Objektovo orientované programovanie

Učiteľ:
Ing. Jozef Wagner PhD.

Učebnica:
https://oop.wagjo.com/

OPG
Teória 13

1. Abstrakcia
2. Abstraktné triedy
3. Rozhrania

Abstrakcia

Abstrakcia je proces, pri ktorom skryjeme
nepodstatné detaily a ponecháme iba dôležité veci

Cieľom je aby sa zverejnilo iba to,
"čo" robí objekt, nie **"ako"** to robí

To zjednodušuje kód, zvyšuje čitateľnosť a modularitu

Ciele abstrakcie:

- Používateľ triedy by mal vidieť, čo trieda robí
  (API, verejné metódy)
- Implementácia triedy *(ako to robí)* má byť skrytá

Abstraktná trieda

Špeciálny typ triedy, ktorá obsahuje abstraktné metódy.

Abstraktné metódy sú bez implementácie - bez tela metódy

Definujú sa s kľúčovým slovom `abstract`

Abstraktná trieda je taká napoly dokončená trieda

Abstraktná trieda môže mať atribúty, konštruktor, aj obyčajné metódy

Abstraktná trieda

```
abstract class Zviera {
    protected String nazov;

    public Zviera(String nazov) {
        this.nazov = nazov;
    }

    public String getNazov() {
        return nazov;
    }

    // metóda nemá telo
    public abstract void zvuk();
}
```

Nevieme priamo vytvoriť objekty abstraktnej triedy

```
Zviera z = new Zviera("Pes");
```

Úlohou abstraktnej triedy je, aby ju dedili iné triedy

Abstraktná trieda

```
class Pes extends Zviera {
    public Pes(String nazov) {
        super(nazov);
    }

    void donesPapuce() {
        // ...
    }

    @Override
    void zvuk() {
        System.out.println("Haf");
    }
}
```

```
class Macka extends Zviera {
    private boolean pazury = false;

    public Macka(String nazov) {
        super(nazov);
    }

    void vysunPazury() {
        pazury = true;
    }

    @Override
    void zvuk() {
        System.out.println("Mňau");
    }
}
```

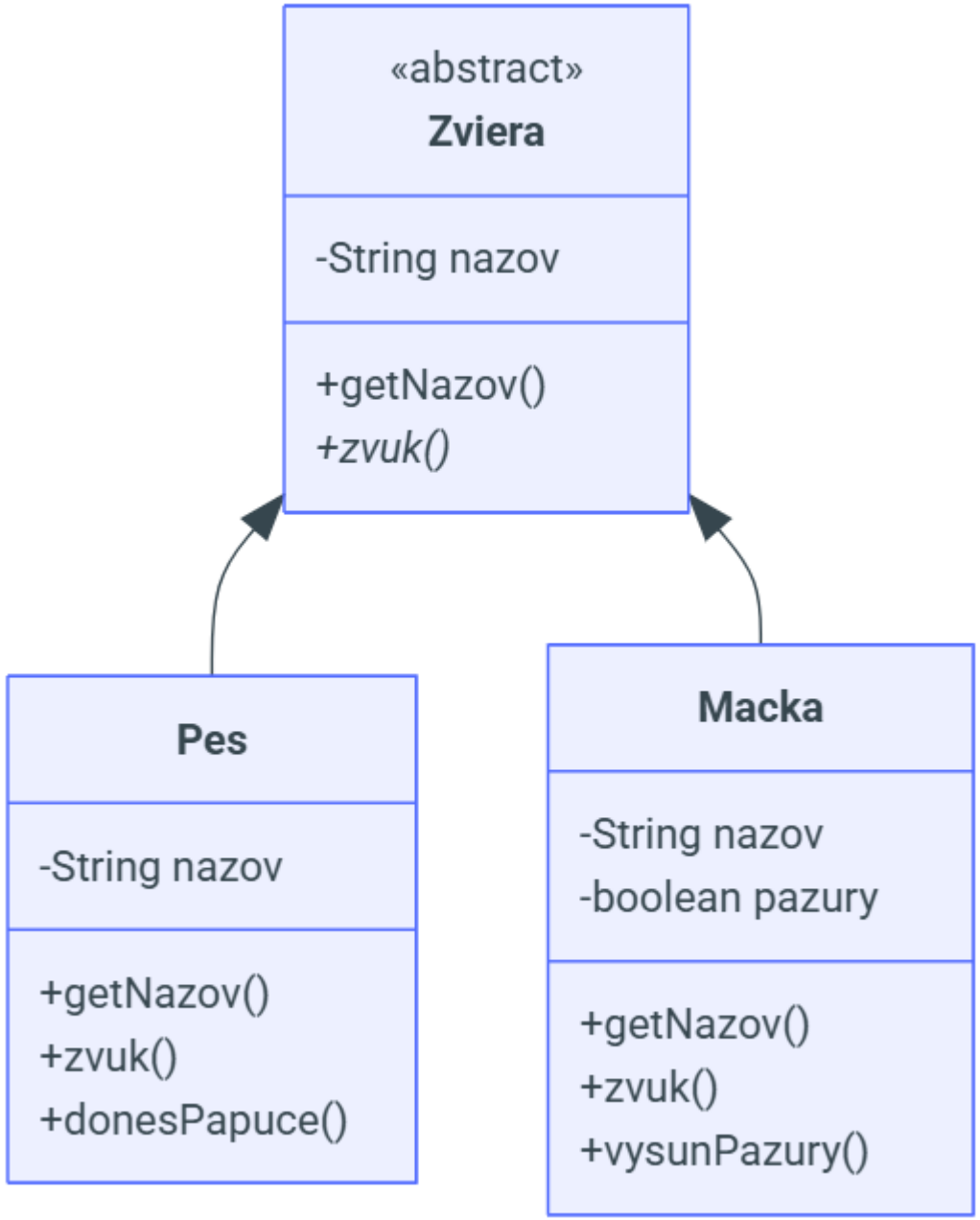
```
class Zverinec {
    private Zviera[] zvierata;

    public Zverinec(Zviera[] zvierata) {
        this.zvierata = zvierata;
    }

    public void hluk() {
        for (Zviera zviera : zvierata) {
            zviera.zvuk();
        }
    }

    public static void main(String[] args) {
        Zviera[] zvierata = {new Pes("Dunco"), new Macka("Elza")};
        Zverinec zverinec = new Zverinec(zvierata);
    }
}
```

