Oppsummering

Kort gjennomgang av pensum hittil ved å løse halvparten av eksamen 2012.

- Klasser
  - Interface
  - Subklasser
  - Klasseparametre
- Datastrukturer
  - Lister

Eksamen

Dere bør nå kunne løse 80 % av tidligere eksamensoppgaver, dvs alt unntatt oppgaver med *tråder* og *GUI*.

På kurssidene ligger linker til tidligere eksamener og løsningsforslag.

NB!

Tidligere eksamener var 6 timer; fra og med i år er de bare på 4 timer. Oppgavene vil derfor bli en del kortere.

Eksamen 2012: Oppgave 1

Emballasjefabrikken Renpakk skal lage et nytt datasystem 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.

Du får beskjed om å beskrive plastflaske med pant, liten plastflaske med pant og liten nedbrytbar plastflaske med pant. Renpakk har fått et nytt og enestående produkt de er veldig stolt av, nemlig stor nedbrytbar pappflaske med pant, og du skal også ta med en klasse for slike flasker.

Tegn klassehierarkiet for de 9 produkttypene som er beskrevet over. Inkluder også eventuelle grensesnitt.

Hva er en klasse egentlig?

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

Konkret

En klasse inneholder

- en **representasjon** (dvs variabler)
- én eller flere **konstruktører**
- diverse **operasjoner** (dvs metoder)

Hva er programmering?

Ett svar (blant flere riktige) er

Idé **Maskinkode**

OO-språk **C**

Hvilke klasser trenger jeg?

Hvilke *substantiver* finner vi i oppgaveteksten?

Emballasjefabrikken Renpakk skal lage et nytt datasystem 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.

Du får beskjed om å beskrive **plastflaske med pant**, **liten plastflaske med pant** og **liten nedbrytbar plastflaske med pant**. Renpakk har fått et nytt og enestående produkt de er veldig stolt av, nemlig **stor nedbrytbar pappflaske med pant**, og du skal også ta med en klasse for slike flasker.

Hva er *ikke* klasser i vår modell?

- «emballasjefabrikk»
- «datasystem»
- «produkt»
- «del»
- «system»

(Dette bekreftes av at oppgaven ber om **9** klasser.)

Hva er et interface?

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

Konkret

Et interface inneholder

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

Hvilke *egenskaper* finner vi?

Emballasjefabrikken Renpakk skal lage et nytt datasystem 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.

Du får beskjed om å beskrive plastflaske med pant, **liten** plastflaske med pant og **liten** nedbrytbar plastflaske med pant. Renpakk har fått et nytt og enestående produkt de er veldig stolt av, nemlig **stor** nedbrytbar pappflaske med pant, og du skal også ta med en klasse for slike flasker.

Hvilke interface-er trenger vi?

- **Nedbrytbar**
- **Pant**

(Vi *kunne* også definert interface for **liten** og **stor**, men vi dropper det fordi de ikke bringer inn noe nytt.)

Hva er en subklasse?

