

Prøveeksamen 2021 / Eksamen 2012 (oppgave 1-2c, 2e, 2g)

- Hvordan ville jeg ha løst den?
- Hva ville jeg lagt vekt på som sensor?
- Hva er relevant fagstoff?

All Java-koden blir lagt ut på semestersiden.

For innspill og spørsmål underveis, ta kontakt på chat-en med Siri Moe Jensen eller Stein Gjessing.

Dag Langmyhr (dag@ifi.uio.no)

Eksamensoppgaver

Oppgaven består typisk av slike oppgaver:

① **Modellering**

Gitt et scenario, hvilke klasser og interfacer trenger vi for å kunne representere det i et oo-program.

② **Datastrukturer og programmering**

Velge riktig datastruktur til et problem, og så programmere det hele.

③ **Programmering med tråder**

Vise en alternativ løsning på en del av prosjektet med tråder.

④ **GUI-programmering**

Utvide prosjektet med et GUI-grensesnitt.



Innsending

Det er mulig at det blir hjemmeeksamen. For å kunne levere svaret, må vi kombinere svarfilene til en ZIP-fil:

Linux zip oppg1.zip Kode.java fig.pdf ...

MacOS Velg filer og control+klikk ⇒
Compress/Komprimer

Windows Velg filer og høyreklikk ⇒ New ⇒ Compressed
(zipped) folder

(Se også <https://www.uio.no/studier/eksamen/innlevering/levere-besvarelsen/zipfiler/hvordan-levere-zipfiler.html>.)

Hva er problemet?

Oppgave 1

Emballasjefabrikken *Renpakk* skal lage et nytt datasystem (i Java) for å administrere sine produkter, og du har fått i oppdrag å lage deler av dette systemet.

Alt *Renpakk* produserer kalles Emballasje, og det er fire hovedtyper: Glassemballasje, Metallemballasje, Plastemballasje og Pappemballasje.

Noe av emballasjen til *Renpakk* er det pant på, og noe emballasje er nedbrytbar. Dette er egenskaper som kommer i tillegg til de andre egenskapene emballasjen har.

Av alle produkttypene i klassehierarkiet produseres det for tiden bare ting av disse tre: liten plastflaske med pant, liten nedbrytbar plastflaske med pant og stor nedbrytbar pappflaske med pant.

1a Klassehierarki

Lever en tegning av klassehierarkiet for de tre produkttypene som er beskrevet over som en besvarelse på denne oppgaven. Inkluder også superklasser og eventuelle grensesnitt.



Hva er en klasse egentlig?

En **klasse** er en *modellering* av noe (en ting eller et begrep).

Konkret

En klasse inneholder

- en **representasjon** (dvs klasse- og instansvariabler)
- ingen, én eller flere **konstruktører**
- diverse **operasjoner** (dvs metoder)

Hvilke klasser trenger jeg?

Substantivmetoden for å finne klasser

Emballasjefabrikken Rempakk skal lage et nytt **datasystem** (i Java) for å administrere sine **produkter**, og du har fått i oppdrag å lage **deler** av dette **systemet**.

Alt Rempakk produserer kalles **Emballasje**, og det er fire hovedtyper: **Glassemballasje**, **Metallemballasje**, **Plastemballasje** og **Pappemballasje**.

Noe av emballasjen til Rempakk er det pant på, og noe emballasje er nedbrytbar. Dette er egenskaper som kommer i tillegg til de andre egenskapene emballasjen har.

Av alle produkttypene i klassehierarkiet produseres det for tiden bare ting av disse tre: **liten plastflaske med pant**, **liten nedbrytbar plastflaske med pant** og **stor nedbrytbar pappflaske med pant**.

1a Klassehierarki

Lever en tegning av klassehierarkiet for de tre produkttypene som er beskrevet over som en besvarelse på denne oppgaven. Inkluder også superklasser og eventuelle grensesnitt.



Hvilke klasser trenger jeg?

Hva er *ikke* klasser i vår modell?

- «emballasjefabrikk» (men kunne kanskje vært det i en annen sammenheng)
- «datasystem»
- «produkt»
- «del»
- «system»

Hva er et interface?

