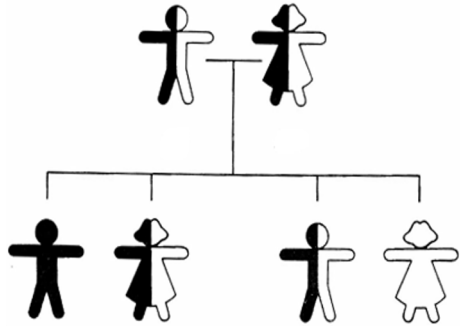




# 10. prednáška (23.11.2020)



## Dedičnosť'

## a

## polymorfizmus



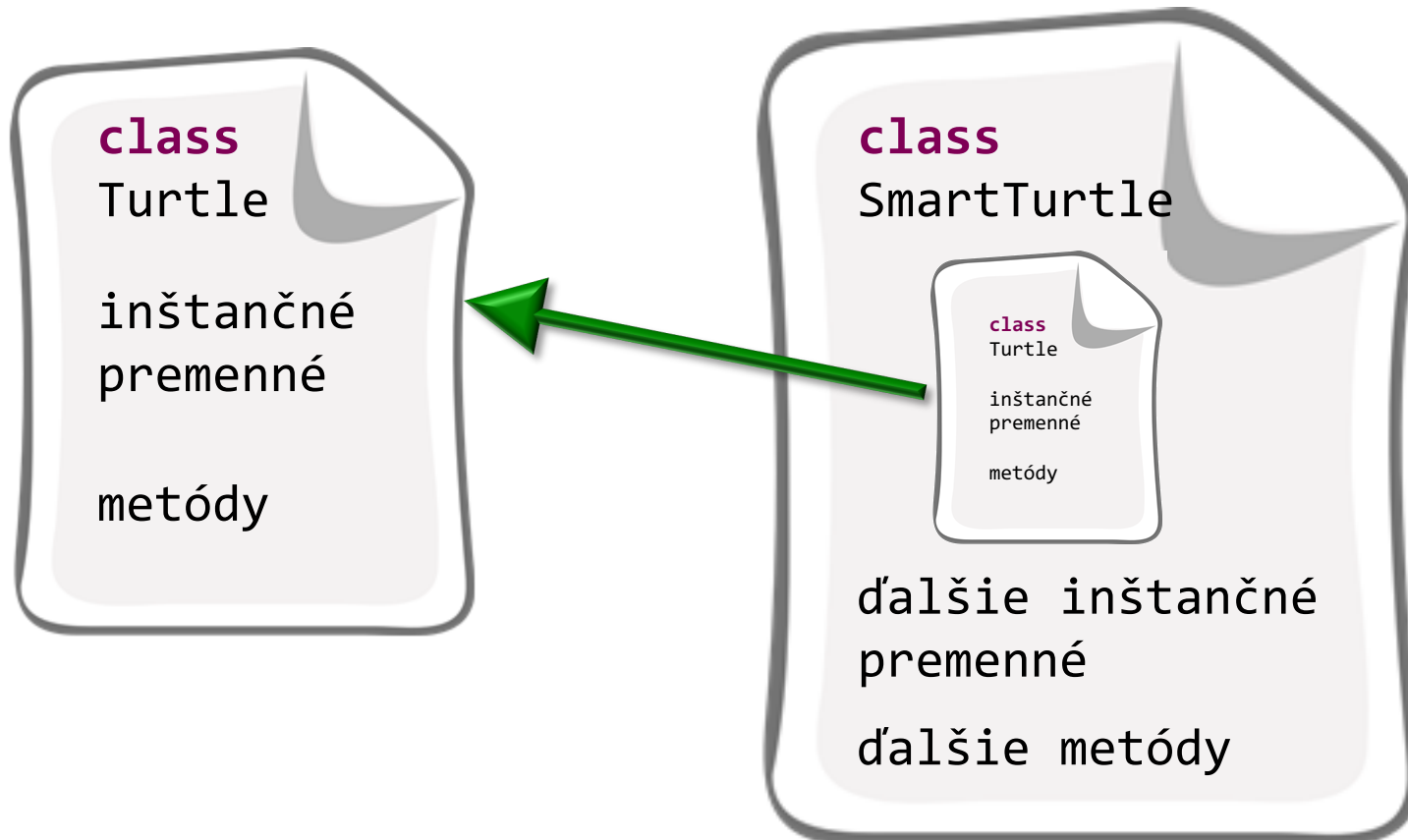
*New Rules!*





# 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
  - „knížnice“





# Zadanie (z minula)

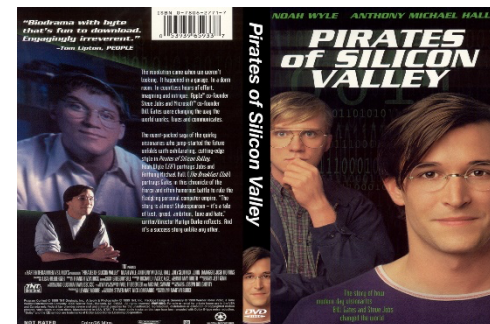
- Ciel': pohodlná správa zbierky filmov
- Vyžadovaná **funkcionalita**:
  - vieme vložit' info o novom filme
  - odstrániť film
  - vypísať všetky filmy v zbierke
  - vypísať tie filmy, ktoré zodpovedajú danému žánru (napr. komédie)
  - vypísať tie filmy, ktoré sa dajú pozrieť do nejakého času (napr. do 90 minút)
  - vypísať všetkých filmy, kde hral daný herec
  - vypísať filmy, ktoré sú podľa nášho hodnotenia na stupnici od 7 do 10.





# Zadanie (z minula)

- Dôležité informácie o každom filme:
  - názov filmu
  - mená hercov, ktorí v ňom hrali
  - žánre, do ktorých spadá
    - film môže mať viac žánrov (napr. "kriminálka a thriller" alebo "romantika, komédia a rodinný")
  - dĺžku filmu
  - hodnotenie kvality filmu: 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 funkcionality** má program (objekty triedy) mať
    - ... metódy





# Zapúzdenie (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 Movie {
```

```
    public Movie(String title) {  
        this(title, 0, 0);  
    }
```

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



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



