EDAA20
Programmering och databaser

Föreläsning 3, **Metoder**, 2023-09-04, Niklas Fors

Läsvecka 2

	Måndag	Tisdag	Onsdag	Torsdag	Fredag
Programmering	Föreläsning	Föreläsning	Labb 3		Labb 3
Databaser		Databaslabb		Databaslabb	

Anmäl er till labbgrupp (länk finns i Moodle)

Se föreläsningsbilder/inspelad föreläsning 1
som förberedelser inför databaslabben

Metod

En ***metod*** är en bit kod som kan ***anropas*** många gånger

Vid anropet skickas värden in (indata)

- ***Argumenten*** överförs till metodens ***parametrar***

När metoden exekverat returneras oftast ett värde (utdata)

Vi har anropat metoder

Anrop av **vanlig metod** på objekt:

```
Scanner scan = new Scanner(System.in);
double x = scan.nextDouble();
```

Anrop av **statisk metod**:

```
double y = Math.sqrt(x);
```

```
public class Sum {
  public static void main(String[] args) {
    int sum5 = 0;
    for (int i = 1; i <= 5; i++) {
      sum5 = sum5 + i;
    }
    System.out.println("1 + 2 + ... + 5 = " + sum5);

    int sum20 = 0;
    for (int i = 1; i <= 20; i++) {
      sum20 = sum20 + i;
    }
    System.out.println("1 + 2 + ... + 20 = " + sum20);
  }
}
```

Programmet skriver ut:

```
1 + 2 + ... + 5 = 15
1 + 2 + ... + 20 = 210
```

```
public class Sum {
  public static void main(String[] args) {
    int sum5 = 0;
    for (int i = 1; i <= 5; i++) {
      sum5 = sum5 + i;
    }
    System.out.println("1 + 2 + ... + 5 = " + sum5);

    int sum20 = 0;
    for (int i = 1; i <= 20; i++) {
      sum20 = sum20 + i;
    }
    System.out.println("1 + 2 + ... + 20 = " + sum20);
  }
}
```

Ungefär samma kod, fast olika slutvillkor.

Kan man undvika duplicerad kod?

Programmet skriver ut:

```
1 + 2 + ... + 5 = 15
1 + 2 + ... + 20 = 210
```

```
public class Sum {
  public static void main(String[] args) {
    int sum5 = sumTo(5);
    int sum20 = sumTo(20);
    System.out.println("1 + 2 + ... + 5 = " + sum5);
    System.out.println("1 + 2 + ... + 20 = " + sum20);
  }

  public static int sumTo(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
      sum = sum + i;
    }
    return sum;
  }
}
```

Abstrahera koden till en ***metod*** **sumTo** med ***parametern*** **n**

Nu kan metoden anropas med olika argument för att beräkna olika summor!

```
public class Sum {
  public static void main(String[] args) {
    int sum5 = sumTo(5);
    int sum20 = sumTo(20);
    System.out.println("1 + 2 + ... + 5 = " + sum5);
    System.out.println("1 + 2 + ... + 20 = " + sum20);
  }

  public static int sumTo(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
      sum = sum + i;
    }
    return sum;
  }
}
```

Metodanrop med ***argumentet*** 5

Metod sumTo med **returtypen** int och **parametern** n av typen int

Parametern n används

Metoden **returnerar** värdet sum.
Värdet måste vara kompatibelt med **returtypen** (dvs, heltal)

Abstrahera koden till en ***metod*** **sumTo** med ***parametern*** **n**

Nu kan metoden anropas med olika argument för att beräkna olika summor!

Parametrar

- En metod har noll eller flera ***parametrar***
  - En parameter har en typ och ett namn
- Parametrarna är bara synliga i metoden
  - De tilldelas värden när metoden anropas
  - De försvinner när metoden exekverat färdigt

```
public static int sumTo(int n) {
  ...
}
```

Parameter

Returvärde

En metod har en ***returtyp***

```
public static int sumTo(int n) {
  ...
  return sum;
}
```

Metoden måste returnera (med `return`) ett värde
som är kompatibelt med returtypen.

