Objekt. Trieda. Enkapsulácia. Konvencie. Atribút. Metóda. Dedenie. Konštruktor. Overloading. Overriding. Polymorfizmus. Abstrakcia.

OLIVER UDVARDI FEBRUÁR 2026

Testovač · Odovzdávanie úloh

Riešenie problémov

Riešenie problémov

1. Napíšte svojmu cvičiacemu

Riešenie problémov

1. Napíšte svojmu cvičiacemu
2. Ak sa problém nepodarí vyriešiť - napíšte prednášajúcemu

Riešenie problémov

1. Napíšte svojmu cvičiacemu
2. Ak sa problém nepodarí vyriešiť - napíšte prednášajúcemu
3. Ak sa problém nepodarí vyriešiť - napíšte garantovi

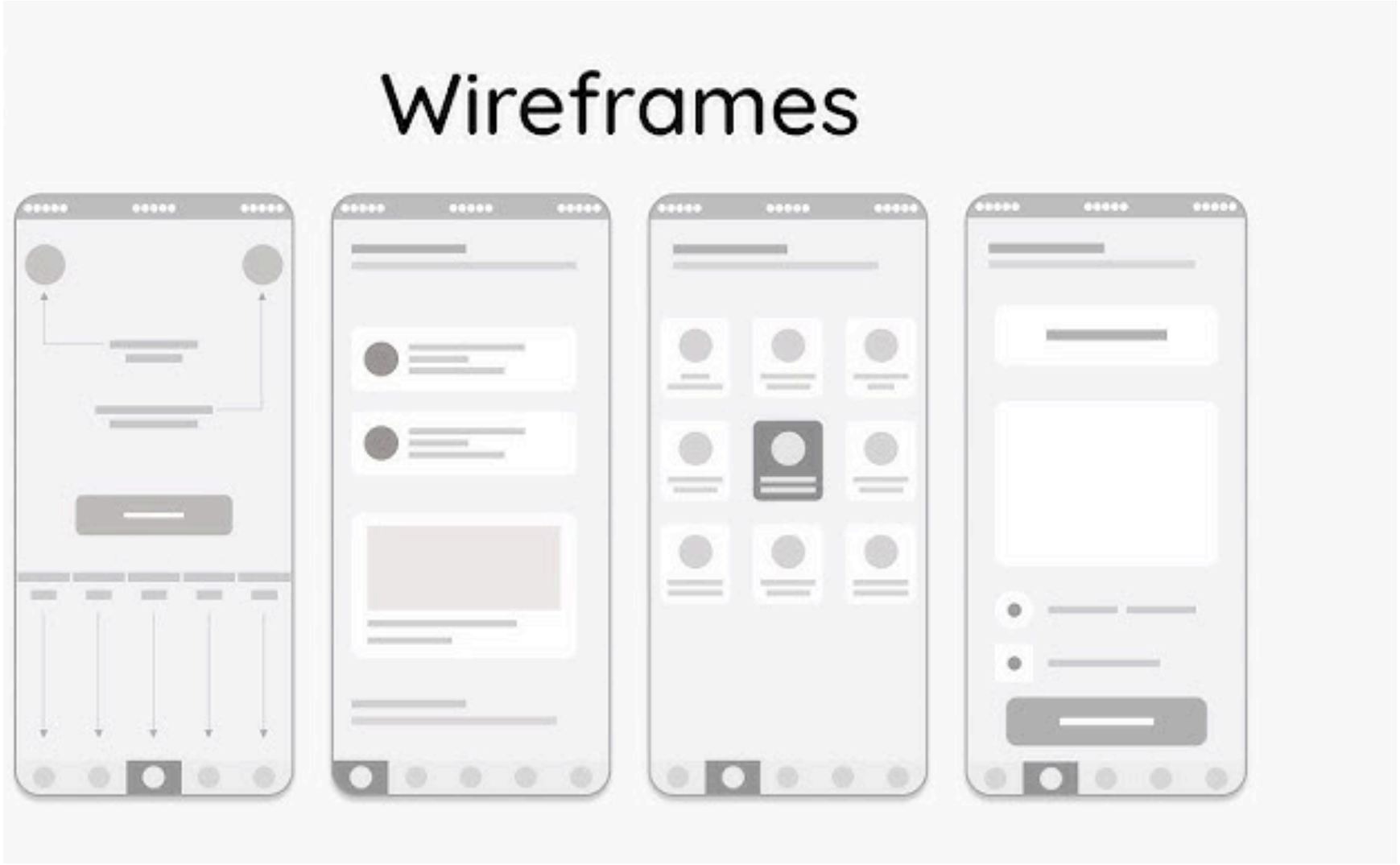
Zadanie 1
Špecifikácia

- Popísať to, čo idete robiť.
- Prečo to robíme?
- Ako to urobiť?

