Java Programmering for Begyndere

En praktisk introduktion til programmering på 1. semester

Datamatiker Uddannelsen

12.09.2025

Indhold

1 Forord

Denne bog er en samling af undervisningsmateriale til Java programmering for begyndere på 1. semester af datamatiker uddannelsen. Bogen kombinerer teoretiske koncepter med praktiske øvelser for at give en solid grundlæggende forståelse af programmering.

Hvert kapitel er bygget op omkring et specifikt emne og indeholder både forklarende tekst og praktiske øvelser, der hjælper med at omsætte teorien til praksis.

2 Java fra scratch

2.1 Introduktion til programmering

En computer er en **standardmaskine**, der kan udføre mange forskellige opgaver. For at forstå hvordan vi programmerer, skal vi først forstå forskellen mellem hardware og software.

2.1.1 Hardware

Hardware er computerens fysiske komponenter:
- **CPU'en** beregner og udfører instrukser
- **RAM** er midlertidig arbejdshukommelse
- **Harddisk** er permanent hukommelse til filer

2.1.2 Software

Software er de programmer, der kører på computeren:
- **Operativsystem** styrer computeren (fx Windows, macOS, Android, iOS)
- **Drivere** får computeren til at kommunikere med hardware (fx keyboard, printer, skærm)
- **Applikationer** er det, vi bruger computeren til (fx Word, Photoshop, browsere, spil)

At skrive software, dvs. programmere, er både kraftfuldt og kreativt, fordi du kan løse mange forskellige problemer og skal forstå problemet for at finde en løsning.

2.2 Programmeringssprog

Ligesom vi har mange forskellige talte sprog til at kommunikere med hinanden, har vi også mange forskellige programmeringssprog til at kommunikere med computeren. Forskellige sprog har forskellige ord og syntaks (dvs. regler for rækkefølgen af ord i en sætning).

Java er et **general purpose programmeringssprog**, hvilket betyder at du kan løse mange forskellige opgaver med det:
- Apps til mobiltelefoner
- Hjemmesider og web-applikationer
- Spil (fx Minecraft er lavet i Java)
- Desktop-applikationer

Dette er i modsætning til "domæne specifikke sprog", der kun bruges til én ting, fx SQL til databaser eller HTML/CSS til hjemmesider.

2.3 Syntaks - reglerne for sproget

Syntaks er reglerne for rækkefølge af ord i programmeringssprog. Ligesom i dansk, hvor vi siger "jeg spiser en sandwich" i stedet for "spiser en sandwich jeg", har Java også specifikke regler.

Her er eksempler på korrekt og forkert Java syntaks:

Korrekt:

```
int age = 18;
System.out.println("Hello, World!");
```

Forkert:

```
age = 18 int;         // Forkert rækkefølge
System.out println("Hello, World!");  // Mangler punktum
```

2.3.1 Whitespace og indrykning

Whitespace (mellemrum, tabulatorer og linjeskift) er ikke vigtigt for computeren, men vigtigt for læsbarhed:

```
// Korrekt og læsbart
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

// Også korrekt, men svært at læse
public class HelloWorld { public static void main(String[] args)
{ System.out.println("Hello, World!"); } }
```

Vi bruger indrykning for at gøre koden lettere at læse og forstå strukturen.

2.4 Opsætning af udviklingsmiljø

For at begynde at programmere i Java, skal vi installere nogle værktøjer på computeren.

2.4.1 Installation af Java Development Kit (JDK)

Øvelse: Installér Java JDK

1. Gå til https://adoptium.net/ og download **Java JDK 21**
2. Klik på "Other Downloads" og vælg fanen **JDK 21**

3. Vælg den version der passer til dit operativsystem (Windows, macOS, Linux)
4. Åbn den downloadede fil og følg installationsvejledningen
5. Bekræft installationen ved at åbne Terminal (macOS) eller PowerShell (Windows)
6. Skriv `java -version` og tryk Enter
7. Tjek at der står **version 21** i outputtet

2.4.2 Installation af IntelliJ IDEA

IntelliJ IDEA er et udviklingsmiljø, der hjælper os med at skrive, organisere og køre Java-kode.

Øvelse: Installér IntelliJ IDEA

1. Gå til https://www.jetbrains.com/idea/download/
2. Scroll ned til **IntelliJ IDEA Community Edition** (ikke Ultimate)
3. Download og installér programmet
4. Start IntelliJ IDEA
5. Hvis du bliver spurgt om at importere indstillinger, vælg **Do not import settings**

2.5 Dit første Java-program

Nu skal vi skrive vores første Java-program - det klassiske "Hello, World!" program.

