Obligatorisk oppgave 4 INF1010 2013

Versjon 2.0 — Endelig versjon. *Større endringer i del C og D. Feil i teksten blir rettet. Ingen andre endringer. Uklarheter og spørsmål oppklares i blogginnlegget.*

Nytt i denne obligen er en beholder beskrevet ved et grensesnitt, grensesnitt med generiske parametre, restrikterte generiske parametre, sortert lenkeliste. Grensesnittene Comparable<T> *og* Iterable<T>*.*

Oppgaven faller naturlig i to deler: En del hvor du skal lage et klassehierarki. Av klassene i hierarkiet skal det lages objekter som skal lagres i en beholder. Den andre delen er å lage denne beholderen, implementere beholderen som en lenkeliste. Vi beskriver denne delen først.

Beholderen skal implementere dette grensesnittet som beskriver de metodene beholderen må ha (grensesnittet skal senere utvides med en iterator):

```
public interface INF1010samling<N extends Comparable<N>,V> {


// Objekter i beholderen har en nøkkel og en verdi
// N er typen til nøkkelen til et objekt
// V er typen (verdien) til objektet som kan knyttes til nøkkel


    void leggInn(N n, V v);
    // legger objektet som v peker på og som har nøkkel n inn i beholderen

    int antall();
    // antall objekter i beholderen

    V hent(N n);
    // hvis det finnes et objekt (assosiert) med nøkkel nkl i beholderen
    // slik at nkl.compareTo(n) == 0  skal objektet returneres

    V hent(int nr);
    // returnerer objekt nummer nr fra beholderen
    // returnerer null hvis nr < 1 eller nr > antall()

    V hentMinste();
    // returnerer objektet med den minste nøkkelen

    V hentStorste();
    // returnerer objektet med den største nøkkelen

    boolean inneholder(N n);
    // true hvis det finnes et objekt med nøkkel nkl
    // i beholderen slik at nkl.compareTo(n) == 0

    boolean fjernElement(N n);
    // fjerner knyttet til nøkkelen n fra beholderen. true hvis fjernet

    void fjernAlle();
    // fjerner alle objektene lagret i beholderen
```

```
    V[] tilArray(V[] a);
    // returnerer mottatt array med objektene i sortert orden. a peker
    // på en array av aktuell type. a.length >= antall().
    // Metoden «fyller opp» a med alle objektene i beholderen.
}
```

Klassen (beholderen) som implementerer dette grensesnittet skal ha en enkel lenkeliste som indre datastruktur. Den skal hete SELLbeholder (sortert enkeltLenket liste) eller SSLLcontainer. Ellers kan du lage så mye tilleggstruktur og så mange tilleggsmetoder du ønsker.

Beholderens signatur skal være:

```
class SELLbeholder<N extends Comparable<N> , V>
                    implements INF1010samling<N,V> {
```

På grunn av N extends Comparable<N>, vet vi i beholderen at variable av type N (eksempel på nøkkeltype er String) har en metode compareTo(N n), som gjør at vi kan sammenligne dem. Det er dette vi bruker når vi skal holde lista sortert på grunnlag av nøkkelen. Samme metode bruker vi også for å finne objekter med en bestemt nøkkel. Hvis vi ønsker at nøkkelen skal være av en egendefinert type, må denne implementere (eller være en subklasse av en klasse som implementerer) Comparable.

Invariante tilstandspåstander i listebeholderen: Hvis to objekter, a og b med nøkkelverdier a.n og b.n er i beholderen skal

1. a.n.compareTo(b.n) != 0. To objekter med lik nøkkel kan ikke begge være i beholderen.
2. Hvis a.n.compareTo(b.n) > 0 skal a ligge etter b i lista.
3. antall() er antallet objekter i lista.

Invariant 2 betyr at lista er sortert fra minst til størst, og at første listeelement er det minste.

Oppgave A Skriv og test klassen SELLbeholder.

Her beskrives en anbefalt framgangsmåte, men rekkefølge og mengde testing osv. bestemmer du selvfølgelig selv.

Lag først «skallet» med grensesnittet INF1010samling, beholderen, en testklasse som oppretter en konkret beholder (bruk f.eks. String for begge klasseparametrene eller skriv egne testklasser hvis du ikke har laget ferdig del C først), samt klassen med main-metoden. Ikke lag mer før dette kompilerer.

Deretter implementeres metodene i beholderen én for én. Begynn med de enkleste. Metoden som legger objekter inn i lista er kanskje den vanskeligste. For hver metode, lag flere kall og test. Når beholderen begynner å få innhold, lag utskriftsmetoder i beholderen så du får testet om strukturen i beholderen er slik den skal være. Ikke glem å teste ved å forsøke å bryte invariant 1.

