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

Indtil nu har vores programmer kun kunne udskrive information. Men for at lave virkelig interaktive programmer har vi brug for at kunne læse input fra brugeren. I dette kapitel lærer vi om Scanner-klassen og hvordan vi konverterer mellem forskellige datatyper.

5.1 Casting - konvertering mellem datatyper

Før vi dykker ned i input, er det vigtigt at forstå hvordan Java håndterer konvertering mellem forskellige datatyper.

5.1.1 Implicit casting (automatisk)

Nogle konverteringer sker automatisk uden tab af information:

```
int x = 4;
double y = x;  // y er nu 4.0 - går godt
System.out.println(y);  // 4.0
```

Dette virker fordi en `double` kan indeholde alle værdier som en `int` kan, plus decimaler.

5.1.2 Eksplicit casting (manuel)

Andre konverteringer kan resultere i tab af information og skal gøres eksplicit:

```
double x = 2.7;
int y = (int) x;  // y er nu 2 - decimaldelen forsvinder
System.out.println(y);  // 2
```

Øvelse: Casting quiz

Gæt hvad følgende kode udskriver, eller om den giver en fejl:

1.
```
int a = 5;
double b = a;
System.out.println(b);
```

2.

```java
double a = 5.5;
int b = (int) a;
System.out.println(b);
```

3.
```java
char a = 'A';
int b = a;
System.out.println(b);  // Hint: ASCII værdi
```

4.
```java
int a = 66;
char b = (char) a;
System.out.println(b);  // Hvad er ASCII 66?
```

5.1.3 Parsing - konvertering fra String

Strings kan ikke castes direkte til tal. I stedet bruger vi parsing:

```java
String x = "4";
int y = Integer.parseInt(x);      // y er nu 4
double z = Double.parseDouble("2.7");  // z er nu 2.7
boolean w = Boolean.parseBoolean("true");  // w er nu true
```

Hvis strengen ikke kan konverteres, får vi en fejl:

```java
int x = Integer.parseInt("hello");  // Fejl!
```

5.2 System.out og System.in

Vi har brugt `System.out.println` mange gange, men hvad er `System.out` egentlig?

```java
System.out.println(System.out);  // java.io.PrintStream@15db9742
```

`System` er en klasse i Java's standardbibliotek:

- `System.out` er et `PrintStream` objekt til at skrive til konsollen
- `System.in` er et `InputStream` objekt til at læse fra tastaturet

Men `System.in` er besværligt at bruge direkte, da det kun læser bytes. Derfor bruger vi Scanner-klassen.

5.3 Scanner-klassen

Scanner-klassen gør det nemt at læse forskellige datatyper fra input:

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Indtast dit navn: ");
        String name = scanner.nextLine();

        System.out.print("Indtast din alder: ");
        int age = scanner.nextInt();

        System.out.println("Hej " + name + ", du er " + age + " år
gammel!");
    }
}
```

Note: Husk at importere Scanner-klassen med `import java.util.Scanner;` øverst i din fil.

5.3.1 Scanner metoder

Scanner har mange nyttige metoder:

```
Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();     // Læser hele linje
String word = scanner.next();         // Læser næste ord
int number = scanner.nextInt();       // Læser heltal
double decimal = scanner.nextDouble(); // Læser decimaltal
boolean bool = scanner.nextBoolean(); // Læser true/false
```

5.3.2 Scanner med strings

Scanner kan også læse fra strings i stedet for tastaturet:

```
String data = "4 121 33 14";
Scanner scanner = new Scanner(data);

int a = scanner.nextInt();  // 4
int b = scanner.nextInt();  // 121
```

```java
int c = scanner.nextInt();  // 33
int d = scanner.nextInt();  // 14
```

Dette er nyttigt til at parse data fra filer eller andre kilder.

Øvelse: Fodboldresultater

Lav et program der læser fodboldresultater:

1. Opret en Scanner der læser fra strengen "3 1"
2. Læs hjemmeholdets og udeholdets mål
3. Udskriv resultatet pænt:

```
Hjemmehold: 3 mål
Udehold: 1 mål
```

4. Udvid programmet til at læse fra `System.in` i stedet
5. Bed brugeren om at indtaste resultatet og vis det pænt formateret

5.4 Betingelser med input

Input bliver særligt kraftfuldt når vi kombinerer det med betingelser (som vi lærer mere om i næste kapitel):

```java
Scanner scanner = new Scanner(System.in);

System.out.print("Indtast resultatet af kampen (hjemme udehold): ");
int home = scanner.nextInt();
int away = scanner.nextInt();

System.out.println("Hjemmehold: " + home + " mål");
System.out.println("Udehold: " + away + " mål");

if (home > away) {
    System.out.println("Hjemmeholdet vandt!");
} else if (away > home) {
    System.out.println("Udeholdet vandt!");
} else {
    System.out.println("Det blev uafgjort!");
}
```

Øvelse: Hvem vandt?

Udvid dit fodboldresultat-program:

1. Læs hjemme- og udeholdets mål fra brugeren

2. Bestem og udskriv hvem der vandt kampen
3. Test med forskellige resultater

Forklar: Hvorfor behøver vi ikke tjekke `away == home` under `else`?

5.5 Interaktive programmer

Med Scanner kan vi lave programmer der reagerer på brugerens input:

```java
import java.util.Scanner;

public class PersonInfo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Hvad hedder du? ");
        String name = scanner.nextLine();

        System.out.print("Hvor gammel er du? ");
        int age = scanner.nextInt();

        System.out.print("Hvor høj er du (i meter)? ");
        double height = scanner.nextDouble();

        System.out.println("\nDine oplysninger:");
        System.out.println("Navn: " + name);
        System.out.println("Alder: " + age + " år");
        System.out.printf("Højde: %.2f meter%n", height);

        if (age ⩾ 18) {
            System.out.println("Du er myndig!");
        } else {
            System.out.println("Du er under 18 år.");
        }
    }
}
```

5.5.1 Fejlhåndtering

Pas på at brugeren indtaster den rigtige type data:

```java
Scanner scanner = new Scanner(System.in);
System.out.print("Indtast et tal: ");

// Dette vil fejle hvis brugeren skriver "hello"
int number = scanner.nextInt();
```

I begyndelsen kan du bede brugeren om at indtaste det rigtige format. Senere lærer du om try-catch til at håndtere fejl.

Øvelse: Prinsessen skal giftes

Prinsessen skal giftes og har specifikke krav til sin prins. Implementer følgende metode:

```
public static boolean canMarry(int age, boolean isHandsome,
                               boolean isBrave, boolean isRich) {
    if (age < 18) {
        return false;
    }
    if (isHandsome) {
        if (isBrave || isRich) {
            return true;
        }
    }
    return false;
}
```

Test med disse kandidater:

- Prins Charming (20 år, flot, ikke modig, ikke rig)
- Prins Ib (22 år, ikke flot, modig, ikke rig)
- Prins Bieber (31 år, flot, ikke modig, rig)
- Prins Blop (17 år, flot, modig, rig)

Hvem kan prinsessen gifte sig med?

5.6 Calculator eksempel

Lad os lave en simpel lommeregner:

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Indtast første tal: ");
        double a = scanner.nextDouble();

        System.out.print("Indtast operation (+, -, *, /): ");
        String operator = scanner.next();

        System.out.print("Indtast andet tal: ");
        double b = scanner.nextDouble();

        double result = 0;
```

```
        if (operator.equals("+")) {
            result = a + b;
        } else if (operator.equals("-")) {
            result = a - b;
        } else if (operator.equals("*")) {
            result = a * b;
        } else if (operator.equals("/")) {
            if (b != 0) {
                result = a / b;
            } else {
                System.out.println("Fejl: Division med nul!");
                return;
            }
        } else {
            System.out.println("Ukendt operation: " + operator);
            return;
        }

        System.out.printf("%.2f %s %.2f = %.2f%n", a, operator, b,
result);
    }
}
```

5.7 Integer division og modulo

Når du arbejder med heltal, skal du være opmærksom på integer division:

```
int a = 7;
int b = 3;

int result1 = a / b;          // 2 (ikke 2.33...)
double result2 = (double) a / b;  // 2.33...

int remainder = a % b;        // 1 (rest ved division)
```

Modulo-operatoren (`%`) giver resten ved division og er nyttig til mange ting:

- Tjekke om et tal er lige: `number % 2 == 0`
- Få sidste ciffer: `number % 10`
- Cyklisk gentagelse: `index % arrayLength`

Øvelse: Mængderabat

Lav et program der beregner mængderabat:

1. Læs antal varer og pris per vare

2. Beregn totalprisen
3. Giv rabat baseret på antal:
   - 10+ varer: 10% rabat
   - 50+ varer: 20% rabat
   - 100+ varer: 25% rabat
4. Vis både original pris, rabat og slutpris

5.8 nextLine() vs next()

Vær opmærksom på forskellen mellem `nextLine()` og `next()`:

```
Scanner scanner = new Scanner(System.in);

// nextLine() læser hele linjen inkl. mellemrum
System.out.print("Indtast dit fulde navn: ");
String fullName = scanner.nextLine();  // "Anna Marie Hansen"

// next() læser kun næste ord
System.out.print("Indtast dit fornavn: ");
String firstName = scanner.next();     // "Anna" (stopper ved mellemrum)
```

5.8.1 Blanding af nextInt() og nextLine()

Pas på når du blander `nextInt()` og `nextLine()`:

```
Scanner scanner = new Scanner(System.in);

System.out.print("Indtast din alder: ");
int age = scanner.nextInt();

// PROBLEM: nextInt() efterlader newline i bufferen
System.out.print("Indtast dit navn: ");
String name = scanner.nextLine();  // Læser tom streng!

// LØSNING: Tilføj ekstra nextLine()
scanner.nextLine();  // "Spis" den resterende newline
System.out.print("Indtast dit navn: ");
String name = scanner.nextLine();
```

5.9 Sammenfatning

I dette kapitel har vi lært:

- **Casting**: Konvertering mellem datatyper (implicit og eksplicit)
- **Parsing**: Konvertering fra String til andre typer
- **Scanner-klassen**: Læsning af input fra brugeren eller strings
- **System.in og System.out**: Javas input/output streams
- **Interaktive programmer**: Kombination af input, beregninger og output
- **Fejlhåndtering**: Hvad der kan gå galt med input

Med disse værktøjer kan du nu lave programmer der interagerer med brugeren og reagerer på deres input. I næste kapitel lærer vi mere om betingelser (`if`/`else`), som gør det muligt at lave endnu mere intelligente programmer der træffer beslutninger baseret på input.

