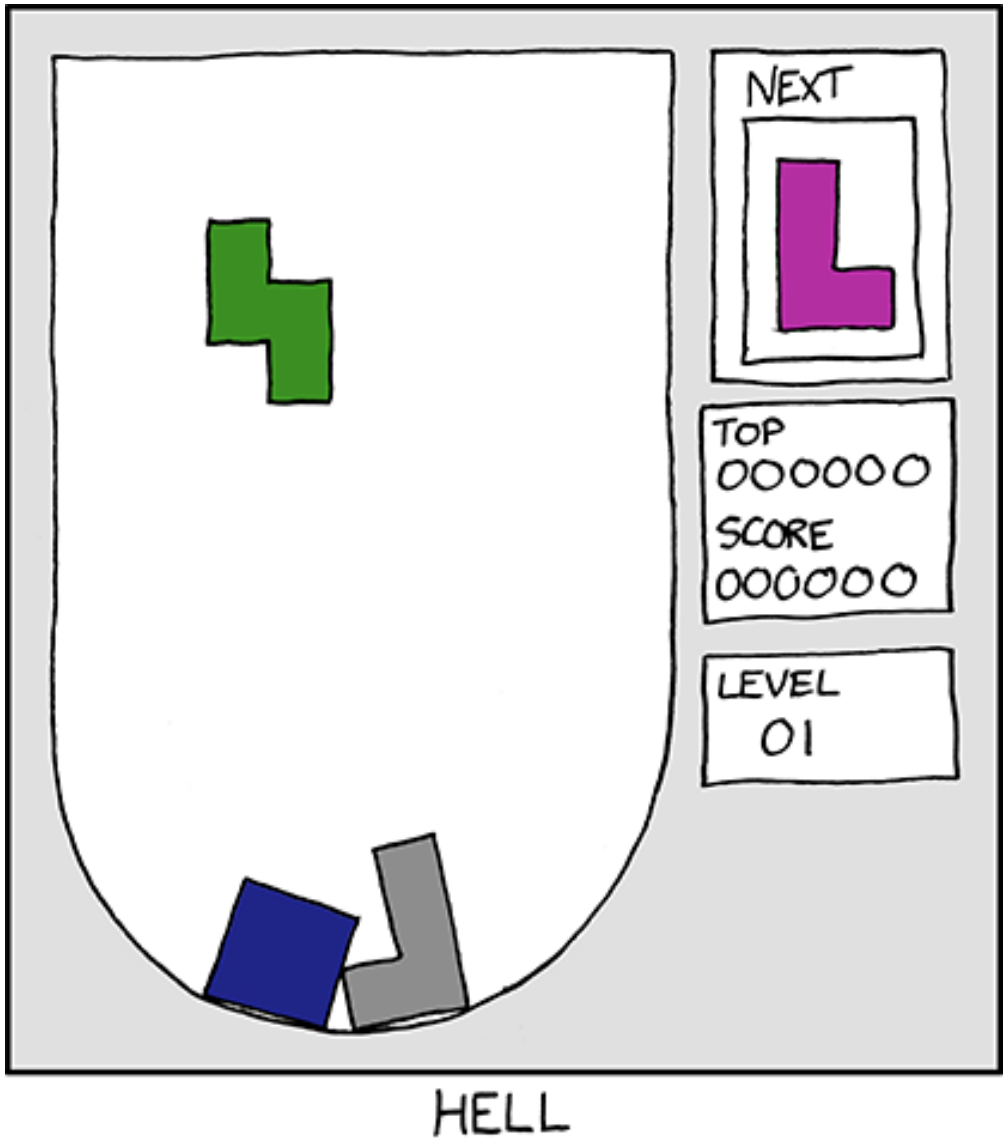
- HashMap – kontejner
- neusporiadaná množina usporiadaných dvojíc (kľúč, hodnota)
- kľúč – unikátny
- poskytuje hodnoty určené kľúčom
- prechádzanie kontejnerom – cyklus for each

Ako sa úpravy programu tvária



http://xkcd.com/888/

Aké sú v skutočnosti



http://xkcd.com/724/

Cieľ prednášky

- polymorfizmus
- interface

- príklad: hra „World of FRI“

„World of FRI“

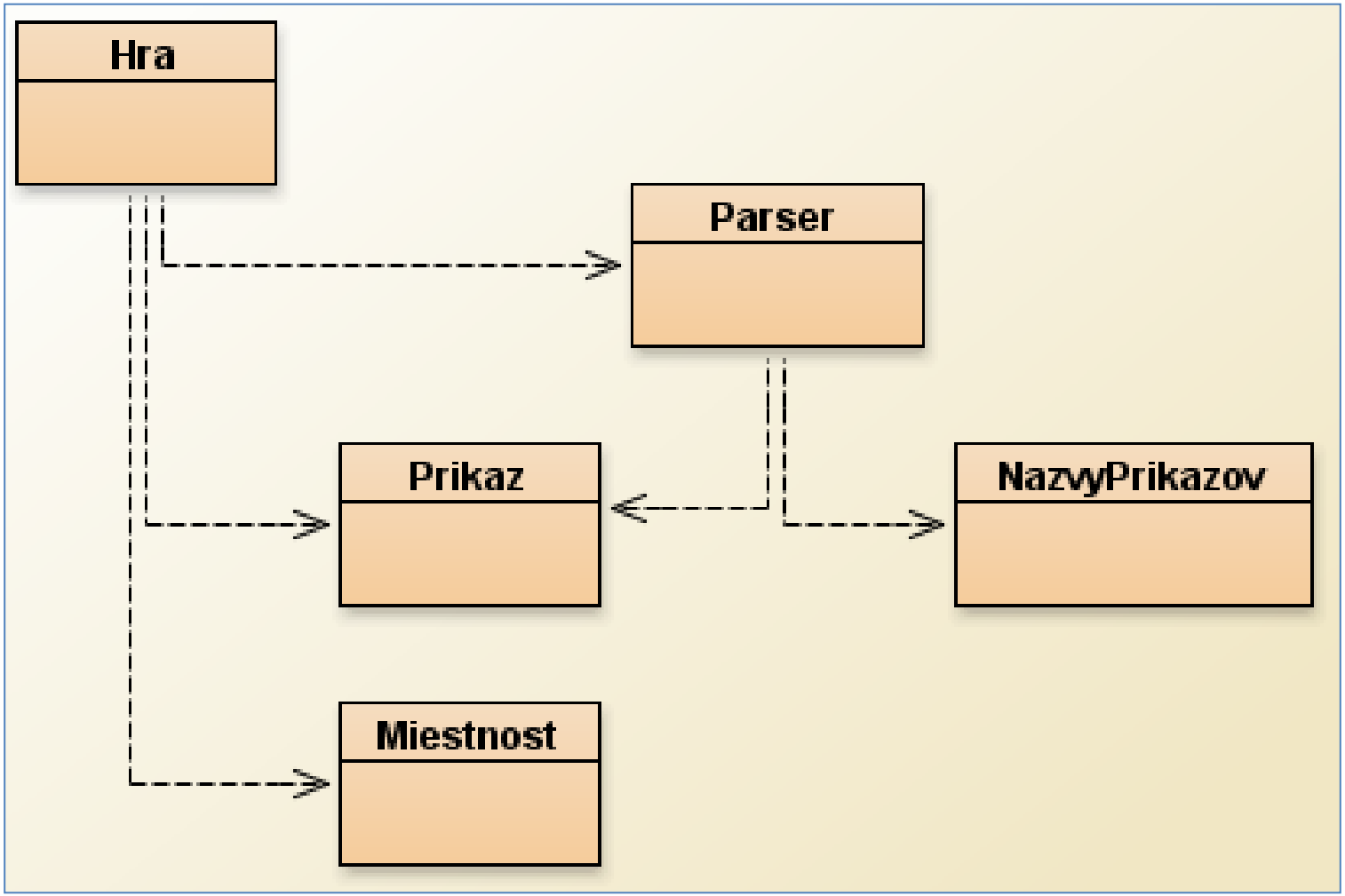
- ľubovoľné smery
- bez implementačnej závislosti
- výsledok refaktoringu

