Oslo, januar 2025

Objekter og klasser i Java

Av Stein Gjessing Versjon 2. januar 2025

Hovedhensikten med dette notatet er den samme som i hele IN1010: At du skal lære å utvikle og skrive Javaprogrammer med god struktur, som løser problemer med gode algoritmer og fornuftige datastrukturer. Objektorientering står sentralt her. Objektorientering kan brukes både til å strukturere hele eller deler av et program og det kan brukes til å modellere og programmere den delen av verden (problemets domene) programmet skal behandle.

Objektorientering løser ikke alle problemer og brukes ikke til all struktur i et program. Men en god programmerer må kjenne til og kunne bruke objektorientert programmering.

En fordel med objektorientering er at tankemåten (metoden) både kan brukes under utvikling av programmer og under implementasjon/programmering. I IN1010 prøver vi å behandle begge deler, men det bli allikevel klart mest vekt på objektorientert programmering, og det er det som er temaet i dette notatet.

Dette notatet legger vekt på detaljer i objektorientering og i programmeringsspråket Java, men det er enda viktigere og vanskeligere å lære seg å sette sammen disse elementene til et større program som løser et problemer som kanskje bare er skissemessig beskrevet. Så etter at du har lært deg detaljene som beskrives i dette notatet må du bruke masse tid på å trene på å løse større problemer.

Når du skal løse et problem ved hjelp av en datamaskin kan du skrive et Javaprogram (som implementerer en algoritme). Dette Javaprogrammet vil opprette og forandre på en datastruktur, og rett før programmet terminerer vil vanligvis datastrukturen reflektere løsningen på problemet.

Når du utvikler en algoritme vil du til å begynne med bare ha en skisse av handlingene og en skisse av hvordan datastrukturen vil se ut. Etter hvert vil denne skissen bli mer og mer detaljert. Når du etter hvert implementerer detaljene i Java må du kunne språket og du må skjønne nøyaktig hvordan datastrukturene i programmet opprettes og manipuleres i Java.

Når et Javaprogram utføres av en datamaskin vil datastrukturene være 0-er og 1-ere inne i maskinens primærlager (Litt mer om dette i notatet «Mer om objekter og klasser i Java».). Men å tenke på 0-er og 1-ere blir alt for detaljert. I stedet tenker vi på og illustrerer disse datastrukturene ved hjelp av tegninger.

I objektorientert programmering kan illustrasjoner også brukes til å forklare hvordan objekter ufører handlinger; det er lett å tenke seg at handlingene blir utført inne i et objekt du ser på en tegning. Når du starter med en skisse til en løsning vil tegningene dine av objektene først

være ganske skissemessige. Jo nærmere du kommer et ferdig Java-program, jo riktigere og nøyaktigere vil også ditt bilde av hvordan objektene ser ut være.

I tillegg til at illustrasjonene i dette notatet skal lære deg å programmere Javas, kan de også brukes som en basis for forståelse og kommunikasjon om datastrukturer og programmer. Når du skal lage en algoritme som skal manipulere en kompleks datastruktur kan det være lurt å tegne den. Du kan tegne og se på den bare for å skjønne den selv, og du kan tegne den for å diskutere den med andre.

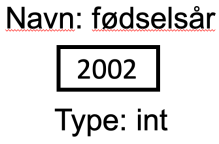
Et siste poeng om tegning av datastrukturer er at du i et kurs som IN1010 skal vise at du har nådd kursets mål. For å vise dette vil du bli bedt om å lage programmer, både som obligatoriske oppgaver og til eksamen. I tillegg vil du bli bedt om å tegne datastrukturer for å vise at du skjønner hva en utførelse av et program produserer av objekter.

Det finnes ikke noen fasit på hvordan man skal tegne datastrukturer. En tegning er god hvis den formidler riktig informasjon. Dette notatet viser hvordan vi tenker på datastrukturer og tegner dem i IN1010. På nettet vil du sikkert finne andre måter å tegne datastrukturer på, men de vil neppe være veldig forskjellige fra slik vi gjør i IN1010. I den grad det er mulig å gi noen generelle anbefalinger vil du finne dem i appendikset til slutten i dette notatet.

Det finnes et notat til: «Mer om objekter og klasser i Java». Dette andre notatet inneholder mange flere detaljer, og selv om de to notateten kan leses uavhengig, er det nok lurt at du, som nybegynner i Java, leser «Objekter og klasser i Java» først.

