# Java for deg som kan Python (Del 3)

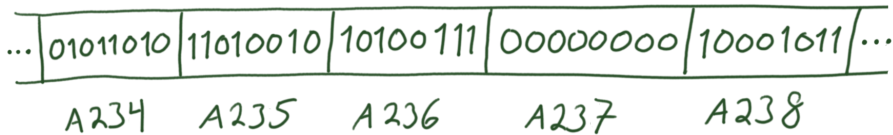
Leif Harald Karlsen

Universitetet i Oslo

22.01.24

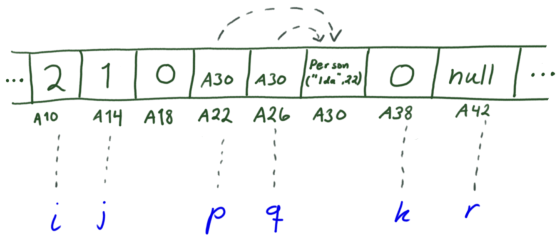


# Variabler i Java




- Minne på en datamaskin kan sees på som en lang serie med bits delt inn i bytes som hver har én addresse i minne
- Én slik addresse peker altså på én byte (8-bits)
- En variabel i Java er egentlig bare en slik adresse sammen med et antall etterfølgende bytes som hører til variabelen
- Typen til variabelen avgjør hvor mange bytes variabelen peker på
  - En `byte` er 8-bit og trenger bare 1 byte
  - En `int` eller `float` er 32-bit og trenger derfor 4 bytes
  - En `long` eller `double` er 64-bit og trenger derfor 8 bytes
- Et objekt krever like mange bytes som summen av plassen til dens (ikke-statiske) felter

# Verdier vs. referanser




- Varibler av basis-typene inneholder verdien direkte i minneadressen dens
- Variabler av objekter inneholder derimot en ny adresse som igjen peker på objektet dens
- Kun referanser kan være `null` (de har da en verdi som ikke er en gyldig adresse)

```
int i = 1;
int j = i;
i++;

System.out.println(i); // -> 2
System.out.println(j); // -> 1

Person p = new Person("Kari", 22);

Person q = p;
p.byttNavn("Ida");

System.out.println(p.hentNavn());
// -> "Ida"

System.out.println(q.hentNavn());
// -> "Ida"

int k;
Person r;
```

# == vs. equals

- Java har to måter å sjekke likhet på
- `a == b` sjekker at `a` har lik verdien som `b`
  - For primitive typer: sjekker at de har lik verdi
  - For referanser: sjekker at de peker på samme objekt
- `equals` er derimot en metode som kan tilpasses dens klasse
  - Ulike Java-objekter kan representerer det samme objektet i virkeligheten
  - F.eks. er to strenger like om de inneholder det samme
  - Alle klasser har automatisk en `equals` som er lik ==
  - Bør overskrives til å passe klassen (mer om dette senere)

```
int x = 1;
int y = 1;
int z = 2;

System.out.println(x == y);
// -> true
System.out.println(x == z);
// -> false

String x = "Hei";
System.out.println(x == "Hei");
// -> false

System.out.println(x.equals("Hei"));
// -> true

Person p1 = new Person("Kari", 22);
Person p2 = new Person("Kari", 22);

System.out.println(p1 == p2);
// -> false
```

# Hjelpeklasser for primitive typer

- Java har en egen klasse for hver primitive type
  - F.eks. `Integer` for `int`
- Disse er nyttige når man ønsker en peker til en verdi av en primitiv type
- Husk: Bruk `equals` for likhet
- Konstruktøren er ikke anbefalt å bruke
- Bruk heller den statiske `valueOf`-metoden
- Disse klassene har også mange statiske metoder som er nyttige

```
Integer i = new Integer(1);
Integer j = new Integer(1);

System.out.println(i == j);
// -> false

System.out.println(i.equals(j));
// -> true

// Bedre å bruke valueOf-metoden
Integer x = Integer.valueOf(1);

// Har også Double, Boolean, osv.
Double d = Double.valueOf(1.2);
Boolean d = Boolean.valueOf(true);

// Nyttige metoder
int a = Integer.parseInt("123");
int b = Integer.max(3, 7);
int c = Integer.min(a, b);
```

