Objektorienterad Programkonstruktion

KTH Datavetenskap
och kommunikation

Föreläsning 18

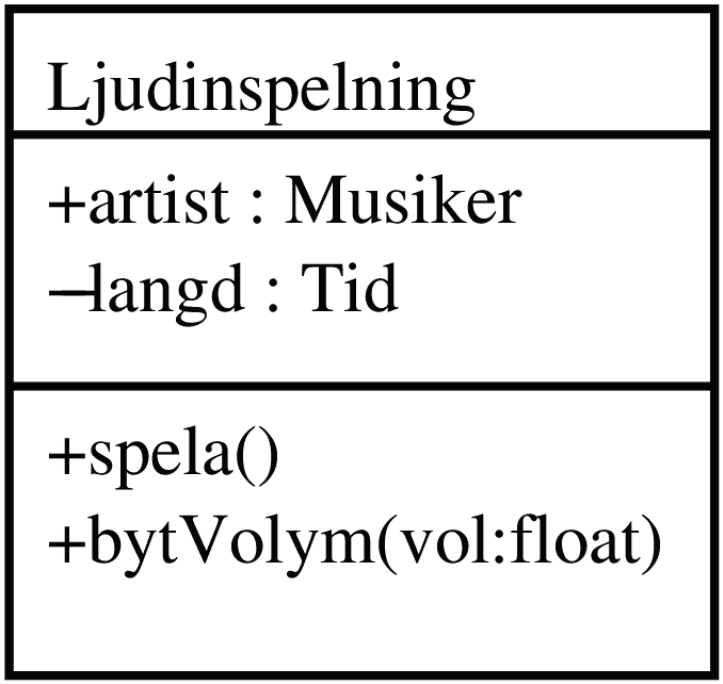
Repetition

Christian Smith

ccs@kth.se

UML klassdiagram

KTH Datavetenskap och kommunikation

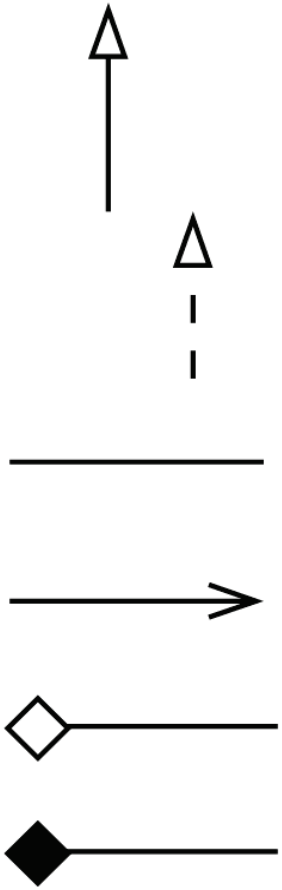


+ Publik	(Public)
- Privat	(Private)
~ Paket	(Package)
# Skyddad	(Protected)
Statisk	(Static)

