Objekter og klasser

Java er et objekt-orientert språk. Objekter er en fundamental del av Java, samt mange av de fundamentale konseptene som følger, bl.a.:

- Polymorfisme
- Arv
- Abstraksjon
- Klasser
- Objekter
- Instanser
- Metoder

Disse blir mer relevante etter hvert, men her skal vi ta for oss to av disse: *objekter* og *klasser*.

Hva er et objekt?

Vi kan se på objekter i Java som objekter i virkeligheten rundt oss. Mennesker, hunder og biler er alle objekter med forskjellige egenskaper og oppførsel.

Hvis vi tar for oss en hund, har den flere grunnleggende egenskaper, f.eks.:

- Navn
- Rase
- Farge
- Alder

Og oppførsler, f.eks.:

- Bjeffing
- Logring
- Soving

Hvis vi skal prøve å modellere en hund i Java, vil egenskapene bli lagret som variabler og oppførsel kan bli vist som metoder. Dette blir vist i Java under.

Klasser

En klasse er en *mal* som et objekt blir opprettet fra. For eksempelet over, vil en klasse for en hund se slik ut:

```
class Hund {
    String navn;
```

```
    String rase;
    String farge;
    int alder;

    void bjeff(){

    }

    void logre() {

    }

    void sov() {

    }
}
```

I tillegg til egenskapene (klassevariabler) og oppførselen (metoder), har hver klasse en *konstruktør*. Hvis vi ikke lager en egen konstruktør, vil Java opprette en *default* konstruktør for oss. Oppgaven til en konstruktør er å opprette et objekt ut ifra klassen, og derfor vil minst én konstruktør bli kalt hver gang vi oppretter et nytt objekt. Vi oppretter en konstruktør slik:

```
public class Hund {
    public Hund() {

    }
}
```

Konstruktøren skal ha samme navn som klassen. En klasse kan ha flere konstruktører.

Opprette objekter

Som nevnt tidligere, er en klasse en *mal* for objekter. I all hovedsak blir et objekt skapt ut ifra en klasse. I Java bruker vi nøkkelordet *new* for å opprette nye objekter. Det er tre steg vi går gjennom når vi oppretter et nytt objekt:

- Deklarasjon - Vi deklarerer en variabel med objekt-typen og et navn på variabelen
- Instansiering - Vi bruker nøkkelordet *new* for å instansiere objektet.
- Initialisering - *new* blir fulgt av et kall til en av konstruktørene til klassen. Dette kallet oppretter objektet.

Følgende er et eksempel på hvordan vi kan opprette et objekt fra klassen *Hund*:

```
Hund fido = new Hund();
```

Nå har vi opprettet et objekt fra klassen *Hund* og lagret objektet i variabelen *fido*.

Bruk av variabler og metoder

Når vi laget klassen *Hund* ga vi den flere variabler og metoder. Når vi opprettet et objekt fra *Hund* vil dette objektet ha de samme variablene og metodene. Siden vi lagret objektet i variabelen *fido*, kan vi bruke denne for å få tilgang til disse. For eksempel, for å hente ut navnet kanv i skrive:

```
fido.navn;
```

Som er en *String* som inneholder navnet, eller:

```
fido.alder;
```

Som er en *int* som inneholder alderen. Vi kan også kalle på metodene vi definerte i klassen:

```
fido.bjeff();
```

Nå har vi ikke verdier i disse variablene, og metoden *bjeff* gjør for øyeblikket ingenting. For å sette variablene kan vi gjøre følgende:

```
fido.navn = "Fido";
fido.rase = "Border Collie";
fido.farge = "Brun";
fido.alder = 3;
```

Vi kan da skrive ut de forskjellige variablene til objektet:

```
System.out.println(fido.navn);
System.out.println(fido.alder);
```