# Pakker og import

- Alle klasser ligger i en pakke
- `package`: spesifiserer pakken til en klasse
- Dersom man ikke spesifiserer noe blir pakken uten navn
- `import`: importerer en pakke
- Trenger ikke importere klasser i samme pakke
- Pakkenavn har ofte flere deler separert med punktum
- Siste delen er klasse-navnet
- Delene før er mappe-navn

```
package uio.ifi.in1010;

import java.time.LocalDate;
// Kan bruke * for å importere
// alle klasser i en pakke
import java.util.*;

public class Person {
  private String navn;
  private LocalDate fodtDato;

  ...
}
// Person har nå fullt navn:
// uio.ifi.in1010.Person
```

# Scanner

- `Scanner` er en klasse som kan lese tekst utenfra inn i et Java-program
- Kan brukes til å hente brukerinput eller lese fra fil
- Tar en input-kilde som argument til konstruktøren
- `System.in` er brukerinput

```
import java.util.Scanner;

public class ScannerTest {
  public static void main(String[] args) {

    Scanner s = new Scanner(System.in);

    // Henter en int fra brukeren
    int i = s.nextInt();
    System.out.println("Du skrev: " + i);

    // Henter ut en tekstlinje fra brukeren
    String t = s.nextLine();
    System.out.println("Du skrev: " + t);
  }
}
```

# Kalendereksempel: Brukerinput via Scanner

## (Viser bare kode som er lagt til)

```
import java.util.Scanner;

public class KalenderMain {
  public static void main(String[] args) {

    Kalender kalender = new Kalender(args[0]); // args[0] er første kommandolinjearg.
    kalender.skrivUtDagensHendelser();
    System.out.println();
    Scanner scanner = new Scanner(System.in);

    while (true) {
      System.out.println("-- Meny: --");
      System.out.println("  0: Gå ut av programmet.");
      System.out.println("  1: Sett inn ny hendelse.");
      System.out.print("Valg: ");
      int valg = scanner.nextInt();
      scanner.nextLine(); // Consumerer newline etter int'en

      if (valg == 0) {
        break; // Avslutter løkken
      } else if (valg == 1) {
        settInnNyHendelse(kalender, scanner);
      } else {
        System.out.println("Vennligst velg et tall fra menyen.");
      }
    }
    // Fortsetter på neste slide
```

# Kalendereksempel: Brukerinput via Scanner (forts.)

```
  private static void settInnNyHendelse(Kalender kalender, Scanner scanner) {

    System.out.print("Dato: ");
    String dato = scanner.nextLine();

    System.out.print("Klokkeslett: ");
    String klokkeslett = scanner.nextLine();

    System.out.print("Beskrivelse: ");
    String beskrivelse = scanner.nextLine();

    Hendelse hendelse;

    if (dato.equals("")) {
      hendelse = new Hendelse(klokkeslett, beskrivelse);
    } else {
      hendelse = new Hendelse(dato, klokkeslett, beskrivelse);
    }
    kalender.settInnHendelse(hendelse);
    System.out.println("Ny hendelse satt inn.");
  }
}
```

# Skrive til fil

- Mange måter å skrive til fil (ulike klasser)
- En enkel måte er `PrintWriter`
- Tar filnavn i konstruktøren
- Bruker `println` for å skrive
- Må bruke `close()` når man er ferdig for å lukke filen

```
import java.io.PrintWriter;

public class SkriveTilFil {
  public static void main(String[] args) {

    File fil = new File("utfil.txt");
    PrintWriter pw = new PrintWriter(fil));
    pw.println("Her er en linje.");
    pw.println("Her er annen linje.");
    pw.close();
  }
}
```

# Kalendereksempel: Lagre kalenderen

```
import java.io.PrintWriter;

public class Kalender {

  private static String FILENDELSE = ".txt";
  //...

  public void skrivHendelserTilFil() { // legg til et kall til denne i slutten av main()

    File fil = new File(this.navn + FILENDELSE);
    fil.delete(); // slett filen dersom den finnes
    PrintWriter skriver = new PrintWriter(fil);

    int i = 0;
    while (i < this.hendelser.length && this.hendelser[i] != null) {
      Hendelse h = this.hendelser[i];
      skriver.println(h.utskriftForFil()); // se neste slide
      i++;
    }
    skriver.close();
  }
}
```

