# DD1385

# Programutvecklingsteknik

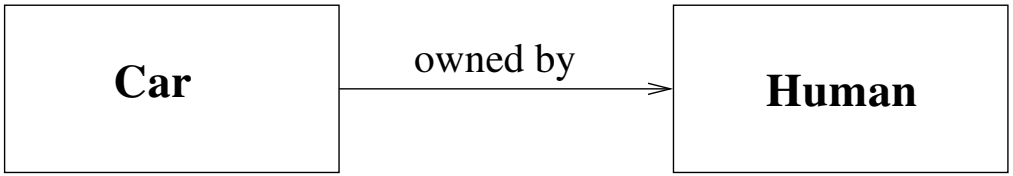
# Bilder till föreläsning 3

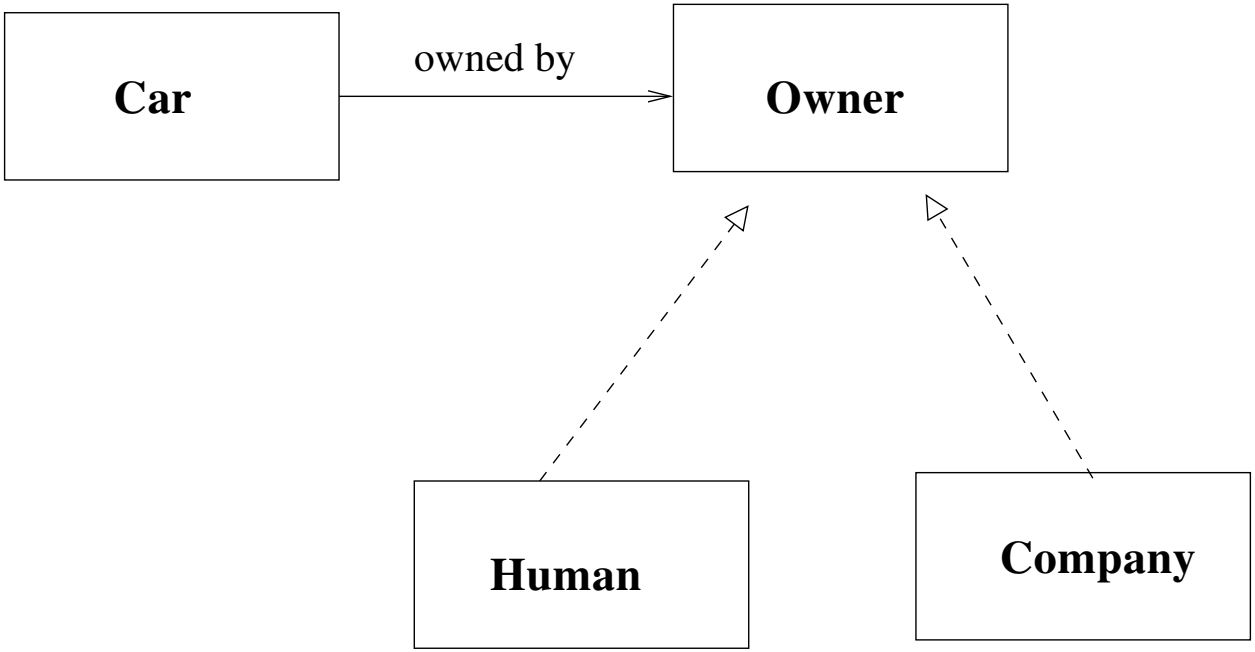
# Innehåll

- Introduktion till designmönster
- Designmönster:
  - Singleton
  - MVC

# High cohesion and low coupling

- Stark sammanhållning
- Lös koppling



# Fördelar

- minskar beroende mellan klasser
- ökar flexibiliteten
- ökar återanvändbarheten
- minskar komplexiteten / underlättar kommunikation mellan utvecklarna
- ökar (underlättar) förståelsen
- modifieringar i en del kräver färre modifieringar i andra delar
- underlättar testningen

# Design Patterns

Återanvändbar lösning på vanligt förekommande problem.

Mönstren delas in i tre kategorier:

- Structural  Creational  Behavioral

Varje mönster beskrivs med

- Namn
- Kategori
- Syfte
- Användningsområde
- Struktur (UML-klassdiagram)
- Deltagare (klasser och objekt)

Flera designmönster finns i Javas klassbibliotek

# Förebygg fel genom god design

Exempel:
Det får bara finnas en instans av klassen S.

- ett filsystem
- en fönsterhanterare
- ett kontrollobjekt

Hur gör man ?

## Lösning 1:

Kom ihåg:
"Skapa bara ett objekt av S, gör `new S()` endast en gång!"

## Lösning 2: *mycket bättre*

- Bygg in i designen av S att det bara får finnas en instans
- Program som försöker skapa flera går inte ens att kompilera

..... men hur gör man ?

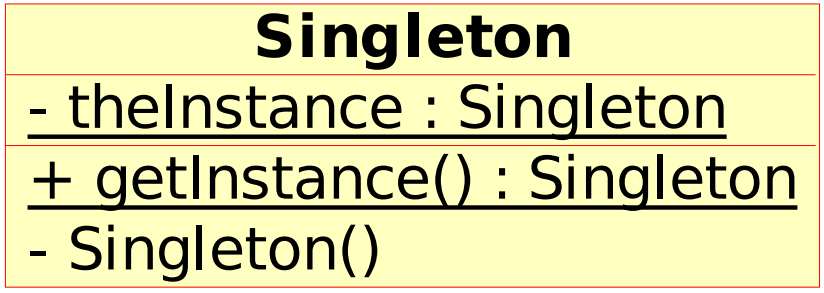
- I Java kan numera Enum-typer lösa problemet.
- I andra språk: mönstret Singleton
- Vi tar upp Singleton i Java ändå,
  som intressant användning av `private` och `static`
  och förberedelse till andra mönster.

