

Ruby and OOP

Šodien

- Pieskarsimies OOP ar Ruby
- Izveidosim mūsu pirmās klases
- Pabeigsim vakardienas uzdevumus

Sagatavošanās

- levelkam jaunākās izmaiņas jūsu fork
- CMD -> Ubuntu -> cd code/spv-prog-II (Linux logs)
 - git pull
 - git checkout main
 - git checkout -b task/SPV-2/basic-oop-classes
- Palaižam Docker Desktop
- CMD -> Ubuntu -> cd code/spv-prog-II -> bin/spv up
- CMD -> Ubuntu -> cd code/spv-prog-II -> bin/spv console

Izmantojam linux konsoli, lai atvēru VSCode

code .

vai

code spv-prog-ll

Atkarībā no tā kurā direktorijā atrodies

Class Car izveidošana

- Definē klasi ar nosaukumu Car
- Klases konstruktors saņem 4 *pozicionālos parametrus*:
 - Ražotājs
 - Modelis
 - Krāsa
 - Izlaiduma gads
- Konstruktorā saglabā šo parametrus kā *instances mainīgos*

```
1  ✓ class Car
2  ✓   def initialize(manufacturer, model, color, year)
3      @manufacturer = manufacturer
4      @model = model
5      @color = color
6      @year = year
7   end
8 end
```

```
my_car = Car.new("Volvo", "S60", "Blue", 2012)
```

Pārveido pozicionālos argumentus par nosauktajiem (named arguments)

Named arguments:

Pros: viegli saprast kas jānorāda, var mainīt vietām, var piešķirt defaultās vērtības

Cons: Klases instances izveidošana sanāk gara, var sanākt atkārtoties, ja padod mainīgos kā argumentus.

```
1  ✓ class Car
2  ✓   def initialize(manufacturer:, model:, color:, year:)
3      @manufacturer = manufacturer
4      @model = model
5      @color = color
6      @year = year
7   end
8 end
```

```
my_car = Car.new(manufacturer:"Volvo", model:"S60", color:"Blue", year:2012)
```


Instances mainīgie ir privāti – pieejami tikai klases instances (objekta) iekšienē.

Getteri nodrošina instances mainīgo izvadīšanu uz āru

Getteris ir publiska metode, kas atgriež kādu no instances mainīgajiem

Izveido visiem instances mainīgajiem getterus

Piemērs:

```
9  def manufacturer
10  |  return @manufacturer
11  end
```

```
9  ✓ def manufacturer
10  |  @manufacturer
11  end
```

Return var
nelietot,
metode
vienmēr atgriež
pēdējās rindīņas
vērtību

Ruby piedāvā saīsinātu pierakstu getteriem, lai nebūtu jāraksta daudz vienādas metodes

```
8  
9 attr_reader :manufacturer, :model, :color, :year  
10
```

Atkrārībā no tā ko apraksta instances
mainīgais, tam varētu mainīties vērtība.

Piemēram, mašīnas gads nemainīsies, bet
krāsa varētu mainīties.

Metodes, kas nodrošina instances mainīgo nomaiņu sauc par setteriem.

Seteriem ir pieņemt konkrēts pieraksts

```
def variable=(value)  
  @variable = value  
end
```

Izveido instances mainīgajam *color* setteri

```
10  
11 def color=(new_color)  
12   | @color = new_color  
13   end  
14
```

Tā pat kā getteriem, tā arī setteriem ir pieejams saīsinājums.

```
9 attr_reader :manufacturer, :model, :color, :year  
10 attr_writer :color  
11
```

Instances mainīgajam @color ir gan setteris, gan getteris. Tos iespējams apvienot vienā saīsinājumā attr_accessor

```
9 attr_reader :manufacturer, :model, :color, :year
10 attr_writer :color
11
```

```
8
9 attr_reader :manufacturer, :model, :year
10 attr_accessor :color
11
```


