DAT102 – Obligatorisk nr 2

Består av både Øving 4 og Øving 5 (denne)

Innleveringsfrist: 25. februar 2022

Vi henstiller at dere er 3-4 på gruppen, men aldri flere. Hvis dere lager grupper på 4, kan 2 og 2 jobbe tett sammen i smågrupper med å programmere samtidig og diskutere underveis. Pass på å veksle mellom hvem som skriver. De to smågruppene blir enige om en felles innlevering slik at dere leverer inn som en gruppe på 4.

Dere eksporterer javaprosjektene, zipper og leverer inn på Canvas. Prosjektene skal kunne kjøres. Teorioppgaver leverer dere inn på canvas som pdf-fil.

OBS! Husk å ta med navn på alle gruppemedlemmene. Canvas tillater ikke å gjør det i ettertid.

Oppgave 1

a) Se Klassen `Person` i prosjektet ListeOrdnetU. Ferdigstill metoden `compareTo`. Metoden skal sammenligne to Person-objekter med hensyn på fødselsår. Vi ønsker at den eldste kommer først (`compareTo` returnerer et tall < 0). Ved sammenligning av like fødselsår skal du sammenligne etternavn og deretter eventuelt fornavn (ønsker ordnet i alfabetisk rekkefølge). Tips: Se gjerne `compareTo` i klassen `Lag`, i prosjektet `ListeOrdnetU`.

b) Ferdigstill klassene `KjedetOrdnetListe` og `TabellOrdnetListe`. Ferdigstill også testmetodene. Kjør testmetoden og se at du får grønt.

c) Lag et enkelt `main()`-program der du oppretter en ordnet liste av `Person`-objekt, leser inn 4-6 objekter fra tastaturet og legger hvert objekt etter tur inn i listen. Til slutt tar du ut ett og ett objekt fra listen og skriver det ut på skjermen i stigende alder inntil listen er tom. Pass på at du har flere personer med samme fødselsår. Bruk både `TabellOrdnetListe`-klassen og `KjedetOrdnetListe`-klassen.

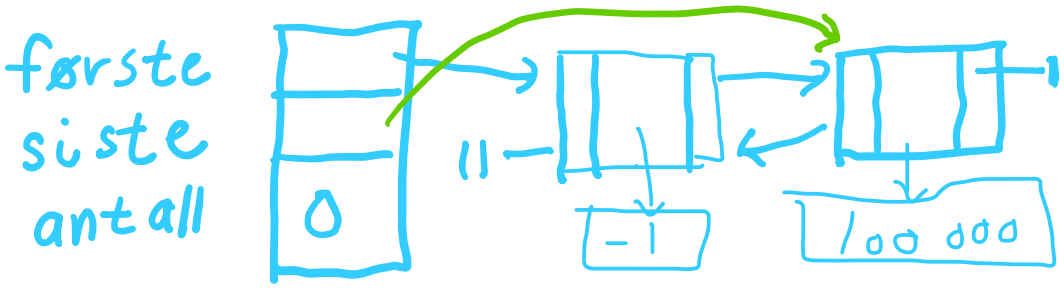
Oppgave 2

Se prosjektet DobbelKjedetListe på github som du kan bygge videre på.

Vi skal se på en variant av dobbeltkjedet ordnet liste der vi bruker kunstige noder først og sist i listen for å gjøre innsetting og sletting enklere. Den første noden inneholder en dataverdi som er mindre enn alle lovlige dataverdier. Den siste noden inneholder en dataverdi som er større enn alle lovlige dataverdier. Dobbelkjedet liste er beskrevet i Forelesning F10b.

Strukturen som vi skal bruke er en **ordnet** liste implementert med dobbeltkjedet struktur (hver node har en peker til forrige node og en peker til neste node). Listen har en peker til første og siste node. I tillegg har vi en objektvariabel antall som angir antall elementer i listen.

Nedenfor ser vi en **tom** dobbeltkjedet liste som kan inneholde heltall i området 0…99999.



a) Lag konstruktøren i `DobbelKjedetListe`.

b) Lag metoden ***fins*** som returnerer true dersom elementet finnes og false ellers.

c) Lag metoden ***visListe*** som viser elementene i listen. NB! De kunstige verdiene skal ikke vises.

d) Lag et lite klientprogram som bruker metodene

Oppgave 3

Vi skal se på "parsing" som blir utført av kompilatorer, nærmere bestemt på parentessetting i et Java-program.
Hvis vi ser bort fra at det inne i kommentarer og strenger også kan være parenteser, *skal* parenteser alltid komme i par. Vi kan dele symbolene {, [, (, },] og) i *åpne*-symboler (venstreparentesene) og *lukke*-symboler (høyreparentesene). Når vi går gjennom et Java-program og finner et lukke-symbol, skal det passe med sist sette åpne-symbol. Det vil si at

[…(…)…] er lovlig

[…(…]…) er ulovlig

For å sjekke om parentesene er rett satt, kan vi bruke en stabel. Vi stabler på når vi finner et åpne-symbol og stabler av og sjekker at vi får et par når vi treffer på et lukke-symbol. I denne oppgaven kan du enten bruke klassen **`KjedetStabel`** eller **`TabellStabel`**. Tre feilsituasjoner kan oppstå:

1. Symbolene danner ikke par.
2. Vi treffer et lukkesymbol og stabelen er tom.
3. Det er slutt på teksten og stabelen det er fremdeles minst ett symbol på stabelen.

a) Implementer grensesnittet, `Parentessjekker`, nedenfor og lag et lite main-program der du sjekker noen strenger der det er minst en streng med hvert av feilene ovenfor og noen strenger som er korrekte.

```java
public interface Parentessjekker {

	/**
	 * Metoden sjekker om et tegn er '(', '[' eller '{'.
	 *
	 * @param p tegn som skal sjekkes
	 * @return true dersom tegnet er en venstrepareentes, false elles
	 */
	boolean erVenstreparentes(char p);

	/**
	 * Metoden sjekker om et tegn er ')', ']' eller '}'.
	 *
	 * @param p tegn som skal sjekkes
	 * @return true dersom tegnet er en hogreparentes, false elles
	 */
	boolean erHogreparentes(char p);

	/**
	 * Metoden sjekkker om et tegn er en parentes.
	 *
	 * @param p tegn som skal sjekkes
	 * @return true dersom tegnet er  (', '[', '{', ')', ']' eller '}', false
ellers.
	 */

	boolean erParentes(char p);

	/**
	 * Metoden sjekker om to tegn er et parentespar.
	 *
	 * @param venstre er første tegn i potensielt par
	 * @param hogre er andre tegn i potensielt par
	 *
	 * @return true dersom de matcher, dvs. venstre er en venstreparentes og
hogre er tilhørende
	 * høgreparentes
	 */

	boolean erPar(char venstre, char hogre);

	/**
	 * Metoden sjekker om en streng som inneholder parenteser er balansert. Den
ser bort
	 * fra tegn som ikke er parenteser.
	 *
	 * @param s streng som skal sjekkes
	 * @return true dersom parentesene i strengen er balansert, false ellers.
	 */

	boolean erBalansert(String s);
}
```

b) (Frivillig, skal ikke leveres inn)

Modifiser og utvid programmet fra a) slik at det leser et Java-program fra fil og at det prøver å gi fornuftige feilmeldinger. For å kunne gi fornuftige feilmeldinger, trenger du linjenummer og posisjon på linjen for åpnesymbolene på stabelen. Dette gjøres ved å definere en klasse som inneholder nødvendige opplysninger. Så stabler du på objekter av denne klassen. Eksempel på feilmeldinger:

1) Symbolene danner ikke et par. Eksempel: Vi treffer på en "]" og øverst på stabelen er det en "{" eller en "(". Da må du skrive ut en passende feilmelding, men du skal likevel behandle resten av teksten også. (Dette kan da i noen tilfeller resultere i en rekke nye feilmeldinger selv om det bare var *en* feil i uttrykket.)

   Eks.: Lukkesymbol] på linje nr x, tegn nr y har feil åpnesymbol

2) Vi treffer på et lukkesymbol og stabelen er tom. Da vil det være rimelig å gi melding om at det er truffet på et lukke-symbol uten tilsvarende åpne-symbol.

   Lukkesymbol] på linje x, tegn nr y mangler tilsvarende åpnesymbol

3) Stabelen er ikke tom når det er slutt på teksten. Her kan du gi melding om at det mangler (et eller flere) lukke-symbol.

   Åpnesymbol (på linje x, tegn nr y har ikke tilsvarende lukkesymbol

   Åpnesymbol [på linje z, tegn nr w har ikke tilsvarende lukkesymbol

Oppgave 4

a) Summen av de n første naturlige tall er gitt ved: $S_n = 1+2+3+\ldots+n$. En formel for å finne S_n er gitt ved:

$S_n = S_{n-1} + n$, $S_1 = 1$

Lag en rekursiv Java-metode som beregner S_n, og skriv et enkelt hovedprogram som bruker denne metoden for å finne S_{100}.

b) Gitt tallfølgen $\{a_n\}$ der de enkelte ledd kan finnes med formelen:
$a_n = 5a_{n-1} - 6a_{n-2} + 2$ for $n > 1$ og startkrav $a_0 = 2$, $a_1 = 5$

Lag en rekursiv Java-metode som beregner a_n, og skriv et enkelt hovedprogram som bruker denne metoden til å vise de 10 første leddene i tallfølgen.

c) Fibonaccitallene dukker opp både i naturen og innenfor økonomi. De kan defineres på ulike måter. En rekursiv definisjon er:
$f_n = f_{n-1} + f_{n-2,}$ og $f_0 = 0, f_1 = 1$

Lag en rekursiv Java-metode som beregner f_n. Lag et hovedprogram der du prøver litt ulike verdier av n. Etter hvert som n øker, vil du oppdage at det tar lang tid å utføre metoden.

d) Det er lett å lage en ikkerekursiv metode for å beregne Fibonaccitallene. Lag en slik metode og observer at den vil være rask å utføre for verdier av n der den rekursive metoden bruker lang tid. Tips: Beregn $f_2, f_3, \ldots, f_n$.