

4

# Polymorfizmus II

# Pojmy zavedené v 3. prednáške<sub>(1)</sub>

- polymorfizmus
- polymorfizmus a protokol
- druhy polymorfizmu

# Pojmy zavedené v 3. prednáške<sub>(2)</sub>

- interface
- interface - Java
- interface - UML

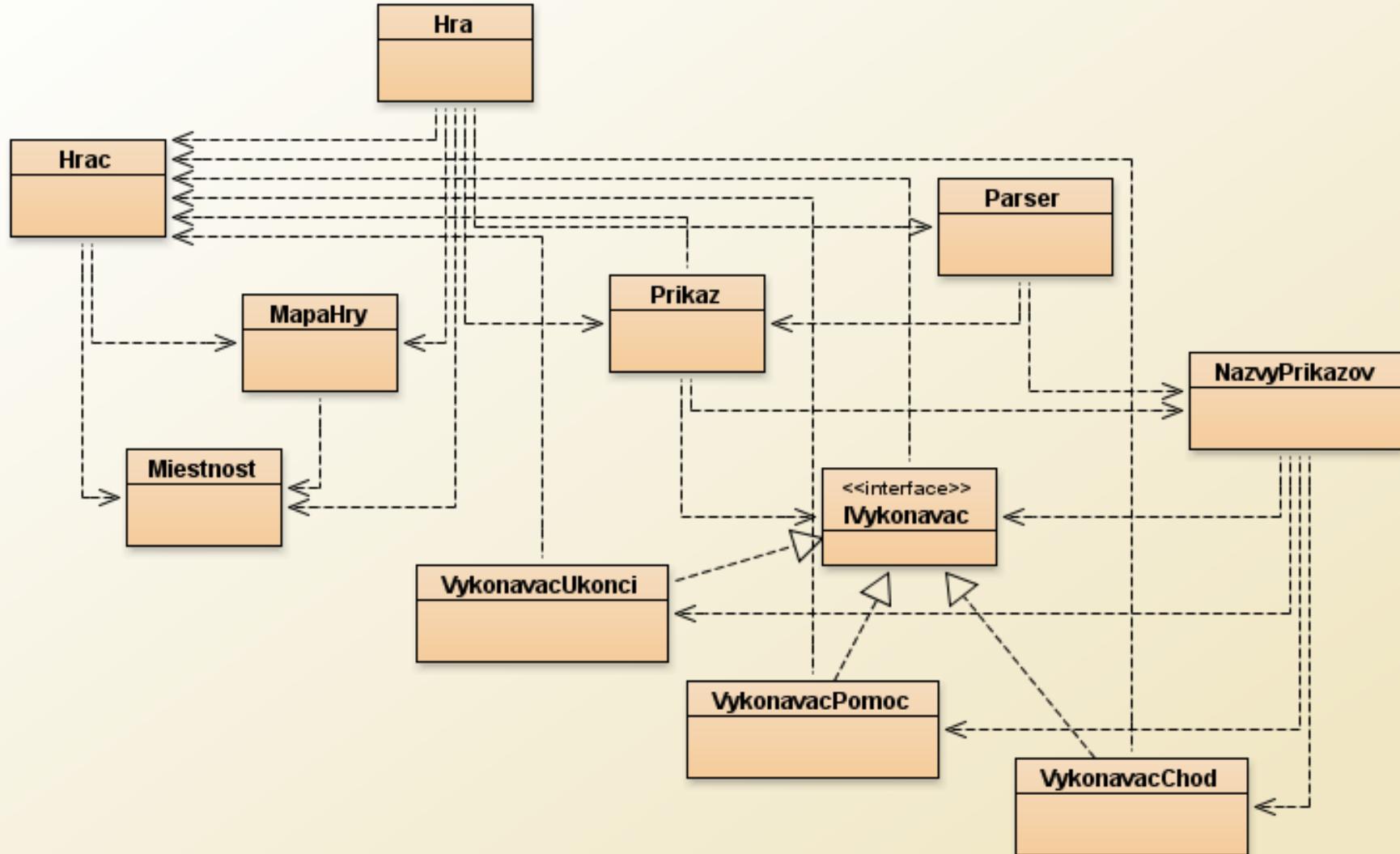
# Pojmy zavedené v 3. prednáške<sub>(3)</sub>

- typová kompatibilita - interface
- statický a dynamický typ

# Cieľ prednášky

- polymorfizmus – pokračovanie
  - pretypovanie – implicitné, explicitné
- testovanie – pokračovanie
  - testy tried pre komunikáciu s používateľom
- príklad: hra „World of FRI“
  - predmety

# Súčasný stav „World of FRI“



# „World of FRI“ – úloha 1

- pridať do hry predmety
- predmety sú v miestnostiach
- predmet môže hráč zdvihnuť a položiť
- príklad: predmet „kamen“

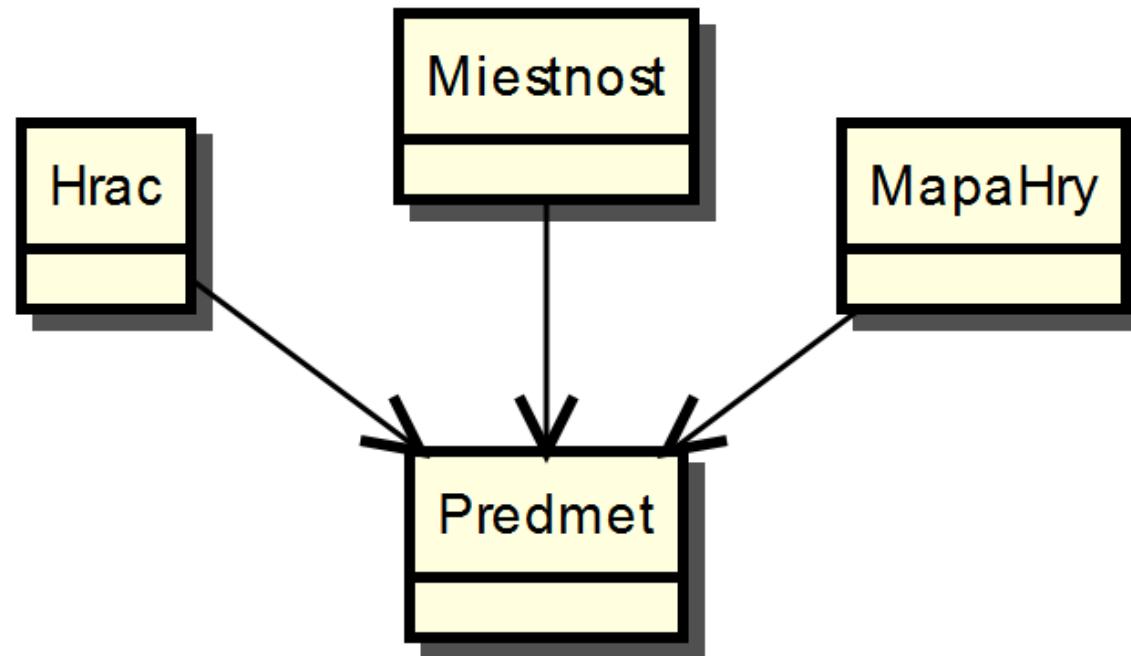
# Trieda Predmet – rozhranie

Predmet

+ new(paNazov: String): Predmet  
+ dajNazov(): String

# Úpravy závislých tried

- predmety môžu byť
  - v miestnosti – trieda Miestnost
  - u hráča – trieda Hrac
  - vytvorenie a vloženie do miestnosti – trieda MapaHry



# Úprava triedy Miestnost

- predmety v miestnosti – atribút aPredmety
- inicializácia atribútu – konštruktor
- uloženie predmetu do miestnosti – metóda
- odobratie predmetu z miestnosti – metóda

# Trieda Miestnost – kód<sub>(1)</sub>

...

```
private HashMap<String, Predmet> aPredmety;
```

...

```
public Miestnost(String paPopis)
```

```
{
```

```
    aPopisMiestnosti = paPopis;
```

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

```
    aPredmety = new HashMap<String, Predmet>();
```

```
}
```

...

# Trieda Miestnost – kód<sub>(2)</sub>

...

```
public Predmet zoberPredmet(String paNazov)
{
    return aPredmety.remove(paNazov);
}
```

```
public void polozPredmet(Predmet paPredmet)
{
    aPredmety.put(paPredmet.dajNazov(), paPredmet);
}
...
```

# Úprava triedy MapaHry

- položenie predmetu do miestnosti
  - konštruktor

# Úprava triedy MapaHry – kód

...

```
aTerasa.nastavVychod("východ", aAula);  
aTerasa.nastavVychod("juh", aLabak);  
aTerasa.nastavVychod("sever", aBufet);
```

```
aTerasa.polozPredmet(new Predmet("kamen"));
```

...

# Úprava triedy Hrac

- predmety u hráča – atribút aPredmety
- inicializácia atribútu – konštruktor
- zdvihnutie predmetu z akt. miestnosti – metóda
- položenie predmetu do akt. miestnosti – metóda

# Trieda Hrac – kód<sub>(1)</sub>

...

```
private HashMap<String, Predmet> aPredmety;
```

...

```
public Hrac(String paMeno)
```

```
{
```

```
    aMeno = paMeno;
```

```
    aAktualnaMiestnost = MapaHry.dajInstanciu()
        .dajVychodziuMiestnost();
```

```
    aPredmety = new HashMap<String, Predmet>();
```

```
}
```

...

