# DA339A
# Objektorienterad programmering

# Föreläsning F9
## Metoder

MALMÖ UNIVERSITET

# Innehåll

- Repetition
- Om metoder
- Skapa metoder
- Mer om metoder
- Sammanfattning


# Repetition

# Repetition

Vi har tidigare använt inbyggda metoder i Java. Exempelvis:

- `System.out.println()`
- `String.length()`
- `Math.sqrt()`
- `Integer.parseInt()`

# Exempel

En liten kodsnutt som använder Pythagoras sats

```
public static void main(String[] args){
    // a^2+b^2=c^2 (Pythagoras sats)
    double a = 3;
    double b = 4;
    double c = Math.sqrt(Math.pow(a,2)+Math.pow(b,2));
    System.out.println(c);
}
```

Här använder vi tre olika metoder: `Math.sqrt()`, `Math.pow()`, och `System.out.println()`

MALMÖ UNIVERSITET

# Om metoder

# Metod vs. funktion

Några av er har säkert stött på begreppet funktion i tidigare programmeringsspråk. När vi pratar om metoder i Java så är det i praktiken samma sak som funktioner.

Nu i början kommer vi att hantera våra metoder som om de vore funktioner.

# Vad är en metod

En metod är en samling kommandon som man kan exekvera med ett anrop

```
public static double max(double a, double b){
    double greatest; // Initierar variabeln
    if (a >= b){
        greatest = a;
    }
    else{
        greatest = b;
    }
    return greatest;
}
```

Nu hade vi kunnat anropa funktionen med `double c = max(5.2, 7.9);`

# Varför använda metoder?

Det finns flera anledningar till att vi vill arbeta med metoder

- Det ökar läsbarheten (detta kan inte överskattas)
- Koden blir lättare att felsöka
- Vi behöver inte upprepa kod i onödan
- Lättare att återanvända kod i andra projekt


# Skapa metoder

# Skapa egna metoder

När du skapar en metod i Java (och flera andra språk) är det ett par saker du behöver ange.

- Tillgänglighet
- Returvärde
- Metodnamn
- Parametrar

Vi kommer att ta upp *returvärde*, *metodnamn*, och *parametrar* idag. *Tillgänglighet* tar vi upp när vi pratar om objekt.

# Skapa egna metoder

```
public static char myMethod(String s){
    char firstLetter = s.charAt(0);
    return firstLetter;
}
```

- `public` anger tillgänglighet
- `static` styr vad den får göra
- `char` anger vad metoden returnerar
- `myMethod` metodens namn
- `(String s)` *parametrar* som metoden tar emot
- `return firstLetter;` skickar tillbaka `firstLetter`

# Anropa metoder

```
public static void main(String[] args) {
    String x = "All that is gold does not glitter";
    char a = myMethod(x); // Anropa metoden, spara svaret i a
    System.out.println(a);
}
public static char myMethod(String s){
    char firstLetter = s.charAt(0); // Hitta första bokstaven
    return firstLetter; // Skicka tillbaka första bokstaven
}
```

# Parametrar

En metods *parametrar* är det som metoden tar emot i metodhuvudet. I exemplet på den tidigare bilden var det en sträng med namnet `s`. Metoder kan ta emot flera parametrar.

```
public static double max(double a, double b){
    double greatest; // Initierar variabeln
    if (a >= b){
        greatest = a;
    }
    else{
        greatest = b;
    }
    return greatest;
}
```

# Parametrar

Parametrarna kan ha olika datatyper

```
public static String secondMethod(String a, char b){
    String newString = a+b; // Konkatenerar texten
    return newString; // Returnerar vår nya sträng
}
```

# Argument

Ett *argument* är värdet vi skickar in till en metod när vi anropar metoden.

```
public static void main(String[] args) {
    String s = secondMethod("Wow", 'a');
    System.out.println(s);
}
```

Här är `"Wow"` och `'a'` argument som vi skickar in till metoden.

argument → parameter

# Returvärde

Java är ett strikt typat språk, vilket betyder att man behöver hålla koll på variablers datatyper. Det gör att vi vill att en metod alltid returnerar samma datatyp. Detta anger vi i metodhuvudet.

```
public static String secondMethod(String a, char b){
    String newString = a+b; // Konkatenerar texten
    return newString; // Returnerar vår nya sträng
}
```

Här ser vi att `secondMethod` anger att den skickar tillbaka en `String`.