Et **interface** angir en *egenskap*. Noen klasser har denne egenskapen, andre har den ikke.

Konkret

Et interface inneholder

- navn og parametre på diverse **operasjoner** (dvs metoder) vi garanterer at en implementasjon inneholder.

Hvilke interface trenger jeg?

Hvilke *egenskaper* finner vi?

Emballasjefabrikken *Renpakk* skal lage et nytt datasystem (i Java) for å administrere sine produkter, og du har fått i oppdrag å lage deler av dette systemet.

Alt *Renpakk* produserer kalles *Emballasje*, og det er fire hovedtyper: *Glassemballasje*, *Metallemballasje*, *Plastemballasje* og *Pappemballasje*.

Noe av emballasjen til *Renpakk* er det **pant** på, og noe emballasje er **nedbrytbar**. Dette er egenskaper som kommer i tillegg til de andre egenskapene emballasjen har.

Av alle produkttypene i klassehierarkiet produseres det for tiden bare ting av disse tre: **liten** plastflaske med pant, **liten** nedbrytbar plastflaske med pant og **stor** nedbrytbar pappflaske med pant.

1a Klassehierarki

Lever en tegning av klassehierarkiet for de tre produkttypene som er beskrevet over som en besvarelse på denne oppgaven. Inkluder også superklasser og eventuelle grensesnitt.



Hvilke interface trenger jeg?

Hvilke interface-er trenger vi?

- Nedbrytbar
- Pant

(Vi *kunne* også definert interface for **liten** og **stor**, men vi har ingen metoder å gi dem.)

Hva er en subklasse?

En **subklasse** er en *spesialisering* av en klasse, noe som skiller den av superklassen og eventuelt andre subklasser.

Konkret

En **subklasse** kan inneholde

- en utvidet representasjon (dvs flere variabler)
- nye konstruktører
- flere operasjoner (dvs metoder)
- redefinering av operasjoner (dvs metoder med @Override)

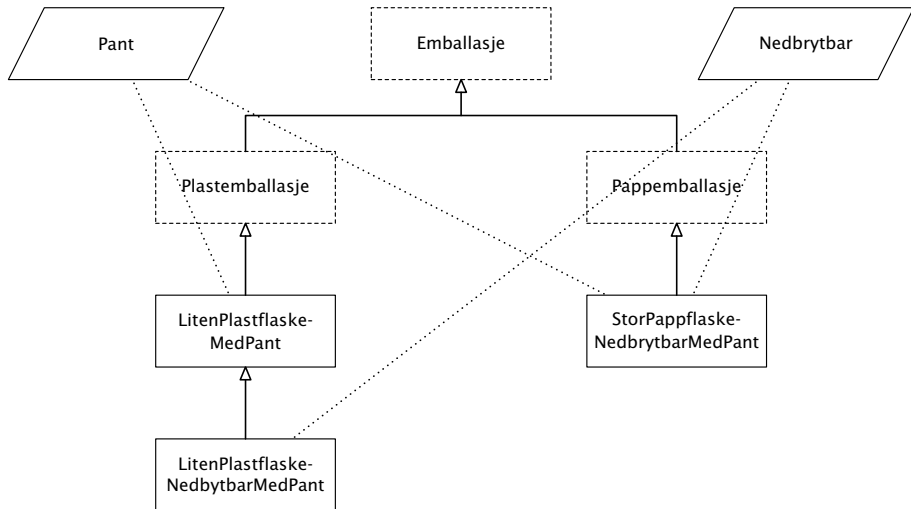
Hva er sammenhengen mellom klassene?

- Alt er **Emballasje**; den er en naturlig superklasse i vår modell.
- **Glassemballasje**, **Metallemballasje**, **Plastemballasje** og **Pappemballasje** er ulike slags emballasje; de må være subklasser av **Emballasje**.
- **LitenPlastflaskeMedPant** er en slags plastflaske, så den er subklasse av **PlastEmballasje** og den må implementere **Pant**.
- **LitenPlastflaskeNedbrytbarMedPant** er en liten plastflaske med pant, så den er subklasse av **LitenPlastflaskeMedPant**; den må implementere **Nedbrytbar**.
- **StorNedbrytbarPappflaskeMedPant** er naturlig nok en slags pappemballasje; den må være subklasse av **Pappemballasje** og den må implementere både **Nedbrytbar** og **Pant**.

