Løsningsforslag

Runde 3

3.5 Klassediagram

3.5.1
Avgjør hvilken relasjon det er mellom klassene i programmene nedenfor:

a)

```python
class Sjåfør():
    def __init__(self, navn:str):
        self.navn = navn
        print('Ny sjåfør: ' + navn)

class Varebil():
    def __init__(self,reg_nr:str):
        self.reg_nr = reg_nr
        self.sjåfør = None
        print('\nNy varebil: ' + reg_nr)

    def bruk_sjåfør(self,sjåfør:Sjåfør)->None:
        self.sjåfør = sjåfør
        print('\nNå er ' + self.sjåfør.navn + ' sjåfør'
              '\npå varebil ' + self.reg_nr)

varebil = Varebil('AJ77309')
sjåfør = Sjåfør('Stig')
varebil.bruk_sjåfør(sjåfør)
```

```
Ny varebil: AJ77309
Ny sjåfør: Stig

Nå er Stig sjåfør
på varebil AJ77309
```

Det er en referanse fra Varebil- til Sjåfør-klassen. Dessuten blir ikke sjåføren borte når varebilen slutter å eksistere. Så dette er ASSOSIASJON. Som sagt så skiller ikke aggregering seg vesentlig fra en vanlig assosiasjon bortsett fra det viser hva som er hovedobjekt og hva som er delobjekt. Her kan det være vanskelig å skille. I prinsippet kan en varebil bli kjørt av flere sjåfører, og en sjåfør kan kjøre flere varebiler. Hvis vi hadde samlet sammen varer i lasterommet hadde det vært mer naturlig med aggregering.

b)

```python
class Robot():
    def __init__(self, navn:str):
        self.navn = navn
        print('Ny robot: ' + navn)
        self.armer = []
        self.armer.append(Arm('høyre'))
        self.armer.append(Arm('venstre'))

class Arm():
    def __init__(self,navn:str):
        self.navn = navn
        print('Ny ' + self.navn + ' arm')

robot = Robot('R2-D2')
print([x for x in dir(robot) if x[0:2] != '__' ])
```

✓ 0.6s

```
Ny robot: R2-D2
Ny høyre arm
Ny venstre arm
['armer', 'navn']
```

Robot-klassen oppretter armene, og det er referanser fra Robot-klassen til Arm-klassen. Arm-objektene forsvinner når Robot-objektet slutter å eksistere. Vi kan si at en robot blant annet "BESTÅR-AV" en eller flere armer. Så dette er KOMPOSISJON.

c)

```python
class Gave:
    def __init__(self,navn:str):
        self.navn = navn
        print('Ny gave: ' + navn)

class Julenisse:
    def __init__(self,navn:str):
        self.navn = navn
        print('Ny julenisse: ' + navn)

    def pakk(self,gave:Gave)->None:
        print(self.navn + ' pakker en ' + gave.navn)

klaus = Julenisse('Klaus')
gave  = Gave('Google Chromecast')
klaus.pakk(gave)
```

✓ 0.1s

```
Ny julenisse: Klaus
Ny gave: Google Chromecast
Klaus pakker en Google Chromecast
```

Det er ingen referanse fra Julenisse-klassen til Gave-klassen, men Gave-klassen blir brukt som parameter i pakk-metoden i Julenisse-klassen. Endringer i Gave-klassen kan medføre at vi må gjøre endringer i Julenisse-klassen. Så dette er AVHENGIGHET.

d)

```python
class Person():
    def __init__(self, navn:str):
        self.navn = navn
        print('Ny person: ' + navn)

    def hils(self):
        print('Jeg heter ' + self.navn +
              ' og er ' + self.__class__.__name__.lower())

class Elev(Person):
    def __init__(self,navn:str):
        super().__init__(navn)

class Lærer(Person):
    def __init__(self,navn:str):
        super().__init__(navn)

personer = []
personer.append(Elev('Hilde'))
personer.append(Lærer('Knut'))
for person in personer: person.hils()
```

