UiT
NORGES
ARKTISKE
UNIVERSITET

SENSORVEILEDNING

For eksamen i: **INF-1400 Objektorientert programmering**

Dato: **Torsdag 28. september 2017**

Sensorveiledningen er på 10 sider inklusiv forside

Fagperson/intern sensor: Lars Brenna

Telefon: 90786723

Eksamen INF-1400
Objektorientert programmering
Høst 2017

Løsningsforslag

Eksamenssettet består av to deler, totalt fire oppgaver.

Les oppgaveteksten grundig og disponer tiden slik at du får tid til å svare på alle oppgavene. I noen oppgaver kan det være nødvendig å tolke oppgaveteksten ved å gjøre noen antagelser - gjør i så fall rede for hvilke antagelser du har gjort, men pass på å ikke gjøre antagelser som trivialiserer oppgaven.

Merk: Fokus i oppgaven er objektorientering og bruk av datastrukturer vi bygger opp. Det gis ikke ekstra poeng for å utvide oppgaven utover det som er spesifisert.

Pseudokode er i de fleste tilfeller godtatt, det kreves ikke kjørbar kode. Python3-syntaks er foretrukket, men ikke strengt nødvendig.

Løsningsforslag

Kode i løsningsforslaget er basert på Python3, men må anses som en skisse og ikke som fullstendig, kjørbar kode. Skissen bør imidlertid være tilstrekkelig for å illustrere omfang og korrekthet på løsning.

Syntaks fra andre OO språk er også godtatt.

Del 1

Selskapet SpaceX produserer og sender opp raketter bygd av gjenbrukbare komponenter. Noen flyvninger plasserer satellitter i bane, andre frakter last til den internasjonale romstasjonen (ISS).

Dagens rakettmodell kalles "Falcon9" og er bygd opp av flere komponenter, også kalt «steg». Første steg er en stor rakett med mye løftekraft. Etter ca 160 sekunder er brennstoffet i første steg brukt opp, og steget kobles fra. Da overtar andre steg, og fører lasten videre. Når ønsket omløpsbane er nådd, slukkes andre steg og kobles fra. Segmentet som sitter på topp er enten en romkapsel («Dragon») eller et rent lasterom, og fortsetter så alene.

Rakettene kan bygges i ulike konfigurasjoner, tilpasset en flyvnings formål, last og lengde. Nye komponenter produseres i høy takt, men for å redusere kostnadene ved oppskytning må rakettens steg/komponenter kunne lande trygt og brukes igjen. Det er ønskelig at en komponent er klar til ny oppskyting innen 24 timer fra landing. Komponenter lander på en av de to autonome droneskipene «Just Read the Instructions» og «Of Course I Still Love You», eller de eksploderer underveis eller i landingen.

SpaceX trenger et system for å holde rede på raketter og komponenter.

Vi definerer følgende klasser:

- `Dragon`: Representerer en romkapsel som kan frakte folk og/eller last. Attributter er:
  - Seter: antallet seter i kapselen.
  - Posisjon: verdi «I luften», «på bakken», «på skip», eller «ISS».
  - Status: verdi «aktiv», «ødelagt», «landet», eller «klargjort».
- `Lasterom`: Representerer et lasterom. Attributter er:
  - Lastekapasitet: maksimalt volum last som kan medbringes.
  - Posisjon: verdi «I luften», «på bakken», «på skip», eller «ISS».
  - Status: verdi «aktiv», «ødelagt», «landet», eller «klargjort».
- `Motor`: Et steg kan ha en eller flere motorer. Denne klassen representerer én spesifikk motor, kalt "Merlin". Attributter er:
  - Skyvekraft: konstanten 934 kiloNewton
  - Posisjon: verdi «I luften», «på bakken», «på skip», eller «i testing».
  - Status: verdi «aktiv», «ødelagt», «landet», eller «klar».
- `Steg`: Attributter er:
  - Motorer: Liste over motorer.
  - Posisjon: verdi «I luften», «på bakken», «på skip», eller «ISS».
  - Status: verdi «aktiv», «ødelagt», «landet», eller «klargjort».

Oppgave 1a - 15%

Alle klassene har felles attributter. For å generalisere dette ønsker vi legge til en ny klasse `Komponent`. En komponent er enten et steg, et lasterom eller en Dragon. Det er kun steg som har motorer.