(Vi kunne også definert en **StorPappflaskeMedPant** men det spørres ikke etter den i oppgaven.)



Å sette alt sammen



Implementasjon

All emballasje har et **volum** (i kubikkcentimeter) og en tekst (String) som er en **produksjonsidentifikator**.

Plastemballasje har ingen flere egenskaper enn Emballasje mens Pappemballasje i tillegg har en **vekt** (i gram).

Når det er pant på en ting, må en kunne vite hvor stor **panten** er (i antall øre) og en kode (en tekst) som identifiserer **returordningen**.

Når noe er nedbrytbart, må en kunne vite hvor **lenge** (hvor mange år) det tar før tingen er gått i oppløsning.

Programmer de tre klassene nevnt i 1a og eventuelle superklasser og grensesnitt. Alle variable i alle klasser skal få verdier i det objektene opprettes. Det er derfor viktig at alle klasser har konstruktører med parametre der disse verdiene kan oppgis, unntatt panten på **små flasker** som alltid har samme verdi, en konstant med verdi **100 øre**. Når en konstruktør utføres, skal det skrives ut i terminalvinduet: «Konstruktoeren til klassen Xxx utfoeres», der Xxx er navnet på klassen.

Nå kan vi programmere

```
abstract class Emballasje {
    public double volum;
    public String produktId;

    public Emballasje(double vol, String id) {
        volum = vol; produktId = id;
        System.out.println("Konstruktoeren til klassen Emballasje utfoeres.");
    }
}
```

```
abstract class Plastemballasje extends Emballasje {
    public Plastemballasje(double vol, String id) {
        super(vol, id);
        System.out.println("Konstruktoeren til klassen Plastemballasje utfoeres.");
    }
}
```

```
abstract class Pappemballasje extends Emballasje {
    public double vekt;

    public Pappemballasje(double vol, String id, double vekt) {
        super(vol, id);
        this.vekt = vekt;
        System.out.println("Konstruktoeren til klassen Pappemballasje utfoeres.");
    }
}
```

Nå kan vi programmere

```
interface Pant {  
    int finnPant();  
    String finnReturordning();  
}  
  
interface Nedbrytbar {  
    double finnNedbrytningstid();  
}
```


Nå kan vi programmere

```
class LitenPlastflaskeMedPant extends Plastemballasje implements Pant {
    public String returordning;

    public LitenPlastflaskeMedPant(double vol, String id, String retur) {
        super(vol, id);
        returordning = retur;
        System.out.println("Konstruktoeren til klassen " +
            "LitenPlastflaskeMedPant utfoeres.");
    }

    public int finnPant() {
        return 100;
    }

    public String finnReturordning() {
        return returordning;
    }
}
```

Nå kan vi programmere

```
class LitenPlastflaskeNedbrytbarMedPant extends LitenPlastflaskeMedPant
    implements Nedbrytbar
{
    private double nedbrytningstid;

    public LitenPlastflaskeNedbrytbarMedPant(double vol, String id,
        String retur, double nedbryt)
    {
        super(vol, id, retur);
        nedbrytningstid = nedbryt;
        System.out.println("Konstruktoeren til klassen " +
            "LitenPlastflaskeNedbrytbarMedPant utfoeres.");
    }

    public double finnNedbrytningstid() {
        return nedbrytningstid;
    }
}
```

Nå kan vi programmere

```

class StorPappflaskeNedbrytbarMedPant extends Pappemballasje
    implements Nedbrytbar, Pant
{
    private int pant;
    private String returordning;
    private double nedbrytningstid;

    public StorPappflaskeNedbrytbarMedPant(double vol, String id, double vekt,
        int pant, String retur, double nedbryt)
    {
        super(vol, id, vekt);
        this.pant = pant;
        returordning = retur;
        nedbrytningstid = nedbryt;
        System.out.println("Konstruktoeren til klassen " +
            "StorPappflaskeNedbrytbarMedPant utfoeres.");
    }

    public int finnPant() {
        return pant;
    }

    public String finnReturordning() {
        return returordning;
    }

    public double finnNedbrytningstid() {
        return nedbrytningstid;
    }
}

