Prednáška 9: Polymorfizmus, Rozhrania, deklarácia a využitie. Implementácia viacerých rozhraní súčasne. Abstraktné triedy

Ján Lang

kanc. 4.34, jan.lang@stuba.sk, http://www2.fiit.stuba.sk/~lang/zoop/

Ústav informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave
28. novembra 2023

Polymorfizmus

- Príklad uplatnenia polymorfizmu na farme
- Dve triedy Zviera, Potrava
- Potrava disponuje metódou **`void`** `nakrm(Zviera z)`

```
public class Potrava {
    void nakrm(Zviera z) {

    }
}

public class Zviera {

}
```

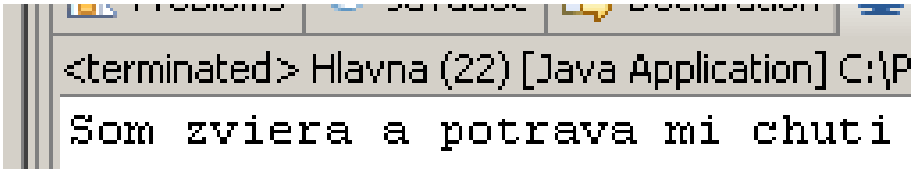
Polymorfizmus

- Zviera už vie zjesť Potravu

```
public class Zviera {
    public void zjedz(Potrava p) {
        System.out.println("Som zviera a potrava mi chuti");
    }
}
```

- Vytvorenie inštancie zvieraťa

```
public static void main(String[] args) {
    new Zviera().zjedz(new Potrava());
}
```

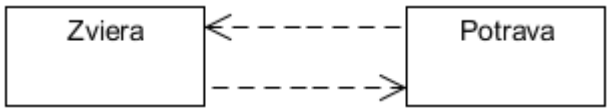


Polymorfizmus

- Symetria vzájomnej závislosti

```
public static void main(String[] args) {
    new Zviera().zjedz(new Potrava());
    new Potrava().nakrm(new Zviera());
}
```

```
<terminated> Hlavna (22) [Java Application] C:\F
Som zviera a potrava mi chuti
Zviera z je nakrmene
```



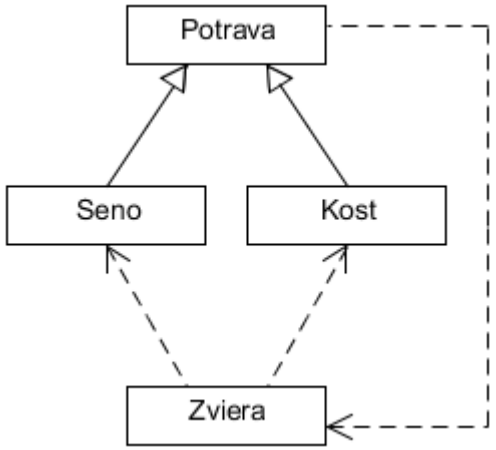
Polymorfizmus

- Špeciálne podtypy Potravy (Seno a Kost)

```
public class Seno extends Potrava {
    public void nakrm(Zviera z) {
        z.zjedz(this);
    }
}

public class Kost extends Potrava {
    public void nakrm(Zviera z) {
        z.zjedz(this);
    }
}
```

- Inštancia Zvieraťa vie zjesť inštanciu Potravy. Kost vie nakŕmiť „ľubovoľné“ Zviera. Podobne seno vie nakŕmiť „ľubovoľné“ Zviera.

Polymorfizmus



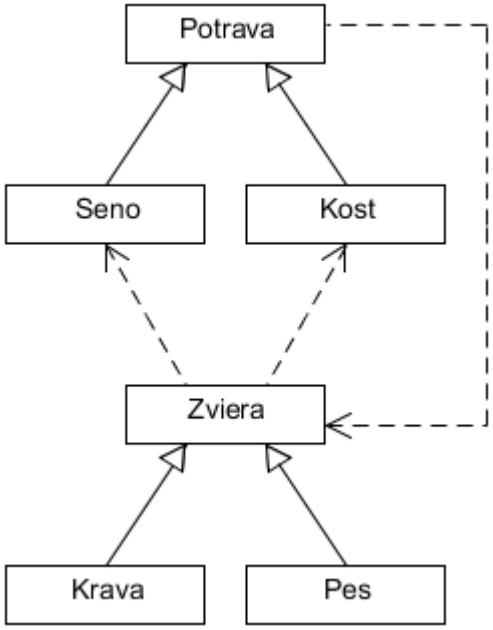
- Upravme implementáciu triedy Zviera

```
public class Zviera {
    public void zjedz(Kost p){
        System.out.println("Som zviera a Kost mi chuti");
    }
    public void zjedz(Seno s){
        System.out.println("Som zviera a Seno mi chuti");
    }
}
```