Irb konsolē izmēģini veikt šādas darbības

- Izveido **Car** klases objektu ar nosaukumu **my_car**
- Laikam ejot arī mašina ir pārkrāsota, pamēģini **my_car** objektam nomainīt krāsu uz ko citu.
- Mainīgajā **manufacturing_year** saglabā **my_car** **year** atribūtu
- Izvedo mainīgo **current_year** un piešķir tam atbilstošu vērtību. Nevis ~~**current_year = 2023**~~, bet iegūlē – “**ruby get current year**”
- Aprēķini **car_age**

```
irb(main):006:0> my_car
=> #<Car:0x00007f4dbf290108 @color="Blue", @manufacturer="Volvo", @model="S60", @year=2012>
irb(main):007:0> my_car.color
=> "Blue"
irb(main):008:0> my_car.color = "red"
=> "red"
irb(main):009:0> my_car.color
=> "red"
irb(main):010:0> my_car
=> #<Car:0x00007f4dbf290108 @color="red", @manufacturer="Volvo", @model="S60", @year=2012>
irb(main):011:0> |
```

```
root@f0263f26e1f3:/ruby/app# irb
irb(main):001:0> my_car = Car.new(manufacturer:"Volvo", model:"S60", color:"Blue", year:2012)
=> #<Car:0x00007f4dbf290108 @color="Blue", @manufacturer="Volvo", @model="S60", @year=2012>
irb(main):002:0> my_car_year = my_car.year
=> 2012
irb(main):003:0> current_year = Time.now.year
=> 2023
irb(main):004:0> car_age = current_year - my_car_year
=> 11
irb(main):005:0> |
```

Tikko mēs ārpus klases izrēķinājām mašīnas vecumu.

Mašīnas vecuma aprēķināšana varētu būt bieži izmantota funkcionalitāte, būtu vērts to ievietot klasē kā metodi

Izveido Car klasei metodi `get_age`, kas aprēķina konkrētās mašīnas vecumu. (izmanto iepriekš izmantoto loģiku)

```
11  
12   def get_age  
13     Time.now.year - year  
14   end  
15 end
```

```
irb(main):001:0> my_car = Car.new(manufacturer:"Volvo", model:"S60", color:"Blue", year:2012)  
=> #<Car:0x00007f06ed247d18 @color="Blue", @manufacturer="Volvo", @model="S60", @year=2012>  
irb(main):002:0> my_car.get_age  
=> 11  
irb(main):003:0> |
```

Šobrīd mēs ražotāju norādam kā vienkāršu String vērtību.

Tavs uzdevums ir uztaisīt jaunu klasi **Manufacturer**

- Izveido jaunu failu manufacturer.rb
- Nodefinē klasi ar nosaukumu Manufacturer
- Konstruktorā norādi šādus atribūtus:
 - name
 - country
- Izveido getterus visiem minētajiem atribūtiem

```
1  class Manufacturer
2
3      def initialize(name:, country:)
4          @name = name
5          @country = country
6      end
7
8      attr_reader :name, :country
9  end
10
```

Rodas iespēja mašīnām ražotāju norādīt nevis kā vienkārši string, bet gan konkrētu ražotāju – Manufacturer klases objektu

```
irb(main):001:0> volvo = Manufacturer.new(name: "Volvo", country: "Sweden")  
=> #<Manufacturer:0x00007f3ba0752d70 @country="Sweden", @name="Volvo">  
irb(main):002:0> my_car = Car.new(manufacturer: volvo, model: "S60", color: "Blue", year: 2012)
```

Attiecīgi varam noskaidrot manas mašīnas ražotāju savienojot komandas

```
irb(main):003:0> my_car.manufacturer.name  
=> "Volvo"  
irb(main):004:0> |
```


Ruby valodā nav tipizācijas!

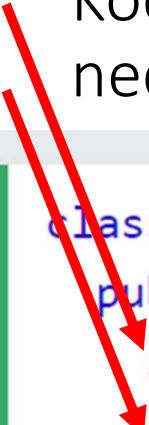
Jekurš mainīgais var būt jekāds objekts, jeb tips.

Piemēram, mūsu mašīnas ražotājs var būt gan klase, gan string, gan jebkas cits, kas tur tiks ievadīts.

Kāds ir šīs atribūta tips, mēs varēsim noskaidrot tika koda izpildes laikā

Pastāv tipizētas valods, piemēram, C++

Kodā tiek norādīts kads tips būs mainīgajam, nekas cits tur nedrīks un nevar būt!



```
class MyClass {           // The class
public:                   // Access specifier
    int myNum;            // Attribute (int variable)
    string myString;      // Attribute (string variable)
};
```

Uzdevums patstāvīgi

- Papildini kodubāzi ar klasi **Person**
 - Izveido klasi jaunā failā
 - Izveido atbilstošu inicializatoru un atribūtus
 - Iekļauj vismaz 3 atribūtus
- **Car** klasei pievieno atribūtus **passangers, max_passanger_count**
- Pievieno **Car** klasei metodes
 - **add_passanger**, kas pievieno pasažieri pasažieru sarakstā (atribūtā **passanger**)
 - **passanger_count**, kas atgriež pasažieru skaitu
 - **full?**, kas atgriež **true**, ja sasniegts maksimālais pasažieru skaits un **false**, ja nav
- **Pārbaudi** savu uzrakstīto kodu irb konsolē ar dažādiem testa scenārijiem

