Introduktion till objektorientering

Vad är objektorientering egentligen?

Hur relaterar det till datatyper?

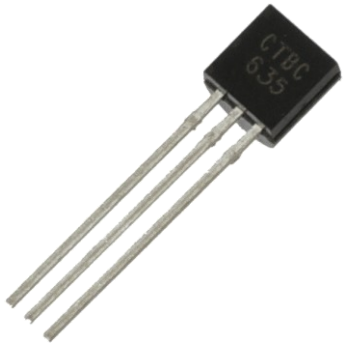
Hur relaterar det till verkligheten?

Perspektiv 1: Från verkligheten till objektorientering

Mojänger

jonkv@ida

Verkligheten innehåller **mojänger**

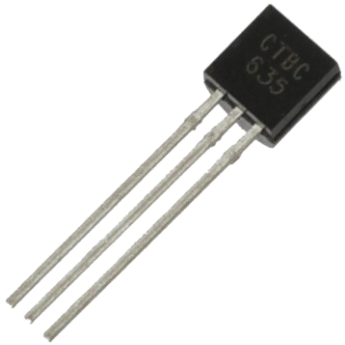


Mojängers egenskaper

jonkv@ida

Mojänger har ofta intressanta **egenskaper**

Längd/höjd
Färg
Motorstyrka
Toppfart



Strömtålighet
Maxfrekvens

Antal pucklar
Längd på ögonfransar
…

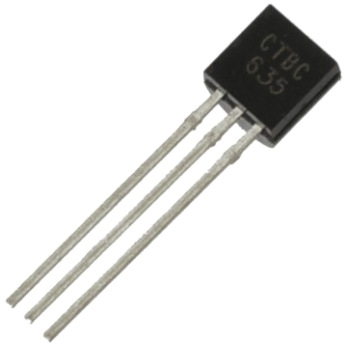
egenskaper = properties, attributes

Mojänger som gör något

jonkv@ida

En del mojänger kan **göra något**

Accelerera/decelerera
Köra till en plats
Slå på helljuset
Stänga av vindrutetorkarna



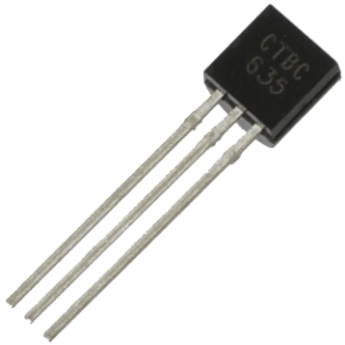
Släppa genom ström

Gå till en plats
Äta
Spotta

Terminologi

jonkv@ida

Vi behöver **terminologi!**



Vi har en mängd individer, **objekt**
(bilen ABC123, bilen XYZ789, …)

Varje **sort** av objekt är en **klass**
(klassen *bilar*, klassen *kameler*, …)
Objekt är **instanser** av en klass

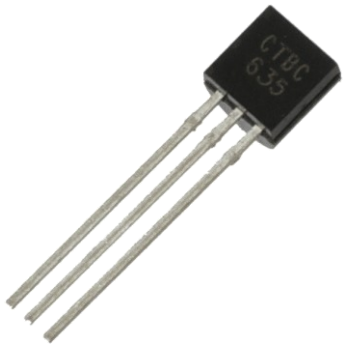
Objekt som tillhör samma klass har samma **egenskaper** och **funktionalitet**
(*alla* bilar har *någon* färg, kan köra)

→
En **objektorienterad syn på den fysiska världen**

Organisation

jonkv@ida

När vi **skriver program** om/för mojänger – *autonoma bilar*:



Vi kan organisera dem som klasser!

```
// Samla all bilrelaterad kod
// ➔ närmare koppling
// till verklighetens objekt och klasser
class Car {
    // egenskaper
    double length, height;
    double topSpeed;
    // funktionalitet
    void driveTo(Location loc) { … }
    void setLights(boolean on) { … }
}
```