✓ 0.4s

```
Ny person: Hilde
Ny person: Knut
Jeg heter Hilde og er elev
Jeg heter Knut og er lærer
```

Elev- og Lærer-klassene arver navn-attributtet og hils-metoden fra Person-klassen. Både elev og lærer "ER-EN" person. Så dette er GENERALISERING.

e)

```python
from abc import ABC, abstractmethod

class Konto(ABC):
    @abstractmethod
    def __init__(self,konto_nr:str):
        self.konto_nr = konto_nr
        print('\nNy ' + self.__class__.__name__ +
              ': ' + konto_nr)

class Sparekonto(Konto):
    maks_antall_uttak = 12

    def __init__(self, konto_nr:str):
        super().__init__(konto_nr)
        print('Maks antall uttak: ' +
            str(Sparekonto.maks_antall_uttak))

class BSU_konto(Konto):
    maks_årlig_sparebeløp = 27500

    def __init__(self, konto_nr:str):
        super().__init__(konto_nr)
        print('Maks årlig sparebeløp: ' +
            str(BSU_konto.maks_årlig_sparebeløp))

karis_konto = Sparekonto('1234 56 78999')
olas_konto  = BSU_konto('4343 55 67677')
```

✓ 0.8s

```
Ny Sparekonto: 1234 56 78999
Maks antall uttak: 12

Ny BSU_konto: 4343 55 67677
Maks årlig sparebeløp: 27500
```

Dette kan se ut som generalisering, for Sparekonto- og BSU_konto-klassene arver konto_nr-attributtet fra Konto-klassen. Konto-klassen er imidlertid abstrakt så vi kan ikke opprette Konto-objekter (Prøv!). Så dette er REALISERING.

f)

```python
from random import shuffle

class Prosjekt():
    def __init__(self, navn:str, medlemmer:list):
        self.navn = navn
        print('Nytt prosjekt: ' + navn)
        self.medlemmer = medlemmer

    def vis_prosjektgruppe(self)->None:
        print('\nMedlemmer i prosjektet ' + self.navn + ' er')
        navn = ', '.join([medlem.navn for medlem in self.medlemmer])
        før, _, etter = navn.rpartition(', ')
        print(før + ' og ' + etter)

class Ansatt():
    def __init__(self,navn:str):
        self.navn = navn
        print('Ny ansatt: ' + navn)

ansatte = []
ansatte.append(Ansatt('Erik'))
ansatte.append(Ansatt('Frida'))
ansatte.append(Ansatt('Gustav'))
ansatte.append(Ansatt('Henriette'))
shuffle(ansatte)
prosjekt = Prosjekt('Julebord',ansatte[0:3])
prosjekt.vis_prosjektgruppe()
```

✓ 0.6s

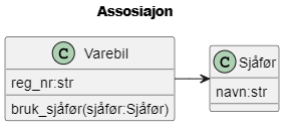
```
Ny ansatt: Erik
Ny ansatt: Frida
Ny ansatt: Gustav
Ny ansatt: Henriette
Nytt prosjekt: Julebord

Medlemmer i prosjektet Julebord er
Frida, Erik og Henriette
```

Det er referanser til Ansatt-objekter i Prosjekt-klassen. medlemmer-attributtet inneholder en liste med noen av Ansatt-objektene. Ansatt-objektene opprettes utenfor Prosjekt-klassen og forsvinner ikke når Prosjekt-objektet slutter å eksistere. Prosjekt-objektet er hovedobjekt og "HAR-EN" eller flere prosjektmedlemmer i form av Ansatt-objekter som delobjekter. Så dette er AGGREGERING.

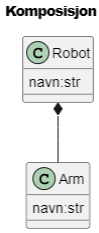
3.5.2

Lag UML klassediagrammer til programmene i oppgave 3.5.1.