2.5.1 Opret et nyt projekt

Øvelse: Opret nyt projekt

1. Start IntelliJ IDEA og klik på **New Project**
2. Vælg **Java** som projekttype
3. Udfyld følgende:
   - **Name**: `helloworld`
   - **JDK**: Vælg JDK 21
   - **Build system**: IntelliJ
   - Fjern fluebenet i **Add sample code**
4. Klik **Create**

2.5.2 Opret din første klasse

Øvelse: Hello World program

1. I venstre sidepanel, højreklik på `src` og vælg **New** → **Java Class**

2. Navngiv klassen `HelloWorld`
3. Udfyld klassen med følgende kode:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

4. Kør programmet ved at klikke på den grønne ▷ ved siden af `main` metoden
5. Se outputtet i konsollen nederst - skriver programmet "Hello, World!"?

2.5.3 Eksperimenter med kommentarer

Øvelse: Kommentarer

1. Sæt `//` foran `System.out.println("Hello, World!");` så det bliver til en kommentar:

```
public class HelloWorld {
    public static void main(String[] args) {
        // System.out.println("Hello, World!");
    }
}
```

2. Kør programmet igen - hvad sker der?
3. Fjern kommentaren (`//`) og kør programmet igen
4. Tilføj en ny linje med en anden besked:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println("I am learning Java!");
    }
}
```

2.6 Kompilering og kørsel

Når vi skriver Java-kode, sker der følgende:

1. **Kildekode** skrives i en `.java` fil (fx `HelloWorld.java`)
2. **Kompilering** oversætter kildekoden til bytecode (`.class` filer)

3. **Java Virtual Machine (JVM)** kører bytecode-filerne

Denne proces gør Java **platform-uafhængigt** - du kan skrive kode på én computer og køre den på en anden, selvom de har forskellige operativsystemer.

Note: IntelliJ håndterer kompilering automatisk, når du kører dit program. Du behøver ikke at bekymre dig om de tekniske detaljer i begyndelsen.

2.7 Opsummering

I dette kapitel har vi lært:

- Forskellen mellem hardware og software
- Hvad et programmeringssprog er, og hvorfor vi bruger Java
- Vigtigheden af korrekt syntaks
- Hvordan man installerer Java JDK og IntelliJ IDEA
- Hvordan man opretter og kører sit første Java-program
- Hvad der sker når Java-kode kompileres og køres

Du har nu grundlaget for at begynde at lære programmering i Java!

3 Variabler og expressions

3.1 Programmer som opskrifter

Et program er ligesom en **opskrift** - en liste af trin der skal følges i rækkefølge. Computeren læser opskriften og udfører hvert trin efter hinanden.

Lad os se på et eksempel med en omelet-opskrift:

```
public class OmeletOpskrift {
    // Opskrift på omelet til 1 person
    public static void main(String[] args) {
        System.out.println("2 æg slås ud i en skål");
        System.out.println("2 æg piskes sammen");
        System.out.println("Mælk tilsættes");
        System.out.println("Salt tilsættes");
        // System.out.println("Chili tilsættes");
        System.out.println("Steg omeletten på panden");
    }
}
```

3.1.1 Statements og kommentarer

Et **statement** er en instruktion som computeren skal udføre. I Java slutter alle statements med semikolon (`;`):

```
System.out.println("Steg på en pande");
```

En **kommentar** er tekst som bliver ignoreret af computeren. Kommentarer starter med `//` og fortsætter til slutningen af linjen:

```
// Dette er en kommentar
System.out.println("Dette er et statement");
```

Kommentarer er nyttige til:

- At forklare hvad koden gør
- At "slå en linje fra" så den ikke bliver udført

Øvelse: ASCII Art

Konsol-applikationer består udelukkende af tekst. I monospace-fonte (kode-fonte) er alle tegn samme bredde, hvilket gør det muligt at lave "grafik" med tekst.

1. Lav et nyt Java-projekt kaldet `asciiart`
2. Opret en klasse kaldet `AsciiArt` med en `main` metode
3. Brug `System.out.println` til at printe følgende ASCII art:

```
#     #
#     # ###### #      #       ####
#     # #      #      #      #    #
####### #####  #      #      #    #
#     # #      #      #      #    #
#     # #      #      #      #    #
#     # ###### ###### ######  ####
```

4. Prøv at sammensætte flere linjer til én streng:

