

Programmeringscamp

Opgave 9 var helt tydeligt for svær – det var imponerende at 7 faktisk afleverede! Vi bruger uge 48 til at gå lidt grundigere igennem objektorienteret programmering, specielt teknikken med at 'subklasse' klasser fra andre moduler. Grunden til at det er vigtigt ligger dels i systemudviklingskurset til foråret hvor objektorienteret analyse og design (ikke programmering!) vil spille hovedrollen, dels i semesteropgaven hvor en løsningsmulighed er at genbruge ting fra de moduler der er udleveret.

Tidspunkterne er:

Mandag d. 27, 8-10

Onsdag d. 29, 15-17

Torsdag d. 30, 14-17

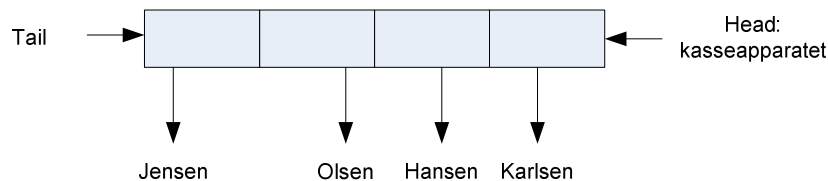
De studerende der *har* afleveret opgave 9 kan nøjes med at lave opgave 9.1 og 9.2. Studerende der ikke fik afleveret opgave 9, laver 9.1, 9.2 og 9.3 hvor 9.3 er en nemmere version af den oprindelige opgave 9.

9.1. Opbygning og specialisering af klassehierarki. Simulering af en kasselinje

Opgaven træner i at opbygge en abstrakt datastruktur og at specialisere klasserne til en konkret applikation.

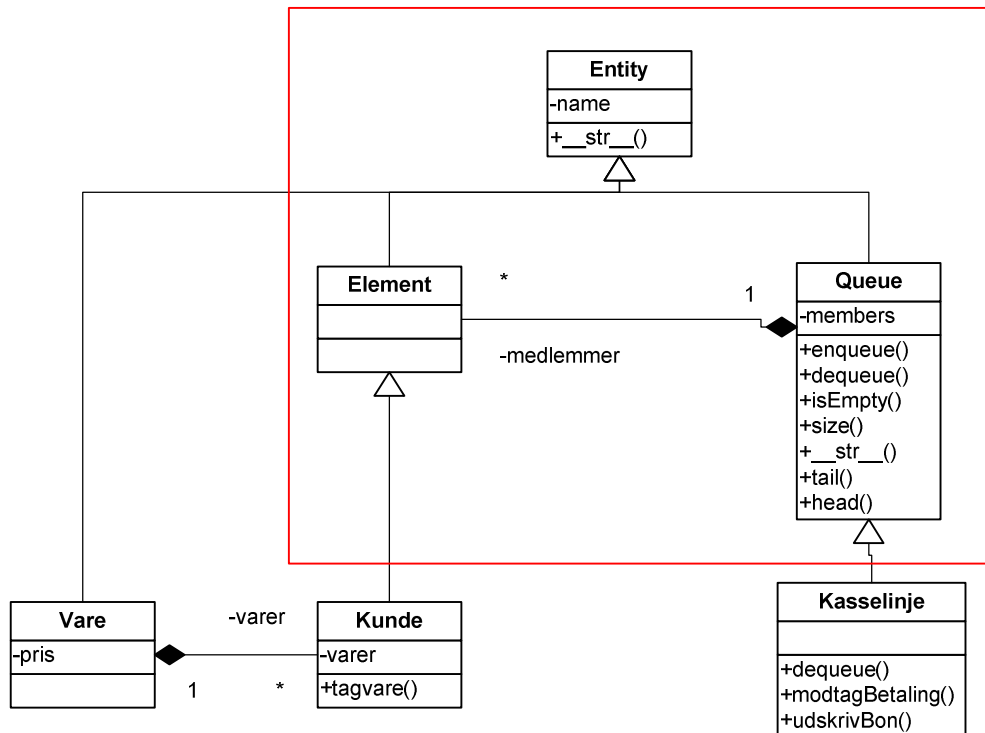
Opgaven går ud på først at implementere den abstrakte datastruktur en kø som beskrevet i slides 43:

- En kø er FIFO: first in first out
- Har følgende metoder:
 - enqueue(elem) indsæt et element bagerst i køen
 - dequeue() fjern elementet forrest i køen og returner dette.
 - size() returner tal der angiver køens længde
 - isEmpty() returner Boolean der angiver om køen er tom



Brug nedenstående diagram til at gøre dette. Hint: brug en *liste* til at implementere køens medlemmer. Lad halen af køen være element nummer 0 i listen, og hovedet være sidste element i listen. I diagrammet er der to hjælpefunktioner, head() der finder hovedet af køen hvis køen ikke er tom, og ellers returnerer None, og tail() der finder halen af køen.