Del I. Variabler og veldig enkle objekter

En *variabel* i Java er en eller flere byte (á 8 bit) i minnet (primærlageret/RAM). Disse bytene har en adresse i minnet, men i Java omtaler vi dem bare med *navn*. Bytene inneholder det bit-mønster som er variabelens *verdi*. I Java må en variabel også ha en *type* som ikke kan forandres (f.eks. int, double, boolean). Anta at vi har deklarasjonen `int fødselsår = 2002;` Denne variabelen kan vi tenke på eller tegne slik:



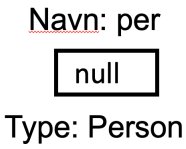
Mange Java-implementasjoner liker ikke norske bokstaver, så vanligvis bør du ikke gjøre slik som her, men unngå norske bokstaver.

Typen er veldig ofte opplagt og da tegner vi bare navnet og verdien hvis den er viktig, f.eks. slik:

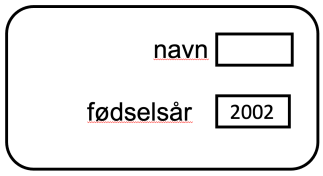


En konstant i Java er en variabel som får en initialverdi men som etter dette ikke kan forandres. I Java deklareres en konstant med modifikatoren `final`. I dette notatet (og de fleste andre steder i IN1010) omtaler vi vanligvis både variabler og konstanter som variabler.

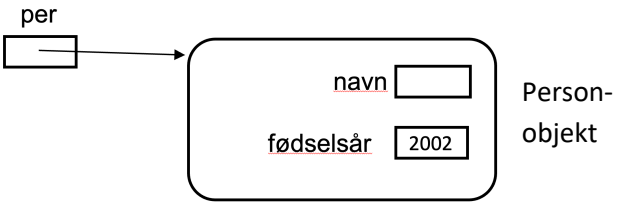
En referansevariabel (en variable som inneholder en referanseverdi, også kalt en peker) inneholder adressen til et objekt. Typen til variabelen er da klassenavnet til de objektene verdien i variabelen kan peke på eller referere. Vi sier også gjerne at variabelen referer dette objektet, selv om vi egentlig mener at det er variabelens innhold eller verdi som referer (peker på) objektet. Innholdet i en referansevariabel kan også være null som betyr at innholdet ikke peker på noe objekt. Hvis vi har deklarert en klasse Person og så en variabel per slik: Person per = null; får vi en variabel vi kan tenke på og tegne slik:



Et objekt er en samling av data og metoder. For å markere denne samlingen tegner vi i IN1010 disse inne i et rektangel med avrundede hjørner. Vi skal se at vi ofte ikke tegner metodene, bare variablene inne i objektene, f.eks. slik:



Hvis vi sier per = new Person(); opprettes det et objekt og innholdet i variabelen per settes til å referere eller peke på dette objektet. Dette kan vi tegne slik:



Det er ofte lurt å gjøre det helt klart hva slags objekt det er snakk om, derfor har vi skrevet navnet på klassen objektet er laget av i figuren over. Variablene inne i et objekt kalles *instansvariabler.*