# Trieda Hrac – kód<sub>(2)</sub>

```
public void zoberPredmet(String paNazov)
{
    Predmet predmet = aAktualnaMiestnost
        .zoberPredmet(paNazov);

    if (predmet == null) {
        System.out.println(
            "Nevidim predmet s nazvom " + paNazov);
    } else {
        aPredmety.put(predmet.dajNazov(), predmet);
    }
}
```

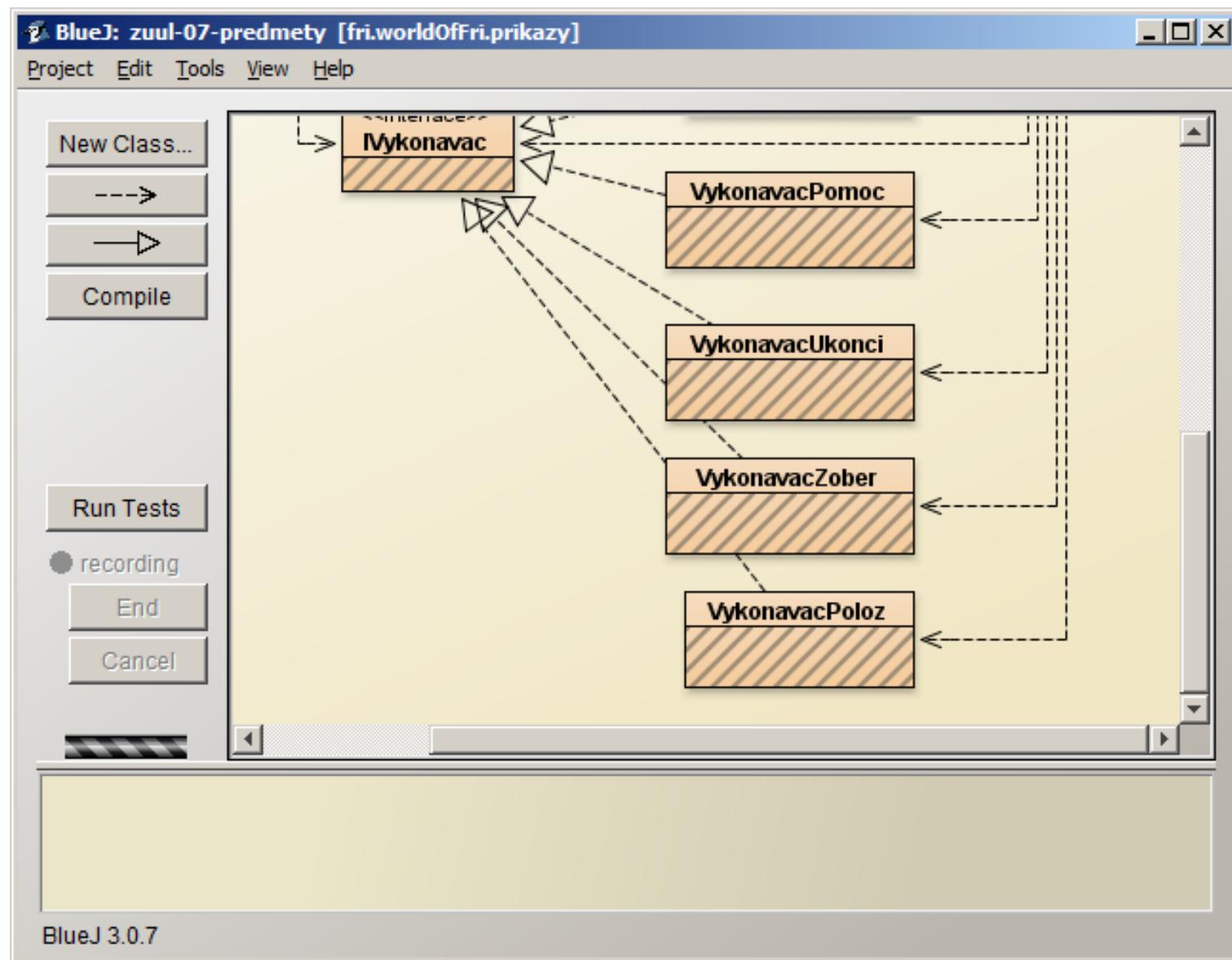
# Trieda Hrac – kód<sub>(3)</sub>

```
public void polozPredmet(String paNazov)
{
    Predmet predmet = aPredmety.remove(paNazov);
    if (predmet == null) {
        System.out.println(
            "Nemas predmet s nazvom " + paNazov);
    } else {
        aAktualnaMiestnost.polozPredmet(predmet);
    }
}
```

# Úpravy príkazov

- nové príkazy pre manipuláciu s predmetmi
  - zober predmet
  - poloz predmet
  - nové triedy VykonavacZozber, VykonavacPoloz
  - doplnenie triedy NazvyPrikazov
- rozšírenie príkazu ukaz – trieda Miestnost
  - pridať výpis predmetov v miestnosti

# Nové príkazy – UML



# „World of FRI“ – úloha 2

- pridať popis predmetov
- príkaz ukaz *parameter*
  - hráč môže pozrieť na predmet, ktorý má
- predmet „kamen“:
  - „Trochu ziari, ale inak je to uuuplne normalny kamen“

# Úpravy triedy Predmet

- nový atribút - aPopis
- nový parameter konštruktora – paPopis
- inicializácia atribúta
- nová metóda dajPopis

# Úpravy závislých tried

- trieda MapaHry doplniť vytváranie predmetov a vkladanie od miestností
- trieda Hrac doplniť prezeranie predmetov
- trieda VykonavacUkaz modifikovať metódu vykonaj

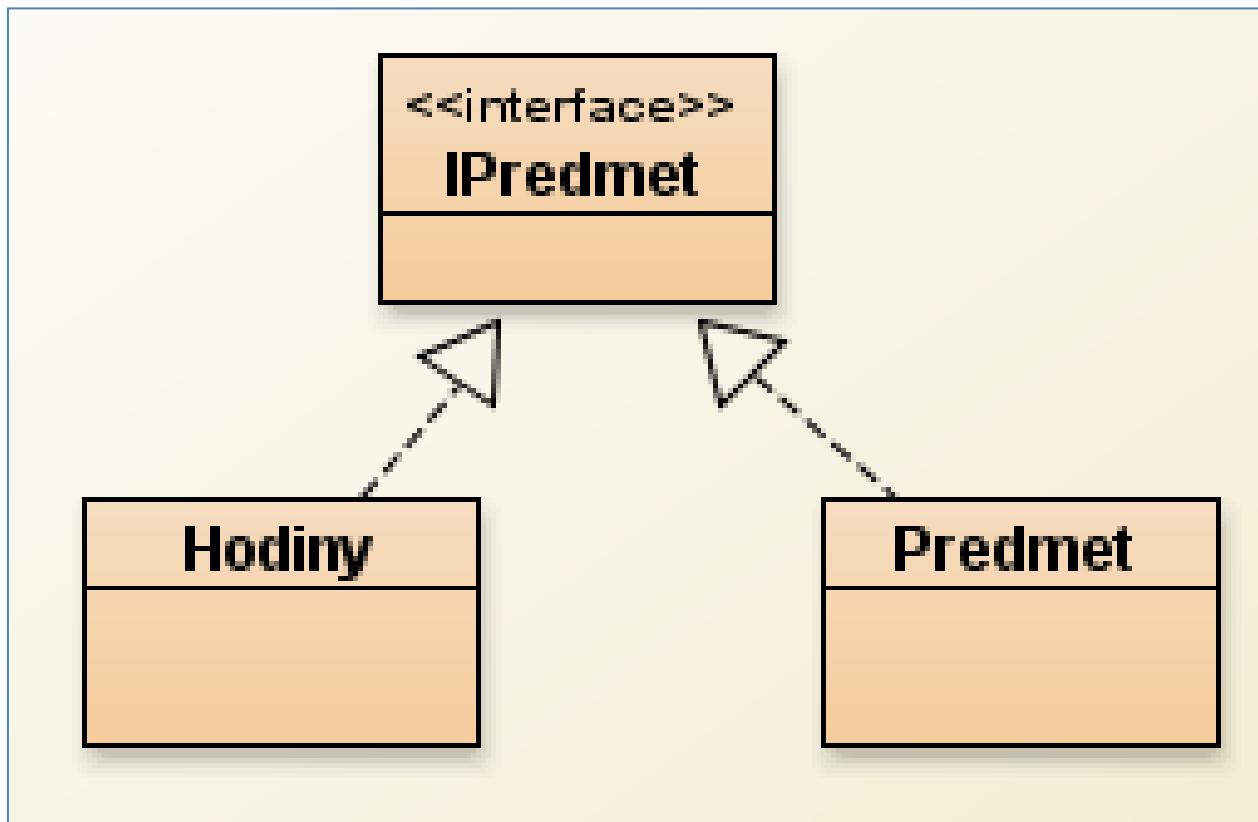
# Trieda VykonavacUkaz – kód

```
public boolean vykonaj(Hrac paHrac, String paPar)
{
    if (paPar == null) {
        System.out.println(paHrac
            .dajAktualnuMiestnost().dajUplnyPopis());
    } else {
        System.out.println(paHrac
            .dajPopisPredmetu(paPar));
    }
    return false;
}
```

# „World of FRI“ – úloha 3

- pridať nový predmet naramkoveHodiny
  - zobrazuje aktuálny čas
- 
- iné správanie ako kameň
  - treba využiť polymorfizmus
  - nový interface IPredmet

# Triedy pre predmety



# Interface IPredmet – kód

```
public interface IPredmet
```

```
{  
    String dajNazov();  
    String dajPopis();  
}
```

# Úpravy závislých tried

- nahradíť komunikáciu s inštanciami triedy Predmet – komunikácia cez rozhranie IPredmet
  - trieda Miestnost a trieda Hrac – zmena typu Predmet na typ IPredmet – statický typ
  - trieda MapaHry – pridanie predmetu hodiny do mapy
- nová trieda Hodiny
- trieda Predmet – implementuje interface IPredmet

# Trieda Miestnost – kód<sub>(1)</sub>

...

```
private HashMap<String, IPredmet> aPredmety;
```

...

```
public Miestnost(String paPopis)
```

```
{
```

```
    aPopisMiestnosti = paPopis;
```

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

```
    aPredmety = new HashMap<String, IPredmet>();
```

```
}
```

...

# Trieda Miestnost – kód<sub>(2)</sub>

...

```
public IPredmet zoberPredmet(String paNazov)
{
    return aPredmety.remove(paNazov);
}

...
public void polozPredmet(IPredmet paPredmet)
{
    aPredmety.put(paPredmet.dajNazov(), paPredmet);
}

...
```

# Trieda MapaHry – kód

...

```
aKancelaria.nastavVychod("zапад", aLabak);
```

```
aKancelaria.polozPredmet(
```

```
    new Hodiny("naramkoveHodiny"));
```

...

