EKSAMENSOPPGAVE I INF-1400

Eksamen i	:	INF-1400 – Objektorientert Programmering
Eksamensdato	:	2011-05-26
Tid	:	09.00-13.00
Sted	:	Åsgårdveien 9
Tillatte hjelpemidler	:	Ingen.

Oppgavesettet er på 5 sider inkl. forside

Kontaktperson under eksamen:

John Markus Bjørndalen, 92671202

FAKULTET FOR NATURVITENSKAP OG TEKNOLOGI
Universitetet i Tromsø

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.

Del A, døden fra rommet

Året er 2234, og vi har erobret solsystemet. Nesten. I motsetning til håpefulle drømmer om feriekolonier på Mars og Europa har vi bare tre romstasjoner og en liten koloni på månen. Romfartøyene som reiser rundt i solsystemet er hovedsakelig forskningsfartøyer og enkelte kommersielle fartøy som driver gruvedrift på forsøksstadiet. Alle er ubemannede.

Og nå har vi et problem.

En feil i et gammelt program fra 2014 har gjort at vi har feilberegnet banen til en komet. I stedet for å suse forbi jorden nær nok til å lage pene farger på himmelen kommer den til å treffe midt på. Flere milliarder mennesker må evakueres bare for å unngå den verste flodbølgen, men tapstallene vil likevel bli store. Langtidsvirkningene vil være enorme.

Heldigvis oppdaget vi feilen tidsnok til at vi klarer å sende noen av forsknings- og gruvefartøyene til kometen for å skyve den ut av banen. Operasjonen vil være vanskelig siden det er en komet: den består av is og stein, så vi må skyve forsiktig på flere steder samtidig. Skulle den knekke opp vil heldigvis tyngdekraften hjelpe oss. Skyver vi de største fragmentene ut av banen vil de mindre fragmentene trekkes etter.

Vi må gjøre en koordinert operasjon med flere fartøy, og vi trenger å overvåke operasjonen underveis. Til det trenger vi et program for å holde styr på fartøyene våre samt kometen og fragmenter av den.

Definisjon av oppgaven

Vi definerer følgende klasser:

- KometFragment – fragmenter av kometen. Disse har attributtene *posisjon* og *størrelse*.
- Komet – representerer kometen som en samlet enhet og inneholder en liste over *fragmentene* som tilhører kometen. Hvis kometen ikke har brutt opp enda vil listen inneholde ett stort fragment som tilsvarer hele kometen.
- Operasjon – som inneholder en liste over *romskipene*.
- SensorRomskip – romskip som har sensorer (kamera, radar osv) som vi trenger for å overvåke operasjonen. Romskipet vil ha nåværende *posisjon* og *målposisjon* (stedet den har fått beskjed om å bevege seg til for å bruke sensorene effektivt).
- SkyveRomskip – romskip som er i stand til å skyve på kometen eller fragmentene. Romskipet vil ha attributtene nåværende *posisjon* og en referanse til fragmentet den skal skyve på (*fragment*). I tillegg vil den også ha en *målposisjon* for tilfeller hvor den holdes unna de andre og avvente nye mål.

For enkelthets skyld antar vi at posisjonene er av typen Posisjon som allerede eksisterer og at størrelse angis med et flyttall.

Merk fokus er på objektorientering og bruk av datastrukturene vi bygger opp. Det er ikke anbefalt å prøve å utvide oppgaven på eksamen (som for eksempel å angi hvor på fragmentene eller kometen man skal skyve eller å legge til farts- og skyvekraftvektorer).

Oppgave 1 (10%)

Vi har to typer romskip med noen ting til felles. Dette gjør at vi ønsker å generalisere ved å introdusere arv og legge til en ny klasse Romskip.

Forklar kort hvordan du kan bruke arv når du legger til Romskip og hvordan du endrer de andre Romskip-klassene. Det er kun behov for å fokusere på arv og på attributtene til klassene siden vi skal jobbe med metodene senere.

Oppgave 2 (15%)

Implementer klassene vi har spesifisert til nå. Du trenger ikke å ta med noen andre metoder enn __init__() siden vi skal jobbe videre med klassene senere.

Oppgave 3 (20%)

a) *Tegn opp klassediagrammet for klassene dine.* Du trenger ikke å ta med metoder.
b) *Tegn opp datastrukturen med et lite antall romskip av begge typer og et lite antall kometfragmenter.* Du trenger ikke å angi verdier for attributter som posisjon og størrelse. Noen av skyveromskipene skal skyve på noen kometfragmenter.

