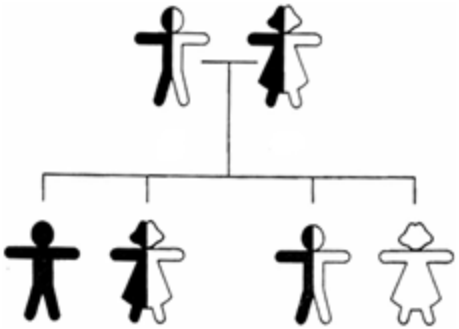
PAZ

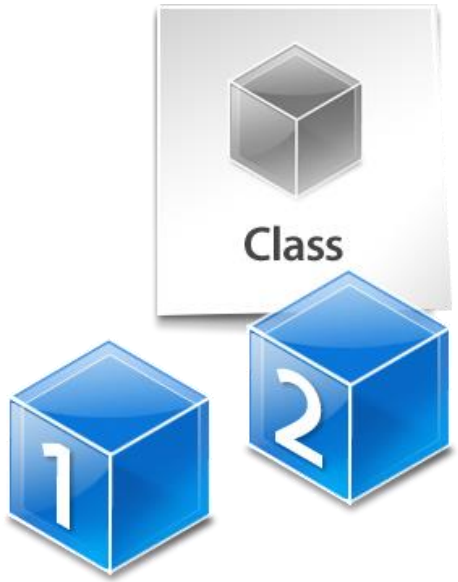
10. prednáška



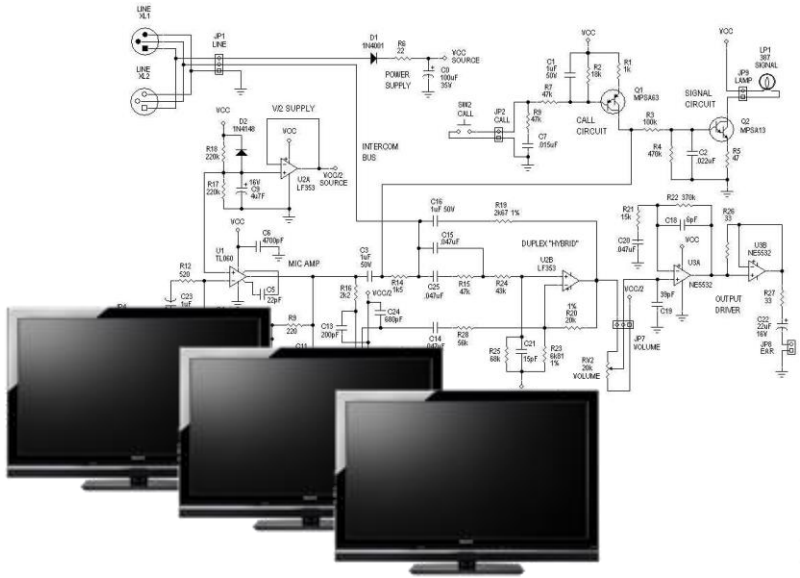
Dedičnosť a polymorfizmus

Čo je to trieda?

- **Trieda** je **šablóna** (vzor), ktorý **predpisuje** aké **inštančné premenné** a aké **metódy** majú objekty danej triedy a čo sa udeje pri zavolaní týchto metód

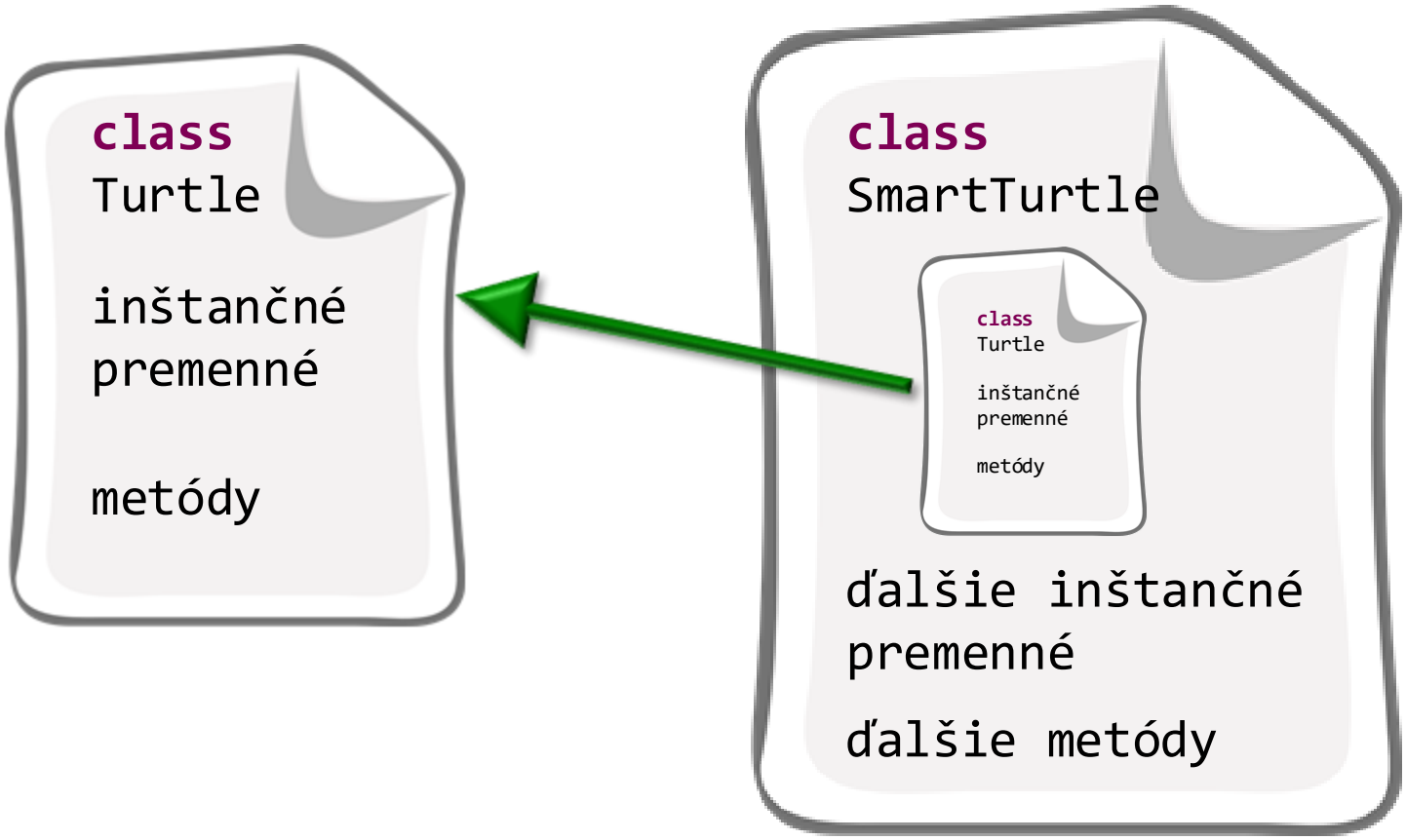



```
class
Turtle

inštančné
premenné

metódy
```