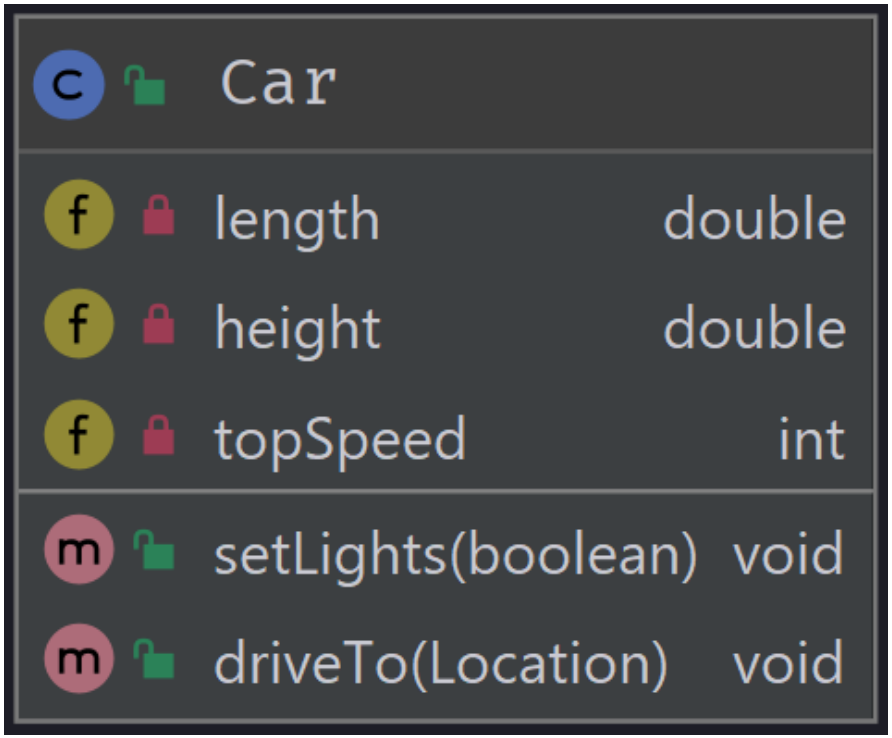
klass = class

Generell beskrivning: UML-diagram

jonkv@ida

Vi kan beskriva detta med ett **klassdiagram**

- En sorts **UML-diagram** i Unified Modeling Language
  - (Skapat av IDEA ➔ inkluderar några ikoner som inte är ren UML…)
  - Mer om detta när vi diskuterar ärvning



Klassnamn
Attribut
Operationer

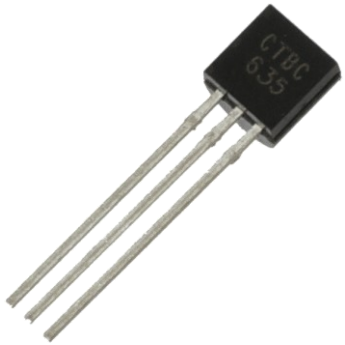
Organisation

jonkv@ida

När vi **skriver program** om/för mojänger – *Googles autonoma bilar*:

Vi skapar objekt vid behov, genom att instansiera klassen:

```
Car myCar = new Car();
Car other = new Car();
myCar.driveTo(home);
```



Varje bil har:

Samma beteenden (kod)

Samma egenskaper (längd, höjd)
Egna värden (*olika* längd)

instansiera = instantiate

Perspektiv 2: Från datatyper till objektorientering

(**Klassbaserad** objektorientering)

Datatyp: Datum

jonkv@ida

- Anta att vi har skapat datatypen **Date**

Exempel i "C-liknande" språk: struct

```
struct Date {
    int year;
    int month;
    int day;
}
```

- Skapa några **instanser** av typen – allokerar *minne*

Exempel i "C-liknande" språk

```
Date date1;
Date date2;
```

Utan OO: Funktioner

jonkv@ida

- Utan OO: Vi **hanterar** datumen med **funktioner**

Exempel i "C-liknande" språk

```
boolean isDivisibleBy(int large, int small) {
   return (large % small) == 0;
}

boolean isLeapYear(Date date) {
   if (isDivisibleBy(date.year, 4) && !isDivisibleBy(date.year, 100)) return true;
   if (isDivisibleBy(date.year, 400)) return true;
   return false;
}
```

Vanlig syntax:
post punkt *medlem*
date punkt *year*

Utan OO: Exempel

jonkv@ida

Vår kod…

```
add(myDate, 14);
if (isLeapYear(myDate)) { … }
```

anrop

Operationer på datum: Funktioner

```
boolean isLeapYear(Date date) {
   return ((date.year % 4 == 0) && (date.year % 100 != 0) ||
           (date.year % 400 == 0));
}
int daysSinceToday(date) { … }
void add(date, days) { … }
```

Inspekterar och manipulerar "**utifrån**"

myDate

year: 2021
month: 12
day: 31

Själva **posten** består bara av **passiva** data!