```



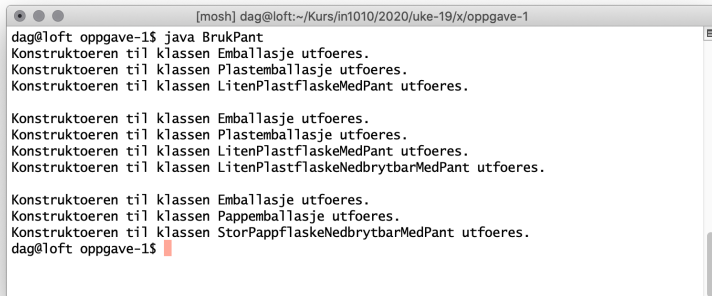
Implementasjon

Skriv også en klasse (en hovedklasse) kalt BrukPant med en main-metode som oppretter ett objekt av hver av de klassene det kan lages objekter av. Finn på noen passende verdier til alle variablene i alle objektene. Programmet ditt skal kunne kompileres og kjøres.

```
class BrukPant {
    public static void main(String[] arg) {
        Emballasje lqp = new LitenPlastflaskeMedPant(0.5, "LQP-h", "Plastretur");
        System.out.println();
        Emballasje lqnp = new LitenPlastflaskeNedbrytbarMedPant(0.5,
            "LQNP-h", "Plastretur", 27);
        System.out.println();
        Emballasje spnp = new StorPappflaskeNedbrytbarMedPant(3,
            "SPNP-3", 0.2, 5, "Papirkonteiner", 7);
    }
}
```

Implementasjon

Kjør programmet du har laget og ta et bilde eller en skjermdump som viser kjøringen av dette programmet med alle linjene som blir skrevet ut i terminalvinduet.



```
[mosh] dag@loft:~/Kurs/in1010/2020/uke-19/x/oppgave-1
dag@loft oppgave-1$ java BrukPant
Konstruktoeren til klassen Emballasje utfoeres.
Konstruktoeren til klassen Plastemballasje utfoeres.
Konstruktoeren til klassen LitenPlastflaskeMedPant utfoeres.

Konstruktoeren til klassen Emballasje utfoeres.
Konstruktoeren til klassen Plastemballasje utfoeres.
Konstruktoeren til klassen LitenPlastflaskeMedPant utfoeres.
Konstruktoeren til klassen LitenPlastflaskeNedbrytbarMedPant utfoeres.

Konstruktoeren til klassen Emballasje utfoeres.
Konstruktoeren til klassen Pappemballasje utfoeres.
Konstruktoeren til klassen StorPappFlaskeNedbrytbarMedPant utfoeres.
dag@loft oppgave-1$
```

Oppgave 2 Datastruktur

I denne oppgaven skal vi utvikle en klasse med en ny form for lagringsstruktur. Klassen skal hete Frekvens og skal kunne lagre **inntil 1000 tekster** (String-er). Konstruktøren skal ha én parameter: en array med tekstene som skal lagres. **Tekstene er alfabetisk sortert.**

Klassen Frekvens skal ha tre metoder:

- 1 void-metoden finnFlest skal finne hvilken tekst som forekommer flest ganger og lagre teksten og antallet forekomster i instansvariabler i Frekvens-objektet.
(Hvis flere tekster forekommer like mange ganger, er det det samme hvilken som lagres.)
- 2 String-metode hentFlest henter teksten som ble lagret av et tidligere kall på finnFlest (se forrige punkt).
- 3 int-metoden hentAntall som henter antallet forekomster funnet i et tidligere kall på finnFlest (se punkt 1).

2a Fritt valgt datastruktur

Implementer klassene Frekvens og BrukFrekvens forklart over. I denne deloppgaven kan du bruke arrayer eller hvilke klasser du vil fra Javas bibliotek til å lagre tekstene.

Datastrukturer i IN1010

array er et sammenhengende området i minnet der man kan regne seg frem til elementers posisjon utfra en heltallsindeks.

+: raske, god notasjon

ArrayList er en fleksibel array-lignende Java-klasse.

+: fleksibel størrelse

lister ...