Oppgave 4 (15%) - formasjonsflyging

Vi må ha et system for å plassere ut SensorRomskip på angitte steder rundt kometen. Vi skal implementere en metode som ser ut slik:

```
def plasserSensorRomskip(self, posisjoner):
        ….kode….
```

Posisjoner er en Python-liste med de posisjonene (av type Posisjon) som vi skal sende SensorRomskip til.

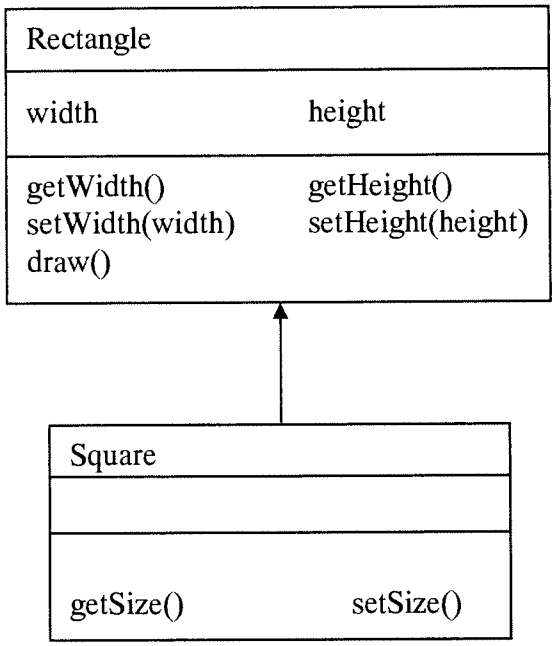
plasserSensorRomskip skal gå gjennom romskiplisten som ligger i Operasjon og finne alle SensorRomskip i denne listen. For hver posisjon i posisjoner skal den velge ut et sensorromskip og kalle settMålPosisjon(posisjon) for å sette et nytt mål.

For SkyveRomskip skal vi også ha en settMålPosisjon(), men den trenger å gjøre litt mer enn sensorromskipets settMålPosisjon(). Når skyveromskipet blir sendt til en posisjon skal det samtidig avbryte skyvejobben den eventuelt er i gang med. Dette kan vi gjøre ved å sette fragment-attributten i skyveromskipet til None.

Angi hvilken klasse du vil plassere plasserSensorRomskip i og skriv koden for metoden. Skriv også koden for de to versjonene av settMålPosisjon og angi hvilke klasser du vil legge dem i.

Del B

Oppgave 5 (10%)



I figuren over har vi angitt to klasser, Square og Rectangle.

a) *Hvorfor har ikke Square-klassen noen attributter?*
b) Vi har implementert Square-klassen ved hjelp av er-forhold (is-a-relationship). Vi kan også implementere implementere klassen ved hjelp av har-forhold (has-a-relationship). *Hvordan vil du gjøre det?* Det holder å skrive noen linjer om hvordan du endrer klassene. Du trenger ikke å opprettholde typeidentitet.

Oppgave 6 (15%)

Velg to av kulepunktene under og gi en kort beskrivelse av uttrykkene/konseptene som er angitt der.

- object vs. class
- instance
- method
- encapsulation (innkapsling)
- mutable vs. immutable
- polymorphism

Oppgave 7 (15%)

Som en del av arbeidet med å rette opp i feilene fra den gamle kildekoden fant vi følgende historiske kodesnutt fra 2019.

```
class Sneetch:
    def __init__(self):
        self._a = 3
        self._b = 4

    def x(self):
        print self._a

    def y(self):
        print self._b

class StarBellySneetch(Sneetch):
    def __init__(self):
        Sneetch.__init__(self)
        self._b = 7
        self._c = 8

    def y(self):
        print self._b, self._c

    def z(self):
        print self._a, self._c

alice = Sneetch()
bob = StarBellySneetch()
```

For hvert av uttrykkene under, angi hva programmet vil skrive ut. Uttrykkene kan også utløse en feil (error) i programmet. Hvis dette skjer, angi hvorfor feilen oppsto.

a) alice.x()
b) alice.y()
c) alice.z()
d) bob.x()
e) bob.y()
f) bob.z()