Kapittel 1

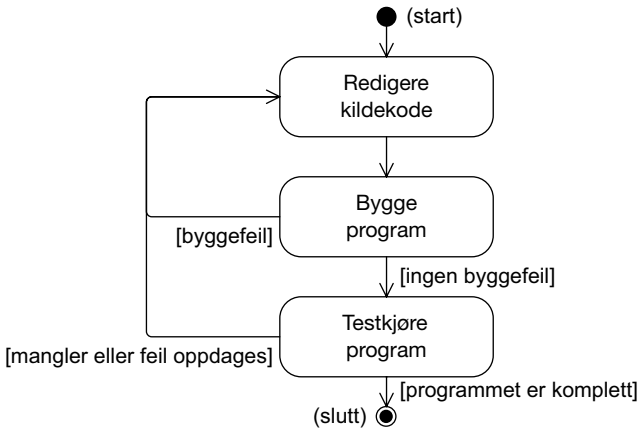
Datamaskiner og programmerings-språk

Dette kapitlet er en kort introduksjon til programmering. Vi vil se på hvordan man skriver, bygger og kjører programmer, samt illustrere noen sentrale programmeringsbegrep ved hjelp av et enkelt programeksempel. Senere kapitler vil gå grundigere igjennom begrepene som blir introdusert her.

1.1 Programmering

Moderne datamaskiner utfører sekvenser av instruksjoner raskt, nøyaktig og pålitelig. Instruksjoner arrangeres på visse måter for å få datamaskiner til å utføre både avanserte og ensformige oppgaver. De fleste som bruker dagens datamaskiner er *sluttbrukere*. De benytter seg kun av ferdiglagde programmer for å utføre vanlige arbeidsoppgaver. De har et utvalg av programmer som de bruker til ulike formål. Tekstbehandlere for å skrive dokumenter, tegne-programmer for å lage illustrasjoner, og regneark for å utføre beregninger. Å la datamaskinen utføre instruksjonene til et program kalles å *kjøre* programmet.

Figur 1.1 *Hovedaktiviteter ved programkodeskriving*



Det å lage nye programmer kalles *programmering*. Som programmerere er det vår oppgave å fortelle datamaskiner hva de skal gjøre. Figur 1.1 viser hovedaktivitetene for å lage program-

mer. Man starter med å opprette *kildekodefiler* som inneholder tekst som beskriver nøyaktig hva maskinen skal gjøre. Fra disse kildekodefilene kan man så bygge et program i maskinkjørbar form. Eventuelle feil i kildekoden som hindrer byggingen av programmet må rettes opp før man kan gå videre. Etter at programmet har blitt bygget vil man vanligvis teste det for å se om det oppfører seg slik som forventet. Hvis man oppdager feil eller mangler i programmet går man tilbake til kildekoden for å redigere og utbedre denne.

Langt mer inngår i utvikling av store programmer, men kjennskap til aktivitetene som dekkes av dette kapitlet vil la deg komme i gang med programmeringen.

I Java bygger man kjørbare programmer fra kildekode ved hjelp av en *kompilator*. Kompilatoren oversetter kildekoden til en form som er nærmere de instruksjonene prosessoren i maskinen faktisk kommer til å utføre under kjøring.

Vi skriver programmer ved hjelp av språkelementene til et *høynivå*-programmeringsspråk. I denne formen kalles programmet *kildekode* og lagres i tekstfiler. Kildekoden inneholder en høynivå-beskrivelse av hva datamaskinen skal gjøre, og må vanligvis oversettes før den kan bli utført på datamaskinen.

Det finnes mange forskjellige *programmeringsspråk*. Program 1.1 er skrevet i språket *Java*, og det er dette språket hele boken vil bruke for å vise hvordan datamaskiner kan programmeres. Selv om denne boken benytter Java, kan mange av prinsippene som vises her også benyttes for andre språk.

Program 1.1 *Kildekoden til et enkelt program*

```
// (1) Denne kildekodefilen heter EnkeltProgram.java
public class EnkeltProgram {

  // Skriver ut et ordtak, og antall tegn i ordtaket.
  public static void main(String[] args) {                    // (2)

    System.out.println("Et kjent ordtak:");                    // (3)

    String ordtak = "Øvelse gjør mester!";                    // (4)
    System.out.println(ordtak);

    int antallTegn = ordtak.length();                         // (5)
    System.out.println("Ordtaket har " + antallTegn + " tegn.");
  }
}
```

Utskrift ved kjøring:

```
Et kjent ordtak:
Øvelse gjør mester!
Ordtaket har 19 tegn.
```

Dette kildekodeeksemplet beskriver et lite program som finner ut og forteller hvor mange

tegn et ordtak inneholder. Det er ikke nødvendig å forstå oppbygningen av programmet på dette tidspunktet. Vi vil se nærmere på denne kildekoden så snart vi har introdusert noen begreper som lar oss beskrive hva programmet består av.

