# UNIVERSITETET I OSLO

## Det matematisk-naturvitenskapelige fakultet

**Eksamen i: INF1010 – Objektorientert programmering**
**Eksamensdag: Tirsdag 12. juni 2012**
**Tid for eksamen: 9:00 – 15:00**
**Oppgavesettet er på 5 sider**
**Vedlegg: Ingen**
**Tillatte hjelpemidler: Alle trykte og skrevne**

*Kontroller at oppgavesettet er komplett før du begynner å besvare spørsmålene.*

Vanskelighetsgraden på oppgavene 1, 2 og 3 er ment å være økende. Oppgave 1 er helt uavhengig av de andre oppgavene. Oppgavene 3 og 4 er uavhengig av hverandre, men bygger begge på oppgave 2. Du skal skrive alle programmer i Java.
Gjør dine egne fornuftige antagelser, og skriv disse ned i besvarelsen din, der du synes dette er nødvendig, f.eks. hvis noe er uklart.

## Innledning oppgave 1. Emballasje (Vekt 20%)

Emballasjefabrikken Renpakk skal lage et nytt datasystem 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.

Du får beskjed om å beskrive plastflaske med pant, liten plastflaske med pant og liten nedbrytbar plastflaske med pant. Renpakk har fått et nytt og enestående produkt de er veldig stolt av, nemlig stor nedbrytbar pappflaske med pant, og du skal også ta med en klasse for slike flasker.

**Oppgave 1a**
Tegn klassehierarkiet for de 9 produkttypene som er beskrevet over. Inkluder også eventuelle grensesnitt.
(Selv om et godt klassehierarki kunne ha tatt med flere eller andre klasser skal du bare ta med de som er nevnt over, ingen andre).
**Slutt oppgave 1a**

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 gjenstand 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 gjenstanden er gått i oppløsning.

Av de 9 produkttypene i klassehierarkiet produseres det bare ting av disse 3: liten plastflaske med pant, liten nedbrytbar plastflaske med pant og stor nedbrytbar pappflaske med pant. Alle de andre klassene skal det ikke kunne lages objekter av.

**Oppgave 1b**
Programmer 7 av de 9 klassene i klassehierarkiet og eventuelle grensesnitt. Du skal ikke programmere klasser for Glassemballasje og Metallemballasje. Alle variable i alle klasser skal få verdier i det objektene opprettes. Det er derfor viktig at alle klasser har konstruktører med parametere der disse verdiene kan oppgis, unntatt panten på små flasker som alltid har samme verdi, en konstant med verdi 100 øre.

## Innledning oppgave 2. Om å slå sammen like objekter (Vekt 30%)

I denne oppgaven skal du programmere en klasse med navn Frekvens, som beskriver en beholder. Du skal helst lage klassen generisk (som en parameterisert klasse), men hvis du vil kan du lage den slik at den bare kan ta vare på objekter av klassen String. Hvis du velger dette siste alternativet kan du få maksimalt 25 av 30 (prosent)poeng på denne oppgaven. I oppgave 2 skal du ikke bruke noen klasser fra Java-biblioteket, du skal programmere alt selv.

Formålet med beholderen Frekvens er å slå sammen alle like objekter, samtidig som den holder orden på hvor mange slike like objekter det var opprinnelig. F.eks. hvis beholderen skal ta vare på tekster (String): Gir vi beholderen de syv tekstene Anne, Anne, Anne, Ole, Rita, Rita, Rita, så vil resultatet være (etter at like objekter er slått sammen) Anne 3, Ole 1, Rita 3.

For å finne ut om to objekter er like skal du bruke metoden equals (som er definert i klassen Object), og som returnerer sann (true) ved likhet. (Metoden equals brukes jo bl.a. til å sjekke om to String-objekter er like.) Når vi slår sammen to objekter vil vi egentlig kaste det ene. Hvilket vi kaster og hvilket vi beholder er likegyldig.

Du skal i denne oppgaven legge alle objektene inn i beholderen ved hjelp av konstruktøren. Du skal ikke lage metoder for å legge inn nye objekter. Konstruktøren skal ha som parameter en tabell (array) med pekere til alle objektene som skal legges inn. Vi skal forenkle oppgaven ved å la denne tabellen være ferdig sortert (du skal altså ikke sortere i ditt program).