Konstruktører med parametre

Det kan være tungvint å måtte sette de forskjellige variablene til et objekt individuelt. Vi kan gjøre det enklere ved å gi verdier til variablene ved å sette verdier direkte i konstruktøren til en klasse. En konstruktør kan ta inn flere parametre, og sette variablene i klassen til verdiene vi får inn fra parametrene:

```
class Hund {
    String navn;
    String rase;
    String farge;
    int alder;

    public Hund(String navn, int alder) {
        this.navn = navn;
        this.alder = alder;
    }
}
```

Her får vi inn variablene *navn* og *alder* som parametre. Vi ønsker å sette variablene i klassen til verdiene vi får inn, men siden det er samme navn på både de lokale variablene vi får i parametrene, bruker vi nøkkelordet ***this*** for å spesifisere at vi bruker variablene fra **klassen**. Vi trenger ikke bruke *this* om det ikke er en konflikt mellom navn.

Med denne konstruktøren kan vi da enklere opprette et objekt fra klassen:

```
Hund fido = new Hund("Fido", 3);
```

Med kun ett kall på konstruktøren vil *instansvariablene navn* og *alder* være satt til verdiene vi satte som argumenter når vi kallet på konstruktøren. **Obs!** Argumentene vi gir til konstruktøren må være i samme rekkefølge som vi definerte i konstruktøren.

Vi kan også definere flere konstruktører som tar inn forskjellige parametre:

```
class Hund {
    String navn;
    String rase;
    String farge;
    int alder;

    public Hund(String navn, int alder) {
        this.navn = navn;
        this.alder = alder;
    }

    public Hund(String navn, String rase, String farge, int alder) {
        this.navn = navn;
        this.rase = rase;
        this.farge = farge;
        this.alder = alder;
    }
```

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

Og vi kan bruke disse konstruktørene som vi ønsker:

```
Hund fido = new Hund("Fido");
Hund max = new Hund("Max", 7);
Hund bella = new Hund("Bella", "Labrador", "Hvit", 2);
```

Vi opprettet her flere objekter med forskjellige konstruktører. Vi kan printe ut variabelen *navn* fra hvert objekt, men i dette eksempelet er det kun *bella* som har en verdi i variablene *rase* og *farge*. Hvis vi prøver å printe `max.rase` vil vi få en feil, da variabelen er deklarert med ikke initialisert.

Gettere og settere

Når vi har en *public* eller *default* variabel i klassen, vil vi kunne endre på verdien til variabelen til hva vi vil så lenge vi har et objekt av klassen. Dette kan ofte være problematisk, da vi ofte vil kontrollere verdiene vi setter inn i variabelen, og holde variablene internt i klassen uten å eksponere disse for direkte tilgang.

For å illustrere dette, se for deg at hodet ditt er et objekt, og at tankene dine er variabler. Du vil som oftest ikke gi hvem som helst direkte tilgang til tankene dine, men om noen spør hva du tenker på, kan du svare om du ønsker.

I Java kan vi deklarere variabler med nøkkelordet *private*, som gjør variabelen tilgjengelig ***kun*** internt i klassen. Disse variablene vil da være synlig for f.eks. metoder og konstruktører i klassen. Vi kan da *ikke* bruke f.eks. `fido.navn` for å hente ut variabelen *navn* fra objektet lagret i *fido*. For å kunne hente ut en *private* variabel, kan vi lage en metode som gir oss verdien av variabelen:

```
class Hund {
    private String navn;

    public Hund(String navn) {
        this.navn = navn;
    }

    public String getNavn() {
        return this.navn;
    }
}
```

En metode som henter ut verdien til en *private* variabel, kaller vi en *get-metode*. Denne blir som oftest satt til *public* slik at vi har tilgang til metoden fra andre steder enn kun internt i klassen. Se *modifikatorer.md* for forskjeller på disse nøkkelordene. Denne metoden navngir vi ofte slik:

`getNavnPåVariabel`

og denne skal returnere samme datatype som variabelen er. I dette tilfellet er *navn* av typen *String*, og derfor må *getNavn()* returnere *String*.