# Nová trieda Hodiny – kód<sub>(1)</sub>

```
import java.util.Date;  
import java.text.DateFormat;
```

```
public class Hodiny implements IPredmet
```

```
{
```

```
    private String aNazov;
```

```
...
```

```
}
```

# Nová trieda Hodiny – kód<sub>(2)</sub>

```
public Hodiny(String paNazov)
```

```
{  
    aNazov = paNazov;
```

```
}
```

```
public String dajNazov()
```

```
{  
    return aNazov;
```

```
}
```

```
...
```

# Nová trieda Hodiny – kód<sub>(3)</sub>

```
public String dajPopis()
```

```
{
```

```
    Date cas = new Date();
```

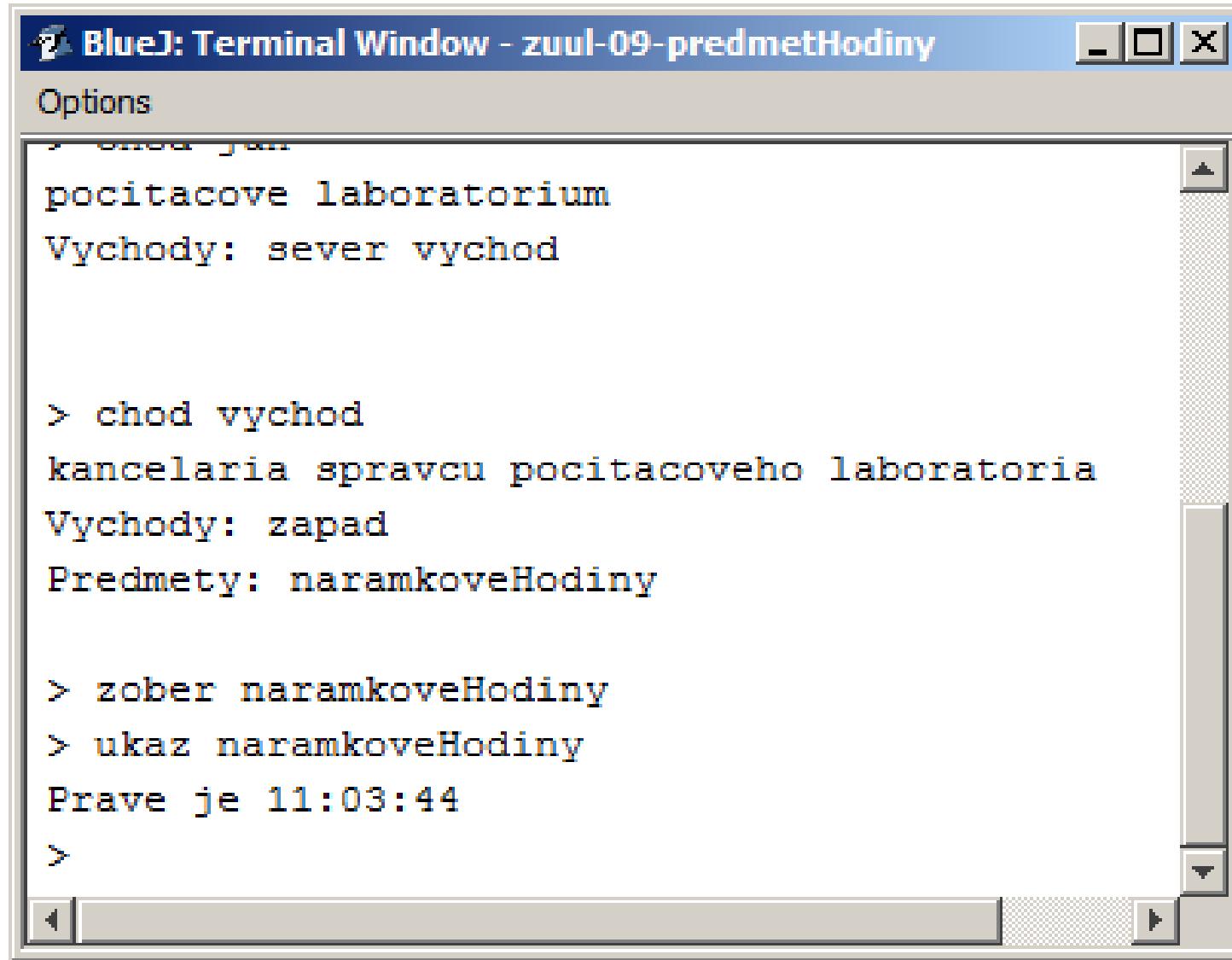
```
    DateFormat formatCasu =
```

```
        DateFormat.getTimeInstance();
```

```
    return "Prave je " + formatCasu.format(cas);
```

```
}
```

# Priebeh hry – príkazy zober, ukaz



The screenshot shows a terminal window titled "BlueJ: Terminal Window - zuul-09-predmetHodiny". The window displays a text-based adventure game interface. The user has typed the command "pocitacove laboratorium" which results in the output "Vychody: sever vychod". Subsequent commands "chod vychod" and "kancelaria spravcu pocitacoveho laboratoria" lead to "Vychody: zapad" and "Predmety: naramkoveHodiny". Finally, the command "zober naramkoveHodiny" followed by "ukaz naramkoveHodiny" shows the current time as "Prave je 11:03:44". The terminal window has standard scroll bars on the right side.

```
pocitacove laboratorium
Vychody: sever vychod

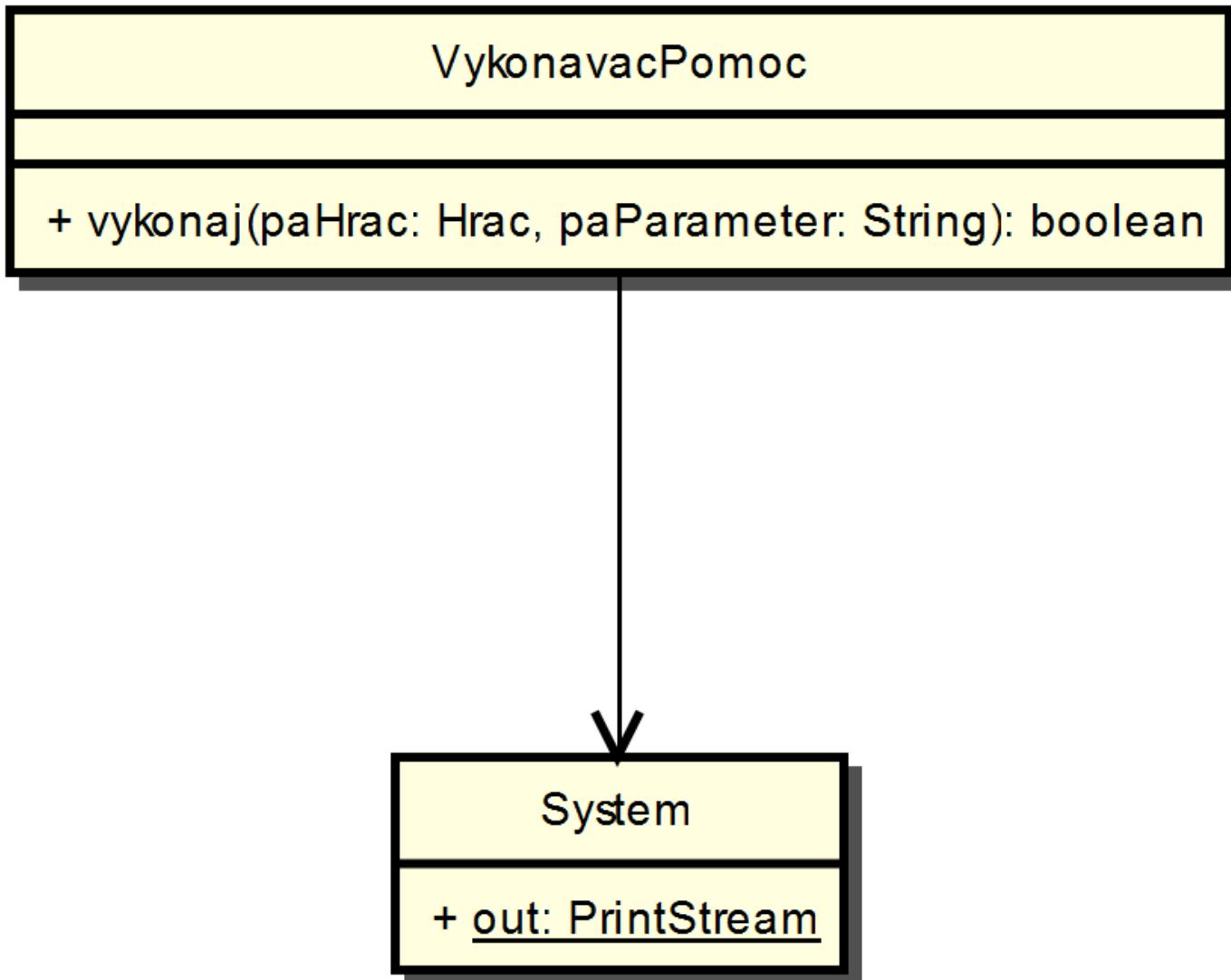
> chod vychod
kancelaria spravcu pocitacoveho laboratoria
Vychody: zapad
Predmety: naramkoveHodiny

> zober naramkoveHodiny
> ukaz naramkoveHodiny
Prave je 11:03:44
>
```

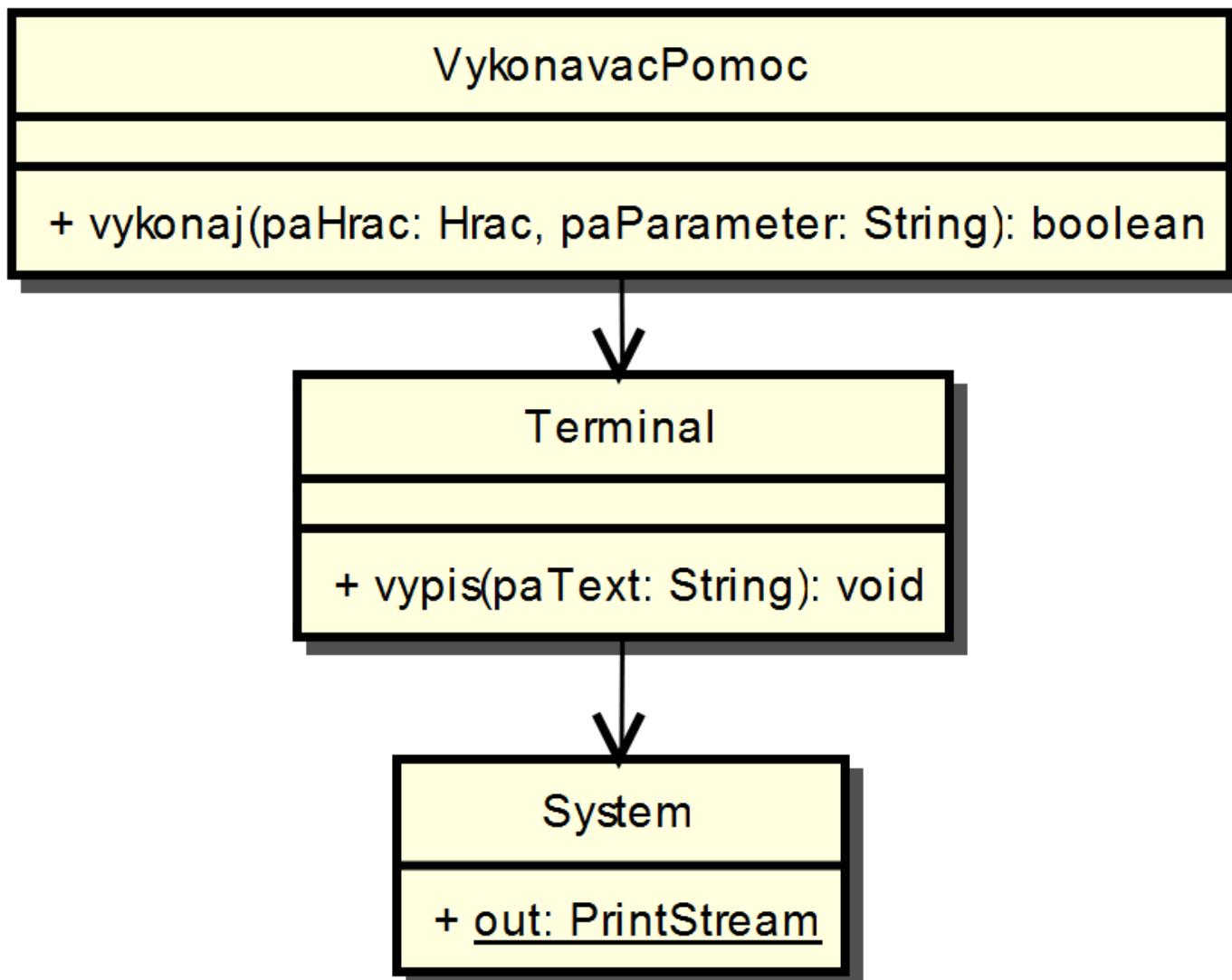
# Testovanie

- testovanie tried pre priamu komunikáciu s hráčom (používateľom)
- výsledok činnosti metódy je vypísaný do terminálového okna
- `System.out.print(...)`
- `System.out.println(...)`

