NTNU

Kunnskap for en bedre verden

IDAT1001 Programmering 1

Klasser som byggeklosser

Mål for dagen:

Objekter og klasser

Sammenhengen mellom klasse og bruken av klassen

Begreper som Minneadministrasjon, Byggeklossprinsippet, Sikre klasser, Testprogram, Innkapsling

Se på konkrete programeksempler

Bjørn Klefstad

Oppmøteregistrering for smittesporing

- Scan QR-kode med egen mobil
- Logg på med FEIDE-bruker
- Registrer tidsrom og bordnummer
- https://innsida.ntnu.no/checkin/room/38377

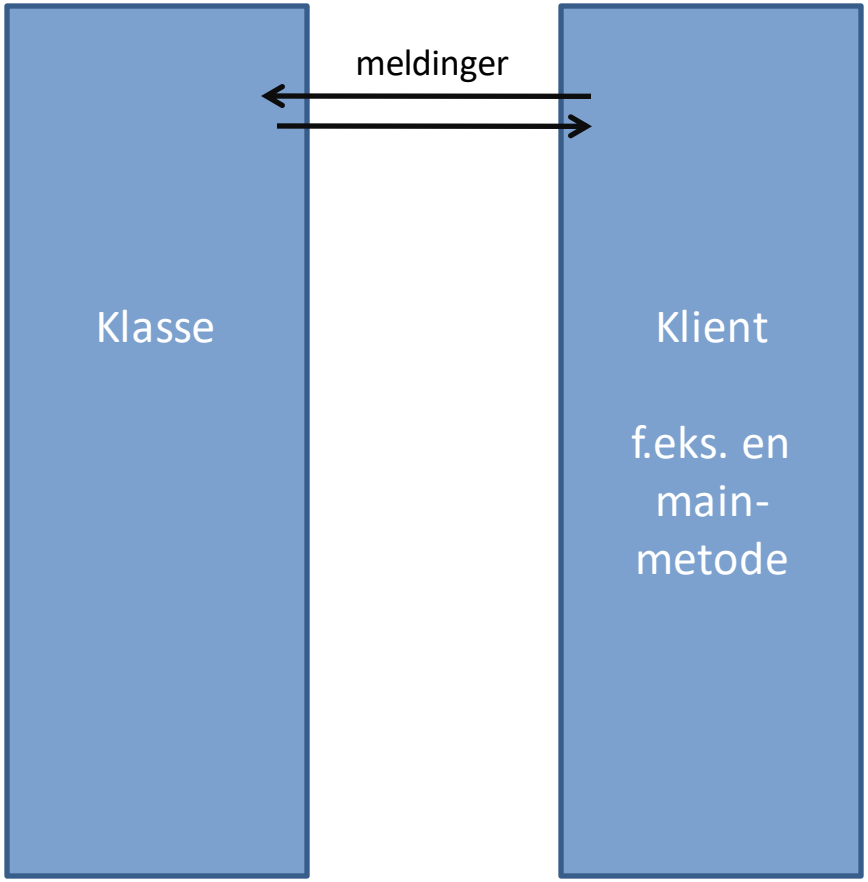
Repetisjon

Repetisjon

- Les tekstlig beskrivelse av oppgaven
- Gjennomfør en objektorientert analyse

- Sett opp et enkelt klassediagram
- Sett opp et utvidet klassediagram
- Programmer klassen i Java-kode

- Sett opp en algoritme for et klientprogram
  (UML-diagram eller pseudokode)
- Programmer klientprogram