- Nutná úprava v hlavnej metóde. Zviera už nevie zjesť potravu ale iba jej špecifický podtyp (Kost alebo Seno)

```
    public static void main(String[] args) {
        new Zviera().zjedz(new Potrava());
        new Potrava().nakrm(new Zviera());
    }
}
```

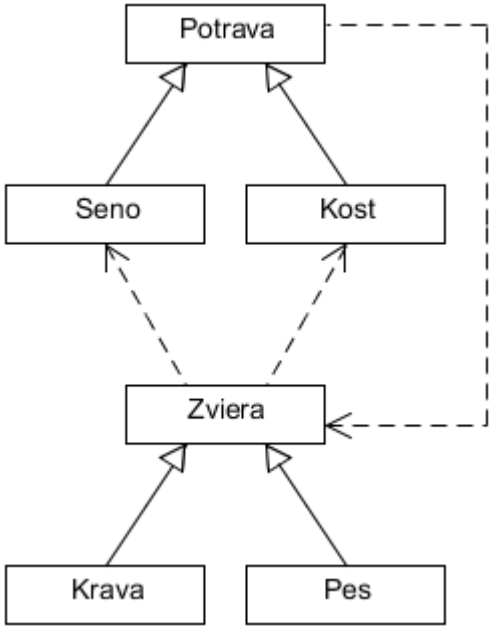
Polymorfizmus



- Rozšírme triedu Zviera o podtypy Krava a Pes

```
class Krava extends Zviera {

    Krava() {
        System.out.println("Som krava");
    }

    @Override
    public void zjedz(Kost p) {
        System.out.println("fuj kosti nejem. Muuuuuuuuuuuu!!!!");
    }

    @Override
    public void zjedz(Seno s) {
        System.out.println("mnam senko....");
    }
}
```

Polymorfizmus



- Rozšírme triedu Zviera o podtypy Krava a Pes

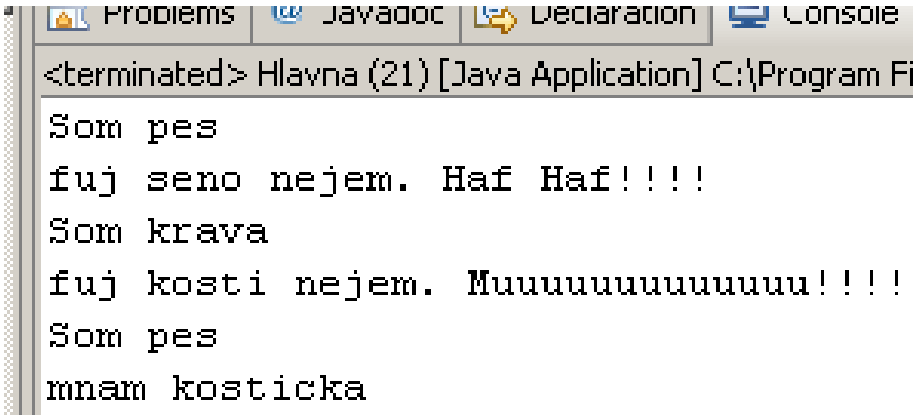
```
public class Pes extends Zviera {

    Pes() {
        System.out.println("Som pes");
    }

    public void zjedz(Kost p) {
        System.out.println("mnam kosticka");
    }

    public void zjedz(Seno p) {
        System.out.println("fuj seno nejem. Haf Haf!!!!");
    }
}
```

Polymorfizmus

```
public class Krmenie {
    Krmenie(Zviera z, Potrava p) {
        p.nakrm(z);
    }
}
```

- Vytvorme triedu Krmenie
- Prejavenie polymorfizmu nachádzame tu:

```
public static void main(String[] args) {
    new Krmenie(new Pes(), new Seno());
    new Krmenie(new Krava(), new Kost());
    new Krmenie(new Pes(), new Kost());
}
```



