

IN1010 V21, Obligatorisk oppgave 3

Innleveringsfrist: Mandag 01.03 kl 23.59

Versjon 1.1

Introduksjon

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.

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? Bli tilstanden som før når du tar ut et elementet 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 en neste-peker) eller dobbeltlenket (nodene har også en peker til forrige node i listen). Du kan ha et listehode (en node uten data), eller la være. Om du velger en dobbeltlenket liste kan Trix-oppgave [7.3](#) være nyttig.

Når man kopierer tekst fra en pdf kan det komme med mange tegn java ikke vil compilere. Vi anbefaler derfor at du laster ned kode du trenger fra emnesiden (sammen med denne oppgaveteksten) og ikke kopierer herfra.

Om enhetstesting av komponenter

For å teste klassene dine skal du laste ned og kjøre en rekke gitte testklasser som du finner ved å [følge denne linken](#).

Du skal teste klassene du lager underveis både mot disse enhetstestene og mot tester du selv skriver, f.eks ved hjelp av klassene du skrev i forrige obligatoriske oppgave. De oppgitte enhetstestene (ikke de du har skrevet selv) skal legges ved i innleveringen din, og det er derfor *viktig* at navnene på de påkrevde metodene følger oppgaveteksten. *Pass også på* at de nedlastede filene ligger i *samme mappe* som klassene dine. **For å få obligen godkjent er det et minstekrav at testene passerer.**

Del A: Klassehierarki

Tegn klassehierarkiet (inkludert grensesnitt) til de forskjellige 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

I denne oppgaven skal vi basere oss på grensesnittet `Liste<T>`. Grensesnittet ser slik ut:

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

I forelesning har dere sett hvordan vi kan lage en klasse `ArrayList` som implementerer et grensesnitt som også ble kalt `Liste<T>`, men i denne oppgaven skal vi lage lenkeliste-klasser som implementerer `Liste`- grensesnitt slik det er deklart over (og litt til).

B1: Skriv klassen `Lenkeliste<T>` som implementerer `Liste<T>`. Vi skal enkelt kunne sette inn elementer på slutten av listen og ta ut fra starten slik at det første elementet som ble satt inn også er det første som blir tatt ut. På denne måten kan listen benyttes som en kø (*First in, First out*). Metoden `leggTil(T x)` skal altså sette inn et element på slutten av listen, mens `fjern()` skal fjerne og returnere elementet på starten av listen.

I tillegg skal det finnes metoder for å sette, legge til og fjerne på gitte plasser (NB: Husk at posisjon indekseres fra 0, dvs første element er i posisjon 0):

- Metoden `sett(int pos, T x)` skal sette inn elementet på angitt posisjon og overskrive det som var der fra før av.
- Metoden `leggTil(int pos, T x)` skal legge inn et nytt element i listen. De resterende elementene kommer da ett hakk lenger bakover i listen (får én høyere posisjon).
- Metoden `fjern(int pos)` skal fjerne elementet på gitt indeks i listen. De etterfølgende elementene kommer da ett hakk forover i listen (de får én lavere posisjon).

Til sist skal du også implementere metodene `stoerrelse()` og `hent(int pos)`, der sistnevnte henter ut et element (uten å fjerne det fra lista) på oppgitt indeks.

Merk: Du skal **ikke** bruke javas egne lister (bla. `ArrayList`, `LinkedList`) i denne obligen.

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. Aha, tenker du - en utskriftsmetode! Nettopp sier vi: `toString()`.

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

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

B3: Når vi arbeider med indekser kan vi møte på feil dersom 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. 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 dersom vi forsøker å nå en ugyldig indeks (eller hvis vi forsøker å fjerne noe fra en tom liste - isåfall skal vi kaste unntaket med indeks -1).

B4: Sørg for at listen din kommer gjennom testene i [TestLenkeliste.java](#) før du går videre til Del C.

Relevante Trix-oppgaver: [5.01](#), [5.02](#), [6.01](#), [6.02](#)

Del C: Stabel

En stabel er en liste som fungerer litt annerledes enn en vanlig lenkeliste: Det siste elementet som er lagt inn er alltid det som hentes ut.

C1: Skriv klassen Stabel<T>. Klassen skal arve fra Lenkeliste<T>, men skal i tillegg ha metodene *leggPaa* (*T x*) og *taAv*(). Disse metodene skal henholdsvis legge til og fjerne elementer *fra slutten av listen*, slik at det siste elementet som legges inn er det første som tas ut (Last in, First out). Merk: Det forventes her at du tar i bruk metodene som er arvet fra Lenkeliste<T>.

Klassen Stabel vil følgelig fungere både som en vanlig lenkeliste (med alle tilhørende operasjoner) - og som en stabel hvis programmet bare kaller de to nye metodene *leggPaa*() og *taAv*().

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: Sortert Lenkeliste

Relevant pensum for denne delen foreleses i uke 7.

D1: Skriv klassen SortertLenkeliste<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), og når vi bruker *fjern*()-metoden (uten parametere) skal det største elementet tas ut.

D2: Du skal nå begrense mulighetene for å sette inn elementer på en vilkårlig posisjon, slik at listen forblir sortert. Dette kan gjøres ved å la `SortertLenkeliste` overskrive metodene `sett(int pos, T x)` og `leggTil(int pos, T x)`. De nye implementasjonene skal ikke sette inn elementet, men i stedet kun kaste et unntak som finnes i Java fra før av:

UnsupportedOperationException (dette trenger ikke å importeres).

D3: Sørg for at listen din kommer gjennom testene i [TestSortertLenkeliste.java](#).

Relevante Trix-oppgaver: [7.01](#) & [7.02](#).

Del E: Datastrukturtegning

Lag en datastrukturtegning som illustrerer denne situasjonen:

```
Lenkeliste<String> lenkeliste = new Lenkeliste<String>();
lenkeliste.leggTil("A");
lenkeliste.leggTil("B");
lenkeliste.leggTil("C");
lenkeliste.fjern(1);
lenkeliste.sett(1, "D");
```

Levering

Du skal levere følgende:

- Tegning av klassehierarkiet (som bildefil eller .pdf)
- Datastrukturtegning (som bildefil eller .pdf)
- Klassene `Lenkeliste.java`, `Stabel.java` og `SortertLenkeliste.java`
- Oppgitte grensesnitt og unntak (`Liste<T>` og `UgyldigListeIndeks`)
- Oppgitte testprogrammer til de forskjellige delene av oppgaven.

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

Alle delene av programmet må kompilere og kjøre på lfi-maskiner for å kunne få oppgaven godkjent. *Ikke levér zip-filer!* Det går an å laste opp flere filer samtidig i Devilry.