Nová úloha

- pridať nový príkaz „ukaz“
- do terminálového okna sa vypíše informácia o miestnosti, v ktorej sa hráč nachádza

- kde sa pracuje s príkazmi?

WoF – diagram tried



Kde sa pracuje s príkazmi?

- trieda NazvyPrikazov
  - udržiava zoznam názvov príkazov
  - odpovedá na otázku „jePrikaz“
- trieda Parser
  - načítava text príkazu z okna terminálu
  - rozkladá ich na dvojice príkaz, parameter
  - požiada o inštanciu triedu Prikaz

Kde sa pracuje s príkazmi?

- trieda Prikaz
  - uchováva názov príkazu a prípadný parameter
- trieda Hra
  - vykonáva príkazy

Úlohy triedy Hra

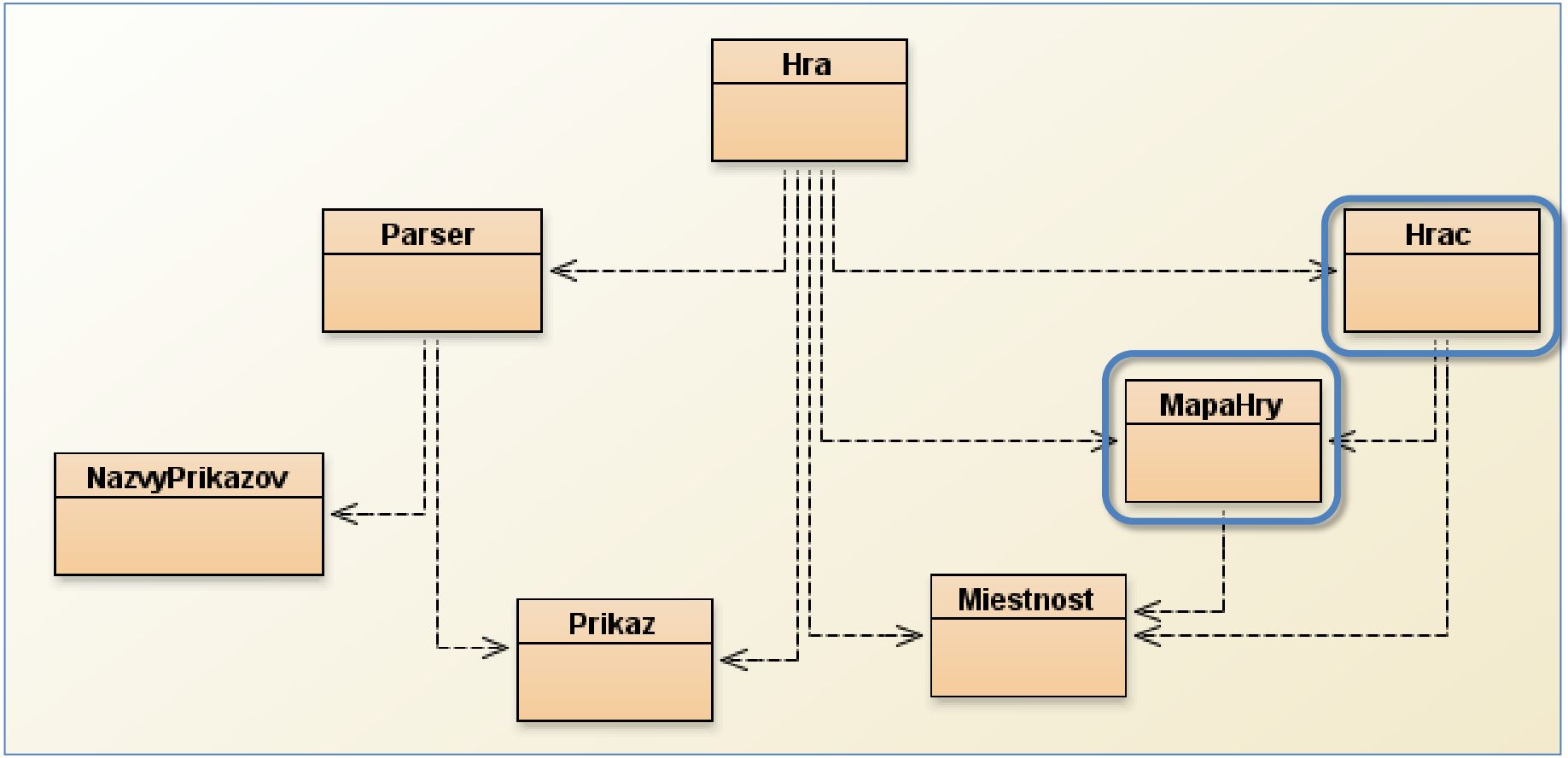
1. Riadenie priebehu hry
2. Udržiavanie informácii o hráčovi (aktuálna miestnosť)
3. Vytváranie miestností – „mapa hry“
4. Vykonávanie príkazov