Polymorfizmus

```
class Krava extends Zviera {

    Krava(int hmotnost) {
        this.hmotnost = hmotnost;
        System.out.println("Som " + this.hmotnost + " kilogramova krava");
    }

    public void zjedz(Kost p) {
        System.out.println("fuj kosti nejem. Muuuuuuuuuuuuu!!!!");
        this.hmotnost--;
    }

    public void zjedz(Seno s) {
        System.out.println("mnam senko....");
        this.hmotnost++;
    }

    public void printHmotnost() {
        System.out.println("Som " + this.hmotnost + " kilogramova krava");
    }
}
```

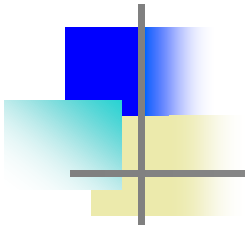
Polymorfizmus

```
public class Pes extends Zviera {

    Pes(int hmotnost) {
        this.hmotnost = hmotnost;
        System.out.println("Som " + this.hmotnost + " kilogramovy pes");
    }

    public void zjedz(Kost p) {
        System.out.println("mnam kosticka");
        this.hmotnost++;
    }

    public void zjedz(Seno p) {
        System.out.println("fuj seno nejem. Haf Haf!!!!");
        this.hmotnost--;
    }

    public void printHmotnost() {
        System.out.println("Som " + this.hmotnost + " kilogramovy pes");
    }
}
```

Polymorfizmus

```
public static void main(String[] args) {
    Zviera p = new Pes(40);
    Zviera k = new Krava(300);
    Potrava s = new Seno();
    Potrava ko = new Kost();
    Pes pes = (Pes) p;
    Krava krava = (Krava) k;


    new Krmenie(p, s);
    new Krmenie(k, ko);
    new Krmenie(k, s);
    pes.printHmotnost();
    krava.printHmotnost();
}
```

```
<terminated> Hlavna (22) [Java Application] C:\Program Fi
Som 40 kilogramovy pes
Som 300 kilogramova krava
fuj seno nejem. Haf Haf!!!!
fuj kosti nejem. Muuuuuuuuuuuuu!!!!
mnam senko....
Som 39 kilogramovy pes
Som 300 kilogramova krava
```



Polymorfizmus

- Polymorfizmus je mechanizmus, ktorý umožňuje objektom rôznych typov odpovedať na volanie rovnakej metódy rôznym spôsobom.

Abstraktné triedy

```
public class Zviera {
    int hmotnost;

    public void zjedz(Kost p){
        System.out.println("Som zviera a Kost mi chuti");
    }
    public void zjedz(Seno s){
        System.out.println("Som zviera a Seno mi chuti");
    }
}
```

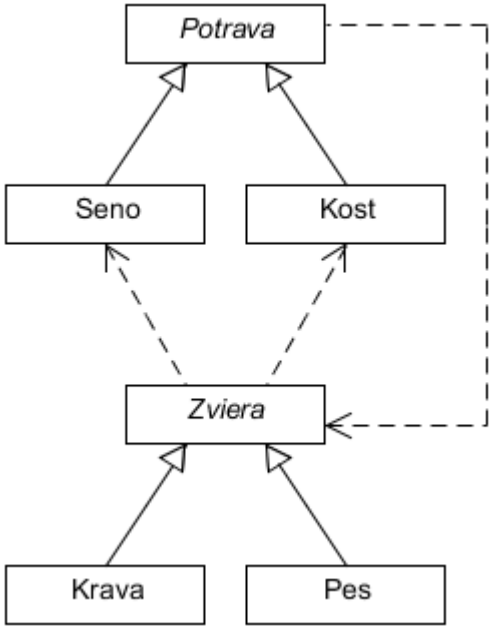
```
public abstract class Zviera {
    int hmotnost;

    public abstract void zjedz(Kost p);
    public abstract void zjedz(Seno s);
}
```

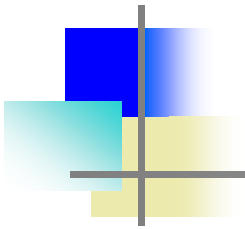
Abstraktné triedy

- Abstraktné triedy sú v UML diagramoch značené kurzívou

```
public class Potrava {
    void nakrm(Zviera z) {
        System.out.println("Zviera z je nakrmene");
    }
}
```

```
public abstract class Potrava {
    abstract void nakrm(Zviera z);
}
```