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

Konkret

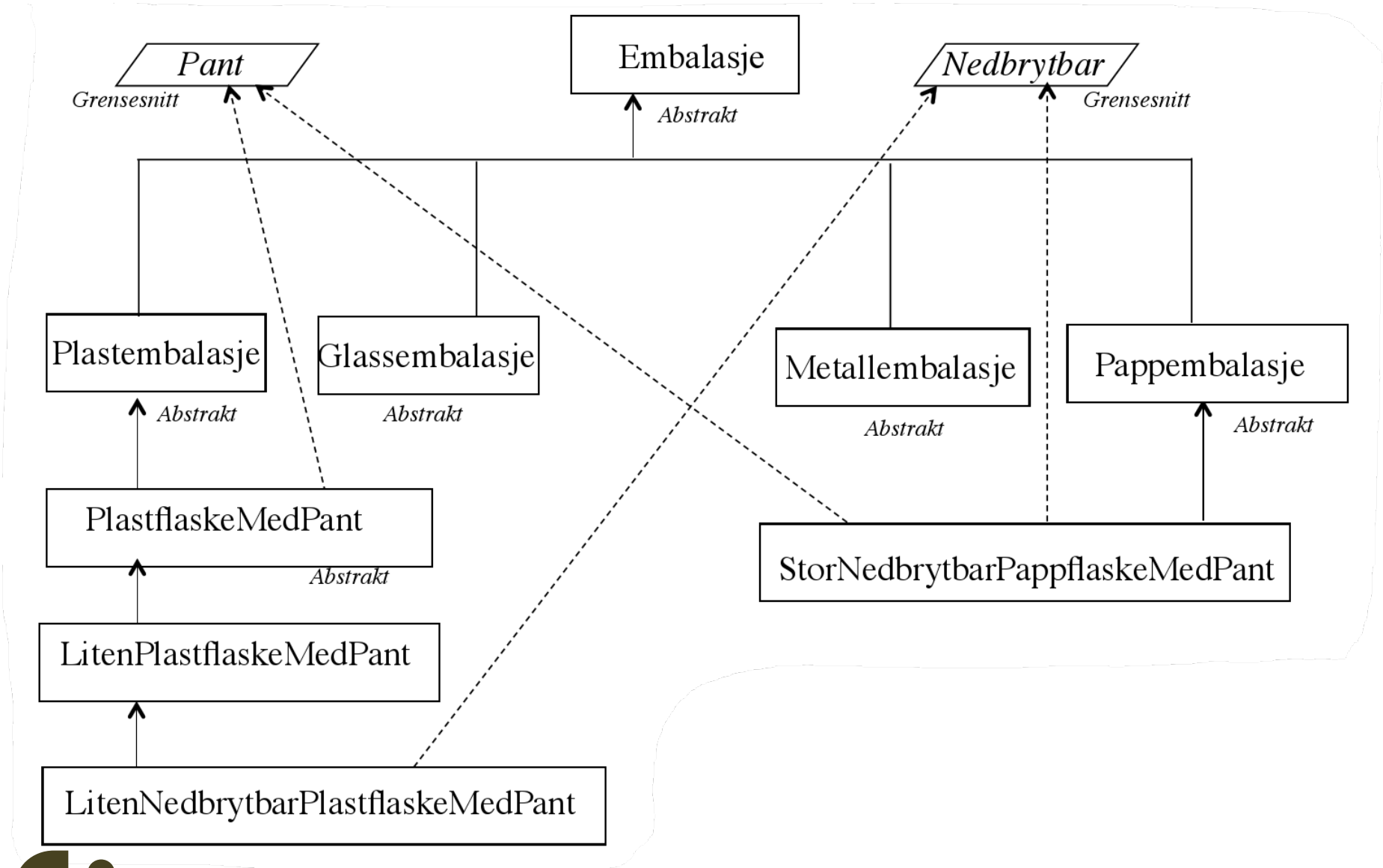
En **subklasse** inneholder

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

Hva er sammenhengen mellom klassene?

- Alt er **Emballasje**; den er superklassen i vår modell.
- **Glassemballasje**, **Metallemballasje**, **Plastemballasje** og **Pappemballasje** er ulike slags emballasje; de må være subklasser av Emballasje.
- **StorNedbrytbarPappflaskeMedPant** er naturlig nok en slags pappemballasje; den må være subklasse av Pappemballasje.
- **PlastflaskeMedPant** er en slags plastflaske; den må være subklasse av Plastemballasje.
- **LitenPlastflaskeMedPant** er en slags plastflaske, så den er subklasse av PlastflaskeMedPant.
- **LitenNedbrytbarPlastflaskeMedPant** er en slags liten plastflaske med pant, så den er subklasse av LitenPlastflaskeMedPant.

Å sette alt sammen



Oppgaven videre

Programmer 7 av de 9 klassene i klassehierarkiet og eventuelle grensesnitt. Du skal ikke programmere klasser for Glassemballasje og Metallemballasje. Alle variable i alle klasser skal få verdier i det objektene opprettes. Det er derfor viktig at alle klasser har konstruktører med parametere der disse verdiene kan oppgis, unntatt panten på små flasker som alltid har samme verdi, en konstant med verdi 100 øre.

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 gjenstand, 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 gjenstanden er gått i oppløsning.