1.2 Oppbygningen av programmer

Operasjoner

Et program består av et sett operasjoner som blir utført når programmet blir kjørt. Operasjonene er bygget opp av sekvenser av handlinger som beskriver hva datamaskinen skal gjøre. Programmering innebærer å bruke programmeringsspråket til å beskrive operasjonene man ønsker datamaskinen skal utføre. Sluttresultatet er programmer som sluttbrukere kan benytte.

Å programmere med objekter

Operasjonene som må utføres for å fullføre en oppgave involverer ofte forskjellige typer objekter. Tenk på handlingene som må utføres for å lage en omelett: åpne kjøleskapet, ta ut en eggekartong, åpne eggekartongen, ta ut to egg, lukke kartongen, sette kartongen tilbake i kjøleskapet, lukke kjøleskapet, skru på stekeplaten, osv. I denne sammenhengen kan vi betrakte kjøleskapet, eggekartongen, eggene og stekeplaten som forskjellige objekter. Handlingene som utføres er operasjoner tilknyttet disse objektene. Hvilke handlinger som er tilgjengelige, avhenger av typen til objektet. Det er for eksempel mulig å *åpne* en eggekartong, men det er ikke mulig å *åpne* en stekepanne.

Objektbasert programmering (OBP) innebærer å beskrive større arbeidsoppgaver ved hjelp av operasjoner som utføres på objekter. Nøyaktig hva objektene representerer er avhengig av hva programmet prøver å oppnå. For eksempel kan et program som skal holde styr på alle utlån i et bibliotek ha objekter både for fysiske gjenstander som bøker, tidsskrifter, lydkassetter o.l., og for abstrakte konsepter som selve utlånet og informasjonen som registreres om hver låntaker. I det siste tilfellet er låntakeren selv et håndgripelig objekt, dvs. en person, men samlingen av informasjonen som bibliotekprogrammet trenger, f.eks. navn og adresse til en person, er et begrepsmessig objekt.

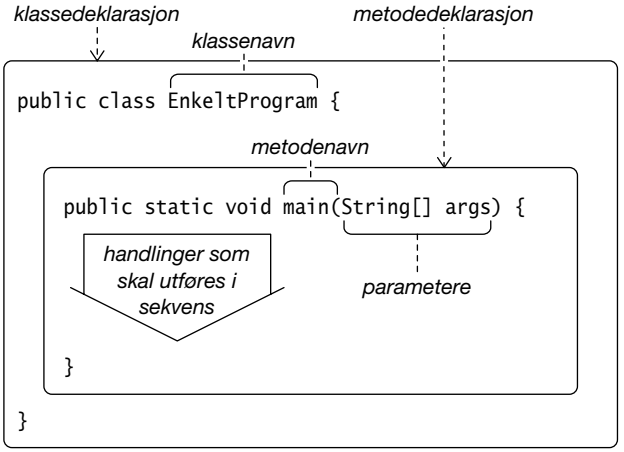
Programmer har vanligvis flere objekter av samme type. La oss gå tilbake til kjøkkenet. Vi kan ha flere objekter som representerer kakestykker. Hvert kakestykke kan håndteres uavhengig av de andre, men de har alle fellestrekk, slik som muligheten til å spise av det. Vi sier at vi har forskjellige *klasser* av objekter. Alle kjøleskap-objekter er konkrete tilfeller av en klasse, mens alle kakestykke-objekter er konkrete tilfeller av en helt annen klasse. Et objekt a som er et konkret tilfelle av en klasse kalles for en *instans* av klassen.

En klasse blir definert ved å beskrive hva som er særegent for objektene av denne klassen, og hvilke operasjoner man kan utføre på dem. Et program kan bestå av både egendefinerte klasser og klasser fra andre kilder, som f.eks. standard klassebiblioteket til Java.

1.3 Programmeringsspråket Java

Språkelementene til programmeringsspråket Java følger en viss struktur. Denne strukturen kalles språkets *syntaks*. En klassedeklarasjon er et språkelement i Java som brukes til å definerer klasser, og operasjonene som defineres i disse klassene kalles *metoder*.