# Rozširujeme zadanie

- rôzne umiestnenia filmov:
  - DVD, počítač, online ...





# Rozširujeme zadanie

- Na DVD
  - očíslované
- V súbore v počítači
  - názov počítača
  - cesta k súboru
  - veľkosť súboru
- V online priestore
  - URL





# Aké triedy?



**class**  
DvdMovie

inštančné  
premenné

metódy



**class**  
ComputerMovie

inštančné  
premenné

metódy



**class**  
OnlineMovie

inštančné  
premenné

metódy



# Veľa rozdielneho aj spoločného...

- Filmy na ľubovoľnom médiu majú niektoré **rozdielne dáta**:
  - identifikácia média
    - očíslovanie, URL
  - doplňujúce údaje
    - veľkosť súboru
- Filmy na ľubovoľnom médiu majú aj **rozdielne správanie funkčných schopností**:
  - výpis umiestnenia filmu
  - spôsob uloženia do súboru a načítania z neho
  - poskytovanie dodatočných informácií



# Veľa rozdielneho aj spoločného...

- Filmy na ľubovoľnom médiu majú ale aj **spoločné dáta**:
  - názov filmu
  - mená hercov, ktorí v ňom hrali
  - žánre, do ktorých spadá - predpokladáme, že film môže mať viac žánrov (napr. "kriminálka a thriller" alebo "romantika, komédia a rodinný")
  - dĺžku filmu
  - hodnotenie kvality filmu na stupnici od nula do desať.
- Filmy na ľubovoľnom médiu majú aj **spoločnú funkcionality**:
  - **String** getTitle()
  - **boolean** hasActor(String actorName)
  - **String** toString() - vypisuje (zatiaľ) len spoločné dáta



# Aké triedy?