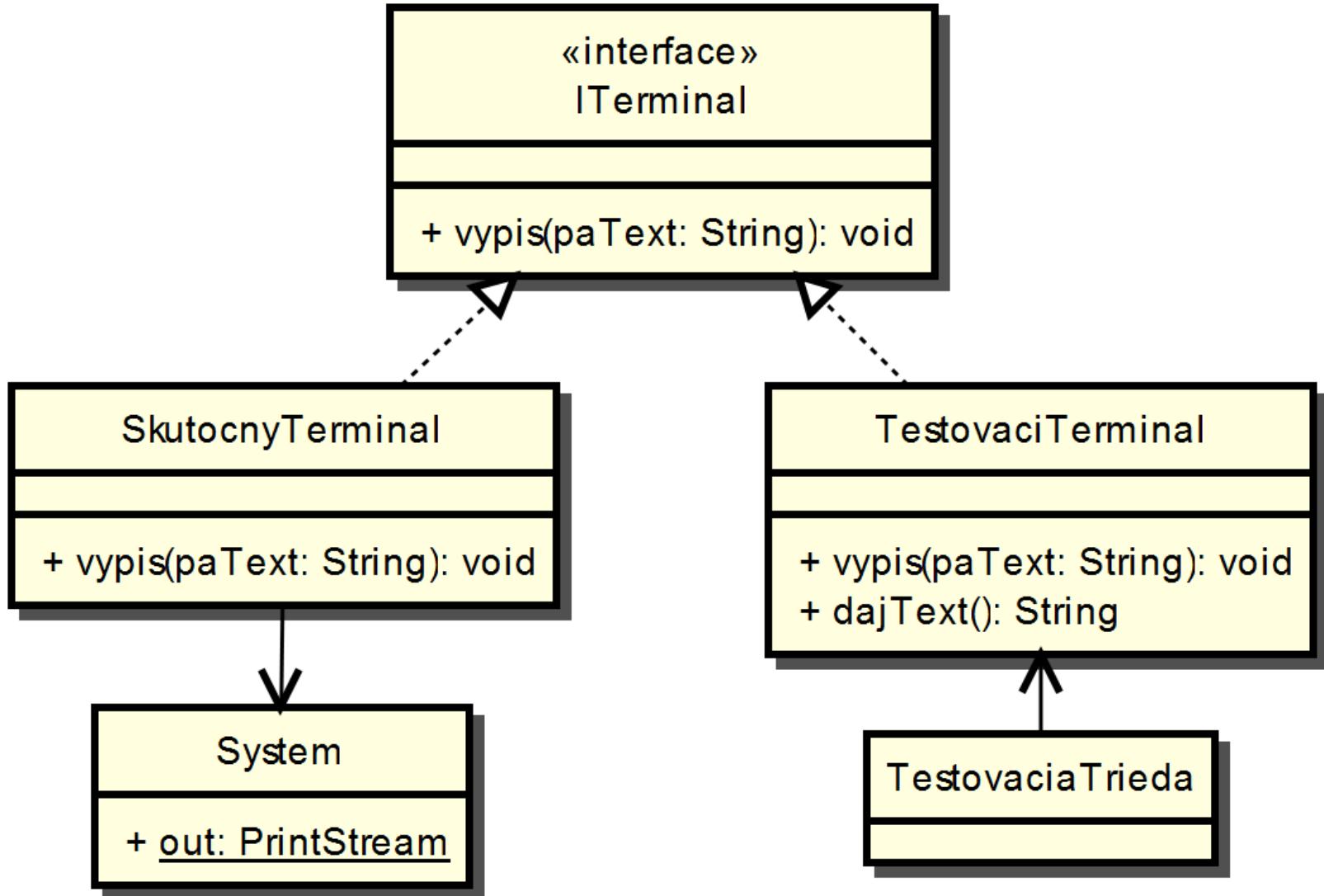
# Priama komunikácia



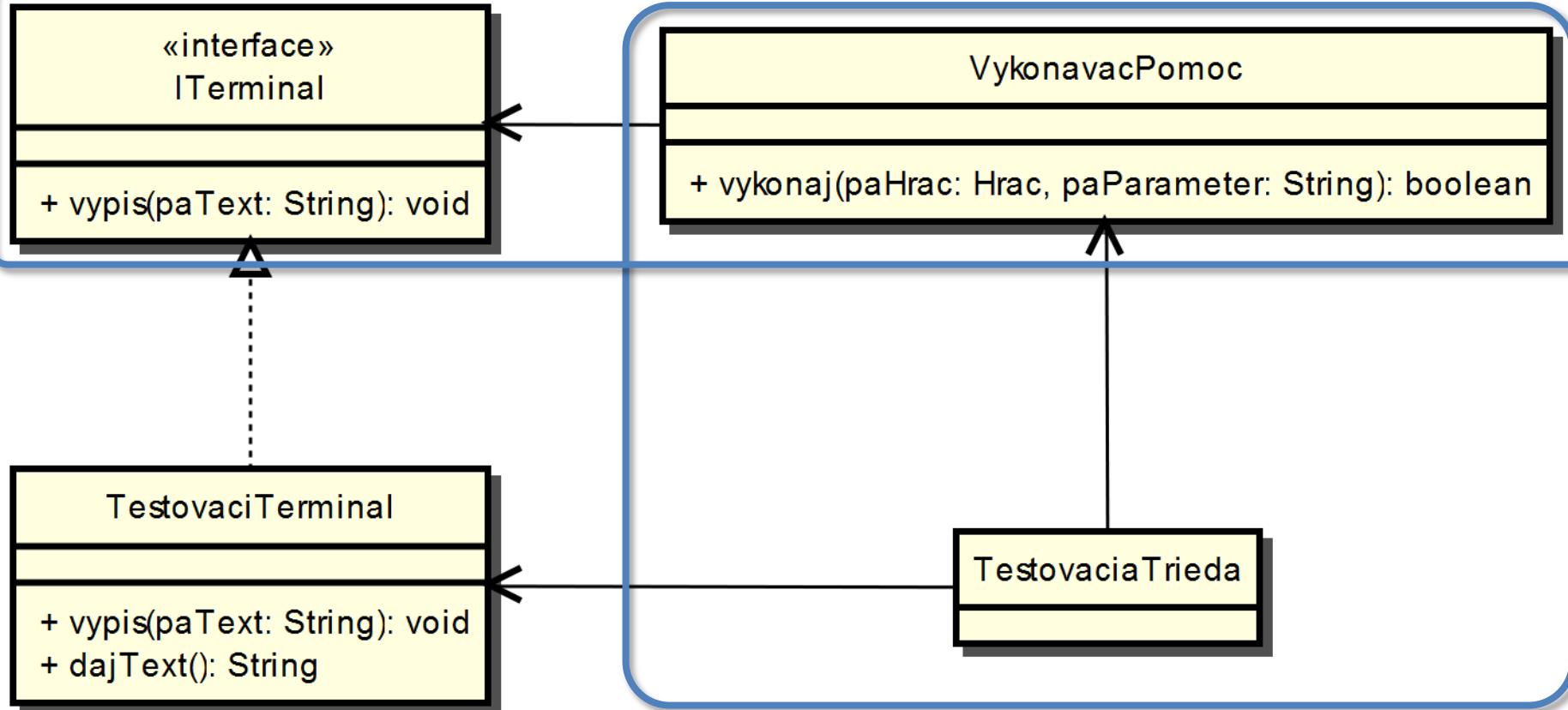
# Terminal – prostredník



# Interface ITerminal



# Testovacia trieda



# Interface ITerminal – kód

```
public interface ITerminal  
{  
    void vypis(String pariadok);  
}
```

# Trieda SkutocnyTerminal

```
public class SkutocnyTerminal implements ITerminal
{
    public void vypis(String paRiadok)
    {
        System.out.println(paRiadok);
    }
}
```

# Trieda TestovaciTerminal<sub>(1)</sub>

```
public class TestovaciTerminal implements ITerminal
{
    private String aVystup;

    public TestovaciTerminal()
    {
        aVystup = "";
    }
    ...
}
```