```
String lines = "Første linje\n" +
               "Anden linje\n" +
               "Tredje linje";
System.out.println(lines);
```

3.2 Variabler - opbevaring af data

En variabel er som en "boks" i computerens hukommelse, hvor du kan gemme data. Dette gør det muligt at genbruge værdier og gøre programmer mere fleksible.

Forestil dig at vi skal lave omelet til flere personer. I stedet for at skrive "2 æg" igen og igen, kan vi gemme antallet i en variabel:

```
public class FlexibelOmelet {
    public static void main(String[] args) {
        int eggs = 2;  // Antal æg til 1 person

        System.out.println("Ingredienser:");
        System.out.println("Æg (stk): " + eggs);

        System.out.println(eggs + " æg slås ud i en skål");
        System.out.println(eggs + " æg piskes sammen");
        System.out.println("Mælk tilsættes");
        System.out.println("Salt tilsættes");
        System.out.println("Steg omeletten på panden");
    }
}
```

3.2.1 Datatyper

Java har forskellige typer af data, som hver har deres egen datatype:

3.2.1.1 Heltal (int)

```
int age = 25;
int numberOfStudents = 30;
```

3.2.1.2 Decimaltal (double)

```
double price = 19.95;
double temperature = 23.5;
```

3.2.1.3 Tegn (char)

```
char grade = 'A';
char firstLetter = 'J';
```

3.2.1.4 Tekst (String)

```
String name = "Anders";
String city = "København";
```

3.2.1.5 Sand/falsk (boolean)

```
boolean isStudent = true;
boolean hasPassed = false;
```

3.2.2 Variabel-erklæring

For at oprette en variabel skal du:
1. Angive datatypen
2. Give variablen et navn
3. Tildele en værdi (valgfrit)

```
// Erklæring med tildeling
int numberOfCars = 5;

// Erklæring uden tildeling
int numberOfBikes;
numberOfBikes = 3;  // Tildeling senere
```

Øvelse: Om mig

1. Lav et nyt Java-projekt kaldet `about-me`

2. Opret en klasse kaldet `AboutMe` med en `main` metode
3. Erstat `???` med værdier der passer til dig:

```
public class AboutMe {
    public static void main(String[] args) {
        String familyName = ???;            // Dit efternavn
        char givenNameInitial = ???;        // Første bogstav i dit
fornavn
        boolean isCoffeeDrinker = ???;      // Drikker du kaffe?
        boolean isTeaDrinker = ???;         // Drikker du te?

        System.out.println("Navn: " + givenNameInitial + ". " +
familyName);
        System.out.println("Kaffe? " + isCoffeeDrinker);
        System.out.println("Te? " + isTeaDrinker);
    }
}
```

3.3 Expressions - beregninger og sammensætninger

Et **expression** er noget der evalueres til en værdi. Dette kan være:

3.3.1 Aritmetiske expressions

```
int sum = 5 + 3;         // Addition
int difference = 10 - 4; // Subtraktion
int product = 6 * 7;     // Multiplikation
int quotient = 15 / 3;   // Division
int remainder = 17 % 5;  // Modulo (rest)
```

3.3.2 String sammensætning

```
String firstName = "Anders";
String lastName = "Jensen";
String fullName = firstName + " " + lastName;  // "Anders Jensen"
```

3.3.3 Blanding af typer

```
String message = "Jeg er " + 25 + " år gammel";  // "Jeg er 25 år gammel"
```

Øvelse: Fødselsdags-beregner

1. Lav et nyt Java-projekt kaldet `birthyear-calculator`
2. Opret en klasse med en `main` metode
3. Brug variabler til at beregne dit fødselsår baseret på din alder:

```
int currentYear = 2024;
int myAge = ???;  // Din alder
int birthYear = currentYear - myAge;

System.out.println("Jeg er " + myAge + " år gammel");
System.out.println("Jeg blev født i " + birthYear);
```

3.4 Navngivning af variabler

Gode variabelnavne er vigtige for læsbar kode:

3.4.1 Regler for variabelnavne

- Må ikke starte med et tal
- Må ikke indeholde mellemrum
- Må ikke være reserverede ord (fx `int`, `class`)
- Skelner mellem store og små bogstaver

3.4.2 Konventioner

```
// Brug camelCase for variabelnavne
int numberOfStudents;
String firstName;
boolean isCompleted;