Repetisjon

En tekstlig beskrivelse av klassen Account

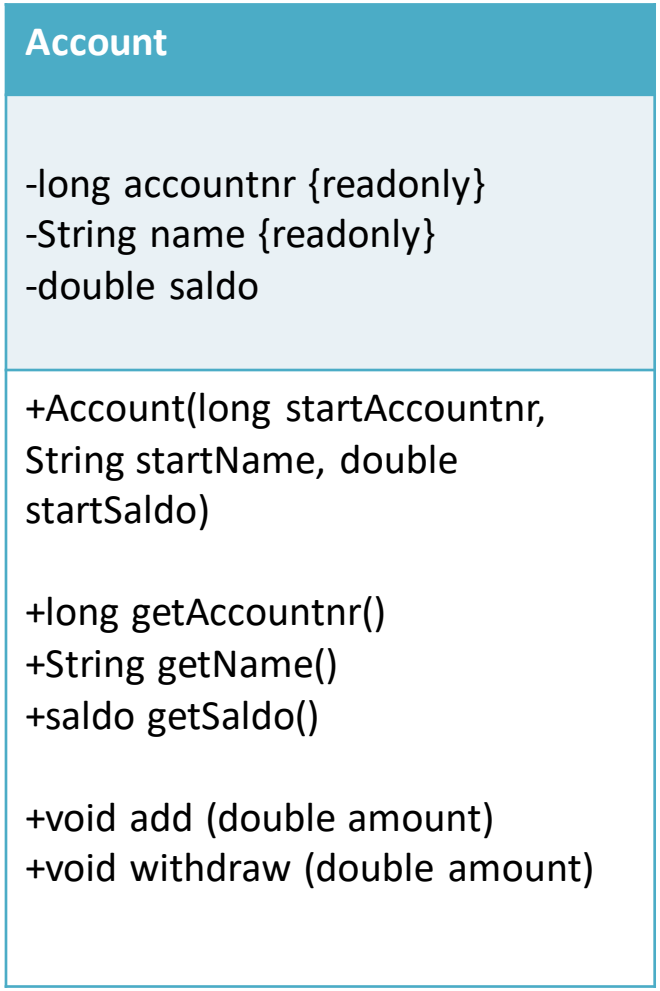
Du skal nå programmere en klasse Account der de ulike objektene skal inneholde opplysninger om accountnr, name og saldo.

I tillegg skal klassen inneholde en konstruktør som gir verdier til alle objektvariablene i klassen når den kalles.

Du skal også programmere metoder som henter ut verdien til alle objektvariablene. Videre skal klassen inneholde metoder for å sette inn penger på konto og å ta ut penger fra konto.

Klientprogrammet til klassen Account skal opprette et objekt av klassen. Deretter skal programmet gå i løkke og la brukeren velge mellom å avslutte programmet, sette inn penger, ta ut penger eller hente ut saldoen.

Klassediagram – overgang til Java-kode




```
class Account {
  private final long accountnr;
  private final String name;
  private double saldo;

public Account(long startAccountnr, String startName,
        double startSaldo) {
    accountnr = startAccountnr;
    name = startName;
    saldo = startSaldo;
  }

public long getAccountnr() {
    return accountnr;
  }
  public String getName() {
    return name;
  }
 public double getSaldo() {
    return saldo;
  }

public void add(double amount) {
    saldo += amount;
  }
public void withdraw(double amount) {
    saldo -= amount;
  }

}
```

