UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF1010 — Objektorientert programmering
Dato:	4. juni 2015
Tid for eksamen:	09.00 – 15.00 (6 timer)
Oppgavesettet er på 6 sider.	
Vedlegg:	4 sider + Kapittel 20 fra BIG JAVA
Tillatte hjelpemidler:	Læreboka (med notater i)

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

Les igjennom hele oppgaveteksten før du begynner å svare. Noter ned uklarheter/spørsmål under gjennomlesingen slik at du har spørmålene klare når faglærer kommer. Husk å skrive i besvarelsen der du gjør egne forutsetninger. Forklar gjerne med ord hva en kodebit det ikke eksplisitt er spurt etter, skal gjøre.

Oppgave 1. Klassehierarki—vekt 10%

Oppgave 1a
Du skal skrive klasse- og interfacedefinisjoner slik at datastrukturen som er tegnet i vedlegg 1 kan opprettes. Du trenger ikke skrive konstruktører og metoder. Heller ikke objektvariable. På tegninga er typen til variablene skrevet over, og navnet skrevet under variabelen (boksen). Du skal tilsammen skrive 7 definisjoner, som svarer til typene QA, QAB, QABC, QAC, G, QK og Q. Med definisjoner menes javakode, men klassene og grensesnittene skal være tomme: { }.
Tegninga er uttømmende, dvs at det ikke er lovlig med andre referanser enn dem som står der. F.eks. kan ikke objekter[1] pekes på av variable av flere typer enn det som tegninga viser, dvs hverken av (pekere av type) QAB, QABC, G eller QK. Og objektet objekter[3] peker på, kan *bare* pekes på av variabler av typene Q, QK og G. Ingen av objektene i tegninga er av samme klasse, og objekter av alle klasser det kan lages objekter av, er med.

Oppgave 1b
Skriv et fullstendig program ved å utvide programkoden fra oppgave 1a med en klasse med en main-metode som oppretter tilstanden som vedlegget viser. For å lette sensuren, opprett først objekter[0], og de variablene som peker på det i den rekkefølge tegninga viser fra venstre mot høyre. Deretter objekter[1], osv.

(Fortsettes på side 2.)

Oppgave 2. Datastruktur—vekt 10%

Når programmet i vedlegg 2 kompileres og kjøres slik:

```
javac Oppgave.java
java Oppgave A B
```

blir det opprettet en datastruktur. Tegn denne datastrukturen når programmet har kommet der det er merket med kommentaren «OPPGAVE». Eksempler på hvordan vi tegner variable, arrayer, metoder, konstruktører og objekter framgår av vedlegg 1 og 3.

Oppgave 3. Ordnet lenkeliste—vekt 25%

Fullfør programmet i vedlegg 4.
Objektene som blir satt inn i beholderen Liste er sammenlignbare med objekter av samme type. I lenkelista i beholderen skal objektene ligge ordnet (sortert) slik at mindre ligger foran større. Det skal ikke gjøres endringer andre steder i prekoden enn det som hører til hver deloppgave. Hvis du foretrekker en top-down utviklingsmåte anbefales det å gjøre deloppgavene i rekkefølgen d, c, b, a. Når testprogrammet i vedlegget kjøres, skal utskriften bli:

```
Setter inn: I dag er det eksamen i INF1010.
 Jeg håper du liker denne oppgaven.
 Lykke til! hilsen oppgaveforfatteren

Alle i lista:
---
I
INF1010.

Jeg
Lykke
dag
denne
det
du
eksamen
er
hilsen
håper
i
liker
oppgaveforfatteren

oppgaven.

til!
 -- SLUTT
```

(Fortsettes på side 3.)

Oppgave 3a
Skriv metoden

```
int sammenlign(Node k) {  }
```

Metoden skal virke som compareTo, dvs returnere et heltall større enn null hvis objektet som k.t peker på er mindre enn det (denne nodens) t peker på.

