# F4 – Klasser och Metoder

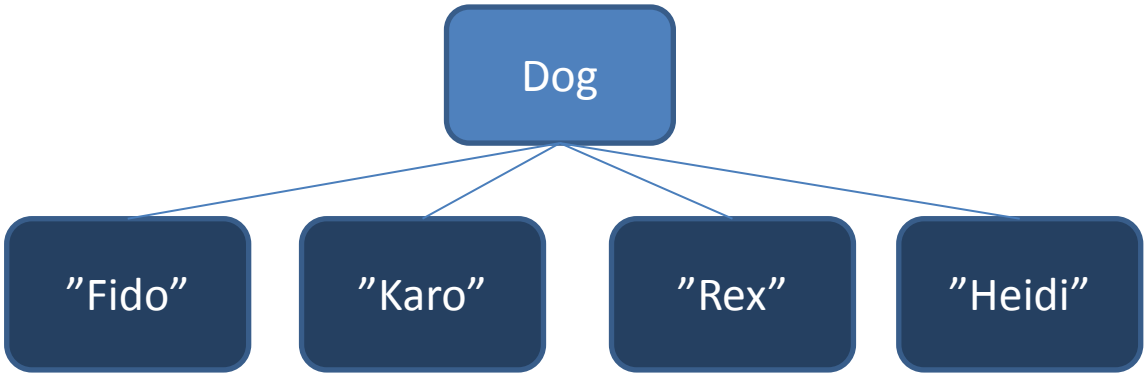
ID1004 Objektorienterad programmering

Fredrik Kilander fki@kth.se

# Klasser och objekt

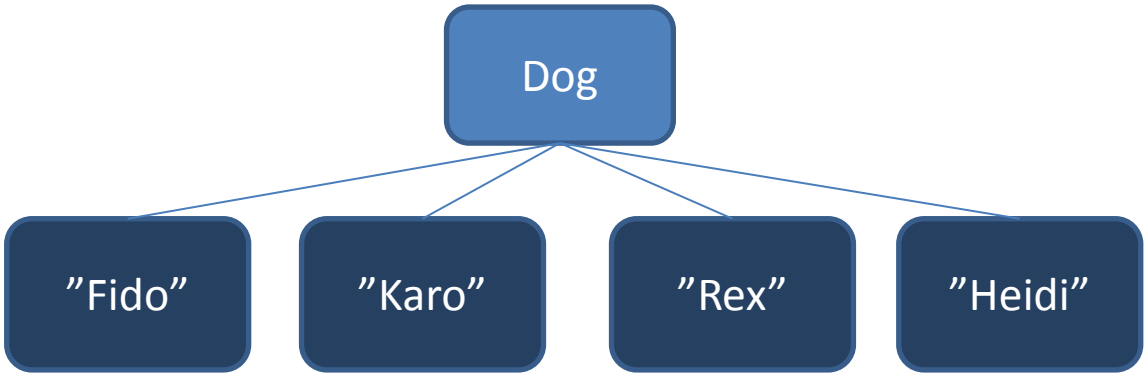
- Klasser definierar (utgör idén)
- Objekt instantierar (utgör förekomsten)

- En klassdefinition
- Många instanser från samma definition



# Klasser och objekt

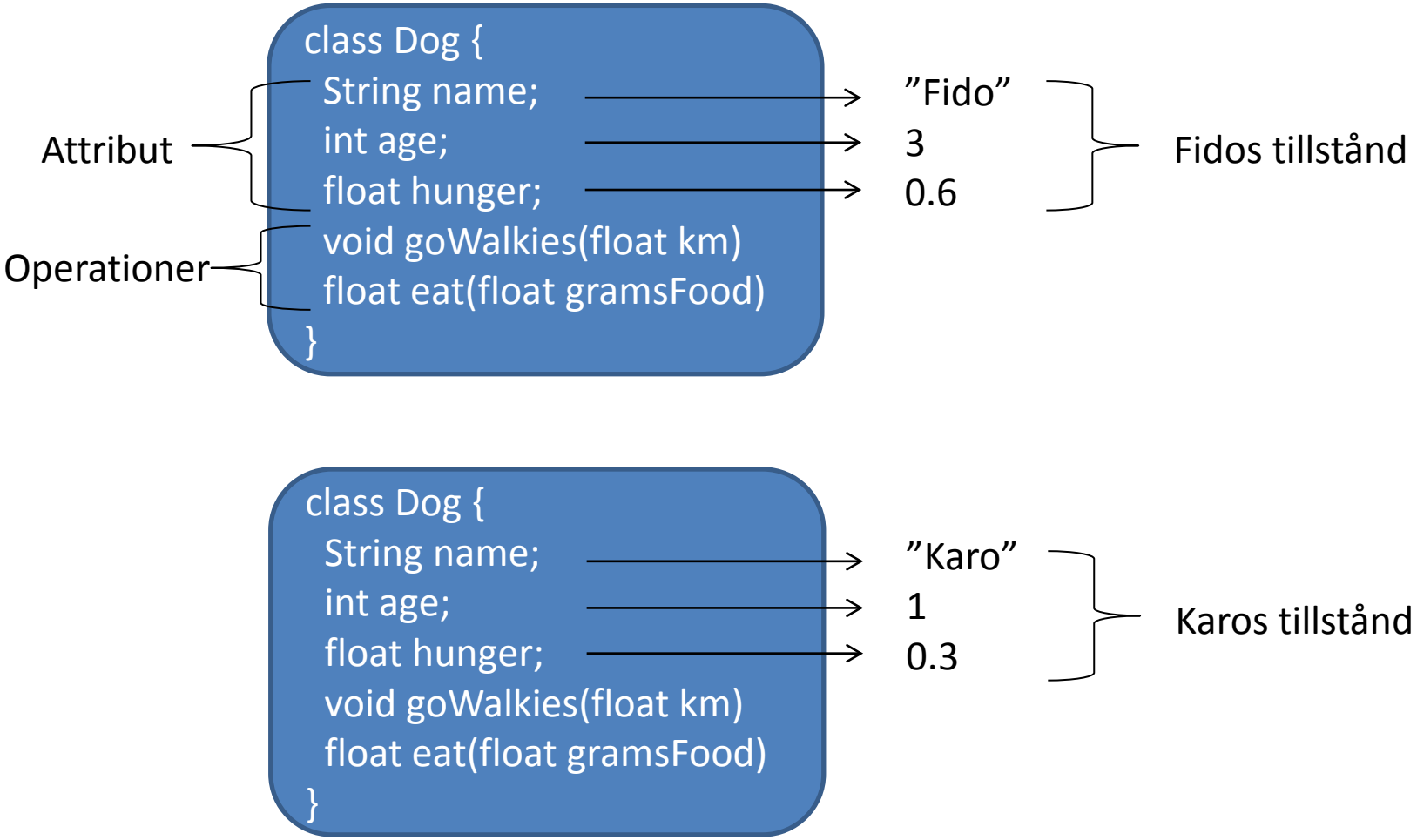
```
public class Dog {
  String myName;
  public Dog (name) {
    myName = name;
  }
}
…
Dog d1 = new Dog(“Fido”);
Dog d2 = new Dog(“Karo”);
Dog d3 = new Dog(“Rex”);
Dog d4 = new Dog(“Heidi”);
```