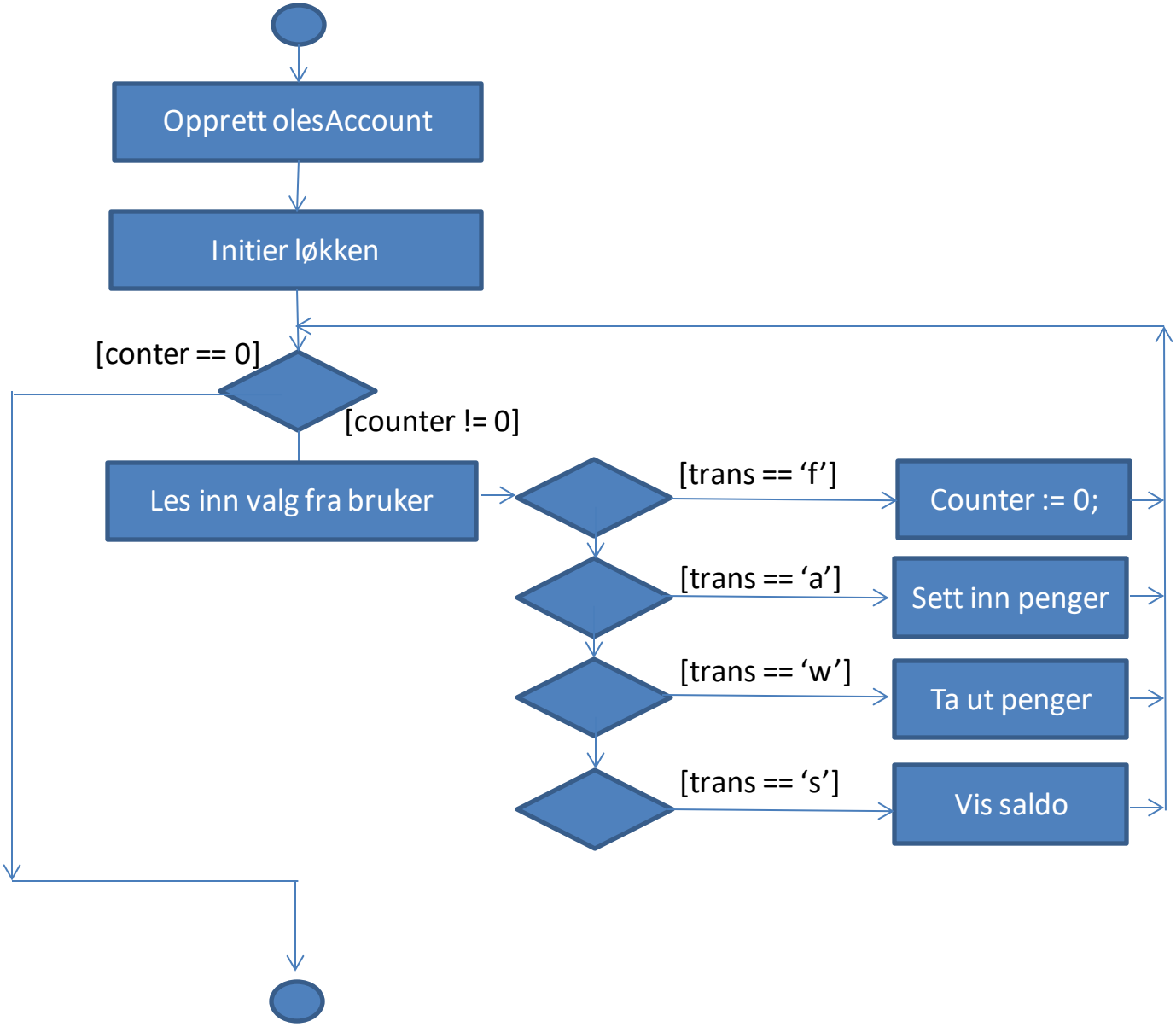
objektvariabler

konstruktør

Tilgangsmetoder

Mutasjonsmetode

Repetisjon


Opprett olesAccount
Initier løkken
[conter == 0]
[counter != 0]
Les inn valg fra bruker
[trans == 'f']
Counter := 0;
[trans == 'a']
Sett inn penger
[trans == 'w']
Ta ut penger
[trans == 's']
Vis saldo