Oppgave 3b
Skriv den *rekursive* metoden

```
void settInn(Node ny) {  }
```

som setter en ny node inn i lenkelista pekt på av foran i Liste-klassen. Du får ikke full uttelling på denne oppgaven hvis metoden ikke er rekursiv.

Oppgave 3c
Skriv klassen

```
private class ListeEnde extends Node {    }
```

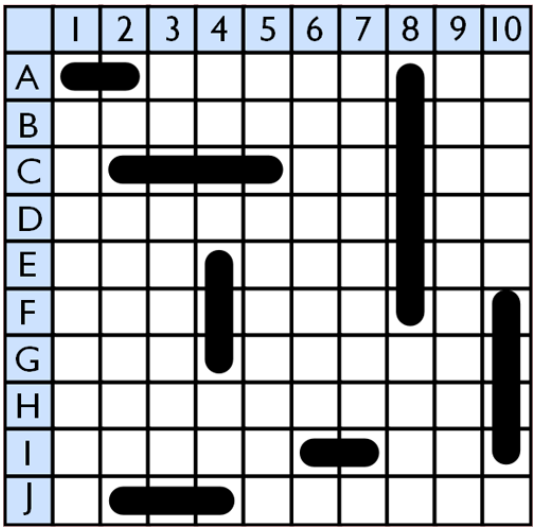
Objekter av denne klassen skal brukes til to spesialnoder, listehode og listehale. Listehodet er noden som ligger først og -halen er noden som alltid ligger sist i lenkelista. Bruk polymorfi slik at listehodenoden blir «mindre enn» alle andre noder, og slik at listehalenoden er «større enn» alle andre noder.

Oppgave 3d
Skriv konstruktøren Liste() {...}
Her initialiseres lista med listehode og -hale.

(Fortsettes på side 4.)

Oppgave 4. Spill; senke skip—20%

Lag en tegning på skjermen med programpakken jawax.swing av et rutenett med svarte og hvite ruter, som på figuren nedenfor. De svarte rutene representerer skip, og de hvite hav. Et skip har to egenskaper som varierer. En *lengde* og en *orientering*. Lengden er antall ruter skipet fyller i rutenenettet, mens orienteringen er enten vertikal (nedover) eller horisontal (bortover). Mao. et skip «vet» hvor i rutenenettet det ligger.



For grafisk å lage de to typer ruter (svart eller hvit) har vi bildene

```
ImageIcon skip = new ImageIcon("skip.png");
ImageIcon hav  = new ImageIcon("hav.png");
```

Oppgave 4a
Skriv klassen Skip som en superklasse til de andre klassene som representerer skip. Konstruktøren skal ha 3 parametre

- en peker til hele brettet, rutenettet
- en peker til første havrute det ligger i
- lengden til skipet

Oppgave 4b
Skriv klassene VSkip og HSkip som subklasser av Skip. HSkip skal være skip som ligger horisontalt i gridet. VSkip vertikalt.

Oppgave 4c
Skriv klassen Hrute (havrute) som en subklasse til JLabel. Ruta skal ha en peker som peker på et skipsobjekt hvis det ligger et skip i ruta. Hvis ikke er denne variabelen null. Eks. Rutene E4, F4 og G4 i figuren peker til samme objekt. I3 peker til null. Senere skal hver rute kunne håndtere museklikk (oppgave d), dette kan du implementere i denne delen hvis du vil.

Oppgave 4d

(Fortsettes på side 5.)

Skriv en klasse som oppretter et JFrame-objekt (direkte eller som subklasse) og som har en datastruktur med en todimensjonal array som peker til 10×10 Hrute-objekter. I klassen skal skip (i denne testfasen av programmeringen) plasseres som på figuren over. Når objektet er ferdig opprettet skal det vises fram på skjermen, som et 10×10 rutenett med blanke ruter (hav). Skipene skal *ikke* vises. Du trenger ikke å lage raden med tall og kolonnen med bokstaver.
Spilleren bruker musa til å klikke på rutene for å prøve å «treffe» skipene som er skjult. Hvis det er et skip i ruta, skifter ruta utseende til fargen for skip/treff med kall på metoden setIcon. Hvert klikk med musa på brettet registereres og målet for spilleren er å treffe alle skipene med færrest klikk. Når alle skipene er «senket» skrives en melding om det på skjermen, sammen med hvor mange «skudd» spilleren brukte.
Et JLabel-objekt lbl skal ha dette utseende når det visualiserer en havrute uten skip:

```
lbl.setIcon(hav);
```