HashMap ...



HashMap (Big Java 15.4)

Det er to måter å forklare hva en HashMap er:

- en slags ArrayList der indeks er en String¹ istedenfor en int.
- en database der vi lagrer verdier med tilhørende nøkler.

Nyttige operasjoner

put(k,v) lagrer verdien v med nøkkel/indeks k.

get(k) henter verdien med nøkkel/indeks k; gir **null** om den ikke finnes.

size() gir antall verdier.

keySet() henter alle nøklene/indeksene; nyttig i for-løkker.

¹Indeksen til en HashMap kan faktisk være en hvilken som helst klasse, men i IN1010 vil vi bare bruke String.

Hva er egentlig en HashMap?

Løsning med HashMap

Min løsning er å lagre tekstene i en HashMap der tekstene er nøkler og verdiene er antall ganger de forekommer.

```
class Frekvens {
    HashMap<String,Integer> data = new HashMap<>();
    String flest = null;
    int flestAntall = 0;

    Frekvens(String[] initData) {
        for (String s: initData) {
            int n = 0;
            Integer v = data.get(s);
            if (v != null) n = v;
            data.put(s, n+1);
        }
    }
}
```

NB: HashMap<String,Integer>

Verdiene i en HashMap kan ikke være primitive typer (dvs int, double etc) så de må *pakkes inn* (Big Java 6.8.5).



Letingen etter den vanligste forekomsten blir helt rett frem:

```
void finnFlest() {
    flest = "";
    flestAntall = 0;
    for (String s: data.keySet()) {
        if (data.get(s) > flestAntall) {
            flest = s; flestAntall = data.get(s);
        }
    }
}
```

De to siste metodene er trivielle:

```
String hentFlest() { return flest; }
int hentAntall() { return flestAntall; }
```

Skriv også en klasse BrukFrekvens med et testprogram; det skal ha en main-metode som setter det hele i gang. Testprogrammet skal ha én parameter: navnet på en fil med tekster, én tekst per linje. **Denne filen vil være alfabetisk sortert.** Det skal opprette et Frekvens-objekt med innleste data og etterpå skrive ut hvilken tekst som forekommer flest ganger og hvor mange ganger det er.

Hvordan løse programet?

- * Få tak i filnavnet
- * Les inn tekstene og lagre i en array
- * Lag sortert array av riktig lengde
- * Lag et Frekvens-objekt.
- * Kall på finnFlest for å finne vanligste tekst.
- * Skriv ut svaret.

Hva er egentlig en HashMap?

```

class BrukFrekvens {
    public static void main(String[] arg) {
        String[] lager = new String[1000];
        int antILager = 0;

        if (arg.length != 1) {
            System.out.println("Usage: java BrukFrekvens datafil");
            System.exit(1);
        }
        try {
            Scanner s = new Scanner(new File(arg[0]));
            while (s.hasNextLine()) {
                lager[antILager++] = s.nextLine();
            }
        } catch (FileNotFoundException e) {
            System.out.println("Kan ikke lese " + arg[0] + "!");
            System.exit(2);
        }

        String[] lager2 = new String[antILager];
        for (int i = 0; i < antILager; i++) lager2[i] = lager[i];
        Frekvens tekster = new Frekvens(lager2);
        tekster.finnFlest();
        System.out.println("Det vanligste navnet er " + tekster.hentFlest() +
            " (" + tekster.hentAntall() + " forekomster).");
    }
}
    
```



2b Énveis liste

I denne deloppgaven (og alle de etterfølgende deloppgavene) får du ikke lov å benytte ArrayList, HashMap eller andre lagringsstrukturer fra Java-biblioteket.

Du skal nå endre svaret ditt fra forrige deloppgave til å bruke en enkeltlinket liste til å lagre dataene.