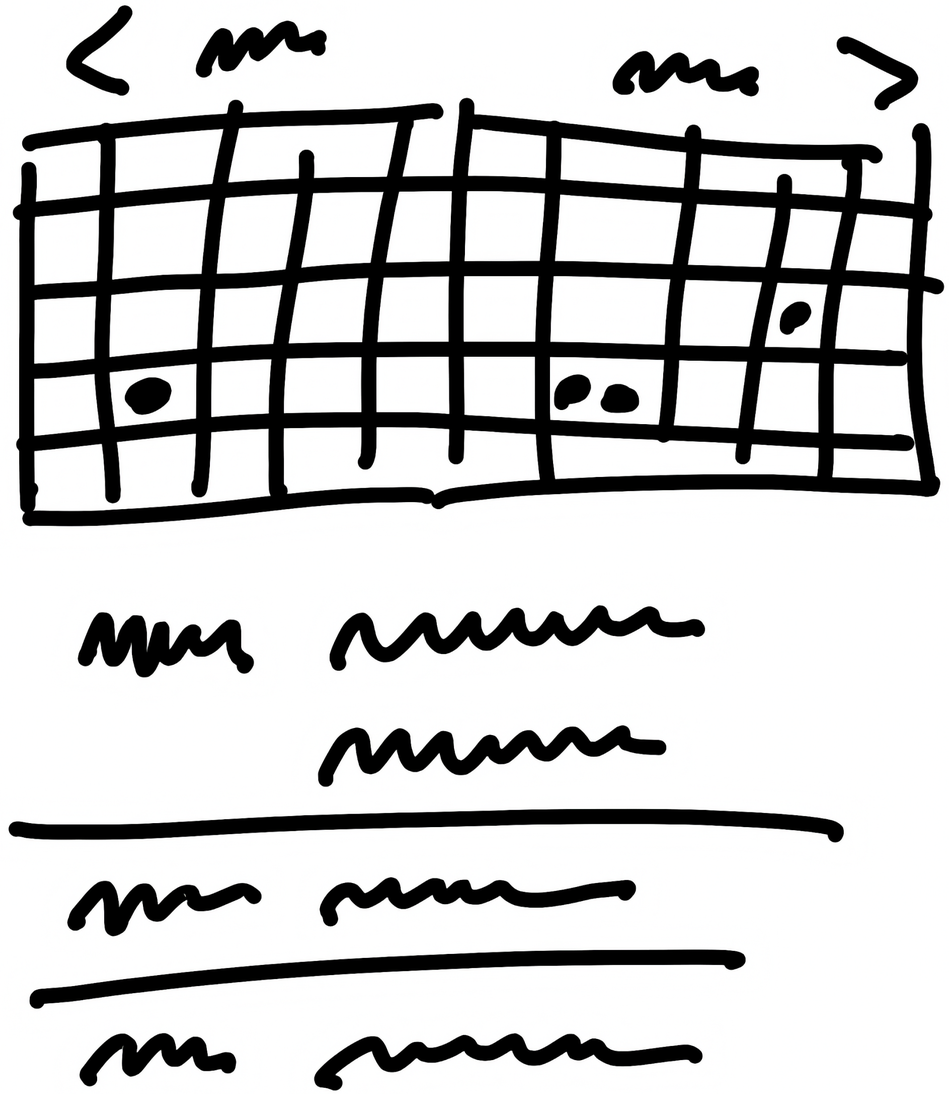
Slová

- *Budem mať úvodnú obrazovku, kde budú tlačidlá Play, Settings a Exit. Bude tam v hornom rohu tlačidlo na vypnutie zvuku. Keď používateľ klikne na Play, tak sa ho to spýta na meno a potom môže začať hrať.*

