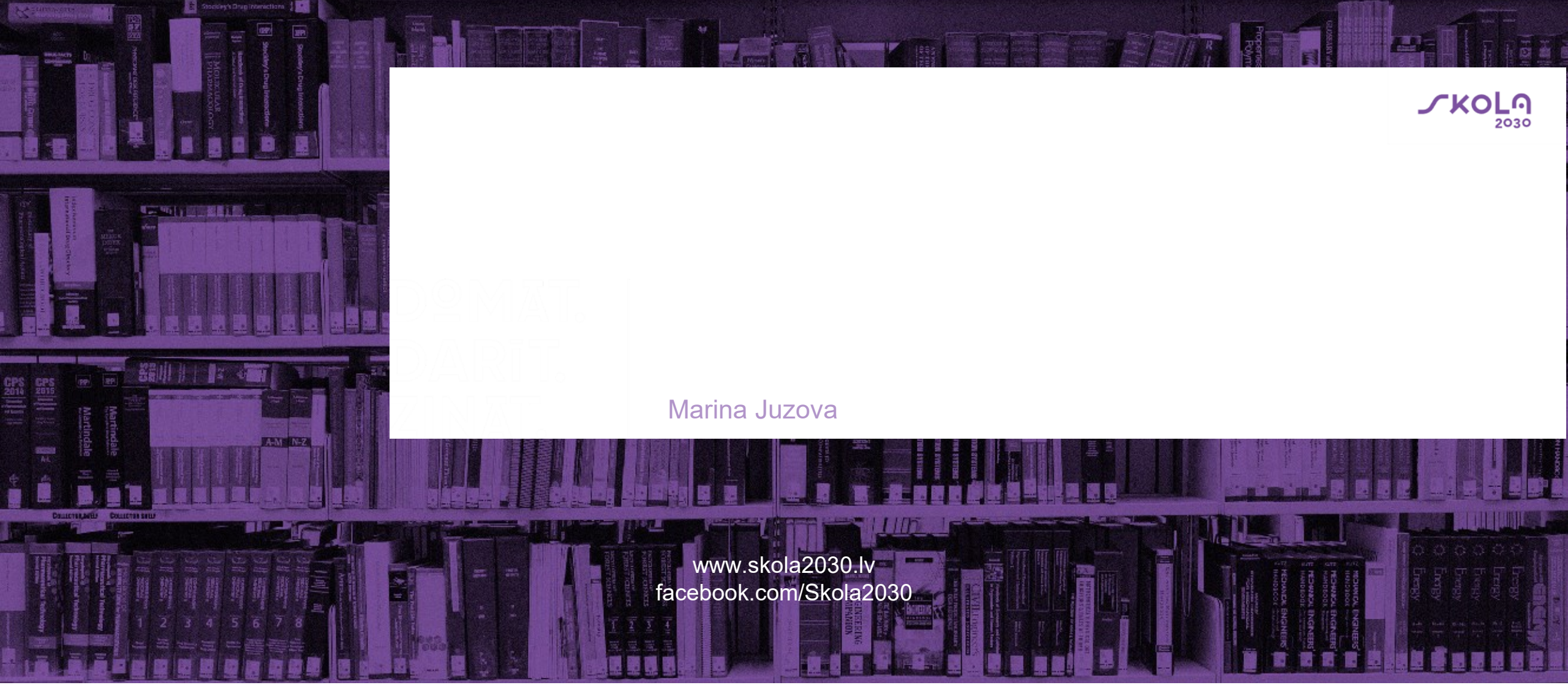


Marina Juzova

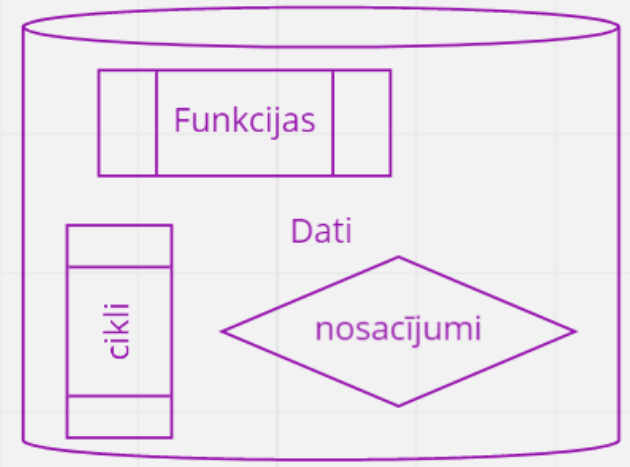
www.skola2030.lv
facebook.com/Skola2030

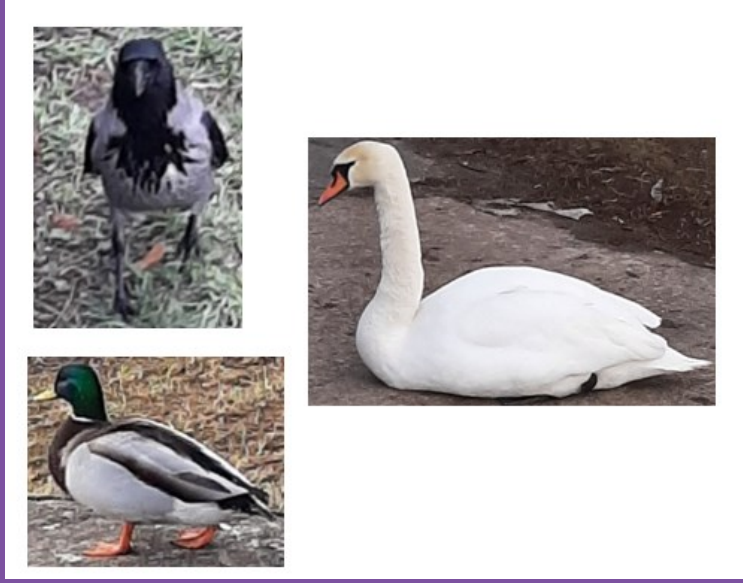
Projekts Nr. 8.3.1.1/16/I/002 Kompetenču pieeja mācību saturā

ObjektOrientētās Programmēšanas (OOP) pamatprincipi

Projekts Nr. 8.3.1.1/16/I/002 Kompetenču pieeja mācību saturā

Pirms OOP





a_01-funkcionaalais

```
1   def greeting (userName):
2     print (" Sveiki, "f"{userName}")
3
4   userName = input("Ievadi vārdu ")
5   greeting(userName)
```