Returtypen **`void`** kan användas om inget värde ska returneras

- Då behövs ingen `return`-sats

Parameteröverföring

Vid anrop ***överförs*** argumenten till parametrarna

- Argument 1 överförs till parameter 1, osv
- Typerna på argumenten måste vara kompatibla med parametertyperna

```
public class Sum {
  public static void main(String[] args) {
    int x = 5;
    int sum15 = sumTo(x+10);
    ...
  }

  public static int sumTo(int n) {
    ...
  }
}
```

Argumentet beräknas till 15

Värdet 15 överförs till ***parametern*** n, dvs, n=15

Övning 1

```
public class Asterisk {
  public static void main(String[] args) {
    printAsterisks(5);
  }

  public static void printAsterisks(int n) {



  }
}
```

Skriv färdigt metoden `printAsterisks` som ska skriva ut **n** antal asterisker (*)

Övning 1 – lösning

```
public class Asterisk {
  public static void main(String[] args) {
    printAsterisks(5);
  }

  public static void printAsterisks(int n) {
    for (int i = 1; i <= n; i++) {
      System.out.println("*");
    }
  }
}
```

Skriv färdigt metoden `printAsterisks` som ska skriva ut **n** antal asterisker (*)

Övning 2

```
public class Asterisk {
  public static void main(String[] args) {
    printAsterisks(5);
  }

  public static void printAsterisks(int n) {
    for (int i = 1; i <= n; i++) {
      System.out.println("*");
    }
  }
}
```

Ändra på metoden så att den skriver ut en *godtycklig* sträng **n** gånger. (Strängen ska anges som en parameter.)

Övning 2 - lösning

```
public class Asterisk {
  public static void main(String[] args) {
    printString(5, "-");
  }

  public static void printString(int n, String s) {
    for (int i = 1; i <= n; i++) {
      System.out.println(s);
    }
  }
}
```

Ändra på metoden så att den skriver ut en *godtycklig* sträng **n** gånger. (Strängen ska anges som en parameter.)

Bra att ändra på metodnamnet till ett mer beskrivande.

Modifierare

Synlighet:

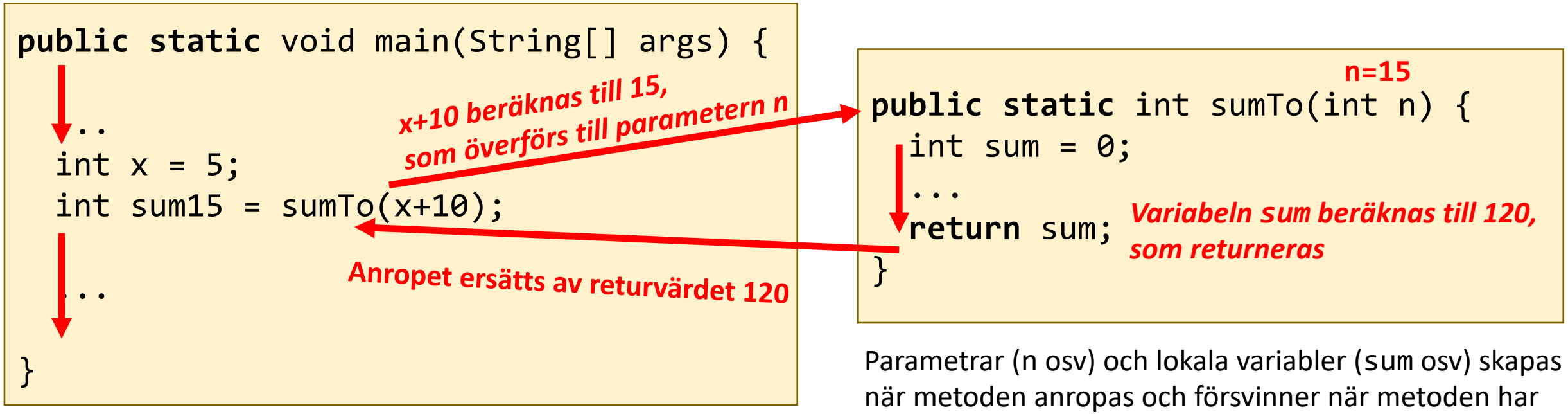
- `public` – Metoden kan även anropas utanför klassen (oftast i kursen)
- `private` – Metoden kan bara anropas inifrån klassen

```
public static int sumTo(int n) {
  ...
}
```

`static` – Metoden hör till klassen och anropas *inte* på objekt.
Exempel: `Math.round(…)`

Klassnamnet behövs (ex: `Math`) om man anropar den statiska metoden från en annan klass.