Rozširovanie tried

```
public class SmartTurtle extends Turtle
```



Premenné referenčného typu

```
Turtle franklin;
```

Referencie na objekty **akej triedy** môžu byť uložené v premennej

Názov premennej

- Premenná `franklin` môže referencovať len objekty triedy `Turtle` („uchovávať ich rodné čísla“)
- Špeciálna hodnota **`null`** určujúca, že premenná neuchováva referenciu na objekt.

Nočná mora: Duplicita kódu

- Duplicita kódu je **nežiadúca** z mnohých dôvodov:
  - chyby
  - údržba
  - optimalizácia
  - budúce zmeny

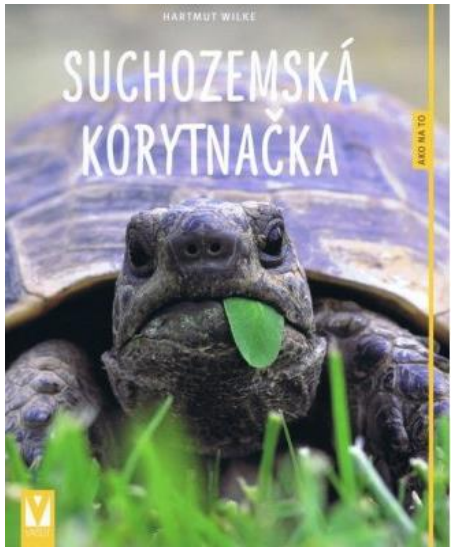
- Riešenie:
  - metódy
  - triedy
  - „knižnice“

Zadanie (z minula)

- Cieľ: pohodlná správa knižnice, resp. zbierky kníh.
- Vyžadovaná **funkcionalita**:
  - vieme vložiť info o novej knihe
  - odstrániť knihu zo zbierky
  - vypísať názvy všetkých kníh v knižnici
  - vypísať tie knihy, ktoré boli vydané v konkrétnom roku
  - vypísať tie knihy, ktoré napísal konkrétny autor
  - vypísať tie knihy, ktoré majú podľa nášho hodnotenia aspoň 8 bodov (na stupnici 0-10)
  - ...

Zadanie (z minula)

- Dôležité informácie o každej knihe:
  - názov knihy
  - mená autorov
  - rok vydania knihy
  - hodnotenie kvality knihy: 0-10



Zadania pre programy

- V každom rozumnom zadaní sa špecifikujú dve kľúčové (základné) množiny požiadaviek:
  - **s akými dátami** bude program pracovať
    - … inštančné premenné
  - **aké služby** má poskytovať resp. **akú funkcionalitu** má program (objekty triedy) mať
    - … metódy

Zapúzdrenie (Encapsulation)

- **Zabraňuje** priamemu prístupu k dátam (vnútorným častiam) objektu
- **Dáta a metódy**, ktoré s nimi pracujú, sú spolu
- Každý objekt navonok sprístupňuje rozhranie (=metódy), pomocou ktorého (a nijako inak) sa s objektom pracuje
  - objekty sú **zodpovedné** za konzistentný obsah svojich inštančných premenných
  - s objektami sa chceme rozprávať **iba cez ich metódy**
- Konštruktory, metódy (getter, setter)

Konštruktor

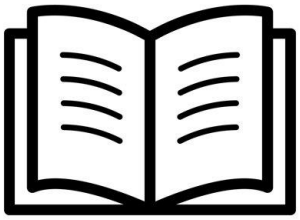
- Inicializuje inštančné premenné (aj na základe hodnôt parametrov)

```
public class Book {

  public Book(String title) {
    this(title, 0, 0);
  }

  public Book(String title, int publicationYear,
              double rating) {
    this.title = title;
    this.publicationYear = publicationYear;
    this.rating = rating;
  }
}
```