Note: Husk altid at lukke din Scanner når du er færdig: `scanner.close()`. I simple programmer er det ikke kritisk, men det er god praksis.

6 Betingelser og Logik

Indtil nu har vores programmer udført det samme hver gang vi kørte dem. Men programmer bliver først virkelig kraftfulde når de kan træffe beslutninger og reagere forskelligt afhængigt af situationen. Det er her betingelser kommer ind i billedet.

Betingelser gør det muligt for programmer at:

- Reagere på brugerinput
- Håndtere forskellige scenarier
- Træffe "intelligente" beslutninger
- Undgå fejl ved at tjekke værdier først

6.1 If-statements - grundlæggende beslutninger

Det mest grundlæggende værktøj til beslutninger er `if`-statements. Tænk på det som "hvis... så...":

```
boolean isWeekend = true;

if (isWeekend) {
    System.out.println("Jeg ser Netflix!");
}
```

Hvis betingelsen i parenteserne er `true`, udføres koden i de krøllede parenteser. Hvis den er `false`, springes blokken over.

6.1.1 If-else statements

Ofte vil vi gøre noget andet, hvis betingelsen ikke er opfyldt:

```
public class WeekendRoutine {
    public static void main(String[] args) {
        boolean isWeekend = false;

        wakeUp();

        if (isWeekend) {
            watchNetflix();
        } else {
            commute();
            work();
```

```
            commute();
        }

        goToBed();
    }

    public static void wakeUp() {
        System.out.println("Jeg vågner op");
    }

    public static void watchNetflix() {
        System.out.println("Jeg ser Netflix");
    }

    public static void work() {
        System.out.println("Jeg arbejder");
    }

    public static void commute() {
        System.out.println("Jeg pendler");
    }

    public static void goToBed() {
        System.out.println("Jeg går i seng");
    }
}
```

Læg mærke til at `wakeUp()` og `goToBed()` altid udføres, uanset om det er weekend eller ej.

Øvelse: Porto beregner

Lav en portoberegner for PostNord:

- Op til 1 kg: 60 kr
- Op til 2 kg: 65 kr
- Op til 5 kg: 70 kr
- Op til 10 kg: 90 kr
- Op til 20 kg: 160 kr

1. Start med hardcoded vægte og test forskellige scenarier
2. Udvid så brugeren kan indtaste vægten via Scanner
3. Håndter vægte over 20 kg med en fejlbesked

6.2 Else-if - flere betingelser

Når vi har flere mulige scenarier, bruger vi `else if`:

```
int temperature = 35;
boolean isWeekend = true;

wakeUp();

if (temperature > 40) {
    goToBeach();  // Varmefri!
} else if (isWeekend) {
    watchNetflix();
} else {
    commute();
    work();
    commute();
}

goToBed();
```

Java tjekker betingelserne i rækkefølge og udfører den første der er `true`. Resten springes over.

6.3 Indlejrede betingelser

Vi kan have if-statements inden i andre if-statements:

```
boolean isWeekend = true;
int temperature = 31;
boolean isRaining = false;
boolean isSick = false;

wakeUp();

if (isWeekend) {
    if (temperature > 30) {
        if (isRaining) {
            watchNetflix();
        } else {
            if (isSick) {
                watchNetflix();
            } else {
                goToBeach();
            }
        }
    } else {
        watchNetflix();
    }
} else {
    commute();
    work();
    commute();
```

```
}

goToBed();
```

Note: Mange indlejrede if-statements kan gøre koden svær at læse. Vi lærer snart bedre måder at organisere kompleks logik.

6.4 Sammenligningsoperatorer

For at lave betingelser bruger vi sammenligningsoperatorer:

```
int age = 20;
double price = 99.99;
String name = "Anna";

// Lighed og ulighed
boolean isAdult = (age == 18);         // false
boolean isNotChild = (age ≠ 5);        // true

// Numeriske sammenligninger
boolean canDrive = (age ⩾ 18);         // true
boolean isExpensive = (price > 100);   // false
boolean isCheap = (price ⩽ 50);        // false

// String sammenligning
boolean isAnna = name.equals("Anna"); // true - VIGTIGT!
boolean isNotBob = !name.equals("Bob"); // true
```

Note: Brug altid `.equals()` til at sammenligne strings, ikke `==`. Operatoren `==` sammenligner kun referencer, ikke indhold.

6.5 Logiske operatorer

Vi kan kombinere flere betingelser med logiske operatorer:

6.5.1 AND-operator (`&&`)

Begge betingelser skal være sande:

```
boolean canVote = (age ⩾ 18) && (hasCitizenship);
boolean isWorkDay = !isWeekend && !isHoliday;
```

6.5.2 OR-operator (`||`)

Mindst én betingelse skal være sand:

```
boolean needsUmbrella = isRaining || isSnowing;
boolean getDiscount = isMember || hasValidCoupon;
```

6.5.3 NOT-operator (`!`)

Vender en boolean værdi om:

```
boolean isNotWeekend = !isWeekend;
boolean isAwake = !isSleeping;
```

6.5.4 Komplekse udtryk

Vi kan kombinere flere operatorer:

```
boolean goToBeach = isWeekend && (temperature > 25) && !isRaining && !
isSick;
```

Brug parenteser til at gøre rækkefølgen tydelig!