- veľa úloh – refaktoring

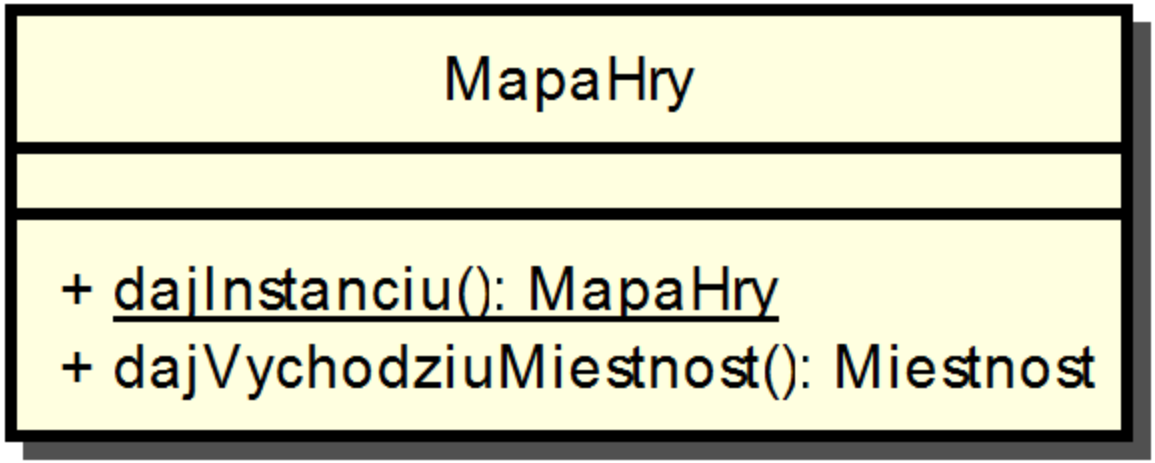
Refaktoring triedy Hra(1)

- informácie o hráčovi do samostatnej triedy
- vytvorenie mapy hry do samostatnej triedy

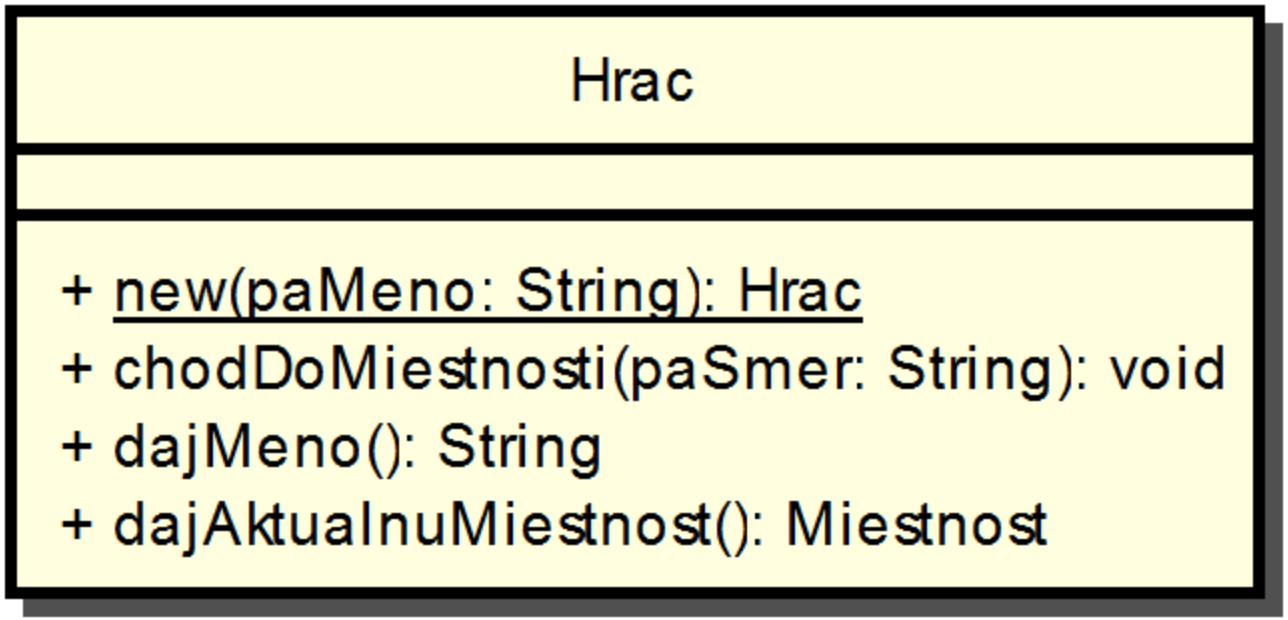
Refaktoring triedy Hra(1)



Trieda MapaHry



Trieda Hrac


Hrac
+ new(paMeno: String): Hrac
+ chodDoMiestnosti(paSmer: String): void
+ dajMeno(): String
+ dajAktualnuMiestnost(): Miestnost

Vykonávanie príkazov

- Hra – metóda vykonajPrikaz

→

- možnosti:
  - trieda Prikaz
  - nová trieda

- vykonávanie príkazu –> práca s dátami príkazu –> zodpovednosť triedy Prikaz

Metóda vykonajPrikaz v triede Hra(1)

```
private boolean vykonajPrikaz(Prikaz paPrikaz)
{
    boolean jeKoniec = false;
    if (paPrikaz.jeNeznamy()) {
        System.out.println("Nerozumiem");
        return false;
    }
...
```

Metóda vykonajPrikaz v triede Hra(2)

```
...
    String nazovPrikazu = paPrikaz.dajNazov();
    if (nazovPrikazu.equals("pomoc")) {
        vypisNapovedu();
    } else if (nazovPrikazu.equals("chod")) {
        chodDoMiestnosti(paPrikaz);
    } else if (nazovPrikazu.equals("ukonci")) {
        jeKoniec = ukonciHru(paPrikaz);
    }
    return jeKoniec;
}
```

Závyslosti metódy vykonajPrikaz

- metóda vykonajPrikaz v hre je závislá od metód:
  - vypisNapovedu
  - chodDoMiestnosti
  - ukonciHru

- tieto metódy treba preniesť tiež

Sprístupnenie hráča v triede Prikaz

- v hre je hráč atributom aHrac
- v príkaze ako parameter metódy vykonaj

- v hre je príkaz parametrom metódy vykonajPrikaz
- v príkaze sú vlastnosti príkazu priamo prístupné

Metóda vykonaj v triede Prikaz$_{(1)}$

```
public boolean vykonaj (Hrac paHrac)
{
    boolean jeKoniec = false;
    if(jeNeznamy()) {
        System.out.println("Nerozumiem");
        return false;
    }
...
```

Metóda vykonaj v triede Prikaz$_{(2)}$

```
...
    String nazovPrikazu = dajNazov();
    if (nazovPrikazu.equals("pomoc")) {
        vypisNapovedu();
    } else if (nazovPrikazu.equals("chod")) {
        chodDoMiestnosti(paHrac);
    } else if (nazovPrikazu.equals("ukonci")) {
        jeKoniec = ukonciHru();
    }
    return jeKoniec;
}
```

Kde sa pracuje s príkazmi?