Definer klassen `Komponent` og endre de andre klassene slik at de arver `Komponent`. Forklar kort (i tekst) hvordan du har brukt arv her.

Det er kun behov for å fokusere på arv og attributtene til klassene siden vi skal jobbe med metodene senere.

Løsningsforslag

Her forventes det:

1. En kort forklaring på hva arv er, og hvordan det er brukt her.
2. En definisjon på den nye klassen, og de andre:

```
class Komponent:
    STATUS_KONST = {"aktiv"=0, "odelagt"=1, "landet"=2, "klargjort"=3}
    POSISJON_KONST = {"i_luften"=0, "pa_bakken"=1, "pa_skip"=2, "ISS"=3

    def __init__(self, status, posisjon):
        if status in STATUS_KONST.values():
            self.status = status
        if posisjon in POSISJON_KONST.values():
            self.posisjon = posisjon

class Steg(Komponent):
     def __init__(self, motorer, status, posisjon):
        Komponent.__init(self, status, posisjon)
        self.motorer = motorer # en liste

class Lasterom(Komponent):
    def __init__(self, status, posisjon, lastekapasitet):
        Komponent.__init(self, status, posisjon)
self.lastekapasitet = lastekapasitet

class Dragon(Komponent):
    def __init__(self, seter, posisjon, status)
        Komponent.__init(self, status, posisjon)
        self.seter = seter
```

Vi godtar både Python2 og 3 syntaks, samt evnt andre språk. Validering og potensielt exceptions er nødvendig for full score.

Oppgave 1b - 15%

Tegn opp klassehierarkiet for den nye klassen `Komponent` og klassene som er definert over.

Løsningsforslag

En tegning av klassehierarkiet.

Oppgave 1c - 5%

I 2010 lanserte SpaceX rakett-typen «Falcon9» bestående av ett første steg med ni Merlin-motorer, et andre steg med én Merlin-motor, og ett nyttelast-steg. Innen få år planlegger de å transportere også mennesker, til ISS eller til månen, og det endelige målet er å bygge en koloni på Mars. I november 2017 har SpaceX derfor planlagt

første oppskytning av raketten «FalconHeavy», som er mye sterkere enn "Falcon9". "FalconHeavy" består av tre første-steg montert i parallel (totalt 27 Merlin-motorer), ett andre steg som er identisk med Falcon9, og et nyttelast-steg som er større enn på Falcon9.

Definer klassen `Rakett`, med de med attributter du mener den trenger. Klassen må kunne brukes til begge typer raketter. Typen rakett kjennetegnes ved komponentene den er sammensatt av.

Løsningsforslag

```
class Rakett:
    def __init__(self):
        self.komponentListe = []
```

Oppgave 1d - 25%

Livsløpet til en rakett er at den settes sammen av de ulike komponentene på bakken og deretter splittes den opp i luften etter oppskytning.

Før oppskyting må komponenter legges til, og status på raketten settes til "klar" når rett antall komponenter er "klargjort". Status sjekkes etterhvert som komponenter legges til.

Etter flygning, det vil si når alle steg står i ro, skal rakett-objektet arkiveres med rakettstatus «vellykket» (det vil si at alle komponenter har statusen «landet») eller rakettstatus «feilet» (det vil si at en eller flere komponenter har status «ødelagt»). Vi trenger da en metode `sjekkRakettStatus()` som kan kalles når raketten står i ro, og gå igjennom alle komponentene og oppdatere rakettens tilstand basert på alle komponentenes tilstand.

Ved neste oppskytning av gjenbrukte komponenter lages et nytt `Rakett`-objekt.

Anta at alle attributter er private, og at metoden `arkiverRakett(rakettStatus)` eksisterer. Alt annet utenfor klassen `Rakett` som trengs for at dette fungerer, kan også antas å eksistere.

For klassen `Rakett`, implementer metodene `__init__()`, de `get/set`-metoder som er nødvendige, og metoden `sjekkRakettStatus()`.