Figur 1.2 *Klasse- og metodedeklarasjoner*



Figur 1.2 fremhever syntaksen til klasse- og metodedeklarasjonene fra kildekoden til Program 1.1. Programmet har en klassedeklarasjon som beskriver en klasse ved navn `EnkeltProgram`. Denne klassen har en metode ved navn `main`. Hvert språkelement har også en bestemt *mening* eller *semantikk* (eng. *semantics*). Meningen med et program er dermed gitt ved meningen til språkelementene som inngår i kildekoden.

Det er ønskelig at kildekode skal være leselig både for datamaskinen og for mennesker. For å hjelpe mennesker å lese kildekoden, er det anbefalt å legge inn kommentarer. Det er to former for kommentarer i Java. Den ene typen starter med // og varer ut resten av linjen. Linjen med `//  (1)`, dvs. den øverste linjen i Program 1.1, er en slik kommentar. For å gjøre det enklere å omtale spesifikke kildekodelinjer vil vi utover i denne boken henvise til kildekodelinje-kjennemerker som (*n*), hvor *n* er et nummer.

Den andre formen for kommentarer består av all tekst mellom tegnsekvensen /* og */. Denne formen er ikke brukt i dette programeksemplet, men vil bli brukt i senere kapitler. Kommentarer som har flere linjer med tekst blir som regel skrevet med denne formen. Kompilatoren ignorerer fullstendig kommentarer og en del spesialtegn som mellomrom og linjeskift. For kompilatorens del, så kan gjerne `main()`-metoden ved (2) i eksemplet skrives som:

```
public static void main(String[]args){System.out.println(
"Et kjent ordtak:");String ordtak="Øvelse gjør mester!";System.out.
println(ordtak);int antallTegn=ordtak.length();System.out.println(
"Ordtaket har "+antallTegn+" tegn.");}
```

Programmet vil oppføre seg helt likt, men det er vanlig å sette opp kildekoden slik at den er lett å forstå. Det er vanlig å skrive en *setning* per linje og bruke innrykk fra venstremargen for å vise at språkstrukturer er nøstet innenfor hverandre. For å utnytte side-arealet bedre vil eksemplene i denne boken bruke to mellomromstegn for innrykk.

I Java er det vanlig å la hvert innrykksnivå bestå av fire mellomromstegn, og vi anbefaler at du bruker denne innrykksdybden når du selv skriver kildekode.

Figur 1.2 viser innrykk av en metodedeklarasjon nøstet innenfor en klassedeklarasjon. Ryddig kildekode er meget viktig for å lette vedlikehold og videreutvikling av programmer.

Program 1.1 definerer en metode med navnet `main` i (2). Metoden `main()` har en spesiell rolle i et program. Utførelsen av programmet starter alltid ved denne `main()`-metoden. Alle Java programmer har en slik `main()`-metode som startpunkt. Som vist i Figur 1.2 har `main()`-metoden parameteren `String[] args`. Denne parameteren blir omtalt i delavsnittet «Metoden main() og kommandolinjeargumenter» på side 131. Når vi nevner en metode i denne boken så legger vi til «`()`» etter metodenavnet for å indikere at det er en metode. Det betyr ikke nødvendigvis at metoden ikke har noen parametere.

Når programmet starter vil datamaskinen utføre handlingene i `main()`-metoden i tur og orden. Alle kjørbare programmer må spesifisere en `main()`-metode og metoden må ha den samme formen som vist i Figur 1.2. Setningene i metoden vil selvsagt avhenge av hva programmet forsøker å oppnå.

Den første setningen som blir utført når programmet starter er (3):

```
System.out.println("Et kjent ordtak:");
```

Denne setningen utfører et *kall* til en operasjon ved navn `println` som tilhører et objekt kjent som `System.out`. Dette objektet er ansvarlig for å skrive ut tekst til terminalvinduet (Figur 1.3), og metoden `println()` kan brukes til å skrive ut en linje med tekst.

Figur 1.3 *Terminalvindu med utskrevet tekst*

Setningen ovenfor ber `System.out` objektet om å skrive ut teksten `"Et kjent ordtak:"`. En slik tekst som dette kalles en *tekststreng*. Navnet `System` angir en klasse som har et objekt (referert ved navnet `out`) som inneholder metoden `println()`.

