UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Prøveeksamen i:	IN1010 — Objektorientert programmering
Eksamensdag:	Våren 2020
Oppgavesettet er på 5 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle

Innhold

Gjør dine egne fornuftige antagelser, og skriv disse ned i besvarelsen din der du synes dette er nødvendig, for eksempel hvis noe er uklart.

Det er viktig at alle programmene dine kan kompileres og kjøres. Det er selvfølgelig best om du klarer å lage programmer som virker perfekt, men om du ikke klarer det, er det allikevel viktig at du lager programmer som kompilerer og kjører uten feilmeldinger, selv om det ikke løser oppgaven helt på riktig måte. Om du har kode som bare nesten virker, kommenter den ut og skriv en kommentar med en begrunnelse for hvorfor du har denne koden der og hvilke problemer du opplever som gjør at koden ikke virker. Programmer som leveres men som ikke kompilerer og kjører, vil ikke gi særlig uttelling.

Ikke bruk norske bokstaver (æ, ø, å, små og store) noe sted i programmene dine. Ikke lever .class-filer!

Oppgave 1 Emballasje (vekt 20%)

Emballasjefabrikken *Renpakk* skal lage et nytt datasystem (i Java) for å administrere sine produkter, og du har fått i oppdrag å lage deler av dette systemet.

Alt Renpakk produserer kalles Emballasje, og det er fire hovedtyper: Glassemballasje, Metallemballasje, Plastemballasje og Pappemballasje.

Noe av emballasjen til Renpakk er det pant på, og noe emballasje er nedbrytbar. Dette er egenskaper som kommer i tillegg til de andre egenskapene emballasjen har.

(Fortsettes på side 2.)

Av alle produkttypene i klassehierarkiet produseres det for tiden bare ting av disse tre: liten plastflaske med pant, liten nedbrytbar plastflaske med pant og stor nedbrytbar pappflaske med pant.

Som svar på denne oppgaven med sine to deloppgaver skal du levere en ZIP-fil med tre filer: bildene/skjermdumpene nevnt i 1a og 1b og Java-filen i 1b.

1a Klassehierarki (vekt 8%)

Lever en tegning av klassehierarkiet for de tre produkttypene som er beskrevet over som en besvarelse på denne oppgaven. Inkluder også superklasser og eventuelle grensesnitt. (Selv om et godt klassehierarki kanskje kunne ha tatt med flere klasser, skal du bare ta med de som er nevnt over, ingen andre.)

Lag tegningen i et tegneverktøy du liker, eller tegn for hånd og ta bilde av tegningen med mobilen din.

1b Implementasjon (vekt 12%)

All emballasje har et volum (i kubikkcentimeter) og en tekst (String) som er en produksjonsidentifikator.

Plastemballasje har ingen flere egenskaper enn Emballasje mens Pappemballasje i tillegg har en vekt (i gram).

Når det er pant på en ting, må en kunne vite hvor stor panten er (i antall øre) og en kode (en tekst) som identifiserer returordningen.

Når noe er nedbrytbart, må en kunne vite hvor lenge (hvor mange år) det tar før tingen er gått i oppløsning.

Programmer de tre klassene nevnt i 1a og eventuelle superklasser og grensesnitt. Alle variable i alle klasser skal få verdier i det objektene opprettes. Det er derfor viktig at alle klasser har konstruktører med parametre der disse verdiene kan oppgis, unntatt panten på små flasker som alltid har samme verdi, en konstant med verdi 100 øre. Når en konstruktør utføres, skal det skrives ut i terminalvinduet: «Konstruktøren til klassen Xxx utføres», der Xxx er navnet på klassen.

Skriv også en klasse (en hovedklasse) kalt BrukPant med en `main`-metode som oppretter ett objekt av hver av de klassene det kan lages objekter av. Finn på noen passende verdier til alle variablene i alle objektene. Programmet ditt skal kunne kompileres og kjøres.

Skriv alle klassene i én .java-fil som heter det samme som hovedklassen din (der `main`-metoden er).

Kjør programmet du har laget og ta et bilde eller en skjermdump som viser kjøringen av dette programmet med alle linjene som blir skrevet ut i terminalvinduet.

(Fortsettes på side 3.)

Oppgave 2 Datastruktur (vekt totalt 80%)

I denne oppgaven skal vi utvikle en klasse med en ny form for lagringsstruktur. Klassen skal hete Frekvens og skal kunne lagre inntil 1000 tekster (String-er). Konstruktøren skal ha én parameter: en array med tekstene som skal lagres. Tekstene er alfabetisk sortert.

Klassen Frekvens skal ha tre metoder:

1. void-metoden finnFlest skal finne hvilken tekst som forekommer flest ganger og lagre teksten og antallet forekomster i instansvariabler i Frekvens-objektet.

   (Hvis flere tekster forekommer like mange ganger, er det det samme hvilken som lagres.)

2. String-metode hentFlest henter teksten som ble lagret av et tidligere kall på finnFlest (se forrige punkt).

3. int-metoden hentAntall som henter antallet forekomster funnet i et tidligere kall på finnFlest (se punkt 1).

Skriv også en klasse BrukFrekvens med et testprogram; det skal ha en main-metode som setter det hele i gang. Testprogrammet skal ha én parameter: navnet på en fil med tekster, én tekst per linje. Denne filen vil være alfabetisk sortert. Det skal opprette et Frekvens-objekt med innleste data og etterpå skrive ut hvilken tekst som forekommer flest ganger og hvor mange ganger det er.