En standard løsning med en lokal klasse Node.

```
class Frekvens {
    class Node {
        Node neste = null;
        String navn;

        Node(String n) { navn = n; }
    }

    Node forste = null, siste = null;
    String flest;
    int flestAntall;

    Frekvens(String[] initData) {
        forste = siste = new Node(initData[0]);
        for (int i = 1; i < initData.length; i++)
            siste.neste = siste = new Node(initData[i]);
    }
}
```

Deloppgave 2b

Letingen blir også helt standard:

```
void finnFlest() {
    flest = null;
    flestAntall = 0;

    String denne = "";
    int denneAntall = 0;
    Node p = forste;
    while (p != null) {
        if (p.navn.equals(denne)) {
            denneAntall++;
        } else {
            denne = p.navn; denneAntall = 1;
        }
        if (denneAntall > flestAntall) {
            flest = denne; flestAntall = denneAntall;
        }
        p = p.neste;
    }
}
```

Resten (metodene hentFlest og hentAntall og klassen BrukFrekvens) blir som i forrige deloppgave.



2c Énveis liste med antall

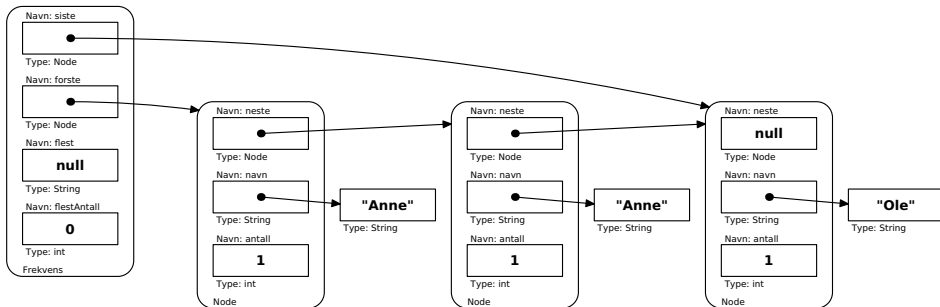
Du skal nå oppdatere koden fra forrige deloppgave til å lagre data i en **enkeltlinket liste** av objekter av en **indre klasse**. I disse objektene skal du ikke bare lagre String-en og nestepekeren, men også **antallet** ganger String-en forekommer. Ved initieringen av Frekvens er dette antallet **1** for alle objektene.

Du skal så utvide klassen Frekvens med en metode komprimer som forkorter listen ved å slå sammen etterfølgende like tekster. (**Husk at alle tekstene er alfabetisk sortert i datafilen.**)

Tegn et eksempel på datastrukturen **før** og **etter** en slik komprimering; velg nok objekter til at strukturen kommer klart frem. Bruk et tegneverktøy du liker, eller tegn på et ark og ta bilde av det.

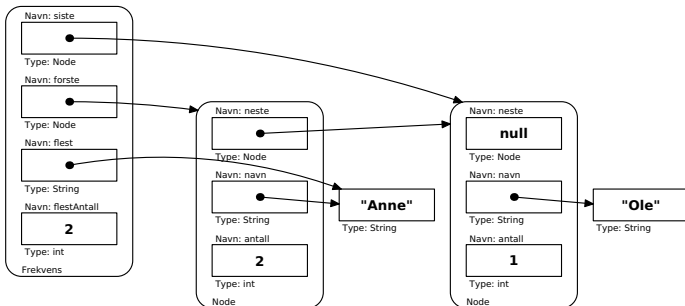
Deloppgave 2c

Før komprimering og leting



Deloppgave 2c

Etter komprimering og leting



Den indre klassen Node blir som i forrige oppgave, bortsett fra at den nå også får et antall. Konstruktøren for Frekvens blir identisk.

```

class Frekvens {
    class Node {
        Node neste = null;
        String navn;
        int antall = 1;
    }
    Node(String n) { navn = n; }
}

Node forste = null, siste = null;
String flest;
int flestAntall;

    Frekvens(String[] initData) {
        forste = siste = new Node(initData[0]);
        for (int i = 1; i < initData.length; i++) {
            siste.neste = siste = new Node(initData[i]);
        }
    }

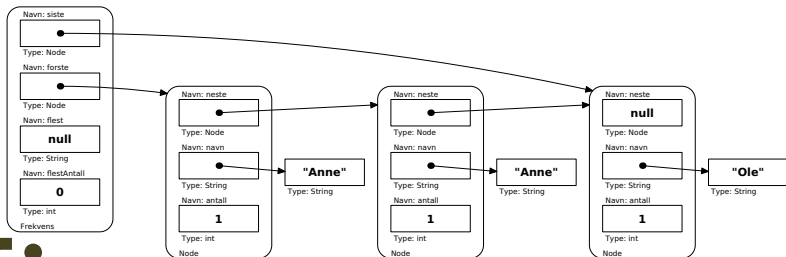
```