Repetisjon

```
import static javax.swing.JOptionPane.*;
class AccountTestLokke {
   public static void main(String[] args) {
    /* Oppretter et objekt av klassen Account. */
    Account olesAccount = new Account(123456676756L, "Ole Olsen", 2300.50);
    int counter = 1;
    while (counter !=0){ // avslutter dersom bruker skriver inn 0
       String readTransaktion = showInputDialog("Velg transaksjon (avslutt, innskudd, uttak, saldo: ");
       char transaktion = readTransaktion.charAt(0);
       switch (transaktion){
          case 'a': counter =0; break;
          case 'i': String readDeposit = showInputDialog("Hvor mye skal du sette inn på konto: ");
             int deposit = Integer.parseInt(readDeposit);
             olesAccount.doTransaktion(deposit);
             if (olesAccount.getSaldo() < 0) showMessageDialog(null, "Etter innskudd har kontoen
              allikevel negativ saldo " + olesAccount.getSaldo());
             else showMessageDialog(null, "Etter innskudd er saldoen lik " + olesAccount.getSaldo());
             break;
          case 'u': String readWithdraw = showInputDialog("Hvor mye skal du ta ut fra konto: ");
             int withdraw = -Integer.parseInt(readWithdraw);
             olesAccount.doTransaktion(withdraw);
             if (olesAccount.getSaldo() < 0) showMessageDialog(null, "Etter uttak har kontoen negativ saldo " +
             olesAccount.getSaldo());
             else showMessageDialog(null, "Etter uttak er saldoen lik " + olesAccount.getSaldo());
             break;
         case 's': showMessageDialog(null, "Din saldo er " + olesAccount.getSaldo()); break;
      } // end switch
    } // end while
  } // end main
} // end class
```

Agenda

- Informasjon om referansegruppe og kullkoordinatorer
- Repetisjon fra forrige uke
  - tekstlig beskrivelse, objektorientert analyse, klassediagram, programmer klassen, aktivitetsdiagram, programmer klientprogram
- Læringsutbytter kapittel 4 Klasser som byggeklosser
- Klasser – praktiske programmeringseksempel (Employe og Grocery)
- Minneadministrasjon, Byggeklossprinsippet, Abstrahere GUI, Sikre klasser, Testprogram, Innkapsling
- Gjennomgang Programutviklingsprosess på ING labben i grupper
- Gjennomfør flervalgsoppgaver

Læringsutbytter, forelesning 5

- Identifisere hva som er brukerkommunikasjon og hva som er logikken i et program
- Anvende unntakshåndtering når konstruktør og metoder har feil i argumentene
- Sette opp testdata og lage testprogram for en klasse som sjekker at alle metoder gjør det som forventes av de.
- Forklare hensiktene med innkapsling av data
- Deklarere og anvende klassekonstanter
- Forklare hvordan minnet allokeres til data og logisk deles i områdene heapen og stakken
- Beskrive en klasse i et utvidet klassediagram(UML)

Vi skal nå jobbe med klassen
Grocery
med klientprogrammet
Pricecalculation.java

Oppgave 1 og 2

1. Utvid klassen **Groceries** med metoder for å beregne prisen på et bestemt antall kilo (parameter) med og uten moms.

Metodene skal lages slik at kunden får 10% rabatt hvis han kjøper mer enn 3 kilo, og 20 % rabatt hvis han kjøper mer enn 5 kilo på en gang.