```
class  
DvdMovie
```



```
class  
ComputerMovie
```



```
class  
OnlineMovie
```







# Nočná mora: Duplicita kódu

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



- Riešenie:
  - metódy
  - triedy
  - „knížnice“





# Aké triedy?

**class**  
Movie



**class**  
DvdMovie

**class**  
ComputerMovie

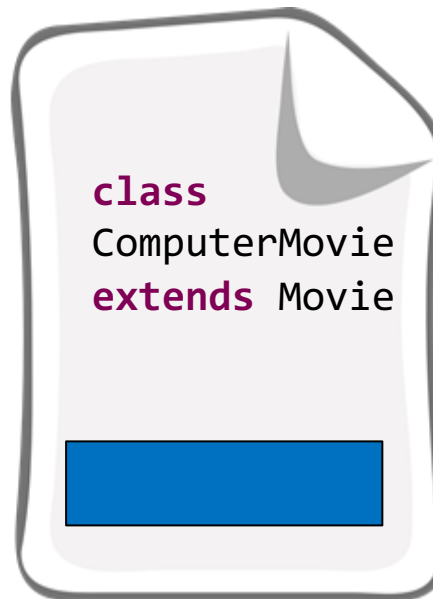
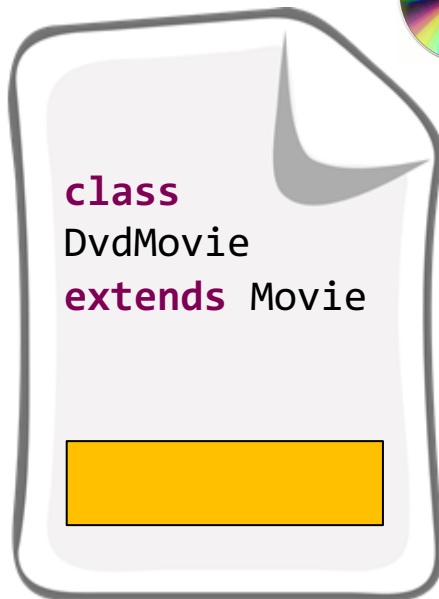
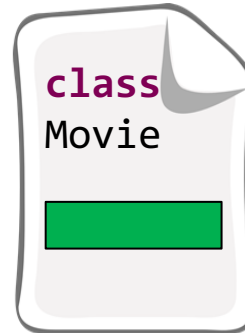


**class**  
OnlineMovie





# Eliminácia duplicity rozšírením





# *Dedičnosť = rozširovanie*

- Vytvoríme si triedu `Movie`, ktorá
  - obsahuje **spoločné dáta a metódy** pre všetky filmy bez ohľadu na médiá, na ktorých sú uložené