Nå kan vi programmere

```
abstract class Emballasje {
 int volum;  // i kubikkcentimeter
 String produsent;

 Emballasje(int vol, String prod) {
     volum = vol; produsent = prod;
 }
}
```

```
abstract class Plastemballasje extends Emballasje {

    Plastemballasje(int vol, String prod) {
        super(vol,prod);
    }
}
```

```
abstract class Pappemballasje extends Emballasje {
    int vekt;  // i gram

    Pappemballasje(int vekt, int vol, String prod) {
        super(vol,prod);
        this.vekt = vekt;
    }
}
```

Nå kan vi programmere

```
interface Pant {
    int panteverdi();  // i øre
    String returordning();
}
```

```
interface Nedbrytbar {
    int antallAar();
}
```

```
abstract class PlastflaskeMedPant extends Plastemballasje
        implements Pant {
    String retur;

    PlastflaskeMedPant(String ret, int vol, String prod) {
        super(vol,prod);
        retur = ret;
    }

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

```
class LitenPlastflaskeMedPant extends PlastflaskeMedPant {
    final static int PANTEVERDI = 100;

    LitenPlastflaskeMedPant (String ret, int vol, String prod) {
        super(ret,vol,prod);
    }

    public int panteverdi() { return PANTEVERDI; }
}
```

Nå kan vi programmere

```
class LitenNedbrytbarPlastflaskeMedPant extends LitenPlastflaskeMedPant
        implements Nedbrytbar {
    int antAar;

    LitenNedbrytbarPlastflaskeMedPant (String ret, int vol,
                                       String prod, int aar) {
        super(ret,vol,prod);
        antAar = aar;
    }

    public int antallAar() { return antAar; }
}
```

Nå kan vi programmere

```
class StorNedbrytbarPappflaskeMedPant extends Pappemballasje
        implements Pant, Nedbrytbar {
    int pantV;
    int antallAa;
    String returOrd;

    StorNedbrytbarPappflaskeMedPant(String ret, int vol, String prod,
                                    int vekt, int pnt, int antA) {
        super(vekt,vol,prod);
        returOrd = ret;
        pantV = pnt;
        antallAa = antA;
    }