Klass

Instanser

# Objekt har tillstånd (state)



äntligen

# TILLVERKA OCH ANVÄNDA OBJEKT

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Antal sidor på tärningen.
Konstant

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Antal ögon upp.
Inte åtkomlig utifrån

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Konstruktor, initierar instansen

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Kastar tärningen

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Sätter tärningens värde

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Returnerar tärningens värde

# Klassen Die.java (tärning)

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Returnerar en sträng som beskriver tärningen

# Reflektioner?

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

# Reflektioner?

```
public class Die {
  private final int MAX = 6;
  private int faceValue;
  public Die() {
    faceValue = 1;
  }
  public int roll () {
    faceValue = (int) (Math.random() * MAX) + 1;
    return faceValue;
  }
  public void setFaceValue(int value) {
    faceValue = value;
  }
  public int getFaceValue() {
    return faceValue;
  }
  public String toString() {
    return Integer.toString(faceValue);
  }
}
```

Vilket värde som helst accepteras. faceValue kunde lika gärna vara public...

# Dice.java (tärningar)

```
public class Dice {
  private Die die1 = new Die();
  private Die die2 = new Die();

  public int roll() {
    return die1.roll() + die2.roll();
  }

  public int getValue() {
    return die1.getFaceValue() +
           die2.getFaceValue();
  }
}
```

^_^

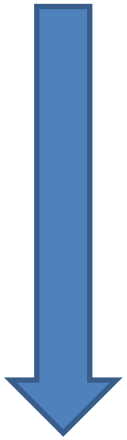
# METODER I DETALJ

# Metoder innehåller kod

- Programsatser utförs enligt
  - sekvens: a; b; c; ...
  - selektion: if (villkor) t-stmt ; else f-stmt ;
  - iteration: while (villkor) a;
- Programsatser
  - Tilldelning: variabel = uttryck ;
  - Metodanrop
  - Kontrollstruktur: if, switch, while, for, do, ...
  - Satsblock: {a; b; c;}

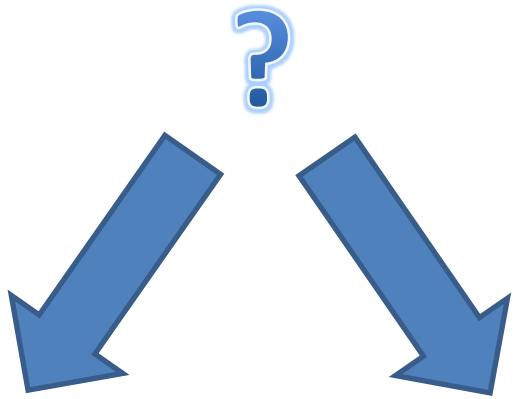
# Sekvens

```
System.out.print(count);

System.out.print(”: ”);

System.out.print(s);

System.out.println();
```



# Selektion – if else

```
if (upperLimit < x) {

}
```



```
if (c == 'a') {

}
else {

}
```

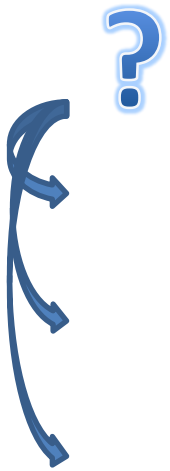
```
if (c == 'a') {

}
else if (c == 'b') {

}
```