Console Shell

```
Ievadi vārdu Marina
 Sveiki, Marina
>
```

Abstrakcija

-
-
-

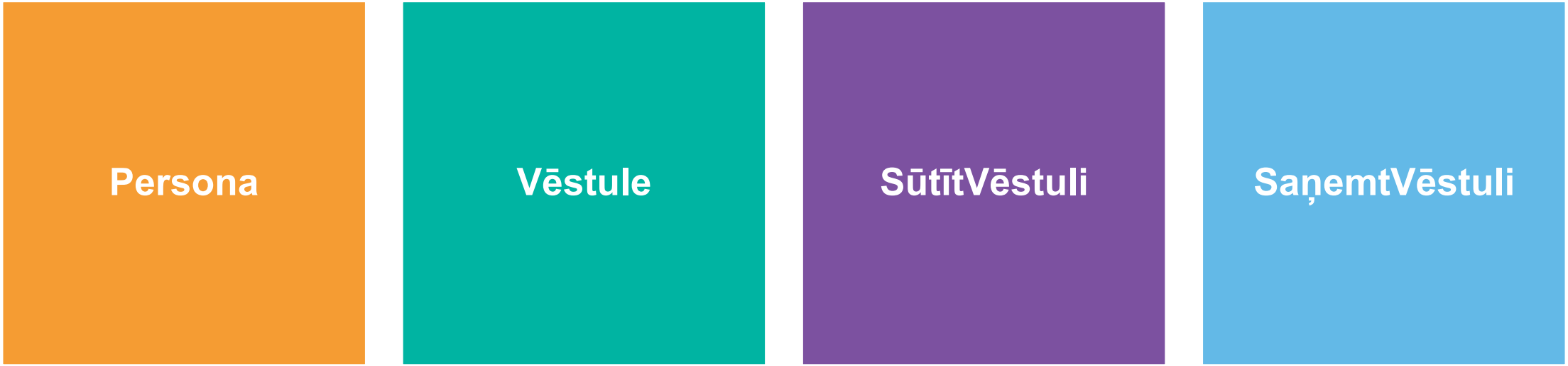
Abstrakcija

- Abstrakcija - eksistē kā ideja, bez noteikta objekta.
- Programmē par to ko programma varētu paveikt, bez konkrēta piemēra izmantošanas.
- Vēlāk šo ideju var ieviest izmantojot abstrakciju (implement).
- Ir zināmas lielās idejas, kā programmai vajadzētu darboties, bet nevar izveidot vienu konkrētu veidu, jo iespējami daudzi risinājumi vienai abstrakcijai.

Abstrakcija

- Piemēram, veidojot funkciju/klasi par vēstuļu sūtīšanu.
- Zināms, ka cilvēki sūtīs vēstules.
- Var definēt klasi SuutiitVeestuli(Persona persona, Veestule veestule), kur konkrēta persona sūtīs konkrētu vēsti.
- Var būt daudz veidu, kā tieši notiek pārsūtīšana. Caur pastu, caur epastu, ar pasta balodi.
- Veidojot klasi SuutiitVeestuli nav nepieciešams zināt, kā notiks ziņu pārsūtīšana, bet jāsaprot ka personas pārsūtīs ziņas. .

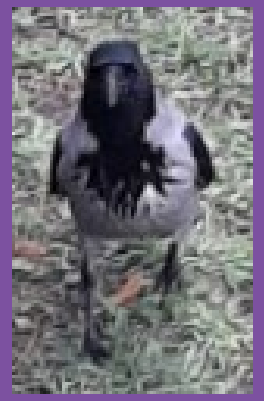
Abstrakcijas piemēri no reālās dzīves

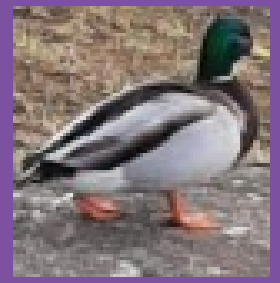


Klase no eksemplāriem

Klase – apraksts

Objekts – konkrēts eksemplārs





Klase un eksemplārs

- Klase – apraksts
- Objekts – konkrēts eksemplārs

Kods

Lietotājvārds

Parole

Pieslēgties

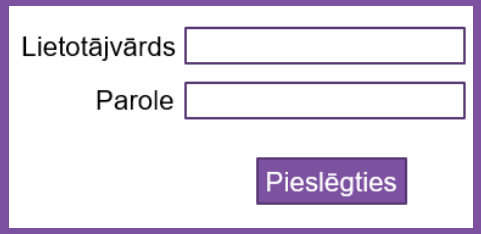
Projekts Nr. 8.3.1.1/16/I/002 Kompetenču pieeja mācību saturā