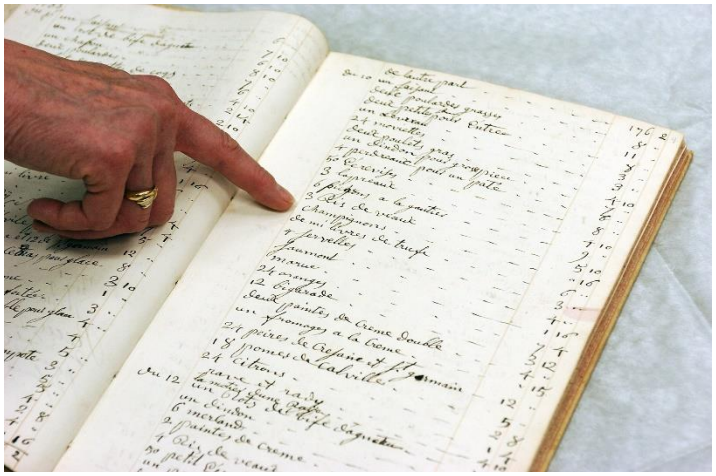
Volanie iného konštruktora (ak sa použije, musí to byť **prvý príkaz** konštruktora)

Rozširujeme zadanie

- rôzne typy kníh:
  - Vytlačená kniha, e-kniha, audio kniha
  - Kde sú uložené?
  - Aký majú rozsah?

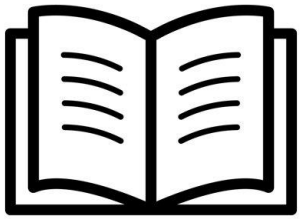


Rozširujeme zadanie



- Vytlačená kniha
  - číslo police
  - typ väzby, formát, počet strán
- E-kniha
  - link (URL alebo cesta k súboru)
  - počet (normo)strán
- Audio kniha
  - adresár
  - veľkosť (napr. v KB) a dĺžka (napr. v minútach)
  - mená interprétov

Aké triedy?



```
class
PrintedBook

inštančné
premenné

metódy
```

```
class
EBook

inštančné
premenné

metódy
```

```
class
AudioBook

inštančné
premenné

metódy
```

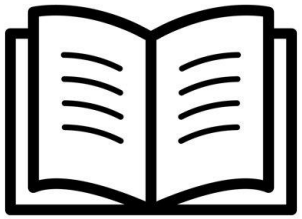
Veľa rozdielneho aj spoločného...

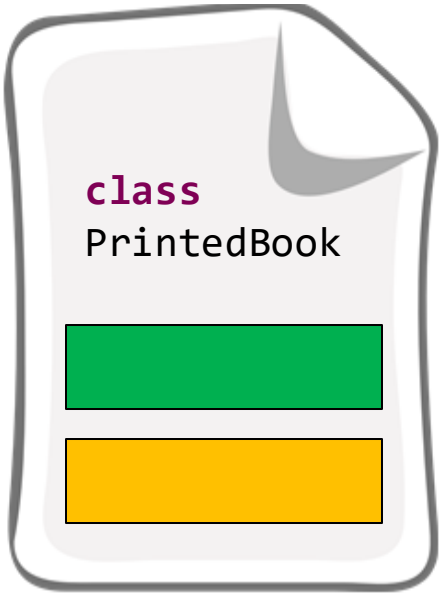
- Knihy v ľubovoľnej forme majú niektoré rozdielne dáta:
  - identifikácia umiestnenia
  - rôzne definovaná veľkosť
    - (počet strán - iný formát, dĺžka v minútach)
  - doplňujúce údaje
- Knihy v ľubovoľnej forme majú aj rozdielne správanie funkčných schopností:
  - výpis umiestnenia knihy
  - spôsob uloženia do súboru a načítania z neho
  - poskytovanie dodatočných informácií

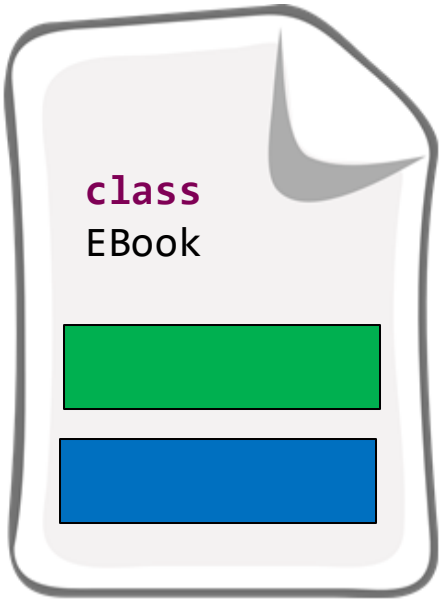
Veľa rozdielneho aj spoločného...

- Knihy v ľubovoľnej forme majú ale aj spoločné dáta:
  - názov knihy
  - mená autorov
  - rok vydania
  - hodnotenie kvality knihy na stupnici od nula do desať.
- Knihy v ľubovoľnej forme majú aj spoločnú funkcionalitu:
  - `String getTitle()`
  - `boolean hasAuthor(String authorName)`
  - `String toString()` - vypisuje (zatiaľ) len spoločné dáta

Aké triedy?