Rozhrania

- V Jave nie je dovolené viacnásobné dedenie
- Možnosť dedenia z viacerých tried je niekedy veľmi potrebná
- Riešením tohto problému je v Jave mechanizmus rozhraní (interfejsov)
- Rozhranie je taktiež trieda (aj keď špeciálna)
- Termín implementácia rozhrania pre triedu potomka znamená, že táto trieda "zdedí" vlastnosti „triedy“ v ktorej definujeme rozhranie
- Rozhranie sa používa pre vytváranie generických, resp. všeobecných šablón, ktoré môžu byť použité v iných triedach (normálnych alebo abstraktných)
- V triede rozhrania sú zvyčajne deklarácie metód, ktoré sú implicitne definované ako verejné (`public`) a abstraktné (`abstract`). Od Java 8 možná aj definícia.

Rozhrania

- Môžu obsahovať aj vlastnosti, ktoré sú implicitne verejné (`public`) a statické konštanty (`static final`)
- Definícia rozhrania začína kľúčovým slovom `interface`. Z rozhrania, podobne ako z abstraktnej triedy nemôže byť vytvorený objekt
- Mechanizmus dedenia je povolený aj pri rozhraniach. Znamená to že môžeme vytvoriť rozhranie, ktoré bude rozširovať iné rozhranie
- Java nepodporuje viacnásobné dedenie, avšak je možné použiť viacnásobnú implementáciu rozhraní [1]
- Rozhranie predpisuje správanie pre triedy, ktoré ho implementujú
- A class defines who you are, and an interface tells what roles you could play[1]

Aké roly môže zohrávať naše zviera?

[1] Java Abstract class and Interface - https://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class

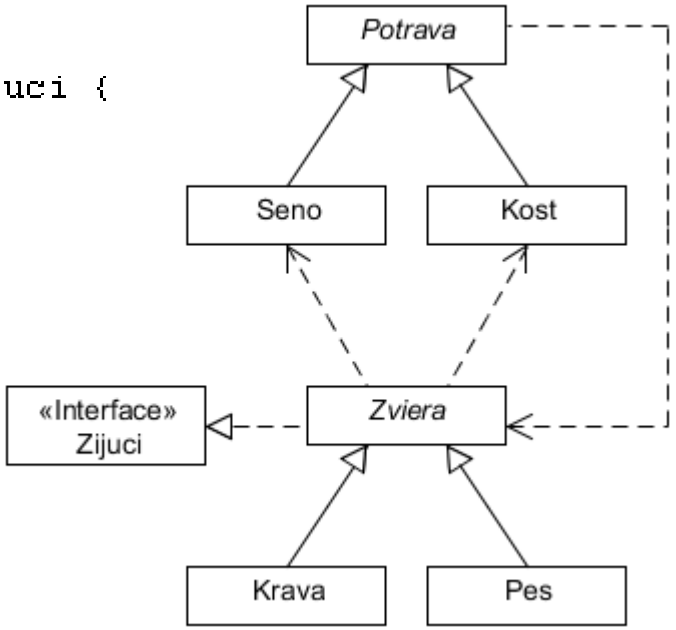
Rozhrania

- Rozhranie bude definovať základné metódy týkajúce sa "života" zvieraťa

```
public interface Zijuci {
    public void rozmnozovat();
}
```

- V abstraktnej triede nie je potrebná jeho implementácia

```
public abstract class Zviera implements Zijuci {

    public abstract void zjedz(Kost p);
    public abstract void zjedz(Seno s);
}
```