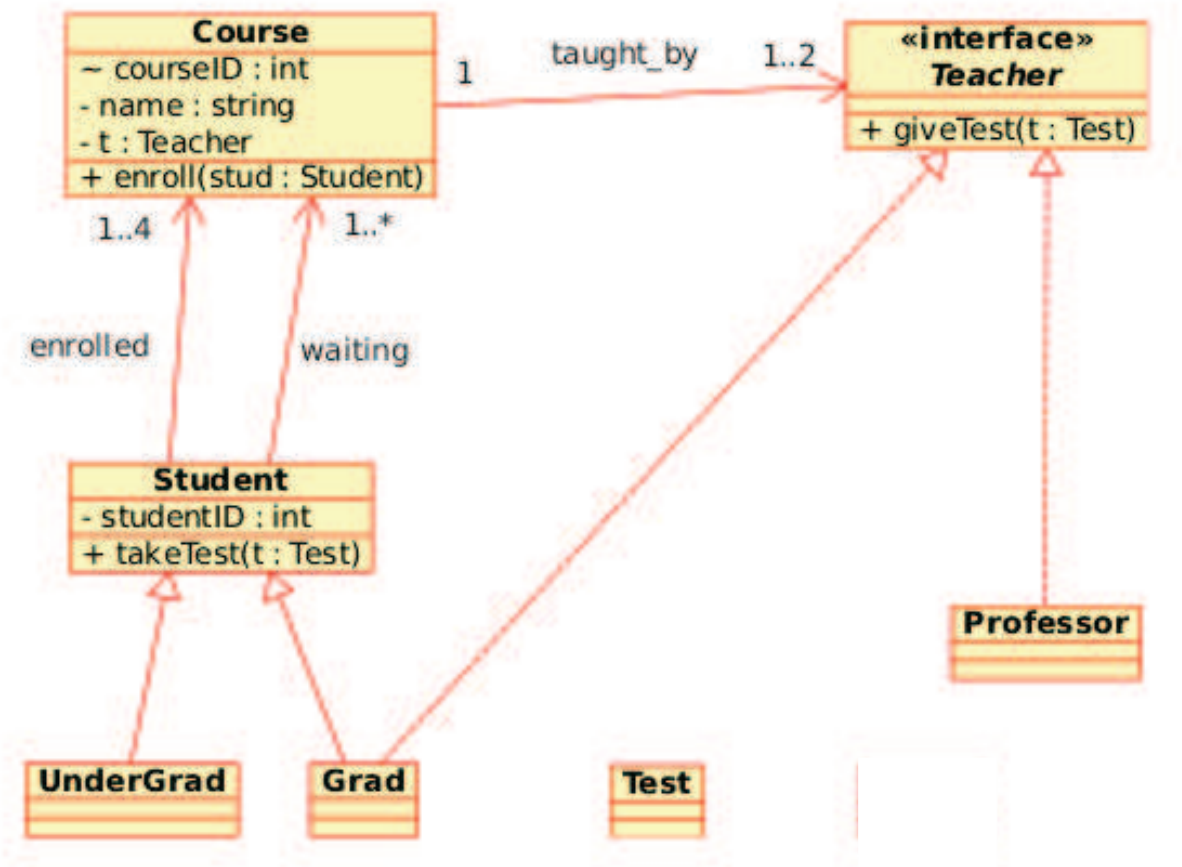
Relationer

KTH Datavetenskap och kommunikation

- Ärver från superklass
- Implementerar gränssnitt
- Dubbelriktad eller oriktad relation
- Riktad relation (har, känner till)
- Aggregat (har, består av)
- Komposition (-"-, äger)



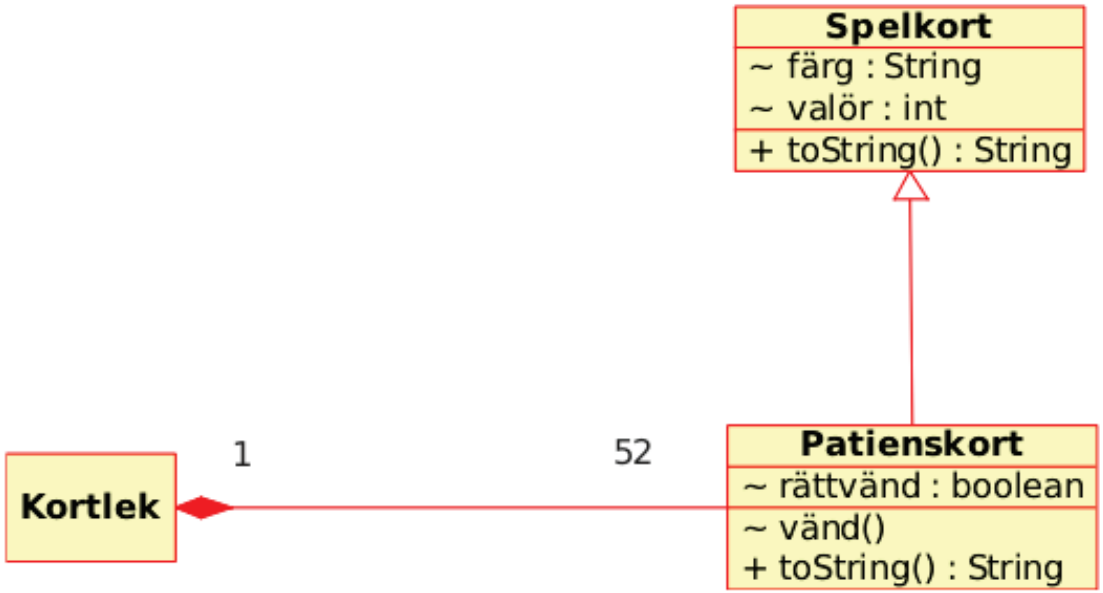
Relationer, exempel



KTH Datavetenskap och kommunikation

Relationer, Exempel 2

KTH Datavetenskap
och kommunikation




```
class Kortlek {
   Patienskort[] kortlek = new PatiensKort[52];
      ...
}
```

Klass eller instans?

- Med hjälp av nyckelordet *static* kan vi bestämma att en metod eller ett fält ska tillhöra själva klassen i stället för en specifik instans av klassen (ett objekt)

- Instansmetoder kan komma åt instansvariabler och -metoder direkt
- Instansmetoder kan komma åt klassvariabler och -metoder direkt
- Klassmetoder kan komma åt klassvariabler och -metoder direkt

men

- Klassmetoder kan **inte** komma åt instansvariabler och -metoder direkt

KTH Datavetenskap och kommunikation

Klass/instans, ett exempel

KTH Datavetenskap
och kommunikation

```
public class Exempel{
    private int a = 5;
    static int b = 3;

    public int getA(){
        return a;
    }

    public int getB(){
        return b;
    }

    public static int getStaticB(){
        return b;
    }
}
```

```
public class FelaktigtExempel{

    // OBS! denna kod är
    // felaktig !!

    private int a = 5;
    static int b = 3;

    public static int getA(){
        return a;
    }
}
```

non-static variable a cannot be referenced from a static context

När bör man använda *static?*

KTH Datavetenskap
och kommunikation

- Egenskaper som är gemensamma för alla objekt
  - ex: räknare för antalet instanser
- Konstanter (kan deklareras `static final` för att förhindra att de ändras av misstag)
  - ex: Math.PI, Math.E
- Metoder som utför operationer med enkla variabler
  - ex: Math.cos(double a), Math.round(double a)
- Metoder som fungerar oberoende av om objekt av en viss klass har skapats
  - ex: Map.konverteraKmTillMiles(double km)

Arv

- En klass **B** som ärver från en annan klass **A** får automatiskt samma fält och metoder som **A**, om man inte deklarerar om dessa i **B**. I **B** kan man lägga till ytterligare fält och metoder, eller definiera om de gamla, sk *överskuggning*
- Om man i **B** vill använda en metod så som den definierades i **A**, kan man ange detta explicit med hjälp av nyckelordet *super*, som `super.görNågot()`
- I **B**:s konstruktor(er) kommer automatiskt **A**:s argumentlösa konstruktor att anropas om man inte explicit anger något annat.
- Kom ihåg att om **A** har fält som är privata, så kommer **B** inte att kunna anropa dessa direkt!