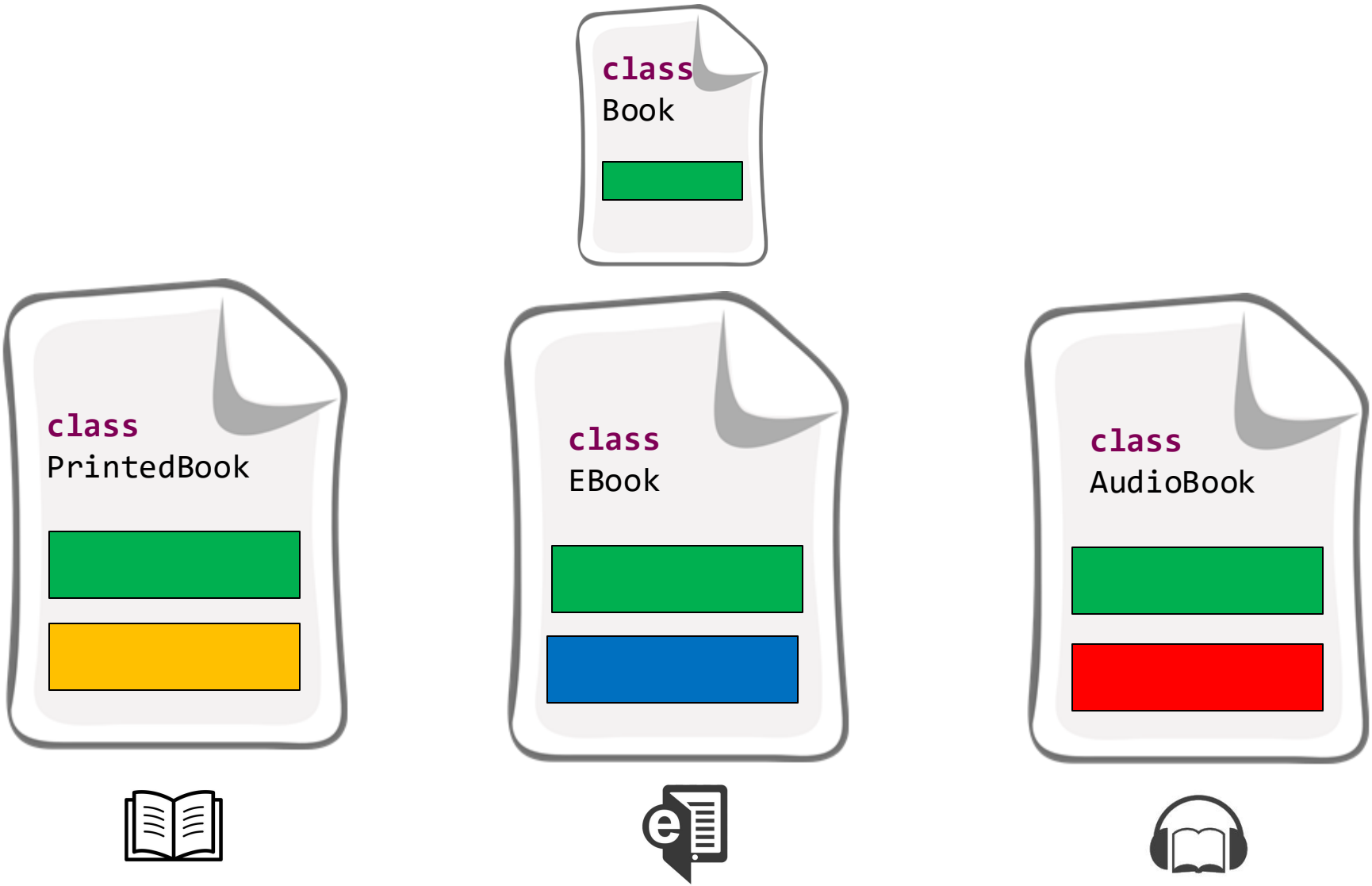


Nočná mora: Duplicita kódu

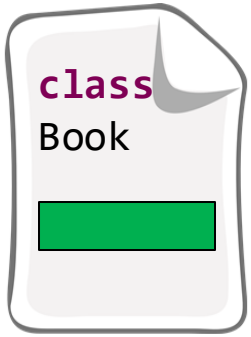
- Duplicita kódu je **nežiadúca** z mnohých dôvodov:
  - chyby
  - údržba
  - optimalizácia
  - budúce zmeny

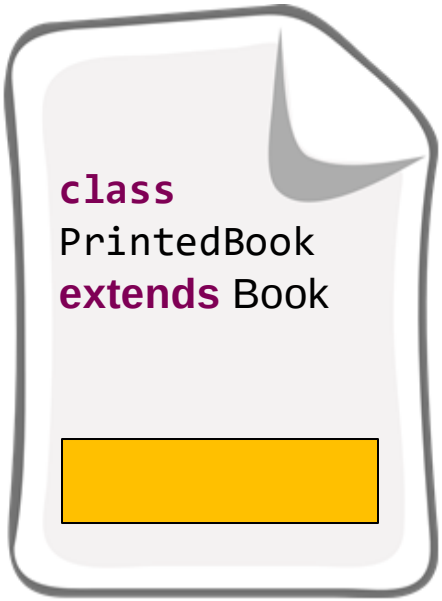
- Riešenie:
  - metódy
  - triedy
  - „knižnice“

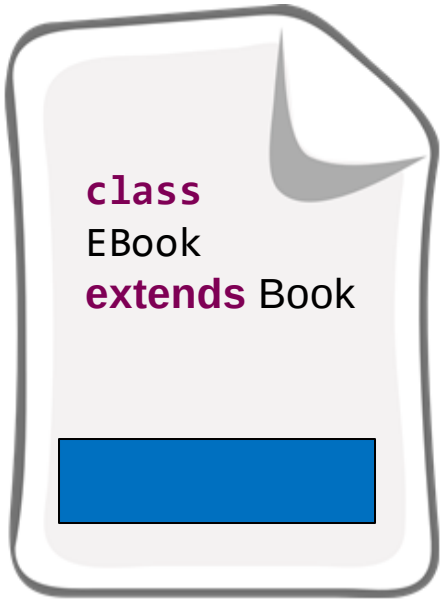
Aké triedy?