Abstraktná trieda

V UML diagrame má značku `<<abstract>>`, a abstraktné metódy sú uvedené šikmým písmom (italic)

Abstraktné triedy používajú dedenie, teda IS-A **"je"** vzťah.

Pes je Zviera, Auto je Vozidlo



Rozhranie - Interface

Sústredí sa vyslovene na správanie, teda na schopnosti

Špeciálna trieda, ktorá **nemá atribúty ani konštruktory**

Rozhranie je **čistý dizajn, bez stavu a bez konkrétnej implementácie**

Rozhranie definujeme s kľúčovým slovom `interface`

Rozhranie

```
interface Lietaj {
    void vzlietni();
}

interface Potapaj {
    void ponorSa();
}

interface NadmorskaVyska {
    double getNadmorskaVyska();
}
```

Podtrieda môže implementovať viacero rozhraní!

```
class Lietadlo implements Lietaj,
NadmorskaVyska {

  @Override
  public void vzlietni() {
        // ...
  }

  @Override
  public void getNadmorskaVyska() {
        // ...
  }
}
```

Rozhranie

```
class Kacka implements Lietaj,
Potapaj NadmorskaVyska {
   @Override
   public void vzlietni() {
         // ...
   }
   @Override
   public void ponorSa() {
         // ...
   }
   @Override
   public void getNadmorskaVyska() {
         // ...
   }
}
```

```
class Mesto implements NadmorskaVyska
{
     @Override
     public void getNadmorskaVyska() {
          // ...
     }
}
```

Rozhranie sa sústredí na schopnosti triedy,
teda ide o CAN-DO **"vie"** vzťah.