// Vær beskrivende
int n;              // Dårligt - hvad er n?
int studentCount;   // Godt - klart hvad det er
```

Øvelse: Cookies beregning

1. Lav et program der beregner hvor mange cookies der skal laves
2. Brug følgende variabler:
   - Antal gæster
   - Cookies per person
   - Total antal cookies

```java
int guests = 8;
int cookiesPerPerson = 3;
int totalCookies = guests * cookiesPerPerson;

System.out.println("Vi har " + guests + " gæster");
System.out.println("Hver skal have " + cookiesPerPerson + "
cookies");
System.out.println("Vi skal bage " + totalCookies + " cookies i
alt");
```

3.5 Ændring af variabel-værdier

Når en variabel er oprettet, kan dens værdi ændres:

```java
int score = 0;
System.out.println("Start score: " + score);  // 0

score = 10;
System.out.println("Ny score: " + score);     // 10

score = score + 5;
System.out.println("Final score: " + score);  // 15
```

Note: En variabel kan kun indeholde én værdi ad gangen. Når du tildeler en ny værdi, forsvinder den gamle værdi.

3.6 Opsummering

I dette kapitel har vi lært:

- Programmer er som opskrifter med trin der udføres i rækkefølge
- Statements er instruktioner til computeren
- Kommentarer forklarer koden og ignoreres af computeren
- Variabler gemmer data i computerens hukommelse
- Java har forskellige datatyper: `int`, `double`, `char`, `String`, `boolean`
- Expressions evalueres til værdier og kan indeholde beregninger
- Gode variabelnavne gør koden lettere at læse og forstå

Med variabler kan vi skrive mere fleksible programmer der kan arbejde med forskellige værdier uden at skulle ændre hele koden.

4 Metoder

Indtil nu har alle vores programmer bestået af en enkelt `main`-metode, hvor vi har skrevet al vores kode. Men efterhånden som programmerne bliver større og mere komplekse, bliver det vigtigt at kunne organisere koden i mindre, genbrugelige dele. Det er her metoder kommer ind i billedet.

En metode er en navngiven blok af kode, der udfører en specifik opgave. Metoder gør det muligt at:

- Organisere koden i logiske enheder
- Genbruge kode i stedet for at skrive den samme kode flere gange
- Gøre programmer lettere at læse og vedligeholde
- Opdele komplekse problemer i mindre, håndterbare dele

4.1 Hvad er en metode?

Tænk på en metode som en mini-program inden i dit program. Ligesom en opskrift har et navn og beskriver en række trin, har en metode et navn og indeholder kode, der udfører en specifik opgave.

```
public class CakeRecipe {
    public static void main(String[] args) {
        System.out.println("Afmål 300 g smør");
        System.out.println("Put det i skålen");
        System.out.println("Afmål 200 g sukker");
        System.out.println("Put det i skålen");
        System.out.println("Afmål 100 g mel");
        System.out.println("Put det i skålen");
        System.out.println("Pisk æggehvider fra 3 æg");
        System.out.println("Put det i skålen");
        System.out.println("Put dejen i en bageform");
        System.out.println("Bag i ovnen i 30 minutter");
    }
}
```

I eksemplet ovenfor gentager vi konstant “Put det i skålen”. Dette kunne vi gøre mere effektivt med metoder.

4.2 Oprettelse af metoder

En metode i Java har følgende struktur:

```
```

```java
public static returtype metodenavn(parametertype parameternavn) {
    // kode der skal udføres
    return værdi; // kun hvis returtype ikke er void
}
```

Lad os se på et simpelt eksempel:

```java
public static void sayHello() {
    System.out.println("Hej!");
}
```

- `public static` - nøgleord der fortæller Java hvordan metoden kan bruges
- `void` - betyder at metoden ikke returnerer en værdi
- `sayHello` - navnet på metoden
- `()` - parenteser til parametre (tomme i dette tilfælde)

Øvelse: Brød-opskrift

Lav et program der udskriver en brød-opskrift. Start med at skrive hele opskriften i `main`-metoden:

```
Tilsæt 300 ml. vand til skålen
Tilsæt 10 g. gær til skålen
Tilsæt 500 g. hvedemel til skålen
Ælt dejen i 5 minutter
Lad dejen hæve i 1 time
Ælt dejen i 5 minutter
Lad dejen hæve i 1 time
Ælt dejen i 5 minutter
Lad dejen hæve i 1 time
Tilsæt 10 g. salt til skålen
Ælt dejen i 5 minutter
Lad dejen hæve i 1 time
Form dejen til et brød
Bag brødet i ovnen ved 220 grader i 30 minutter
```

Læg mærke til de gentagne trin. Lav metoder for de dele der gentager sig, f.eks. `eltDejen()` og `ladDejenHaeve()`.

4.3 Parametre og argumenter

Metoder bliver endnu mere nyttige når de kan modtage input i form af parametre:

```java
public static void greet(String name) {
    System.out.println("Hej, " + name + "!");
}

public static void main(String[] args) {
    greet("Anna");
    greet("Peter");
}
```

Her har metoden `greet` en parameter `name` af typen `String`. Når vi kalder metoden, sender vi et argument (f.eks. "Anna") som bliver tildelt til parameteren.

4.4 Returværdier

Metoder kan returnere en værdi ved hjælp af `return`-nøgleordet:

```java
public static int add(int a, int b) {
    return a + b;
}

public static void main(String[] args) {
    int result = add(5, 3);
    System.out.println("Resultatet er: " + result);
}
```

Når en metode returnerer en værdi, skal vi specificere returtypen (i dette tilfælde `int`) i stedet for `void`.

Øvelse: Returtyper

Udfyld de manglende returtyper i følgende metoder:

```java
public static ??? add(int a, int b) {
    return a + b;
}

public static ??? add(double a, double b) {
    return a + b;
}

public static ??? isWeekend(int dayOfWeek) {
    return dayOfWeek == 6 || dayOfWeek == 7;
}

public static ??? fullName(String firstName, String lastName) {
```

```
    return firstName + " " + lastName;
}

public static ??? greet(String name) {
    System.out.println("Hello, " + name + "!");
}
```

4.5 Forskellige typer af metoder

4.5.1 Void-metoder

Metoder der ikke returnerer en værdi bruger `void` som returtype:

```
public static void printWelcome() {
    System.out.println("Velkommen til programmet!");
}
```

4.5.2 Metoder med returværdi

Metoder der beregner og returnerer en værdi:

```
public static double calculateBMI(double weight, double height) {
    return weight / (height * height);
}
```

4.5.3 Metoder med flere parametre

Metoder kan have mange parametre:

```
public static void printPersonInfo(String name, int age, String city) {
    System.out.println(name + " er " + age + " år og bor i " + city);
}
```

4.6 Formateret udskrift med printf

Indtil nu har vi brugt `System.out.println` til at udskrive tekst. For mere præcis formatering kan vi bruge `System.out.printf`:

```java
String name = "Benny";
int age = 25;
double height = 1.75455;
System.out.printf("%s på %d år er %.2f m høj%n", name, age, height);
```

Dette udskriver: "Benny på 25 år er 1.75 m høj"

Formatstrenge:

- `%s` - String
- `%d` - heltal (int)
- `%.2f` - decimal med 2 decimaler
- `%10.2f` - decimal med 2 decimaler i et felt på 10 tegn
- `%n` - ny linje

Øvelse: Formateret prisudskrift

Brug `printf` til at udskrive følgende formateret:

```
Pizza Margherita:  89,00 kr
Pizza Pepperoni:   99,00 kr
Pizza Portobello:  69,50 kr
```

4.7 Børnesangsgenerator

Mange børnesange har vers der gentager sig med få ændringer. "Jens Hansens bondegård" er et godt eksempel, hvor hvert vers har et dyr og dets lyde.

```java
public static void generateVerse(String animal, String sound) {
    System.out.println("På Jens Hansens bondegård");
    System.out.println("Er der en " + animal);
    System.out.println("Som laver " + sound + ", " + sound);
    System.out.println("Hele dagen lang");
    System.out.println();
}

public static void main(String[] args) {
    generateVerse("hest", "prr-prr");
    generateVerse("ko", "muh-muh");
    generateVerse("gris", "øf-øf");
}
```

4.8 Math-klassen

Java har en indbygget `Math`-klasse med mange nyttige matematiske metoder:

```
double result1 = Math.sqrt(16);       // Kvadratrod: 4.0
double result2 = Math.pow(2, 3);      // 2 i 3. potens: 8.0
double result3 = Math.abs(-5);        // Absolut værdi: 5.0
double result4 = Math.max(10, 20);    // Største værdi: 20.0
double result5 = Math.min(10, 20);    // Mindste værdi: 10.0
```

4.9 Escape-tegn i strings

Nogle gange skal vi bruge specielle tegn i strings:

```
System.out.println("Han sagde \"Hej!\"");          // Anførselstegn
System.out.println("Første linje\nAnden linje");    // Ny linje
System.out.println("Tab\tmellemrum");               // Tabulator
System.out.println("Backslash: \\");                // Backslash
```

Øvelse: Email-brevfletning

Lav en metode der kan generere personlige emails:

```
public static void generateEmail(String name, String product, double
price) {
    // Generer en email der hilser på personen og fortæller om
produktet
}
```

Metoden skal udskrive en email som:

```
Kære [navn],

Vi har et særligt tilbud på [produkt] til kun [pris] kr.