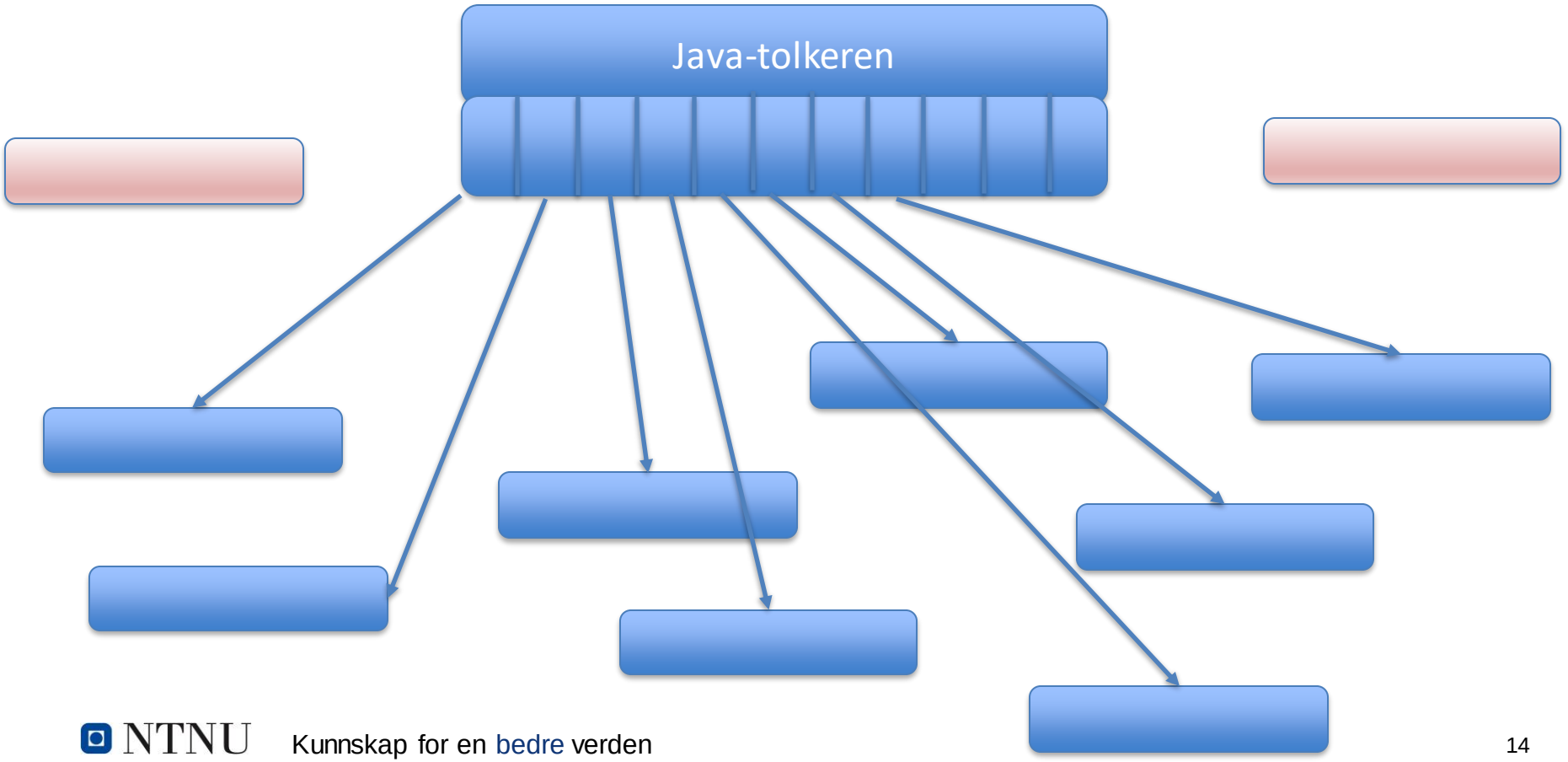
2. Endre klassen **Pricecalculation** slik at du får prøvd de nye metodene.

Minneadministrasjon

- Ved kjøring av et program ligger både programkode og data i minnet
- Programkontrollen holder adressen til neste instruks som skal utføres
- Dataområdet deles logisk i to områder en *heap* og en *stakk*
- *Heap*
  - *Plass til objekter allokeres med new og de lagres i et område vi kan betrakte som en heap (haug).*
- *Stakk*
  - *Referanser og variabler av primitive datatyper inkludert parametre og returverdier fra metoder lagres i en stakk (stabel)*

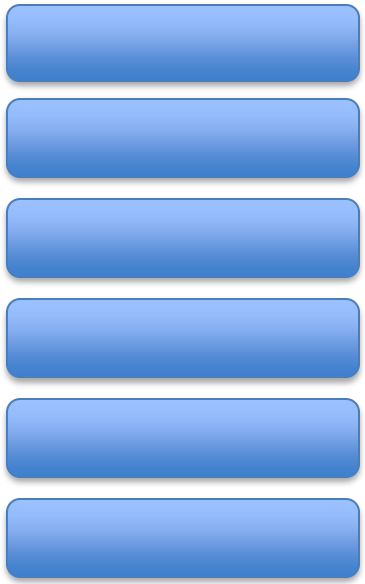
Heap

- *Plass til objekter allokeres med new og de lagres i et område vi kan betrakte som en heap (haug).*
- *Java tolkeren holder orden på den logiske strukturen*
- *Automatisk datasanering ved Java tolkeren*