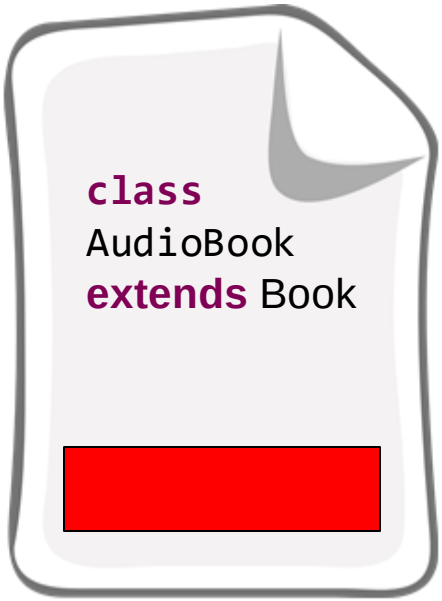


Eliminácia duplicity rozšírením









Dedičnosť = rozširovanie

- Vytvoríme si triedu `Book`, ktorá
  - obsahuje spoločné dáta a metódy pre všetky knihy bez ohľadu na formu, v akej vystupuje
- Od nej oddedené triedy, t.j. triedy, ktoré rozširujú vlastnosti triedy `Book` o:
  - `PrintedBook`
    - **shelfNumber**, pagesCount, format, cover
  - `EBook`
    - **link**, standardPagesCount
  - `AudioBook`
    - **folder**, size, length, narrators

getLocation

- Každý „knižný“ objekt vie povedať, kde sa nachádza
  - `String getLocation()`
- Každý to povie po svojom (nejde o kópiu)
  - `PrintedBook`
    - polica č. 15
  - `Ebook`
    - https://eknizky.sk/books/robinson-crusoe-2/
  - `AudioBook`
    - C://Users/PAZ/Documents/Books/FranklinGoesToSchool/

Konštruktory a dedičnosť

- Prvý príkaz konštruktora **musí byť vždy** volanie **konštruktora rodičovskej** (rozširovanej) triedy alebo iného konštruktora vytváranej triedy.
- Konštruktor rodičovskej triedy voláme cez

```
super(...parametre...);
```

- Java pre „lenivých“:
  - ak programátor pravidlo nedodrží, Java doplňuje do prvého riadku konštruktora: `super();`
  - pozor: rodičovská trieda nemusí mať bezparametrový konštruktor → problém (chyba) už pri vytvorení triedy

Implicitný konštruktor

- Každá trieda **má aspoň jeden konštruktor**
- Ak nie je žiaden konštruktor napísaný programátorom, doplní sa neviditeľný implicitný konštruktor:

```
public class Book{

   public Book() {
      super();
   }
...
}
```

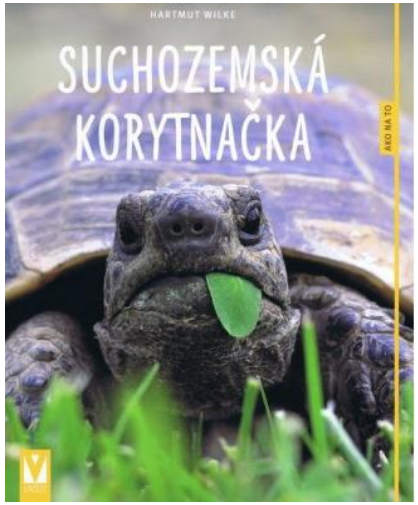
Takto by vyzeral implicitný konštruktor keby ho bolo vidieť

Minule...

Zoznam/správca kníh

```
public class Library
```

Library **závisí od** triedy Book



Zmysel života?

```
public class Book
```

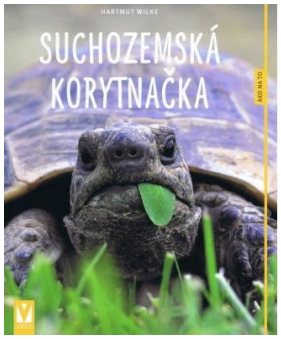
Kniha

Dnes

Zoznam/správca kníh

```
public class Library
```

čo a ako má uchovať?




```
public class Book
```

```
public class EBook
```

```
public class PrintedBook
```

```
public class AudioBook
```

Renovujeme zoznam kníh

- Prvý nápad:
  - Máme 3 triedy, dáme 3 polia

```
public class Library {
     private PrintedBook[] printedBooks;
     private EBook[] eBooks;
     private AudioBook[] audioBooks;
...
}
```

