Informasjon

Prøveeksamen IN1000 våren 2022

Tid 4 timer (samme som eksamen våren 2022)

Kopi av digital skoleeksamen fra H2019. Det anbefales at du forsøker å løse denne med realistiske omgivelser. Det vil si med kun tillatte hjelpemidler og tidskontroll. Det kan også være lurt å gjøre dette på en fysisk maskin som har et tastatur som ligner mest mulig på maskinene i Silurveien. Slik unngår du vanskeligheter med et tastatur som er forskjellig fra det du er vant med. Mer info om dette i Mattermost.

Oppgavene

Oppgavene under **1 og 2** er kortsvarsoppgaver som rettes automatisk. Pass på å skrive inn nøyaktig svar med riktig type (ikke desimalpunktum eller -komma for heltall). Tekster kan skrives inn direkte (som her) eller med doble anførselstegn ("som her"), men ikke med enkle anførselstegn ('som her').
Dersom du mener en programmeringsoppgave er uklar, kan du gjøre egne forutsetninger og beskrive disse. Du kan også legge til egne funksjoner eller metoder ved behov, med begrunnelse. Videre kan du bruke løsninger fra tidligere deloppgaver selv om du selv ikke har skrevet disse.

Tillatte hjelpemidler

Alle trykte og skrevne hjelpemidler. Ingen elektroniske.

1 Oppgave 1a

Hva er verdien til variabelen tall etter at følgende kode er utført?

```
tall = 1+2+3
tall = tall*2 - 1
```

(11)

Maks poeng: 1

2 Oppgave 1b

Hva er verdien til variabelen tekst etter at følgende kode er utført?

```
tall = 4*2
t1 = "a"
if tall>9:
    t1 = t1 + "a"
tekst = "b" + t1
```

(ba, ba)

Maks poeng: 1

3 Oppgave 1c

Hva skrives ut til terminalen når følgende kode kjøres?

```
if (5>7) or (2>1):
    print("ja")
else:
    print("nei")
```

(ja, ja)

Maks poeng: 1

4 Oppgave 1d

Hva skrives ut til terminalen når følgende kode kjøres?

```
liste = [1,2,-3,4,-5,6]
total=0
i = 0
while liste[i]>0:
    total += liste[i]
    i += 1
i=len(liste)-1
while liste[i]>0:
    total += liste[i]
    i -= 1
print(total)
```

(9, 9)

Maks poeng: 3

5 Oppgave 1e

Hva skrives ut til terminalen når følgende kode kjøres?

```
a=3
while a<25:
    a = a*2
    for b in [1,2,3]:
        a += b
print(a)
```

(30, 30)

Maks poeng: 3

6 Oppgave 1f

Hva skrives ut til terminalen når følgende kode kjøres?

```
tall = 0
operasjoner = "idd.d..did"
for operasjon in operasjoner:
    if operasjon == "d":
        tall = tall*2
    elif operasjon == "i":
        tall = tall+1
    else:
        tall = 0
print(tall)
```

(2, 2)

Maks poeng: 3

7 Oppgave 1g

Hva skrives ut til terminalen når følgende kode kjøres?

```
bok = {"a":[3,4,5], "b":[6,7]}
bok["a"].append(8)
print( len(bok["a"]) )
```

(4, 4)

Maks poeng: 2

8 Oppgave 2a

Hva skrives ut til terminalen når følgende kode kjøres?

```
def funk(a, b):
    return a*b

c = funk(2+3, 2) * 4
print(c)
```

(40, 40)

Maks poeng: 2

9 Oppgave 2b

Hva skrives ut til terminalen når følgende kode kjøres?

```
def funk2(a):
    return a*3

b = 2
c = funk2( funk2(b+1) )
print(c)
```

(27, 27)

Maks poeng: 2

10 Oppgave 2c

Hva skrives ut til terminalen når følgende kode kjøres?

```
class A:
    def __init__(self, a, b):
        self._c = b
        self._d = a
    def sett_c(self, ny):
        self._c = ny
    def hent_sum(self):
        return self._c + self._d

a1 = A(1,2)
a2 = A(3,4)
a1.sett_c(5)
print( a1.hent_sum() * a2.hent_sum() )
```

(42, 42)

Maks poeng: 3

11 Oppgave 2d

Hva skrives ut til terminalen når følgende kode kjøres?

```
class A:
    def __init__(self, verdi):
        self._verdi = verdi
    def hent_okt(self):
        return self._verdi+1

class B:
    def __init__(self, a):
        self._a = a
    def hent_modifisert(self):
        return self._a.hent_okt() * 2

def hovedprogram():
    a1 = A(5)
    b1 = B(a1)
    print(b1.hent_modifisert())

hovedprogram()
```

(12, 12)

Maks poeng: 3

12 Oppgave 3a

Knut vil beregne pris inkludert frakt når han skal handle på nettet.