Konstruktøren skal opprette en enkel liste som inneholder alle objektene i tabellen. (Siden du ikke kan bruke noen verktøy fra Java-biblioteket må du programmere denne listen selv.) Når konstruktøren er ferdig skal listen inneholde like mange objekter som tabellen, den skal fortelle at alle objekter forekommer nøyaktig en gang, og listen vil være sortert (siden tabellen var sortert). NB! Det er viktig at du følger denne beskrivelsen nøye. Du skal ikke gjøre noe av oppgaven til metoden likeSammen (se nedenfor) i konstruktøren (selv om det kanskje kunne vært naturlig i denne forenklede versjonen av klassen Frekvens).

Når konstruktøren er ferdig vil eksemplet over gi en liste som inneholder: Anne 1, Anne 1, Anne 1, Ole 1, Rita 1, Rita 1, Rita 1.

**Oppgave 2a**
Tegn datastrukturen i et objekt av klassen Frekvens, ved først å vise hvordan datastrukturen ser ut rett etter at objektet er opprettet (og konstruktøren er utført), og deretter hvordan datastrukturen ser ut etter at metoden likeSammen er kalt (se nedenfor). Bruk gjerne eksemplet vist over.

**Oppgave 2b**
Programmer deklarasjonene og datastrukturen i klassen Frekvens slik den er beskrevet over, og med alt du trenger for å besvare oppgavene 2c og 2d.

**Oppgave 2c**
Programmer konstruktøren i klassen Frekvens.
**Slutt oppgave 2c**

I oppgave 2 skal beholderen bare inneholde tre offentlige metoder: en metode kalt likeSammen, en kalt hentFlestObjektet og en kalt hentFlestAntallet.

Metoden likeSammen skal ikke ha noen parametere og ikke returnere noe. Metoden skal gå gjennom listen, slå sammen like objekter og holde orden på hvor mange like objekter det var i den opprinnelige listen. Siden listen er ferdig sortert vil like objekter ligge etter hverandre i listen. Resultatet vil være en kortere (sortert) liste der ingen objekter er like. Samtidig som metoden går gjennom listen og slår sammen objekter, skal den også finne ut (til bruk i de to andre metodene) hvilket objekt som hadde flest like forekomster, og hvor mange dette var. Resultatet i vårt eksempel ville vært (som allerede vist over): Anne 3, Ole 1, Rita 3, og det er Anne (eller Rita) som forekommer flest ganger, og dette er 3 ganger.

Metoden hentFlestObjektet skal returnere en peker til det objektet som representerer flest like forekomster, og metoden hentFlestAntallet skal returnere dette største antallet av like forekomster. Hvis det er flere forskjellige objekter som har flest like forekomster spiller det ingen rolle hvilket av disse objektene som velges.

**Oppgave 2d**
Programmer de tre metodene likeSammen, hentFlestObjektet og hentFlestAntallet i klassen Frekvens.

## Innledning oppgave 3. Sammenslåing med tråder (Vekt 35%)

I denne oppgaven skal du bruke tråder for raskere å finne det objektet i en stor tabell som har flest like forekomster. Du skal gjøre dette ved å programmere en trådklasse som bruker klassen Frekvens fra oppgave 2. Du kan opprette objekter og kalle på metoder som er definert i oppgave 2, selv om du ikke har løst oppgave 2. I denne oppgaven spiller det ingen rolle for karakteren om du bruker generiske klasser eller ikke.

Tabellen (arrayen) som skal undersøkes, er sortert (som i oppgave 2), men nå skal du dele tabellen i mange nesten like store deler. Hver del skal gis til en tråd som finner det største antallet like forekomster i sin del av tabellen. Til slutt skal de enkelte trådenes resultater sammenliknes for å finne det aller største antallet.

Du kan velge mellom to måter å løse denne oppgaven på:

A. Omtrent hvor lang hver del av tabellen skal være er forhåndsbestemt av en konstant i programmet. Deretter lager programmet så mange tråder det trenger for at hele tabellen skal blir undersøkt.
B. Det er forhåndsbestemt av en konstant i programmet omtrent hvor mange tråder som skal arbeide. Programmet må dele tabellen opp i omtrent like store deler, slik at det blir en del per tråd.

Skriv tydelig i besvarelsen din om du velger fremgangsmåte A eller B. Valget skal ikke begrunnes og det spiller ingen rolle for karakteren din hvilken fremgangsmåte du velger.