- Od nej oddedené triedy, t.j. triedy, ktoré rozširujú vlastnosti triedy `Movie` o:
  - `DvdMovie`
    - `dvdNumber` (ID)
  - `ComputerMovie`
    - `computerName`, `file`, `fileSize`
  - `OnlineMovie`
    - `url`

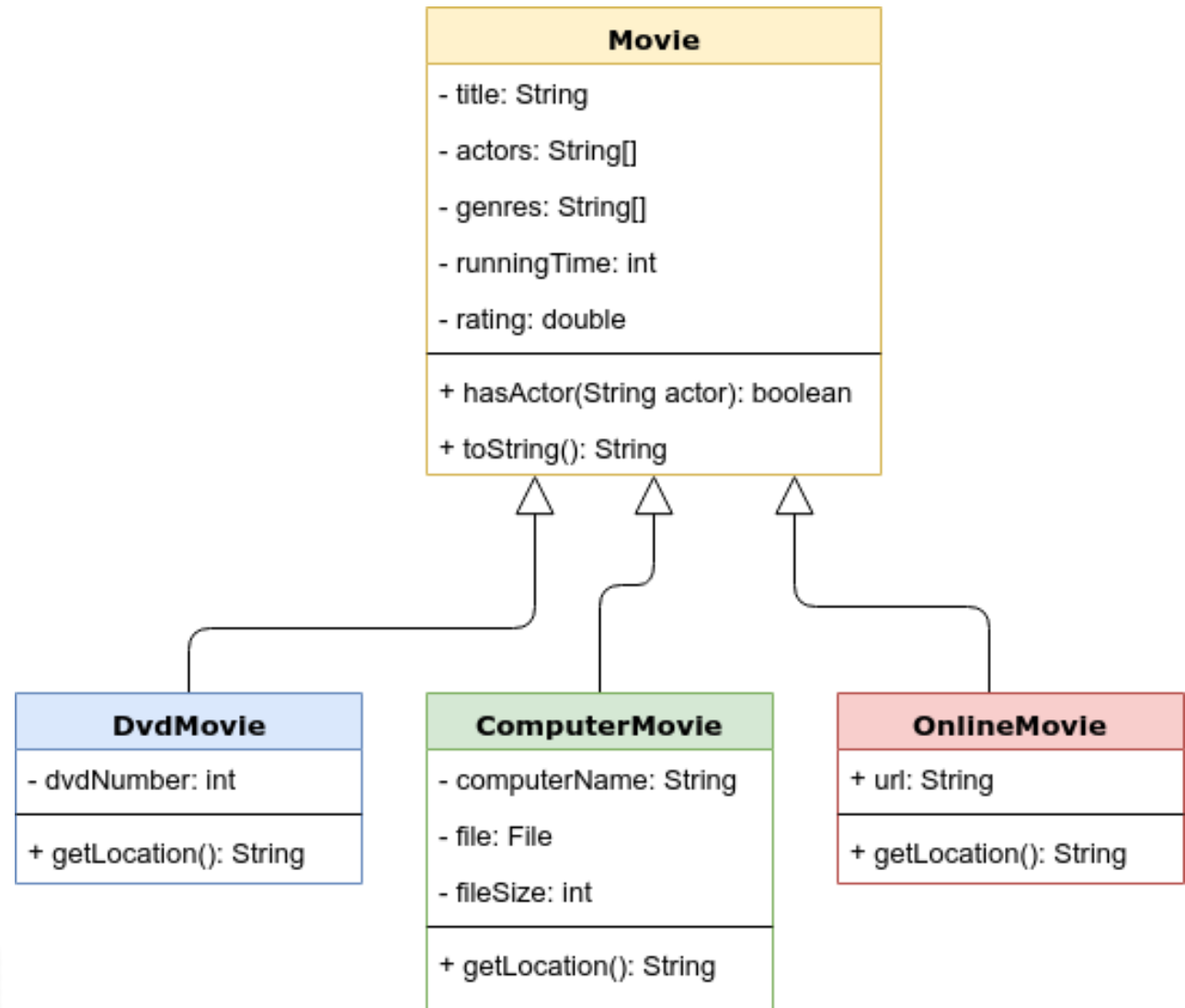


# *getLocation*

- Každý „filmový“ objekt vie povedať, kde sa nachádza
  - `String getLocation()`
- Každý to povie po svojom (nejde o kópiu)
  - `DvdMovie`
    - DVD číslo 34
  - `ComputerMovie`
    - Počítač FranklinPC v súbore `C:\filmy\forrest_gump.mp4`
  - `OnlineMovie`
    - URL: `https://www.youtube.com/watch?v=UuMnJ5yPRXc`



# Triedový diagram



Programujeme...



# 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 Movie {
```

```
    public Movie() {  
        super();  
    }
```