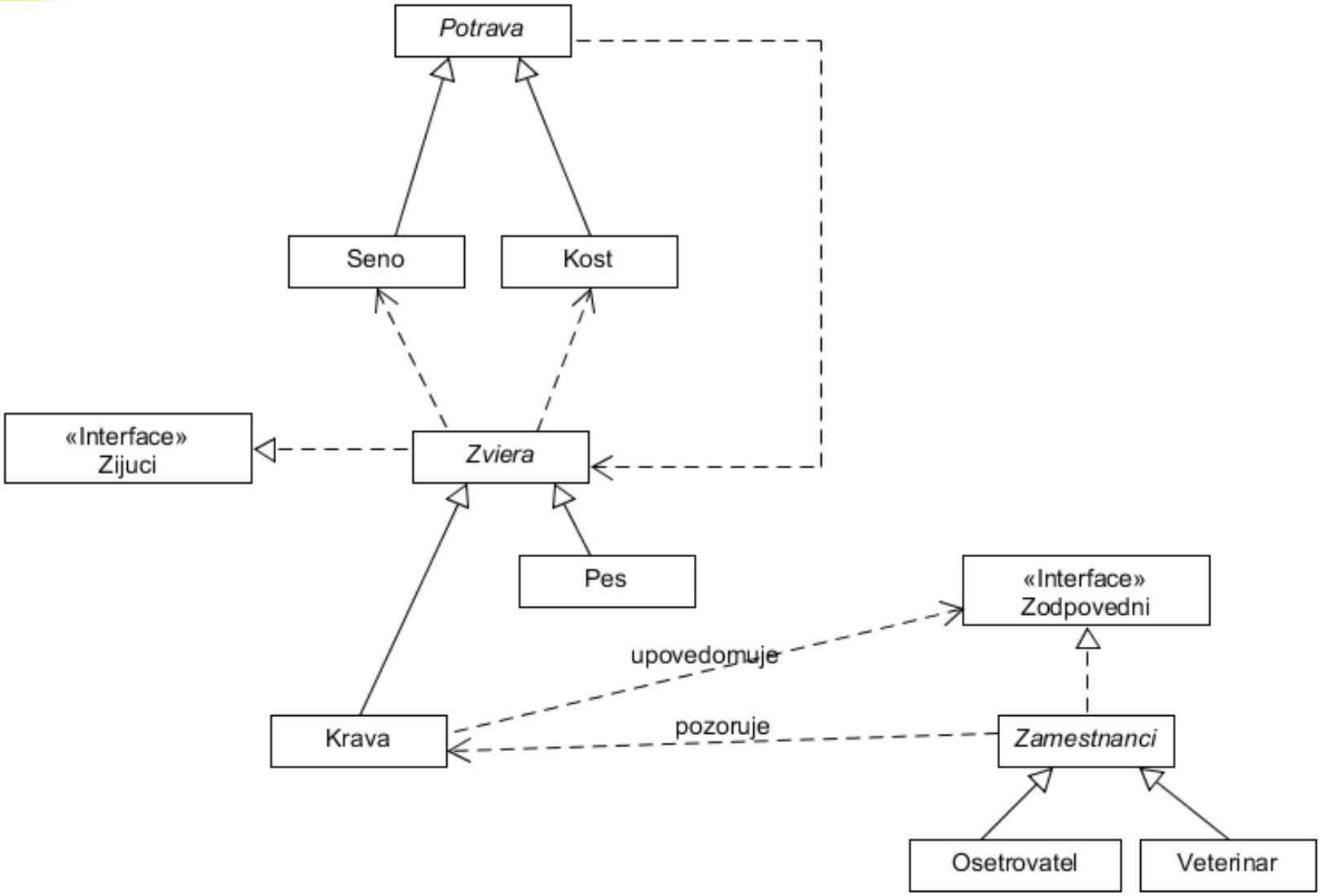
Rozhrania

- Použitie rozhrania
  - Za účelom upcast-u do viac než jedného bázového typu (nadtypu)
  - Za podobným účelom ako abstraktné triedy
  - Abstraktné triedy neumožňujú viacnásobné dedenie. Je to možné vyriešiť implementáciou viacerých rozhraní - preto odpoveď na otázku čo skôr použiť abstraktnú triedu/rozhranie = použite rozhranie
  - Pozor na možné kolízie v mennom priestore