Variabler er lagringsplasser for verdier. Tallverdier er svært vanlige, men som vi skal se senere, finnes det også andre *typer* verdier. Variabler kan brukes lokalt i metoder, slik som `ordtak` og `antall` blir brukt i `main()`-metoden i eksemplet. Å lagre en verdi i en variabel kalles

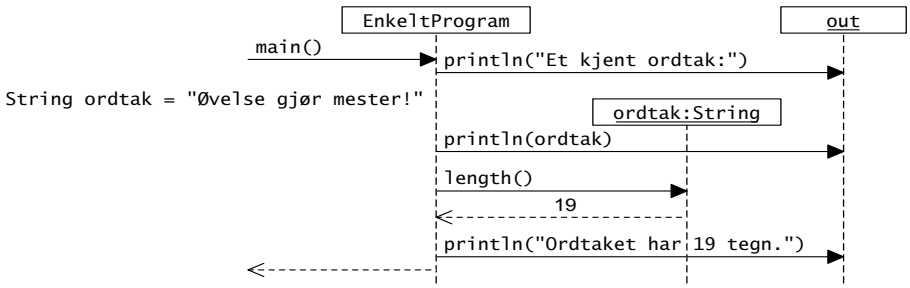
å *tilordne* (eng. *assign*) verdien til variabelen. Verdien som tilordnes kan brukes senere i programmet ved å referere til variabelen.

Setningen ved (4) deklarerer en variabel ved navn `ordtak` og tilordner denne variabelen tekststrengen `"Øvelse gjør mester"`. Programmet kan nå referere til denne strengen ved å bruke navnet `ordtak`. I den neste linjen skrives så strengen som `ordtak` refererer til ut i terminalvinduet.

I Java er strenger objekter, og setning (5) kaller metoden `length()` på streng-objektet for å finne ut hvor mange tegn strengen inneholder. Dette antallet lagres i variabelen `antallTegn`, som er definert på samme linje.

Når en metode blir brukt i en setning, vil innholdet i metoden bli utført før kjøringen av programmet fortsetter til neste setning. Dette kalles et *metodekall*. Figur 1.4 viser metodekallene som blir utført ved kjøring av programeksemplet. Ved å lese metodekallene fra topp til bunn i Figur 1.3 ser du skritt for skritt hva programmet vil gjøre.

Figur 1.4 *Sekvens av metodekall ved kjøring av program*



1.4 Redigering av kildekode

Kildekode lagres i *rene tekstfiler*. Med dette menes det at kildekodefiler kun inneholder tegnene som utgjør selve teksten i kildekoden, og har ingen stilformatering. Tekstbehandlere som Microsoft Word er ikke egnet til å skrive kildekode, siden de legger stor vekt på formateringen og utseendet til dokumentet. Det finnes egne tekstbehandlere som er egnet til å skrive kildekode, men enhver applikasjon som kan lagre ren tekst kan brukes.

Kompilatoren som bygger programmet fra kildekoden forlanger at kildekodefilen har et bestemt navn. Når du lagrer koden du redigerer, bør du være nøye med at:

- navnet til filen er lik navnet på klassen den inneholder, etterfulgt av «`.java`».
- bruk av store og små bokstaver i filnavnet er likt bruken i navnet på klassen.

Enkelte operativsystemer viser vanligvis ikke slutten på filnavnet. Det er viktig at filen ikke blir lagret med en ekstra filnavnsending. Typiske feil er å lagre kildekoden ved navn «<*navn*>`.java.doc`» eller «<*navn*>`.java.txt`». Selv om operativsystemet kun viser navnet

«<*navn*>.java», så vil ikke dette fungere når man skal kompilere kildekoden på kommandolinjen. Merk også at Microsoft Windows har alternative kortversjonsnavn for filer, som f.eks. «Klasse~1.jav». Slike filnavn kan heller ikke brukes for kildekodefiler.

1.5 Utviklingsverktøy for Java

Kompilering og kjøring av programmer kan gjøres på mange forskjellige måter, avhengig av hvilke verktøy som benyttes. De neste avsnittene vil vise hvordan man kompilerer og kjører programmer ved hjelp av standardverktøyene til Java.

Sun Microsystems tilbyr en pakke med verktøy som kalles *Java Development Kit (JDK)*. Denne pakken inneholder de grunnleggende verktøyene som trengs for å programmere i Java, og er beskrevet i vedlegg F.