## Singleton - klass som bara har en instans

- Gör konstruktorn `private`
- Skapa objektet inuti klassen
- Användare får tag på det enda objektet genom metodanrop

# Singleton

Det får bara finnas ett objekt av klassen




Går att utvidga till

"Det får finnas exakt N objekt av klassen"

# Javakod för Singleton, alternativ 1

Det enda objektet skapas när klassen laddas av JVM.

```
public class Singleton {

    private static Singleton theInstance
        = new Singleton();

    private Singleton () {}

    public static Singleton getInstance(){
        return theInstance;
    }
}
```

# Javakod för Singleton, alternativ 2

Här skapas det enda objektet när det efterfrågas första gången.

```
public class Singleton {

    private static Singleton
                          theInstance = null;

    private Singleton () {}

    public static Singleton getInstance(){
        if (theInstance == null)
            theInstance = new Singleton();
        return theInstance;
    }
}
```

# Kan Singleton-egenskapen ärvas?

Nej, inte i Java!

- Konstruktorer ärvs inte (i Java)
- Privata variabler och metoder ärvs inte (i Java)

$\Rightarrow$ Singleton-egenskapen kan inte ärvas

# Model-View-Control

- Användargränssnitt benägna med att komma med nya krav, så som nytt språk, anpassa till olika användare...
- Uppdatering av fönstersystem kan innebära modifieringar i användargränssnittet men inte några modifieringar hos systemets funktionella kärnan.
- Stark koppling mellan användargränssnittet och funktionella kärnan kan leda till utveckling och underhåll av flera mjukvarusystem, ett för varje typ av användare.

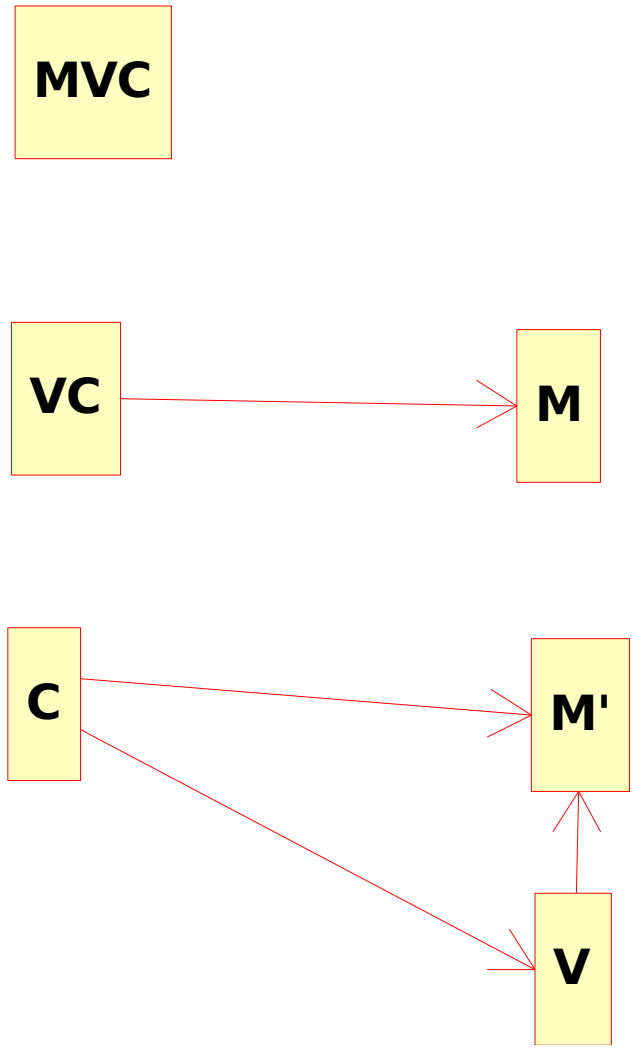
# Model-View-Control

- Att bygga ett system med erforderlig flexibilitet kommer att vara dyrt och felbenägen om användargränssnittet är tätt sammanflätade med den funktionella kärnan. (lös koppling)
- Anpassning eller portning av användargränssnitt bör inte påverka kod i den funktionella kärnan i programmet.
- Hur ska man göra?

# Model-View-Control

- Princip för strukturering av program(delar) med grafisk interaktion
- Model för data, logik, algoritmer
- View för "grafisk" representation av Model
- Control för användarens interaktion som uppdaterar Model (och View).
- Design-mönster eller arkitektur ?

# Model-View-Control



# MVC

- Inte alltid rätt att skilja på View och Control
- T.ex. om View består av knappar och Control består av samma knappar
- M – VC, modellen för sig och View-Control tillsammans är vanligt
- Varje M, V, C kan bestå av flera klasser.

Vi tittar på några små
programexempel
under föreläsningen

# Mock Object

Lite olika definitioner finns!

1. Ett interface implementeras med liten funktionalitet
2. Ett interface implementeras med simulerad funktionalitet