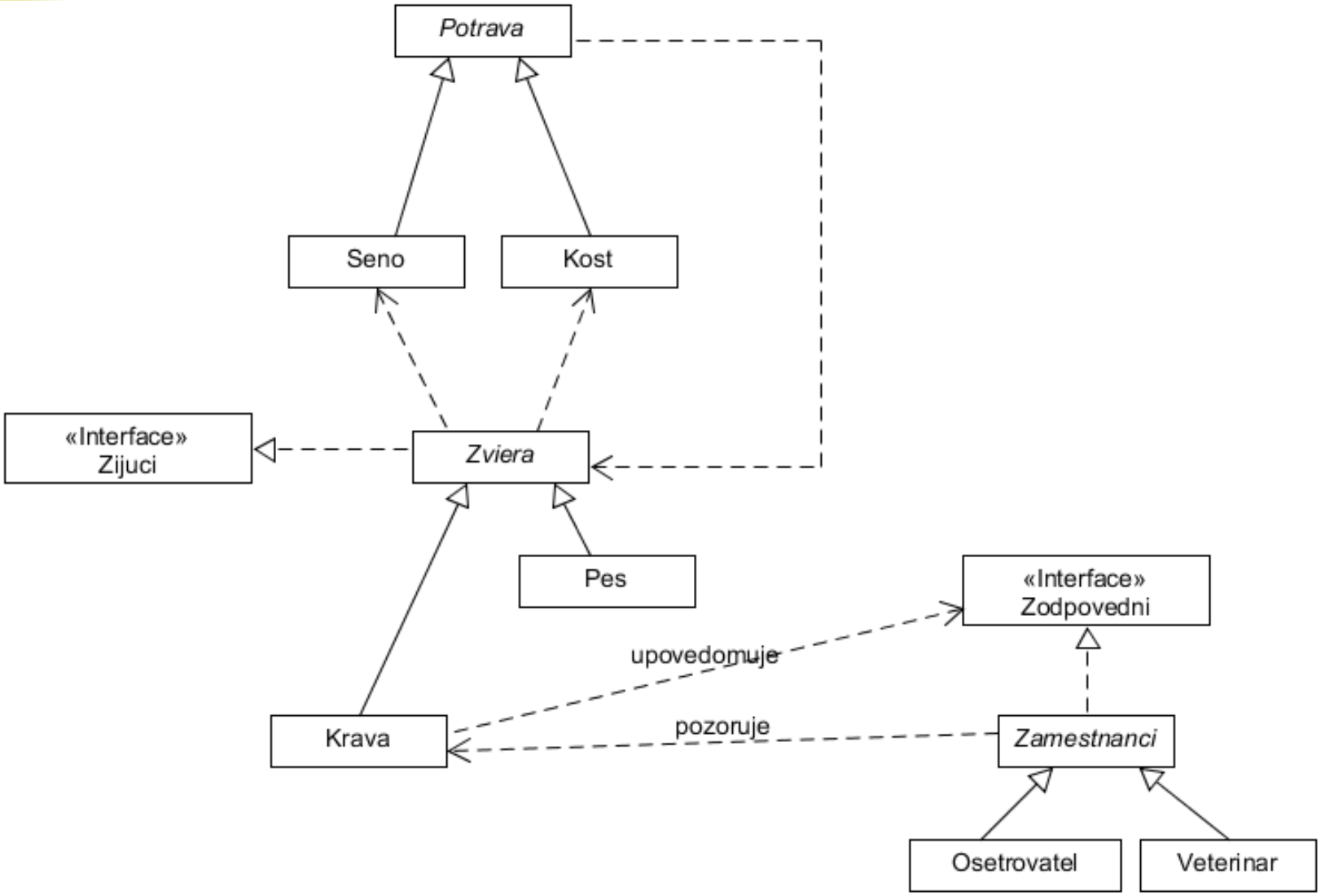
`sk.stuba.fiit.zoo_spravca`

  - Správca???