Operationer på objekt

OO 1: En utökning till poster

- Med (klassbaserad) objektorientering:
  - Post-typer blir **klasstyper** med **två sorters** medlemmar
    1. *Fält* eller *medlemsvariabler*, precis som tidigare
    2. *Metoder* eller *medlemsfunktioner*, nära kopplade till datatypen

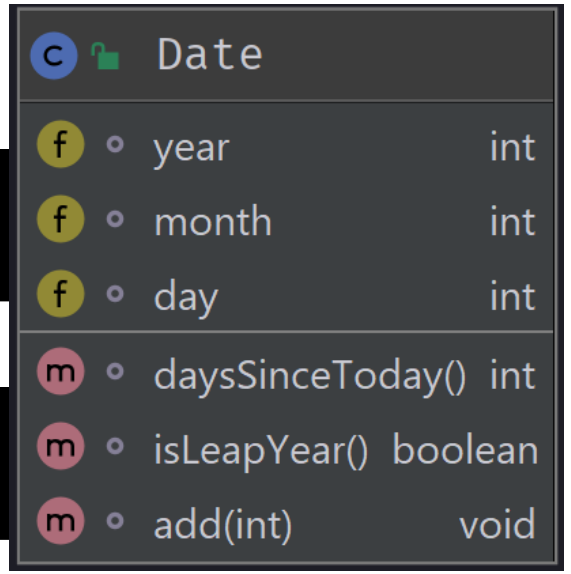
Exempel i "C-liknande" språk

```
class Date {
   int year;
   int month;
   int day;

   int daysSinceToday() { ... }
   boolean isLeapYear() { ... }
   void add(int days) { ... }
}
```

Vad "vet" ett datum?
Vilken information finns?

Vad kan ett datum göra?

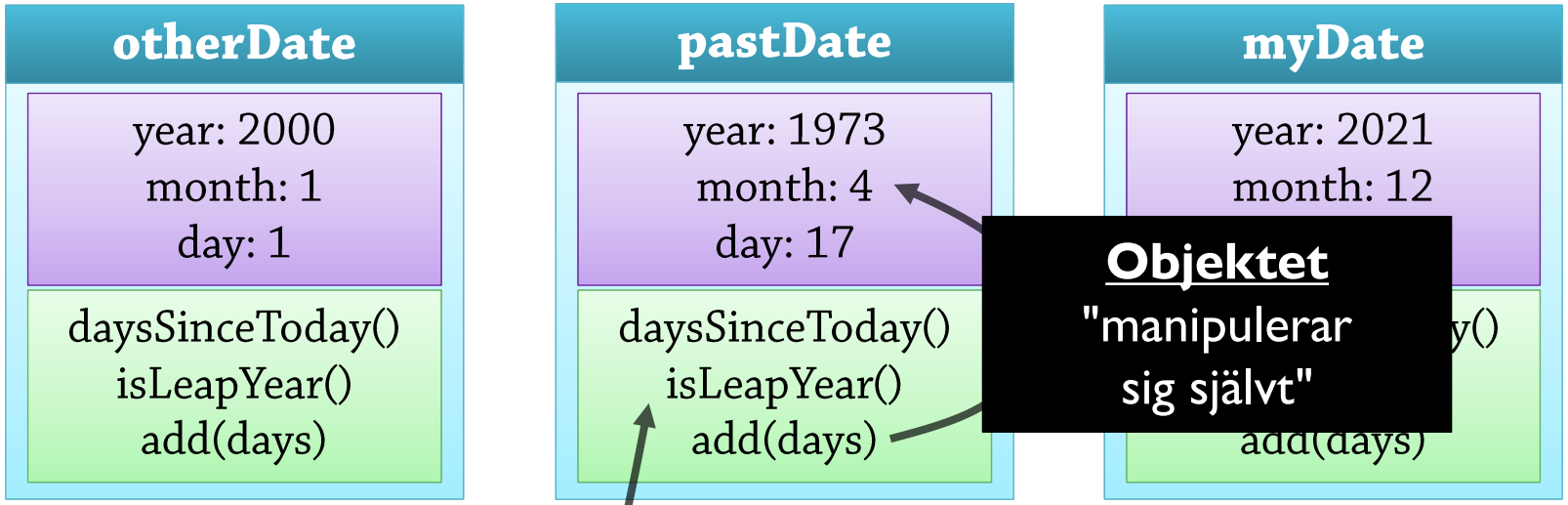


klass = class
fält = field
medlemsvariabel = member variable
metod = method
medlemsfunktion = member function

OO 2: Be objektet att göra något

jonkv@ida

- Själva värdet kallas inte **post** utan **objekt**



Vår kod…

```
Date pastDate = …;
pastDate.add(14);
if (pastDate.isLeapYear()) …;
```

Vanlig syntax:
objekt punkt *medlem*

Vi kan **be datumet tala om för oss**:
"Infaller **du** under ett skottår?"

➔ **Fundamental** princip:
"Klassen bestämmer över sina objekt"
(kan förhindra manipulation utifrån)

➔ **Grunden** för
ärvning, overriding, polymorfism, inkapsling, …

Klasser som begrepp

jonkv@ida

- Varje **klass** blir ett **”djupare” begrepp**!
  - ”Detta är en sträng, och alltså kan den…”:
    Helhetsförståelse för vad ett objekt *vet* och vad det *kan göra*
  - Söndra och härska – mellan olika datatyper
  - Men *samla ihop kod och information* inom en och samma datatyp