    public int panteverdi() { return pantV; }
    public String returordning() { return returOrd; }
    public int antallAar() { return antallAa; }
}
```

Oppgave 2

I denne oppgaven skal du programmere en klasse med navn Frekvens som beskriver en beholder. Du skal lage klassen generisk, og du skal ikke bruke noen klasser fra Java-biblioteket.

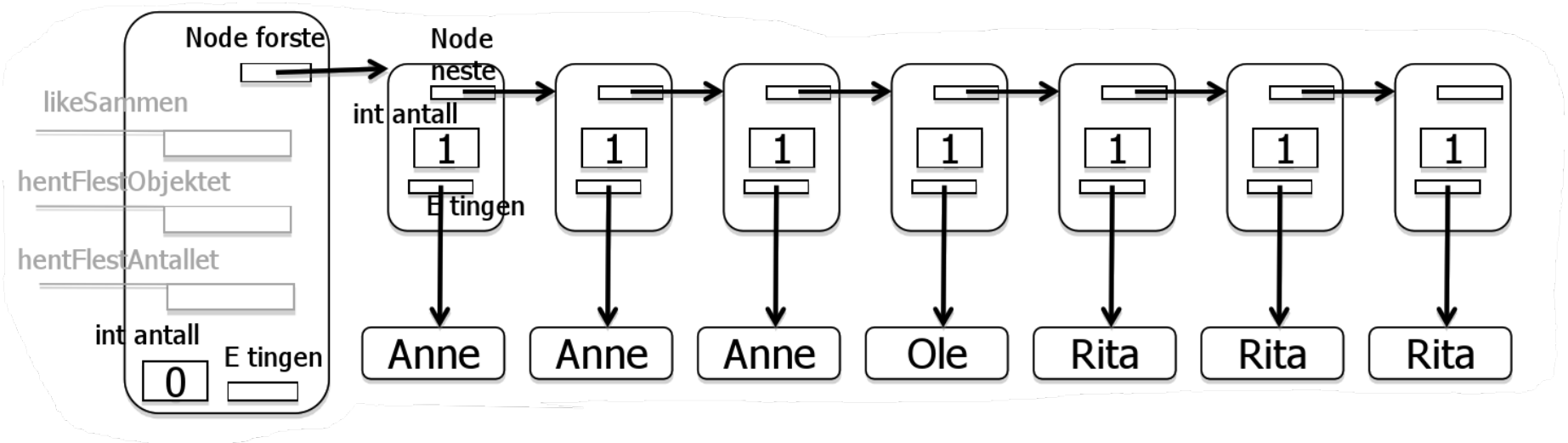
Formålet med beholderen Frekvens er å slå sammen alle like objekter, samtidig som den holder orden på antallet slike like objekter det var opprinnelig. F.eks. hvis beholderen skal ta vare på String: Gir vi beholderen tekstene Anne, Anne, Anne, Ole, Rita, Rita, Rita, vil resultatet bli (etter sammenslåingen) Anne 3, Ole 1, Rita 3.

For å finne ut om to objekter er like skal du bruke metoden equals. Når vi slår sammen to objekter vil vi kaste det ene. Hvilket vi kaster og hvilket vi beholder er likegyldig.

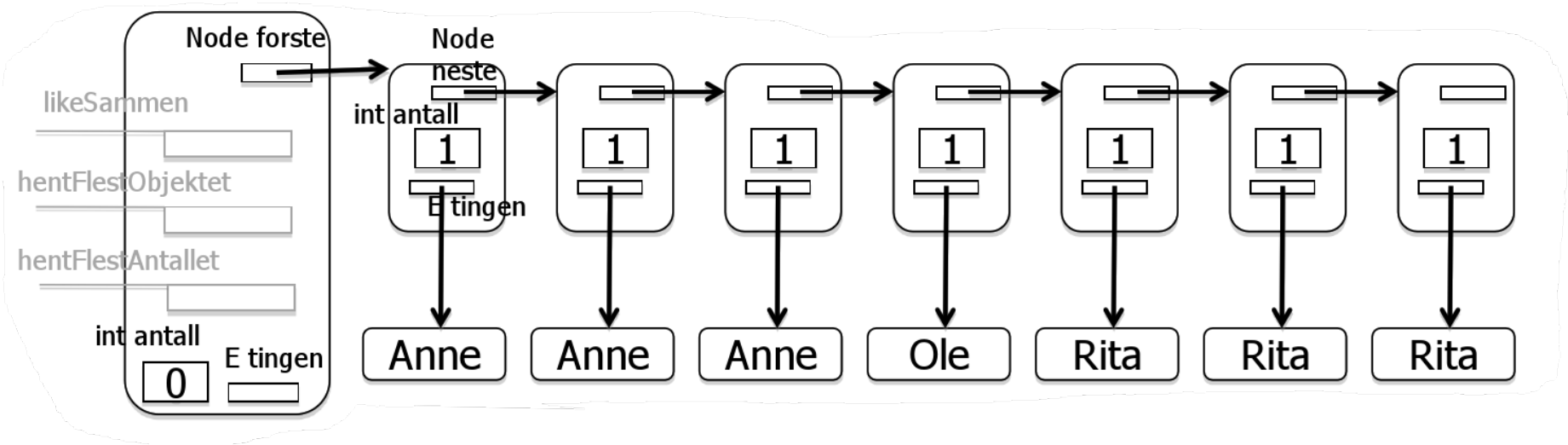
Du skal i denne oppgaven legge alle objektene inn i beholderen ved hjelp av konstruktøren som skal ha som parameter en tabell (array) med pekere til objektene som skal legges inn. Vi antar at tabellen er ferdig sortert.

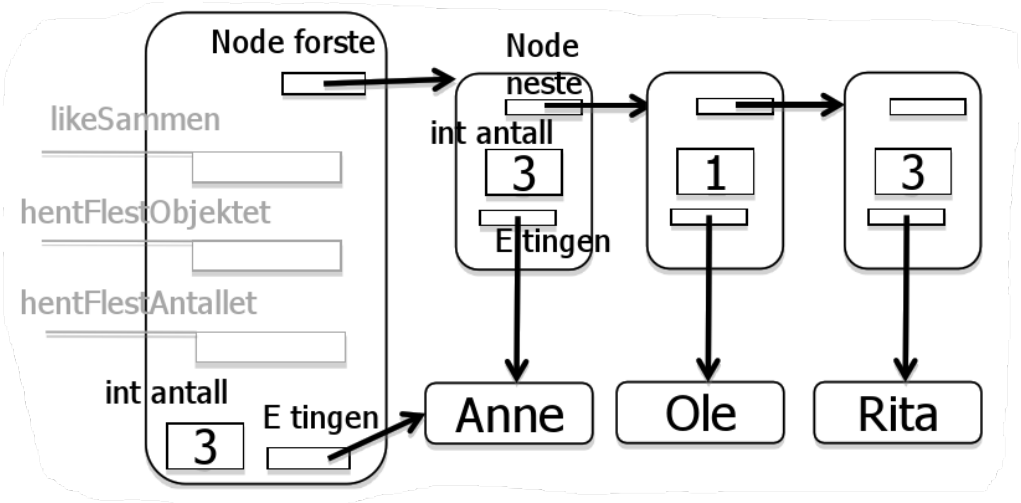
Alt blir mer oversiktlig med en tegning av datastrukturen

Tegn datastrukturen i et objekt av klassen Frekvens, ved først å vise hvordan datastrukturen ser ut rett etter at objektet er opprettet (og konstruktøren er utført), og deretter hvordan datastrukturen ser ut etter at metoden likeSammen er kalt (se nedenfor). Bruk gjerne eksemplet vist over.



Slik ser det ut før og etterpå





Hva er generiske klasser (= klasseparametre)?

Klasser kan gis en parameter som er navnet på en annen klasse:

```
class Par<T> {
    T a, b;

    Par(T a, T b) {
        this.a = a;  this.b = b;
    }

    String join() {
        return a.toString() + "&" + b.toString();
    }
}
```

```
class TestPar {
    public static void main(String[] a) {
        Par<String> s = new Par<>("abc", "xyz");
        Par<Integer> i = new Par<>(12, 17);

        System.out.println("s = " + s.join());
        System.out.println("i = " + i.join());
    }
}
```

```
$ javac TestPar.java
$ java TestPar
s = abc&xyz
i = 12&17
$
```

Hva er innpakking («boxing»)?

Problem

Klasseparametre kan bare være andre klasser, ikke primitive typer som int, float etc.

Løsning

Lag klasser for hver av av de primitive typene: Integer, Float etc.

Forbedret løsning

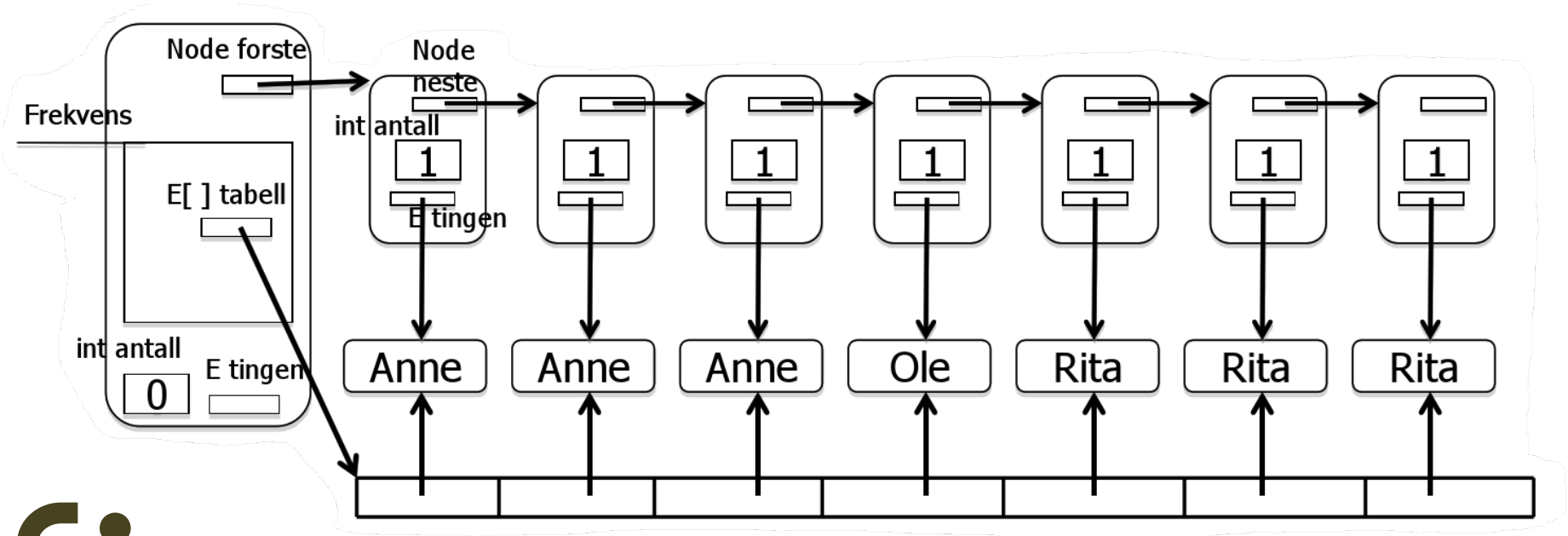
La kompilatoren sørge for automatisk innpakking av de primitive typene.

Løsningen