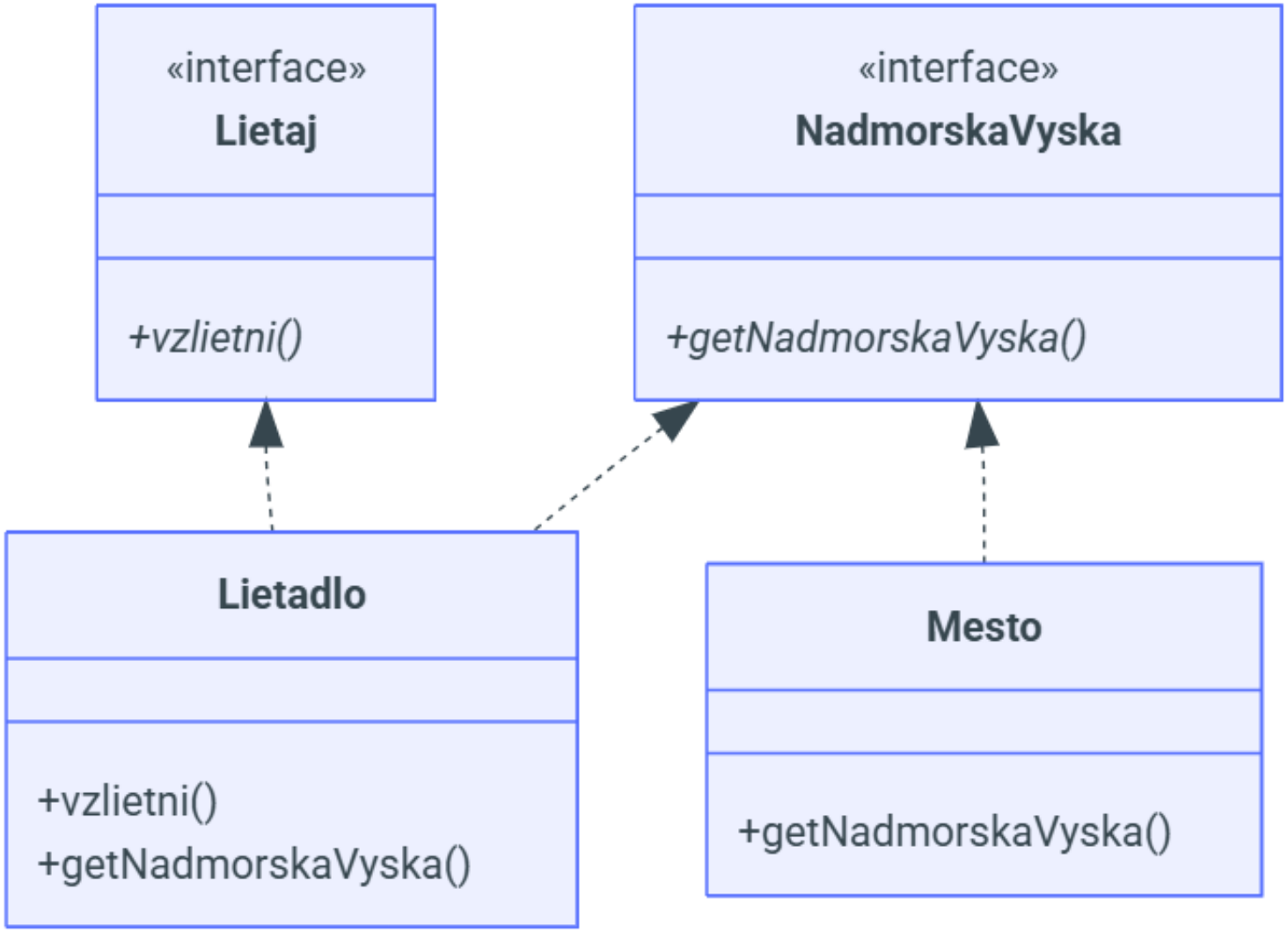
Lietadlo vie Lietať,

Súbor vieme Zatvoriť

Rozhranie

Rozhrania v UML Class diagrame majú pred menom značku `<<interface>>`, a abstraktné metódy sú uvedené šikmým písmom (italic).

Od triedy, ktorá implementuje rozhranie, vedie prerušovaná uzatvorená šípka k danému rozhraniu.



Rozhrania v Java API

Rozhrania sa vo veľkom používajú aj v knižniciach Javy

- `List`, `Set`, `Map` - kolekcie
- `Iterable`, `Iterator` - iterácia prvkov v cykle
- `Comparable`, `Comparator` - zoradenie hodnôt
- `AutoCloseable`, `Cloneable` - zatvorenie a kopírovanie

Abstraktná trieda Vs Rozhranie

Rozhranie je jednoduchšie, flexibilnejšie a menej obmedzujúce

Cieľom je zložitosť umenšovať a skrývať do implementácie, nie ju pridávať

Abstraktná trieda: Pre IS-A "je" vzťahy (napr. Pes je Zviera), zdieľa stav a správanie

Rozhranie: Pre **"môže-robiť"** vzťahy (napr. Lietadlo môže Lietať), čisto o správaní

Obidve prístupy môžeme kombinovať