LØSNINGSFORSLAG - Seminaroppgaver, uke 8 - Repetisjon før midtveis

1.)

```
public class Lenkeliste<E> {
   protected class Node {
       E data;
       Node neste = null;
       Node forrige = null;

       Node (E x) {
           data = x;
       }
   }

   protected Node foerste = null, siste = null;
   protected int antall = 0;
   public void settInn(E x) {
       Node nyNode = new Node(x);
       if (antall == 0) {
           foerste = siste = nyNode;
       } else {
           siste.neste = nyNode;
           nyNode.forrige = siste;
           siste = siste.neste;
       }
       antall++;
   }

   // Eksisterende noder fjernes basert på argument til fjern-metoden
   public E fjern (int pos) {
       if (antall == 0 || pos < 0 || pos >= antall) {
           throw new IndexOutOfBoundsException();
       }

       E fjernes;
       if (pos == 0) {
           fjernes = foerste.data;
           foerste = foerste.neste;
           if (foerste == null) {
               siste = null;
           }
       } else {
           Node p = foerste;
           for (int i = 0;  i < pos; i++) {
               p = p.neste;
           }
           fjernes = p.data;

           if (p == siste) {
               siste = p.forrige;
           } else {
               p.neste.forrige = p.forrige;
           }
           p.forrige.neste = p.neste;

       }
```

```
        antall--;
        return fjernes;
    }
}
```

2.a)

```
class Person implements Comparable<Person> {
    private String navn;
    private int alder;

    public Person(String navn, int alder) {
        this.navn = navn;
        this.alder = alder;
    }

    @Override
    public int compareTo(Person annenPerson) {
        return this.alder - annenPerson.alder;
    }
}
```

2.b)

```
public class Beholder<T extends Comparable<T>> implements Comparable<Beholder<T>> {
    private T verdi;

    public Beholder(T verdi) {
        this.verdi = verdi;
    }

    @Override
    public int compareTo(Beholder<T> annenBeholder) {
        return this.verdi.compareTo(annenBeholder.verdi);
    }
}
```

I dette eksemplet er `Beholder` en generisk klasse som kan inneholde et hvilket som helst objekt så lenge objektet implementerer Comparable grensesnittet. Dette gjør at vi kan sammenligne `Beholder`-objekter basert på verdiene de holder, ved å bruke verdienes egne compareTo-metoder.

2.c)

I oppgave a), `Person`-klassen er ikke-generisk og compareTo-metoden er spesifikk for sammenligning av `Person`-objekter basert på et spesifikt attributt (i dette tilfellet alder).

I oppgave b), `Beholder`-klassen er generisk og designet for å kunne holde og sammenligne objekter av hvilken som helst type (som kan sammenlignes). Dette gir større fleksibilitet og gjenbrukbarhet.

2.d)

Sammenligning av strenger (String-klassen): Når strenger sammenlignes med compareTo-metoden, skjer sammenligningen leksikografisk (basert på Unicode-verdien av hver karakter i strengene). Det betyr at det første tegnet i hver streng sammenlignes, og hvis de er

forskjellige, bestemmer forskjellen i deres Unicode-verdier resultatet av sammenligningen. Hvis de er like, sammenlignes det neste tegnet, og så videre. “A” er mindre enn “B”. Store bokstaver er også sett på som mindre enn små, altså “A” < “a”.

Sammenligning av heltall (Integer-klassen): Når heltall sammenlignes med compareTo-metoden, sammenlignes deres numeriske verdier direkte, og resultatet indikerer om det første tallet er mindre enn, likt med, eller større enn det andre tallet.

4.a) Referansevariabler vs. Primitive Variabler

Likheter:

- Både for primitive typer og referansevariabler kan verdier eller referanser tildeles ved bruk av =.

Forskjeller:

- Verdi vs. Referanse: Primitive variabler holder direkte på dataens verdi (som int, double, boolean, char), mens referansevariabler lagrer adressen til hvor objektet er lagret i minnet, ikke selve verdien direkte.
- Standardverdier: Uinitialiserte primitive variabler har standardverdier (som 0 for int, false for boolean), mens uinitialiserte referansevariabler har standardverdien null.
- Minnebruk: Primitive datatyper har ofte lavere minnebruk sammenlignet med objekter, siden objekter kan inneholde flere primitive verdier samt referanseverdier.

4.b) Casting og Typekonvertering

Casting refererer til å eksplisitt konvertere en variabel fra en datatype til en annen. Dette må du gjøre hvis du vil caste “ned” til en subklasse.

Implisitt Typekonvertering (Automatisk): Skjer f.eks. når du har en variabel av superklassen til en klasse.

Eksempel:

```
class Forelder {}

class Barn extends Forelder {}

class Hovedprogram {
   public static void main(String[] args) {
       // Implisitt typekonvertering fra Barn til Forelder
       Forelder f = new Barn();

       // Eksplisitt typekonvertering (casting) fra Forelder til Barn
       Barn b = (Barn) f;
   }
}
```

4.c) Statiske vs. Ikke-statiske Variabler og Metoder

Statiske Variabler:

- Klassevariabler: De er knyttet til klassen selv, ikke til noe spesifikt objekt av klassen. Dette betyr at en statisk variabel deler samme verdi blant alle instanser av klassen.

Ikke-statiske Variabler (Instansvariabler):

- Unik per objekt: Hvert objekt av klassen har sin egen kopi av variabelen, som kan ha ulik verdi for ulike objekter (f.eks. alder).

Statiske Metoder:

- Tilhørighet: Kan kalles uten å lage et objekt av klassen.
- Begrensninger: Kan ikke bruke ikke-statiske variabler og metoder i klassen fordi de er knyttet til spesifikke objekter.

Ikke-statiske Metoder (Instansmetoder):

- Tilhørighet: Krever et objekt av klassen som man kaller metoden på.
- Fleksibilitet: Kan bruke både statiske og ikke-statiske variabler/metoder av klassen.