6.6 Forbedring af læsbarhed

Komplekse betingelser kan gøres mere læselige på flere måder:

6.6.1 Navngivne variabler

```
// Svært at læse
if (isWeekend && (isRaining || isSick || temperature ⩽ 30)) {
    watchNetflix();
}

// Lettere at læse
boolean stayInside = isRaining || isSick || temperature ⩽ 30;
if (isWeekend && stayInside) {
    watchNetflix();
}
```

6.6.2 Metoder der returnerer boolean

```
public static boolean isBeachWeather(int temp, boolean isRaining) {
    return temp > 30 && !isRaining;
}

// I main metoden
if (isWeekend && isBeachWeather(temperature, isRaining)) {
    goToBeach();
}
```

6.6.3 Guard clauses

I stedet for indlejrede if-statements kan vi bruge "guards":

```
public static boolean isBeachWeather(int temp, boolean isRaining) {
    if (temp ⩽ 30) {
        return false;  // For koldt
    }

    if (isRaining) {
        return false;  // Det regner
    }

    return true;  // Ellers er det strandvejr
}
```

Øvelse: Enten eller

Implementer følgende metoder med passende betingelser:

1. ```
   public static boolean isTeenager(int age) {
       // Returnér true hvis alderen er mellem 13 og 19
   }
   ```

2. ```
   public static boolean canDrive(int age, boolean hasLicense) {
       // Returnér true hvis personen er 18+ og har kørekort
   }
   ```

3. ```
   public static String seasonForMonth(int month) {
       // Returnér årstiden for måneden (1-12)
   }
   ```

4.
   ```

```java
public static boolean isBetween(int number, int min, int max) {
    // Returnér true hvis number er mellem min og max (inklusiv)
}
```

6.7 Switch-statements

Når vi har mange specifikke værdier at tjekke, kan `switch` være mere læseligt end mange `else if`:

```java
public static String getDayOfWeek(int dayNumber) {
    switch (dayNumber) {
        case 1:
            return "Mandag";
        case 2:
            return "Tirsdag";
        case 3:
            return "Onsdag";
        case 4:
            return "Torsdag";
        case 5:
            return "Fredag";
        case 6:
            return "Lørdag";
        case 7:
            return "Søndag";
        default:
            return "Ugyldig dag";
    }
}
```

6.7.1 Switch med break

Hvis vi ikke bruger `return`, skal vi huske `break`:

```java
public static boolean isWeekend(int dayOfWeek) {
    boolean weekend = false;

    switch (dayOfWeek) {
        case 6:  // Lørdag
        case 7:  // Søndag
            weekend = true;
            break;
        case 1:  // Mandag
```

```
        case 2:  // Tirsdag
        case 3:  // Onsdag
        case 4:  // Torsdag
        case 5:  // Fredag
            weekend = false;
            break;
        default:
            System.out.println("Ugyldig ugedag!");
    }

    return weekend;
}
```

6.7.2 Switch med Strings

Switch virker også med strings:

```
public static int getSeasonNumber(String season) {
    switch (season.toLowerCase()) {
        case "vinter":
            return 1;
        case "forår":
            return 2;
        case "sommer":
            return 3;
        case "efterår":
            return 4;
        default:
            return -1;  // Ugyldig årstid
    }
}
```

6.8 Ternary operator

For simple if-else kan vi bruge den kompakte ternary operator:

```
// I stedet for:
String message;
if (age ⩾ 18) {
    message = "Du er myndig";
} else {
    message = "Du er ikke myndig";
}

// Kan vi skrive:
String message = (age ⩾ 18) ? "Du er myndig" : "Du er ikke myndig";
```

Syntaks: `betingelse ? værdi_hvis_sand : værdi_hvis_falsk`

Men brug det med måde - det kan hurtigt blive ulæseligt!

6.9 Almindelige fejl og faldgruber

6.9.1 Assignment vs. comparison

```
// FORKERT - tildeler værdi i stedet for at sammenligne
if (age = 18) {  // Kompileringsfejl

// RIGTIGT
if (age == 18) {
```

6.9.2 Floating point sammenligning

```
double price = 0.1 + 0.2;  // Ikke præcis 0.3!

// FORKERT
if (price == 0.3) {

// RIGTIGT
if (Math.abs(price - 0.3) < 0.00001) {
```

6.9.3 String sammenligning

```
String name1 = "Anna";
String name2 = scanner.nextLine();

// FORKERT
if (name1 == name2) {

// RIGTIGT
if (name1.equals(name2)) {
```

6.9.4 Null pointer

```
String name = null;

// FARLIGT - kan give NullPointerException
```

```java
if (name.equals("Anna")) {

// SIKKERT
if ("Anna".equals(name)) {  // eller
if (name != null && name.equals("Anna")) {
```