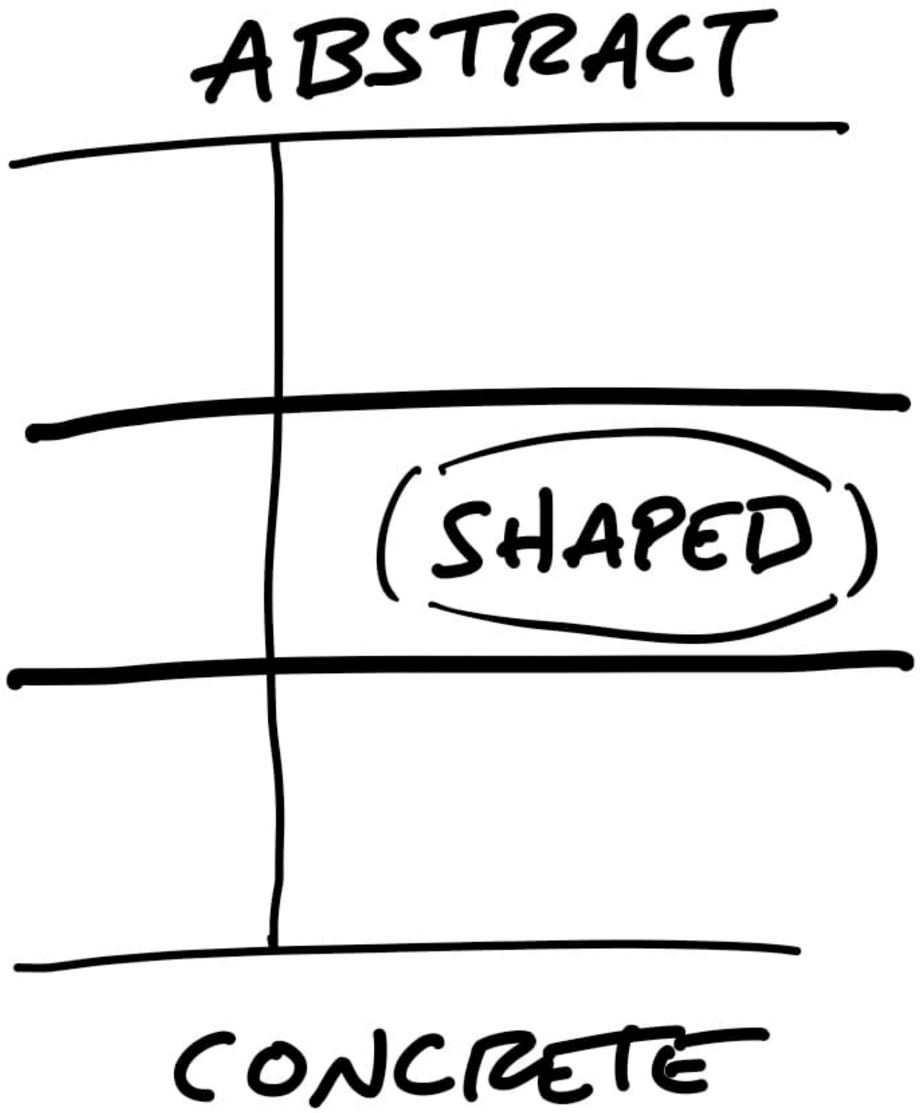
Wireframe



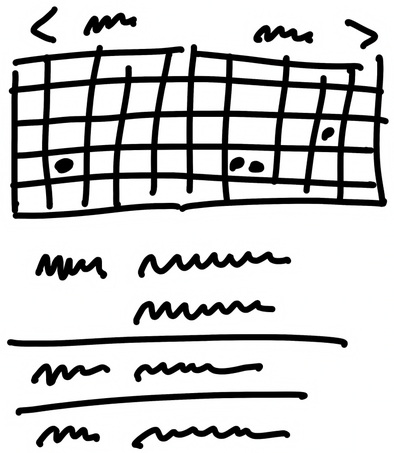
Shaping

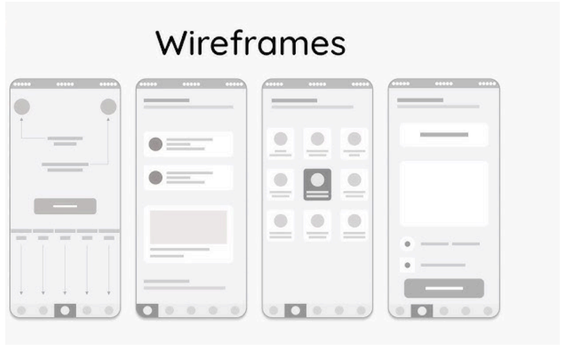


Shape up



Slová





Previously.
Objekt. Trieda. Enkapsulácia.
Konvencie. Atribút. Metóda.