- trieda Prikaz
  - uchováva názov príkazu a prípadný parameter
  - vykonáva príkazy
- trieda Hra
  - žiada príkaz o vykonanie (ako celok)

Úlohy triedy Hra

1. Riadenie priebehu hry

- refaktoring dokončený
- rozšírenie funkcií hry – príkaz ukaz

Nový príkaz „ukaz“

- dve rôzne riešenia
  - nový príkaz do triedy Prikaz
  - nové triedy – vykonanie každého príkazu

Nový príkaz do triedy Prikaz

- pozitívum riešenia
  - mechanické rozšírenie súčasného stavu
- negatíva riešenia
  - zväčší sa veľkosť triedy
  - zhoršuje sa udržiavateľnosť

Triedy na vykonávanie

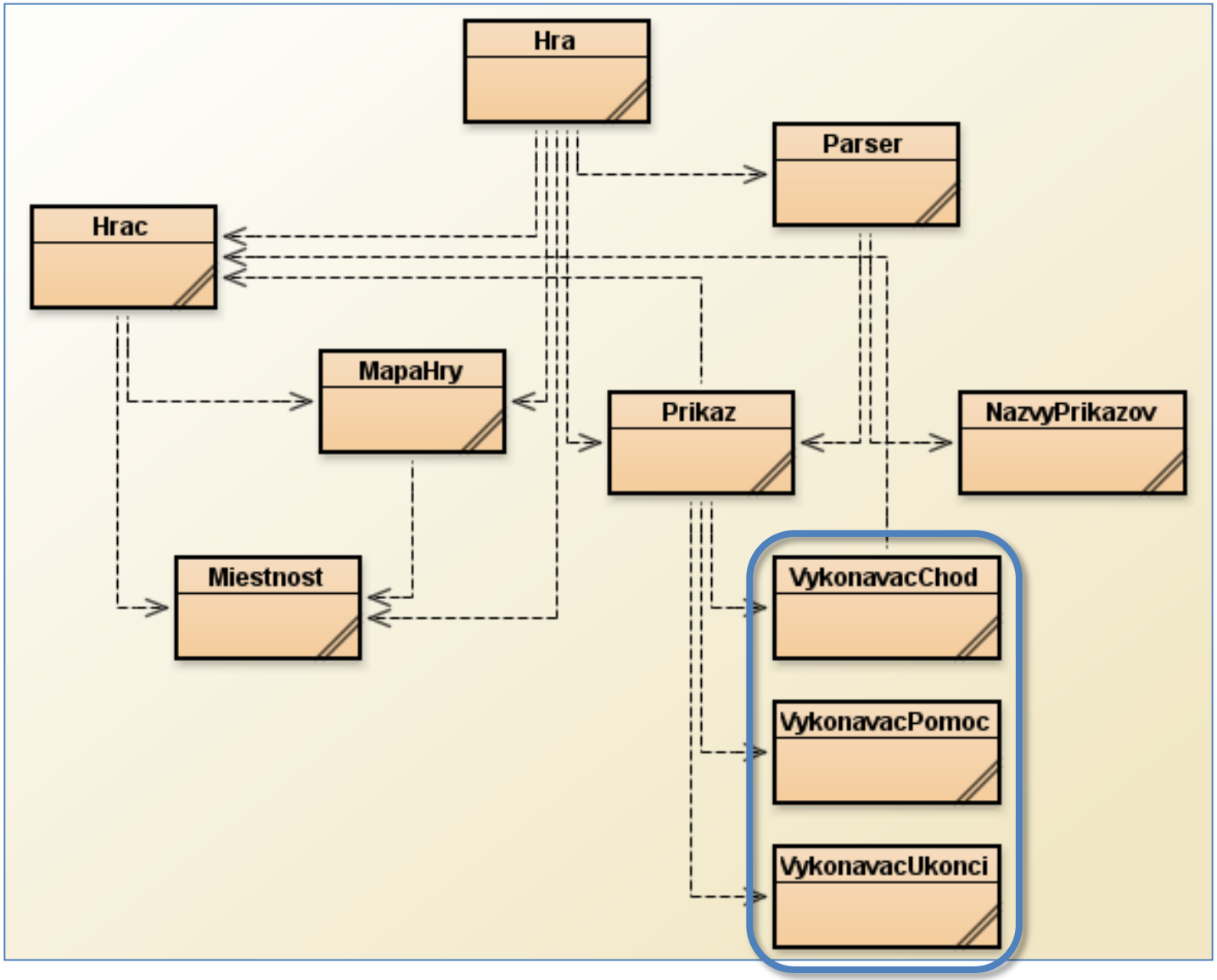
- riešenie – vykonávanie každého príkazu (podľa názvu) v samostatnej triede

- pozitívum riešenia
  - každá trieda má jasne určenú zodpovednosť
  - pridanie príkazov nemá vplyv na už zavedené
- negatívum riešenia
  - väčší počet tried

Refaktoring – triedy vykonavacov

- VykonavacChod
  - void vykonaj(Hrac paHrac, String paParameter)
- VykonavacPomoc
  - void vykonaj()
- VykonavacUkonci
  - boolean vykonaj(String paParameter)

Model tried „WoF“



Ukážka novej triedy

```
public class VykonavacPomoc
{
    public void vykonaj()
    {
        System.out.println("Zabludil si. Si sam. Tulas sa po fakulte.");
        System.out.println();
        System.out.println("Mozes pouzit tieto prikazy:");
        System.out.println("  chod ukonci pomoc");
    }
}
```

Metóda vykonaj v triede Prikaz

```
if (nazovPrikazu.equals("pomoc")) {
    vypisNapovedu();
} else if (nazovPrikazu.equals("chod")) {
...
```

```
if (nazovPrikazu.equals("pomoc")) {
    VykonavacPomoc vykonavac;
    vykonavac = new VykonavacPomoc();
    vykonavac.vykonaj();
} else if (nazovPrikazu.equals("chod")) {
...
```

Metóda vykonaj v triede Prikaz – cieľ

```
public boolean vykonajPrikaz(Hrac paHrac)
{
    if (jeNeznamy()) {
        System.out.println("Nerozumiem");
        return false;
    }

    Vykonavac vykonavac;
    vykonavac = aPrikazy.dajVykonavac(aNazov);
    return vykonavac.vykonaj(paHrac, aParameter);
}
```

Polymorfizmus(1)