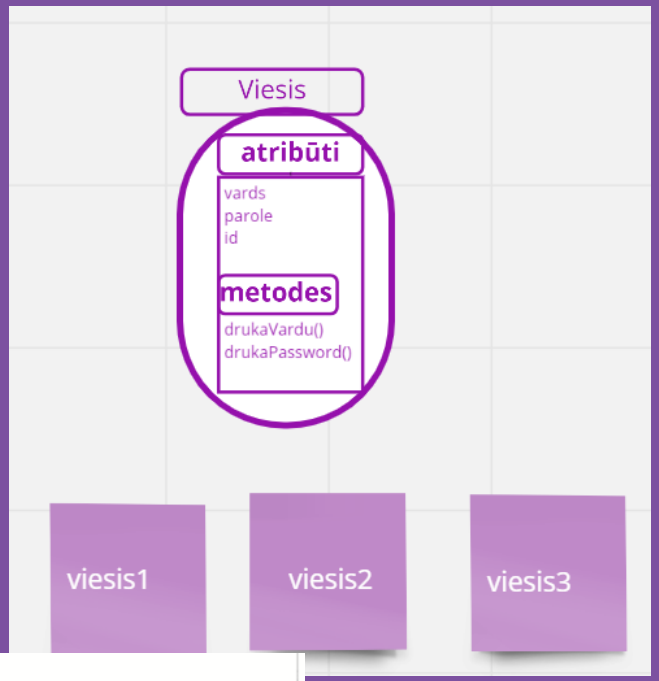
Klase - Persona

- Īpašības, kuras jāsauc par **atribūtiem**
- Darbības – kuras jāsauc par **metodēm**

- Klasē iekapsulē īpašības un darbības konkrētam uzdevumam, izveidojot **atsevišķu programmu**

InKapsulācija






```
1   class Viesis:
2       def __init__(self,vards, parole):
3           self.vards = vards
4           self.parole = parole
5       def drukaVardu(self):
6           print ("Viesis "+f"{self.vards}"+" izveidots")
7       def drukaParoli(self):
8           print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")
11  vards = "Valdis"
12  parole = "vvvvv"
```

```
13    viesis1 = Viesis(vards, parole)
14  viesis1.drukaVardu()
15  viesis1.drukaParoli()
```

```
Lietotājs Valdis izveidots
Lietotājs vvvvvvv parole  izveidota
```

??

- ? Cik eksemplāru jeb objektu bija programmas kodā?
- ? Cik eksemplāru jeb objektu bija uz attēla?

Publiskos atribūtus un metodes drīkst izmantot

```python
class Viesis:
    def __init__(self,vards, parole):
        self.vards = vards
        self.parole = parole
    def drukaVardu(self):
        print ("Viesis "+f"{self.vards}"+" izveidots")
    def drukaParoli(self):
        print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")


vards = "Valdis"
parole = "vvvvv"
viesis1 = Viesis(vards, parole)
viesis1.drukaVardu()
viesis1.drukaParoli()
```

```
Viesis Valdis izveidots
Lietotājs vvvvv parole  izveidota
>
```

Atrodi atšķirības kodā

```python
class Viesis:
    def __init__(self,vaards, parole):
        self.vards = vaards
        self.parole = parole
        self.id=3
    def drukaVardu(self):
        print ("Lietotājs "+f"{self.vards}"+" izveidots")
    def drukaParoli(self):
        print ("Lietotājs "+f"{self.parole}"+" parole
        izveidota")


vards = "Valdis"
parole = "vvvvvvv"
viesis1 = Viesis(vards, parole)
viesis1.drukaVardu()
viesis1.drukaParoli()
print(viesis1.id)
```

```python
class Viesis:
    def __init__(self,vaards, parole):
        self.vards = vaards
        self.parole = parole
        self.__id=3
    def drukaVardu(self):
        print ("Lietotājs "+f"{self.vards}"+" izveidots")
    def drukaParoli(self):
        print ("Lietotājs "+f"{self.parole}"+" parole
        izveidota")


vards = "Valdis"
parole = "vvvvvvv"
viesis1 = Viesis(vards, parole)
viesis1.drukaVardu()
viesis1.drukaParoli()
print(viesis1.__id)
```