# Selektion – switch case

```
switch(c) {
  case 'a':
    …
    break;
  case 'b':
    …
    break;
  case 'c':
    …
    break;
  default:
    …
  break;
}
```

Switch-uttrycket skall ge en primitiv datatyp

Case-konstanterna skall matcha typen



Fr o m Java version 7 är även String tillåten

# Selektion – switch case

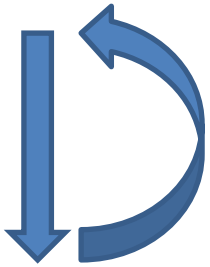
```
switch(c) {
  case ’a’:
    …
    break;
  case ’b’:
    …
    break;
  case ’c’:
    …
    break;
  default:
    …
  break;
}
```

```
if (c == ’a’) {
  …
}
else if (c == ’b’) {
  …
}
else if (C == ’c’) {
  …
}
else {
  …
}
```

# Iteration - for

```
for (<initiering>;<villkor>;<uppdatering>) {
  …
}
```

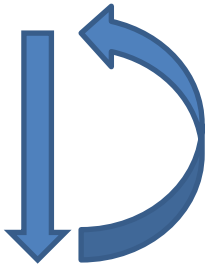
```
for (int i = 0; i < limit; i++) {
  …
}
```



# Iteration – for each

```
for (<variabel> : <datasamling>) {
   …
}
```

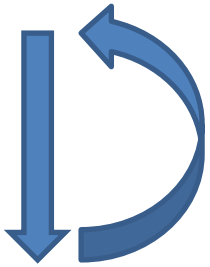
```
for (String s : args) {
   …
}
```



# Iteration – while

```
while(<villkor>) {
   …
}
```

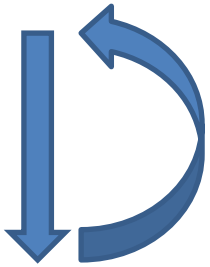
```
while(i < limit) {
   …
}
```



# Iteration – do

```
do {
   …
} while(<villkor>);
```

```
do {
   …
} while (i < limit);
```



# Loopkontroll

- **`break`** – hoppa till slutet av närmast omgivande for, while, do eller switch

- **`continue`** – hoppa till början av den närmast omgivande for, while eller do

# Parameteröverföring

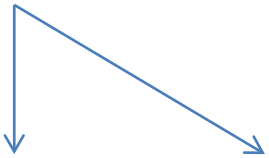
Metodens deklaration

```
char calc(int num1, int num2, String message){
  int sum = num1 + num2;
  char result = message.charAt(sum);
  return result;
}
```

