Innleveringsoppgave 3

Datastrukturer for leger og resepter

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

Merk at 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 element i listen? Slike spørsmål bør du stille deg underveis.

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 07.07 være nyttig.

Noen hint:

- Husk å legge inn `@Override` hver gang du redefinerer en metode i en subklasse.
- 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 Innleveringsmappen (der du fant denne oppgaveteksten).
- I innleveringsmappen finnes også skjelett av de klassene du skal programmere. Vi anbefaler at du laster ned disse Java-filene og tar dem som utgangspunkt for programmeringen.

Om enhetstesting av komponenter

For å teste klassene dine skal du laste ned og kjøre en rekke gitte testprogrammer som du finner i Innlevering-mappen.

Du skal teste klassene du lager underveis både mot disse testprogrammene og mot tester du selv skriver, for eksempel ved hjelp av klassene du skrev i oppgaven før. Pass på at disse filene ligger i *samme mappe* som klassene dine.

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 inkludert, listens størrelse. Unntaket er metoden `settInn(int pos, E 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 listeindeks: " + indeks);
    }
}
```

Denne klassen finnes blant skjelettfilene. Last den ned og lagre den sammen med Java-filene dine, og sørg for at det kastes et unntak dersom vi forsøker å bruke en ugyldig indeks (eller hvis vi forsøker å fjerne noe fra en tom liste – i så fall skal vi kaste unntaket med indeks `-1`).

Oppgave 1 – Klassehierarki

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

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

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

Oppgave 2 – Lenkeliste

Skriv den abstrakte klassen `Lenkeliste<E>`:

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

(a) Skriv hele klassen `Lenkeliste<E>` som implementerer `Liste<E>` 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<E>`:

- Metoden `størrelse()` skal returnere hvor mange elementer det er i listen.

- Metoden `leggTil(E 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.

Du skal *ikke* bruke Java sine egne lister (som `ArrayList`, `LinkedList` eller andre) i denne oppgaven.

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

(b) Det er svært usannsynlig at listen din 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<E>`.

Denne metoden skal bygge opp en streng 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
07.01 & *07.02*

Oppgave 3 – Stabel

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

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

(b) Sørg for at listen din kommer gjennom testene i `TestStabel.java` før du går videre til neste oppgave.

Relevante Trix-oppgaver
06.02 & *07.04*

Oppgave 4 – Kø

(a) Skriv klassen `Kø<E>` som skal være en subklasse av `Lenkeliste<E>` og som skal fungere som en kø. Finn selv ut hvilke metoder som må redefineres og skriv disse.

(b) Sørg for at listen din kommer gjennom testene i `TestKø.java`.

Oppgave 5 – IndeksertListe

Vi skal nå utvide listen i Oppgave 2 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<E> extends Lenkeliste<E> {
    public void leggTil (int pos, E x) { ... }
    public void sett (int pos, E x) { ... }
    public E hent (int pos) { ... }
    public E fjern (int pos) { ... }
}
```

(a) Fyll inn det som mangler i `IndeksertListe<E>`. Metodene skal gjøre dette:

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

(b) Sjekk klassen med testene i `TestIndeksertListe.java`.

Oppgave 6 – Prioritetskø

(a) Skriv klassen `Prioritetskø<E extends Comparable<E>>`. Denne listen arver også fra `Lenkeliste<E>`, men vi ønsker at listen skal være sortert og krever derfor at elementer som settes inn, skal være sammenlignbare.

Kall på `leggTil(E x)` skal altså sette inn elementer i sortert rekkefølge (fra minst til størst, der minst ligger først i listen). Like elementer (det vil si 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.

(b) Sørg for at listen din kommer gjennom testene i `TestPrioritetskø.java`.

Relevante Trix-oppgaver

07.05 & *07.06*

Oppgave 7 – 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/grenesesnittene `Liste.java`, `Lenkeliste.java`, `IndeksertListe.java`, `Kø.java`, `Prioritetskø.java` og `Stabel.java`
- Oppgitt unntak `UgyldigListeindeks.java`