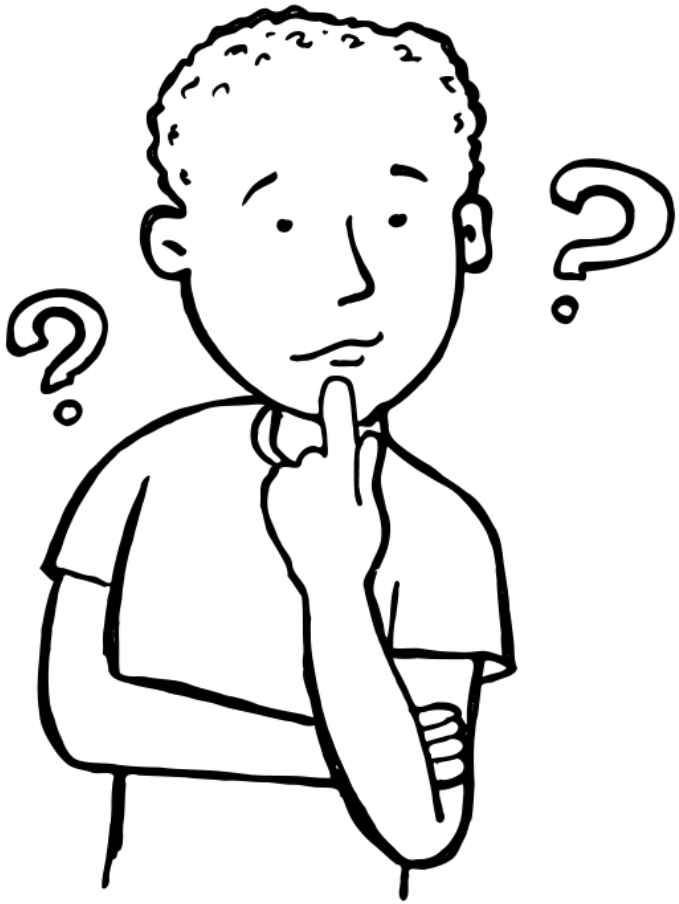
Renovujeme zoznam kníh

- Prvý nápad:
  - Ale potom máme všade 3 cykly

```
public class Library {
...
 public void vypisVsetko() {
  for (int i = 0; i < printedBooks.length; i++)
    System.out.println(printedBooks[i].toString());
  for (int i = 0; i < eBooks.length; i++)
   System.out.println(eBooks[i].toString());
  for (int i = 0; i < audioBooks.length; i++)
    System.out.println(audioBooks[i].toString());
 }
...
}
```

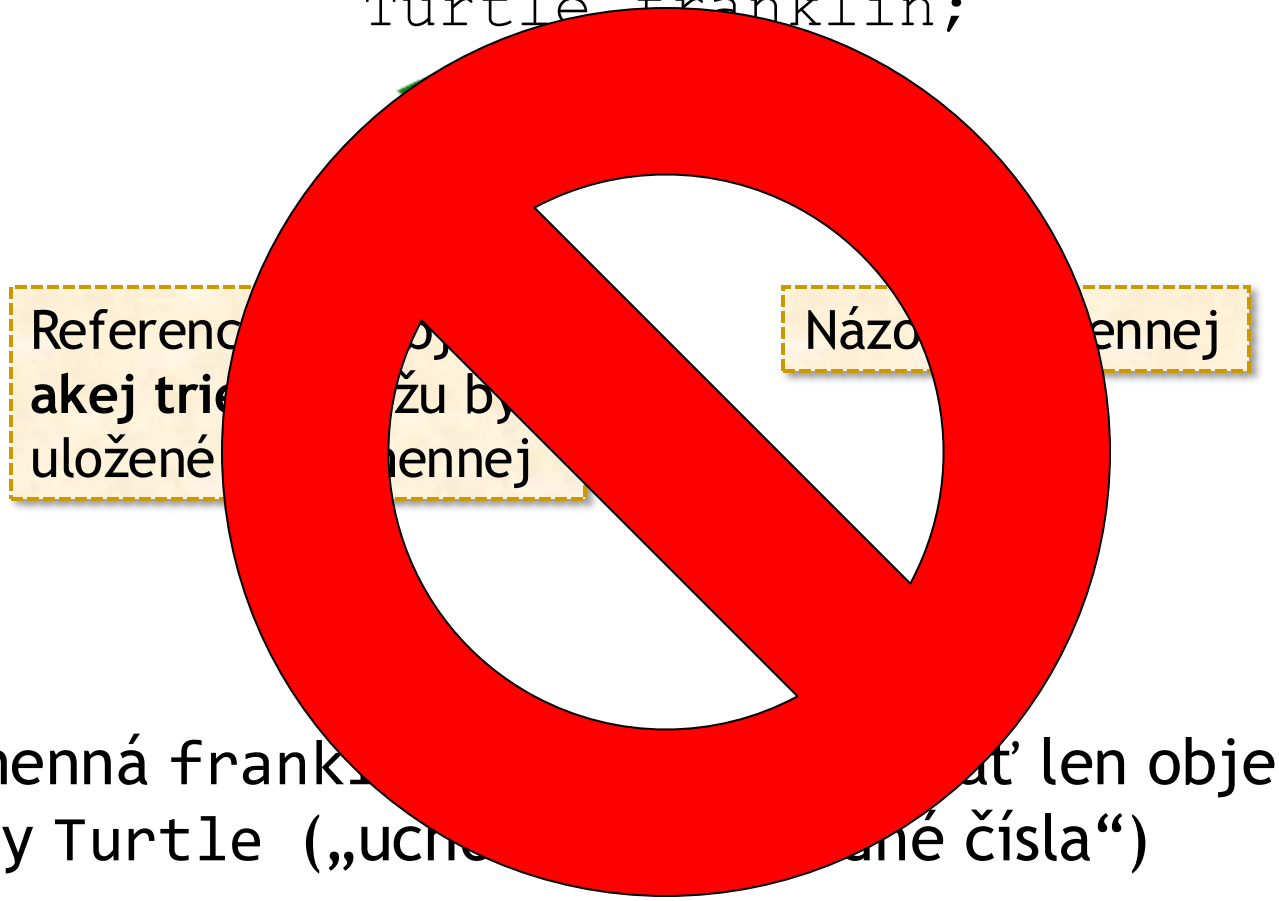
Renovujeme zoznam kníh

- Filozofická úvaha:
  - `PrintedBook` je `Book`
  - `EBook` je `Book`
  - `AudioBook` je `Book`



Premenné referenčného typu

```
Turtle franklin;
```



- Premenná frank… …ať len objekty triedy Turtle („uch… …né čísla“)
- Špeciálna hodnota **null** určujúca, že premenná neuchováva referenciu na objekt.