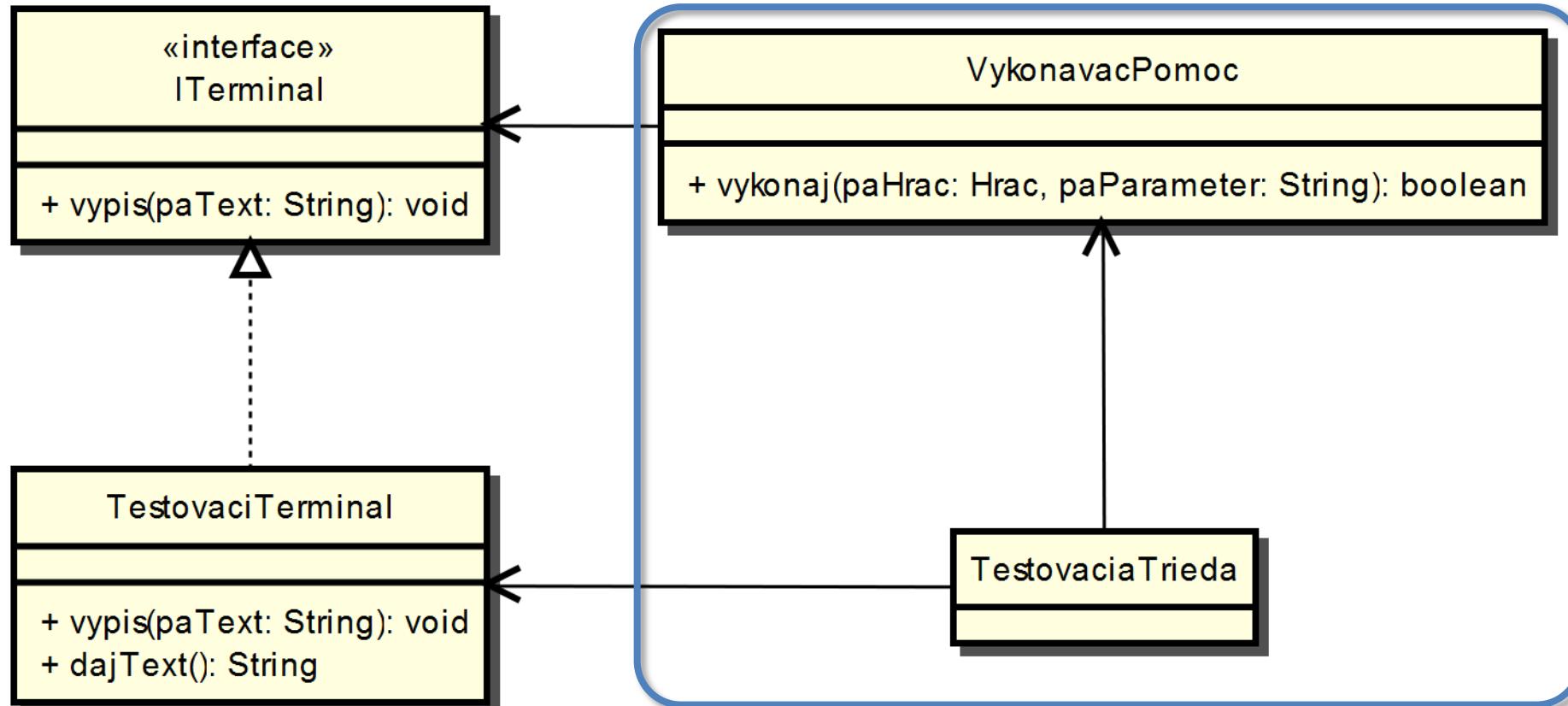
# Trieda TestovaciTerminal<sub>(2)</sub>

...

```
public void vypis(String paRiadok)
{
    aVystup += paRiadok + "\n";
}
```

```
public String dajVypis()
{
    return aVystup;
}
```

# Testovaná a testovacia trieda



# Trieda VykonavacPomoc<sub>(1)</sub>

```
public class VykonavacPomoc
{
    implements IVykonavac

    private ITerminal aTerminal;

    public VykonavacPomoc(ITerminal paTerminal)
    {
        aTerminal = paTerminal;
    }

    ...
}
```

# Trieda VykonavacPomoc<sub>(2)</sub>

```
public boolean vykonaj(  
    Hrac paHrac, String paParameter)  
{  
    aTerminal.vypis("Zabloudil si. Si sam. Tulas sa po fakulte.");  
    aTerminal.vypis("");  
    aTerminal.vypis("Mozes pouzit tieto prikazy:");  
    aTerminal.vypis("  chod ukonci pomoc");  
    return false;  
}
```

# VykonavaceTest – Prípravky

```
private TestovaciTerminal terminal;  
private Hrac hrac;  
private VykonavacPomoc vykonavac;
```

```
@Before  
public void setUp()  
{  
    terminal = new TestovaciTerminal();  
    hrac = new Hrac("Test");  
    vykonavac = new VykonavacPomoc(terminal);  
}
```

# VykonavacTest – Testovacia metóda

```
@Test
```

```
public void testVyknavacPomoc()
```

```
{
```

```
String ocakavany =
```

```
    "Zabloudil si. Si sam. Tulas sa po fakulte.\n" +
```

```
    "Mozes pouzit tieto prikazy:\n" +
```

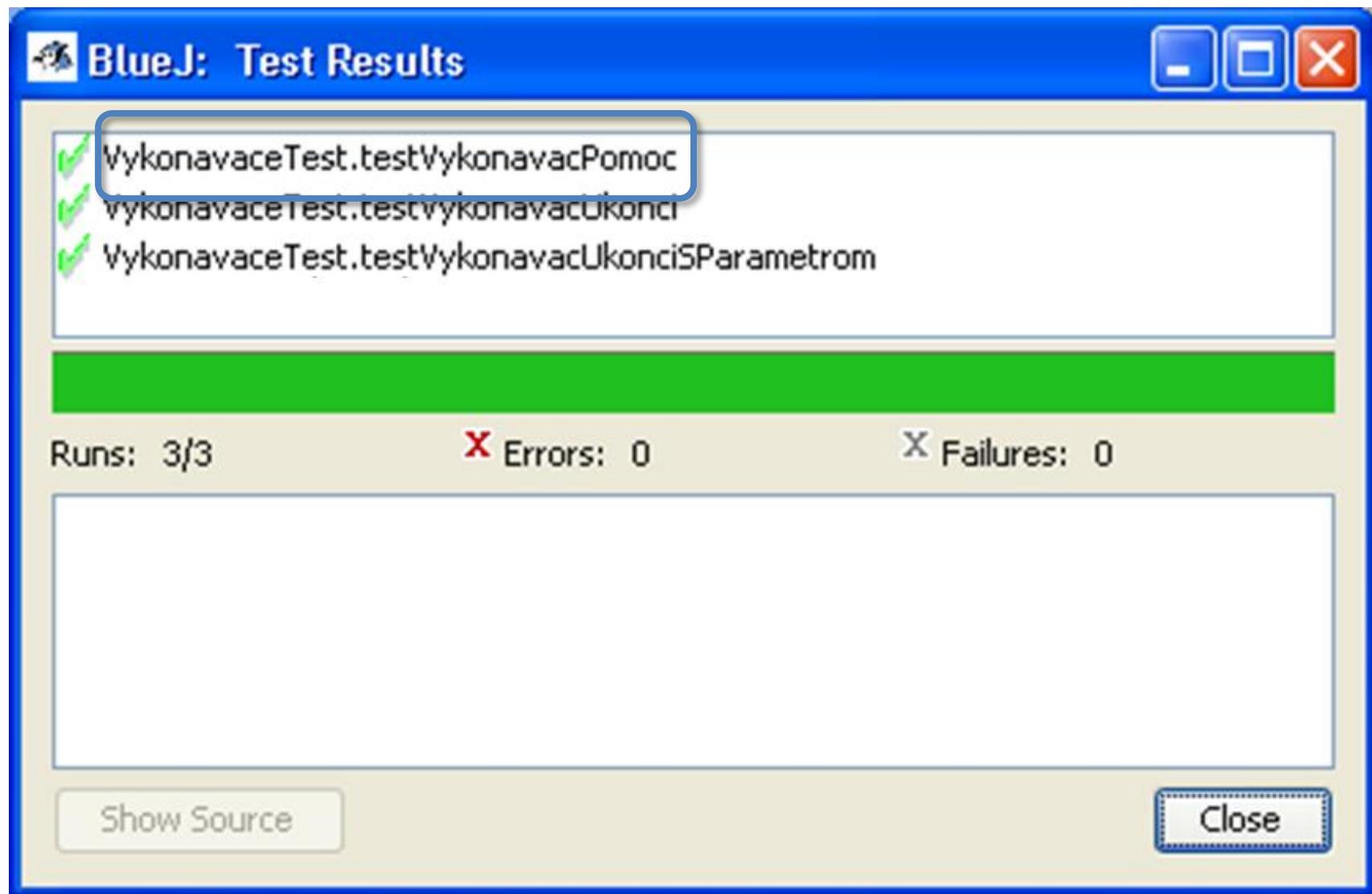
```
    " chod ukonci pomoc\n";
```

```
assertFalse(vykonavac.vykonaj(hrac, null));
```

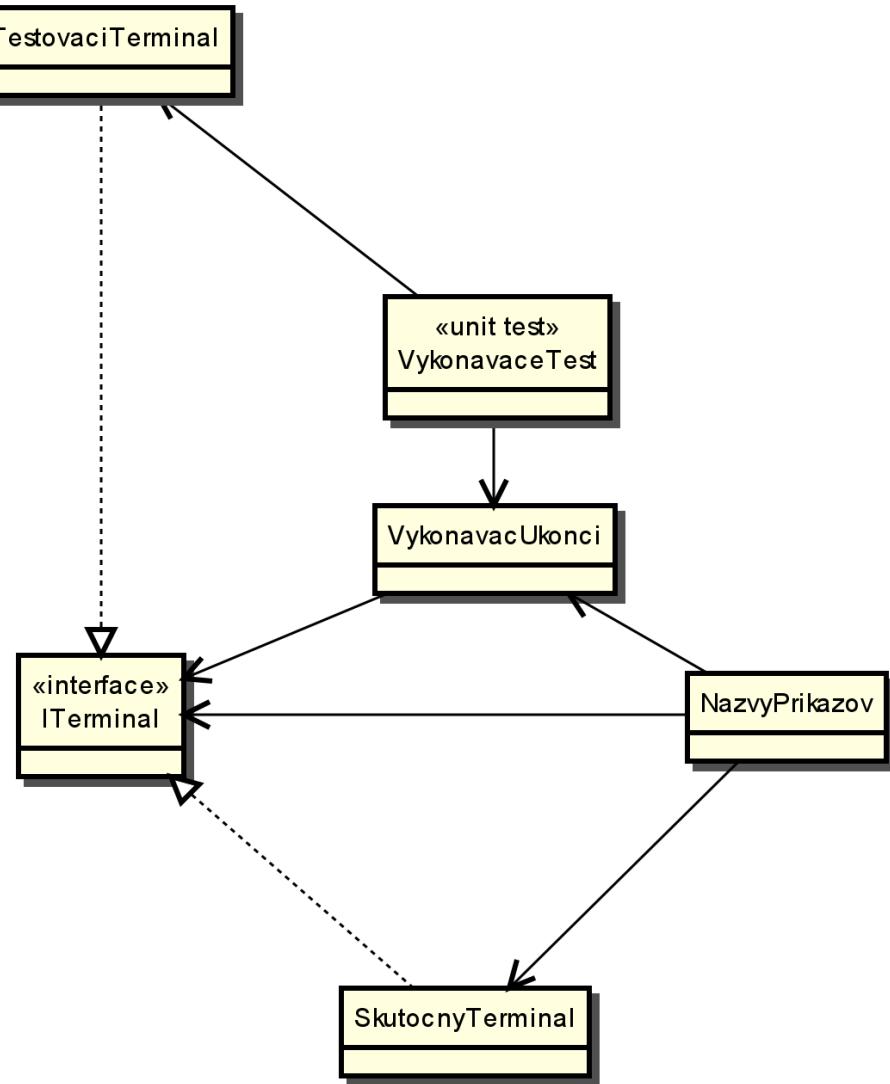
```
assertEquals(ocakavany, terminal.dajVypis());
```

```
}
```