OOP princípy #2
Dedenie. Konštruktor. Overloading. Overriding. Polymorfizmus. Abstrakcia.

Dedenie

- Prevzatie Atribútov a Metód medzi dvoma triedami, ktoré sú v **hierarchickom** vzťahu
- **Podtrieda (subclass) preberá atribúty a metódy nadtriedy (superclass).**
- Atribúty, Metódy
- Každý object dedí od Triedy Object
  - https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Stolička

- počet Nôh, farba, materiál

Stolička • počet Nôh, farba, materiál

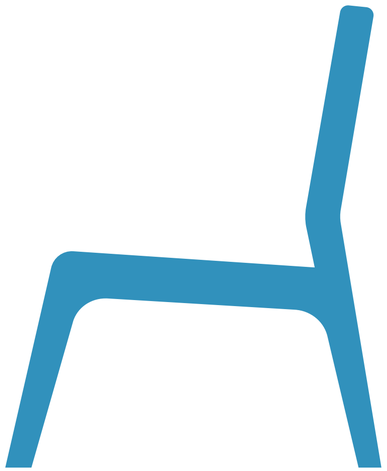
Stolička • početNôh, farba, materiál

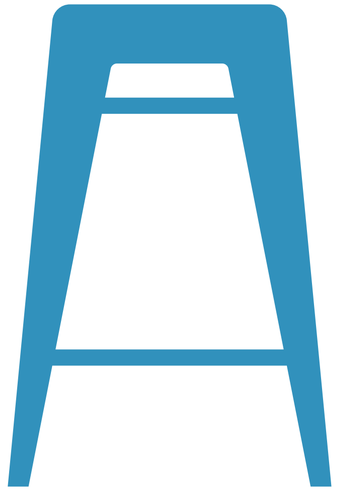
Stolička • počet Nôh, farba, materiál

Stolička

- počet Nôh, farba, materiál

nastavVýšku()





Konštruktor

- V reálnom svete je potrebné veci okolo nás *vyrobiť/vytvoriť/vydolovať...*
- Vyrobiť, konkrétnym postupom, receptom, návodom. Skonštruovať.
- iPhone potrebujete poskladať v továrni a otestovať, či ho zapnete
- Knižnicu poskladáte podľa obrázkového návodu
- Brownies upečiete podľa receptu

Konštruktor 2

- Akonáhle niečo skonštuujete, už to nepotrebujete robiť znova. *Nemôžete upiecť koláč dvakrát*.
- Konštruktor je špeciálny člen triedy bez návratového typu, ktorý **má rovnaký názov ako trieda** a vykoná sa automaticky pri vytvorení objektu za účelom jeho inicializácie.
- Ak žiadny konštruktor nešpecifikujete, bude definovaný implicitne. Prázdny konštruktor

- class Brownies extends Cake {

```
  public Brownies(){
    //Overenie atribútov, uloženie referencii...
    System.out.println("Brownies baked");
  }
}
```

Čo sa stane?

Čo sa stane?

- Ak máme `Brownies extends Cake`

Čo sa stane?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`

Čo sa stane?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:

Čo sa stane?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies

Čo sa stane?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies
- b) len Cake

Čo sa stane?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies
- b) len Cake
- c) najskôr Brownies potom Cake

Čo sa stane?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies
- b) len Cake
- c) najskôr Brownies potom Cake
- d) najskôr Cake potom Brownies

Čo sa vypíše?

Čo sa vypíše?

- Ak máme `Brownies extends Cake`

Čo sa vypíše?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`

Čo sa vypíše?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:

Čo sa vypíše?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies

Čo sa vypíše?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies
- b) len Cake

Čo sa vypíše?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies
- b) len Cake
- c) najskôr Brownies potom Cake

Čo sa vypíše?