Privātajiem atribūtiem un metodēm piekļuve tiek kontrolēta

main.py

```
1  class Viesis:
2    def __init__(self,vaards, parole):
3      self.vards = vaards
4      self.parole = parole
5      self.__id=3
6    def drukavardu(self):
7      print ("Lietotājs "+f"{self.vards}"+" izveidots")
8    def drukaParoli(self):
9      print ("Lietotājs "+f"{self.parole}"+" parole
       izveidota")
10
11
12 vards = "Valdis"
13 parole = "vvvvvvv"
14 viesis1 = Viesis(vards, parole)
15 viesis1.drukaVardu()
16 viesis1.drukaParoli()
17 print(viesis1.__id)
```

Console Shell

```
Lietotājs Valdis izveidots
Lietotājs vvvvvvv parole  izveidota
Traceback (most recent call last):
  File "main.py", line 17, in <module>
    print(viesis1.__id)
AttributeError: 'Viesis' object has no at
tribute '__id'
>
```

SKOLA 2030

???

- ? Publiskais atribūts tika nomainīts uz privāto
  - ? Jā
  - ? Nē

??

- ? Kuri ir **privātie** atribūti
  - ? privats
  - ? __lokaals
  - ? publisks
  - ? __nePublisks

Iekapsulēšana

- Iekapsulēšana - no vārda kapsula, līdzīgi, kā zāļu tabletes slēpj sevī zāles, programmas 'kapsulā' tiek slēpta tiešā pieeja datiem un darbības, kas tiek darītas ar šiem datiem.
- Jo ne vienmēr lietotājam vajag zināt, kas notiek programmas iekšpusē.
- Lietotājam pieejama viegli izmantojama saskarne.

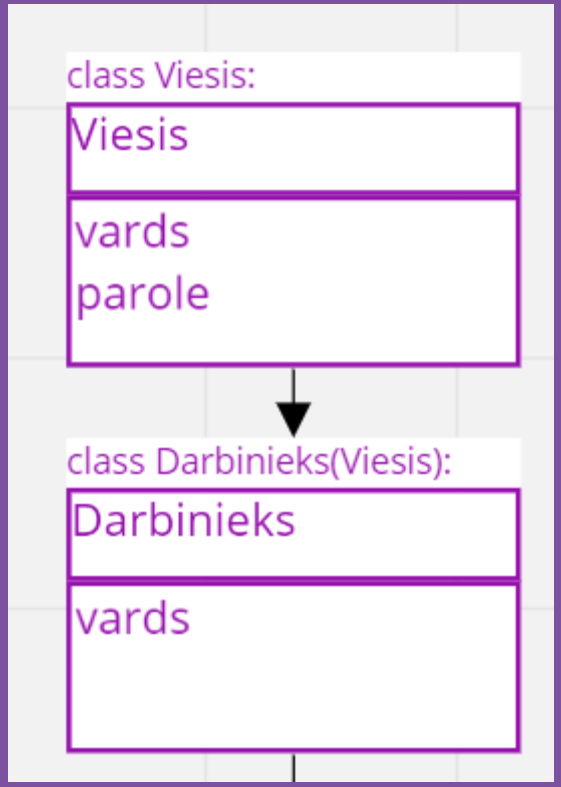
Iekapsulēšana

- Piemēram, kas notiek katru reizi ierakstot savu lietotājvārdu un paroli eklasē? Saskarnē redz pogu - "Pieslēgties".
- Bet tikmēr tiek pārsūtīti dati un salīdzināti ar parolēm datubāzēs, vai lietotājvārdi eksistē, pārbaudīts vai šifrētās paroles sakrīt atbilstošajam lietotājam esošajai.
- Lietotājam par to nav jāzina un jāuztraucas. Šīs lietas par datu pārsūtīšanu ir iekapsulētas un paslēptas no lietotāja skata, lietotājam nav jāzina tik sīki par programmu, bet lietotājs vienalga programmu spēj lietot.