Den abstrakte datastruktur kø



Specialiser dernæst Element til en Kunde og Queue til en Kasselinje. Kunden skal besidde en liste af varer han har købt. En vare har egenskaben pris. Specialiser dequeue så den kalder modtagBetaling der tæller priserne sammen og lagrer totalbeløbet i variabelen bon. Definer en funktion udskrivBon der udskriver bon'en.

Specialiseringen af *dequeue* kan foregå således:

```

def dequeue(self):
    kunde = Queue.dequeue(self)
    self.modtagBetaling(kunde)
    return kunde
  
```

hvor Queue.dequeue(self) genbruger overklassen Queue's dequeue-funktion.

En kørsel kunne se således ud:

```

if __name__ == "__main__":
    #den abstrakte kø

    print 'Den abstrakte kø'
    #lav en kø
    aQueue = Queue('kasse A')
    #indsæt 4 elementer i køen
    for n in ['a', 'b', 'c', 'd']:
        print 'indsæt ' + n
        aQueue.enqueue(Element(n))
    #udskriv køen
    print aQueue
    #fjern hovedet i køen indtil den er tom
    while not(aQueue.isEmpty()):
        print 'fjern ', aQueue.dequeue()
  
```

```

#udskriv køen
print aQueue

#kasselinjen
print
print 'Specialiseringen kasselinjen'
#lav to kunder
jensen = Kunde('Jensen', [Vare('ost', 53), Vare('mælk', 10)])
hansen = Kunde('Hansen', [Vare('tobak', 60), Vare('øl', 36)])
#lav en kasselinje
aQueue = KasseLinje('kasse A')

#indsæt kunderne i kasselinjen
for n in [jensen, hansen]:
    print 'indsæt ', n
    aQueue.enqueue(n)
#jensen fristes af slikposerne
jensen.tagvare(Vare('slik', 20))
#print køen
print aQueue
#ekspeder kunderne indtil køen er tom
while not(aQueue.isEmpty()):
    print'Ekspedition: ', aQueue.dequeue(), aQueue.udskrivBon()
#print køen
print aQueue

```

med resultatet

```

Den abstrakte kø
indsæt a
indsæt b
indsæt c
indsæt d
kasse A contains:
  a b c d
fjern a
fjern b
fjern c
fjern d
kasse A contains:
  nothing

Specialiseringen kasselinjen
indsæt Jensen
indsæt Hansen
kasse A contains:
  Jensen Hansen
Ekspedition: Jensen Bon, total: 83
Ekspedition: Hansen Bon, total: 96
kasse A contains:
  nothing

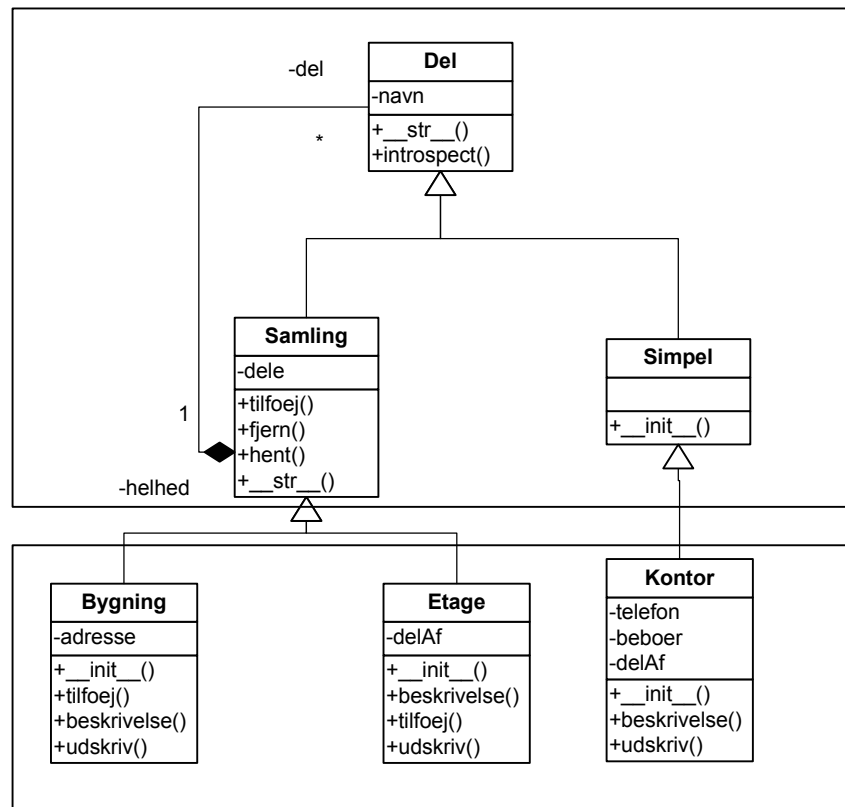
```