Premenné referenčného typu

```
Turtle franklin;
```

Referencie na objekty **akej triedy** môžu byť uložené v premennej

Názov premennej

- Premenná `franklin` môže referencovať len objekty triedy `Turtle` **a tried, ktoré rozširujú triedu `Turtle`**

Premenné referenčného typu

```
Turtle franklin = new SmartTurtle();
franklin.step(100);
franklin.smartMethod();
```

Chyba: Cez premennú `franklin` môžem volať len metódy z triedy `Turtle`. Metódy definované v triede `SmartTurtle` sú nedostupné.

```
SmartTurtle franklin = new Turtle();
```

Renovujeme zoznam kníh

- Referencie typu `Book` môžu uchovávať aj referencie na objekty tried `PrintedBook`, `EBook`, `AudioBook`
  - vieme pristupovať k metódam z triedy Book
  - nevieme pristupovať k metóde `getLocation`

```
public class Library {
  private Book[]  books;
  ...

}
```

Renovujeme zoznam kníh

- Riešenie:
  - Máme síce 3 triedy, ale stačí nám 1 pole
  - Stačí nám všade iba 1 cyklus

```
public class Library {
    private Book[]  books;
...
    public void printAll() {
        for (int i = 0; i < books.length; i++) {
            System.out.println(books[i].toString());
        }
    }
...
}
```

Ale máme problém

- Problém:
  - Keď všetko je kniha, ako zistíme umiestnenie?
  - Nevieme predsa zavolať `getLocation...`

```
public class Library {
...
    public void printAll() {
        for (int i = 0; i < books.length; i++) {
            System.out.println(books[i].toString());
            System.out.println(books[i].getLocation());
        }
    }
...
}
```

The method `getLocation()` is undefined for the type Book

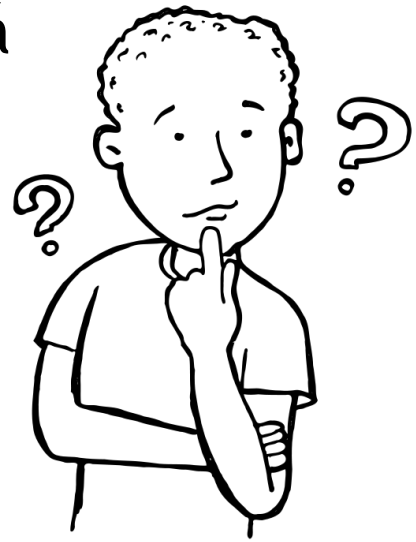
Identifikujeme a pretypujeme

- Operátor **`instanceof`**:
  - Vieme porovnať triedu objektu referencovaného z premennej

```
public class Library {
...
  public void printAll() {
    for (int i = 0; i < books.length; i++)
      System.out.println(books[i].toString());
      if (books[i] instanceof PrintedBook) {
        PrintedBook pBook = (PrintedBook) books[i];
        System.out.println(pBook.getLocation());
      }
      if (books[i] instanceof AudioBook) {
        AudioBook aBook = (AudioBook) books[i];
        System.out.println(aBook.getLocation());
      }
      ...
  }
}
```