Takto by vyzeral  
implicitný konštruktor  
keby ho bolo vidieť

```
    ...  
}
```





# Minule...



Zmysel života?



Zoznam/správca filmov

```
public class MovieLibrary
```

MovieLibrary závisí od triedy Movie

```
public class Movie
```

Movie



# Dnes...



Zoznam/správca filmov

**public class** MovieLibrary  
čo a ako má uchovať?



**public class** Movie



**public class** OnlineMovie



**public class** DvdMovie



**public class** ComputerMovie



# Renovujeme zoznam filmov

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

```
public class MovieLibrary {  
    private DvdMovie[]      dvdMovies;  
    private ComputerMovie[] computerMovies;  
    private OnlineMovie[]  onlineMovies;  
    ...  
}
```



# Renovujeme zoznam filmov

- Prvý nápad:
  - Ale potom máme všade 3 cykly

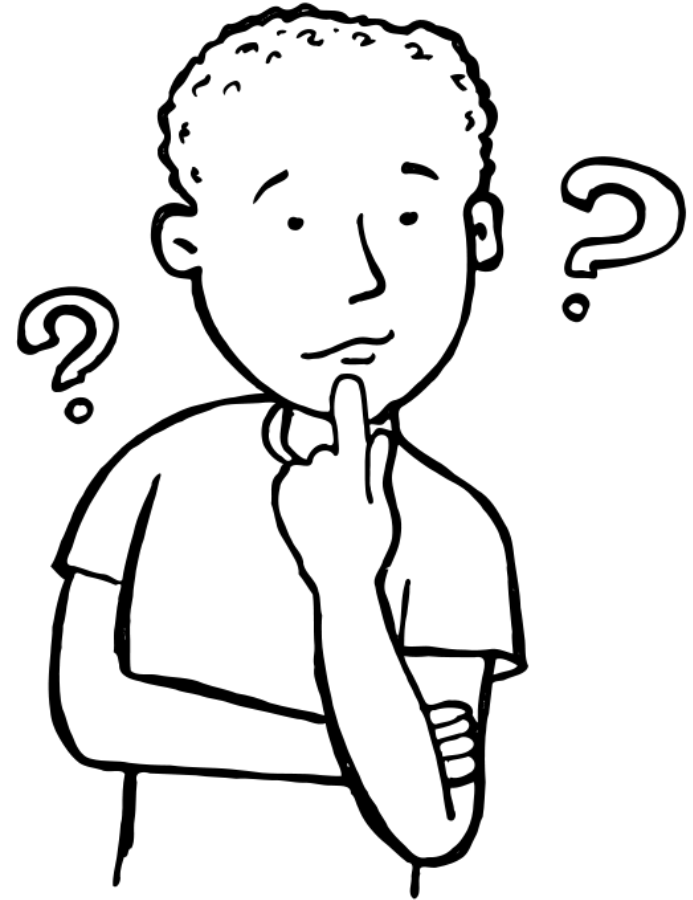


```
public class MovieLibrary {  
    ...  
    public void vypisVsetko() {  
        for (int i = 0; i < dvdMovies.length; i++)  
            System.out.println(dvdMovies[i].toString());  
        for (int i = 0; i < computerMovies.length; i++)  
            System.out.println(computerMovies[i].toString());  
        for (int i = 0; i < onlineMovies.length; i++)  
            System.out.println(onlineMovies[i].toString());  
    }  
    ...  
}
```



# Renovujeme zoznam filmov

- Filozofická úvaha:
  - DvdMovie je Movie
  - ComputerMovie je Movie
  - OnlineMovie je Movie





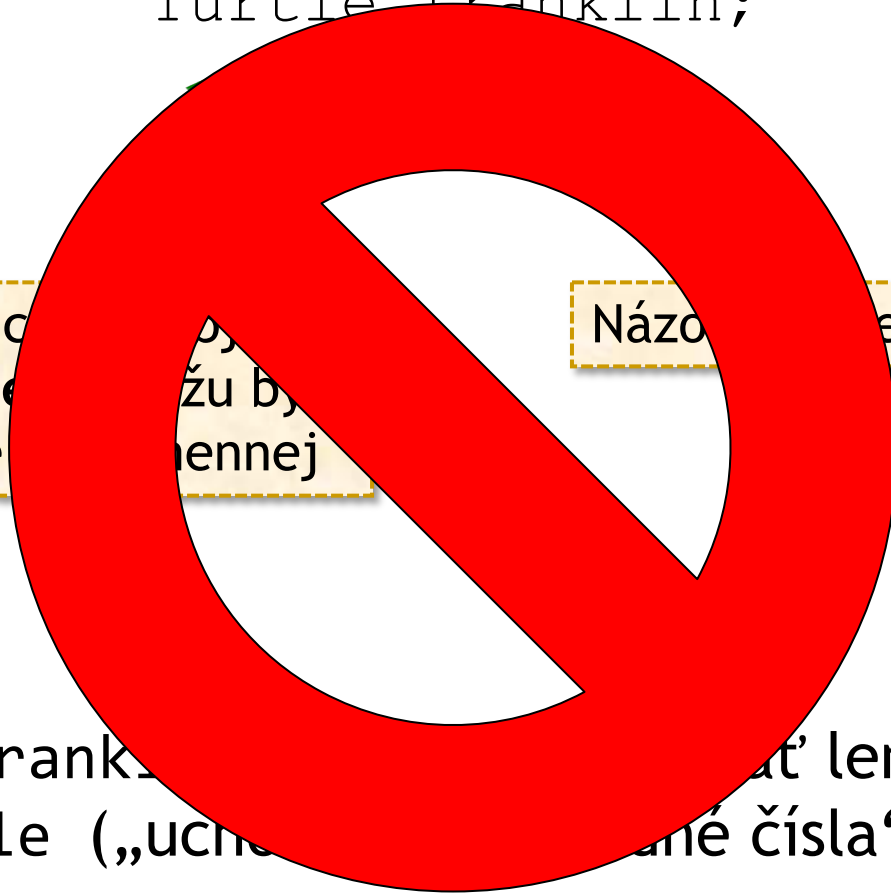
# Premenné referenčného typu

```
Turtle franklin;
```