6.10 Avanceret eksempel

Lad os kombinere alt i et større eksempel:

```java
import java.util.Scanner;

public class WeatherAdvisor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Indtast temperatur (°C): ");
        int temperature = scanner.nextInt();

        System.out.print("Regner det? (true/false): ");
        boolean isRaining = scanner.nextBoolean();

        System.out.print("Er det weekend? (true/false): ");
        boolean isWeekend = scanner.nextBoolean();

        String advice = getWeatherAdvice(temperature, isRaining,
isWeekend);
        System.out.println("Anbefaling: " + advice);
    }

    public static String getWeatherAdvice(int temp, boolean raining,
boolean weekend) {
        if (!weekend) {
            return "Arbejdsdag - tag på arbejde uanset vejret!";
        }

        if (raining) {
            return "Det regner - bliv hjemme og hygge dig";
        }

        if (temp > 25) {
            return "Perfekt strandvejr - tag på stranden!";
        } else if (temp > 15) {
            return "Godt vejr til en gåtur i parken";
        } else if (temp > 5) {
            return "Tag en jakke på og gå en tur";
        } else {
            return "Koldt ude - måske læse en bog indendørs?";
```

```
        }
    }
}
```

Øvelse: BMI klassificering

Lav en forbedret BMI beregner:

1. Læs vægt og højde fra brugeren
2. Beregn BMI: `weight / (height * height)`
3. Klassificer resultatet:
   - Under 18.5: Undervægt
   - 18.5-24.9: Normal vægt
   - 25-29.9: Overvægt
   - 30+: Fedme
4. Giv passende sundhedsråd baseret på kategorien
5. Håndter ugyldige input (negative værdier)

6.11 Sammenfatning

Betingelser og logik er fundamentale for at skabe intelligente programmer. Vi har lært:

- **If-statements**: Grundlæggende beslutninger med `if`, `else`, og `else if`
- **Sammenligningsoperatorer**: `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logiske operatorer**: `&&` (og), `||` (eller), `!` (ikke)
- **Switch-statements**: Effektiv håndtering af mange specifikke værdier
- **Læsbarhed**: Navngivning, guards, og opdeling i metoder
- **Almindelige fejl**: String sammenligning, null values, og floating point

Med disse værktøjer kan du nu skrive programmer der reagerer intelligent på forskellige situationer og brugerinput. I næste kapitel lærer vi om løkker, som gør det muligt at gentage kode baseret på betingelser.

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

Indtil nu har vi arbejdet med variable der gemmer en enkelt værdi ad gangen. Men ofte har vi brug for at håndtere mange værdier af samme type - for eksempel temperature for alle ugens dage, navne på alle elever i klassen, eller point for alle spillere på et hold.

Arrays giver os mulighed for at gemme flere værdier af samme type i én enkelt variabel, og får dermed kraftfulde værktøjer til at organisere og manipulere data.

8.1 Hvad er et Array?

Et array er en samling af værdier af samme datatype, arrangeret i en fast rækkefølge. Tænk på det som en række af kasser, hvor hver kasse kan rumme en værdi:

```
// En enkelt værdi
int coins = 50;

// Et array med flere værdier
int[] chests = new int[4];  // Fire skattekister
```

Hvert element i arrayet har et unikt **indeks** (position), som starter fra 0. Det første element er ved indeks 0, det andet ved indeks 1, og så videre.

8.2 Array Deklaration og Initialisering

Der er flere måder at oprette arrays på:

8.2.1 Deklaration med `new`

```
// Opretter et array med 4 pladser (alle med default værdi 0)
int[] numbers = new int[4];

// Opretter et array med 10 pladser til strings (alle med default værdi
null)
String[] names = new String[10];

// Opretter et array med 5 pladser til booleans (alle med default værdi
false)
boolean[] flags = new boolean[5];
```

8.2.2 Initialisering med værdier

```
// Opretter og initialiserer direkte
int[] scores = {85, 92, 78, 94, 88};

// Alternativ syntaks
int[] temperatures = new int[]{20, 22, 19, 25, 23, 21, 18};

// Array med strings
String[] weekdays = {"Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag"};
```

Note: Når vi bruger `{}` til initialisering, bestemmer Java automatisk størrelsen baseret på antallet af elementer.

8.3 Adgang til Array Elementer

Vi bruger indekset i firkantede parenteser til at få adgang til elementer:

```
int[] chests = {50, 0, 0, 10};

// Læs værdier
int firstChest = chests[0];    // 50
int lastChest = chests[3];     // 10

// Ændre værdier
chests[1] = 25;                // Sætter anden kasse til 25
chests[0] -= 20;               // Fjerner 20 fra første kasse (nu 30)

System.out.println("Første kasse: " + chests[0]);  // 30
System.out.println("Anden kasse: " + chests[1]);   // 25
```

8.3.1 Array længde

Hvert array har en `length` egenskab der fortæller hvor mange elementer det indeholder:

```
int[] numbers = {10, 20, 30, 40, 50};
int size = numbers.length;  // 5

// Adgang til sidste element (uden at kende størrelsen på forhånd)
int lastElement = numbers[numbers.length - 1];  // 50
```

Øvelse: Array basics

1. Hvilken af følgende er korrekt syntaks for at deklarere et array af 10 heltal?

```
int a[10] = new int[10];      // A
int[10] a = new int[10];      // B
[]int a = [10]int;            // C
int a = new int[10];          // D
int[] a = new int[10];        // E
```

2. Opret et array kaldet `data` med fem elementer og følgende indhold: 27, 51, 33, −1, 101

3. Hvad indeholder arrayet `numbers` efter følgende kode?

```
int[] numbers = new int[8];
numbers[1] = 4;
numbers[4] = 99;
numbers[7] = 2;
int x = numbers[1];
numbers[x] = 44;
numbers[numbers[7]] = 11;
```

8.4 Hvorfor starter indekset ved 0?

Det kan virke underligt at det første element er ved indeks 0 i stedet for 1. Årsagen er at indekset faktisk repræsenterer en **forskydning** (offset) fra starten af arrayet:

- Indeks 0: 0 positioner fra start (det første element)
- Indeks 1: 1 position fra start (det andet element)
- Indeks 2: 2 positioner fra start (det tredje element)

Denne konvention kommer fra ældre programmeringssprog og er blevet standard i de fleste moderne sprog.