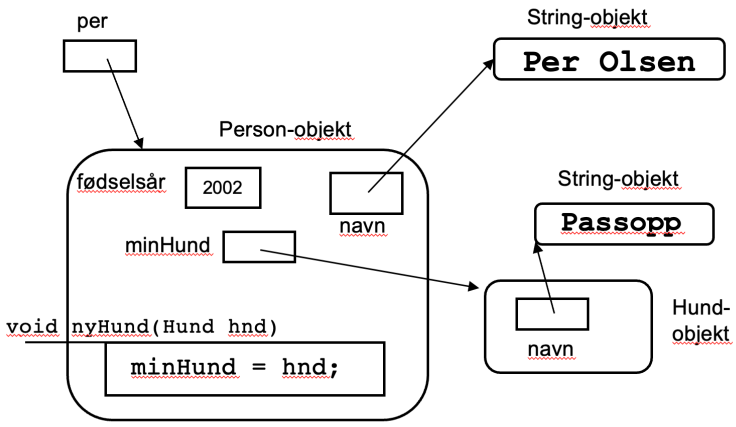
La oss nå lage et litt større og mer konkret eksempel ved å se på to klasser pluss en hovedklasse som bruker de to første klassene:

```
class Hund {
     private String navn;
     Hund (String navn) {
          this.navn = navn;
     }
}

class Person {
     private int fødselsår;
     private String navn;
     Person (String nvn, int år) {
          fødselsår = år;
          navn = nvn;
     }
     private Hund minHund;
     public void nyHund(Hund hnd) {
          minHund = hnd;
     }
}

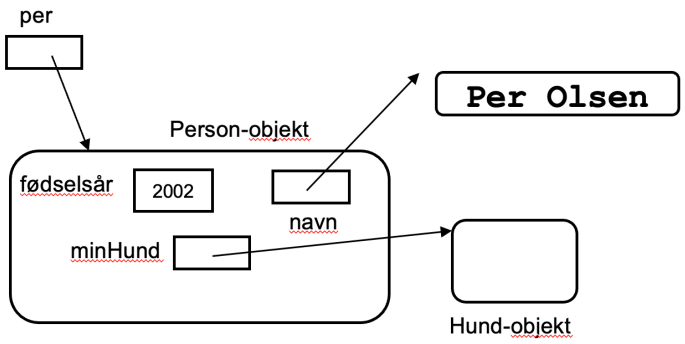
class HundOgPerson {
     public static void main (String[] args) {
          Person per = new Person("Per Olsen", 2002);
          Hund passopp = new Hund("Passopp");
          per.nyHund(passopp);
          // Tegningen viser datastrukturen slik den er nå
     }
}
```

Denne datastrukturen ser slik ut på slutten av main-metoden:



Vi ser lett fra klassedeklarasjonen at Person-objektet inneholder en metode nyHund, og vanligvis vil vi tegne Person-objektet uten denne metoden. Her har vi tegnet inn metoden i objektet for å gjøre det helt klart at når metoden utføres gjøres dette inne i objektet slik at metoden har tilgang til instansvariablene i objektet. Metoden nyHund er en viktig egenskap ved dette objektet, og også for å understreke dette har vi tegnet inn metoden i objektet denne gangen.

Hvis vi fokuserer på personen Per, er det ikke så viktig hvordan Hund-objektet ser ut inni, så vi kan jo lage tegningen av hunden litt enklere, se nedenfor. Tegninger som dette vil du bli bedt om å lage i IN1010.



Del II. HashMap, array og ArrayList

Dette kapitelet omhandler bl.a. to verktøy fra Javas bibliotek, HashMap og ArrayList. Det er ikke noe du behøver å kunne den første uken av IN1010, så ved første gjennomlesning er det mulig å hoppe over dette kapitlet. Men husk at du må lære deg disse to klassene så snart som mulig. Det står noe mer om disse klassene i det andre notatet: Mer om objekter og klasser i Java. I Python bruker man en ordbok (dictionary) når man skal lagre nøkkel-verdi-par. I Java finnes det også ordbøker og en av disse er klassen *HashMap*.

I Python lagres mange elementer eller verdier i lister. I Java har vi muligheten til å lagre mange verdier i en *array*. En array er en tett samling av variabler i minnet. Alle variablene i en array må være av samme type. Array er ikke en klasse, men en iboende konstruksjon i Java og i minnet i en datamaskin. Arrayer har ingen ting med objektorientering å gjøre. I minnet består en array av mange lagerceller (variabler) som ligger rett etter hverandre. I Java er en array indeksert fra 0 til lengden av arrayen minus en. Å skrive og lese fra en array er meget enkelt og effektivt hvis du kjenner indeksen. Hvis programmet derimot må flytte på verdier for å få plass til nye verdier på bestemte plasser (indekser) er det mer komplisert og tungvint å bruke en array. Et viktig læringsmålet i IN1010 er å kunne programmere med arrayer.

I Java kan vi også lagre mange verdier av samme type i mer fleksible lister. En klasse som gjør dette er *ArrayList*. Her kan vi f.eks. sette nye verdier inne i mellom andre på en enkel måte (men bak kulissene blir det utført mer komplekse programmer som kan ta litt tid).

En annen viktig forskjell på en array og en ArrayList er at en array kan inneholde variabler av primitive typer eller referanse-typer mens en ArrayList bare kan inneholde referansevariabler. Men husk at alle variabler i en bestemt array eller ArrayList er av samme type.

Slå opp i Javas bibliotek på HashMap og ArrayList og se hvilke tjenester disse klassene tilbyr. Lær deg også hvordan du bruker arrayer i Java.