I vårt tidigare exempel `max` returnerade vi alltid en `double`

# Void-metoder

Det finns tillfällen när en funktion inte ska returnera något. Då anger man detta med `void`

```
public static void greet(String name, int age){
    System.out.println("Hello " + name+ ", it's so cool that you are " +
age + "years old!");
}
```

# Statiska metoder

Statiska metoder är speciella på så vis att de är kopplade direkt till klassen de är skrivna i och inte till en instans av en klass (mer om detta längre fram i kursen)

Vi kommer att använda statiska metoder fram tills att vi börjar jobba med klasser och objekt.

# Metodkropp

Med *metodkropp* menar vi det som kommer mellan måsvingarna.

Det är i metodkroppen vi skriver all kod som ska exekveras.

```
public static String secondMethod(String a, char b){
    String newString = a+b; // Konkatenerar texten
    return newString; // Returnerar vår nya sträng
}
```

I `secondMethod` är det raderna 2 och 3 som utgör metodkroppen.

# Metodkropp

Metodkroppen kan vara hur lång som helst. Men en rekommendation är att hålla den kort. Skulle den bli lång kan man fundera på om man kanske ska bryta ut delar av metodkroppen till egna metoder.

En tumregel är att man ska kunna se hela metoden utan att behöva scrolla på skärmen. *Detta går inte alltid att följa.*

# Kodningsfilosofi

Försök att skapa metoder som gör en sak.

Ge metoderna korta men beskrivande namn. Om du inte kan ge den ett kort namn fundera på om det borde vara flera metoder istället.

Metoder kan anropa andra metoder.

# Metod som anropar andra metoder

Här har vi ett exempel på en metod `checkSensors` som anropar flera andra metoder.

```
public void checkSensors(){
    checkDoorSensors();
    checkStaircaseSensors();
    checkLobbySensors();
    checkOfficeSensors();
}
```

Genom att dela upp `checkSensors` i flera  mindre metoder så är det fortfarande tydligt vad metoden gör.

# Exempel

```
public double calculateSTD(double[] values){
    double sum = 0; // Härifrån
    for (int i = 0; i < values.length; i++){
        sum += values[i];
    } // Allt detta hade kunnat vara en egen metod, calculateMean
    double mean = sum/values.length; // Hit
    double s = 0;
    for (int i = 0; i < values.length; i++){
        s += Math.pow((values[i]-mean), 2);
    }
    double std = s/values.length;
    return std;
}
```

# Return

Kommandot `return` skickar tillbaka det värde som kommer efteråt.

```
public static String secondMethod(String a, char b){
    String newString = a+b; // Konkatenerar texten
    return newString; // Returnerar vår nya sträng
}
```

Här är det `newString` som skickas tillbaka till anropet.

# Flera return

Tidigare hade vi följande exempel:

```
public static double max(double a, double b){
    double greatest; // Initierar variabeln
    if (a >= b){
        greatest = a;
    }
    else{
        greatest = b;
    }
    return greatest;
}
```

# Flera return

Vi kan korta ner den koden, och göra den läsligare:

```
public static double max(double a, double b){
    if (a > b){
        return a; // Returnerar a
    }
    else{
        return b; // Returnerar b
    }
}
```

# Flera return

När programmet når en `return` så avslutas metoden och värdet returneras.

Koden kommer alltså inte att fortsätta och returnera något mer senare.