Referencia  
akej triedy  
uložené

objektu  
žiu by  
ennej

Názov  
ennej



- Premenná franklin môže obsahovať len objekty triedy Turtle („učenie“ a „číslo“)
- 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 filmov

- Referencie typu `Movie` môžu uchovávať aj referencie na objekty tried `DvdMovie`, `ComputerMovie`, `OnlineMovie`
  - vieme pristupovať k metódam z triedy `Movie`
  - nevieme pristupovať k metóde `getLocation`

```
public class MovieLibrary {  
    private Movie[] movies;  
    ...  
}
```



# Renovujeme zoznam filmov

- Riešenie:

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

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





# Ale máme problém

## ● Problém:

- Keď všetko je film, ako zistíme umiestnenie?
- Nevieme predsa zavolať getLocation...

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

The method getLocation()  
is undefined for the type Movie



# Identifikujeme a pretypujeme

- Operátor **instanceof**:

- Vieme porovnať triedu objektu referencovaného z premennej

```
public class MovieLibrary {
```

```
...
```

```
    public void printAll() {
```

```
        for (int i = 0; i < movies.length; i++)
```

```
            System.out.println(movies[i].toString());
```

```
            if (movies[i] instanceof DvdMovie) {
```

```
                DvdMovie dvd = (DvdMovie) movies[i];
```

```
                System.out.println(dvd.getLocation());
```

```
            }
```

```
            if (movies[i] instanceof OnlineMovie) {
```

```
                OnlineMovie youtubeMovie = (OnlineMovie) movies[i];
```

```
                System.out.println(youtubeMovie.getLocation());
```

```
            }
```

```
        ...
```