Gränsnitt (Interface)

- Gränssnitt kan liknas vid en superklass som inte innehåller några konkreta metoder eller instansvariabler
- Gränsnitt kan ärva från **flera** andra gränsnitt
- Alla metoder i ett gränsnitt blir automatiskt *publika*
- exempel (från Java Tutorial):

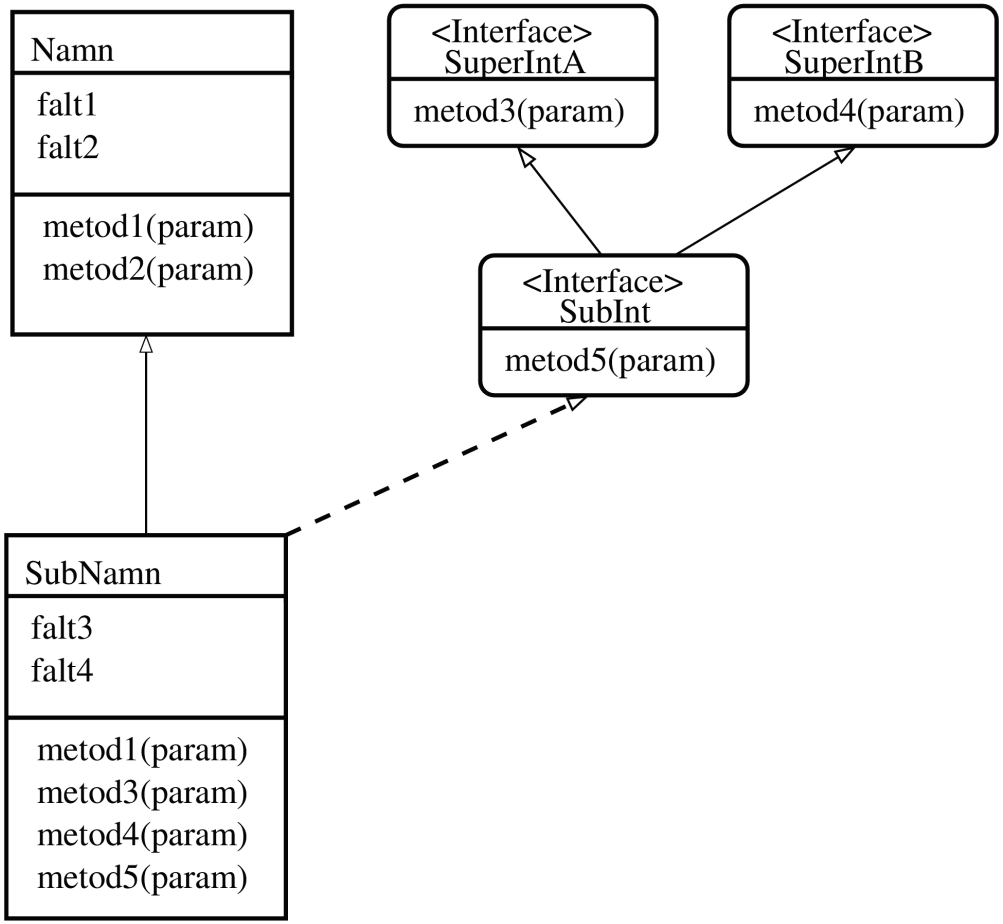
```
public interface GroupedInterface extends Interface1,
                                          Interface2,Interface3 {

   // constant declarations
   double E = 2.718282;  // base of natural logarithms

   // method signatures
   void doSomething (int i, double x);
   int doSomethingElse(String s);
}
```

KTH Datavetenskap och kommunikation

Gränsnitt och arv



KTH Datavetenskap och kommunikation

Nästade klasser

- I Java går det att deklarera en klass inuti en annan klass
- Vi kallar detta för att en *yttre klass* innehåller en *inre klass*.
- Det finns fyra sorters inre klasser
  - en statisk nästad klass fungerar som en vanlig klass definierad i samma paket, men finns i samma fil
  - en inre klass kan bara instansieras i en instans av den yttre klassen, och har tillgång till alla delar av denna instans, inklusive privata fält och metoder
  - en lokal inre klass definieras inuti ett annat kodblock, och kan sedan användas fritt inuti detta block
  - en anonym inre klass definieras och instansieras vid ett specifikt tillfälle och kan inte användas någon annanstans

