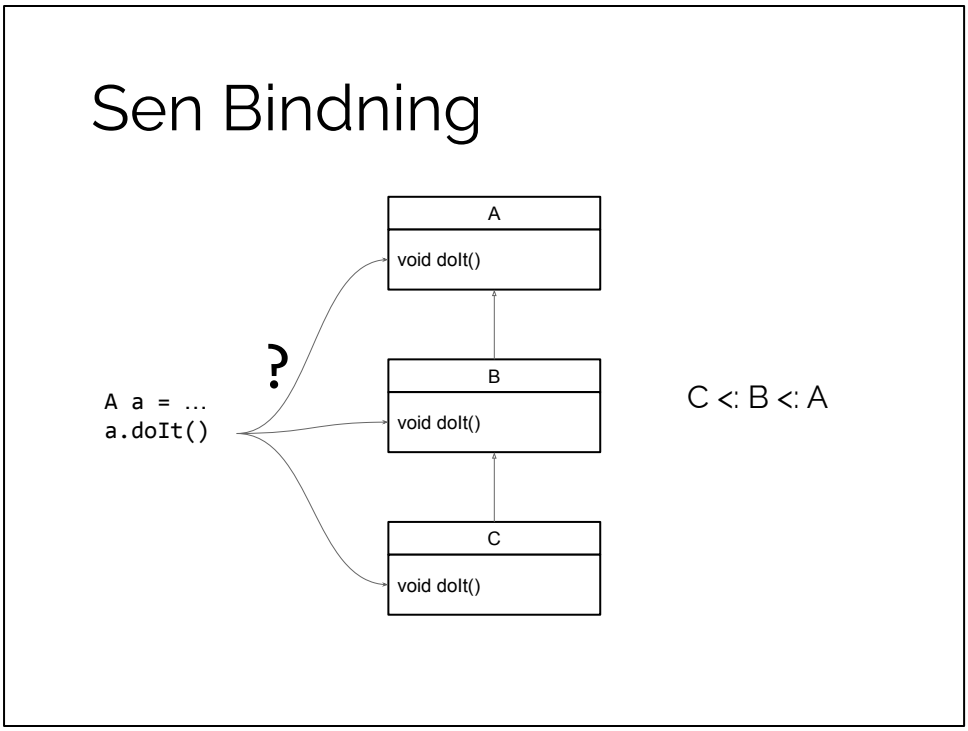
ex3override

Joachim von Hacht

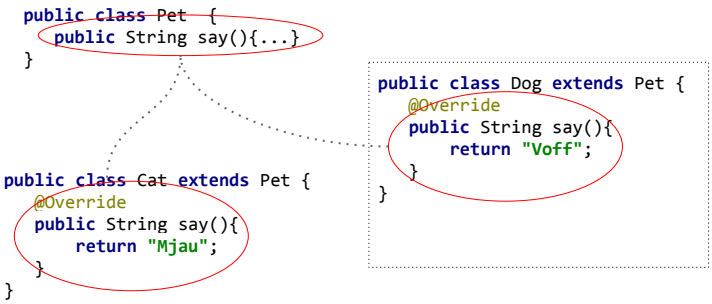


Vi har tidigare sett att vi kan skapa egna versioner av ärvda metoder, kallas överskugga/override.

Antag att vi har som i bilden.

- Klasser B och C överskuggar metoden doIt() från A
- En supertypvariabel (a) kan referera objekt av typen eller någon subtyp …
- … och alla dessa har identiska versioner av metoden.
- Kompilatorn vet inget om värden (objekten) … vilken metod skall den välja för anropet (i byte-koden den skall producera)?
- Svar: Kompilatorn kan inte veta!
    - Kompilatorn noterar (i byte-koden) att det finns flera versioner av metoden och att metoden skall väljas utifrån objektets typ är under körningen.
    - Metod bestäms alltså under körning, kallas sen bindning (late binding).

Överskuggade Metoder




```
Pet p = ...;
out.println(p.say()); // Depends on object type!
```

3

Lite sammanfattning

En **överskuggad (overridden)** metod förekommer bara i samband med arv (kallas override även för gränssnittsarv)

- Innebär att man i subklassen ersätter en ärvd metod från superklassen med en egen version av metoden (identiskt metodhuvud).
    - @Override gör så att kompilatorn kontrollerar att metodhuvudet exakt matchar det ärvda …. annars kan det bli fel av misstag, en säkerhetsåtgärd, mer strax …
- Vilken metod som körs bestäms under körning utifrån objektets typ (inte typen på referensvariabeln).