Når beholderen er testet ferdig, kan du gå igang med neste del.

Oppgave B Utvid beholderen slik at den kan itereres over. Hvis vi har en beholder

```
INF1010samling<String ,Person> minBeholder;
    //pekervariabel til en beholder som kan romme
    //personobjekter med en tekst som nøkkel
```

skal vi altså kunne iterere over elementene i minBeholder i en for-each-løkke, slik:

```
for (Person p: minBeholder) {
    // kan behandle en og en p
}
```

Da må INF1010samling implementere Iterable<V>. Iteratoren skal skrives «fra bunnen av», dvs. den godkjennes ikke hvis den bruker en predefinert iterator, som f.eks. array sin. remove() skal fjerne det siste objektet som next() returnerte fra beholderen.

Det er derfor ikke bare lov, men anbefalt at man skriver om signaturen til INF1010samling*. En versjon av grensesnittet med iterator blir lagt ut, men du må gjerne legge det til selv. Men det anbefales ikke å bruke versjon med iterator før man har testet ferdig de andre metodene i beholderen.*

Oppgave C—Et konkret klassehierarki

Denne delen av oppgaven er mer løst spesifisert enn resten. Her er det færre krav, skal og må. Du står derfor langt friere til å gjøre dette på din *måte. Poenget er å lage et fornuftig klassehierarki. Objekter av disse klassene skal brukes til å teste resten av oppgaven. Eneste krav er at hierarkiet har (noen) subklasser. Siden dette kan gjøres på mange måter, er det viktig at du forklarer med kommentarer og egne tilstandspåstander hvordan du løser dette.*

Denne «testmodellen» er laget med testing for øye. For ikke å gjøre dette for omfattende, kan oppgaven virke virkelighetsfjern. Du må gjerne innføre nye begreper for å gjøre «kjøretøysregisteret» mer realistisk. I såfall er det viktig å dokumentere godt og snakke med gruppelæreren din om det.

For å bruke og teste beholderen skal du lage et klassehierarki (klasser og grensesnitt) for personer og et for kjøretøy.

Personer eier og reparerer kjøretøy. Noen personer eier kjøretøy, noen reparer (mekanikere) men eier ikke, noen gjør begge deler og noen ingen av delene. Alle eiere skal ha en liste (bruk beholderen fra oppgave A) over kjøretøy de eier. Alle som har en slik beholder er eiere. Alle personer identifiseres ved navn (String).

For kjøretøy skiller vi mellom personbiler, lastebiler og busser. Alle kjøretøy identifiseres ved registreringnummer (String). I tillegg har alle kjøretøy en takst (verdi) som danner grunnlag for avgift og skatt. Det skal ikke være mulig å lage objekter som «bare er» kjøretøy. Kjøretøy skal ha en beholder (fra oppgave A) over personer som har reparert kjøretøyet.

Lag et program (metode) som beregner avgift for alle kjøretøy slik:

For biler: 5% av taksten hvis bilen ikke har vært reparert eller bare er reparert av mekanikere. 7,5% av taksten hvis bilen er reparert av mekaniker mer enn halvparten av gangene bilen har vært reparert. 10% ellers.

For lastebiler og busser: 3,4% av taksten hvis kjøretøyet ikke har vært reparert eller bare er reparert av mekanikere. Hvis det er reparert av en ikke-mekaniker, blir avgiften 12%.

Programmet skal skrive ut en melding dersom den oppdager at et kjøretøy som er reparert av sin eier. Hvis eieren ikke er mekaniker skal kjøretøyet avskiltes og fjernes fra datastrukturen.

Du står fritt til å lage klassehierarkiet slik at krav til modellen og avgiftsberegningen blir oppfylt. Dette gjelder også all ekstrastruktur.

Oppgave D—Testprogram
Skriv et (gjerne flere) program som tester beholderen (alle metodene) på klassehierarkiet du laget i oppgaven ovenfor. Tilslutt skal du teste programmet med dataene som er laget nedenfor.

En testdatafil (personer og kjøretøy) som blir lagt ut i blogginnlegget, har følgende format:

```
PERSONER
<antall personer, et heltall k>
< k navn>
MEKANIKERE
<antall mekanikere, et heltall k>
< k navn>
PERSONBILER
<antall personbiler, et heltall k>
<k registreringsnr og heltall for takst>
LASTEBILER
<antall lastebiler, et heltall k>
<k registreringsnr og heltall for takst>
BUSSER
<antall busser, et heltall k>
<k registreringsnr og heltall for takst>
```

NB! Dataene, men ikke formatet, kan bli endret. Testdata for eierskap og reparasjoner kommer senest 7 dager før fristen.

Testdata og -program blir lagt ut i blogginnlegget.

Obligatorisk oppgave 4 slutt.