

# EKSAMENSOPPGAVE

**Eksamen i:** INF-1400 Objektorientert Programmering  
**Dato:** Tirsdag 26. mai 2015  
**Tid:** Kl 09:00 - 13:00  
**Sted:** Åsgårdveien 9

**Tillatte hjelpemidler:** Ingen.

**Oppgavesettet er på 4 sider inklusiv forside.**

**Kontaktperson under eksamen:** John Markus Bjørndalen.  
**Telefon:** 90148307

**NB! Det er ikke tillatt å levere inn kladd sammen med besvarelsen**

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

Romskipet Dawn har fotografert flere lyse punkter på miniplaneten Ceres. Mens vi venter på at den skal komme enda nærmere underholder vi oss selv med å lage et enkelt spill. Siden konspirasjonsteoretikere allerede har bestemt seg for at lysene er fra romvesener lager vi et rom-invasjonsspill.

Det er selvfølgelig helt utenkelig at romvesener er fredelige.

Vi skal bare skissere deler av spillet siden vi har begrenset tid. Derfor skal dere ikke utvikle mer av spillet enn det som er angitt i oppgavene under — i hvert fall ikke før dere har forlatt eksamenslokalet.

### Oppgave 1 - 25%

Vi har følgende klasser som representerer romskip:

- Jagere - som har `n_kanoner`, `n_skudd`, `n_raketter`, `skjold`, `skade`
- Hangarskip - som har `n_kanoner`, `n_skudd`, `n_raketter`, `skjold`, `skade`, `skip` (liste over skip som har landet)
- Fraktskip - som har `n_kanoner`, `n_skudd`, `n_raketter`, `skjold`, `skade`, `last` (liste over ting som skipet frakter)

Hvor `n_` brukes for attributter som angir antall. `n_skudd` og `n_raketter` angir hvor mange skudd og raketter som er igjen. `skjold` angir hvor mye skjold som er igjen (romskip tar bare 50% skade når skjoldene er oppe) og `skade` angir "hitpoints" som er igjen for romskipet.

De forskjellige romskipene vil også ha egenskaper som vi foreløpig ignorerer (maks antall skudd, hastighet, maks skade og så videre).

Vi har kun angitt attributter i denne oppgaven, så metoder trenger du foreløpig ikke tenke på.

a) Generaliser klassene over ved å introdusere en ny klasse (`Romskip`). Tegn opp klassehierarkiet for den nye klassen og klassene som er angitt over. Du trenger ikke å skrive kode.

## 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 klasseattributen `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*<sup>1</sup> 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")
```

---

<sup>1</sup> Tenk i forhold til *generaliseringen* vi gjorde i oppgave 1

- a) Bruk spesialisering av Weapon-klassen for å forbedre koden. Angi 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).

## Del B

### Oppgave 4 - 15 %

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

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

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

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

    def foo(self, val):
        print('B', 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 -15%

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

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