# Parameteröverföring

Varsitt uttryck av typen int

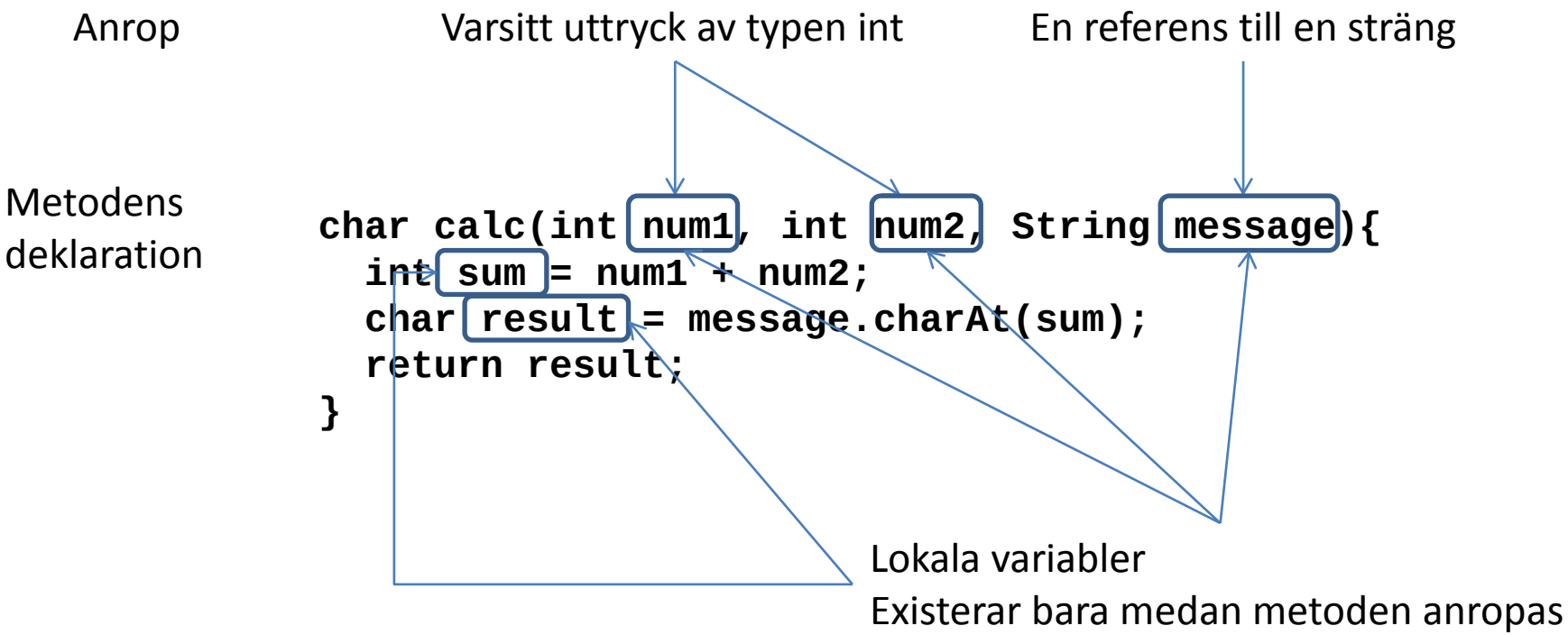
En referens till en sträng



Metodens deklaration

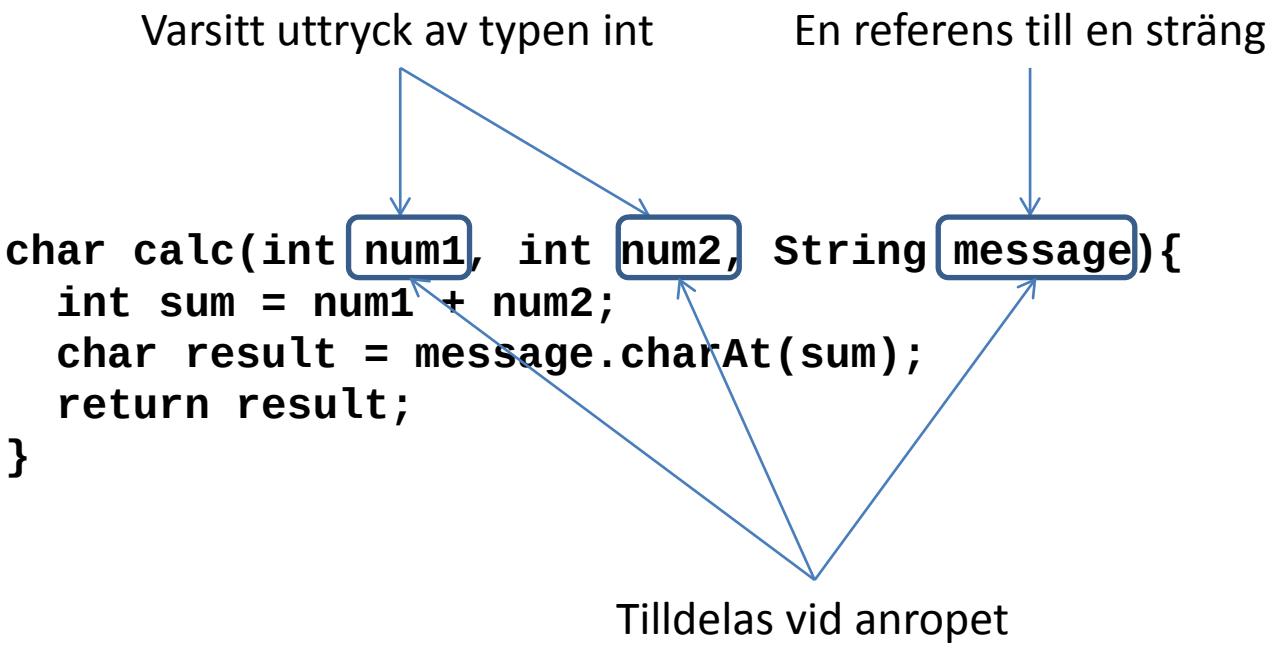
```
char calc(int num1, int num2, String message){
  int sum = num1 + num2;
  char result = message.charAt(sum);
  return result;
}
```