For å lage et meget kort eksempel som illustrere alle disse tre datastrukturen skal vi utvide klassen Person. Vi skal legge til en ordbok der personer kan lagre vennene sine. Vi bruker navnet på vennen som nøkkel. Så skal vi anta at en person kan ha flere hunder, ikke bare én. Til å lagre alle hundene en person eier skal vi bruke en ArrayList.

Eksemplet på bruk av array er litt kunstig: Vi skal anta at alle personer kan ha maksimalt fire barn, nummerert fra barn 0 tom. barn nummer 3.

Verktøyene for ordbøker og lister ligger i et eget bibliotek, kalt java.util, så disse må importeres før de kan brukes.

Her er det nye programmet:

```
import java.util.HashMap;
import java.util.ArrayList;

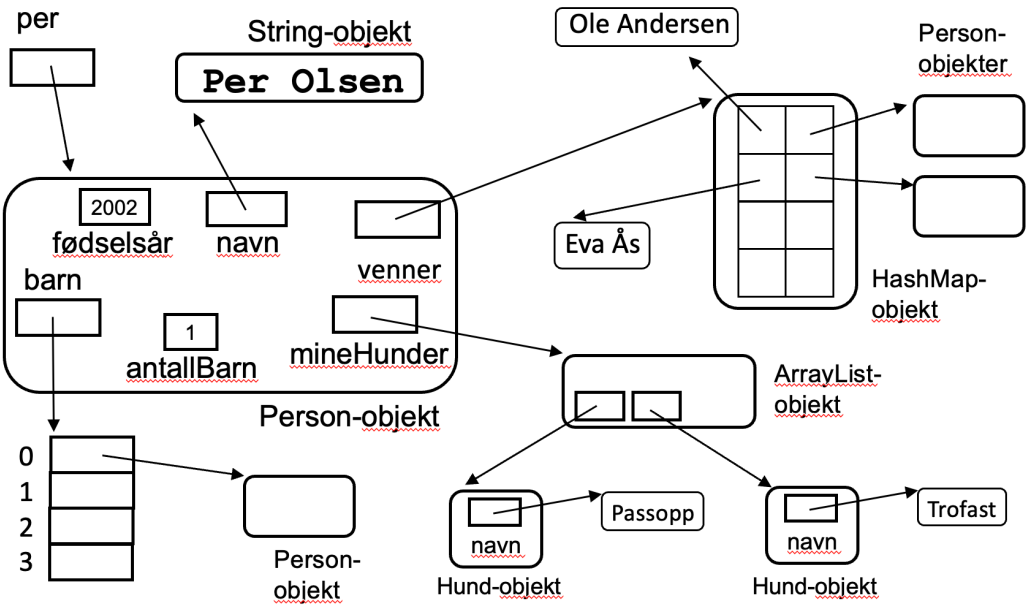
class Person {
     private int fødselsår;
     private String navn;
     Person (String nvn, int år) {
          fødselsår = år;
          navn = nvn;
     }
```

```
    private ArrayList<Hund> mineHunder = new ArrayList<>();
    private HashMap<String,Person> venner = new HashMap<>();
    private Person[] barn = new Person[4];
    private int antallBarn = 0;
    public void nyHund(Hund hnd) {
          mineHunder.add(hnd);
    }
    public void nyVenn(String navn, Person pers) {
          venner.put(navn,pers);
    }
    public void nyttBarn(Person nyttB) {
          barn[antallBarn] = nyttB;
          antallBarn ++;
    }
}

class Hund {
    private String navn;
    Hund (String navn) {
         this.navn = navn;
    }
}

class PersonerMM {
    public static void main (String[] args) {
         Person per = new Person("Per Olsen", 2002);
         Hund passopp = new Hund("Passopp");
         per.nyHund(passopp);
         Hund fast = new Hund("Trofast");
         per.nyHund(fast);
         Person ole = new Person("Ole Andersen", 2004);
         per.nyVenn("Ole Andersen", ole);
         Person eva = new Person("Eva Ås", 2003);
         per.nyVenn("Eva Ås", eva);
         Person anne = new Person("Anne Persdottir", 2);
         per.nyttBarn(anne);
         // Tegningen viser datastrukturen slik den er nå
    }
}
```

Slik ser noe av datastrukturen ut på slutten av en utførelse av programmet over:



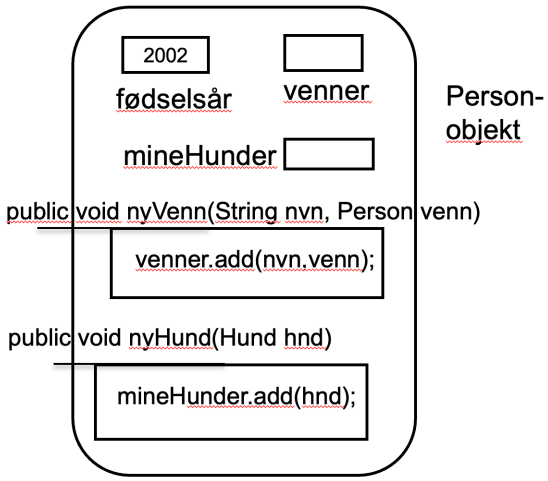
Legg merke til at ikke alle objekter er tegnet like nøyaktig. Vi tegner datastrukturen slik som over hvis hovedhensikten er å skjønne hvordan Person-objektet som refereres av variabelen `per` ser ut. Det er jo mange Person-objekter på tegningen, men bare ett er tegnet ganske nøyaktig.

Det er viktig for deg som tar IN1010 å skjønne at main-metoden i programmet over produserer denne datastrukturen. Mer om main-metoden lenger ned i del IV.

Legg merke til referansevariabelen kalt `per` helt øverst til venstre i figuren over. Dette er en lokal variable i main-metoden, og hvis vi skulle vært helt nøyaktig skulle vi tegnet metodeinstansen av main-metoden. I denne metodeinstansen er det jo også en del andre lokale variabler: `passopp`, `fast`, `ole`, `eva` og `anne`. Vi kommer tilbake til dette senere i dette notatet.

Del III. Metoder i objekter.

Når vi utfører setningene per.nyHund(. .) og per.nyVenn(. . .) utføres metodene nyHund og nyVenn inne i objektet som er referert av innholdet i variabelen `per`:



Når vi leser et program vil klassedeklarasjonen vise oss hvilke metoder som er inne i et objekt (alle metoder som ikke er deklarert med modifikatoren static). Derfor er det så opplagt hvilke metoder som er inne i et objekt at metodene vanligvis ikke tegnes. I de tilfellene der det er nyttig å tegne metoder inne i objekter er det ofte snakk om *rekursjon* (et tema som blir behandlet senere i IN1010). Lærerne i IN1010 tegner også noen ganger metoder og metodeinstanser som illustrasjon på programutførelser.

Uansett om vi tegner det eller ikke må vi **alltid tenke oss** at en metode (som ikke er static) utføres inne i et objekt. Disse metodene definerer de handlingene objektet kan utføre slik at når vi programmerer dem definerer vi hva slags liv objektet skal ha.

Vi vet at når en metode kalles opprettes det en metodeinstans. Denne metodeinstansen inneholder de lokale variablene inne i metoden, dvs. alle parametrene og eventuelle variabler som deklareres inne i metoden. Inne i en metode er alle disse lokale variablene pluss (instans)variablene i det omsluttende objektet tilgjengelig for programmet. Dette kalles programmets skop (engelsk scope). Mer om alt dette i «Mer om objekter og klasser i Java».

Legg merke til at vi skrev at vi må alltid "**tenke oss"** at en metode utføres inne i et objekt. Men i virkeligheten vil ikke Javas kjøretidsystem duplisere koden for metodene inn i hvert objekt. Det som dupliseres i hvert objekt er bare instansvariablene. Bak kulissene vil en metode som utføres ha en referanse til objektet den utføres inne i. Dette er omtrent slik programmereren selv må gjøre det i Python.

Parameteroverføring i Java er alltid det vi kaller «med verdi». Det vil si at aktuellparameteren på kallstedet beregnes og verdien av dette uttrykket blir overført til metodeinstansen. Det er denne verdien som blir startverdien til den formelle parameteren som altså er en lokal variabel i metoden.

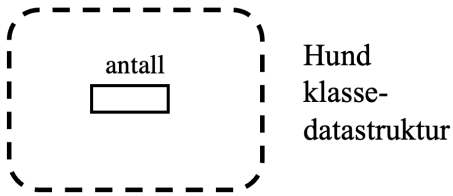
Del IV. Modifikatoren `static`

Vi har lært at alle variabler og metoder som ikke er deklarert med modifikatoren static vil oppstå som egenskaper inne i hvert eneste objekt som lages av denne klassen. Men hva med de egenskapene som er deklarert som static?