Mantošana (Inheritance)

```
1   class Viesis:
2     def __init__(self,vaards, parole):
3       self.vards = vaards
4       self.parole = parole
5     def drukaVardu(self):
6       print ("Lietotājs "+f"{self.vards}"+" izveidots")
7     def drukaParoli(self):
8       print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")
```

```
11    class Darbinieks(Viesis):
12      def drukaaVardu(self):
13        print ("Administrātorss "+f"{self.vards}"+" izveidots")
```



Mantošana (Inheritance)

```
15    vards = "Valdis"
16    parole = "vvvvvvv"
17    viesis1 = Viesis(vards, parole)
18    viesis1.drukaVardu()
19    viesis1.drukaParoli()
```

```
22    vards = "Daina"
23    parole = "ddddddd"
24    darbinieks1 = Darbinieks(vards, parole)
25    darbinieks1.drukaVardu()
26    darbinieks1.drukaParoli()
```

```
Lietotājs Valdis izveidots
Lietotājs vvvvvvv parole  izveidota
Lietotājs Daina izveidots
Lietotājs ddddddd parole  izveidota
```

Instance - eksemplārs

```
1   class Viesis:
2     def __init__(self,vaards, parole):
3       self.vards = vaards
4       self.parole = parole
5     def drukaVardu(self):
6       print ("Lietotājs "+f"{self.vards}"+" izveidots")
7     def drukaParoli(self):
8       print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")
11    class Darbinieks(Viesis):
12      def drukaaVardu(self):
13        print ("Administrātorss "+f"{self.vards}"+" izveidots")
14
15
16
17
18
19    print(isinstance(darbinieks1,Viesis))
20    print(isinstance(darbinieks1,Darbinieks))
21    print(isinstance(viesis1,Viesis))
22    print(isinstance(viesis1,Darbinieks))
23
```

```
True
True
True
False
>
```

SKOLA 2030

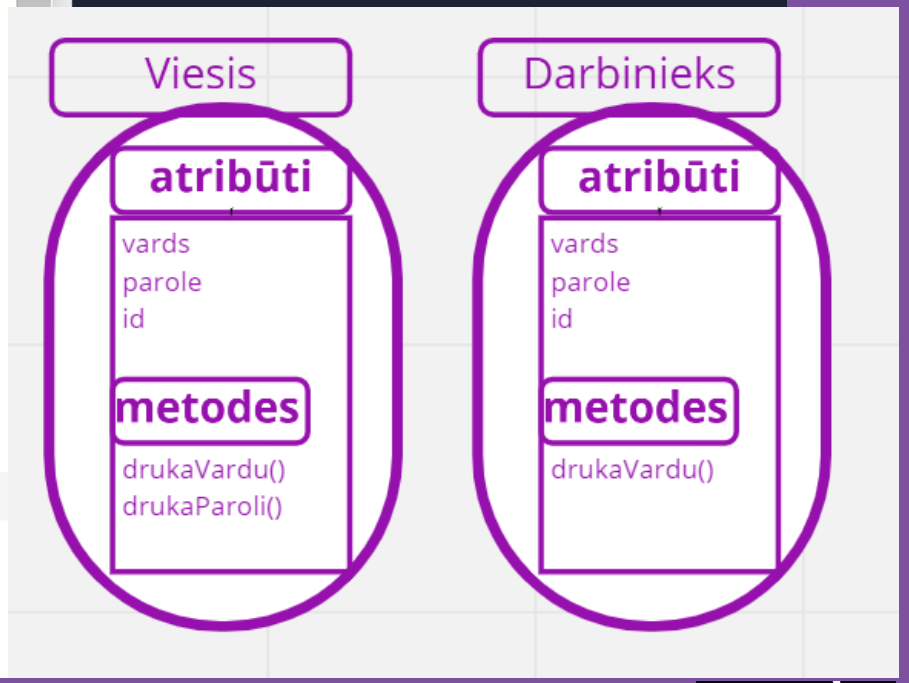
Mantošana (Inheritance)

