UiT
NORGES
ARKTISKE
UNIVERSITET

SENSORVEILEDNING

For eksamen i: **INF-1400 Objektorientert Programmering**
Dato: **Tirsdag 23. mai 2017**

Sensorveiledningen er på 7 sider inklusiv forside

Fagperson/intern sensor: John Markus Bjørndalen.
Telefon/mobil: 90148307

Les gjennom hele oppgavesettet før du begynner å løse oppgavene.

Oppgavene kan besvares på Norsk, Engelsk, Svensk eller Dansk.

I besvarelsen kan du bruke Python, pseudokode eller en kombinasjon.

Noen oppgaver har kun en "a)". Dette er for å tydeliggjøre hva som er spørsmålet som skal besvares. Det har ikke falt ut noen delspørsmål.

Noen norske termer som er brukt i stedet for de engelske fra boka:

- Klasse - class
- Arv - inheritance
- Atributt - Attribute
- Metode - Method

Vi skal lage et system for å koordinere mange roboter og sensorer som skal undersøke et katastrofeområde. Robotene blir styrt av flere uavhengige operatører, og det finnes flere typer roboter som skal koordineres.

Vi trenger å lage et oversiktsbilde og å ha muligheten til å sende og motta meldinger til operatørene for de forskjellige robotene og sensorene.

Oppgave 1 (20%)

Gitt følgende grove beskrivelser av roboter og sensorer skal vi generalisere og lage et klassehierarki. Felles for dem er at vi kan kalle dem *sensorplatformer* (vi ser bare på sensorsiden for å gjøre oppgaven litt enklere).

Klassenavn	Attributter
Drone	inbox, outbox, position, flight time, images
AirQualityDrone	inbox, outbox, position, flight time, images, air quality log
Rover	inbox, outbox, position, cargo, sensorlist
Sensor	inbox, outbox, position, sensor data

Rover er litt spesiell siden den er mobil og har en liste av objekter av typen Sensor (sensorlist). Du kan også ha en bunke med stasjonære sensorer som bare er av typen Sensor.

Du velger selv typen på attributtene med unntak av `sensorlist`, `inbox` og `outbox`. Disse skal alle være lister. For enkelthets skyld kan du også la luftkvalitetsloggen være en enkel liste.

a) Beskriv hvordan du vil endre klassene over for å *generalisere* og introdusere et klassehierarki. Angi hvordan du vil bruke arv.

b) Implementer klassene du endte opp med. Du trenger kun å ta med attributter og `__init__()`-metoden i denne runden (andre metoder ser vi bort fra nå).

c) Skriv koden som lager et objekt av typen AirQualityDrone.

a) Her er det viktigste at de får skissert et fornuftig hierarki (med arv) og får med hvor attributter bør defineres.

b) Se kode under for en tilnærming. Om de tar inn init-verdier til attributtene som parametre er

mindre viktig her siden jeg ikke har bedt om det.

c) Se kode under. Det viktigste her er å få med attributter som skal initialiseres og at arv brukes korrekt (noe som delvis er besvart over).

```
class SensorPlatform:
    def __init__(self):
        self.inbox = []
        self.outbox = []
        self.position = []

class Drone(SensorPlatform):
    def __init__(self):
        super().__init__()
        self.flight_time = 100
        self.images = []

class AirQualityDrone(Drone):
    def __init__(self):
        super().__init__()

class Sensor(SensorPlatform):
    def __init__(self):
        super().__init__()
        self.sensordata = []

class Rover(SensorPlatform):
    def __init__(self, sensorlist):
        super().__init__()
        self.cargo = []
        self.sensorlist = sensorlist

objs = [
    Drone(),
    AirQualityDrone(),
    Sensor(),
    Rover([Sensor(), Sensor()])
    ]


aqd = AirQualityDrone()
```

Oppgave 2 (20%)

Vi lager en ny klasse (`SiteManager`) som skal hjelpe oss å kontrollere robotene og sensorene. Foreløpig er den veldig enkel:

```
class SiteManager:
    def __init__(self):
        self.splist = []

    def add_sensor(self, sensorobj):
        """Legger til en robot eller sensor i listen over
        sensorplatformer vi holder øye med"""
```

```
        self.splist.append(sensorobj)
```

Anta at vi har satt opp et objekt av klassen over (kall det `site_manager`) og at vi har lagt til flere roboter og sensorer. Nå skal vi først lage en metode som går gjennom listen og legger inn en melding i inbox på alle droner og rovere (men ikke sensorer).

a) Lag en metode (`send_msg_to_robot_operators(self, msg)`) som legger inn en melding hos robotene (droner og rovere). Du kan anta at inbox er lister og at du bare kan kjøre `append` på listene (vi trenger ikke utvide klassene til å ta i mot meldinger). Merk at sensorer ikke skal ta i mot meldinger.