KTH Datavetenskap och kommunikation

Abstrakta klasser

KTH Datavetenskap
och kommunikation

- Deklareras med nyckelordet *abstract*
- Går inte att instansiera, utan är en mall att ärva från
- En klass som har minst en metod som är deklarerad som *abstract* måste själv vara abstrakt
- För att en ärvande klass ska gå att instansiera måste den explicit deklarera alla metoder som är abstrakta i superklassen, annars blir den ärvande klassen också abstrakt
- Gränssnitt och deras metoder är implicit abstrakta
- Abstrakta klasser behöver inte implementera alla metoder ur gränssnitt
- Abstrakta klasser kan ha statiska fält och metoder

```
public abstract class MyAbstractClass {
    abstract void doAbstractStuff();
}
```

Abstrakta klasser & gränssnitt

KTH Datavetenskap
och kommunikation

- Om man deklarerar några instansvariabler och konkreta metoder, men inte vill specificera något konkret, bör man använda en abstrakt klass
- Om man inte har några instansvariabler, och alla metoder är abstrakta, bör man använda ett gränsnitt

- Som exempel kan vi tänka oss
  - den abstrakta klassen `Kort`, som innehåller information om storlek, och har en gemensam metod för att lägga korten i en hög, med konkreta subklasser som är t.ex spelkort, visitkort, mm
  - gränsnittet `Flippable`, som gäller för alla kort (och andra föremål) som går att vända på

Polymorfism

- Man kan deklarera variabler av en superklass och instansiera dem med en subklass

```
Human putte = new Fysiker();
```

- Java kommer att behandla variabeln som den deklararerade typen för att avgöra om en metod eller ett fält finns.
- Java kommer att behandla dem som den instansierade klassen för att avgöra hur metoden eller variabeln skall tolkas, sk *dynamisk bindning*.
- Statiska fält och variabler tas från variabeltypen, oavsett instansiering

KTH Datavetenskap och kommunikation

Designmönster

KTH Datavetenskap
och kommunikation

- Färdiga "recept" för att lösa (del-)problem i strukturering av ens program
- Mönster kan beskriva små komponenter eller stora strukturer
- Idéen kommer ursprungligen från arkitekturvärlden på 70- och 80-talet, där man använde funktionella beskrivningar av olika komponenter för att sätta ihop mönsterlösningar
- Formulerades för datalogin 1994 i boken "Design Patterns", av Gamma, Helm, Johnson och Vlissides (GoF)
- Har sedan utvecklats och blivit en genomgående tema i framför allt objektorienterad programmering och "software engineering"

- Exempel på hur man inte bör göra kallas "Anti-Patterns" efter en bok med samma namn av Brown *et al*

Olika sorters mönster

KTH Datavetenskap
och kommunikation

- *Creational Patterns* - beskriver olika sätt att skapa objekt på
- *Structural Patterns* - beskriver olika sätt att realisera relationer mellan objekt
- *Behaviorial Patterns* -beskriver olika sätt att implementera kommunikation mellan objekt
- *Architectural Patterns* - beskriver olika sätt att strukturera program
- *Concurrency Patterns* - beskriver olika sätt att hantera samtidighet och turordningar i flertrådade program

Singleton

KTH Datavetenskap
och kommunikation

- *Creational Pattern*
- Ett objekt som det bara finns (kan finnas) en enda instans av, ex filsystem, fönsterhanterare, kontrollobjekt (se MVC)
- Gör det lättare att göra rätt genom bra design
- Dålig lösning: skriv i dokumentationen att man bara för skapa ett objekt av denna klass en gång
- Bättre lösning: Gör så att det bara går att skapa ett objekt, försök att skapa fler går inte ens att kompilera
- Gör det lättare att göra rätt genom att göra det omöjligt att göra fel!

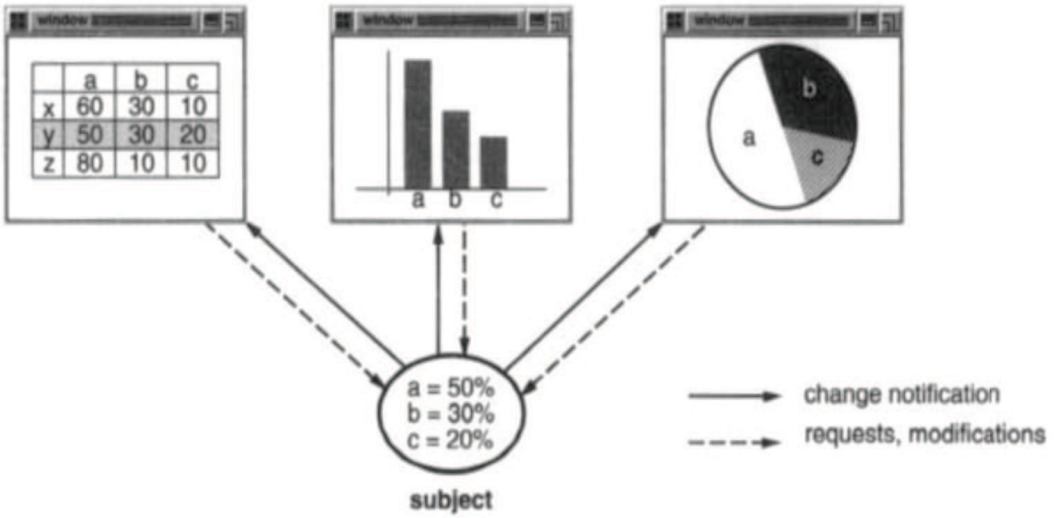
Model View Controller (MVC)