# Mer om metoder

# Överlagring av metoder

Det händer att man har en metod där man vill att den ska kunna ta emot olika parametrar. Kanske för att man vill kunna ta emot olika antal parametrar eller för att man vill göra olika saker beroende på datatyperna som skickas in.

Det är möjligt att skapa flera metoder med samma namn, förutsatt att de alla returnerar samma datatyp, samt att de har olika parameterlistor.

# Exempel

```
public double max(double a, double b){
    if (a>=b){return a;} // Vertical space är prime real estate
    else {return b;}
}
public double max(double a, double b, double c){
    if (a>=b && a>=c){return a;}
    else if (b>=a && b>= c){return b;}
    else {return c;}
}
```

Nu gör kommandona `max(1.0,3.0)` och `max(5.0.3.0,7.0)` lite olika saker.

# Exempel

```
public double max(double a, double b){
    if (a>=b){return a;} // Vertical space är prime real estate
    else {return b;}
}
public double max(double a, double b, double c){
    return max(a, max(b, c));
}
```

Hur fungerar det här?

# Exempel

```
public double max(double a, double b){
    if (a>=b){return a;} // Vertical space är prime real estate
    else {return b;}
}
```

Vad händer om man skickar in två `int` till `max` nu?

# Exempel

Skulle vi ändra parametrarna till `int` istället och skicka in två `double`, vad händer då?

```
public double max(int a, int b){
    if (a>=b){return a;} // Vertical space är prime real estate
    else {return b;}
}
```

# Scope

Som regel kan en metod inte komma åt variabler som skapas utanför metoden.

På samma sätt kan andra metoder inte komma åt variabler som instansierats i en annan metod.

Detta skapar en inkapsling och decentralisering som gör att varje metod bara gör sin grej.

Man säger ibland att variablerna är *lokala*.

# Call stack

När du anropar en metod så skapas något som kallas för en *stack frame*.

En *stack frame* håller koll på initialiserade variabler och vart de pekar i minnet.

När en metod avslutas så returneras eventuellt return-värde och sen förstörs *stack framen* och alla lokala variabler glöms bort ur minnet.

# Call stack

För att förstå hur call stacken fungerar behöver man förstå hur *stackar* fungerar.

En stack är en *datastruktur* som lagrar data enligt modellen *Last in First out (LiFo)*

Vi kommer inte att diskutera stackar på djupet i den här kursen.

# Call stack

Förenklat kan vi jämföra en stack med en hög med disk. Du lägger ny disk överst i högen, och diskar den översta disken först.

Likt diskhögen hamnar den senast anropade metoden överst. När den avslutas kastas det som är kvar i framen och programmet fortsätter med den näst översta framen i stacken.

När den sista framen är avslutad så är vår main-metod avslutad och programmet stängs ner.

MALMÖ UNIVERSITET

# Sammanfattning

# Sammanfattning

- **Metoder ökar läsligheten av vår kod**
- Metoder ska vara specifika och namnade på ett beskrivande sätt
- Metoder returnerar alltid samma typ av värden
- En metod kan vara void och inte returnera något alls
- I metodhuvudet anger vi tillgängligheten, datatyp som returneras, metodnamn, och *parametrar*
- Vi anropar metoder genom att skriva metodnamnet och skicka med *argument*
- En metod avslutas när den når return

# Sammanfattning

- **Metoder ökar läsligheten av vår kod** (om någon skulle glömt det)
- Metoder kan *överskugga* varandra om de har samma namn men olika *parameter-listor*
- När vi anropar en metod skapas en ny *stack frame*
- En *stack frame* håller reda på alla lokala variabler.

# Rekommenderad läsning

Boken:

Deitel & Deitel, kapitel 5, sidorna 204–236

Url:

https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

https://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html

https://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html

Allmänt:

Cornelia Funke, *The Colour of Revenge*

MALMÖ
UNIVERSITET