I denne boken vil vi vise hvordan kompilering og kjøring av programmer gjøres på *kommandolinjen*. De fleste maskinplattformer har terminalvinduer med kommandolinjer hvor man kan skrive inn systemkommandoer man ønsker å få utført. Kommandolinjen kan være et svært effektivt verktøy for programmerere som har tatt seg tid til å lære hvordan man drar full nytte av den. Du bør ta en titt på dokumentasjonen til plattformen du bruker, hvis du trenger informasjon om hvordan du bruker kommandolinjen.

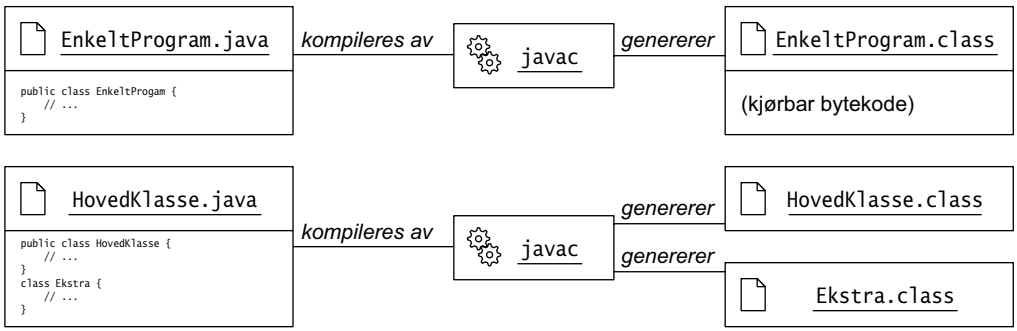
1.6 Kompilering av Java-programmer

Syntaksen for kjøring av Java-kompilatoren fra kommandolinjen er:

```
> javac EnkeltProgram.java
```

Kompilatoren `javac` leser en ren tekstfil som inneholder kildekoden, og vil for hver klassedeklarasjon i kildekoden bygge en fil som inneholder bytekoden til klassen. (Se Figur 1.5.) En av klassene definert i kildekodefilen er en *primærklasse*, og kildekodefilen er navngitt etter denne klassen. Den primære klassen er markert med nøkkelordet `public`. For små programmer er dette som oftest klassen som inneholder `main()`-metoden. Det er praktisk å dele opp store programmer i flere kildekodefiler. Da inneholder hver kildekodefil vanligvis kun én klassedeklarasjon.

Figur 1.5 *Kompilering av kildekode med en eller flere klasser*



Flere kildekodefiler kan spesifiseres i samme kommando. Kompilatoren lager filer som heter «<*klassenavn*>`.class`» for hver klasse. Disse filene inneholder bytekode som er blitt kompilert fra klassedeklarasjonene i kildekoden.

Kompilatoren kan oppdage feil i kildekoden når den prøver å oversette den til bytekode. Kompilatoren vil rapportere slike feil og avbryte kompileringen. Kildekoden må da rettes opp slik at kompilatoren kan kompilere programmet. Feilmeldingene fra kompilatoren gir opplysninger om hva den mener er galt. Med litt øvelse går det stort sett greit å tolke feilmeldingene og finne årsaken til feilene.

1.7 Kjøring

Kommandoen for å starte kjøring av kompilerte Java-programmer er `java`. Denne kommandoen må ikke forveksles med kommandoen for å starte kompilatoren som er `javac`.

Syntaksen for kjøring av Java-programmer fra kommandolinjen er:

```
> java -ea EnkeltProgram
```

Dette vil starte opp et kjøremiljø for Java som kalles for en *virtuell maskin*, og vil begynne utføringen av `main()`-metoden til den angitte klassen. Kommandoen `java` trenger det eksakte navnet til klassen med `main()`-metoden. Vær nøye med at:

- kun det eksakte klassenavnet blir angitt. Det skal ikke være noen «`.class`» eller «`.java`» ending.
- store og små bokstaver ikke forveksles i klassenavn.
- kildekoden er blitt kompilert, slik at det finnes «`.class`» filer for alle klassene som programmet består av.

Når programmet kjøres med kommandoen ovenfor skrives følgende linjer ut i terminalvinduet:

```
Et kjent ordtak:
Øvelse gjør mester!
Ordtaket har 19 tegn.
```

Ved kjøring av Java-programmer under Windows forekommer det ofte at innstillingene til terminalvinduet hvor programmet kjører får norske tegn til å se merkelige ut, f.eks:

```
+velse gj°r mester!
```

Dette kan ordnes ved å åpne egenskapene til terminalvinduet og velge «Lucida Console» istedenfor «Raster Fonts» og utføre kommandoen «`chcp 1252`» i terminalvinduet.