```
class Frekvens <E> {
    E tingen = null;  // tingen det er flest forekomster av
    int antall = 0;   // tilsvarende antall

    Node forste = null;

    class Node {
        Node neste = null;
        E tingen;
        int antall = 1;

        Node(E s) { tingen = s; }
    }
```

Nå kan vi programmere

```
Frekvens (E[] tabell) {
    Node siste = null;

    for (E s: tabell) {
        if (forste == null) {
            forste = new Node(s);  siste = forste;
        } else {
            siste.neste = new Node(s);  siste = siste.neste;
        }
    }
}
```



I oppgave 2 skal beholderen bare inneholde tre offentlige metoder: én metode kalt likeSammen, én kalt hentFlestObjektet og én kalt hentFlestAntallet.

Metoden likeSammen skal ikke ha noen parametre og ikke returnere noe. Metoden skal gå gjennom listen, slå sammen like objekter og holde orden på hvor mange like objekter det var i den opprinnelige listen. Siden listen er ferdig sortert vil like objekter ligge etter hverandre i listen. Resultatet vil være en kortere (sortert) liste der ingen objekter er like. Samtidig som metoden går gjennom listen og slår sammen objekter, skal den også finne ut (til bruk i de andre to metodene) hvilket objekt som har flest like forekomster, og hvor mange dette er. Resultatet i vårt eksempel ville vært (som allerede vist over): Anne 3, Ole 1, Rita 3, og det er Anne (eller Rita) som forekommer flest ganger, og dette er 3 ganger.

Metoden hentFlestObjektet skal returnere en peker til det objektet som representerer flest like forekomster, og metoden hentFlestAntallet skal returnere dette største antallet av like forekomster. Hvis det er flere forskjellige objekter som har flest like forekomster spiller det ingen rolle hvilket av disse tre objektene som velges.

Programmer de tre metodene likeSammen, hentFlestObjektet og hentFlestAntallet i klassen Frekvens.

Metoden 'likeSammen'

```
public void likeSammen() {
    Node denne = forste;
    if (forste != null) {
        while (denne.neste != null) {
            if (denne.tingen.equals(denne.neste.tingen)) {
                // Fjern, dvs hopp over neste:
                denne.neste = denne.neste.neste;
                denne.antall++;
                sjekkOmBest(denne.tingen, denne.antall);
            } else {
                denne = denne.neste;
            }
        }
    }
}

private void sjekkOmBest(E s, int t) {
    if (t > antall) { tingen = s;  antall = t; }
}
```

Synlighet i klasser

Operasjonene i en klasse kan være av tre varianter:

- **Hemmelige** (angitt med **private**) er ukjent utenfor klassen.
- **Lokale** (uten angivelse) er ukjent utenfor pakken.
- **Beskyttede** (angitt med **protected**) er synlig i pakken og i klassens subklasser.
- **Offentlige** (angitt med **public**) er synlige overalt.

Resten er trivielt:

```
public E hentFlestObjektet() { return tingen; }
public int hentFlestAntallet() { return antall; }
```