- Ett arkitekturmönster för interaktiva program
- Ett av de äldsta mönstren, (Trygve Reenskaug 1979)
- Man separerar programmet i tre funktioner
  - **Model** innehåller en model av processen, t.ex en varukorg på en e-handelsida, data och formler för en fysikalisk simulering, textdokumentet och avstavningsregler i en ordbehandlare
  - **View** presenterar resultatet för användaren, genererar t.ex hemsidan för en e-handelsida, ritar grafer för simuleringen, eller visar upp dokumentet på skärmen i ordbehandlaren
  - **Controller** styr processen, t.ex genom att säga åt modellen att uppdatera varukorgen när någon trycker på "köp"-knappen, säger åt simuleringen att börja när användaren trycker på "start", eller genom att säga åt radbrytaren att avstava texten när nya tecken skrivs in. Kan även säga åt view att uppdatera bilden

KTH Datavetenskap och kommunikation

Observer (GoF)

- Man definierar ett "ett-till-många"-förhållande mellan objekt så att när ett objekt byter tillstånd så uppdateras alla beroende objekt automatiskt
- Man slipper att explicit uppdatera alla inblandade objekt
- Tillåter lösare koppling mellan objekt, så att dessa lättare kan återanvändas
- exempel: flera *View*-objekt observerar ett *Model*-objekt



KTH Datavetenskap och kommunikation

Factory (GoF)

KTH Datavetenskap
och kommunikation

- *Creational Pattern*
- I stället för att skapa objekt med en publik konstruktor så har man en eller flera publika *factory*-metoder som returnerar objekt av den önskade typen
- Fabriksmetoden kan inkludera kod som bedömmer vilken klass av objekt som ska returneras
- Fabriksmetoden kan bestämma om ett nytt objekt alls ska instansieras, eller om man ska returnera en referens till ett som redan finns (jmf Singleton)
- Man kan ha flera fabriksmetoder med samma argumentsignatur

Builder (GoF)

KTH Datavetenskap
och kommunikation

- *Creational Pattern*
- Man separerar ett objekts representation och dess konstruktion
- En *Builder* är ett objekt som kan sätta ihop andra objekt, steg för steg
- Vi har sett ett exempel, `StringBuilder`
- Det objekt som anropar de olika stegen i Builder-objektet brukar kallas för *Director* (regissör)

Builder

KTH Datavetenskap
och kommunikation

- Man kan konstruera ett objekt i lugn och ro, utan att vara begränsad av objektets interna representation, och när man är nöjd skapas ett objekt av den önskade typen
- Många klasser som har en tillhörande Builder-klass har en konstruktor som tar ett Builder-objekt som argument.

```
public String(StringBuilder sb);
```

- Man kan ha en struktur i det färdiga objektet som är mer effektiv, och en struktur i Buildern som är mer flexibel.
- För att konstruera ett lite mer komplext objekt kan *Director*-objektet anropa ett stort antal olika metoder i Buildern, men man riskerar inte att någon annan tråd kommer åt ett halvfärdigt objekt innan man är klar
- En Builder kan ta ett färdigt objekt som argument i konstruktorn och använda det för att bygga ett nytt

Iterator (GoF)

KTH Datavetenskap
och kommunikation

- *Behaviorial Pattern*
- Iteratorer är objekt som tillhandahåller ett sätt att stega sig igenom element i en *Collection*
- Ett sätt att möjliggöra algoritmer som verkar på datastrukturer utan att man vet hur strukturen ser ut.
- I Java kan alla klasser som implementerar *Collection* returnera ett Iterator-objekt som man sedan kan använda för att stega sig igenom alla element.

Composite (GoF)

KTH Datavetenskap
och kommunikation

- *Structural Pattern*
- Låter en behandla grupper av objekt på samma sätt som ett enskilt objekt
- Passar när man har en trädstruktur och vill göra exakt samma sak med en nod och alla dess barn
- Definiera en abstrakt *Component*-klass som rymmer alla de metoder man vill kunna anropa, en struktur för att rymma barn-noder, samt metoder för att lägga till barn och komma åt dessa
- Nu kan man definiera en *Leaf*-klass och en *Composite*-klass som ärver från *Component*
- Ett program som arbetar med dessa strukturer behandlar alla delar som om de vore *Component*-objekt, och ett anrop av någon metod utförs för samtliga *Leaf*-objekt nedanför i hierarkin

Kloning

KTH Datavetenskap
och kommunikation

- Att skapa en exakt kopia av ett objekt
- En kopia kommer att ha alla fält satta till samma värden som originalet
- Om **A** är en kopia av **B**, gäller *oftast* att:

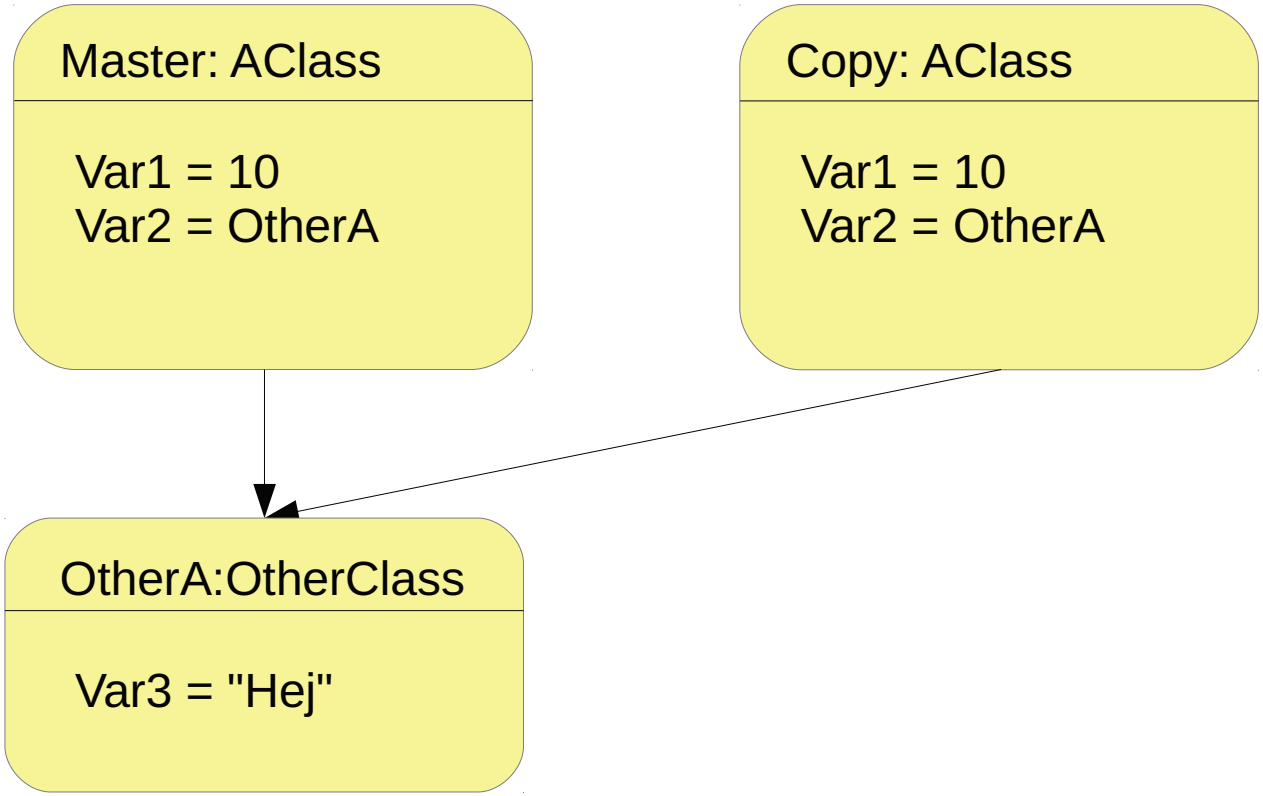
```
A == B        = false
A.equals(B)   = true
```

- I Java finns metoden clone() definierad i klassen `Object`, och är alltså tillgänglig i alla klasser