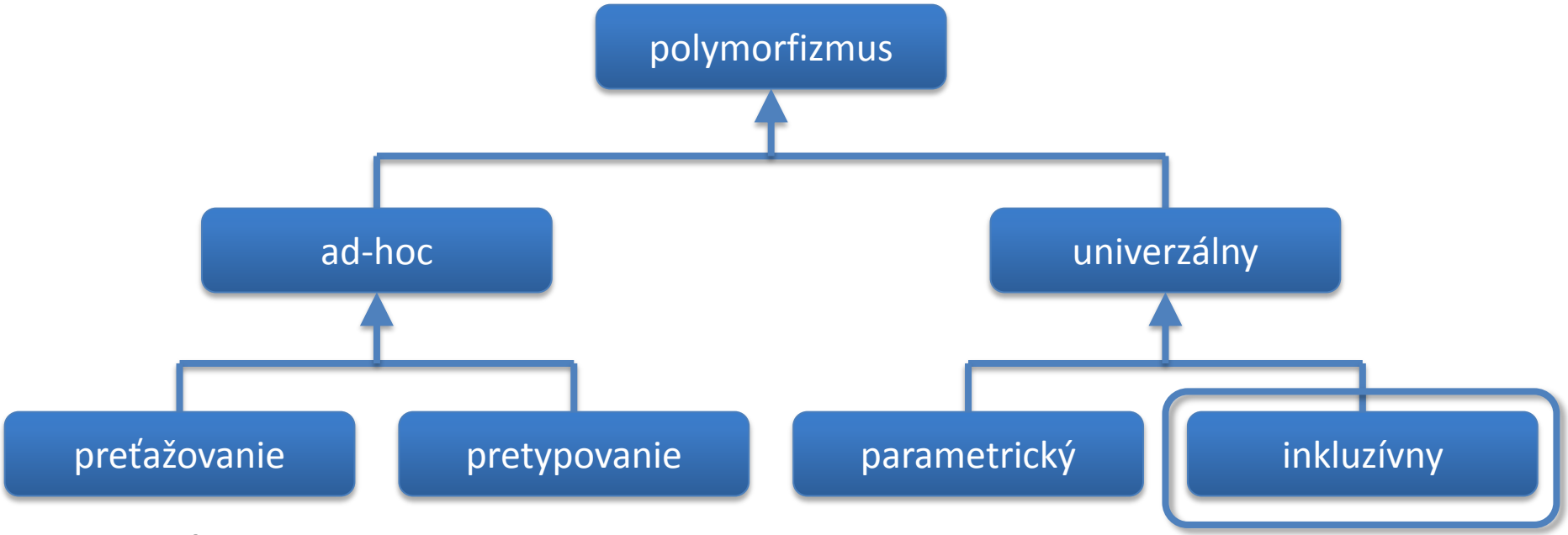
- riešenie možné vďaka polymorfizmu

- polymorfizmus:
  využitie určitej špeciálnej situácie, keď dva rôzne objekty môžu prijať tú istú správu a pritom každý z nich reaguje inak – použije inú metódu

Polymorfizmus(1)

- odosielateľ správy sa nestará o typ adresáta
  - zjednodušenie na strane odosielateľa
- postačuje, že adresát je schopný správu prijať
  - má správu v rozhraní
- každý adresát použije svoju metódu

Polymorfizmus - druhy

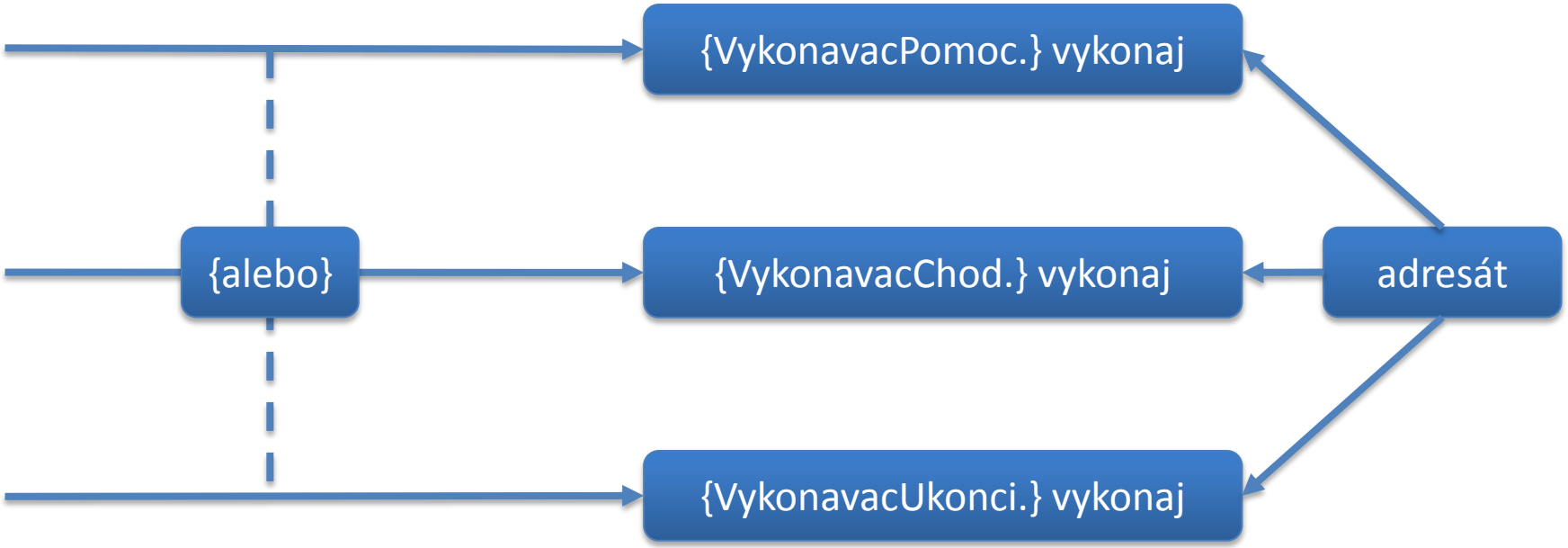


- preťažovanie – metódy a konštruktory s rôznym typom alebo počtom parametrov
- pretypovávanie – autoboxing
- parametrický – generické triedy

