IN1010 V22, Obligatorisk oppgave 3

Innleveringsfrist: *Mandag 07.03 kl 23.59*

Innledning

Denne obligatoriske oppgaven skal gjøres som en individuelle oppgave og skal løses på egen hånd, dvs at du skal programmere alle deler av løsningen selv. Dette betyr ikke at vi fraråder samarbeid – tvert imot. Vi oppfordrer deg til å utveksle faglige erfaringer med andre. Følgende er imidlertid viktig:

> Du kan diskutere en løsning med andre, men dere skal ikke dele noen deler av koden (f.eks. ved å levere inn lik kode hvor kun variabelnavn er byttet ut).
>
> Dersom du tar med tekst, programkode, illustrasjoner, og annet som andre har laget, må du tydelig merke det og angi hvor det kommer fra – i en selvstendig oppgave som denne er dette noe som sjelden forekommer. Hvis du er i tvil om hva som er lovlig samarbeid, må du kontakte gruppelærer eller faglærer.

Datastrukturer for leger og resepter

I denne oppgaven skal du lage en rekke beholdere som du vil få bruk for i neste obligatoriske oppgave. Det er derfor viktig å skrive god kode som er lett å lese, for du må komme tilbake til og gjenbruke koden du skriver underveis. Vi trenger noen forskjellige varianter av lenkelister, så vi skal lage flere klasser som arver fra hverandre og implementerer forskjellige grensesnitt.

Begrepene *liste*, *lenkeliste*, *indeksert liste*, *stabel*, *kø* og *prioritetskø* kan være litt ulikt definert i diverse bøker og nettsider; spesielt gjelder det hvilke operasjoner de tilbyr. Les derfor oppgaveteksten nøye og programmer begrepene og operasjonene du finner der.

Når du løser oppgaven, er det spesielt viktig at du tenker på de forskjellige tilstandene en beholder kan ha. Hvordan skal tilstanden være for en tom lenkeliste, og hvordan endrer dette seg når du setter inn et element? Blir tilstanden som før når du tar ut et elementet i listen? Slike spørsmål bør du stille deg underveis. Beskriv tilstanden i et beholder-objekt ved hjelp av en invariant i beholder-klassen.

Hvis du ønsker mer fleksible klasser, kan de selvfølgelig ha flere metoder i tillegg til de som kreves av oppgaveteksten. Merk at det også er opp til deg hvordan du velger å strukturere referansene mellom nodene dine – du velger selv om listen din skal være enkeltlenket (alle noder har én neste-peker) eller dobbeltlenket (nodene har også en peker til forrige node i listen). Om du velger en dobbeltlenket liste, kan Trix-oppgave 7.3 være nyttig.

Hint. Husk å legge inn **`@Override`** hver gang du redefinerer en metode i en subklasse.

Hint. Når man kopierer tekst fra en PDF-fil, kan det komme med mange tegn Java ikke vil kompilere. Vi anbefaler derfor at du laster ned kode du trenger fra emnesiden (sammen med denne oppgaveteksten).

Om enhetstesting av komponenter

For å teste klassene dine skal du laste ned og kjøre en rekke gitte testprogrammer som du finner ved å følge denne linken.

Du skal teste klassene du lager underveis både mot disse testprogrammene og mot tester du selv skriver, f eks ved hjelp av klassene du skrev i forrige obligatoriske oppgave. Pass også på at disse filene ligger i *samme mappe* som klassene dine. **For å få obligen godkjent er det et minstekrav at testene passeres.**

Når vi arbeider med indekser, er det en feil om vi forsøker å nå en indeks som ikke eksisterer. Gyldige indekser i listen vil være som vi er vant til i en array eller en ArrayList, altså fra og med 0 og til, men ikke med, listens størrelse. Unntaket er metoden settInn(int pos,T x) der det er lov å sette inn et element rett etter det siste. For å ta høyde for eventuelle feil skal vi bruke denne egendefinerte unntaksklassen:

```
class UgyldigListeindeks extends RuntimeException {
    UgyldigListeindeks (int indeks) {
        super("Ugyldig indeks: "+indeks);
    }
}
```

Last ned og lagre denne unntaksklassen sammen med Java-filene dine, og sørg for at det kastes et unntak dersom vi forsøker å nå en ugyldig indeks (eller hvis vi forsøker å fjerne noe fra en tom liste – i så fall skal vi kaste unntaket med indeks -1).

Del A: Klassehierarki

I denne obligatoriske oppgaven skal vi basere oss på et interface Liste<T> som ser slik ut:

```
interface Liste <T> {
    int stoerrelse ();
    void leggTil (T x);
    T hent ();
    T fjern ();
}
```

(I forelesningene har dere sett hvordan vi kan lage en klasse ArrayListe som implementerer et grensesnitt som også ble kalt Liste<T>, men i denne oppgaven skal vi implementere det interface-et Liste<T> du ser her.)

Tegn klassehierarkiet til de forskjellige interface-ene og klassene som skal skrives. Det er viktig at du **leser hele oppgaveteksten** før du løser denne oppgaven!

Relevant Trix-oppgave: 5.04 (spesielt deloppgave 7).

Del B: Lenkeliste

Skriv den abstrakte klassen Lenkeliste<T>:

```
abstract class Lenkeliste <T> implements Liste<T> { ... }
```

B1: Skriv hele klassen Lenkeliste<T> som implementerer Liste<T> med en lenket liste. Nye elementer settes inn på slutten av listen og tas ut fra starten slik at det elementet som ble satt inn først, er det første som blir tatt ut. Implementer også alle metodene i Liste<T>:

- Metoden *stoerrelse()* skal returnere hvor mange elementer det er i listen.