- Ak máme `Brownies extends Cake`
- Zavoláme `Brownies b = new Brownies();`
- Zavolá sa konštruktor:
- a) len Brownies
- b) len Cake
- c) najskôr Brownies potom Cake
- **d) najskôr Cake potom Brownies**

Overloading

- Preťaženie metódy. Voláme to isté správanie, ale meníme vstupy resp. parametre.

```
public void eat() {
    System.out.println("I am eating!");
}

public String eat(String food) {
    return "I am eating " + food + "!";
}

public String eat(String food, int count) {
    return "I am eating " + food + " " + count + " times!";
}
```

Overriding · Prekonanie (override) metódy je mechanizmus, pri ktorom podtrieda poskytne vlastnú implementáciu metódy zdedenej z nadtriedy, pričom zachová jej názov, parametre a návratový typ.

Polymorfizmus

Polymorfizmus

Člověk

Polymorfizmus

Študent

Človek

Polymorfizmus

Študent

Hasič

Človek

Polymorfizmus

Študent

Hasič

Je každý študent aj človek?

Človek

Polymorfizmus

Študent

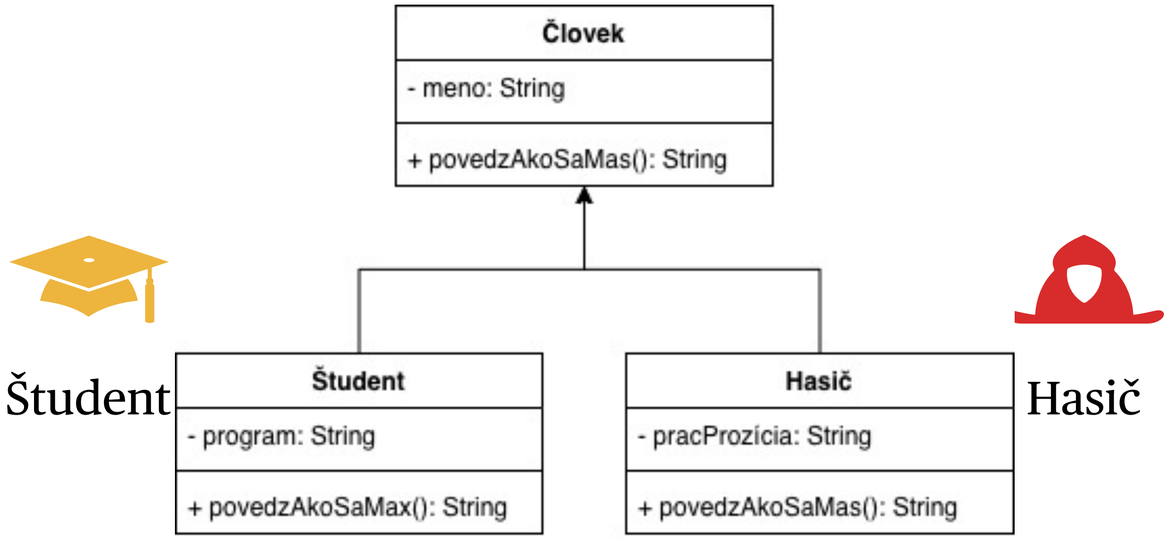
Hasič

Je každý študent aj človek?

Je každý hasič aj človek?