b) I hvilken klasse vil du legge denne metoden? Hvorfor?

```
class SiteManager:
    def __init__(self):
        self.splist = []

    def add_sensor(self, sensorobj):
        """Legger til en robot eller sensor i listen over
        sensorplatformer vi holder øye med"""
        self.splist.append(sensorobj)

    def send_msg_to_robot_operators(self, msg):
        for sp in self.splist:
            if isinstance(sp, (Rover, Drone)):
                sp.inbox.append(msg)

sm = SiteManager()

for o in objs:
    sm.add_sensor(o)

sm.send_msg_to_robot_operators('foo')
for o in objs:
    print(o, o.inbox)
```

a) De bør klare å gå igjennom listen og bruke isinstance eller tilsvarende for å sjekke typer. Det er ikke noe krav at de bruker en tuple (Rover, Drone) istedet for to if-tester.

b) I SiteManager siden man trenger tilgang til listen av objekter.

Oppgave 3 (15%)

En trøtt foreleser skrev denne koden rett før han sovnet på kontoret:

```
class Otter(list):
    def __init__(self):
        pass

    def add_stone_on_belly(self, stone):
        self.append(stone)

    def remove_one_stone(self):
        return self.pop(0)
```

```
o = Otter()
o.add_stone_on_belly(42)
o.add_stone_on_belly(4222)
print("Got", o.remove_one_stone())
```

a) Vi har implementert koden over via et *er-forhold* (is-a relationship). Hva betyr det?

b) For dette eksemplet er det kanskje ikke hensiktsmessig at oteren er implementert som den er. Skriv om koden til et har-forhold (has-a relationship) og beskriv kort hva vi mener med et har-forhold sammenlignet med et er-forhold.

```
class Otter():
    def __init__(self):
        self.stones = []

    def add_stone_on_belly(self, stone):
        self.stones.append(stone)

    def remove_one_stone(self):
        return self.stones.pop(0)

o = Otter()
o.add_stone_on_belly(42)
o.add_stone_on_belly(4222)
print("Got", o.remove_one_stone())
```

a) De bør få med koblingen til arv og at Otter er en liste.

b) Legge til en steinliste og fjerne arven. Forklare at Oteren ikke er en liste etter endringen.

Oppgave 4 (15%)

```
A_VAL = 45
B_VAL = 100

class A:
    pass

class B:
    pass

lst = [ A(), B() ]

for v in lst:
    val = None
    if type(v) is A:
        val = A_VAL
    elif type(v) is B:
        val = B_VAL
    print("Val is", val)
```

Ett problem med koden over er at vi må sjekke typen på et objekt i if-tester siden vi ønsker å finne en verdi basert på typen til objektet. Det gjør det kronglete å legge til nye klasser siden vi må finne alle steder vi gjør lignende ting og oppdatere dem for å støtte de nye klassene.

a) Hva er en klasseattributt?

b) Hvordan kan vi forenkle koden via klasseattributter slik at vi ender opp med en ny (og enklere) for-løkke som ikke må endres når vi legger til nye typer?

a) Attributt som er definert på klassen og dermed synlig/delt mellom objektene.

b) Hovedpoeng at man fjerner if-testene og legger VAL inn i klassene. Se eksempelkode under.

```
class A:
    VAL = 45

class B:
    VAL = 100

lst = [ A(), B(), ]

print("Test 3")
for v in lst:
    print("Val for obj ", v, " is ", v.VAL)
```

Oppgave 5 (15%)

```
class A:
    def __init__(self):
        pass

    def foo(self, val):
        print('snadder', val)

class  B(A):
    def __init__(self):
        super().__init__()

class  C(B):
    def __init__(self):
        super().__init__()

    def foo(self, val):
        print('pling', val)

a = A()
b = B()
c = C()

a.foo(1)
b.foo(2)
c.foo(3)
```

a) Hva er polymorfi?

b) Angi hva programmet over skriver ut.

a) De bør i hvert fall ha med poenget at oppførselen til et objekt er forskjellig avhengig av typen til objektet. For eksempel vil et metodekall kjøre en gitt implementasjon av metoden avhengig av klassen til objektet.

b)
```
snadder 1
snadder 2
pling 3
```

Oppgave 6 (15%)

a) Gi en kort beskrivelse av 2 av de følgende uttrykk/konsepter:

- Klasse vs. objekt
- Multiple inheritance
- Mutable/immutable
- getter/setter
- Pattern (mønster)

Det holder med en kort beskrivelse av hvert av de to punktene de velger som tar med hovedtrekkene/konseptet.