- Metoden *leggTil(T x)* skal legge inn et nytt element; det skal legges sist i listen.
- Metoden *hent()* skal returnere det første elementet i listen, men det skal ikke fjernes fra listen.
- Metoden *fjern()* skal fjerne det første elementet i listen og returnere det.

Merk: Du skal **ikke** bruke Javas egne lister (som ArrayList, LinkedList eller andre) i denne obligen.

Hint: Datastrukturen i Lenkeliste bør *ikke* være privat, for du trenger å modifisere den i noen av subklassene du skal lage senere.

B2: Det er svært sannsynlig at listen din ikke vil fungere 100 % korrekt ved første forsøk. For å finne logiske brister i koden kan det være lurt å skrive ut listen for hvert element vi setter inn eller tar ut. Da trenger vi en metode som kan gi oss en String med innholdet av listen.

Skriv en *toString()*-metode i Lenkeliste<T>.

Denne metoden skal bygge opp en svarstreng av elementene i listen. Til å begynne med er svarstrengen tom. Deretter skal du gå gjennom listen og legge til innholdet fra hvert element til strengen. Metoden skal returnere svarstrengen når den er ferdig bygget opp.

Relevante Trix-oppgaver: 5.01, 5.02, 6.01, 6.02

Del C: Stabel

En stabel er en liste som fungerer litt spesielt: det siste elementet som er lagt inn, er alltid det som hentes ut først.

C1: Skriv klassen Stabel<T>. Klassen skal arve fra Lenkeliste<T>, men den skal redefinere metoden leggTil(T x) slik at nye elementer legges *først* i listen.

C2: Sørg for at listen din kommer gjennom testene i *TestStabel.java* før du går videre til Del D.

Relevante Trix-oppgaver: 6.05 & 6.06.

Del D: Koe

D1: Skriv klassen Koe<T> som skal være en subklasse av Lenkeliste<T> og som skal fungere som en kø. Finn selv ut hvilke metoder som må redefineres; skriv disse.

D2: Sørg for at listen din kommer gjennom testene i *TestKoe.java*.

Del E: IndeksertListe

Vi skal nå utvide listen i deloppgave B med muligheten for å indeksere listen, dvs kunne finne elementer i vilkårlige posisjoner, ikke bare først. Vi definerer derfor en subklasse med noen flere operasjoner:

```
class IndeksertListe <T> extends Lenkeliste<T> {
    public void leggTil (int pos, T x) { … }
    public void sett (int pos, T x) { … }
    public T hent (int pos) { … }
    public T fjern (int pos) { … }
}
```

E1: Fyll inn det som mangler i IndeksertListe<T>. Metodene skal gjøre dette:

- Metoden leggTil(int pos,T x) skal sette inn x i listen i posisjon pos der 0<=pos<=stoerrelse(). Dette betyr at alle elementene lenger ut i listen forskyves og heretter får en høyere indeks.
- Metoden sett(int pos,T x) skal erstatte elementet i posisjon pos med x. Lovlig pos er 0<=pos<stoerrelse().
- Metoden hent(int pos) skal hente elementet i gitt posisjon der 0<=pos<stoerrelse(). Elementet skal bli stående i listen.
- Metoden fjern(int pos) skal fjerne elementet i posisjon pos (der 0<=pos<stoerrelse()) og returnere det.

E2: Sjekk klassen med testene i TestIndeksertListe.java.

Del F: Prioritetskoe

Relevant pensum for denne delen foreleses tirsdag 1. mars..

F1: Skriv klassen Prioritetskoe<T extends Comparable<T>>. Denne listen arver også fra Lenkeliste<T>, men vi ønsker at listen skal være sortert og krever derfor at elementer som settes inn, skal være sammenlignbare. Kall på *leggTil(T x)* skal altså sette inn elementer i sortert rekkefølge (fra minst til størst, der minst ligger først i listen). Like elementer (dvs elementer der compareTo(...) returnerer 0) kan ligge i vilkårlig rekkefølge. Når vi bruker *hent*()- og *fjern*()-metodene, skal det minste elementet hentes frem og eventuelt fjernes.

F2: Sørg for at listen din kommer gjennom testene i *TestPrioritetskoe.java*.

Relevante Trix-oppgaver: 7.01 & 7.02.

Del G: Datastrukturtegning

Lag en datastrukturtegning som illustrerer situasjonen i programmet ditt etter at disse operasjonene er utført::

```
IndeksertListe<String> liste = new IndeksertListe<>();

liste.leggTil("A");
liste.leggTil("B");
liste.leggTil("C");
liste.fjern();
liste.sett(1, "D");
```

Du trenger ikke tegne noen metoder, kun objekter og deres instansvariabler.

Levering

Du skal levere følgende:

- Tegning av klassehierarkiet (som bildefil eller .pdf)
- Datastrukturtegning (som bildefil eller .pdf)

- Klassene/interfacene Liste.java, Lenkeliste.java, IndeksertListe.java, Koe.java, Prioritetskoe.java og Stabel.java
- Oppgitt unntak (UgyldigListeindeks.java)

Skriv om du vil ha detaljert tilbakemelding uavhengig om du får godkjent eller ikke.

Alle delene av programmet må kunne kompileres og kjøres på Ifis Linux-maskiner for å kunne få oppgaven godkjent.

Ikke lever zip-filer! Det går an å laste opp flere filer samtidig i Devilry.