Človek

Polymorfizmus

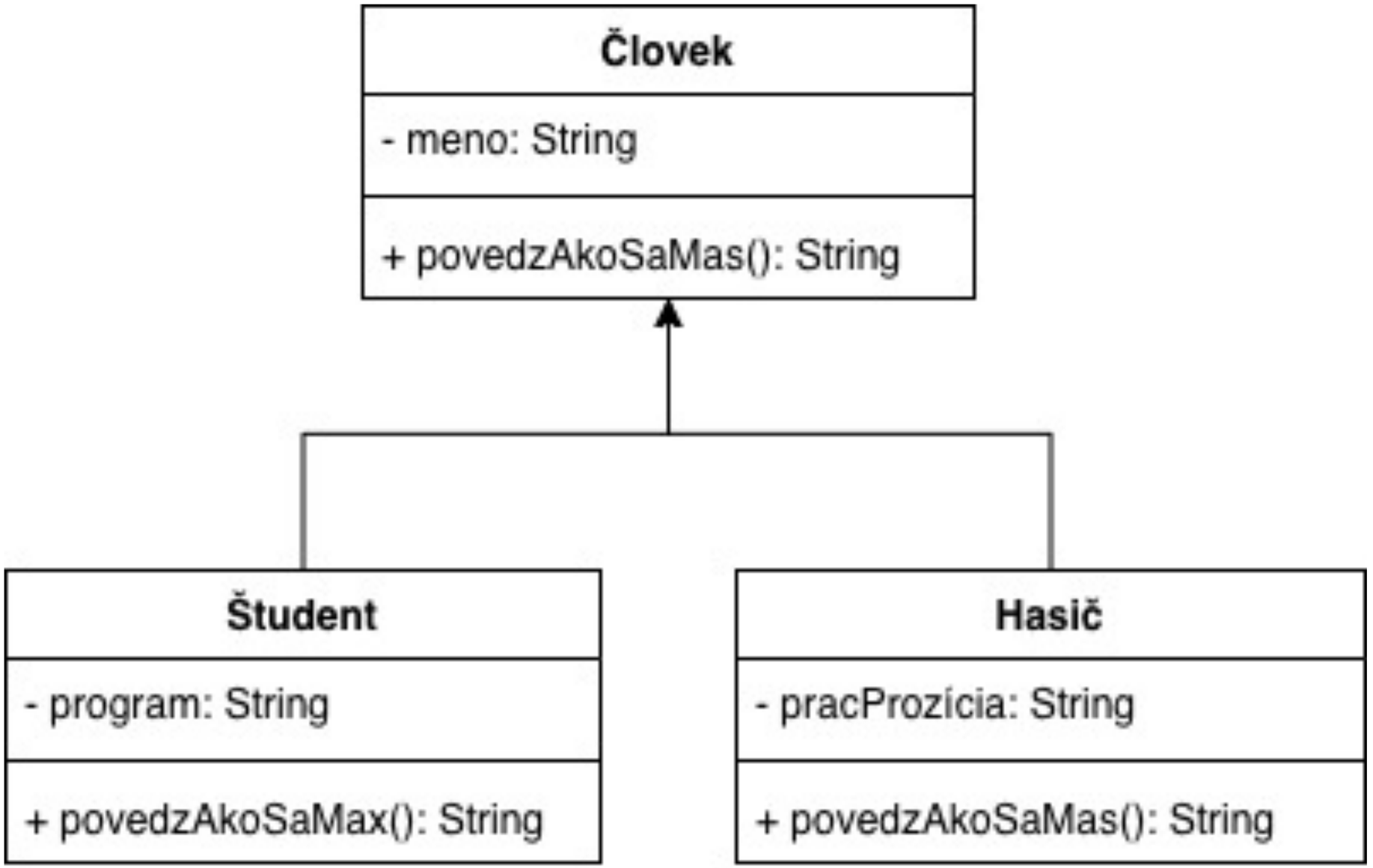


Je každý študent aj človek?

Je každý hasič aj človek?

Človek

UML


Človek
- meno: String
+ povedzAkoSaMas(): String
Študent
- program: String
+ povedzAkoSaMax(): String
Hasič
- pracProzícia: String
+ povedzAkoSaMas(): String

Compile-time

Polymorfizmus

```
class Clovek {
    void pracuj() {
        System.out.println("Človek pracuje");
    }
}

class Student extends Clovek {
    @Override
    void pracuj() {
        System.out.println("Študent sa učí");
    }
}

class Hasic extends Clovek {
    @Override
    void pracuj() {
        System.out.println("Hasič hasí požiar");
    }
}
```

Run-time

polymorfizmus

```
public static void main(String[] args) {

    Clovek c1 = new Student();
    Clovek c2 = new Hasic();

    c1.pracuj();  // Študent sa učí
    c2.pracuj();  // Hasič hasí požiar
}
```

Polymorfizmus

Polymorfizmus umožňuje, aby referencia nadtriedy ukazovala na objekty rôznych podtried a aby sa vykonala správna implementácia metódy podľa **skutočného typu objektu.**

Abstrakcia

Abstrakcia je zameranie sa na podstatné vlastnosti objektu a ignorovanie nepodstatných detailov.

