UNIVERSITETET I TROMSØ UiT
FAKULTET FOR NATURVITENSKAP OG TEKNOLOGI

INF-1400 EKSAMENSOPPGAVE

Eksamen i: INF-1400 Objektorientert programmering

Dato: Torsdag 24. mai 2012

Tid: Kl 09:00 – 13:00

Sted: Åsgårdvegen 9

Tillatte hjelpemidler: Ingen

Oppgavesettet er på 5 sider inkl. forside

Kontaktperson under eksamen: John Markus Bjørndalen

Telefon: 92671202

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 norske termer som er brukt i stedet for de engelske fra boka:
- Klasse – class
- Arv – inheritance
- Atributt – Attribute
- Metode - Method

Del A, Nobina tar over

Nobina har tatt over bussdriften i Tromsø, og som vanlig er når man gjør større omlegginger dukker utfordringer opp.

Vi skal nå jobbe med et program som hjelper oss å holder oversikt over forsinkelser og andre utfordringer. Vi *forenkler også problemene* slik at vi ikke trenger å forholde oss til virkelige praktiske problemer (som f.eks. at bussene ikke starter og avslutter alle ruter på samme sted).

Merk at 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 legge flere detaljer enn nødvendig i klassene).

Definisjon av oppgaven

Vi definerer følgende klasser:
- Busspark – har en liste over *busser.*
- Buss – en buss skal ha en liste over *hendelser*, som kan være forsinkelser eller innstillinger. Den har også en en referanse til *ruten* den kjører i øyeblikket.
- Forsinkelse – siden en buss kan kjøre flere ruter må vi registrere følgende: *rute*, *tidspunkt* og hvor mange *minutter* forsinkelsen er på.
- Innstilling – her registrerer vi *rute* og *tidspunkt.*
- Rute - har et *rutenummer* (vi trenger ikke flere detaljer i denne oppgaven)

I oppgavene under er selve oppgaven angitt i **fet tekst (boldface)**, mens forklarende tekst er angitt som normal tekst.

Oppgave 1 (15%)

Noen av bussene kan ikke kjøre effektivt med kjetting. En del av forsinkelsene skjer når disse bussene kjører ruter med mange bakker. For å skaffe oss bedre oversikt ønsker vi å skille mellom busstyper. Vi skal derfor introdusere to nye klasser *ElBuss* og *LangBuss* hvor langbussene er de eneste som kan kjøre med kjetting hele dagen.

Bussene trenger nå en metode som returnerer *True* for busser som kan bruke kjetting. Vi kaller denne metoden *kanBrukeKjetting()*.

Den andre endringen vi skal gjøre er å generalisere *Forsinkelse* og *Innstilling* slik at det blir enklere å jobbe med dem. Vi gjør dette ved å introdusere en ny klasse *Hendelse*.

Alle hendelses-objekter skal ha en metode vi kan kalle for å sjekke om forsinkelsen er lengre enn et angitt antall minutter, denne kaller vi *forsinketLengreEnn(minutter)*. For innstillings-objekter vil denne bestandig returnere *True*, mens for forsinkelses-objekter må vi sammenligne de angitte minuttene med hvor mange minutter som er angitt i forsinkelses-objektet.

Forklar kort ved hjelp av et klassediagram hvordan du kan bruke arv når du legger til

a) **Hendelse og**
b) **de nye Buss-klassene.**

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

Oppgave 2 (10%)

I oppgaven over brukte vi to måter å endre på klassehierarkiet på.

a) **Generalisering var den ene, hva er navnet på den andre?**
b) **Hva er forskjellen på disse to? Vi trenger bare en kort forklaring av hovedforskjellen, ikke en lang utdypende forklaring.**

Oppgave 3 (15%)

Implementer klassene vi har spesifisert i oppgave 1. Du trenger ikke å ta med noen andre metoder enn __init__() og metodene vi spesifiserte i oppgave 1 siden vi skal jobbe videre med klassene senere.

Oppgave 4 (10%)

Tegn opp en datastruktur med et lite antall busser, ruter, forsinkelser og innstillinger. Du trenger ikke angi verdier for alle attributtene i alle objektene.

Oppgave 5 (15%)

Fylkeskommunen trenger en oversikt over hvor mange forsinkelser og innstillinger det har vært på en gitt dag. Som en forenkling antar vi at bussene kun har registrert hendelser fra siste døgn, slik at vi ikke trenger å sjekke datoene.

Metoden *finnHendelser(self, minForsinkelse)* skal lage en liste over alle hendelsene som har skjedd siste døgn som har ført til en forsinkelse over «minForsinkelse» minutter. Metoden skal returnere en tuple som har følgende informasjon:

(antall innstillinger, antall forsinkelser over «minForsinkelse» minutter,
liste over alle hendelser som førte til forsinkelser over «minForsinkelse»).

En Innstilling vil automatisk være en forsinkelse over N minutter for alle mulige valg av N.

Lag metoden finnHendelser. Angi hvilken klasse du vil legge den i.

Del B

Oppgave 6 (15%)

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

1. object vs. class
2. parameter
3. instance
4. method
5. mutable vs. immutable

Oppgave 7 (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?

```
a) dv1 = DefaultVar(100)
b) dv2 = DefaultVar()
c) dv3 = DefaultVar(100, var3 = 900)
```

Oppgave 8 (10%)

```
class Vector(object):
    def __init__(self, u = 0, v = 0):
        self.u = u
        self.v = v

    def getU(self):
        return self.u

    def setU(self, u):
        self.u = u

    def normalize(self):
        factor = math.sqrt(self.u ** 2 + self.v ** 2)
        self.u /= factor
        self.v /= factor

    def __str__(self):
        return "<%f, %f>" % (self.u, self.v)
```

a) **Hva er forskjellen på en accessor og en mutator?**
b) **Gitt koden over, angi hvilke metoder som er accessor og mutator.**