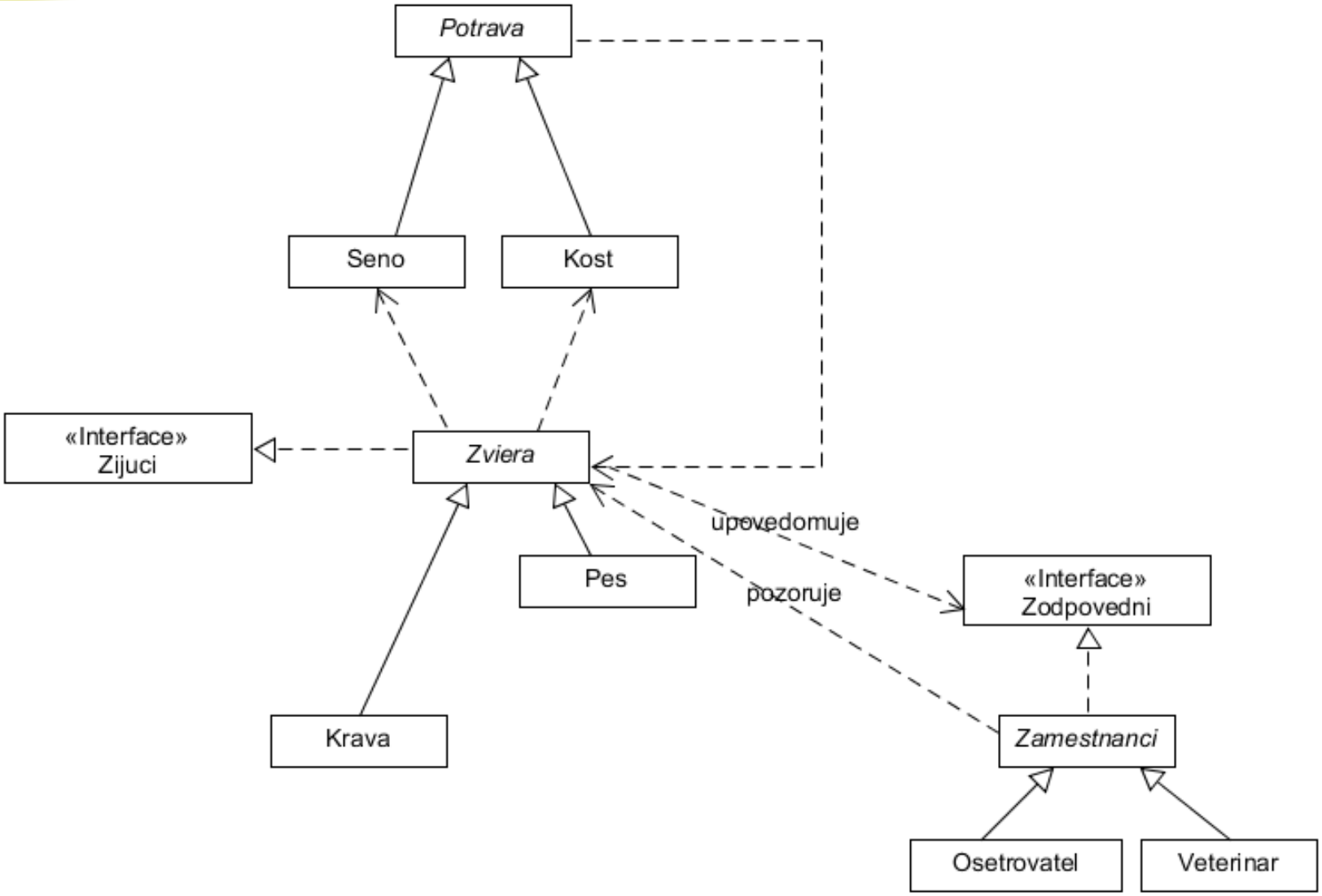
Viac rozhraní



Prečo je tá krava tak dôležitá?



Prečo je tá krava tak dôležitá?



Rozhrania

```
class Hero extends ActionCharacter implements CanFight, CanSwim, CanFly {

    Hero() {
        System.out.println("Hero was created");
    }

    public void swim() {
    }

    public void fly() {
    }
}
```

```
interface CanFight {
    void fight();
}

interface CanSwim {
    void swim();
}

interface CanFly {
    void fly();
}

class ActionCharacter {
    public void fight() {
    }
}
```

Bruce Eckel, TIJ, 2002

Rozhrania

```
public class Adventure {
    public static void t(CanFight x) {
        x.fight();
    }

    public static void u(CanSwim x) {
        x.swim();
    }

    public static void v(CanFly x) {
        x.fly();
    }

    public static void w(ActionCharacter x) {
        x.fight();
    }

    public static void main(String[] args) {
        Hero h = new Hero();
        t(h); // Považujeme za inštanciu CanFight
        u(h); // Považujeme za inštanciu CanSwim
        v(h); // Považujeme za inštanciu CanFly
        w(h); // Považujeme za inštanciu ActionCharacter
    }
}
```

TODO nezabudnite

- Aj vy môžete pomôcť vylepšiť tento predmet študentom pre nasledujúci akademický rok. Vaše odporúčanie, komentár či otázka.

  ...cez spätnoväzobný formulár.