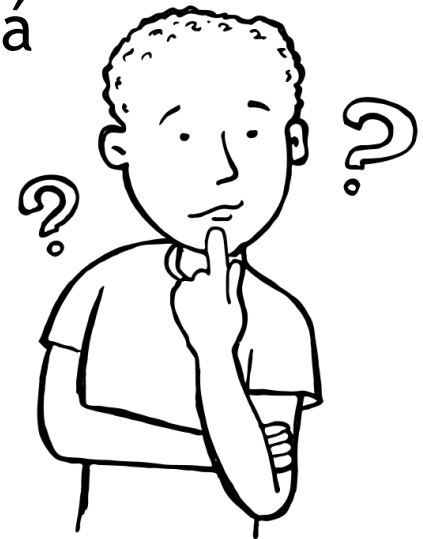
```
    }
```

```
}
```



# Renovujeme zoznam filmov

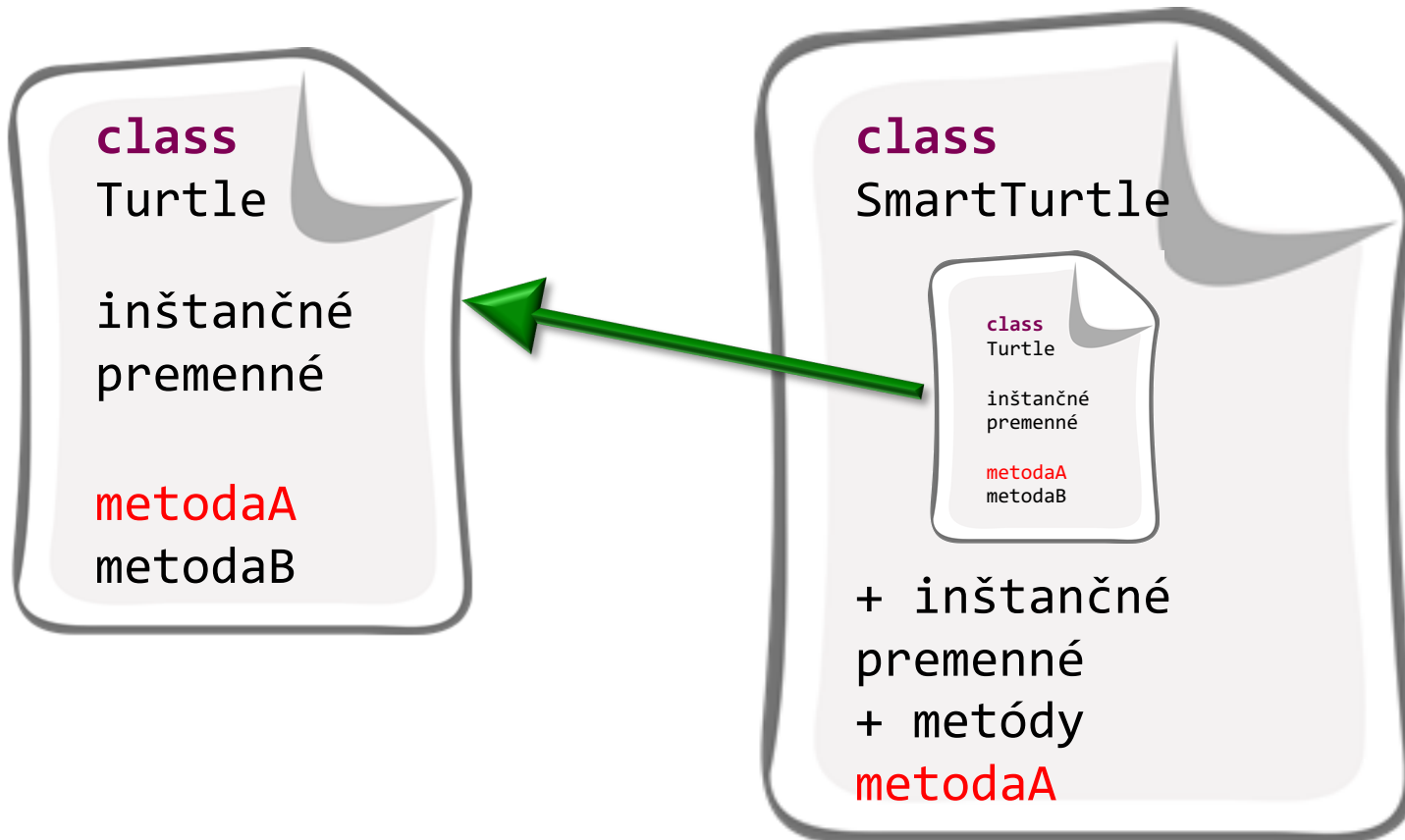
- Každý film má nejaké umiestenie
  - pridáme metódu getLocation do triedy Movie?
- DvdMovie, ComputerMovie, OnlineMovie v rámci metódy getLocation spravujú úplne inú vec a s inými dátami
  - ...o ktorých tvorca triedy Movie nemá poňatia...





# Prekrývavanie metód

```
public class SmartTurtle extends Turtle
```





# Prekrývanie metód

- Tvorca triedy môže **prekryt'** 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 (viactvarovost')
  - Na objekte vieme volat' 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 volat'