Skriv en funksjon **pris_inkl_frakt(varepris)**.
Dersom varepris er over 1000 kroner legges det ikke på noe kostnad til frakt og funksjonen skal returnere samme varepris som den fikk inn som argument. Dersom varepris er mellom 500 og 1000 kroner (fra og med 500, til og med 1000) skal det legges på frakt, slik at funksjonen returnerer en pris som er 50 kroner høyere enn varepris. Dersom varepris er under 500 kroner skal det både legges på frakt og ekstragebyr, slik at funksjonen skal returnere en pris som er 80 kroner høyere enn varepris.

Med andre ord skal f.eks. kallet pris_inkl_frakt(300) evaluere til 380, pris_inkl_frakt(600) evaluere til 650, og pris_inkl_frakt(1300) evaluere til 1300.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

13 Oppgave 3b

Per vil lage seg en handleliste og raskt kunne fjerne utsolgte varer fra denne lista.

Skriv en funksjon **fjern_utsolgte(handleliste, utsolgte).**
Funksjonen tar som argumenter en liste med navn på varer (handleliste: liste av streng-verdier) og en liste over utsolgte varer (utsolgte: liste av streng-verdier). Funksjonen skal returnere en ny liste som inneholde de ønskede varene som ikke er utsolgte, dvs en ny liste som inneholder alle streng-verdier som finnes i listen handleliste, men ikke finnes i listen utsolgte. I tillegg skal funksjonen skrive til terminalen navnet på hver vare fra handlelisten som er utsolgt.

Altså skal for eksempel kallet fjern_utsolgte(["melk", "brus", "pasta"], ["kanel","brus"]) evaluere til listen ["melk", "pasta"], samtidig som teksten brus skrives til terminalen.

Skriv ditt svar her...

```
1
```

Maks poeng: 5

14 Oppgave 3c

Lise skal reise som backpacker gjennom flere land i asia og vil finne ut hvilke vaksiner hun må ta før reisen.

Skriv en funksjon **samlet_vaksinasjon(krav_hvert_land).**
Funksjonen tar som argument en liste av lister hvor hver indre liste er hvilke vaksiner som trengs for et av landene hun skal reise til. Funksjonen skal returnere en samlet liste over hvilke vaksiner som trengs, dvs en liste av streng-verdier som inkluderer alle vaksinene funnet i et av landene. Dersom samme vaksine finnes i lista for mange ulike land, skal den bare være med én gang i den returnerte lista.

Altså skal for eksempel kallet samlet_vaksinasjon([["difteri","tyfoid"], ["hepatit","difteri"]]) evaluere til en liste ['difteri', 'tyfoid', 'hepatit'] (rekkefølgen av vaksinene i den returnerte lista er uviktig).

Skriv ditt svar her...

```
1
```

Maks poeng: 5

15 Oppgave 3d

Koden under definerer en variabel setning og lager varianter av denne setningen hvor først ordet "en" er fjernet og deretter også ordet "skal" er fjernet. Det er store likheter mellom kodelinjene som lager setning uten ordet "en" og kodelinjene som lager setning uten ordet "skal". Skriv en alternativ versjon av programmet som unngår denne redundansen ved å definere en funksjon forkort_setning og som kaller denne funksjonen to ganger for å lage setning uten henholdsvis ordene "en" og "skal". Skriv det fulle resulterende programmet, inkludert definisjon av funksjonen (og dens parametre), kall på funksjonen (med argumenter) og andre kodelinjer som trengs for å oppnå samme funksjonalitet som den opprinnelige koden under.

```
setning = "en krabbe skal en dag ut av skallet "

#fjerner alle ord "en":
setning_v2 = ""
for ord in setning.split():
    if not ord=="en":
        setning_v2 = setning_v2 + ord + " "

#fjerner alle ord "skal":
setning_v3 = ""
for ord in setning_v2.split():
    if not ord=="skal":
        setning_v3 = setning_v3 + ord + " "

print(setning_v3) #krabbe dag ut av skallet
```

Skriv ditt svar her...

```
1
```

Maks poeng: 5

16 Oppgave 4a

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave a) fra vedlegget her:

```
1
```