Exekvering av metodanrop



Ett anrop av en metod innebär att exekveringen fortsätter med den första satsen i den anropade metoden.

När den anropade metoden (sumTo) är klar återupptas exekveringen i den metod där anropet gjordes.

Parametrar (n osv) och lokala variabler (sum osv) skapas när metoden anropas och försvinner när metoden har exekverat färdigt.

Övning: nästlade anrop

Vad skriver följande program ut?

```
public class NestedCall {
  public static void main(String[] args) {
    System.out.println(f(f(2)));
  }

  public static int f(int x) {
    return x*2+1;
  }
}
```

Övning: nästlade anrop

Vad skriver följande program ut?

```
public class NestedCall {
  public static void main(String[] args) {
    System.out.println(f(f(2)));
  }

  public static int f(int x) {
    return x*2+1;
  }
}
```

Nästlade anrop exekveras inifrån och ut:

```
System.out.println(f(f(2)))  =>
System.out.println(f(5))     =>
System.out.println(11)
```

```
public class CircleProgram {
  public static void main(String[] args) {
    System.out.println("Ange radie: ");
    Scanner scan = new Scanner(System.in);
    double radius = scan.nextDouble();
    System.out.println("Area: " + calcArea(radius));
    System.out.println("Omkrets: " + calcCircumference(radius));
  }



  Hur ska metoderna calcArea och calcCircumference deklareras?



}
```

Ett program med metoder som läser in radien av en cirkel och beräknar cirkelns area och omkrets.

```
public class CircleProgram {
  public static void main(String[] args) {
    System.out.println("Ange radie: ");
    Scanner scan = new Scanner(System.in);
    double radius = scan.nextDouble();
    System.out.println("Area: " + calcArea(radius));
    System.out.println("Omkrets: " + calcCircumference(radius));
  }

  public static double calcArea(double radius) {
    return Math.PI * radius * radius;
  }

  public static double calcCircumference(double radius) {
    return 2 * Math.PI * radius;
  }
}
```

Ett program med metoder som läser in radien av en cirkel och beräknar cirkelns area och omkrets.

Metoden `main`

Metoden `main` är en särskild metod som anropas automatiskt av Java-systemet när programmet startar (startpunkten för programmet).

Java kräver att metoden är deklarerad på följande sätt:

```
public class CircleProgram {
  public static void main(String[] args) {
    ...
  }
}
```

(`args` är en vektor av strängar som anges när programmet startas, men som ej används i kursen)

Block

Block omges av klammerparenteser/måsvingar { } och är satser som hör ihop

```
public static void printAsterisks(int n) {
    ...  // Satserna i en metod
}

for (int i = 1; i <= n; i++) {
    ...  // Satserna som utförs i en for-sats
}
```

I ett block *indenterar* man koden ett steg (typiskt 4 mellanslag/1 tab).
Variabler deklarerade inuti ett block är inte synliga utanför blocket.

Kommentarer

Man kan kommentera sin Java-kod med /* */ eller //

```
/* En kommentar som kan sträcka sig över flera rader och som
   beskriver vad metoden printAsterisks gör */
public static void printAsterisks(int n) {
    ...  // Satserna i en metod
}

// En kommentar på en rad som beskriver for-satsen
for (int i = 1; i <= n; i++) {
    ...  // Satserna som utförs i en for-sats
}
```

Konstanter

I Klassen `Math` finns *konstanten* `Math.PI`:

```
public static final double PI = 3.14159265358979323846;
```

`final` betyder att värdet inte får ändras

Konvention: konstanter brukar enbart använda *VERSALER*

Övning – Implementera metoden `isEven`

```
public class EvenNumber {
  public static void main(String[] args) {
    System.out.println("Skriv ett heltal: ");
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    if (isEven(n)) {
      System.out.println(n + " är ett jämnt tal.");
    } else {
      System.out.println(n + " är ett udda tal.");
    }
  }

  public static boolean isEven(int n) {
    ...
  }
}
```