Deloppgave 2c

Metoden komprimerer går gjennom listen og fjerner de Node-ene der etterfølgeren har samme navn.

```
void komprimer() {
    Node p1 = forste, p2 = forste.neste;

    while (p2 != null) {
        if (p1.navn.equals(p2.navn)) {
            p1.antall++; p1.neste = p2.neste;
        } else {
            p1 = p2;
        }
        p2 = p2.neste;
    }
}
```



Metoden finnFlest blir nå enklere:

```
void finnFlest() {
    flest = null;
    flestAntall = 0;

    Node p = forste;
    while (p != null) {
        if (p.antall > flestAntall) {
            flest = p.navn; flestAntall = p.antall;
        }
        p = p.neste;
    }
}
```

Og det var det!

2e Generelle data

Du skal nå ta utgangspunkt i koden din fra del 2c (dvs den med metoden komprimer). Gjør klassen Frekvens *generisk* slik at den kan brukes til å lagre ulike typer data, ikke bare String-er.

```
class Frekvens<T extends Comparable<T>> {
    class Node {
        Node neste = null;
        T data;
        int antall = 1;

        Node(T n) {
            data = n;
        }
    }
}
```

Opprett også en egen klasse du kan bruke som data.

```
class Dato implements Comparable<Dato> {
    int aa, m, d;

    Dato(int aa, int m, int d) {
        this.aa = aa; this.m = m; this.d = d;
    }

    @Override
    public int compareTo(Dato x) {
        if (aa != x.aa) return aa - x.aa;
        if (m != x.m) return m - x.m;
        return d - x.d;
    }

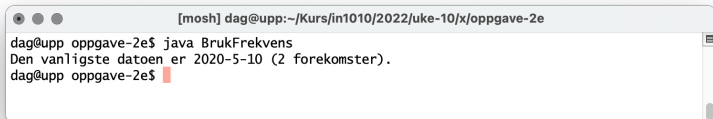
    @Override
    public String toString() {
        return aa + "-" + m + "-" + d;
    }
}
```


Deloppgave 2e

Du skal ikke lese data fra en fil denne gangen, men lage objekter av denne dataklassen og legge disse inn i Frekvens selv. Dette skal gjøres i main-metoden. Lag minst fem dataobjekter.

```
class BrukFrekvens {
    public static void main(String[] arg) {
        Frekvens<Dato> mineData = new Frekvens<>();
        mineData.add(new Dato(2000, 1, 1));
        mineData.add(new Dato(2020, 5, 10));
        mineData.add(new Dato(2020, 5, 10));
        mineData.add(new Dato(2020, 5, 17));
        mineData.add(new Dato(2020, 7, 17));

        mineData.komprimer();
        mineData.finnFlest();
        System.out.println("Den vanligste datoen er " +
            mineData.hentFlest() + " (" +
            mineData.hentAntall() + " forekomster).");
    }
}
```



```
[mosh] dag@upp:~/Kurs/in1010/2022/uke-10/x/opp-gave-2e
dag@upp opp-gave-2e$ java BrukFrekvens
Den vanligste datoen er 2020-5-10 (2 forekomster).
dag@upp opp-gave-2e$
```



2g Frekvensoversikt

Til sist skal du ta utgangspunkt i koden du laget i 2c og utvide den til også å vise en oversikt over hvor mange forekomster det er av unike navn, 2 like navn, 3 like navn osv.

Oppgaven løses greit ved å lage en frekvenstabell:

```
void skrivOversikt() {
    int antForekomster[] = new int[flestAntall+1];
    Node p = forste;
    while (p != null) {
        antForekomster[p.antal]++;
        p = p.neste;
    }

    for (int i = flestAntall; i > 0; i--) {
        if (antForekomster[i] > 0)
            System.out.printf("%3d like navn: %d forekomster\n",
                               i, antForekomster[i]);
    }
}
```