Delene av tabellen behøver ikke være akkurat like lange. Du må være oppmerksom på at tabellen ikke skal deles mellom to like objekter. Om du er i ferd med å gjøre dette kan du f.eks. øke antallet som skal undersøkes av en tråd (helt til alle like objekter blir undersøkt av samme tråd).

For å kunne bruke klassen Frekvens fra oppgave 2 på en enkel måte, skal konstruktøren i trådklassen opprette en egen lokal tabell (array) (av riktig lengde), og kopiere den delen av tabellen som denne tråden har fått i oppgave å undersøke, over i denne lokale tabellen. Når tråden oppretter en beholder (av klassen Frekvens) er det denne tabellen som brukes som parameter til konstruktøren i beholderen.

Når en tråd har funnet det største antallet av like forekomster i sin del av tabellen, skal den legge disse verdiene inn i en monitor som har som oppgave å ta i mot resultatene fra alle trådene, og til slutt sitte igjen med det største resultatet. Når hovedprogrammet har startet alle trådene må det vente på at alle er ferdige, og da først kan det hente ut av monitoren det største resultatet.

**Oppgave 3a**
Programmer trådklassen og monitorklassen.

**Oppgave 3b**
Skriv et fullstendig program med en hovedklasse og en main-metode som finner det ordet som har flest forekomster i en stor tabell (av typen String). Main-metoden skal lese en fil med navn ”mangeordsortert.txt”, der første linje inneholder antallet ord i filen, og resten av filen inneholder dette antallet ord (som du for eksempel kan lese med next-metoden i Scanner-klassen) i ferdig sortert rekkefølge. Disse ordene skal legges inn i String-tabellen. Deretter skal main-metoden opprette monitoren og alle trådene, og til slutt skal main-metoden hente ut resultatet fra monitoren og skrive ut det ordet som har flest forekomster og hvor mange dette er.

## Innledning oppgave 4. Antall like objekter (Vekt 15%)

I denne oppgaven skal du utvide klassen Frekvens fra oppgave 2. Du skal bruke deklarasjonene og datastrukturen i klassen Frekvens slik du programmerte det i oppgave 2b, men du kan besvare oppgave 4 selv om du ikke har besvart oppgavene 2c og 2d. Det spiller ingen rolle for karakteren på denne oppgaven om klassen Frekvens er programmert som en generisk klasse eller ikke.

I oppgave 4 skal du programmere to metoder til i klassen Frekvens: lagAlleAntall og finnAntall. Hvis et program skal kalle finnAntall må det først ha kalt lagAlleAntall, og for å kalle lagAlleAntall må det først ha kalt likeSammen (fra oppgave 2).

Metoden lagAlleAntall har ingen parametere og returnerer ingen verdi. Den skal lage en oversikt over hvor mange (forskjellige) objekter som forekommer bare 1 gang, hvor mange (forskjellige) objekter som forekommer 2 ganger, hvor mange (forskjellige) objekter som forekommer 3 ganger, osv., helt opp til det antallet som angir flest like objekter. F.eks. hvis listen er (som før): Anne 3, Ole 1, Rita 3, så skal oversikten si at 1 objekt forekommer 1 gang, 0 objekter forekommer 2 ganger og 2 objekter forekommer 3 ganger.

Metoden finnAntall skal gjøre oppslag i denne oversikten. Metoden har som parameter et heltall som angir et antall like forekomster, og metoden skal returnere hvor mange ganger beholderen inneholder så mange like objekter. Høyeste parameterverdi vil derfor være det største antallet like forekomster. I vårt eksempel vil finnAntall(1) returnere 1, finnAntall(2) returnere 0 og finnAntall(3) returnere 2.

**Oppgave 4a**
Programmer metodene lagAlleAntall og finnAntall samt de deklarasjoner og datastruktur du trenger til disse metodene i klassen Frekvens.

**Oppgave 4b**
Noen av metodene i klassen Frekvens må kalles i en bestemt rekkefølge. Diskuter meget kort hvordan metodene ideelt sett bør oppføre seg hvis denne rekkefølgen ikke følges. Det er ikke nødvendig å gå tilbake og endre noen av de metodene du har skrevet.

Lykke til og god sommer!

Stein Gjessing