9.2. Brug af underklassificering af klasser i eksisterende moduler

Opgaven træner i at definere underklasser af mere generelle klasser der er gjort tilgængelige i et modul. Her er modulet *composite* som er blevet gennemgået.

I skal underklassificere klasserne *Samling* og *Simpel* i composite-mønstret så de kan repræsentere en bygning og dens dele som vist i figuren.

Composite mønstret



Denne del skal I lave

Start jeres program med

```
from composite import *
```

der importerer alle klasser fra modulet *composite* (husk at *composite* skal ligge i samme mappe som jeres program).

De nye klasser *Bygning*, *Etage* og *Kontor* skal føre kontrol med at kun etager kan være del af bygninger og kun kontorer kan være del af etager. De skal give fejlmeddelelse hvis man forsøger at tilføje en ukorrekt del. Det kræver en tilføjelse til den nedarvede tilfoej-funktion. Man kan bruge funktionen `isinstance(objekt, klasse)` til at teste om objektet er medlem af klassen. Den modificerede tilfoej-funktion kan følge nedenstående skema der stadig benytter sig af *Samling*'s tilfoej-funktion.

```
def tilfoej(self,etKontor):
    if <etKontor tilhører klassen Kontor>:
        Samling.tilfoej(self,etKontor)
        ....
    else:
        print 'Galt. Kun kontorer kan være dele af en etage'
```

De nye klasser skal have de nye attributter der er vist i figuren. I skal definere en udskriv-funktion der producerer en beskrivelse af klassernes instanser som vist neden for.

Bygning:

Bygning: Wienerbygningen
 Adresse: Helsingforsgade 14, 8200 Århus N

Etage:

Etage: E2
 Bygning: Wienerbygningen
 Adresse: Helsingforsgade 14, 8200 Århus N

Kontor:

Kontornummer: K165
 Beboer: Steffen Brandorff
 Telefon: 9234
 Etage: E1
 Bygning: Wienerbygningen
 Adresse: Helsingforsgade 14, 8200 Århus N

Det kræver at delene har adgang til den helhed de er en del af, og det kræver igen en ny tilføjelse til den nedarvede tilfoej-funktion: delen skal forsynes med en variabel der peger på helheden.

En kørsel kan se således ud hvor output er vist i kursiv:

```
Etage2 = Etage('E2')
Etage1 = Etage('E1')
Etage0 = Etage('E0')
Wiener = Bygning('Wienerbygningen', 'Helsingforsgade 14, 8200 Århus
N')
```

```
Wiener.tilfoej(Etage2)
Wiener.tilfoej(Etage1)
Wiener.tilfoej(Etage0)
Etage2.tilfoej(Kontor('K232', '9250', 'Peter Bøgh Andersen'))
Etage2.tilfoej(Kontor('K265', '9260', 'Claus Bossen'))
Etage1.tilfoej(Kontor('K165', '9234', 'Steffen Brandorff'))
```

```
print Wiener
  (Wienerbygningen
    (E2
      (K232)
      (K265)
    )
    (E1
      (K165)
    )
    (E0
      no parts
    )
  )
```

```
Wiener.udskriv()
Bygning: Wienerbygningen
Adresse: Helsingforsgade 14, 8200 Århus N
```

```
Etage2.udskriv()
Etage: E2
Bygning: Wienerbygningen
Adresse: Helsingforsgade 14, 8200 Århus N
```

```
Etage2.hent('K232').udskriv()
Kontornummer: K232
Beboer: Peter Bøgh Andersen
Telefon: 9250
Etage: E2
Bygning: Wienerbygningen
Adresse: Helsingforsgade 14, 8200 Århus N
```

```
Etage1.hent('K165').udskriv()
Kontornummer: K165
Beboer: Steffen Brandorff
```

```

Telefon: 9234
Etage: E1
Bygning: Wienerbygningen
Adresse: Helsingforsgade 14, 8200 Århus N