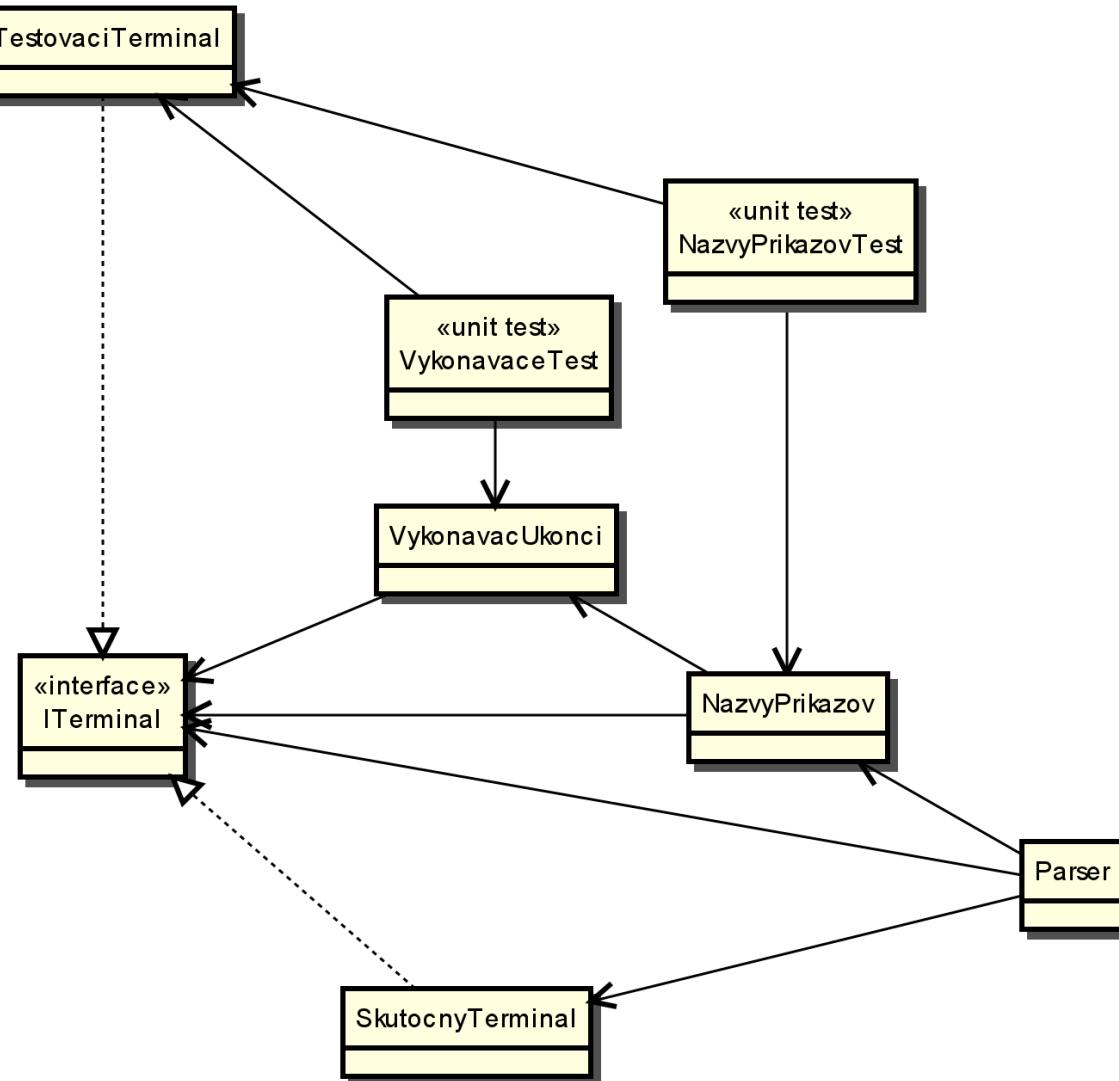
# Výsledok testov



# Testovanie – vykonávače<sub>(1)</sub>



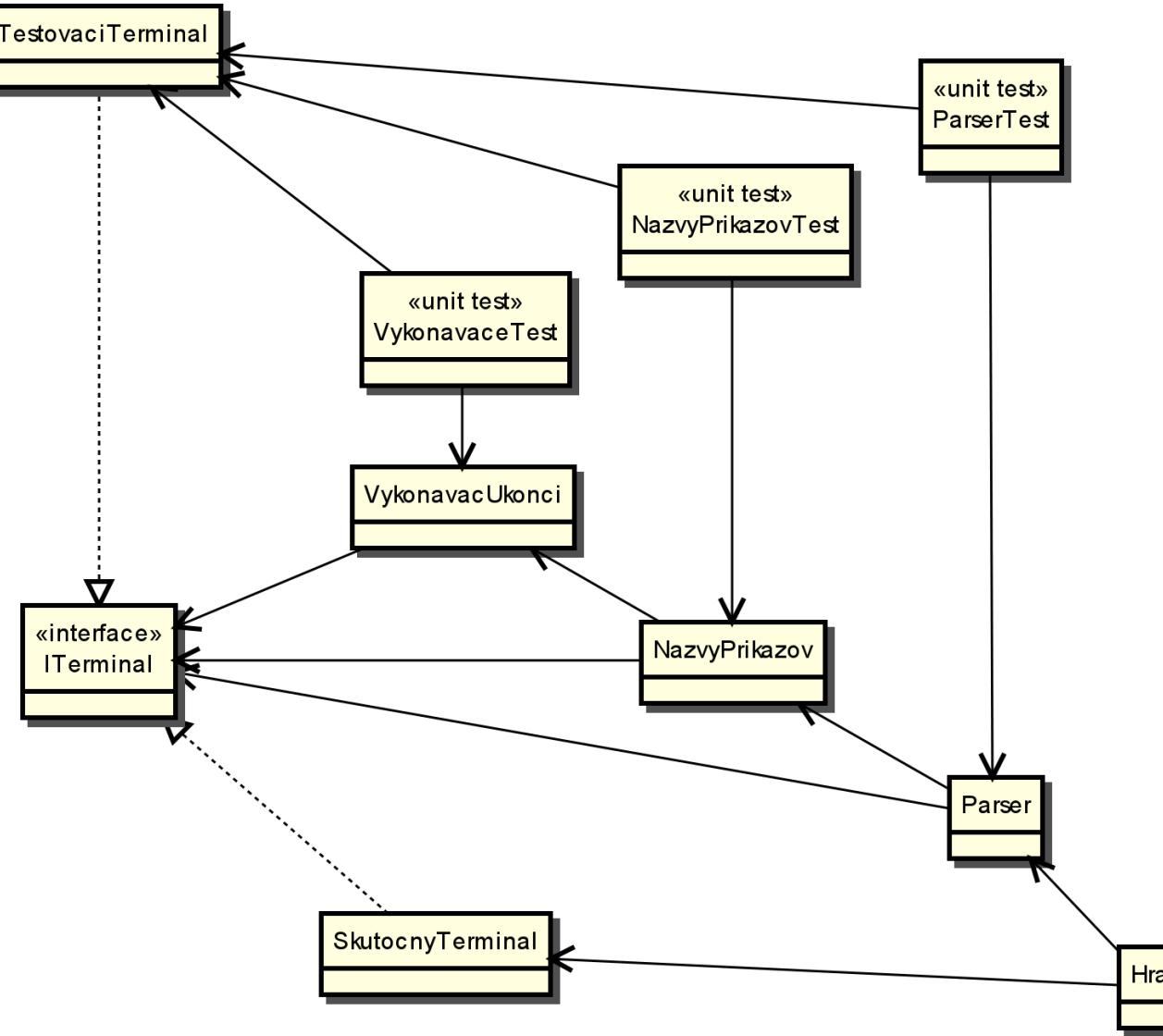
# Testovanie – vykonávače<sub>(2)</sub>



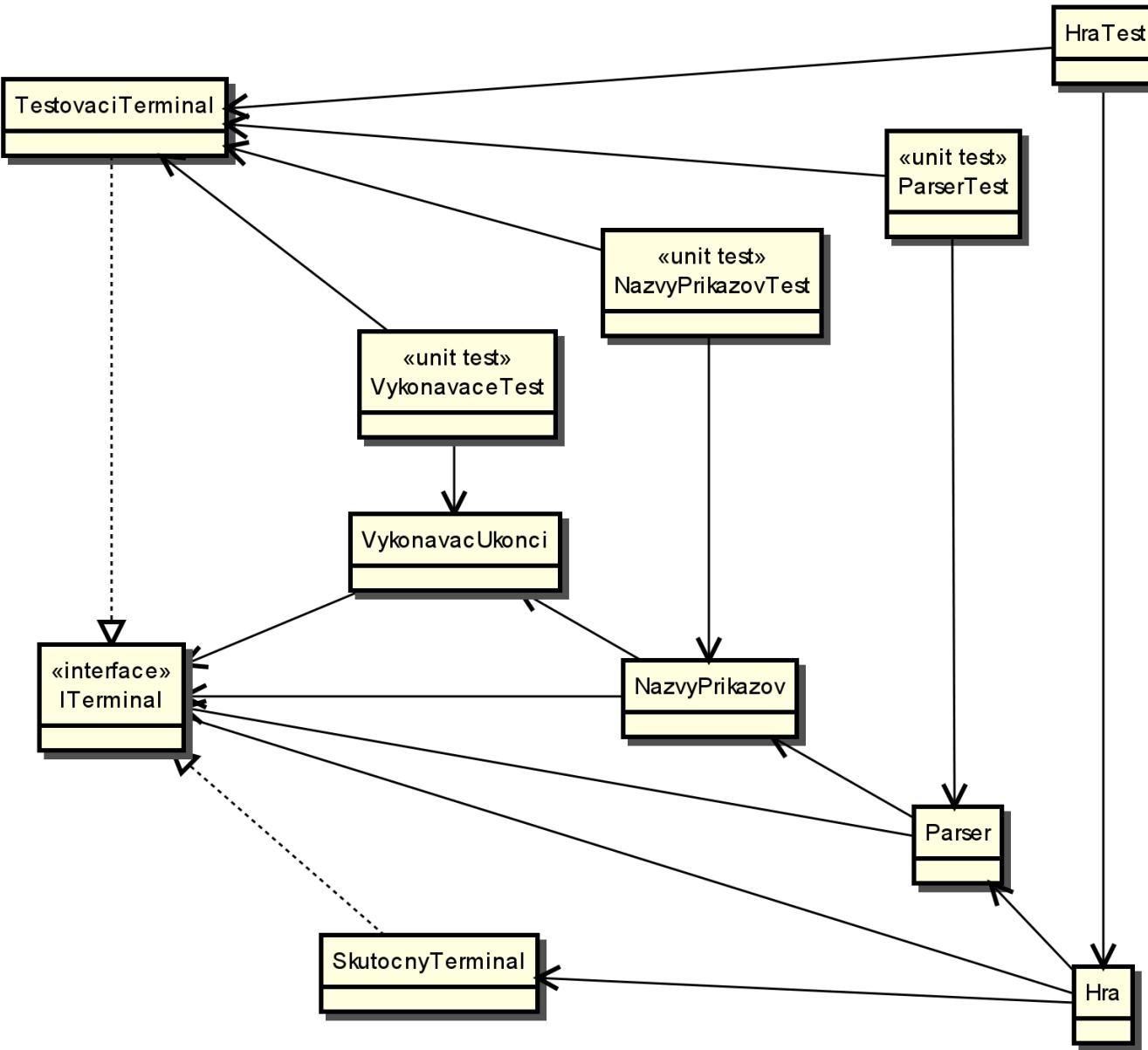
# Testovanie – Parser

- čítanie z klávesnice a testovanie
- rozšírenie terminálu o čítanie klávesnice
- skutočný terminál preberá vstup od System.in
- testovací terminál dostáva vstup ako parameter konštruktora
- detailly realizácie – projekt na vzdelávaní