2a Fritt valgt datastruktur (vekt 5%)

Implementer klassene Frekvens og BrukFrekvens forklart over. I denne deloppgaven kan du bruke arrayer eller hvilke klasser du vil fra Javas bibliotek til å lagre tekstene. Kjør programmet på angitte testdata noennavn.txt og vis resultatet av kjøringen i et foto eller en skjermdump.

Innleveringen her skal være en ZIP-fil med bildet/skjermdumpen og en Java-fil med all koden.

2b Énveis liste (vekt 10%)

I denne deloppgaven (og alle de etterfølgende deloppgavene) får du ikke lov å benytte ArrayList, HashMap eller andre lagringsstrukturer fra Java-biblioteket.

Du skal nå endre svaret ditt fra forrige deloppgave til å bruke en énveis linket liste til å lagre dataene. Kompiler og kjør programmet med samme testdata som i forrige deloppgave, og ta et foto eller en skjermdump av resultatet.

Innleveringen her skal være en ZIP-fil med bildet/skjermdumpen og en Java-fil med all koden.

(Fortsettes på side 4.)

2c Énveis liste med antall (vekt 25%)

Du skal nå oppdatere koden fra forrige deloppgave til å lagre data i en énveis liste av objekter av en indre klasse. I disse objektene skal du ikke bare lagre String-en og nestepekeren, men også antallet ganger String-en forekommer. Ved initieringen av Frekvens er dette antallet **1** for alle objektene.

Du skal så utvide klassen Frekvens med en metode komprimer som forkorter listen ved å slå sammen etterfølgende like tekster. (Husk at alle tekstene er alfabetisk sortert i datafilen.)

Tegn et eksempel på datastrukturen før og etter en slik komprimering; velg nok objekter til at strukturen kommer klart frem. Bruk et tegneverktøy du liker, eller tegn på et ark og ta bilde av det.

Implementer, kompiler og kjør denne versjonen av Frekvens med samme testdata som i deloppgave 2a, og ta bilde eller en skjermdump av kjøringen.

Innleveringen skal være en ZIP-fil med tre filer: tegningen av datastrukturen, bilde/skjermdump av kjøringen og en fil med Java-kodde.

2d Parallell leting (vekt 20%)

Du skal nå gå tilbake til deloppgave 2a. Ta utgangspunkt i koden du laget til den og la data i Frekvens være lagret i en array. (Du skal altså ikke bruke noen énveis liste.)

For å prøve å forbedre hastigheten skal du bruke tråder til å parallellisere tellingen av tekster som gjøres av finnFlest. Metoden skal opprette én eller flere tråder og la hver tråd lete i sin del av arrayen.

NB! Når du deler opp arrayen, kan du ikke alltid dele den i like store deler. Du må kanskje justere grensene noe slik at sekvenser av samme tekst ikke havner i to forskjellige deler.

Dette programmet skal ha to parametre: antallet tråder og datafilen. Kjør derfor programmet to ganger med disse kommandoene

```
java BrukFrekvens 3 navn.txt
java BrukFrekvens 4 navn.txt
```

Implementer, kompiler og kjør denne koden, og ta et bilde eller en skjermdump av kjøringen. I denne deloppgaven skal du bruke disse testdata: navn.txt.

Innleveringen her skal være en ZIP-fil med bildet/skjermdumpen av kjøringen og en Java-fil med koden.

2e Generelle data (vekt 10%)

Du skal nå ta utgangspunkt i koden din fra del 2c (dvs den med metoden komprimer). Gjør klassen Frekvens *generisk* slik at den kan brukes til å lagre ulike typer data, ikke bare String-er.

(Fortsettes på side 5.)

Opprett også en egen klasse du kan bruke som data. Du skal ikke lese data fra en fil denne gangen, men lage objekter av denne dataklassen og legge disse inn i Frekvens selv. Dette skal gjøres i main-metoden. Lag minst fem dataobjekter.

Implementer, kompiler og kjør denne koden, og ta et bilde eller en skjermdump av kjøringen.

Innleveringen her skal være en ZIP-fil med bildet/skjermdumpen og en Java-fil med koden.

2f GUI-grensesnitt (vekt 5%)

Ta igjen utgangspunkt i koden fra 2c og gi programmet et GUI-grensesnitt. Det kan for eksempel se slik ut:

Når programmet ditt starter, kan det se ut som det venstre bildet, og etter at brukeren har valgt en datafil ved å klikke på den øverste knappen, kommer resultatet midt i vinduet.

Implementer, kompiler og kjør programmet, og ta et bilde eller en skjermdump av kjøringen.

Innleveringen her skal være en ZIP-fil med bildet/skjermdumpen og en Java-fil med koden.

2g Frekvensoversikt (vekt 5%)

Til sist skal du ta utgangspunkt i koden du laget i 2c og utvide den til også å vise en oversikt over hvor mange forekomster det er av unike navn, 2 like navn, 3 like navn osv. Utskriften kan se slik ut med samme testdata som i deloppgave 2a:

```
4 like navn: 1 forekomster
3 like navn: 2 forekomster
1 like navn: 2 forekomster
```

Innleveringen her skal også være en ZIP-fil med en Java-fil og bilde/skjermdump fra kjøringen.