```

```

Etage2.fjern(Etage2.hent('K232'))
print Wiener
  (Wienerbygningen
    (E2
      (K265)
    )
    (E1
      (K165)
    )
    (E0
      no parts
    )
  )

```

Hvis I er usikre på hvilke attributter og metoder et objekt indeholder, kan I kalde *introspect* funktionen:

```

print Etage1.hent('K165').introspect()

Dictionary:
{'beboer': 'Steffen Brandorff', 'delAf': <__main__.Etage instance
at 0x0118E260>, 'telefon': '9234', 'navn': 'K165'}
Attributes and functions:
['__doc__', '__init__', '__module__', '__str__', 'beboer', 'beskri-
velse', 'delAf', 'introspect', 'navn', 'telefon', 'udskriv']

```

9.3. Ændring af og tilpasning af modul

I modulet `OOogSql1` er der en funktion `makeTable(tablecontents)` der tager en liste af lister som input og blot returnerer en tekstuel version af listen. I stedet for linjen `htmltabel = str(tablecontents)` skal I indsætte den funktion I tidligere lavede, som konverterer en liste af lister til en html-tabel.

I selve kørselsdelen efter `if __name__ == "__main__"` skal I ændre koden så den kan køre med data fra jeres egen database.

Hvis I er usikre på hvilke attributter og metoder et objekt indeholder, kan I kalde *introspect* funktionen:

```

print
print 'Introspect a Table'
print aTable.introspect()
print
print 'Introspect a Row'
print aRow.introspect()
print
print 'Introspect a Selection'
print aSelection.introspect()

```

Det giver følgende resultat:

Introspect a Table

```

Dictionary: {
'values': [['1111111111', 'Jeppe Boegh', 'Andersen', 'Thorsgade
20', '8410', 'dreng', '', '86379790', 'Baltica', '1111111112',
'1111111113', '1111111114'], ['1111111114', 'Lotte Boegh', 'Ander-
sen', 'Thorsgade 20', '8410', 'pige', '', '86379790', 'Baltica',

```

```
'1111111112', '1111111113', '1111111114'], ['1111111115', 'Stine',
'Jacobsen ', 'Thorsgade 20', '8410', 'pige', '', '86379790', 'bal-
tica', '', '1111111116', '1111111114']],
'columnNames': ['cpr', 'firstname', 'lastname', 'address', 'city',
'gender', 'email', 'phone', 'insurance', 'hasFather', 'hasMother',
'hasDoctor'],
'rows': [<__main__.Row instance at 0x012262B0>, <__main__.Row
instance at 0x012267B0>, <__main__.Row instance at 0x012268C8>],
'tableName': 'Child',
'primaryKey': 'cpr'}
```

```
Functions:
['__doc__', '__init__', '__module__', 'columnNames', 'getPrimary-
Key', 'getRow', 'introspect', 'makeHTMLTable', 'primaryKey',
'printContents', 'rows', 'tableName', 'values']
```

Introspect a Row

```
Dictionary: {
'values': ['1111111111', 'Jeppe Boegh', 'Andersen', 'Thorsgade 20',
'8410', 'dreng', '', '86379790', 'Baltica', '1111111112',
'1111111113', '1111111114'],
'columnNames': ['cpr', 'firstname', 'lastname', 'address', 'city',
'gender', 'email', 'phone', 'insurance', 'hasFather', 'hasMother',
'hasDoctor'],
'tableName': 'Child',
'primaryKey': 'cpr'}
```

```
Functions:
['__doc__', '__init__', '__module__', 'columnNames', 'delete',
'get', 'getPrimaryKey', 'insert', 'introspect', 'makeHTMLTable',
'primaryKey', 'printContents', 'put', 'putAllValues', 'tableName',
'update', 'values']
```

Introspect a Selection

```
Dictionary: {
'columnNames': ['cpr', 'firstname', 'lastname'],
'values': (('1111111115', 'Stine', 'Jacobsen '),)}
Functions:
['__doc__', '__init__', '__module__', 'columnNames', 'introspect',
'makeHTMLTable', 'printContents', 'values']
```