Protokol: správa → metóda

- jedna správa

- rôzne metódy



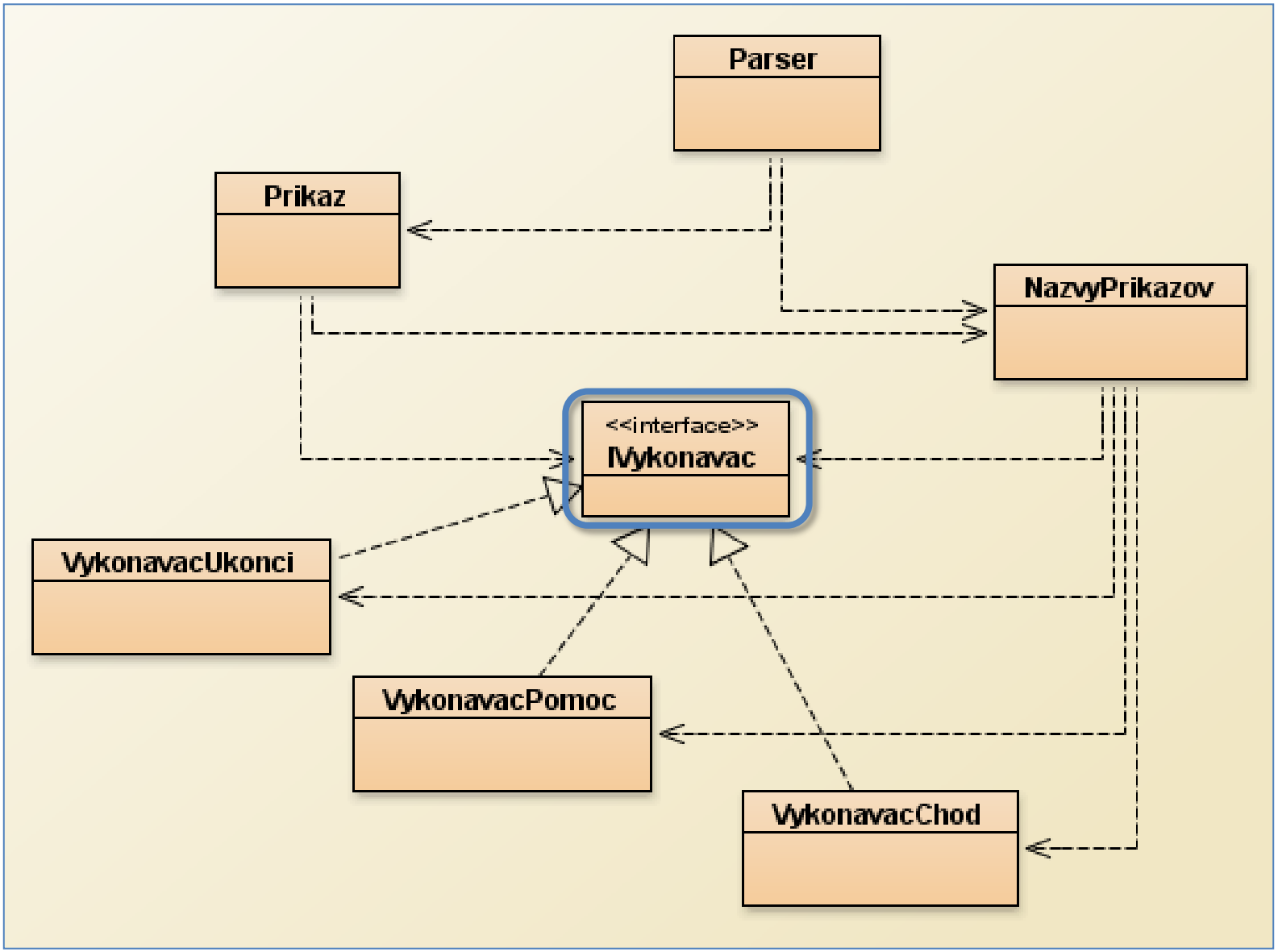
Prostriedky jazyka Java

- správa v rozhraní – podmienka pre jej poslanie

- interface
  - explicitná definícia zoznamu správ v rozhraní
  - definuje typ objektových premenných

- trieda implementuje interface
  - zaručuje, že má implementované všetky metódy pre správy uvedené v interface
  - inštancie sú schopné chovať sa polymorfne - inkluzívny

Aplikácia interface do hry



Interface IVykonavac

```
public interface IVykonavac
{
    boolean vykonaj(Hrac paHrac, String paParam);
}
```

- kľúčové slovo interface
- správy bez modifikátora prístupu ukončené ;
- vždy public

Implementácia rozhrania

- explicitné uvedenie inteface v hlavičke triedy
- ľubovoľný počet implementovaných interface

- metódy pre všetky správy z interface
  - musia byť verejné
  - správy sa musia presne zhodovať

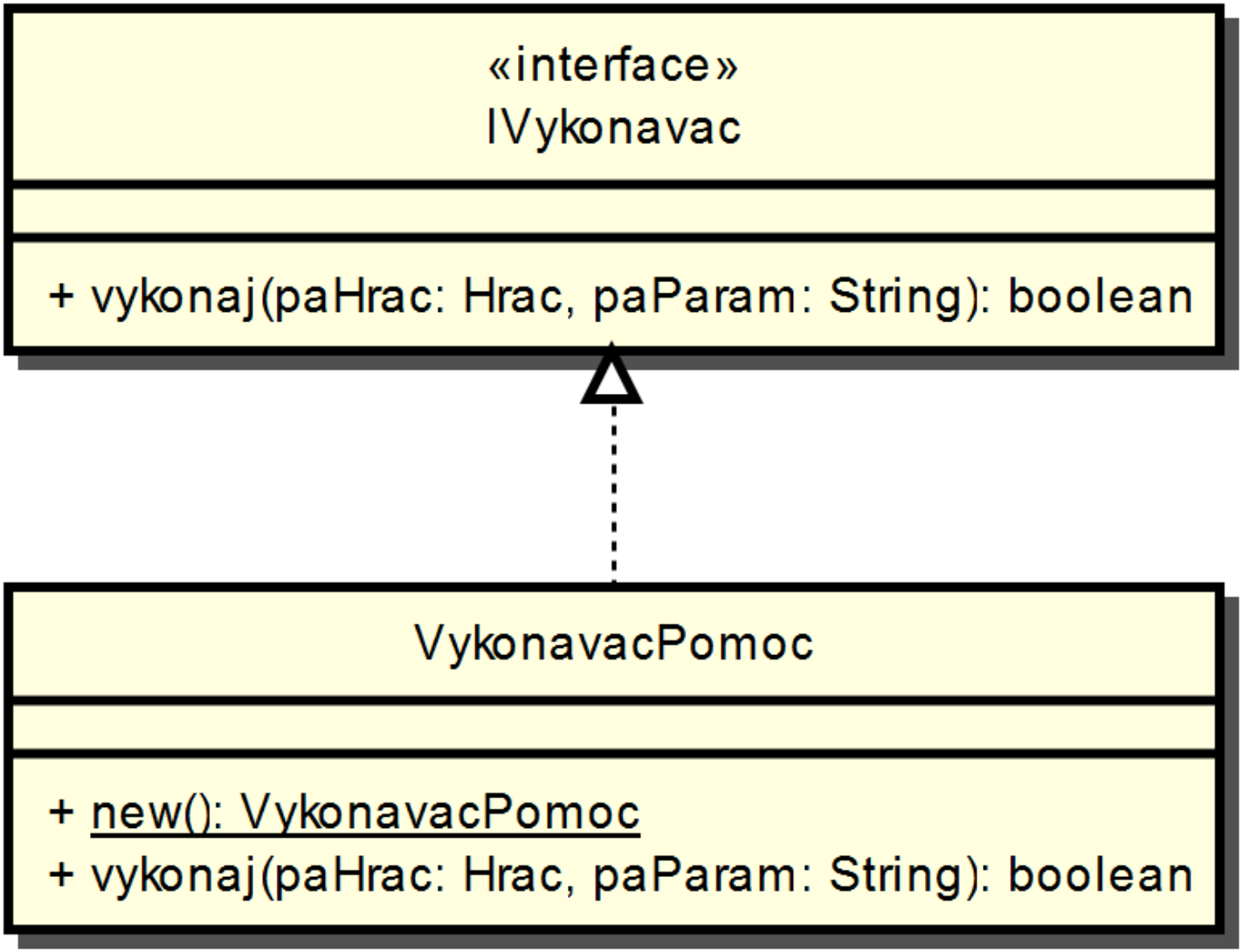
Java – implementácia rozhrania

```
public class VykonavacPomoc implements IVykonavac
{
    public boolean vykonaj(Hrac paHrac, String paParam)
    {
        System.out.println("Zabludil si. Si sam.");
        System.out.println();
        System.out.println("Mozes pouzit tieto prikazy:");
        System.out.println("   chod ukonci pomoc");
        return false;
    }
}
```