Maks poeng: 5

17 Oppgave 4b

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave b) fra vedlegget her:

```
1
```

Maks poeng: 5

18 Oppgave 4c

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave c) fra vedlegget her:

```
1
```

Maks poeng: 10

19 Oppgave 4d

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave d) fra vedlegget her:

```
1
```

Maks poeng: 2

20 Oppgave 4e

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave e) fra vedlegget her:

```
1
```

Maks poeng: 8

21 Oppgave 4f

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave f) fra vedlegget her:

```
1
```

Maks poeng: 5

22 Oppgave 4g

Denne oppgaven består av en rekke deloppgaver der du skal implementere komponenter av et større system. Om du hopper over en deloppgave er det likevel viktig at du leser hele denne teksten (den ligger samlet som en pdf-fil vedlagt hver deloppgave).

Skriv svar på deloppgave g) fra vedlegget her:

```
1
```

Maks poeng: 15

23 Oppgave 5

Karl vil kunne sikre seg at han er forskånt for å måtte lese et bestemt fy-ord og dets synomer.

Skriv en funksjon **sjekk_om_fyord(setning, fyord, synonym_liste)**.
Funksjonen tar som argument en setning (en streng bestående av flere ord med mellomrom mellom) , et fyord (en streng bestående av ett enkelt ord uten noen mellomrom) og en samling av ulike synonymer (en liste av lister, hvor hver indre liste består av streng-verdier som er enkeltord). Funksjonen skal sjekke om minst ett av ordene i setningen er et synonym med det oppgitte fyordet (ifølge den oppgitte samlingen av synonymer) eller om fyordet i seg selv finnes i setningen. Dersom fyordet eller et synonym av fyordet finnes i setningen skal funksjonen returnere True. Hvis ikke skal funksjonen returnere False.

Altså skal for eksempel kallet *sjekk_om_fyord("spis masse godsaker", "snop", [["saft","lemonade"],["snacks","snop","godsaker"],["mye","masse"]])* returnere True, mens kallet *sjekk_om_fyord("spis masse godsaker", "lemonade", [["saft","lemonade"], ["snacks","snop","godsaker"],["mye","masse"]])* skal returnere False.

Skriv ditt svar her...

```
1
```

Maks poeng: 6

Question 16
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 17
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 18
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 19
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 20
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 21
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.

Question 22
Attached

Oppgave 4 Administrasjon av obligatoriske oppgaver (50 poeng)

Universitetet i Ruritania skal digitalisere sin håndtering av studentenes obligatoriske oppgaver («obliger»), og du skal lage deler av en pilot med et objektorientert design basert på innkapsling, dvs. instansvariable og non-public metoder (navn starter med _) skal kun benyttes i den klassen de er definert. **Det er i mange av deloppgavene nødvendig å bruke metoder fra andre klasser/ deloppgaver. Du kan bruke disse selv om du ikke har skrevet dem.**

Emnene har ulike antall obligatoriske oppgaver som kreves godkjent, og hver oblig i et emne må registreres manuelt hvert semester. En student må ha alle oppgaver gitt i emnet godkjent for å få gå opp til eksamen, men i denne piloten foretas det ikke kontroll av om studenten har fått godkjent forrige oblig før godkjenning av besvarelse for neste oblig. Foruten obliger har et emne registrerte studenter og ansatte rettere. Både rettere og studenter har unike brukernavn.

Innleveringen av besvarelser for en oblig foregår i et annet system som du ikke skal lage. Dette systemet lager en fil for hver oblig, med et entydig navn etterfulgt av «.txt», for eksempel *oblig1.txt*. Filen inneholder en linje per student på følgende format (eksempel med to linjer):
student1 obl1student1.txt
student2

I eksempelet er *student1* et entydig brukernavn og *obl1student1.txt* er navnet på filen der student1s besvarelse ligger. *student2* i eksempelet har ikke levert noen besvarelse på oblig 1.

Hver besvarelse skal vurderes av en retter for emnet. Når fristen for en obligatorisk oppgave er utløpt, fordeles besvarelser på retterne, som vurderer en og en til godkjent (1) eller underkjent (0).

Piloten har funksjonalitet for hente ut en eksamensliste med brukernavn på alle studenter som er kvalifisert for eksamen – dvs har godkjent alle registrerte obliger i emnet.