Venlig hilsen,
Webshop Team
```

4.10 Metoders fordele

Metoder giver os flere fordele:

1. **Genbrugelighed**: Vi kan kalde den samme metode mange steder

2. **Læsbarhed**: Koden bliver lettere at forstå når den er opdelt i logiske dele
3. **Vedligeholdelse**: Ændringer skal kun laves ét sted
4. **Testing**: Hver metode kan testes separat
5. **Abstraktion**: Vi kan skjule komplekse detaljer bag simple metodekald

Note: Når du nævner en metode, skal du altid inkludere parenteserne, selv hvis den ikke har parametre. Skriv `sayHello()` i stedet for bare `sayHello`.

4.11 Sammenfatning

Metoder er fundamentale byggeklodser i Java-programmering. De gør det muligt at:

- Organisere kode i genbrugelige blokke
- Modtage input gennem parametre
- Returnere resultater
- Formatere output præcist med printf
- Arbejde med matematiske funktioner gennem Math-klassen

I næste kapitel vil vi lære om input og hvordan vi kan læse data fra brugeren for at gøre vores programmer mere interaktive.

5 Input og Scanner

Dette kapitel vil handle om hvordan man læser input fra brugeren i Java.

5.1 Scanner-klassen

Scanner-klassen bruges til at læse input fra brugeren.

6 Betingelser og Logik

Dette kapitel vil handle om betingelser og logik i Java.

6.1 If-statements

If-statements bruges til at lave beslutninger i kode.

7 Løkker og Rekursion

Indtil nu har vi skrevet programmer hvor hver linje kode bliver udført præcis én gang. Men ofte har vi brug for at gentage den samme kode flere gange. Det er her løkker kommer ind i billedet.

En løkke er en programmeringsstruktur, der gør det muligt at gentage en blok af kode flere gange. Dette sparer os for at skrive den samme kode igen og igen, og gør det muligt at håndtere opgaver der kræver gentagelse.

7.1 Hvorfor løkker?

Forestil dig at du skal lave en vejrtrækningsøvelse, hvor hver cyklus består af fire trin:

```
public class BreathingExercise {
    public static void main(String[] args) {
        System.out.println("1. Træk vejret langsomt ind");
        System.out.println("2. Hold vejret");
        System.out.println("3. Pust langsomt ud");
        System.out.println("4. Hold vejret");

        System.out.println("1. Træk vejret langsomt ind");
        System.out.println("2. Hold vejret");
        System.out.println("3. Pust langsomt ud");
        System.out.println("4. Hold vejret");

        // ... gentag 8 gange mere?
    }
}
```

Som du kan se, bliver dette hurtigt kedeligt og upraktisk. Hvad hvis vi skal gentage det 100 gange?

Med metoder kan vi gøre det lidt bedre:

```
public static void breathCycle() {
    System.out.println("1. Træk vejret langsomt ind");
    System.out.println("2. Hold vejret");
    System.out.println("3. Pust langsomt ud");
    System.out.println("4. Hold vejret");
}

public static void main(String[] args) {
    breathCycle();
    breathCycle();
```

```
    breathCycle();
    // ... stadig meget kedeligt
}
```

Men med løkker kan vi gøre det meget nemmere!

7.2 While-løkker

En `while`-løkke gentager en blok af kode så længe en betingelse er sand.