8.5 Array Gennemløb med Løkker

Arrays og løkker passer perfekt sammen. Her er forskellige måder at gennemgå alle elementer:

8.5.1 For-løkke med indeks

```
int[] scores = {85, 92, 78, 94, 88};

// Udskriv alle scores
for (int i = 0; i < scores.length; i++) {
    System.out.println("Score " + (i + 1) + ": " + scores[i]);
```

```
}

// Beregn gennemsnit
int sum = 0;
for (int i = 0; i < scores.length; i++) {
    sum += scores[i];
}
double average = (double) sum / scores.length;
System.out.println("Gennemsnit: " + average);
```

8.5.2 Enhanced for-løkke (for-each)

Når vi ikke behøver indekset, kan vi bruge den mere læselige for-each løkke:

```
int[] scores = {85, 92, 78, 94, 88};

// Udskriv alle scores
for (int score : scores) {
    System.out.println("Score: " + score);
}

// Find højeste score
int highest = scores[0];
for (int score : scores) {
    if (score > highest) {
        highest = score;
    }
}
System.out.println("Højeste score: " + highest);
```

8.6 Praktisk Eksempel: Parkeringshus

Lad os bygge et system til at håndtere et parkeringshus med 10 pladser:

```
public class ParkingHouse {
    private boolean[] spots;  // true = optaget, false = ledig
    private long[] timeOccupied;  // Tidspunkt for hvornår plads blev
optaget

    public ParkingHouse(int numberOfSpots) {
        spots = new boolean[numberOfSpots];
        timeOccupied = new long[numberOfSpots];

        // Alle pladser starter som ledige
        for (int i = 0; i < spots.length; i++) {
            spots[i] = false;
```

```
            timeOccupied[i] = 0;
        }
    }

    public boolean parkCar(int spotNumber) {
        if (spotNumber < 0 || spotNumber ⩾ spots.length) {
            System.out.println("Ugyldig plads nummer!");
            return false;
        }

        if (spots[spotNumber]) {
            System.out.println("Plads " + (spotNumber + 1) + " er
allerede optaget!");
            return false;
        }

        spots[spotNumber] = true;
        timeOccupied[spotNumber] = System.currentTimeMillis();
        System.out.println("Bil parkeret på plads " + (spotNumber + 1));
        return true;
    }

    public boolean leaveParkingSpot(int spotNumber) {
        if (spotNumber < 0 || spotNumber ⩾ spots.length) {
            System.out.println("Ugyldig plads nummer!");
            return false;
        }

        if (!spots[spotNumber]) {
            System.out.println("Plads " + (spotNumber + 1) + " er
allerede ledig!");
            return false;
        }

        spots[spotNumber] = false;
        timeOccupied[spotNumber] = 0;
        System.out.println("Bil forlod plads " + (spotNumber + 1));
        return true;
    }

    public void printStatus() {
        System.out.println("Parkeringshus status:");
        for (int i = 0; i < spots.length; i++) {
            String status = spots[i] ? "Optaget" : "Ledig";
            System.out.println("Plads " + (i + 1) + ": " + status);
        }
    }

    public int countAvailableSpots() {
        int available = 0;
        for (boolean spot : spots) {
```

```
            if (!spot) {
                available++;
            }
        }
        return available;
    }

    public void checkOvertime() {
        long currentTime = System.currentTimeMillis();
        long twoHoursInMillis = 2 * 60 * 60 * 1000;  // 2 timer

        System.out.println("Pladser optaget i over 2 timer:");
        for (int i = 0; i < spots.length; i++) {
            if (spots[i] && (currentTime - timeOccupied[i]) >
twoHoursInMillis) {
                System.out.println("Plads " + (i + 1) + " - Bøde
påkrævet!");
            }
        }
    }
}
```

Øvelse: Parkeringshus simulation

Brug `ParkingHouse` klassen til at:

1. Opret et parkeringshus med 10 pladser
2. Parker biler på plads 2, 5, og 8
3. Udskriv status for parkeringshuset
4. Udskriv antallet af ledige pladser
5. Prøv at parkere på en allerede optaget plads
6. Lad en bil forlade plads 5
7. Udskriv status igen