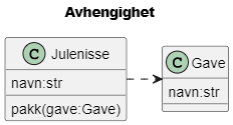
a)



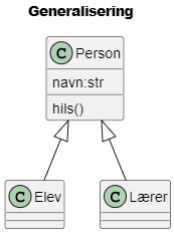
b)



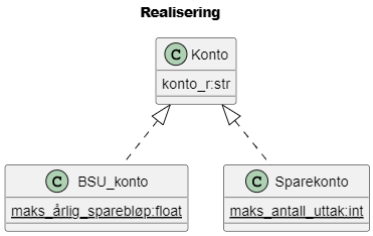
c)



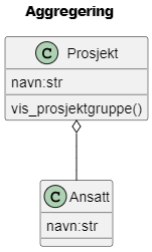
d)



e)



f)



3.5.3

En idrettsklubb har 10 medlemmer. En kveld skal de ha 4x100 meter stafettkonkurranse med to lag. Hvert av lagene trekkes ut tilfeldig blant medlemmene, som skal løpe etappene i den rekkefølgen de blir trukket ut. De to som ikke blir trukket ut, skal starte stafetten og ta tiden på lagene. Det er fem jenter og fem gutter i klubben. Jentene løper 100 meter på mellom 11.5 og 13.5 sekunder. Guttene bruker mellom 11.0 og 13.0 sekunder.

Lag en modell og et program som simulerer stafetten. Bruk random.uniform(a,b) som returnerer et vilkårlig desimaltall mellom a og b.

a) Bruk substantivmetoden til å bestemme klasser og attributter i modellen.

En idrettsklubb har 10 medlemmer. En kveld skal de ha 4x100 meter stafettkonkurranse med to lag. Hvert av lagene trekkes ut tilfeldig blant medlemmene, som skal løpe etappene i den rekkefølgen de blir trukket ut. De to som ikke blir

trukket ut, skal starte stafetten og ta tiden på lagene. Det er fem jenter og fem gutter i klubben. Jentene løper 100 meter på mellom 11.5 og 13.5 sekunder. Guttene bruker mellom 11.0 og 13.0 sekunder.

klubb (2), medlem (2), kveld (1), stafett (2), lag (3), etappe (1), rekkefølge (1), tid (1), jente (2), gutt (2), meter (1) og sekund(2).

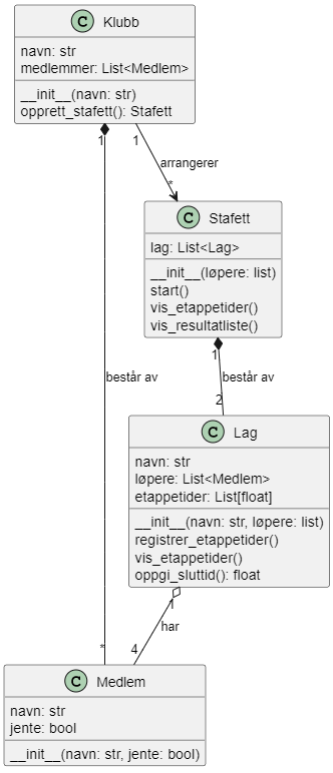
"Jente" og "gutt" er enkle attributtverdier som kan plasseres i klassen "Medlem". "Kveld" virker uinteressant for vår stafettsimulering og kan utelates. "Etappe" og "rekkefølge" er gitt av lagoppsettet og trenger ikke være egne attributter. For å holde oversikt over tiden til hver løper i stafetten, kan vi opprette en liste over tider i "Lag"-klassen. "Lag"-klassen vil inneholde "Medlem"-objekter som representerer løperne for hvert lag. En ordnet liste med "Medlem"-objekter vil gi oss all nødvendig informasjon om hvert lag i stafetten. Dermed ser det ut som vi kan klare oss med en "Klubb"-klasse som kan opprette "Medlem"-objekter og "Stafett"-objekter med "Lag"-objekter. "Medlem"-objekter kan ha et navn og et attributt "jente" som er sann for jenter og usann for gutter. Tid vil bli representert som en ordnet liste i "Lag"-klassen, og "meter" og "sekund" er benevninger i denne sammenhengen og ikke klasser eller attributter.

b) Bestem forholdene mellom klassene.