Løsningsforslag

Vi vil se at kandidaten kan legge til en liste av komponenter, og bruke get/set med private variabler.

```
class Rakett:
    RAKETTSTATUS_CONST = {"klargjort"=0,"vellykket"=1, "feilet"=2}
    __komponentListe = []
```

```
    def __init__(self, komponentListe):
        self.__komponentListe = komponentListe

    def leggTilKomponent(self, komponent):
        if komponent.status == Komponent.STATUS_CONST["klargjort"]:
            self.__komponentListe.add(komponent)
            if isFalcon9Ready():
                print "Raketten␣Falcon9␣er␣klar!"
            elif isFalconHeavyReady():
                print "Raketten␣FalconHeavy␣er␣klar!"
        else:
            print "komponent␣er␣ikke␣klargjort␣for␣ny␣oppskyting"

    def isFalcon9Ready(self):
     # maa ha 3 komponenter:
     if self.__komponentListe.length()!=3:
         return False
     # maa ha 2 steg og 1 lasterom eller Dragon
     stegTeller = 0
     nytteLastTeller = 0
     for komponent in self.__komponentListe:
         if komponent.__type__ == Dragon:
             nytteLastTeller+=1
         elif komponent.__type__ == Lasterom:
             nytteLastTeller+=1
         elif komponent.__type__ == Steg:
             stegTeller += 1
        if stegTeller == 2 and nytteLastTeller == 1:
            return True
        else:
            return False

    def isFalconHeavyReady(self):
        # maa ha 5 komponenter:
     if self.__komponentListe.length()!=5:
         return False
     # maa ha 4 steg og 1 lasterom eller Dragon
     stegTeller = 0
     nytteLastTeller = 0
     for komponent in self.__komponentListe:
         if komponent.__type__ == Dragon:
             nytteLastTeller+=1
         elif komponent.__type__ == Lasterom:
             nytteLastTeller+=1
         elif komponent.__type__ == Steg:
             stegTeller += 1
        if stegTeller == 4 and nytteLastTeller == 1:
            return True
```

```
        else:
            return False


    def hentKomponentListe():
        return self.__komponentListe

    def sjekkRakettStatus():
        rakettStatus = RAKETTSTATUS_CONST["vellykket"]
        for komponent in komponentListe:
            if komponent.status == Komponent.STATUS_CONST["oedelagt"]:
                rakettStatus = RAKETTSTATUS_CONST["feilet"]
                break

       arkiverRakett(rakettStatus)
```

Oppgave 2 - 20%

Når man ønsker å endre et objekts verdier, er bruk av `get/set`-metoder et vanlig design. Diskuter fordeler og ulemper med `get/set`-metoder, sett i forhold til å bruke offentlig tilgjengelige (public) attributter.

Løsningsforslag

Her forventes det en diskusjon rundt tilgang til attributter, utvikling av grensesnitt over tid og gjenbruk av komponenter.

Oppgave 3 - 15%

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

    def foo(self, value):
        print ("A", value)

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

class C(B):
    def __init__(self, multiplier=2):
        super().__init__()
        self.multiplier = multiplier

    def foo(self, value):
        print ("C", value*multiplier)

a = A()
b = B()
c = C()
d = C(3)

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

a) Hva er polymorfi?

b) Angi hva programmet over skriver ut.

Løsningsforslag

a) Her forventes det en forståelse av skillet mellom grensesnitt og implementasjon. At flere ulike implementasjoner kan holde den samme "kontrakten", det være seg gjennom arv eller overstyring. I eksempelet over ser vi både arv og overstyring av foo i C.

b) "A 1", "A 2", "C 6", "C 9"

Oppgave 4 - 15%

Så langt på denne eksamen har vi nevnt innkapsling (encapsulation), arv (inheritance), og polymorfisme. Et fjerde konsept innen objektorientering er abstraksjon.

Forklar hva abstraksjon er i objektorientert programmering, gjerne (men ikke nødvendigvis) ved hjelp av et eksempel.

Løsningsforslag

Abstraksjon handler om å skjule implementasjonsdetaljer, gjerne gjennom et grensesnitt eller arv. Objekter kan brukes som en delt abstraksjon for kode og data. Abstraksjon tilrettelegger for gjenbruk av kode og data, og muliggjør at implementasjon kan endres uten å bryte kontrakter (grensesnitt brukt av andre klasser).