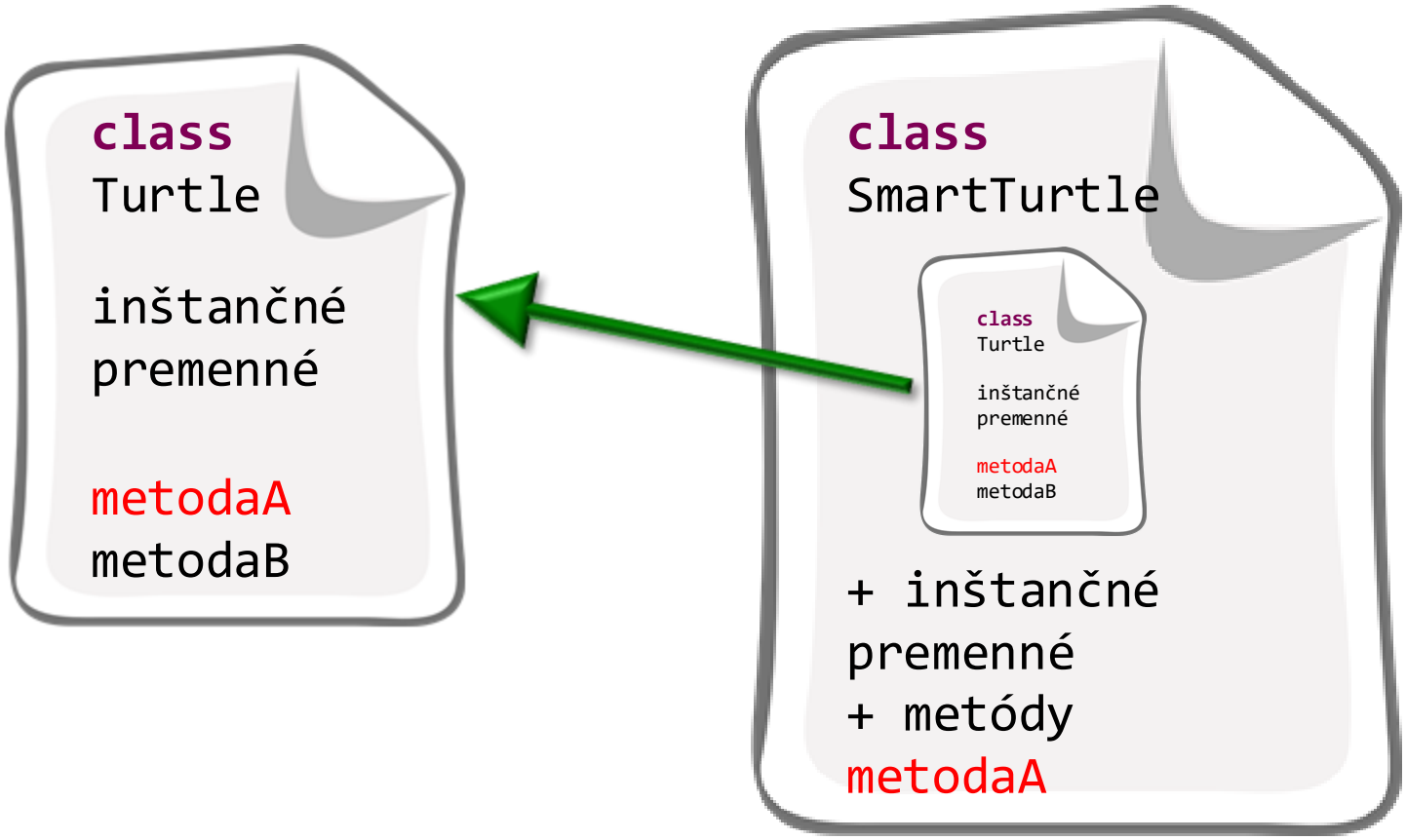
Renovujeme zoznam kníh

- Každá kniha má nejaké umiestenie
  - pridáme metódu `getLocation` do triedy Book?
- `PrintedBook`, `EBook`, `AudioBook` v rámci metódy `getLocation` spravia úplne inú vec a s inými dátami
  - ...o ktorých tvorca triedy `Book` nemá poňatia...



Prekrývanie metód

```
public class SmartTurtle extends Turtle
```



Prekrývanie metód

- Tvorca triedy môže **prekryť** implementáciu **metódy zdedenej** z rodičovskej triedy
  - čo dedím, musím mať
  - aj keď dedím metódu, môžem ju „preprogramovať“
  - prekrytie = ak ktokoľvek zavolá metódu (cez akúkoľvek premennú referenčného typu) vykoná sa preprogramovaná implementácia metódy

- Prekrytie = `override`

Polymorfizmus

- Polymorfizmus (viactvarovosť)
  - Na objekte vieme volať metódy definované v jeho triede, alebo v jej predkoch
  - Ak trieda-potomok definuje rovnakú metódu ako trieda-predok nastáva prekrytie metódy
    - Metódu predka z objektu „nevidno“ – použije sa „nová“ metóda potomka
    - Rovnaká metóda = rovnaká signatúra metódy (rovnaký názov a rovnaký počet, poradie a typy parametrov)
- Tento mechanizmus je nezávislý od typu referencujúcej premennej
  - Objekt vie to, akej triedy je, a nie to, akého typu je premenná, ktorá ma uloženú referenciu naňho
  - Typ premennej určuje, čo vieme na referencovaných objektoch volať

Polymorfizmus

- Dopíšeme metódu `getLocation()` aj do triedy `Book`.
  - Premenné typu `Book` už vedia takúto metódu zavolať
  - Túto metódu však objekty tried `PrintedBook`, `EBook`, `AudioBook` nebudú používať, lebo použijú svoje metódy `getLocation()`, ktorými túto metódu prekryjú

```
public class Book {
...
  public String getLocation() {
     return ”Location unknown.”;
  }
...
}
```

Polymorfizmus filmov

- Každý objekt si zavolá `getLocation()` zo svojej triedy

```
public class Library {
...
    public void printAll() {
        for (int i = 0; i < books.length; i++) {
            System.out.println(books[i].toString());
            System.out.println(books[i].getLocation());
        }
    }
...
}
```

Polymorfizmus: Vykoná sa implementácia zodpovedajúca triede, ktorej ten objekt je inštanciou.

Problém vyriešený

- Ak by sme sa chceli predsa len dostať k pôvodnej metóde rodiča použijeme v metóde dieťaťa volanie cez **super**

```
public class AudioBook {
...
  public String getBookLocation() {
     return “book location:" + super.getLocation();
  }
...
}
```

Anotácie

- Informácia pre kompilátor *(.java -> .class)*
- @Override
  - Upozorní kompilátor, že ide o prekrývanie metódy - 'IntelliJ' kontroluje, či zodpovedá signatúra metódy

```
public class EBook {
  private String link;
...
  @Override
  public String getLocation() {
     return this.link;
  }
...
}
```

Polymorfizmus najhrubšieho zrna :)

- Čo keby sme chceli, aby `toString()` vrátil aj umiestnenie?
  - Ale veď `toString()` je v triede `Book` a nevidí na inštančné premenné tried `PrintedBook`, `EBook`, `AudioBook`
  - Zavoláme `getLocation()` v metóde `toString()` v triede `Book`
    - voláme `toString()` na objekte triedy potomka
    - objekt si teda zavolá svoju prekrytú metódu `getLocation()`

Polymorfizmus – novinka?

- Myšacie udalosti v JPAZe:
  - `onMousePressed` – len obyčajné **prekrytie** metódy z triedy `WinPane`, kde sme dali vlastnú implementáciu a vďaka polymorfizmu sa zavolal náš obslužný kód
- Záhadný `toString` a `System.out.println`:
  - metóda toString pochádza z triedy Object
    - má ju každý objekt
    - **jej prekrytím** implementujeme ako má vyzerať reťazcová (textová) reprezentácia objektov danej triedy
  - metódu `toString` využíva `System.out.println`, ale aj Java, keď pri zlepovaní reťazcov potrebuje referenciu prerobiť na reťazec

Ďakujem za pozornosť !