En klubb BESTÅR AV medlemmer og som ikke kan være medlemmer hvis klubben legges ned. Det blir **komposisjon**. Klubben arrangerer stafetter. Det blir **assosiasjon**. Stafetten BESTÅR AV lag som opphører når stafetten opphører. Det blir **komposisjon**. Lagene HAR medlemmer som løper etappene og fortsetter å eksistere etter at stafetten og lagene opphører. Det blir **aggregering**.

c) Hvilke operasjoner må med?

Vi må kunne opprette medlemmer, og kan gjøre det i __init__ når vi oppretter klubben. Klubben må kunne opprette en stafett som må kunne startes. Når klubben oppretter en stafett, må den først trekke ut de 8 medlemmene som skal være med. Deretter kan den opprette en stafett som må opprette lagene. Dette kan vi gjøre i stafettens __init__-funksjon. Når vi starter en stafett, må etappetidene frembringes. Start operasjonen kan be lagene om å registrere etappetidene. Stafetten må kunne vise resultatliste og eventuelt vise etappetider som den ber lagene om å gjøre fordi lagene har etappetidene Når stafetten skal vise resultatlisten kan vi be hvert lag om å oppgi sluttiden siden lagne også kan summere etappetidene.



d) Tegn et UML klassediagram for modellen.

Se filen p_3_5_3_d.puml og figuren til høyre.

e) Lag et program som simulerer en stafett, skriver ut etappetidene og en resultatliste.

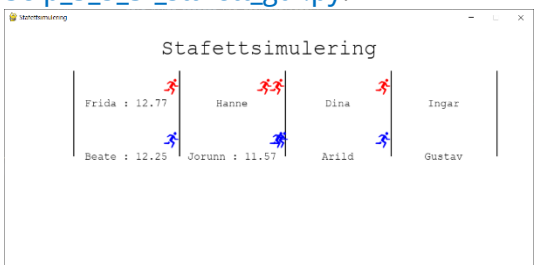
```
Ny klubb: Rask
Ny stafett

Etappetider:
************* Lag Rød
Arild   12.70
Beate   11.76
Jorunn  12.49
Dina    12.24
************* Lag Blå
Edward  11.78
Cato    12.84
Ingar   11.87
Frida   11.72

Resultatliste
1.Blå   48.21
2.Rød   49.19
```

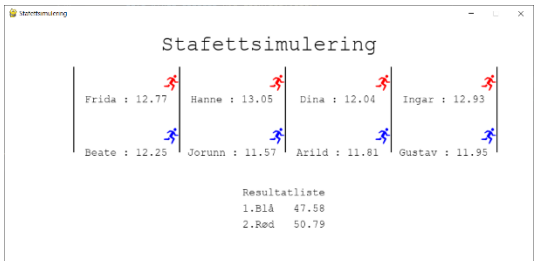
```
Ny klubb: Rask
Ny stafett

Etappetider:
************* Lag Rød
Jorunn  12.77
Beate   13.42
Frida   13.16
Ingar   12.10
************* Lag Blå
Cato    11.56
Dina    11.83
Gustav  11.08
Hanne   13.32

Resultatliste
1.Blå   47.78
2.Rød   51.45
```

```
Ny klubb: Rask
Ny stafett

Etappetider:
************* Lag Rød
Beate   11.99
Edward  11.26
Frida   11.59
Ingar   11.93
************* Lag Blå
Hanne   11.88
Jorunn  12.30
Gustav  11.52
Cato    12.07

Resultatliste
1.Rød   46.78
2.Blå   47.77
```

Se p_3_5_3e_stafett.py.

f) (Tilleggsoppgave - valgfri) Lag et program med GUI og 8 figurer som står oppstilt og flytter seg som det var en stafett langs en rett linje.
Se p_3_5_3f_stafett_gui.py.





Husk å bruke random.uniform(a, b) for å generere tilfeldige desimaltall mellom a og b for løpetidene til utøverne. Medlemslisten kan stokkes med random.shuffle(x).