# Parameteröverföring

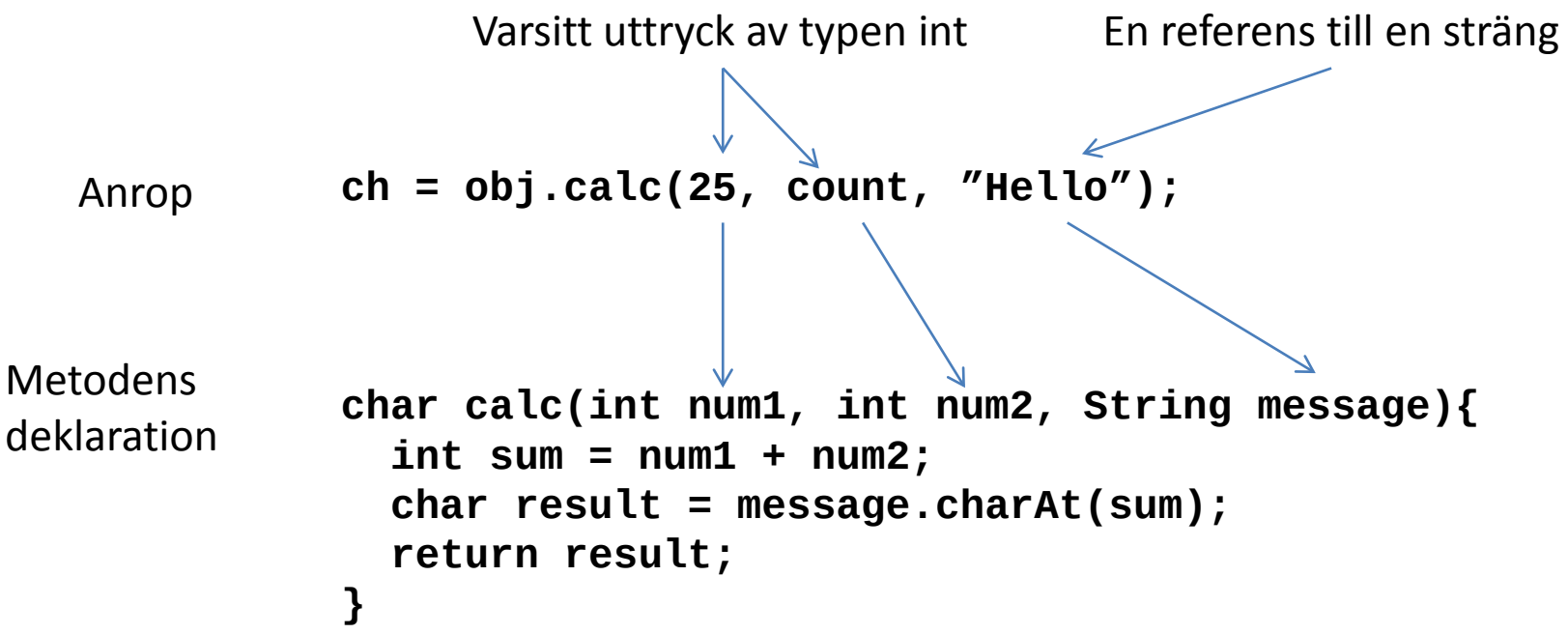


# Parameteröverföring

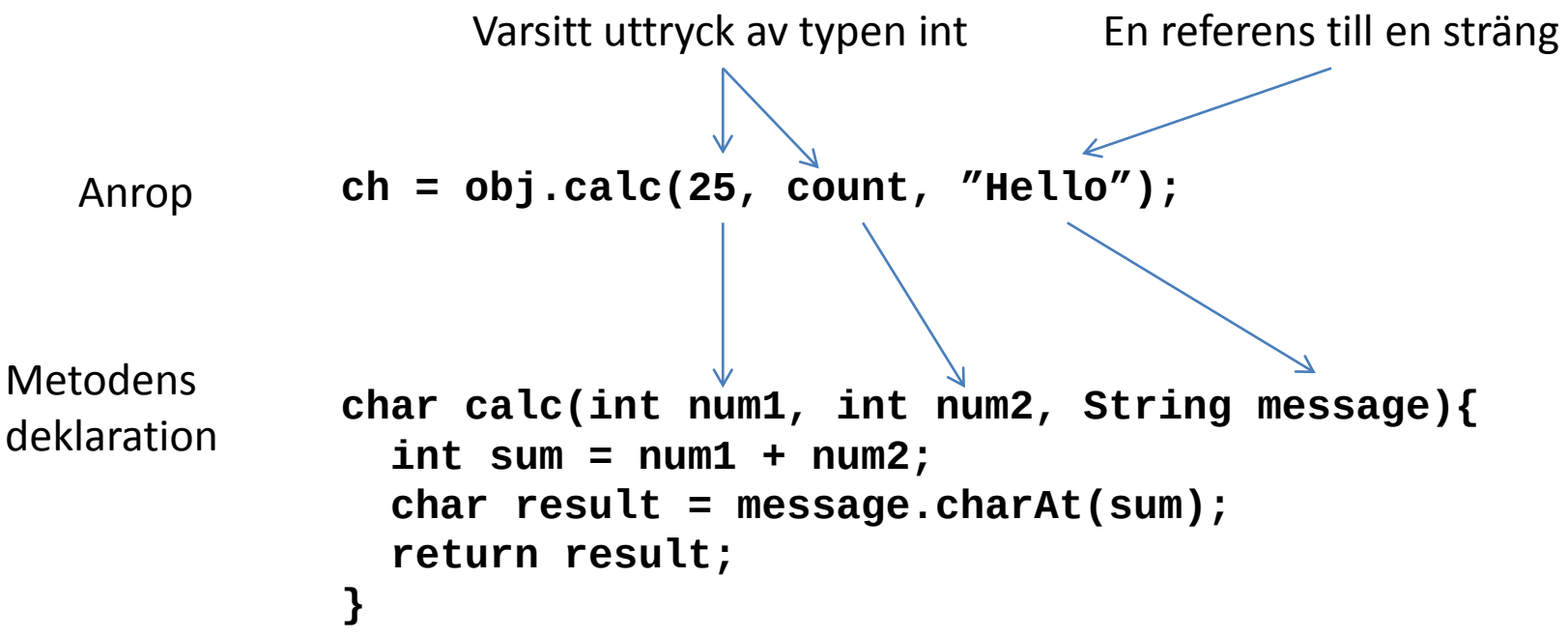
Metodens deklaration



# Parameteröverföring



# Parameteröverföring




Parameteröverföring i Java är by-value (ekvivalent med vanlig tilldelning)

Primitiva datatyper kopieras

Objektreferensen kopieras men *inte* objektet som refereras!

# Returvärde - return

Anrop

```
ch = obj.calc(25, count, ”Hello”);
```

Metodens deklarerade typ

Metodens deklaration

```
char calc(int num1, int num2, String message){
  int sum = num1 + num2;
  char result = message.charAt(sum);
  return result;
}
```

Uttrycket efter return måste ha samma typ som metoden.