main.py

```
1  class Viesis:
2      def __init__(self,vaards, parole):
3          self.vards = vards
4          self.parole = parole
5      def drukaVardu(self):
6          print ("Viesis "+f"{self.vards}"+" izveidots")
7      def drukaParoli(self):
8          print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")
9
10 class Darbinieks(Viesis):
11     def drukaaVaardu(self):
12         print ("Viesis "+f"{self.vards}"+" izveidots")
13
14
15 vards = "Daina"
16 parole = "ddddd"
17 darbinieks1 = Darbinieks(vards, parole)
18 darbinieks1.drukaVardu()
19 darbinieks1.drukaParoli()
```

Console Shell

```
Viesis Daina izveidots
Lietotājs ddddd parole  izveidota
```



??

- ? Vai metode drukaVardu bija katrā klasē ierakstīta?
- ? Vai metode drukaParoli bija katrā klasē ierakstīta?
- ? Paskaidro kāpēc dažādi tika izmantotas norādītās metodes?
- ? Vai klase Viesis ir vecāks klasei Darbinieks

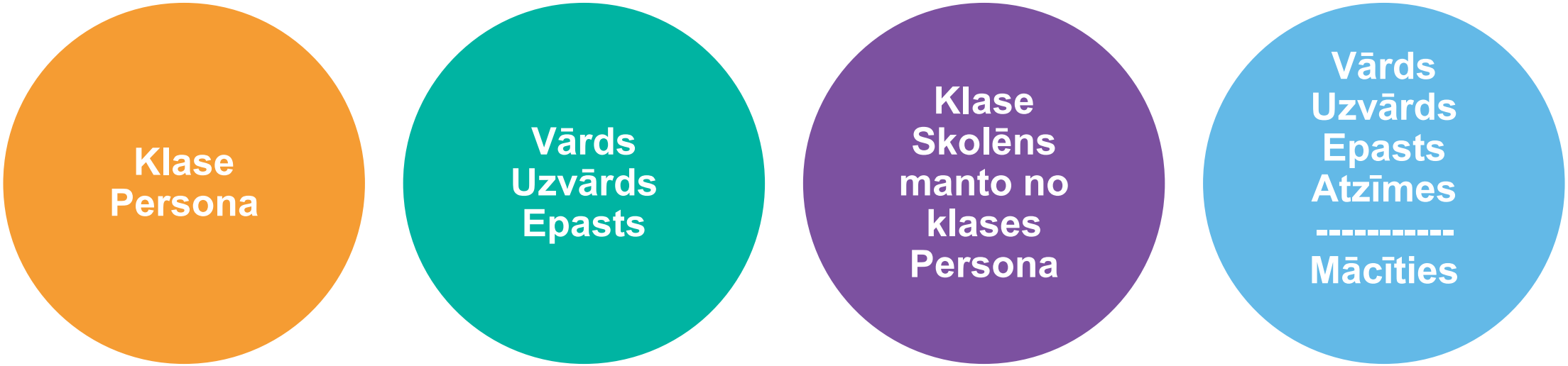
Mantošana

- Mantošana - iegūst, manto kādus datus vai darbības no kāda cita.
- Piemēram, manto citas klases funkcijas vai metodes vai datus.
- Pie tam netiek dublēts kods tajā vietā kur tiek mantots, jo mantošanai ir speciāli atslēgvārdi.

Mantošana

- Piemēram, klase Persona definē, ka katrai personai ir vārds, uzvārds, epasts utt.
- Skolēns manto klasi Persona, tātad skolēnam ir vārds, uzvārds, epasts.
- Un vēl tikai Skolēnu klasei vēl ir atzīmes, un darbība mācīties().
- Skolotājs manto klasi Persona, skolotājam ir vārds, uzvārds, epasts. Bet tikai Skolotājam ir klases žurnāls un darbība mācīt().