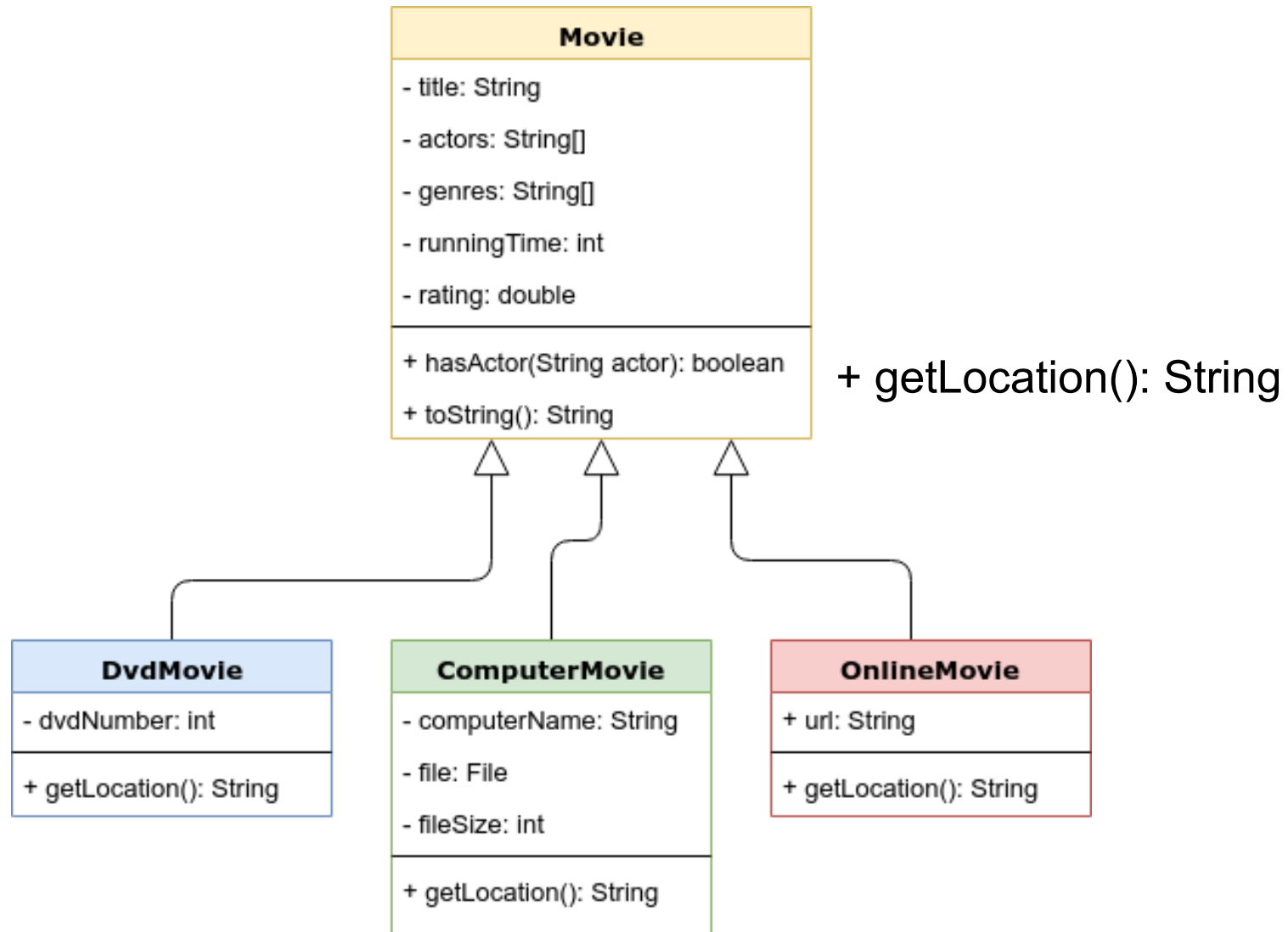
# Polymorfizmus

- Dopíšeme metódu `getLocation()` aj do triedy `Movie`.
  - Premenné typu `Movie` už vedia takúto metódu zavolať
  - Túto metódu však objekty tried `DvdMovie`, `ComputerMovie` a `OnlineMovie` nebudú používať, lebo použijú svoje metódy `getLocation()`, ktorými túto metódu prekryjú

```
public class Movie {  
    ...  
    public String getLocation() {  
        return "Location unknown.";  
    }  
    ...  
}
```



# Triedový diagram






# Polymorfizmus filmov

- Každý objekt si zavolá getLocation() zo svojej triedy

```
public class MovieLibrary {  
    ...  
    public void printAll() {  
        for (int i = 0; i < movies.length; i++) {  
            System.out.println(movies[i].toString());  
            System.out.println(movies[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 DvdMovie {  
    ...  
    public String getMovieLocation() {  
        return "movie location:" + super.getLocation();  
    }  
    ...  
}
```



# Anotácie

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

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



# Polymorfizmus najhrubšieho zrna :)

- Čo keby sme chceli, aby `toString()` vrátil aj umiestnenie?
  - Ale ved' `toString()` je v triede `Movie` a nevidí na inštančné premenné tried `DvdMovie`, `ComputerMovie` a `OnlineMovie`
  - Zavoláme `getLocation()` v metóde `toString()` v triede `Movie`
    - 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 zavolať náš obslužný kód
- Záhadný toString a System.out.println:
  - metóda toString pochádza z triedy Object
    - má ju každý objekt
    - **jej prekrytím** implementujeme ako má vyzerat' 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 !**