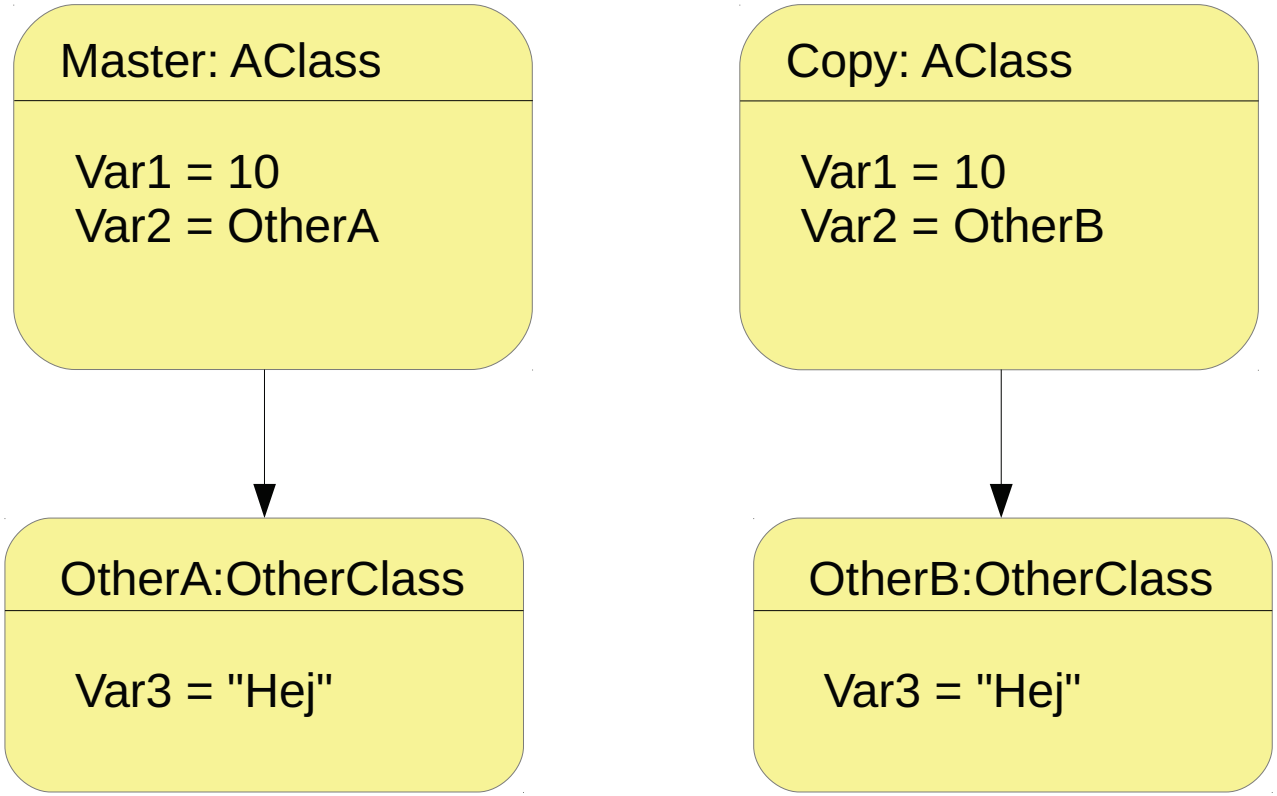
men

- I `Object` är metoden definierad så att den kastar en `CloneNotSupportedException` om inte den aktuella klassen implementerar gränssnittet `Cloneable`, och definierar en egen publik `clone()`-metod

Grund kopia


Master: AClass
Var1 = 10
Var2 = OtherA
Copy: AClass
Var1 = 10
Var2 = OtherA
OtherA:OtherClass
Var3 = "Hej"

Djup kopia


Master: AClass
Var1 = 10
Var2 = OtherA
Copy: AClass
Var1 = 10
Var2 = OtherB
OtherA:OtherClass
Var3 = "Hej"
OtherB:OtherClass
Var3 = "Hej"

Prototype (GoF)

KTH Datavetenskap
och kommunikation

- *Creational Pattern*
- Man börjar med att skapa en objekt som får vara prototyp, och sedan klonar man detta när man vill skapa nya objekt.
- Kan användas för att förse fält med startvärden som kan ändras under programmets körning
- Om det är beräkningsmässigt tungt att räkna ut startvärdena, eller initiera de objekt som skall innehållas, kan man göra detta en gång för alla i prototypen, för att sedan kopiera resultaten till de nyskapade objekten
- Kan med fördel kombineras med mönstret **Factory**

Flyweight (GoF)

KTH Datavetenskap
och kommunikation

- *Structural Pattern*
- Ger ett sätt att minska mängden resurser (minne) som behövs av varje instans av en klass
- Så mycket som möjligt av informationen i objektet lagras utanför objektet, på en plats som delas med andra objekt
- Exempel:

ett tecken i en ordbehandlartext innehåller ett fält som säger vilket tecken det ska vara, några fält med information om fontstorlek, huruvuda det är kursivt, fetstil eller dyl. Själva den grafiska informationen av hur tecknet skall se ut kan dock delas med alla andra objekt som representerar samma tecken.

Adapter (GoF)

- *Structural Pattern*
- Kallas även för *Wrapper*
- Gör det möjligt för två klasser (**A**, **B**) att interagera trots att de inte har explicit kompatibla gränssnitt
- Erbjuder det nödvändiga gränssnittet till det anropande objektet **A**, och anropar sedan **B** med **B**:s egna gränssnitt
-

KTH Datavetenskap
och kommunikation

Facade (GoF)

KTH Datavetenskap
och kommunikation

- *Structural Pattern*
- Tillhandahåller ett enklare gränssnitt till en större mängd kod, t.ex ett eller flera paket
- Kan användas om man ofta gör flera komplicerade operationer på flera olika objekt, men egentligen bara är intresserad av resultatet
- Möjliggör högnivåanrop
- Ger ingen detaljkontroll över de objekt eller metoder som ingår

- Exempel: GUI-objekt i Swing, Systemanrop, Kompilator

Proxy (GoF)

KTH Datavetenskap
och kommunikation

- *Structural Pattern*
- Skapar en ersättare **A** för ett objekt **B** som är svårt eller omöjligt att komma åt för tillfället
- Proxyn **A** har samma gränssnitt som objektet **B** som det är ställföreträdare för
- **Remote Proxy** är en ställföreträdare för ett objekt som finns någon annanstans
- **Virtual Proxy** är en ställföreträdare för ett objekt som inte har skapats/initialiserats än
- **Protection Proxy** kontrollerar åtkomster och skyddar objekt som inte ska gå att komma åt

TCP/IP

KTH Datavetenskap
och kommunikation

- TCP - Transmission Control Protocol
- Kontrollerar att alla datapaket kommer fram, skickar nya paket om inget svarsmeddelande kommer tillbaka
- Garanterar ordningen, dvs, alla paket kommer fram i samma ordning som de skickades
- Plockar bort dubletter
- Lämplig för t.ex filöverföring
- Kan ibland få fördröjningar på flera sekunder om borttappade paket behöver skickas om flera gångar

UDP

- User Datagram Protocol
- Enkelt protokoll utan felkontroller
- Garanterar inte att alla paket kommer fram
- Meddelar inte om paket kommit bort
- Garanterar inte att alla paket kommer i rätt ordning
- Plockar inte bort dubletter
- Antar att applikationen gör alla felkollar som behövs
- Lämplig för t.ex realtidsapplikationer

KTH Datavetenskap och kommunikation

Socket

- En slutpunkt för en internetanslutning
- Kan användas av processer för att skriva till/läsa från nätverket
- Initieras av användarprocesser, administreras av operativsystemet
- Identifieras unikt genom
  - *Local Socket Address* (lokal IP och port)
  - *Remote Socket Address* (TCP:motpartens IP och port)
  - *Protocol* (t.ex TCP, UDP)
- När kontakt har etablerats kan man skriva och läsa mellan två processer på två olika datorer