Kompilatorn klagar om **`return`** fattas eller har fel typ.
Man kan ha fler än en return-sats.
Om metodens typ är **`void`** så krävs inte return (men **`return;`** är ok)

# Metoder och signaturer

- Varje metod har en signatur (kännetecken)
- Signaturen bestäms av metodens
  - klass
  - namn
  - parametrarnas typ och ordningsföljd
- Dessa metoder är alltså olika:
  - **`remove(int index)`**
  - **`remove(Object o)`**

# Metoder och signaturer

- Kompilatorn använder signaturen för att hitta rätt metod att anropa:
  - **`PrintStream.print(boolean b)`**
  - **`PrintStream.print(char c)`**
  - **`PrintStream.print(char[] c)`**
  - **`...`**
- Metodens returtyp ingår *inte* i signaturen:
  - **`int foo(int x)`**
  - **`long foo(int z)`**
- Kompilatorn kan inte avgöra vilken som avses i t ex **`foo(99);`**

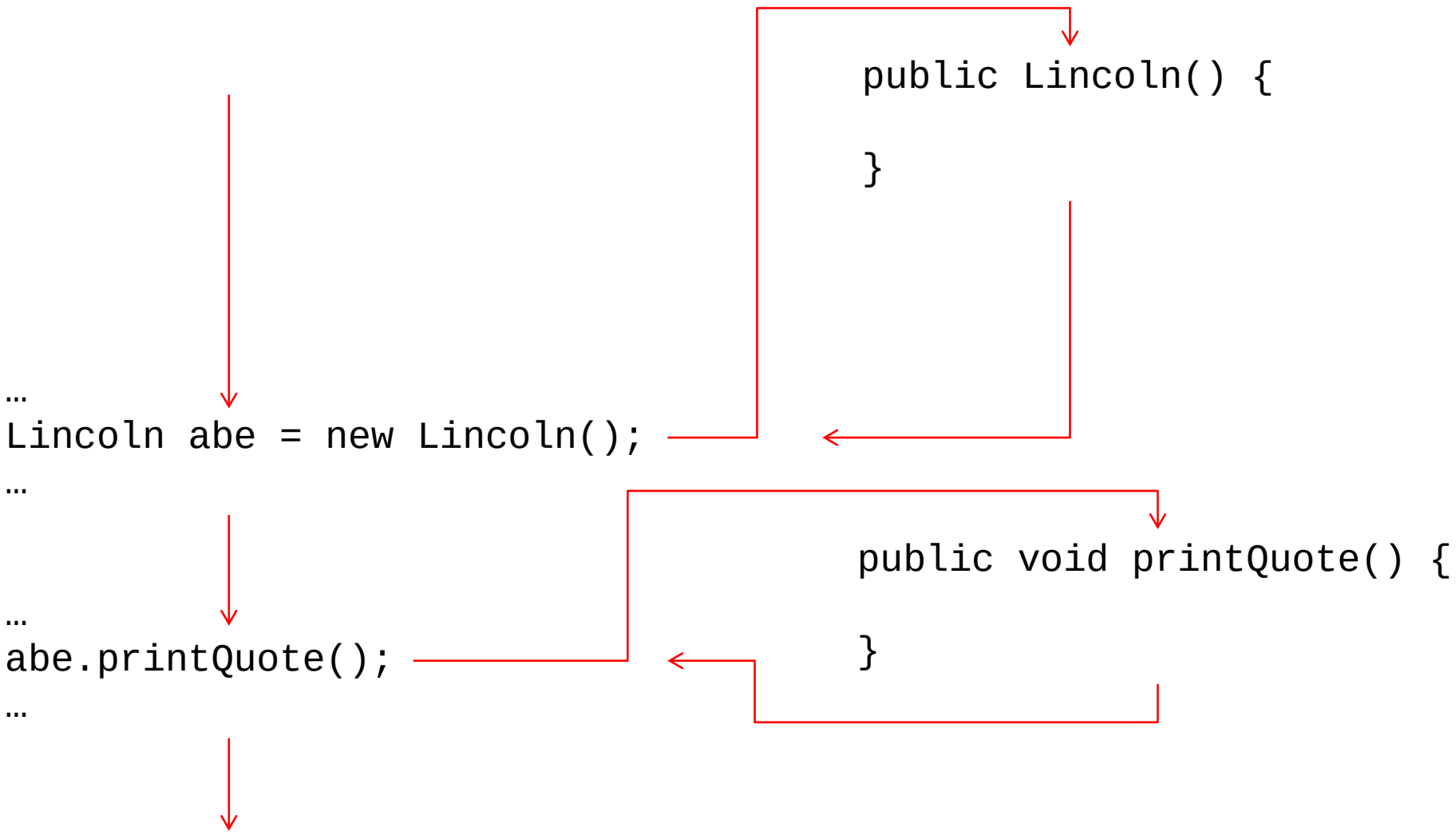
# Konstruktorn

- Konstruktorn anropas när instansen skapas
- Konstruktorn initierar och returnerar instansen, och har därför samma typ som klassen:
  - **`public Die () {}`**
- Man kan ha flera konstruktorer med olika parametrar
- **`public Die(int firstValue) {faceValue = firstValue;}`**

# Lincoln.java (igen)

```
//********************************************************************
//  Lincoln.java       Author: Lewis/Loftus
//  Demonstrates the basic structure of a Java application.
//********************************************************************
public class Lincoln
{
  public void printQuote() {
    System.out.println ("A quote by Abraham Lincoln:");

    System.out.println ("Whatever you are, be a good one.");
  }
  //-----------------------------------------------------------------
  //  Prints a presidential quote.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    Lincoln abe = new Lincoln();
    abe.printQuote();
  }
}
```