Varför Överskuggade Metoder?

```
List<Pet> pets = ...;

// Assume NO say()-method in classes
// Non overriding style, if new Pet added have to change code
for (Pet p : pets) {
   if (p instanceof Cat) {
       out.println("Mjau");
   } else if (p instanceof Dog) {
       out.println("Voff");
   }
   ...
}

// Assume overridden method say() in all classes
// Each Pet know what to say, if new pets no change to code!
for (Pet p : pets) {
   out.println(p.say());
}
```

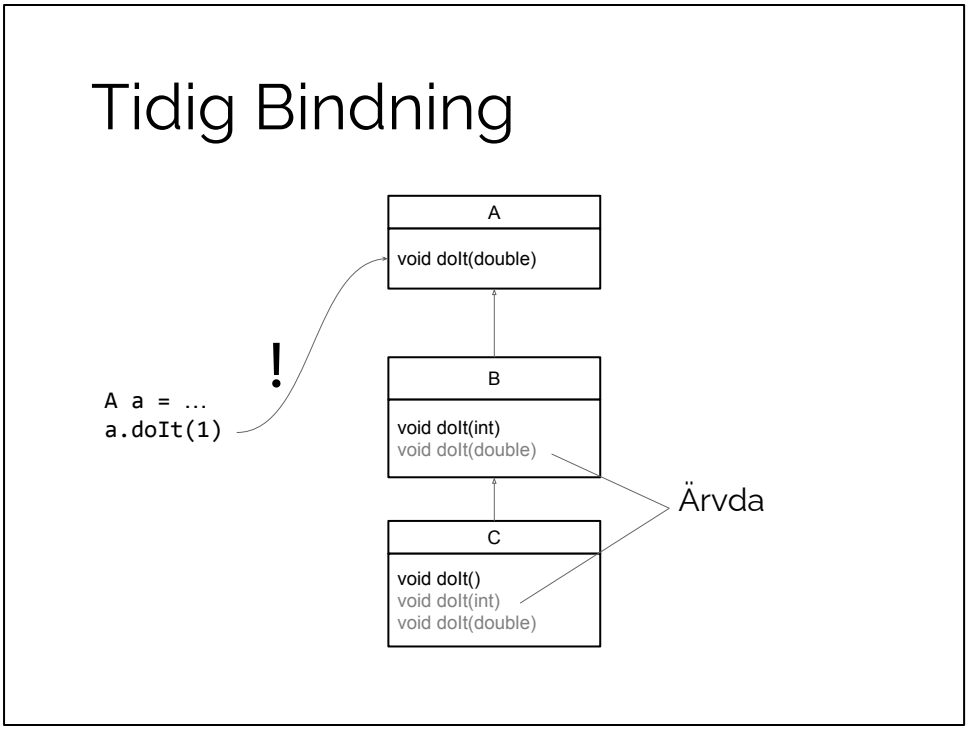
Cat <: Pet and Dog <: Pet

Överskuggning gör att vi kan skriva mer abstrakt kod, ger flexibilitet.

- Alla objekt kan implementera operationerna utifrån sina specifika krav/möjligheter
- Programmet blir utbyggbart med nya objekt (operationerna finns ju i objekten, inte i koden utanför, koden utanför behöver inte ändras)

I bilden:

- instanceof
  - Att använda typinformation under körning blir oflexibelt.
  - Dessutom är risken stor att om vi har en if-sats på ett ställe så finns liknande på flera ställen.
  - Risk att behöva ändra mycket!
- Mycket bättre med överskuggning (nedre loopen).
  - Vill vi utöka programmet lägger vi till ett nytt objekt, inget annat behöver ändras.



Om en metod med samma namn finns i flera klasser i en arvshierarki men inte har identiska metodhuvuden får vi helt enkelt överlagring i subklasserna p.g.a. arv

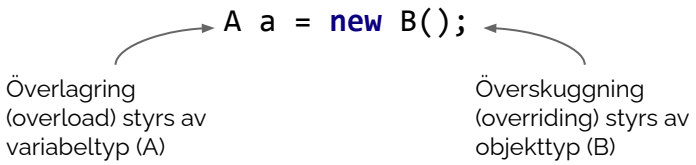
- Inte bra skall undvikas, men kan uppstå av misstag.
- Överlagring handlar om att "samma sak" skall göras men med olika parametrar,
    - Samma sak skall göras i en klass (en klass har ett ansvar)

Typen A i bilden har en metod som matchar anropet (1 kan implicit omvandlas till 1.0). …

- … alltså lägger kompilatorn in detta anrop i byte-koden. Kompilatorn kan redan vi kompileringen bestämma metod, kallas tidig bindning (early binding).
- Hade vi gjort anropet a.doIt() så saknas sådan metod i A. Vi får ett kompileringsfel!
- Hade vi haft B b = new B(), b.doIt(1) så hade den exakt matchande metoden i B valts (doIt(int)).

Överlagring, Överskuggning och Typ

Antag B <: A



Summa: Överlagring (overloading) kontra överskuggning (overriding)

- Överlagring bestäms vid kompilering utifrån variabelns typ (den statiska/deklarerade typen).
- Överskuggning bestäms vid körning utifrån objektets typ (den dynamiska/körningstypen).

super

```
public class FacingCreep extends
                        AbstractFacingCreep {

  @Override
  public void move() {
     super.move(); // First run move in superclass
     facing = getDir(); // Then run this
  }


}
```

7

Bara super i koden förekommer i samband med arv och syftar det direkta basklassobjektet.

- I detta fallet har vi en överskuggad metod men vill i denna köra basklassobjektets metod först.
- super är inte en referens (vilket ju this är)