```
while (betingelse) {
    // kode der skal gentages
}
```

Lad os bruge dette til vejrtrækningsøvelsen:

```
public static void main(String[] args) {
    int repetitions = 10;

    while (repetitions > 0) {
        breathCycle();
        repetitions = repetitions - 1;
    }
}
```

Note: Husk at ændre variablen der bruges i betingelsen (her `repetitions`) inde i løkken, ellers får du en uendelig løkke!

7.2.1 While-løkke struktur

En while-løkke har tre vigtige dele:

1. **Initialisering** - sæt startværdier før løkken
2. **Betingelse** - tjek om løkken skal fortsætte
3. **Opdatering** - ændr variabler inde i løkken

```
int counter = 0;                    // 1. Initialisering
while (counter < 5) {               // 2. Betingelse
    System.out.println(counter);
    counter++;                      // 3. Opdatering
}
```

Øvelse: Gæt et tal

Lav et program hvor brugeren skal gætte et hemmeligt tal:

1. Sæt et hemmeligt tal (f.eks. 42)
2. Bed brugeren om at gætte
3. Hvis gættet er forkert, bed dem prøve igen
4. Fortsæt indtil de gætter rigtigt
5. Udskriv hvor mange forsøg det tog

Brug Scanner til at læse input fra brugeren.

7.3 For-løkker

Når vi ved præcis hvor mange gange noget skal gentages, er `for`-løkker ofte mere praktiske:

```
for (int i = 0; i < 10; i++) {
    breathCycle();
}
```

En for-løkke har tre dele adskilt af semikolon:

1. **Initialisering** (`int i = 0`) - udføres én gang før løkken starter
2. **Betingelse** (`i < 10`) - tjekkes før hver iteration
3. **Opdatering** (`i++`) - udføres efter hver iteration

7.3.1 For-løkke variationer

Du kan tælle på mange måder:

```
// Tæl op fra 0 til 9
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}

// Tæl ned fra 10 til 1
for (int i = 10; i > 0; i--) {
    System.out.println(i);
}
System.out.println("Lift off!");

// Tæl med andre intervaller (lige tal)
for (int i = 0; i < 20; i += 2) {
    System.out.println(i);
}

// Tæl gennem bogstaver
for (char c = 'A'; c ⩽ 'Z'; c++) {
```

```
    System.out.print(c + " ");
}
```

Øvelse: 7-tabellen

Lav et program der udskriver 7-tabellen:

1. Udskriv tallene 7, 14, 21, ..., 70
2. Formatér det som "1 **7 = 7**", "**2** 7 = 14", osv.
3. Brug `printf` til at få kolonnerne til at stå pænt:

```
 1 * 7 =  7
 2 * 7 = 14
 3 * 7 = 21
...
10 * 7 = 70
```

4. Udvid til at vise alle tabeller fra 1 til 10

7.4 Tilfældige tal med Random

Java's `Random` klasse gør det muligt at generere tilfældige tal:

```
import java.util.Random;

public class RandomExample {
    public static void main(String[] args) {
        Random random = new Random();

        int randomNumber = random.nextInt(100);      // 0-99
        int diceRoll = random.nextInt(6) + 1;        // 1-6
        double randomDouble = random.nextDouble();   // 0.0-1.0
        boolean coinFlip = random.nextBoolean();     // true/false
    }
}
```

Øvelse: Gæt et random tal

Udvid dit "gæt et tal" program:

1. Lad computeren vælge et tilfældigt tal mellem 1 og 100
2. Giv brugeren hints ("for højt" eller "for lavt")
3. Tæl antallet af forsøg
4. Eksperimenter med seeds: `new Random(123)` - hvad sker der?

7.5 Indlejrede løkker

Du kan have løkker inden i andre løkker. Dette er nyttigt til at arbejde med todimensionale strukturer:

```
// Udskriv en multiplikationstabel
for (int row = 1; row ⩽ 10; row++) {
    for (int col = 1; col ⩽ 10; col++) {
        System.out.printf("%4d", row * col);
    }
    System.out.println(); // Ny linje efter hver række
}
```

Øvelse: Sten-saks-papir

Lav et sten-saks-papir spil:

1. Brugeren vælger sten (1), saks (2) eller papir (3)
2. Computeren vælger tilfældigt
3. Bestem vinderen
4. Spørg om brugeren vil spille igen
5. Tæl hvor mange gange hver spiller vinder

7.6 Løkker med Strings

Løkker er særligt nyttige til at arbejde med strings:

```
String text = "Hello, World!";

// Udskriv hvert tegn på en separat linje
for (int i = 0; i < text.length(); i++) {
    char c = text.charAt(i);
    System.out.println(c);
}

// Tæl hvor mange gange hvert bogstav forekommer
for (char letter = 'a'; letter ⩽ 'z'; letter++) {
    int count = 0;
    for (int i = 0; i < text.length(); i++) {
        if (Character.toLowerCase(text.charAt(i)) == letter) {
            count++;
        }
    }
    if (count > 0) {
        System.out.println(letter + ": " + count);
    }
}
```

7.6.1 String metoder til løkker