# Metodanrop och kontrollflöde



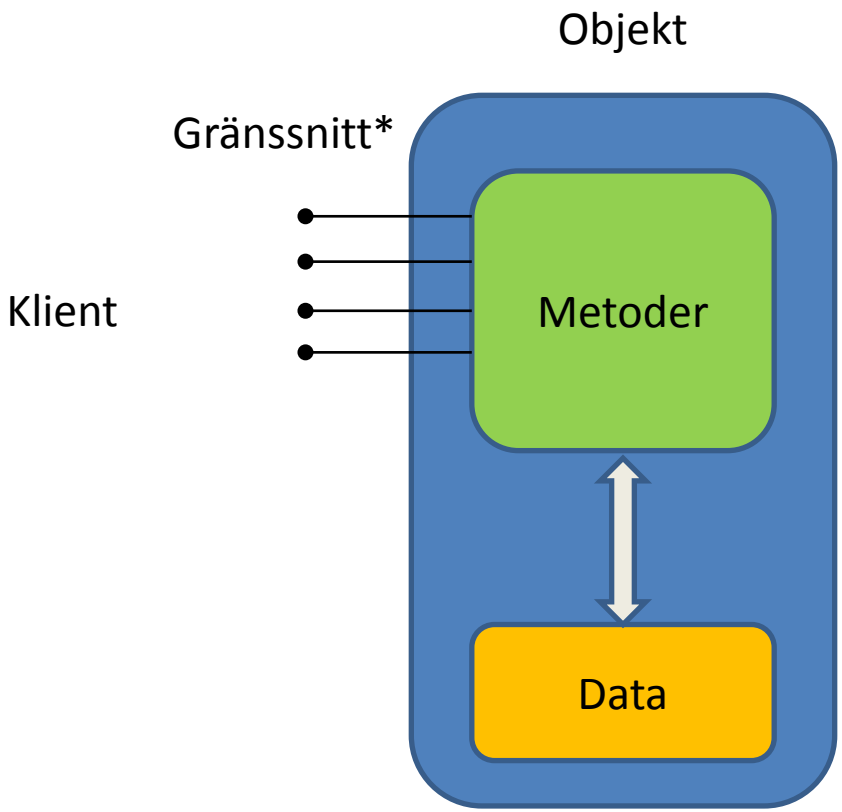
igen

# SLUT PÅ BILDER

rocket science

# INKAPSLING

# Inkapsling




* Begreppet *interface* har en väldefinierad betydelse i Java

# Åtkomstskydd till variabler och metoder

- Graden av inkapsling av ett objekt styrs med hjälp av åtkomstskydd:
  - **`public`** (synlig för alla klasser)
  - **`-(default)`** (synlig inom sitt package)
  - **`protected`** (synlig i underklasser)
  - **`private`** (synlig i den egna klassen)

- Skyddet gäller variabler, metoder och klasser
- Det är inte meningsfullt för lokala variabler

# Dice.java (tärningar)

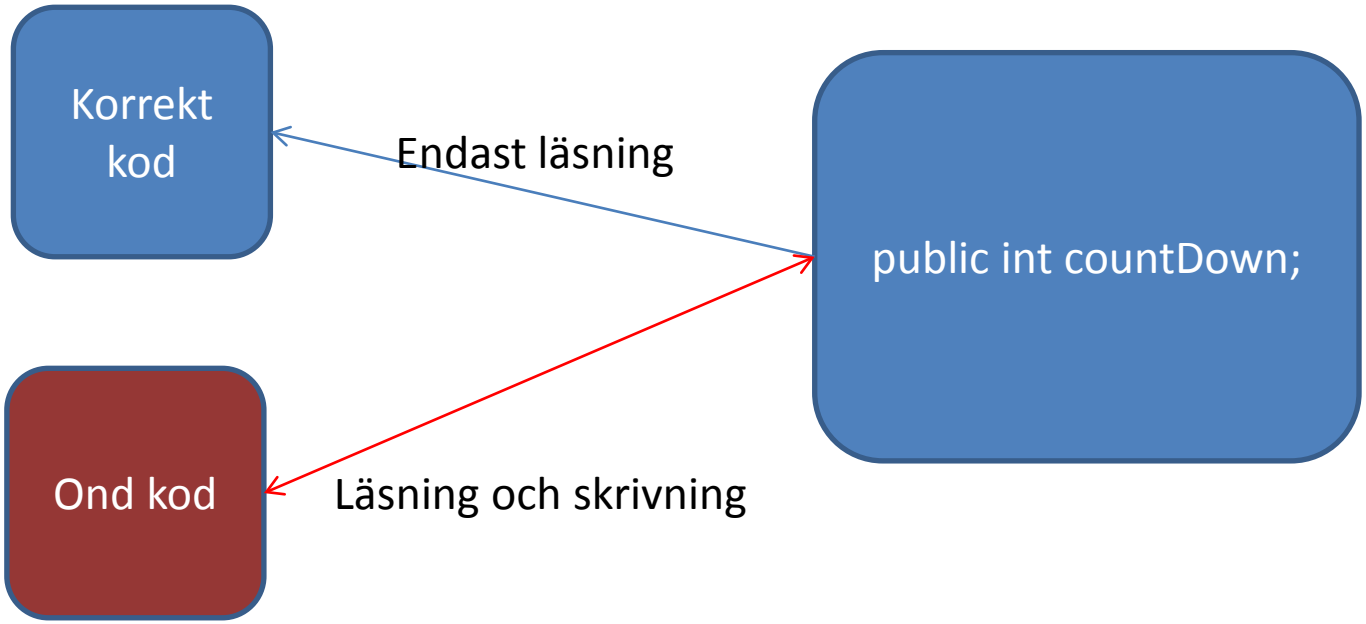
```
public class Dice {
  private Die die1 = new Die();
  private Die die2 = new Die();

  public int roll() {
    return die1.roll() + die2.roll();
  }

  public int getValue() {
    return die1.getFaceValue() +
           die2.getFaceValue();
  }
}
```