Ledtråd: använd operatorn %

```
public class EvenNumber {
  public static void main(String[] args) {
    System.out.println("Skriv ett heltal: ");
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    if (isEven(n)) {
      System.out.println(n + " är ett jämnt tal.");
    } else {
      System.out.println(n + " är ett udda tal.");
    }
  }

  public static boolean isEven(int n) {
    if (n % 2 == 0) {
      return true;
    } else {
      return false;
    }
  }
}
```

```
public class EvenNumber {
  public static void main(String[] args) {
    System.out.println("Skriv ett heltal: ");
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    if (isEven(n)) {
      System.out.println(n + " är ett jämnt tal.");
    } else {
      System.out.println(n + " är ett udda tal.");
    }
  }

  public static boolean isEven(int n) {
    return n % 2 == 0;
  }
}
```

Förenkling då det logiska uttrycket beräknas till ett logiskt värde (`true` eller `false`).

Då kan man returnera värdet direkt!

Logiska uttryck

Logiska uttryck används som villkor i `if`- och `while`-satser:

```
if (x < 5) {
  ...
}
while (true) {
  ...
}
```

Logiska uttryck beräknas till antingen `true` eller `false`.

Variabler av typen `boolean` kan tilldelas resultatet av logiska uttryck:

```
boolean ok = true;
ok = false;
ok = x < 5;
```

de Morgans lagar

- Antag att **A** och **B** är två logiska uttryck. Då gäller:

```
!(A && B)   ⇔   !A || !B
!(A || B)   ⇔   !A && !B
```

- Att **x** tillhör intervallet [5, 10] kan skrivas som:

```
x >= 5 && x <= 10
```

  Att **x** *inte* tillhör intervallet [5, 10] är negation till ovanstående:

```
!(x >= 5 && x <= 10)
```

  Som kan förenklas till:

```
x < 5 || x > 10
```

  (enligt regeln ovan)

Algoritm

En ***algoritm*** är en följd av instruktioner som beskriver hur man ska göra för att stegvis lösa ett problem.

Exempel:

- Matrecept
- Beskrivning (på svenska eller pseudokod) hur man löser ett problem
  - Hitta minsta talet bland många tal
  - Hitta talet x bland många tal
- Programkod

Minsta talet av två tal

```
public class Minimum {
  public static void main(String[] args) {
    System.out.println("Skriv två heltal: ");
    Scanner scan = new Scanner(System.in);
    int n1 = scan.nextInt();
    int n2 = scan.nextInt();
    int min;
    if (n1 < n2) {
      min = n1;
    } else {
      min = n2;
    }
    System.out.println("Minsta talet är " + min);
  }
}
```

Hur hanterar man godtyckligt många tal?

Algoritm: Hitta minsta tal

Algoritm: hitta minsta tal

Indata: en sekvens av tal
Utdata: minsta talet, *min*
min = <ett väldigt stort tal>
för varje tal *x* i sekvensen
    om *x* < *min*
        min = x

Algoritmen undersöker ett tal i taget och håller reda på det minsta talet hittills

Konstanter för minsta och största tal

```
int i1 = Integer.MIN_VALUE;     // -2147483648
int i2 = Integer.MAX_VALUE;     // 2147483647

double d1 = Double.MIN_VALUE;   // 4.9E-324 (OBS! minsta möjliga
                                //           positiva flyttal)
double d2 = Double.MAX_VALUE;   // 1.8E308
double d3 = -Double.MAX_VALUE; // -1.8E308
```

Med **E** menas "gånger 10 upphöjt till"

En implementation av hitta minsta tal

```
public class ComputeMin {
  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int min = Integer.MAX_VALUE;
    while (scan.hasNextInt()) {
      int x = scan.nextInt();
      if (x < min) {
        min = x;
      }
    }
    System.out.println("Minsta talet: " + min);
  }
}
```

Metoden `hasNextInt` returnerar `true` om det finns ett heltal att läsa in.

Inmatningen kan stoppas genom att användaren skriver in något annat än ett heltal (exempelvis en bokstav).