Et JLabel-objekt lbl skifter utseende til treff/skip med metodekallet:

```
lbl.setIcon(skip);
```

Et JFrame-objekt får riktig layout for dette spillet med kallet:

```
setLayout(new GridLayout(10,10));
```

Et objekt blir synlig med kallene (i objektet):

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
```

For å fange opp museklikk, får du bruk for grensesnittet:

```
public interface MouseListener extends EventListener {
        public void  mouseClicked(MouseEvent e);
        public void  mouseEntered(MouseEvent e);
        public void  mouseExited(MouseEvent e);
        public void  mousePressed(MouseEvent e);
        public void  mouseReleased(MouseEvent e);
}
```

I tillegg til klassene og metodene som er nevnt her, trenger du metoder for å legge komponenter inn i bildeflater, og for å legge til en muselytter til en komponent. De du trenger her framgår av denne kodebiten:

```
JPanel jp = new JPanel();
JLabel lbl = new JLabel();
lbl.addMouseListener()
jp.add(lbl)
```

Oppgave 5. Boblesortering med tråder—vekt 35%

Fra no.wikipedia.org: *Boblesortering (Bubble sort) er en svært enkel sorteringsalgoritme som først og fremst brukes i pedagogisk sammenheng. Algoritmen begynner først i datasettet og sammenligner de to første elementene, og dersom det første er større enn det andre, bytter de plass.*

(Fortsettes på side 6.)

Dette gjøres med alle par av etterfølgende elementer gjennom hele datasettet. Deretter begynner algoritmen ved starten igjen, og repeterer inntil ingen elementer må byttes på siste gjennomgang. Boblesortering er svært lite effektivt og tidsbruken øker vanligvis kvadratisk med økende datamengde.

Datasettet her er en stor array med n String-objekter. $n > 64000$. Vi skal, til tross for ineffektiviteten ved boblesortering skrive en metode som sorterer en del av arrayen på denne måten.

Oppgave 5a
Skriv en metode

```
sorterDel( String[] a, int fom, int tom )
```

som ordner arrayelementene fra og med a[fom] til og med a[tom] i stigende orden, a[i] <= a[i+1], med boblesortering.

Oppgave 5b
Del arrayen opp i 32 omtrent like store deler og skriv et program som lar 32 tråder sortere hver sin del. For at ordene skal kunne byttes fra del til del, må det være overlapp på minst en indeks mellom hver tråd. Vi kaller mellomrommet mellom to delarrayer for *skjøten* mellom dem. Det holder at en av de to trådene kan bytte over skjøten.

Oppgave 5c
Skriv en monitor med en synkronisert metode som kalles av trådene når de skal bytte innhold imellom indekser som er delt mellom to tråder. Dette for å unngå at det oppstår feil hvis to tråder driver og bytter om på samme element samtidig.

Oppgave 5d
Utvid monitoren med metoder som synkroniserer termineringen av trådene. Husk en tråd kan tro at den er ferdig, men så dukker et element opp i skjøten fra nabotråden. Det enkleste er nok at når en tråd er ferdig med sin del, legger den seg til å sove, men vekkes opp igjen når det har skjedd et bytte over en skjøt. Etter et bytte over en skjøt vekkes da alle sovende tråder opp. Når alle trådene har lagt seg til å sove er sorteringen ferdig.

En mer effektiv måte er om det lages én monitor for hver skjøt, slik at bare den andre tråden vekkes opp igjen hvis den har lagt seg til å sove.

Oppgavesett slutt.

Lykke til, og god sommer!

Stein Michael Storleer