# Åtkomstskydd till variabler och metoder

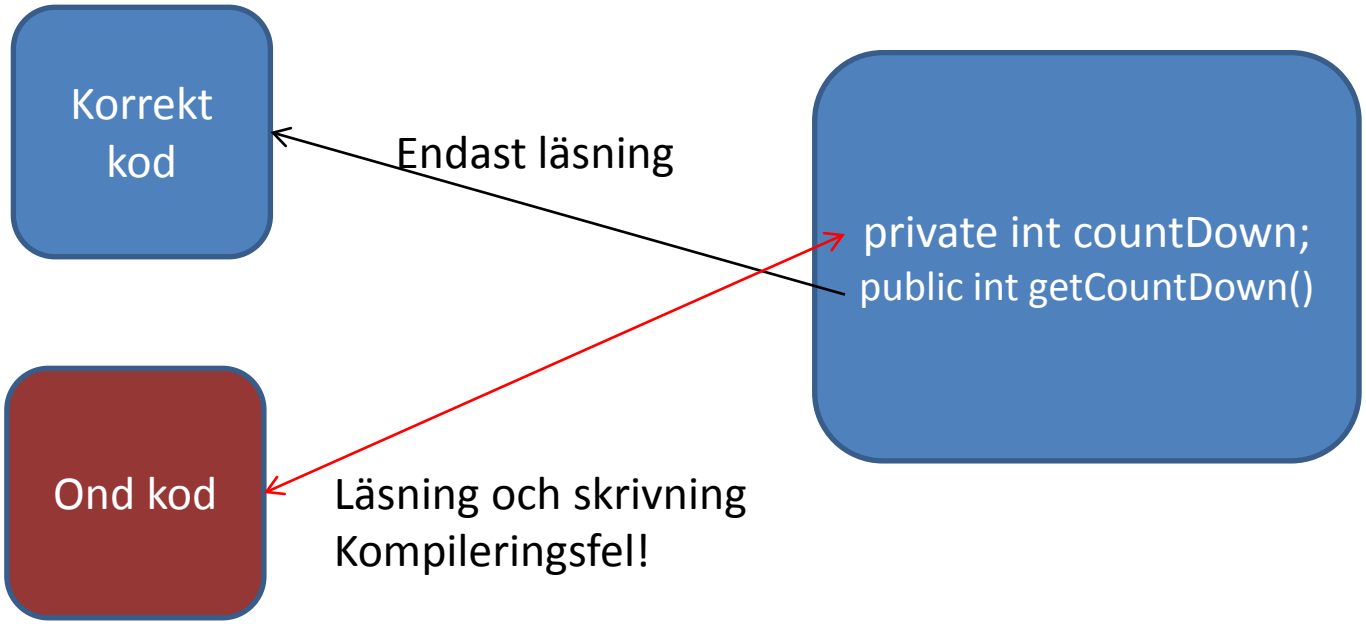
- Klasser är nästan alltid **`public`**
  - `public class Person { ...`
- Instansvariabler oftast **`private`** eller **`protected`**
  - `private int errorCount;`
- Metoder är oftast **`public`** eller **`protected`**
  - `public String getName()`
  - `protected void setLazyState()`

# Åtkomstskydd till variabler och metoder

- Läs av nedräkningen



# Åtkomstskydd till variabler och metoder



# Åtkomstskydd

| | `public` | (default) | `protected` | `private` |
|---|---|---|---|---|
| Variabler | Ingen inkapsling | Inkapsling i package | Inkapsling i klass och underklasser | Inkapsling i klass |
| Metoder | Gränssnitt för alla klienter | Stödfunktion i package | Stödfunktion i klass och underklasser | Stödfunktion i klass |

# Åtkomstskydd

Börja restriktivt och öppna upp vid behov.

| | `public` | `(default)` | `protected` | `private` |
|---|---|---|---|---|
| Variabler | Ingen inkapsling | Inkapsling i package | Inkapsling i klass och underklasser | Inkapsling i klass |
| Metoder | Gränssnitt för alla klienter | Stödfunktion i package | Stödfunktion i klass och underklasser | Stödfunktion i klass |

# Åtkomstskydd till variabler och metoder

- Om en instansvariabel kan läsas utifrån är den också skrivbar!
- Såvida det inte är en konstant (public final)

```
import java.awt.Color;
...
setColor(Color.blue);
...
setColor(Color.gray);
...
```

# Metodanrop och kontrollflöde