En variable som er deklarert som static i en klasse vil det bare finnes én av, uansett hvor mange objekter det lages av denne klassen. En static-variabel brukes til noe som er felles for alle objektene av klassen, f.eks. hvor mange objekter det er opprettet av denne klassen. En static-variabel er i skopet til alle objektene av denne klassen. Det settes av plass til en static-variabel i noe som kalles klassens datastruktur. For å tenke på og illustrere dette lager vi for hver klasse noe vi i IN1010 kaller en *klassedatastruktur*. Vi tegner samlingen av static-egenskaper inne i et stiplet rektangel med avrundede hjørner. La oss modifisere Hund-klassen:

```
class Hund {
     static int antall= 0;
     private String navn;
     Hund (String navn) {
          this.navn = navn;
          antall++;
     }
}
```

I det et program som bruker klassen Hund starter opp lager Javas kjøretidsystem (run time system) denne klassedatastrukturen:

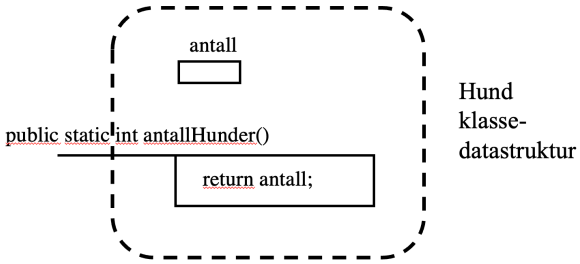


Som nevnt over er en static-variabel i skopet til alle metodene i klassen, men også andre deler av det Javaprogrammet som bruker denne klassen kan aksessere (lese og skrive) denne variabelen ved å si `Hund.antall`. Dette betyr at hvem som helst utenfor klassen også f.eks. kan si `Hund.antall++` og ødelegge hele programmet. Derfor kan vi også beskytte static-variabler med `private`, men hvis vi da ønsker å vite hvor mange Hund-objekter det er laget må vi skrive en metode som gir oss dette. Da kan klassen se slik ut:

```
class Hund {
     private static int antall= 0;
     public static int antallHunder() {
           return antall;
     }
```

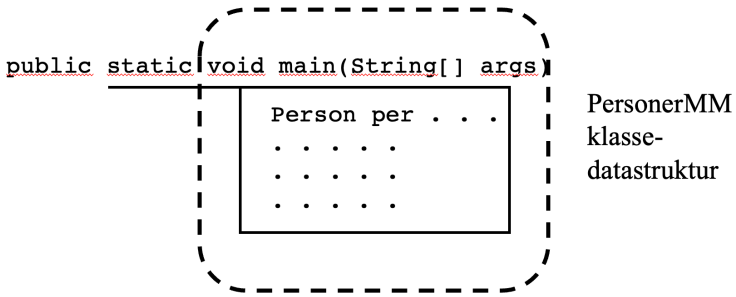
```
    private String navn;
    Hund (String navn) {
        this.navn = navn;
        antall++;
    }
}
```

Som vi ser over kan en klasse også inneholde metoder som er modifisert med `static`. Disse metodene har bare klassens static-variabler i sitt skop (pluss variabler deklarert inne i metoden, såkalte lokale variabler). Vi tenker på dette på denne måten:



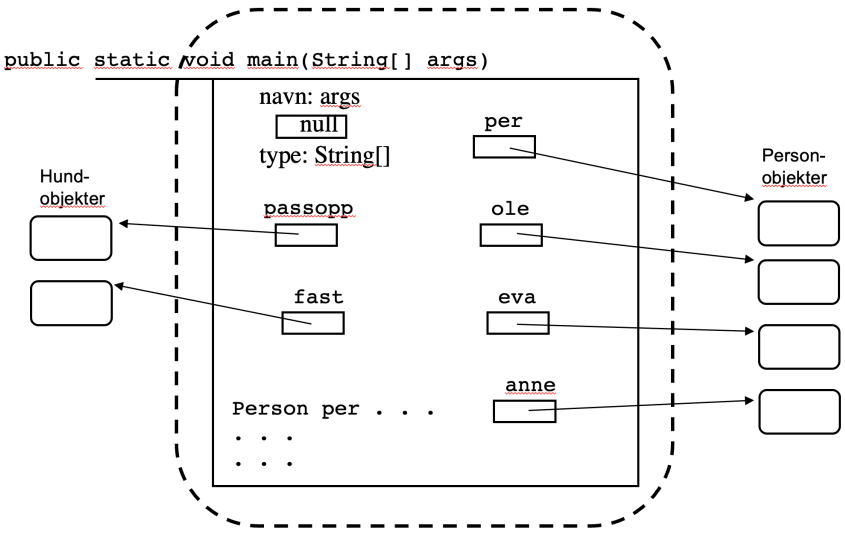
Vanligvis er klassedatastrukturer så enkle at det ikke er nødvendig å tegne dem. De er tegnet i dette notatet mest som et pedagogisk verktøy.