8.7 String Arrays og Tekstbehandling

Arrays af strings er meget nyttige til at håndtere tekstdata:

```
public class TextAnalyzer {
    public static void main(String[] args) {
        String[] weekdays = {"Mandag", "Tirsdag", "Onsdag", "Torsdag",
"Fredag"};

        // Find den længste ugedag
        String longest = weekdays[0];
        for (String day : weekdays) {
```

```
            if (day.length() > longest.length()) {
                longest = day;
            }
        }
        System.out.println("Længste ugedag: " + longest);

        // Tæl hvor mange dage der indeholder bogstavet 'a'
        int countWithA = 0;
        for (String day : weekdays) {
            if (day.toLowerCase().contains("a")) {
                countWithA++;
            }
        }
        System.out.println("Antal dage med 'a': " + countWithA);
    }
}
```

8.8 Tilfældig Pizza Generator

Lad os bruge arrays til at bygge en tilfældig pizza generator:

```
import java.util.Random;

public class PizzaGenerator {
    public static void main(String[] args) {
        String[] bases = {"tynd", "tyk", "fuldkorn"};
        String[] toppings = {"ost", "pepperoni", "ananas", "champignon",
"løg", "paprika"};
        String[] sauces = {"tomat", "hvidløg", "BBQ", "pesto"};

        Random random = new Random();

        // Generer tilfældig pizza
        String base = bases[random.nextInt(bases.length)];
        String sauce = sauces[random.nextInt(sauces.length)];

        // Vælg 2-4 tilfældige toppings
        int numToppings = 2 + random.nextInt(3);  // 2, 3 eller 4
        String[] selectedToppings = new String[numToppings];

        for (int i = 0; i < numToppings; i++) {
            String topping;
            boolean alreadySelected;

            // Sørg for at vi ikke vælger samme topping to gange
            do {
                topping = toppings[random.nextInt(toppings.length)];
                alreadySelected = false;
```

```
                for (int j = 0; j < i; j++) {
                    if (selectedToppings[j].equals(topping)) {
                        alreadySelected = true;
                        break;
                    }
                }
            } while (alreadySelected);

            selectedToppings[i] = topping;
        }

        // Udskriv opskrift
        System.out.println("=== Tilfældig Pizza Opskrift ===");
        System.out.println("Bund: " + base);
        System.out.println("Sovs: " + sauce);
        System.out.print("Toppings: ");
        for (int i = 0; i < selectedToppings.length; i++) {
            System.out.print(selectedToppings[i]);
            if (i < selectedToppings.length - 1) {
                System.out.print(", ");
            }
        }
        System.out.println();
    }
}
```

8.9 To-dimensionelle Arrays

Nogle gange har vi brug for at organisere data i både rækker og kolonner, som en tabel eller et gitter. Til det bruger vi to-dimensionelle arrays:

```
// Opret et 3x3 gitter (3 rækker, 3 kolonner)
int[][] grid = new int[3][3];

// Alternativt med værdier
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

// Adgang til elementer
grid[0][0] = 10;      // Øverste venstre hjørne
grid[2][2] = 99;      // Nederste højre hjørne
int value = matrix[1][2];  // Værdi 6 (række 1, kolonne 2)
```

8.9.1 Kryds og Bolle Spil

Lad os implementere et simpelt kryds og bolle spil:

```
public class TicTacToe {
    private int[][] board;
    private static final int EMPTY = 0;
    private static final int CROSS = 1;
    private static final int CIRCLE = -1;

    public TicTacToe() {
        board = new int[3][3];
        // Alle felter starter som tomme (0)
    }

    public boolean makeMove(int row, int col, int player) {
        if (row < 0 || row ⩾ 3 || col < 0 || col ⩾ 3) {
            return false;  // Ugyldigt felt
        }

        if (board[row][col] ≠ EMPTY) {
            return false;  // Feltet er allerede optaget
        }

        board[row][col] = player;
        return true;
    }

    public void printBoard() {
        System.out.println("Spillebræt:");
        for (int row = 0; row < 3; row++) {
            for (int col = 0; col < 3; col++) {
                char symbol;
                switch (board[row][col]) {
                    case CROSS:
                        symbol = 'X';
                        break;
                    case CIRCLE:
                        symbol = 'O';
                        break;
                    default:
                        symbol = '-';
                }
                System.out.print(symbol + " ");
            }
            System.out.println();
        }
    }

    public boolean checkWinner(int player) {
        // Tjek rækker
        for (int row = 0; row < 3; row++) {
```

```
            if (board[row][0] == player &&
                board[row][1] == player &&
                board[row][2] == player) {
                return true;
            }
        }

        // Tjek kolonner
        for (int col = 0; col < 3; col++) {
            if (board[0][col] == player &&
                board[1][col] == player &&
                board[2][col] == player) {
                return true;
            }
        }

        // Tjek diagonaler
        if (board[0][0] == player && board[1][1] == player && board[2][2]
== player) {
            return true;
        }
        if (board[0][2] == player && board[1][1] == player && board[2][0]
== player) {
            return true;
        }

        return false;
    }
}
```

8.10 Primitive vs Reference Datatyper

Nu hvor vi har lært om arrays, er det vigtigt at forstå forskellen mellem primitive og reference datatyper:

8.10.1 Primitive Datatyper

```
int age = 34;          // Gemmer værdien 34 direkte
double height = 1.75;  // Gemmer værdien 1.75 direkte
boolean isStudent = true;  // Gemmer værdien true direkte
```

Primitive datatyper gemmer selve værdien direkte i variablen.