Oppgave 4a) 5 poeng

Definer en klasse **Emne** (inkludert konstruktør) som holder orden på og administrerer obliger, rettere og studenter i et emne. Konstruktørens parametere gir verdier til emnekode (streng), registrerte studenter (ordbok med Student-objekter) og rettere (liste med Retter-objekter).

Oppgave 4b) 5 poeng

Skriv metoden **administrer** i klassen Emne. Metoden skriver ut emnekoden for emnet og en meny på terminalen før den ber om kommando. Den tar imot følgende lovlige kommandoer fra en bruker:
O: Ny oblig
F: Frist ute, start retting
L: Lag eksamensliste
A: Avslutt

Annen input skal gi feilmelding og nytt forsøk. En kommando bør gjenkjennes selv om det er blanke foran eller bak, og uansett om brukeren skriver liten eller stor bokstav (metodene **strip** og **upper** er nyttige her). For å utføre kommandoene skal metoden kalle på følgende non-public metoder som også ligger i klassen **Emne**:
O -> **_opprettOblig**
F -> **_startRetting**
L -> **_skrivEksamensListe**
Disse metodene skal skrives i senere deloppgaver. De opererer kun på instansvariabler i klassen, og har ingen parametere (annet enn **self**) eller returverdier.

Oppgave 4c) 10 poeng

Skriv en klasse **Student** med instansvariabler for brukernavn og fullt navn (begge får verdi fra parametere til konstruktøren) og en ordbok som skal ta vare på resultater av rettinger, der obligId er nøkkel og resultatet verdien. Klassen har følgende metoder i tillegg til konstruktør:

registrer med parametere **oblig** (en streng som identifiserer en oblig entydig) og **resultat** (et heltall som angir om en besvarelse er vurdert som godkjent). Metoden registrerer hvilket resultat en student har fått på en oblig.

altGodkjent med parameter **antObliger** som angir antall obliger registrert i emnet. Metoden returnerer en boolsk verdi: **True** hvis studenten har fått alle obliger godkjent, **False** om en eller flere obliger mangler eller har et annet resultat enn godkjent (kodet som heltallet 1).

Oppgave 4d) 2 poeng

Skriv klassen **Retter** med instansvariabel retterens brukernavn (parameter til konstruktøren) og metoden:

vurder med en parameter som angir et filnavn for en besvarelse. I denne foreløpige utgaven av piloten ignorerer programmet ditt filen med besvarelsen, og returnerer alltid 1 (godkjent).

Oppgave 4e) 8 poeng

Skriv en klasse **Oblig** med tre instansvariabler: Entydig id for obligen, en leveringsfrist på formen ååmmdd (f. eks. 190906 for 6. september 2019) og om obligen er rettet eller ikke. De to første initialiseres med parametere til konstruktøren. Skriv følgende metoder i klassen Oblig:

klarForRetting med parameter som angir dagens dato. Metoden returnerer **True** hvis fristen er før dagens dato og besvarelsene for obligen ikke allerede er rettet – ellers **False**.

hentBesvarelser leser en fil med oversikt over studenters besvarelser (navn og format på denne filen er beskrevet i oppgave-innledningen) og returnerer en ordbok der en students brukernavn er nøkkel og filnavn med studentens besvarelse er verdi.

Oppgave 4f) 5 poeng

Utvid klassen **Oblig** med en ny metode

fordelRetting tar som parametere en ordbok med studenters besvarelser og en liste med rettere, og lar retterne vurdere én og én besvarelse (bruk metoden **vurder** i klassen **Retter**) slik at alle retterne ideelt sett vurderer like mange. Altså skal ingen retter vurdere mer enn 1 besvarelse mer enn de andre. Resultatene av retternes vurderinger samles i en ordbok der nøkkelverdiene er studentenes brukernavn. Metoden markerer obligen som rettet og returnerer ordboken med resultater.

Oppgave 4g) 15 poeng

Skriv følgende non-public metoder i klassen **Emne**:

_opprettOblig genererer et unikt oblignavn, og ber bruker på terminalen om å oppgi frist på formen ååmmdd. Deretter legges en ny oblig til emnet.

_startRetting ber om dagens dato og sjekker om noen av obligene i emnet har en utløpt frist uten at besvarelsene er rettet. I så fall henter den alle besvarelser for obligen, fordeler dem på rettere og registrerer resultatet av rettingen i Student-objekter (på riktig oblig) for studenter som har levert.

_skrivEksamensListe bygger opp en liste med brukernavn for alle studenter som har fått godkjent alle obliger registrert for emnet, og skriver til sist ut denne på terminal med en passende overskrift.