# Testovanie – vykonávače<sub>(3)</sub>



# Testovanie – vykonávače<sub>(4)</sub>



# Dependency injection

- slovensky: vkladanie závislostí
- návrhový vzor
- znižovanie implementačnej závislosti pomocou polymorfizmu
  - nahradenie závislosti pomocou interface
- zlepšenie možnosti testovania aplikácie

# „World of FRI“ – úloha 4

- pridať predmet walkman
  - prehráva hudbu
  - dá sa zapnúť
  - nefunguje bez bateriek
- pridať predmet baterky
  - dajú sa vložiť do walkmanu

# Vyžadované zmeny

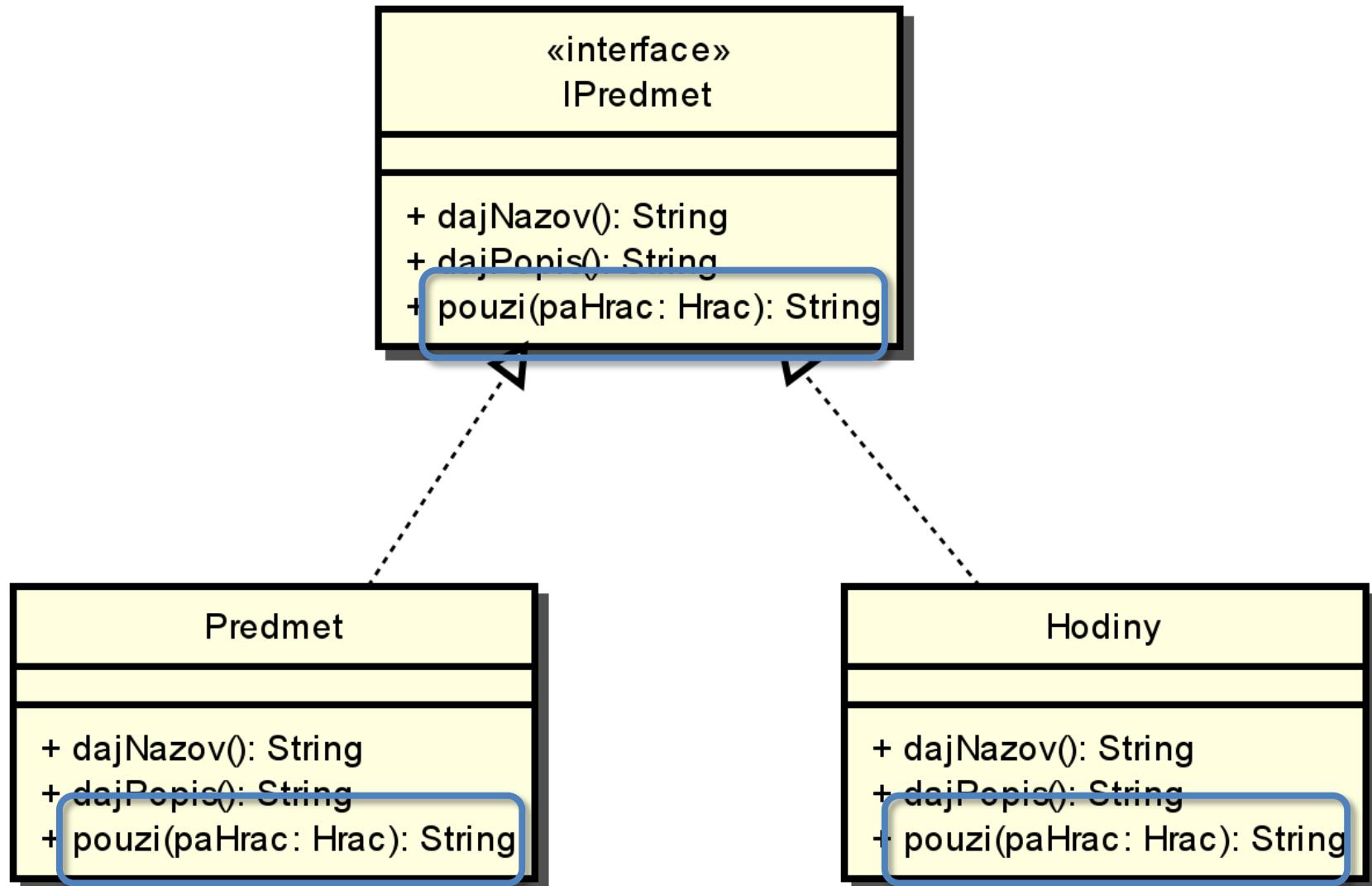
- každý predmet sa musí dať použiť
  - nový príkaz pouzi
  - nová trieda VykonavacPouzi
  - nová položka do zoznamu v triede NazvyPrikazov
- nová správa pouzi do interface IPredmet
  - hráč musí vedieť použiť držaný predmet
  - nová metóda pouziPredmet(String paNazov) do triedy Hrac

# Nová správa – použi

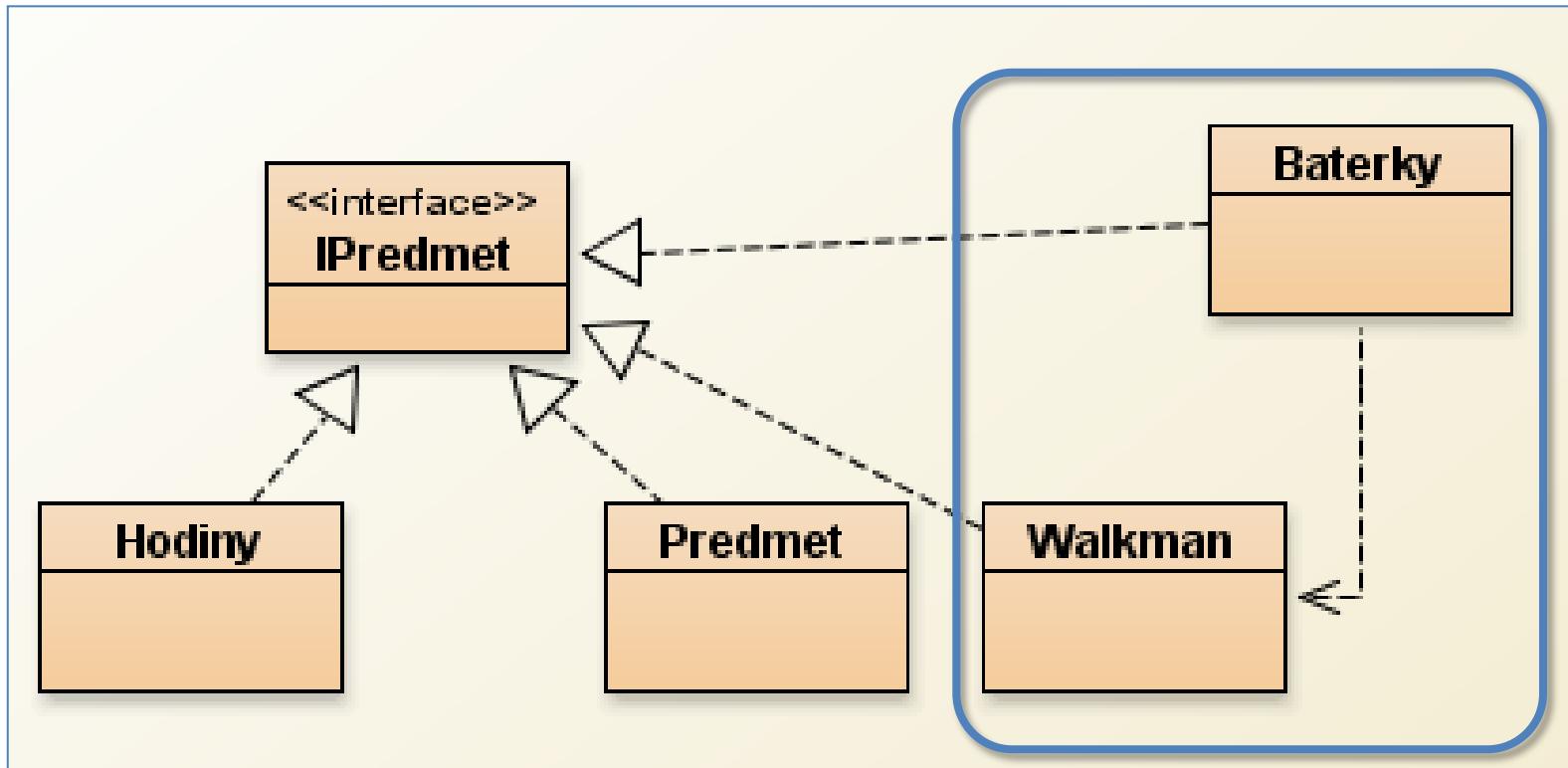
## String použi(Hrac paHrac)

- parameter paHrac – hráč, ktorý predmet používa
- návratová hodnota – odpoveď hráčovi vo forme reťazca, zobrazí sa v termináli

# Nová správa – použi



# Nové predmety



# Nový predmet – walkman

- `dajNazov` – vráti názov (walkman)
- `dajPopis` – informuje o možnosti objekt spustiť
- `pouzi` – ak sú vložené batérie, pustí hudbu – vypíše text na terminál

# Nový predmet – baterky

- `dajNazov` – vráti názov (baterky)
- `dajPopis` – nejaký vtipný popis bateriek
- `pouzi` – zistí, či hráč drží walkman, ak áno, vloží doň baterky

# Trieda Walkman<sub>(1)</sub>

```
public class Walkman implements IPredmet
{
    private String aNazov;
    private boolean aMaBaterky;

    public Walkman(String paNazov)
    {
        aNazov = paNazov;
        aMaBaterky = false;
    }
    ...
}
```