# Kalendereksempel: Lagre kalenderen (forts.)

```
public class Hendelse {

  private static String DELER = ";";
  //...

  public String utskriftForFil() {
    return this.dato + DELER + this.klokkeslett + DELER + this.beskrivelse;
  }
}
```

# Exceptions og Try/Catch

- I ulike tilfeller kan ting feile
- F.eks. skrive til fil vi ikke har rettigheter til
- Da blir det kastet en feil, `Exception`
- Disse er objekter av ulike `Exception`-klasser
- Håndteres med `try` og `catch`
- Alle `Exception`s har en melding
- Meldingen hentes med `getMessage`-metoden
- Må av og til håndtere `Exception`s
  - Kompilering feiler hvis ikke

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class SkriveTilFil {
  public static void main(String[] args) {

    try {

      File fil = new File("utfil.txt");
      PrintWriter pw = new PrintWriter(fil));
      pw.println("Her er en linje.");
      pw.println("Her er annen linje.");
      pw.close();

    } catch (FileNotFoundException ex) {

      System.out.println("File not found: "
            + ex.getMessage());
      System.exit(1); // Stopper programmet
    }
  }
}
```

# Throws

- Kan også velge å kaste feilen videre (til metodene som kaller denne)
- Kan enten kaste helt til brukeren
- Eller til en metode høyere oppe (som så fanger den med `try`/`catch`)
- Burde håndtere feilen (`try`/`catch`) der man vet hvordan
- Legger da til `throws` på metoden slik:

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class SkriveTilFil {
  public static void main(String[] args) throws FileNotFoundException {

    File fil = new File("utfil.txt");
    PrintWriter pw = new PrintWriter(fil));
    pw.println("Her er en linje.");
    pw.println("Her er annen linje.");
    pw.close();
  }
}
```

# Lese fra fil

- Mange måter å lese fra fil (mange forskjellige klasser)
- Vi bruker `Scanner` og et `File`-objekt i konstruktøren
- Et `File`-objekt representerer en fil
- `File` tar en streng-sti som argument til konstruktøren

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class SkriveTilFil {
  public static void main(String[] args)
            throws FileNotFoundException {

    File f = new File("utfil.txt");
    Scanner s = new Scanner(f);

    while (s.hasNextLine()) {
      String linje = s.nextLine();
      System.out.println(linje);
    }
    s.close();
  }
}
```

# Kalender-eksempel: Lese fra fil

```
//...
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class Kalender {
  //...
  public Kalender(String navn, int antall) {
    this.hendelser = new Hendelse[antall];
    this.navn = navn;
    lesHendelserFraFil(); // Leser inn hendelser
  }
  //...
  private void lesHendelserFraFil() {

    try {
      File fil = new File(this.navn + FILENDELSE);
      fil.createNewFile(); // lager filen dersom den ikke finnes
      Scanner leser = new Scanner(fil);

      while (leser.hasNextLine()) {
        String linje = leser.nextLine();
        Hendelse hendelse = Hendelse.fraFil(linje); // neste slide
        settInnHendelse(hendelse);
      }
    } catch (IOException ex) {
      System.out.println("Feil ved lesing fra fil: " + ex.getMessage());
      System.exit(1);
    }
  }
}
```

# Kalender-eksempel: Lese fra fil (forts.)

```
public class Hendelse {
  //...

  public static Hendelse fraFil(String linje) {

    String[] deler = linje.split(DELER);
    return new Hendelse(deler[0], deler[1], deler[2]);
  }
}
```

# Forslag til ekstraoppgaver
(av ulik vanskelighetsgrad)

Utvid programmet slik at brukeren kan:

- legge til et slutt-tidspunkt for hendelser
- få se alle hendelser på en gitt dato
- slette hendelser (og så endre på hendelser)
- lagre ubegrenset antall hendelser (etter Beholdere)
- bruke GUI for å intragere med programmet (etter GUI)
- jobbe med flere kalendere samtidig