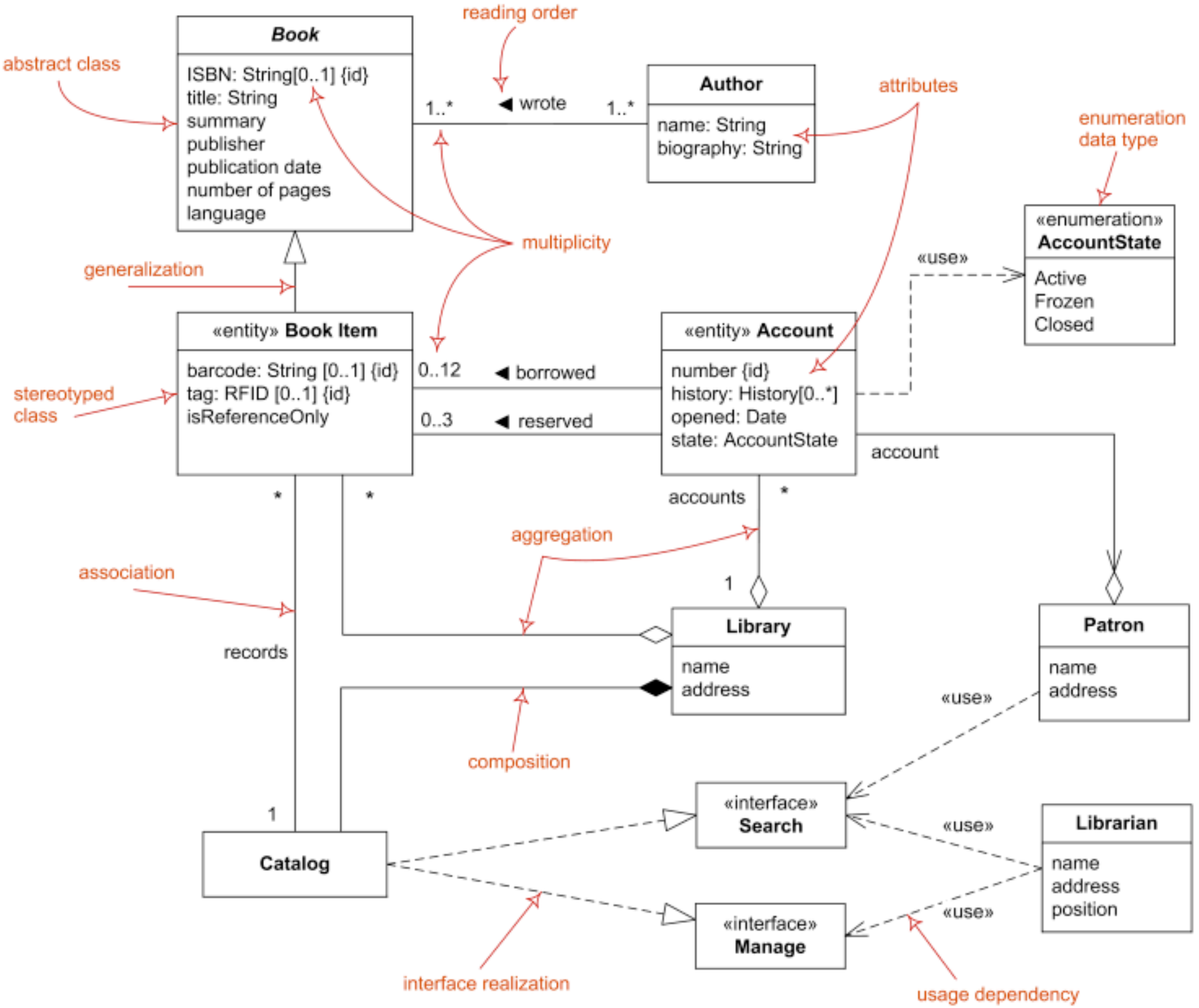
Abstrakcia

Skrýva zložitosť systému

Enkapsulácia

Skrýva vnútorný stav objektu

Class diagram



Quiz time!

OOP - kvíz #1