# Trieda Walkman<sub>(2)</sub>

```
public String dajPopis()
{
    return "Walkman. Ma cudlik na zapnutie.";
}
```

```
public String dajNazov()
{
    return aNazov;
}
```

# Trieda Walkman<sub>(3)</sub>

```
public String pouzi(Hrac paHrac)
{
    if (aMaBaterky) {
        return "La.. laaa~ walkmen vyhrovavaaaa~";
    } else {
        return "Nema baterky";
    }
}
```

# Trieda Walkman<sub>(4)</sub>

```
public void vlozBaterky()  
{  
    aMaBaterky = true;  
}
```

# Trieda Baterky<sub>(1)</sub>

```
public class Baterky implements IPredmet
{
    private String aNazov;

    public Baterky(String paNazov)
    {
        aNazov = paNazov;
    }
    ...
}
```

# Trieda Baterky<sub>(2)</sub>

```
public String dajNazov()
```

```
{  
    return aNazov;  
}
```

```
public String dajPopis()
```

```
{  
    return "Tie baterky so zajacom, co hra fotbal.  
            Nebudem robit reklamu...";  
}
```

# Trieda Baterky<sub>(3)</sub>

```
public String pouzi(Hrac paHrac)
{
    Walkman walkman =
        paHrac.dajPredmet("walkman");

    if (walkman == null)
        return "Nemas ich ako pouzit";

    paHrac.vyberPredmet(aNazov);
    walkman.vlozBaterky();
    return "Vlozil si baterky do walkmana.";
}
```

# Chyba pri preklade

The screenshot shows a Java code editor window titled "Baterky". The menu bar includes "Class", "Edit", "Tools", and "Options". The toolbar contains buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", and "Close". A dropdown menu labeled "Source Code" is open. The code editor displays the following Java code:

```
24 public String pouzi(Hrac paHrac)
25 {
26     Walkman walkman = paHrac.dajPredmet("walkman");
27
28     if (walkman == null)
29         return "Nemas ich ako pouzit";
30
31     walkman.vlozBaterky();
32     paHrac.zahodPredmet(aNazov);
33
34     return "Vlozil si baterky do walkmana.";
35 }
36 }
```

A red box highlights the line of code: `Walkman walkman = paHrac.dajPredmet("walkman");`. Below the code editor, a message box displays the error: **incompatible types - found fri.worldOffri.predmety.IPredmet but expected fri.worldOffri.predmety.Walkman**. The message box has a blue border and contains a question mark icon and a "saved" button.

# Trieda Baterky – inak

```
public String pouzi(Hrac paHrac)
{
    IPredmet walkman =
        paHrac.dajPredmet("walkman");

    if (walkman == null)
        return "Nemas ich ako pouzit";

    paHrac.vyberPredmet(aNazov);
    walkman.vlozBaterky();
    return "Vlozil si baterky do walkmana.";
}
```

# Trieda Baterky – správne

```
public String pouzi(Hrac paHrac)
{
    IPredmet pred = paHrac.dajPredmet("walkman");
    Walkman walkman = (Walkman) pred;

    if (walkman == null)
        return "Nemas ich ako pouzit";

    paHrac.vyberPredmet(aNazov);
    walkman.vlozBaterky();
    return "Vlozil si baterky do walkmana.";
}
```

# Pretypovanie

- zmena statického typu
- zmena množiny správ, ktoré je možné poslať
- implicitné
- explicitné

# Implicitné pretypovanie

- automatické pretypovanie
- iba obmedzenie množiny správ
- vykonáva a kontroluje prekladač pri preklade
- statický typ musí byť typovo kompatibilný s cieľovým typom
- literál null sa dá implicitne pretypovať na ľubovoľný objektový typ

IPredmet predmet = **new** Walkman();

# Explicitné pretypovanie

- pretypovanie „na požiadanie“
- rozšírenie, alebo výmena množiny správ
- prekladač robí iba čiastočnú kontrolu
- vykonáva a kontroluje JVM za behu programu
- dynamický typ musí byť typovo kompatibilný s cieľovým typom
- hodnota null sa dá explicitne pretypovať na ľubovoľný objektový typ

# Explicitné pretypovanie – príklad

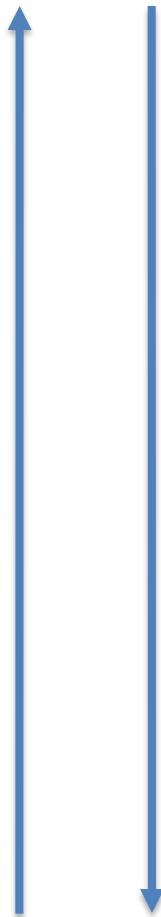
Walkman predmet =

```
(Walkman)paHrac.dajPredmet("walkman");
```

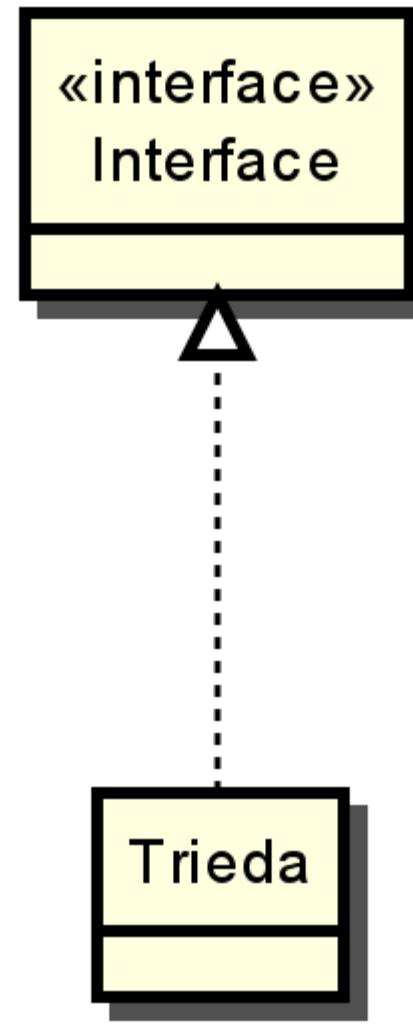
- nový typ do zátvorky pred výraz, ktorý má byť pretypovaný

# Pretypovanie<sub>(1)</sub>

Implicitné  
I



Explicitné  
E



# Pretypovanie<sub>(2)</sub>

IPredmet predmet;

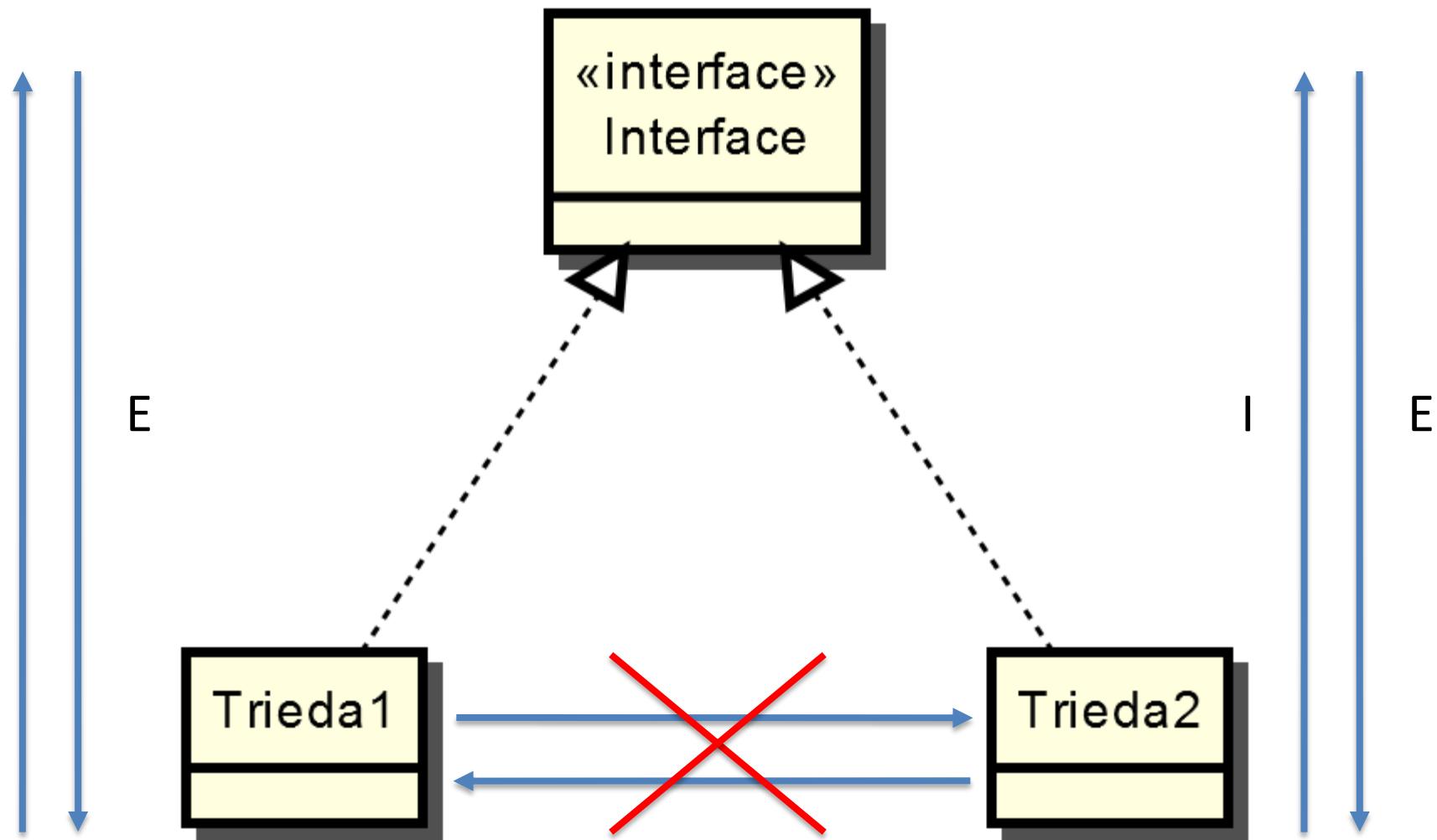
Walkmen walkmen = **new** Walkmen();

predmet = walkmen; // OK

walkmen = predmet; // Chyba

walkmen = (Walkmen) predmet; // OK

# Pretypovanie<sub>(3)</sub>



# Pretypovanie<sub>(4)</sub>

I Vozidlo vozidlo;

Auto auto = new Auto();

Bicykel kolo = new Bicykel();

vozidlo = auto; // OK

auto = kolo; // Chyba

auto = (Auto) kolo; // Chyba

auto = (Auto) vozidlo; // OK

kolo = (Bicykel) vozidlo; // Chyba za behu

# Vďaka za pozornosť