Klase Skolēns manto no klases Persona



Klasē Darbinieks tikai viena metode:

main.py

```
1   class Viesis:
2     def __init__(self,vaards, parole):
3       self.vards = vards
4       self.parole = parole
5     def drukaVardu(self):
6       print ("Viesis "+f"{self.vards}"+" izveidots")
7     def drukaParoli(self):
8       print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")
9
10  class Darbinieks(Viesis):
11    def drukaaVaardu(self):
12      print ("Viesis "+f"{self.vards}"+" izveidots")
13
```

Polimorfisms – viena un tā pati metode dažādās klasēs strādā dažādi

```
15    vards = "Daina"
16    parole = "ddddd"
17    darbinieks1 = Darbinieks(vards, parole)
18    vards = "Valdis"
19    parole = "vvvvv"
20    viesis1 = Viesis(vards, parole)
21
22    visi = [viesis1, darbinieks1]
23    for x in visi:
24        x.drukaVardu()
25        x.drukaParoli()
```

```
Viesis Valdis izveidots
Lietotājs vvvvv parole  izveidota
Viesis Daina izveidots
Lietotājs ddddd parole  izveidota
>
```

Polimorfisms

```python
class Viesis:
  def __init__(self,vards, parole):
    self.vards = vards
    self.parole = parole
  def drukaVardu(self):
    print ("Viesis "+f"{self.vards}"+" izveidots")
  def drukaParoli(self):
    print ("Lietotājs "+f"{self.parole}"+" parole  izveidota")
class Darbinieks(Viesis):
  def drukaVardu(self):
    print ("Viesis "+f"{self.vards}"+" izveidots")
vards = "Daina"
parole = "ddddd"
darbinieks1 = Darbinieks(vards, parole)
vards = "Valdis"
parole = "vvvvv"
viesis1 = Viesis(vards, parole)
visi = [viesis1, darbinieks1]
for x in visi:
  x.drukaVardu()
  x.drukaParoli()
```

```
Viesis Valdis izveidots
Lietotājs vvvvv parole  izveidota
Viesis Daina izveidots
Lietotājs ddddd parole  izveidota
```

Polimorfisms

- Polimorfisms - poli nozīmē vairāki, morfisms nozīmē mainīties.
- Mainīties no viena datu tipa uz citu vai viena data tipi uzvedās kā citi.
- Piemēram mantošanā, ja objekts A manto to objekta B, tad kods, kurā izmanto B objektu var izmantot arī A objekta datus un darbības.

Polimorfisms – piemērs, visi dzīvnieki var pārvietoties, mainās veids

Čūska - rāpo

Zivs - peld

Ērglis - lido

Kaķis - iet

??

- Lai ar vienu un to pašu metodi «drukaDarbiibu» izdrukātu katram dzīvniekam pareizo pārvietošanos šī metode
  - ? jāieraksta katrā klasē, bet drukā vienā vietā
  - ? jāieraksta katrā klasē, un drukā katrā klasē
  - ? nav jāraksta katrā klasē, bet drukā vienā vietā

Mājas darbs

- Izveidot klases: kvadrātam, taisnstūrim, trijstūrim.
  - Izveidot katrā klasē metodi, kura sevi nosauc
  - Ārpus klasēm ciklā ar vienu metodi izdrukāt visu daudzstūru nosaukumus un malu skaitu
- Darbu iesniegt txt formātā

Kurš no OOP pamatprincipiem reālajā dzīvē palīdz kļūt bagātam?

- ? Abstrakcija
- ? Inkapsulācija
- ? Mantošana
- ? Polimorfisms

SKOLA 2030

OOP ir veids, kā domāt par uzdevumu

Tad uzdevumu var atrisināt veidojot objektus un darbības ar tiem

Paldies!

www.skola2030.lv
facebook.com/Skola2030