Løsningsforslag, gruppetime 6

Oppgave 1 - Løsningsforslag

```
// Comparable Person
class Person implements Comparable<Person> {
  private String fornavn, etternavn;
  private int alder;

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

  public int compareTo(Person annen){
    if(alder == annen.hentAlder()){
        return this.toString().compareTo(annen.toString());
    }
    return alder - annen.hentAlder();
  }

  public String toString(){
    return etternavn + ", " + fornavn;
  }

  public int hentAlder(){
    return alder;
  }
}
```

Oppgave 2 - Løsningsforslag

```
class FIFOListe<T> implements Iterable<T> {

  private Node hode;
  private Node hale;

  private class Node {
    Node neste;
    T data;

    Node(T data) {
      this.data = data;
    }
  }

  public int storrelse() {
    int teller = 0;
    for (Node tmp = hode; tmp != null; tmp = tmp.neste) {
      teller++;
    }
    return teller;
  }
```

```
  public boolean erTom() {
    return hode == null;
  }

  public void leggTil(T t) {
    Node nyNode = new Node(t);
    if (erTom()) {
      hode = nyNode;
    }
    else {
      hale.neste = nyNode;
    }
    hale = nyNode;
  }

  public void fjernAlt() {
    hode = hale = null;
  }

  public T pop() {
    Node returnode = hode;
    hode = hode.neste;
    return returnode.data;
  }

  public void skrivUtListe() {
    Node tmp = hode;
    System.out.println("[");
    while (tmp != null) {
      System.out.println("\t" + tmp.data);
      tmp = tmp.neste;
    }
    System.out.println("]");
  }

  public Iterator<T> iterator() {
    return new FIFOIterator();
  }

  private class FIFOIterator implements Iterator<T> {
    Node denne = hode;

    @Override
    public boolean hasNext() {
      return denne != null;
    }

    @Override
    public T next() {
      Node tmp = denne;
      denne = denne.neste;
      return tmp.data;
    }
  }
}
```

```
class TestFIFO {
  public static void main(String[] args) {
    FIFOListe<String> liste = new FIFOListe<>();

    liste.leggTil("Hei");
    liste.leggTil("på");
    liste.leggTil("deg");
    liste.leggTil("!");

    System.out.println("\nBruker skrivUtListe() ... ");
    liste.skrivUtListe();

    System.out.println("\nBruker iterator ... ");
    for (String string : liste) {
      System.out.print(string + " ");
    }
    System.out.println();

    System.out.println("\nBruker pop() ... ");
    System.out.print(liste.pop() + " ");
    System.out.print(liste.pop() + " ");
    System.out.print(liste.pop());
    System.out.println(liste.pop());
  }
}
```

Oppgave 3a - Løsningsforslag

```
class EnkeltlenketListe {
    private Node forste;
    private Node siste;

    public void leggTilForst(String verdi) {
        Node node = new Node(verdi);
        if(forste != null) {
            node.neste = forste;
        } else {
            siste = node;
        }
        forste = node;
    }

    public void leggTilSist(String verdi) {
        Node node = new Node(verdi);
        if(siste != null) {
            siste.neste = node;
        } else {
            forste = node;
        }
        siste = node;
    }
```

```
    public String fjern(String verdi) {
        Node tmp = forste, forrige = null;
        while (tmp != null && !tmp.verdi.equals(verdi)) {
            forrige = tmp;
            tmp = tmp.neste;
        }
        if (tmp == null) return null;

        String fjernetVerdi = tmp.verdi;
        if (tmp == forste) forste = tmp.neste;
        if (tmp == siste) siste = forrige;
        if (forrige != null) forrige.neste = tmp.neste;
        return fjernetVerdi;
    }

    private class Node {
        String verdi;
        Node neste;
        Node(String verdi) { this.verdi = verdi; }
    }
}
```

Oppgave 3b - Løsningsforslag

```
class DobbeltlenketListe {
    private Node forste;
    private Node siste;

    public void leggTilForst(String verdi) {
        Node node = new Node(verdi);
        if(forste != null) {
            node.neste = forste;
            forste.forrige = node;
        } else {
            siste = node;
        }
        forste = node;
    }

    public void leggTilSist(String verdi) {
        Node node = new Node(verdi);
        if(siste != null) {
            siste.neste = node;
            node.forrige = siste;
```

```
        } else {
            forste = node;
        }
        siste = node;
    }

    public String fjern(String verdi) {
        Node tmp = forste;
        while (tmp != null && !tmp.verdi.equals(verdi)) {
            tmp = tmp.neste;
        }
        if (tmp == null) return null;

        String fjernetVerdi = tmp.verdi;
        if (tmp == forste) forste = tmp.neste;
        if (tmp == siste) siste = tmp.forrige;
        if (tmp.neste != null) tmp.neste.forrige = tmp.forrige;
        if (tmp.forrige != null) tmp.forrige.neste = tmp.neste;
        return fjernetVerdi;
    }

    private class Node {
        String verdi;
        Node neste;
        Node forrige;
        Node(String verdi) { this.verdi = verdi; }
    }
}
```

Oppgave 3c, 3d og 3e - Løsningsforslag

c) Når vi legger til en node foran i en enkeltlenket liste, lar vi den nye noden peke på den gamle første noden. Når vi legger til en node bak i en enkeltlenket liste, lar vi den gamle siste noden peke på den nye noden. Når vi legger til en node foran i en dobbeltlenket liste, oppdaterer vi neste-pekeren til den nye noden til å peke på den nåværende første noden. Deretter oppdaterer vi forrige-pekeren til den nåværende første noden til å peke på den nye noden. Når vi legger til en node bak i en dobbeltlenket liste, oppdaterer vi forrige-pekeren til den nye noden til å peke på den nåværende siste noden. Deretter oppdaterer vi neste-pekeren til den nåværende siste noden til å peke på den nye noden. For begge listetyper må vi huske å oppdatere første-pekeren eller siste-pekeren.

d) I begge metodene ser vi etter noden med en gitt verdi. I den enkeltlenkede listen trenger vi også å holde styr på den forrige noden slik at vi kan fjerne noden ved å oppdatere neste-pekeren fra den forrige noden. Alternativt kan vi i fjern-metoden til en enkeltlenket liste stoppe på noden før noden som skal fjernes. Dette eliminerer behovet for å lagre referansen til den forrige noden i en egen variabel. I den dobbeltlenkede listen kan vi fjerne noden direkte ved å oppdatere neste-pekeren fra den forrige noden og forrige-pekeren fra den neste noden.

e) Dobbeltlenkede lister er hensiktsmessige når du ofte trenger å navigere eller endre listen i begge retninger, siden hver node beholder en referanse til både den foregående og påfølgende noden. Enkeltlenkede lister er mer plassbesparende, siden de bare lagrer en enkelt peker per node, og kan være tilstrekkelige hvis du bare trenger å iterere gjennom listen i en retning.