La oss avslutte med klassedatastrukturen til klassen PersonerMM. Denne inneholder bare én metode, og vi kan illustrere den slik:



Nå vi kjører et Javaprogram starter det alltid ved at Javas kjøretidsystem kaller main-metoden i klassen som oppgis i Java-kommandoen. Siden main-metoden er static kan vi tenke oss at kjøretidsystemet kaller denne metoden slik: PersonerMM.main(. . .);

Når en metode kalles vil både den formelle parameteren (args i vårt tilfelle) og de variablene som er deklarert inne i metoden oppstå som lokale variabler i metodeinstansen. Dette kan vi illustrere slik:



PersonerMM. klassedatastruktur

Når en metode terminerer forsvinner metodeinstansen, og med den alle de lokale variablene. Akkurat den metoden vi har tegnet her (main-metoden) er jo litt spesiell, for når den terminerer avsluttes også hele programmet (dette er sant helt til vi kommer til tråder og GUI (Graphical User Interface) senere i semesteret).

Lykke til med programmeringen din og med kommunikasjon og samarbeid med andre i programutviklingsprosessen.

Appendiks: Oversikt over notasjonen som brukes

Når en klasse brukes i et program, blir alle "static"-egenskapene gjort tilgjengelig for programmet. Vi tegner slike egenskaper inne i en *klassedatastruktur* som ser slik ut:



Klassedatastruktur for klassen <Klassenavn>

Hver gang programmet oppretter et objekt med "`new <Klassenavn>()`" blir alle egenskapene i klassen som ikke er static samlet inne i et nytt objekt:



Et <Klassenavn>-objekt

De to firkantene over med avrundede hjørner er ment å avgrense data (variabler og konstanter) og metoder inne i hhv. en klassedatastruktur og et objekt. Omrisset er bare der for å vise hva som er inne i klassedatastrukturen eller objektet.

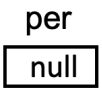
Inne i klassedatastrukturen tegner vi viktige statiske variabler, og inne i et objekt tegner vi viktige instansvariabler og eventuelt markerer vi viktige metoder.

En variabel og en konstant (deklarert med final) er et sted i primærlageret som har et navn, en type og ett innhold (en verdi). Vanligvis er typen opplagt, og da utelater vi typen fra tegningen.

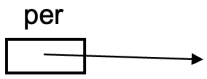
Hvis vi f.eks. har deklarasjonen `int antall = 17;` så kan vi tegne variabelen slik:



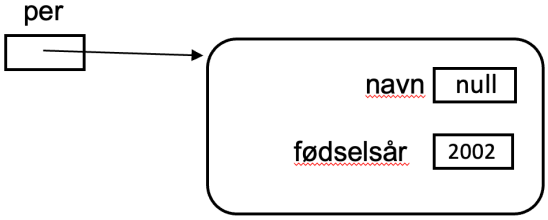
En variabel av en referansetype inneholder en verdi som er adressen til (er pekeren til) et objekt. Hvis vi har klassen Person kan vi deklarere `Person per = null;`



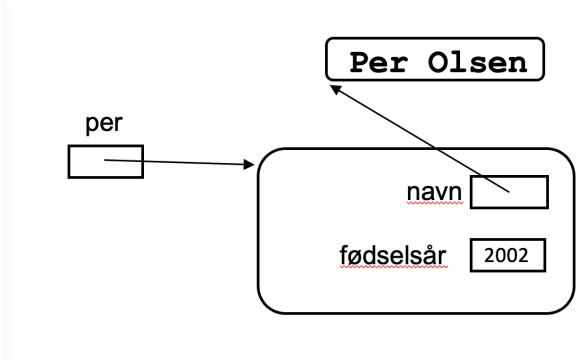
Når `per` referer eller peker på et objekt tegner vi denne variabelen slik:



Hvis klassen Person har en konstruktør som angir personens fødselsår, og vi sier:
`per = new Person(2002);` kan vi tegne slik:

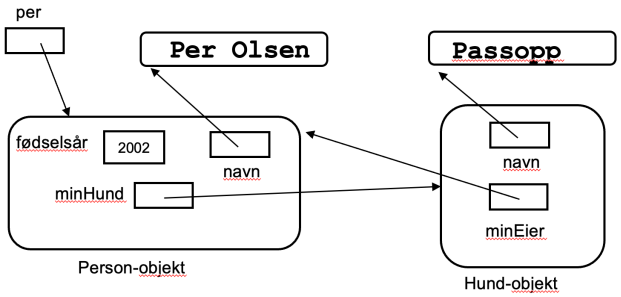


Hvis nå objektet `per` peker på får et navn vil vi f.eks. tegne det slik:

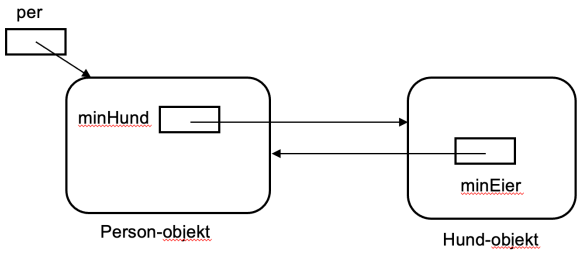


Adressen som ligger i en referansevariabel angir starten av et objekt, derfor vil vi så langt som mulig la pekeren ende opp nær toppen / starten av objektet.

Det viktigste vi tegner er vanligvis instansvariabler som er referanser og som viser sammenhenger eller relasjoner mellom objekter. Hvis en person eier en hund, og en hund har en eier, kan vi tegne dette slik:



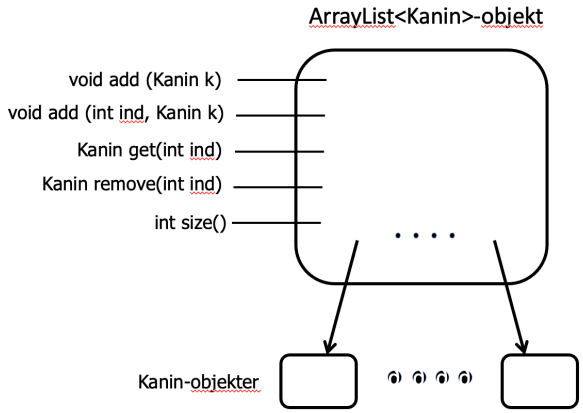
Men hvis vi bare ønsker å legge vekt på relasjonen mellom eier og hund kan vi kanskje forenkle slik:



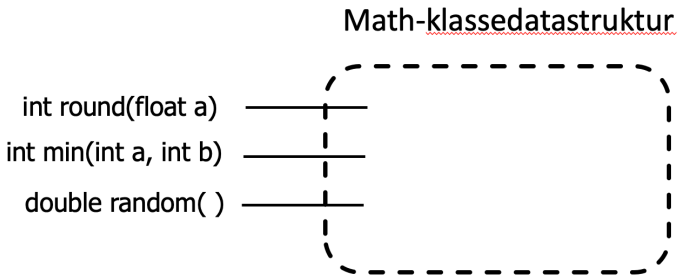
Metoder kan enten være statiske eller ikke. De siste kaller vi instansmetoder. En statisk metode utføres inne i en klassedatastruktur, mens en instansmetode utføres inne i et objekt.

Vanligvis er det opplagt ut fra klassedeklarasjonen hvilke metoder som finnes, og da er det ikke nødvendig å tegne at metodene finnes inne i objektene eller inne i klassedatastrukturen.

Hvis du noen ganger ønsker å poengtere grensesnittet eller tjenestene til et objekt eller en klassedatastruktur, kan du f.eks. gjøre det slik som det er gjort i det andre notatet «Mer om objekter og klasser i Java». Her har vi i tillegg til å beskrive tjenestene også skissert en tenkt datastruktur:



Også klassedatastrukturer kan tilby tjenester. Klassen Math i Java-biblioteket er et typisk eksempel. Da kan vi tegne:



Og til slutt husk at det ikke er noen fasit for hvordan du tegner datastrukturer og at de er gode bare om de overbringer den informasjonen de er ment å gjøre.

SLUTT.