```
String text = "Hello World";

// Længde af string
int length = text.length();

// Få tegn på position i
char c = text.charAt(i);

// Del af string (substring)
String part = text.substring(0, 5); // "Hello"

// Find position af tegn/string
int pos = text.indexOf('o');        // 4

// Konverter til store/små bogstaver
String upper = text.toUpperCase();
String lower = text.toLowerCase();
```

Øvelse: Palindrom checker

Lav et program der tjekker om et ord er et palindrom (læses ens forfra og bagfra):

1. Læs et ord fra brugeren
2. Sammenlign første bogstav med sidste, andet med næstsidste, osv.
3. Udskriv om ordet er et palindrom
4. Test med ord som "racecar", "hello", "radar"

7.7 Sum og gennemsnit med løkker

Løkker er perfekte til beregninger over flere værdier:

```
// Beregn sum af tal fra 1 til 100
int sum = 0;
for (int i = 1; i ⩽ 100; i++) {
    sum += i;
}
System.out.println("Sum: " + sum);

// Beregn gennemsnit af karakterer
double[] grades = {7, 4, 10, 12, 2, 7, 10};
double sum = 0;
for (int i = 0; i < grades.length; i++) {
    sum += grades[i];
}
```

```java
double average = sum / grades.length;
System.out.println("Gennemsnit: " + average);
```

Øvelse: Diplomberegning

Lav et program der beregner et uddannelsesdiplom:

1. Definer karakterer for forskellige fag
2. Beregn gennemsnittet
3. Bestem om den studerende består (gennemsnit >= 2.0)
4. Udskriv et pænt diplom

Bonus: Beregn vægtet gennemsnit hvor forskellige fag har forskellige ECTS point.

7.8 Hvornår bruger man hvilken løkke?

- **For-løkker**: Når du ved præcis hvor mange gange noget skal gentages
  - Tælle fra 1 til 10
  - Gå gennem hvert tegn i en string
  - Bearbejde alle elementer i et array
- **While-løkker**: Når du ikke ved hvor mange gange det skal gentages
  - Læs input indtil brugeren skriver "quit"
  - Gæt et tal indtil det er rigtigt
  - Fortsæt indtil en betingelse er opfyldt

7.9 Uendelige løkker og fejlsøgning

Pas på uendelige løkker! Disse opstår når betingelsen aldrig bliver falsk:

```java
// FARLIGT - uendelig løkke!
int i = 0;
while (i < 10) {
    System.out.println(i);
    // Glemte at øge i!
}

// OGSÅ FARLIGT
for (int i = 0; i < 10; i--) {  // Tæller nedad i stedet for op!
    System.out.println(i);
}
```

Hvis dit program "hænger", er det sandsynligvis en uendelig løkke. Du kan stoppe programmet med Ctrl+C i terminalen.

7.10 Rekursion - en alternativ tilgang

Rekursion er når en metode kalder sig selv. Det kan nogle gange erstatte løkker:

```
public static void countdown(int n) {
    if (n ⩽ 0) {
        System.out.println("Lift off!");
    } else {
        System.out.println(n);
        countdown(n - 1);  // Metoden kalder sig selv
    }
}
```

Rekursion kræver altid:
1. En **base case** - betingelse for hvornår rekursionen stopper
2. Et **recursive call** - metoden kalder sig selv med ændrede parametre

Note: Rekursion er kraftfuld, men kan være svær at forstå i begyndelsen. Start med løkker og kom tilbage til rekursion senere.

7.11 Sammenfatning

Løkker er et af de vigtigste værktøjer i programmering. De gør det muligt at:

- Gentage kode effektivt uden duplikering
- Arbejde med samlinger af data
- Lave interaktive programmer der kører indtil brugeren stopper
- Beregne værdier over mange elementer

For-løkker er bedst når du ved hvor mange gange noget skal gentages. **While-løkker** er bedst når du skal fortsætte indtil en betingelse er opfyldt.

I næste kapitel vil vi lære om arrays, som giver os mulighed for at gemme og bearbejde mange værdier på én gang - perfect til brug med løkker!

8 Arrays og Referencer

Dette kapitel vil handle om arrays og referencer i Java.

8.1 Arrays

Arrays bruges til at gemme flere værdier af samme type.