KTH Datavetenskap
och kommunikation

Processer

KTH Datavetenskap
och kommunikation

- Vad vi i dagligt tal menar när vi pratar om ett program som kör
- En process har referenser till en mängd reserverat minne
- En process har referenser till systemresurser, som t.ex filer, sockets, mm
- Ett operativsystem har (minst) en *scheduler* som ansvarar för att tilldela olika processer tid i processorn för att exekvera sina instruktioner
- Processer har olika prioritet, utifrån vilken turordningen i schedulern bestäms
- Det går att kommunicera mellan processer, men det är ungefär lika bökigt inom en dator som mellan två olika datorer
- I **vanliga fall** körs javamotorn som **en** process, men *det går att skapa flera ny processer inifrån ett javaprogram*

Trådar

KTH Datavetenskap
och kommunikation

- En process är i sin tur uppdelad i trådar (minst en)
- Trådarna kan exekvera parallellt med varandra
- Varje tråd kan schemaläggas separat av schedulern, utifrån en given prioritetsordning. Vid samma prioritet får trådarna turas om, hur ofta de byts ut varierar med schedulern (och OS:et)
- Alla trådar kan ha tillgång till allt minne och filpekare som hör till processen.
- Det delade minnet gör att en *context switch* ofta går fortare mellan trådar än mellan processer
- Det delade minnet gör att det är lätt att kommunicera mellan olika trådar i samma process
- I Java skapas trådar med klassen `Thread` och gränssnittet `Runnable`

Trådar i Java

KTH Datavetenskap
och kommunikation

- I Java skapas trådar ur klassen `Thread`
- Trådobjektet innehåller kod som körs parallellt med den tråd som startade den
- Koden som ska köras i en tråd finns definierad i dess `run()`-metod
- Trådklassen definierar bl.a
  - `start()`, sätter igång tråden
  - `sleep(int ms)` låter tråden vänta tillfälligt
  - `setPriority(int newPriority)` sätter prioriteten, **har nästan ingen effekt!**
  - `join(int ms)` väntar på att tråden ska bli klar

Runnable

KTH Datavetenskap
och kommunikation

- Ett alternativ till att ärva från klassen `Thread` är att skapa en egen klass som implementerar `Runnable`
- Då måste ens klass implementera metoden `run()`
- Ett trådobjekt kan skapas med en Runnable som argument till konstruktorn, då körs `run()` från Runnable-objektet när `start()` anropas på tråden

- Fördelen med att ärva från `Thread` är att det ofta blir mindre kod att skriva
- Fördelen med att implementera en egen `Runnable`-klass är att man kan ärva från en helt annan klass, om man behöver specialiserad funktionalitet
- `Thread` implementerar `Runnable`

Parallella Problem

KTH Datavetenskap
och kommunikation

- I program med flera parallella exekveringstrådar kan det uppstå problem, fel och andra fenomen som inte förekommer i enkeltrådade program.
- **Thread interference** - två (eller fler) trådar stör varandra
- **Deadlock** - två (eller fler) trådar blockerar varandra. Detta fel uppstår ofta när man försöker lösa ovanstående fel
- **Starvation** - trådar tillbringar all sin tid med att vänta på samma begränsade resurs i stället för at göra nytta
- **Livelock** - trådar reagerar på varandras handlingar på ett sätt som skapar nya reaktioner i oändliga loopar

Programutvecklingsmetodik

KTH Datavetenskap
och kommunikation

- Code-and-fix
- Waterfall
- Spiral
- Rapid Prototyping
- COTS
- Agile methods, t.ex.
  - XP = eXtreme Programming
  - SCRUM

Fortsättningskurser
Programmering och datalogi

KTH Datavetenskap
och kommunikation

- DD1377 Maskinnära programmering och datorarkitektur 6,0 hp, per 4
- DD2387 Programsystemkonstruktion med C++, 6 hp, per 1,2
- DD2352 Agoritmer och Komplexitet, 7,5 hp, per 3,4
- DD2440 Avancerade algoritmer 6,0 hp, per 1,2

- http://www.csc.kth.se/utbildning/kurslistor/12-13/datalogi/
- http://sisu.it.su.se/search/show_subject/18?subject_group=5

Fortsättningskurser, nätverk

- DM2518 Mobilutveckling med webbteknologier 7,5 hp, per 4
- DD2390 Internetprogrammering, 6hp, per 3
- DD2395 Datasäkerhet 6,0 hp, per 1,2
- DD2448 Kryptografins grunder 7,5 hp, per 3,4

KTH Datavetenskap och kommunikation

Fortsättningskurser, AI & robotar

- DD2380 Artificiell Intelligens, 6hp, per 1
- DD2418 Språkteknologi, 6 hp, per 2
- DD2423 Bildbehandling och datorseende 7,5 hp, per 2
- DD2425 Robotics and autonomous systems, 9 hp, per 1,2
- DD2427 Bildbaserad igenkänning och klassificering 6,0 hp, per 4
- DD2429 Datorfotografi 6,0 hp per 1
- DD2431 Maskininlärning 6,0 hp, per 1

KTH Datavetenskap och kommunikation