Nå kan vi hente ut variabelen på denne måten:

```
fido.getNavn();
```

Hvis vi vil *endre* på verdien til en variabel, er vi nødt til å gjøre det motsatte av en *get-metode* og lage en metode som kan ta inn en verdi som parameter, og sette den private variablen til denne verdien. Vi har allerede gjort noe lignende i konstruktørene. En slik metode kaller vi en *set-metode*:

```
class Hund {
    private String navn;

    public void setNavn(String navn) {
        this.navn = navn;
    }
}
```

Denne metoden er *void* da vi ikke skal returnere noe som helst, bare sette en verdi. Typen vi tar inn i som parameter må være lik typen til variabelen, i dette tilfellet *String*. Siden navnet på parameteret er det samme som navnet på variabelen i klassen, bruker vi *this* for å differensiere disse to. Settere kan vi bruke slik:

```
Hund fido = new Hund(); // Tom konstruktør, så ingen variabler blir satt
fido.setNavn("Fido");
```

Vi kan også utvide settere til å modifisere verdiene vi gir som parametre i metoden. F.eks. hvis vi ønsker å formattere verdiene før vi lagrer de. Et eksempel kan være å forsikre oss om at en alder ikke er under 0:

```
class Person {
    private int alder;

    public void setAlder(int alder) {
        if(alder > 0)
            this.alder = alder;
    }
}
```

```
Person arne = new Person();
arne.setAlder(-10); // Vil ikke sette variabelen
arne.setAlder(10); // Setter variabelen til 10
```

Generelt sett er malen til gettere og settere slik:

```
public T getVariabel() {
    return this.variabel;
}

public void setVariabel(T variabel) {
    this.variabel = variabel;
}
```

Hvis du har reagert på at det blir brukt *this* på gettere selv om jeg skrev lengre oppe at de ikke var nødvendige, det er helt riktig! *this* er ikke nødvendig om det ikke er uklart hvilke variabler som er lokale og hvilke som er instansvariabler, men for å gjør koden tydelig har jeg inkludert bruken av *this* her også.

Metoder i klassen

Alle variabler i klassen kan bli brukt av alle metodene i klassen, og alle metodene kan endre på variabler internt i klassen, og på verdier som blir sendt inn i metoden som argumenter. Et eksempel på en metode som bruker variablene i klassen kan være:

```
class Hund {
    private String navn;
    private int alder;

    public Hund(String navn, int alder) {
        this.navn = navn;
        this.alder = alder;
    }

    public int tilMenneskeÅr() {
        return alder * 7;
    }
}
```

Noen ganger vil vi skrive ut informasjon om et objekt. Da trenger vi en metode i klassen som samler sammen alle variablene og gir oss en *String* som oppsummerer de forskjellige verdiene. En slik metode kaller vi ofte *toString*. En *toString*-metode ser ofte slik ut:

```
class Hund {
    private String navn;
    private int alder;

    public Hund(String navn, int alder) {
        this.navn = navn;
        this.alder = alder;
    }

    public String toString() {
        return (navn + " er " + alder " år gammel!");
    }
}
```

Denne blir ofte brukt slik:

```
Hund fido = new Hund("Fido", 3);
System.out.print(fido.toString());
// Fido er 3 år gammel!
```

Alternativt kan vi ta for oss printingen direkte i klassen ved hjelp av en ny metode:

```
public void printInfo() {
    // Siden vi allerede har en toString-metode kan vi bruke denne
    System.out.print(toString());
}
```

Et eksempel på en metode som både tar inn et argument og som bruker klassevariabler:

```
class Hund {
    private String navn;
    private int alder;

    public Hund(String navn, int alder) {
        this.navn = navn;
        this.alder = alder;
    }

    public int getAlder() {
        return this.alder;
    }

    public boolean erLikeGamle(Hund hund) {
        return (hund.getAlder() == this.alder);
    }
}
```

Her får vi inn et objekt av typen *Hund*, som vi vet har en *getter* for variabelen *alder*. Vi sammenligner disse og returnerer *true* hvis de er like gamle og *false* om de ikke er like gamle.