8.10.2 Reference Datatyper

```
String name = "Alice";           // Gemmer reference til String objektet
int[] numbers = {1, 2, 3, 4, 5}; // Gemmer reference til array objektet
```

Reference datatyper gemmer ikke selve værdien, men en **reference** (adresse) til hvor værdien er gemt i hukommelsen.

8.10.3 Praktisk forskel

```
// Primitive datatyper - kopiering af værdi
int a = 10;
int b = a;    // b får en kopi af værdien 10
a = 20;       // Ændrer ikke b
System.out.println(b);  // Udskriver stadig 10

// Reference datatyper - kopiering af reference
int[] array1 = {1, 2, 3};
int[] array2 = array1;  // array2 peger på samme array som array1
array1[0] = 99;         // Ændrer arrayet
System.out.println(array2[0]);  // Udskriver 99 (samme array!)
```

Note: Når vi arbejder med arrays, skal vi være opmærksomme på at de er reference typer. Det betyder at hvis vi tildeler et array til en anden variabel, får vi ikke en kopi - begge variable peger på det samme array i hukommelsen.

8.11 ASCII og Unicode

Karakterer i Java repræsenteres som tal ifølge ASCII og Unicode standarder:

```
char c1 = 74;   // ASCII værdi for 'J'
char c2 = 65;   // ASCII værdi for 'A'
char c3 = 86;   // ASCII værdi for 'V'
char c4 = 65;   // ASCII værdi for 'A'

System.out.println("Jeg elsker " + c1 + c2 + c3 + c4);  // "Jeg elsker
JAVA"

// Vi kan også iterere gennem alfabetet
for (char c = 'A'; c ⩽ 'Z'; c++) {
    System.out.print(c);  // ABCDEFGHIJKLMNOPQRSTUVWXYZ
}
```

Øvelse: Football holdet

Implementer et system til Danmarks fodboldlandshold:

1. Opret et array med bruttotruppen (23 spillere)
2. Opret et array med startopstillingen (11 spillere) ved at vælge fra bruttotruppen
3. Organiser startopstillingen i en 4-3-3 formation:
   - Målmand (1 spiller)
   - Forsvar (4 spillere)
   - Midtbane (3 spillere)
   - Angreb (3 spillere)
4. Udskriv startopstillingen formateret som en tabel med trøje nummer, navn og position

8.12 Array Algoritmer

Her er nogle nyttige algoritmer til arrays:

8.12.1 Find minimum og maksimum

```
public static int findMin(int[] array) {
    if (array.length == 0) return Integer.MAX_VALUE;

    int min = array[0];
    for (int i = 1; i < array.length; i++) {
        if (array[i] < min) {
            min = array[i];
        }
    }
    return min;
}

public static int findMax(int[] array) {
    if (array.length == 0) return Integer.MIN_VALUE;

    int max = array[0];
    for (int value : array) {
        if (value > max) {
            max = value;
        }
    }
    return max;
}
```

8.12.2 Søgning

```
public static int findFirst(int[] array, int target) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == target) {
            return i;  // Returnér indeks
        }
    }
    return -1;  // Ikke fundet
}

public static boolean contains(int[] array, int target) {
    return findFirst(array, target) != -1;
}
```

8.12.3 Kopiering

```
public static int[] copyArray(int[] original) {
    int[] copy = new int[original.length];
    for (int i = 0; i < original.length; i++) {
        copy[i] = original[i];
    }
    return copy;
}
```

8.13 Almindelige Fejl og Faldgruber

8.13.1 Array Index Out of Bounds

```
int[] numbers = {1, 2, 3, 4, 5};

// FARLIGT - kan give ArrayIndexOutOfBoundsException
int value = numbers[5];  // Indeks 5 eksisterer ikke!

// SIKKERT - tjek grænser
if (index >= 0 && index < numbers.length) {
    int value = numbers[index];
}
```

8.13.2 Null Pointer Exception

```
int[] numbers = null;

// FARLIGT - giver NullPointerException
int length = numbers.length;

// SIKKERT - tjek for null først
if (numbers =/= null) {
    int length = numbers.length;
}
```

8.13.3 Reference vs Værdi forvirring

```
// Pas på med arrays som parametre!
public static void modifyArray(int[] arr) {
    arr[0] = 999;  // Ændrer det originale array!
}

int[] myNumbers = {1, 2, 3};
modifyArray(myNumbers);
System.out.println(myNumbers[0]);  // Udskriver 999, ikke 1!
```

8.14 Sammenfatning

Arrays er kraftfulde datastrukturer der gør det muligt at:

- **Gemme multiple værdier** af samme type i én variabel
- **Effektiv adgang** til elementer via indeks
- **Systematisk gennemløb** med løkker
- **Organisere data** i strukturer som tabeller og matricer

Vi har lært:

- Array deklaration og initialisering
- Indeksering (starter ved 0)
- Gennemløb med for-løkker og for-each
- To-dimensionelle arrays
- Forskellen mellem primitive og reference datatyper
- Praktiske eksempler og algoritmer
- Almindelige fejl og hvordan man undgår dem

Arrays danner grundlaget for mange avancerede datastrukturer og algoritmer, som vi vil møde senere i vores programmeringsrejse. I næste kapitel vil vi lære om objektorienteret programmering og hvordan vi kan skabe vores egne datatyper med klasser og objekter.