Interface v UML(1)

- obdĺžnik – ako trieda
- stereotyp «interface» nad menom
- len správy inštancii

Interface v UML(2)



Poslanie správy prem. typu interface

```
IVykonavac vykonavac;
vykonavac = aPrikazy.dajVykonavac(aNazov);
vykonavac.vykonaj(paHrac, aParameter);
```

Statický kontra dynamický typ(1)

- statický typ
  - uvedený v definícii objektovej premennej
  - určuje sa pri preklade
  - určuje množinu správ, ktoré je možno poslať (iné nie)
- dynamický typ
  - skutočný typ objektu referencovaného premennou
  - určuje sa za behu pri priradení hodnoty premennej
  - určuje chovanie objektu, reakcie na správy

Statický kontra dynamický typ(2)

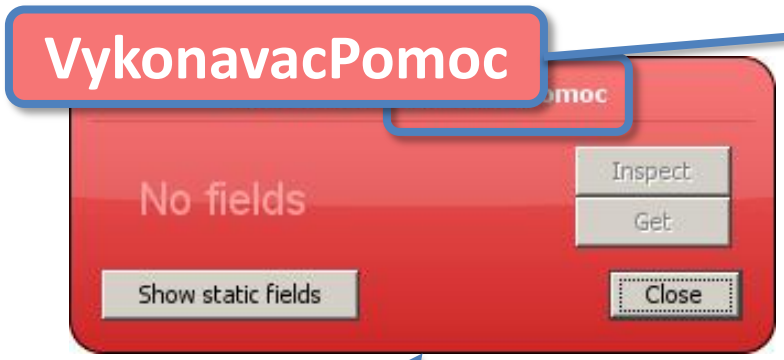
statický typ

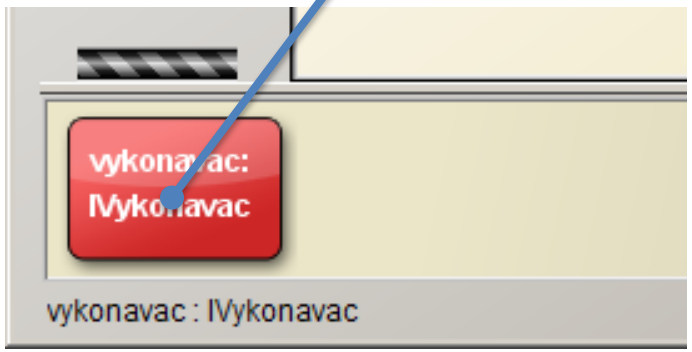
```
IVykonavac vykonavac;
vykonavac = aPrikazy.dajVykonavac(aNazov);
vykonavac.vykonaj(paHrac, aParameter);
```

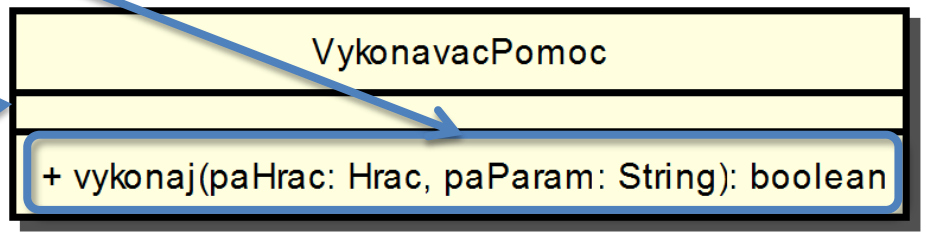
dynamický typ
nevieme odvodiť z príkazu
môže byť
VykonavacUkonci, VykonavacPomoc, VykonavacChod

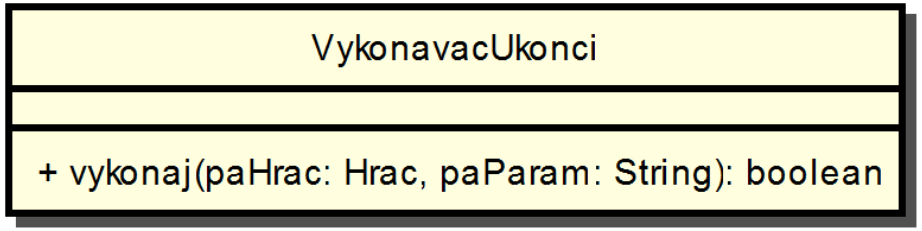
Spracovanie správy pri polymorfizme(1)

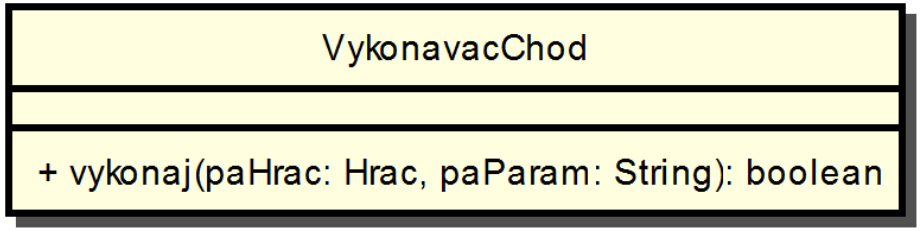
```
vykonavac.vykonaj(paHrac, aParameter);
```





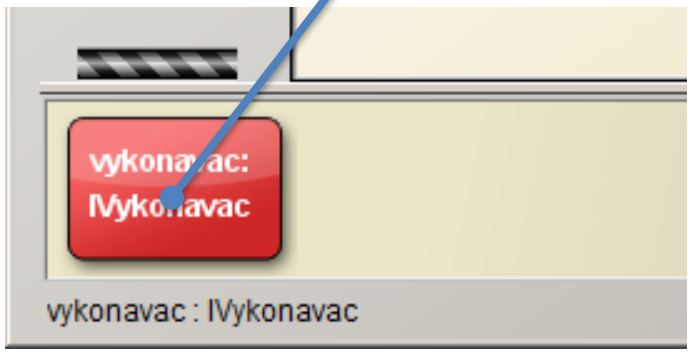






Spracovanie správy pri polymorfizme(1)

vykonavac.vykonaj(paHrac, aParameter);



VykonavacPomoc

+ vykonaj(paHrac: Hrac, paParam: String): boolean

VykonavacUkonci

+ vykonaj(paHrac: Hrac, paParam: String): boolean

VykonavacChod

+ vykonaj(paHrac: Hrac, paParam: String): boolean

Statický kontra dynamický typ

- príkaz bol rovnaký
- statický typ je rovnaký

- vykonávajú sa rôzne metódy

Metóda dajVykonavac v NazvyPrikazov

```
public IVykonavac dajVykonavac(String paNazov)
{
    if (paNazov.equals("pomoc")) {
        return new VykonavacPomoc();
    } else if (paNazov.equals("chod")) {
        return new VykonavacChod();
    } else if (paNazov.equals("ukonci")) {
        return new VykonavacUkonci();
    }
    return null;
}
```