Stakk

- *Referanser og variabler av primitive datatyper inkludert parametre og returverdier fra metoder lagres i en stakk (stabel)*
- *Vi legger til og fjerner elementer på toppen*
- *Stakken vokser og avtar etter hvert som programkontrollen går inn og ut av metoder/ blokker i programmet*



Byggeklossprinsippet i programmering

- Hvorfor lage flere klasser?

- Vi deler et komplekst problem opp i mindre deler og lager klasser for hver av dem
- En klasse er en logisk måte å **dele opp** et problem på
- En klasse beskriver **objekter**

- Vi kan **fokusere** på en klasse i gangen og lage denne mest mulig **uavhengig** av resten av totalsystemet
- Klasser kan testes og forandres uavhengig av hverandre
- Klasser kan **gjenbrukes** i andre sammenhenger

- I starten vil vi dele programsystemet i to klasser:
  - En klasse for å løse det aktuelle problemet
  - En klasse for å kommunisere med brukeren (**Brukergrensesnitt**)

Byggeklossprinsippet i programmering

- Hvordan lage metodene i en klasse?

- Løser metoden min kun en oppgave?
- Bør jeg bruke metoder på flere nivåer?

- Bruker jeg anonyme konstanter i metoden som burde vært erstattet av parametre?
- Er navnet på metoden og parameterne så generelle som innholdet i metoden tilsier?

- Gjenbruk!!!!!!

Å lage sikre klasser

- Klasser bør stå på egen ben og være så robuste som mulig
- Konstruktører og metoder bør melde fra til klienten om feil
- Skiller mellom to tilfeller

- 1. Metoder som returnerer en verdi
  - Forventer returverdi et positivt tall. Returner et negativt tall ved feil
  - Returtype boolean sann visst OK og usann hvis feil
  - Forventer en referanse. Returner null ved feil

- 2. Konstruktør og set-metoder som gir verdi til objektvariabler
  - Metoden i klassen må kaste et unntaksobjekt
    throw new IllegalArgumentException ("Tekststreng");
  - Unntaksobjekter må behandles i klienten for å unngå programstopp

Klassen Employee, feilrapportering

```
public Employee(int employeenr, String name, double salary) {
    if (employeenr < 1000 || employeenr > 9999) {
        throw new IllegalArgumentException(”Employee-number must have four
        digits”);
    }
    this.employeenr = employeenr;
    this.name = name;
    this.salary = salary;
}

public void setTax(double newTax) {
    if (newTax < 0.0 || newTax > MAKS_TAX_PROS){
        throw new IllegalArgumentException("Unvalid tax");
    }
    tax = newTax;
  }
```

Oppgave

Legg inn følgende datakontroll i klassen Grocery3

- Grocerynumber skal være et femsifret tall
- Price skal være minst kr 0,50.
- Antall kilo skal være mellom 0,1 og 10.

Prøv ut at datakontrollen virker, ved å gjøre nødvendige endringer i klientprogrammet Pricecalculation3.

Testprogram for en klasse

- Et testprogram prøver systematisk ut de enkelte metoder
- Tester metodene i klassen slik at vi vet de fungerer som ønsket
- Sett opp testdatasett

- Se programeksempel for klassen Account.java

Innkapsling

- Innkapsling betyr at:
  - Dataene er gjemt inne i objektene
  - Hvordan en oppgave blir løst er gjemt inne i objektene

- Innkapslede data er merket med private
- Metoder merket med public er offentlig tilgjengelige

- Klassevariabler bør være private for å gjøre det enklere å gjøre endringer

- Se programkoden for Account2.java