UiT
NORGES
ARKTISKE
UNIVERSITET

EKSAMENSOPPGAVE

Eksamen i:	INF-1400 Objektorientert Programmering
Dato:	Fredag 2019-09-27
Klokkeslett:	Kl 09:00 - 13:00
Sted:	Adm.bygget K1.04
Tillatte hjelpemidler:	Ingen
Type innføringsark (rute/linje):	Digital eksamen
Antall sider inkl. forside:	6
Kontaktperson under eksamen: Telefon/mobil:	Edvard Pedersen 40458598 (EP)
Runde i eksamenslokalet ca. kl.: 11.	

NB! Det er ikke tillatt å levere inn kladdepapir som del av eksamensbesvarelsen.

Hvis det likevel leveres inn, vil kladdepapiret bli holdt tilbake og ikke bli sendt til sensur.

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 selve spørsmålet som skal besvares er, det har ikke falt ut noen delspørsmål.

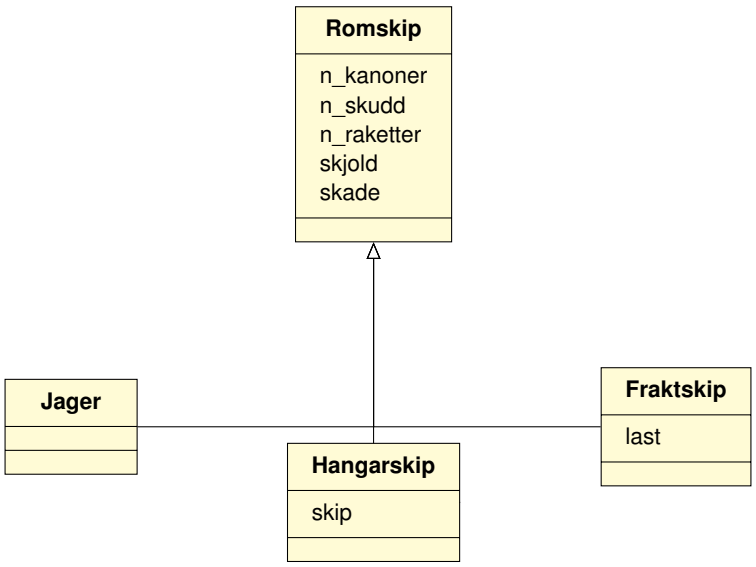
Noen norske termer som er brukt i stedet for de engelske fra boka:

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

Del A

En veldig nyttig måte å lære programmering på er å delta i større prosjekter. Etter å ha meldt deg frivillig til et ambisiøst prosjekt som ønsker å lage et spill som skal rekruttere fremtidige piloter til USAs nye “Space Force” oppdager du at man kanskje også kan lære noe om hvordan man *ikke* skal gjøre ting. Det er mange gode idéer her, men ting må ryddes opp i før noen kan begynne å leke seg med første utgave.

Oppgave 1 (20%)



a) Implementer klassehierarkiet som er angitt i UML-diagrammet over. Få tydelig fram arv og attributter.

b) Skriv kode for å lage et objekt av typen Fraktskip. Hvilke attributter kan du nå fra objektet?

Oppgave 2 - 20 %

Vi antar at det kan være flere lag med i spillet som alle har mange romskip. For å regne ut styrken til hvert lag trenger vi en funksjon som regner ut styrken til et gitt lag basert på en liste over romskipene til laget.

```
POENG_JAGER = 10
POENG_HANGARSKIP = 100
POENG_FRAKTSKIP  =  70

def team_score(romskip):
    """Regner ut en styrkefaktor for et lag basert på en liste (romskip) med
    romskipene til laget"""
    pass
```

a) Lag funksjonen `team_score` som regner ut poeng basert på hvilken type hvert romskip er. Konstantene øverst (`POENG_JAGER`, `POENG_HANGARSKIP`, `POENG_FRAKTSKIP`) angir hvor mange poeng hver type romskip gir.

Ulempen med dette er at vi må endre `team_score` og legge til konstanter hvis vi ønsker å legge til flere klasser. Vi kan gjøre dette mer fleksibelt ved å erstatte konstantene over med en **klasseattributt** (`STYRKEPOENG`) i hver av romskipklassene.

b) Vis hvordan du gjør dette ved å bruke `Jager`-klassen som eksempel. Vis hvordan du implementerer `team_score` når du kan bruke klasseattributten STYRKEPOENG.

Et lite hint: funksjonen i b) burde bli merkbart enklere enn den i a).

Oppgave 3 - 25 %

En romstasjon i spillet trenger forsvarsverk rundt seg. Foreløpig har vi bare lagt til raketter og kanoner. Weapon-klassen under kan vi riktignok bruke, men det vil være mer hensiktsmessig å *spesialisere*[1] den slik at vi enkelt kan legge til flere våpentyper i framtiden.

```
class Weapon:
    def __init__(self, weapon_type, cannon_shots=0, rockets=0, pos=None):
        self.weapon_type = weapon_type
        self.cannon_shots = cannon_shots
        self.rockets = rockets
        self.pos = pos

    def aim(self, target):
        """Returns a vector towards the target"""
         # placeholder, you don't need to change this method
        return []

    def fire(self, target):
        target_vector = self.aim(target)
        if self.weapon_type == "cannon":
            if self.cannon_shots > 0:
                print("Booom")
                objects.append(CannonBall(self.pos, target_vector))
                self.cannon_shots -= 1
        elif self.weapon_type == "rocket_launcher":
            if self.rockets > 0:
                print("Wooosh")
```

a) Bruk spesialisering av Weapon-klassen for å forbedre koden. Beskriv hvordan du ønsker å gjøre det og hvordan du bruker arv.

b) Implementer koden. Du trenger ikke å skrive noe av støttekoden rundt Weapon-klassen. Du trenger heller ikke å implementere `aim` (bare angi hvor du vil ha den).

[1] Tenk i forhold til *generaliseringen* vi gjorde i oppgave 1

Del B

Oppgave 4 (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.

Oppgave 5 (10%)

```
class DefaultVar(object):
    def __init__(self, var1, var2 = 42, var3 = 22):
        self.var1 = var1
        self.var2 = var2
        self.var3 = var3
        print("Init with v1", self.var1, "v2", self.var2, "v3", self.var3)
```

Gitt koden over, hva vil programmet skrive ut for hvert av uttrykkene under? Du trenger ikke angi eksakt hva Python skriver, vi ønsker å se at du forstår hva som skjer.

a) `dv1 = DefaultVar(100)`

b) `dv2 = DefaultVar()`

c) `dv3 = DefaultVar(100, var3 = 900)`

Oppgave 6 (10%)

```
class EvenOnly(list):
    def append(self, integer):
        if not isinstance(integer, int):
            raise TypeError("Only integers can be added")
        if integer % 2:
            raise ValueError("Only even numbers can be added")
        super().append(integer)

e = EvenOnly()
```

Gitt koden over, hva vil programmet skrive ut for hvert av uttrykkene under? Du trenger ikke angi eksakt hva Python skriver, vi ønsker å se at du forstår hva som skjer.

a) `e.append(42)`

b) `e.append(43)`

c) `e.append('hei')`

d) `e.append(3/0)`