1.8 Den virtuelle Java-maskinen

I Java blir kildekode først kompilert til et sett lavnivå-instruksjoner som går under navnet *Java-bytekode*. Denne bytekoden blir under kjøring tolket av en virtuell maskin (*Java Virtual Machine*, JVM). Istedenfor en fysisk maskin består den virtuelle maskinen av et program som tolker og utfører bytekode-instruksjoner på samme måte som en reell maskin ville ha utført elementære instruksjoner.

Figur 1.6 viser gangen i hvordan noen linjer kildekode oversettes til kjørbar form. Java-kompilatoren kan kompilere dette til Java-bytekodeform. Denne bytekoden kan flyttes til og tolkes på alle maskiner som har en virtuell Java-maskin. Man sier at bytekoden er *plattformuavhengig*, siden den ikke er låst til en bestemt maskintype. Noen virtuelle maskiner tolker denne koden direkte, mens andre omkompilerer den til en tredje form ved kjøring. Den tredje formen vil alltid være tilpasset den typen prosessor som maskinen har. Dette betyr at kode i denne formen kun kan brukes på maskiner av en bestemt type. Denne siste formen som er vist i Figur 1.6 er spesifikk for x86-prosessorserien. Med litt trening er det mulig å følge oversettelsen fra høynivå- til lavnivåkode, f.eks. hvordan subtraksjonen i siste linje i kildekoden blir oversatt til en «`isub`» bytekode og en «`sub`» lavnivåinstruksjon.

Figur 1.6 *Programkode i forskjellige former*

1.9 KONTROLLSPØRSMÅL

Spørsmål 1.1 Datamaskiner har en _____ som utfører generelle instruksjoner og driver hele maskinen.

Spørsmål 1.2 _____ er en høynivå-beskrivelse av hva datamaskinen skal gjøre, skrevet i et _____.

Spørsmål 1.3 Hvilke av disse komponentene er programvare?

(a) en kompilator
(b) et tastatur
(c) en tolker
(d) resultatet av kompilert kildekode
(e) en prosessor

Spørsmål 1.4 Man beskriver hva som er særegent ved objekter ved å definere _____.

Spørsmål 1.5 Alle Java-programmer har en _____ ved navn _____ der utføringen av programmet starter.

Spørsmål 1.6 Hvilket utsagn beskriver objektbasert programmering (OBP) best?

(a) å lage en omelett
(b) å definere klasser av objekter og operasjoner som kan utføres av disse objektene
(c) å kompilere kildekode til Java-bytekoder
(d) å oversette programmer til prosessoravhengig kode

Spørsmål 1.7 Program 1.1 har en klasse ved navn _____ og en metode ved navn _____.

Spørsmål 1.8 For å kompilere en kildekodefil `TestProgram.java` kan kommandoen _____ utføres på kommandolinjen.

Spørsmål 1.9 For å kjøre et Java-program med en primærklasse som heter `TestProgram`, så kan kommandoen _____ utføres på kommandolinjen.

Spørsmål 1.10 Hvilken form er programkode vanligvis skrevet og redigert i?

(a) kildekode
(b) prosessoruavhengige bytekoder
(c) prosessoravhengige instruksjoner

Spørsmål 1.11 Hvilke av disse filnavnene er gyldige for en Java-kildekodefil som definerer en primærklasse som heter `Hund`?

(a) `Katt.java`
(b) `Hund.jav`
(c) `Hund.java`
(d) `Hund.java.doc`
(e) `HUND.JAVA`

1.10 OPPGAVER

Oppgave 1.1 Bruk en tekstebehandler til å skrive en fil ved navn `EnkeltProgram.java` som inneholder kildekoden fra Program 1.1.

Oppgave 1.2 Kompiler kildekodefilen du lagde i oppgave 1.1. Rett eventuelle feil i kildekoden som kompilatoren oppdager.

Oppgave 1.3 Kjør programmet som ble kompilert i oppgave 1.2.

Oppgave 1.4 Lag et program som skriver ut antall tegn i etternavnet ditt. Bruk kildekoden fra `EnkeltProgram.java` som utgangspunkt.

Oppgave 1.5 Wales har en by ved navn Llanfairpwllgwyngyllgogerychwyrndrobwyllllantysiliogogogoch. På New Zealand er det en ås som har blitt gitt det lange navnet Taumatawhakatangihangakoauauotamateapokaiwhenuakitanatahu. Lag et program som beregner hvor mange flere tegn stedsnavnet i New Zealand har i forhold til navnet på byen i Wales.