Typová kompatibilita a interface

- inštancia je typovo kompatibilná s interface ak jej trieda implementuje daný interface

- hodnota null je typovo kompatibilná so všetkými interface

Typová kompatibilita v dajVykonavac(1)

```
public IVykonavac dajVykonavac(String paNazov)
{
    if (paNazov.equals("pomoc")) {
        return new VykonavacPomoc();
    } else if (paNazov.equals("chod")) {
        return new VykonavacChod();
    } else if (paNazov.equals("ukonci")) {
        return new VykonavacUkonci();
    }
    return null;
}
```

class VykonavacUkonci implements IVykonavac

Typová kompatibilita v dajVykonavac(2)

```
public IVykonavac dajVykonavac(String paNazov)
{
    if (paNazov.equals("pomoc")) {
        return new VykonavacPomoc();
    } else if (paNazov.equals("chod")) {
        return new VykonavacChod();
    } else if (paNazov.equals("ukonci")) {
        return new VykonavacUkonci();
    }
    return null;
}
```

Zjednodušenie pomocou HashMap

- inštancie vykonávačov nemajú atribúty
- => nemenia svoj stav
- => všetky inštancie budú mať rovnaký stav
- => možno ich predpripraviť do zoznamu

Trieda NazvyPrikazov s HashMap(1)

```
private final HashMap<String, IVykonavac> aVykonavace;

public NazvyPrikazov()
{
    aVykonavace = new HashMap<String, IVykonavac>();

    aVykonavace.put("pomoc", new VykonavacPomoc());
    aVykonavace.put("chod", new VykonavacChod());
    aVykonavace.put("ukonci", new VykonavacUkonci());
}
```

Trieda NazvyPrikazov s HashMap(1)

```
public boolean jePrikaz(String paNazov)
{
    return aVykonavace.containsKey(paNazov);
}

public IVykonavac dajVykonavac(String paNazov)
{
    return aVykonavace.get(paNazov);
}
```

Nový príkaz ukaz

- refaktoring
- jednoduché pridávanie príkazov
  1. nová trieda – vykonávač
  2. zaradenie inštancie do HashMap

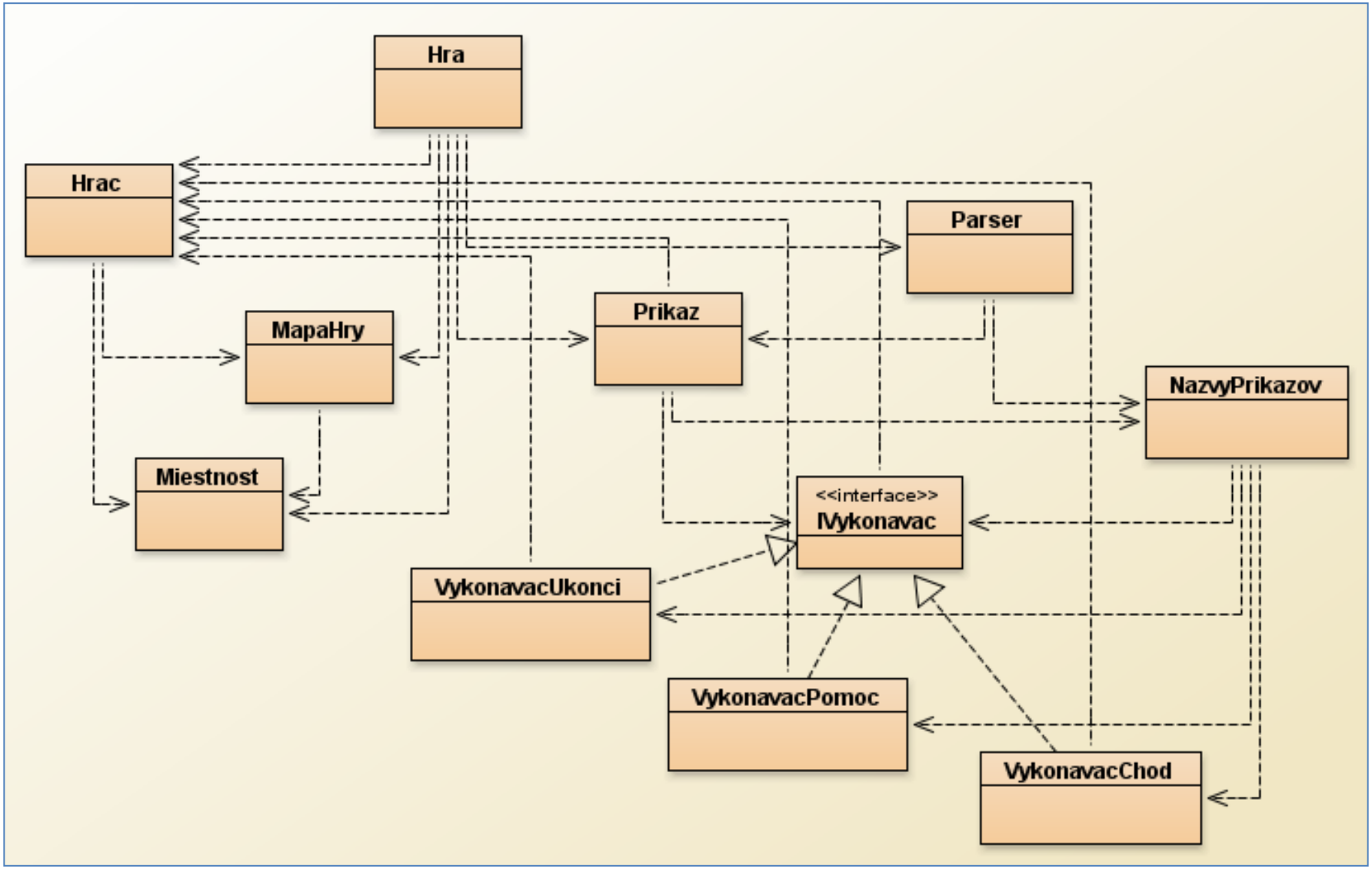
Pridanie nového príkazu – vykonávač

```
public class VykonavacUkaz implements IVykonavac
{
    public boolean vykonaj(Hrac paHrac, String paPar)
    {
        Miestnost akt = paHrac.dajAktualnuMiestnost();
        System.out.println(akt.dajUplnyPopis());

        return false;
    }
}
```

Pridanie nového príkazu – HashMap

```
aVykonavace = new HashMap<String, IVykonavac>();

aVykonavace.put("pomoc", new VykonavacPomoc());
aVykonavace.put("chod", new VykonavacChod());
aVykonavace.put("ukonci", new VykonavacUkonci());
aVykonavace.put("ukaz", new VykonavacUkaz());
```

Konečný stav



Vďaka za pozornosť