Objects First With Java
A Practical Introduction Using BlueJ

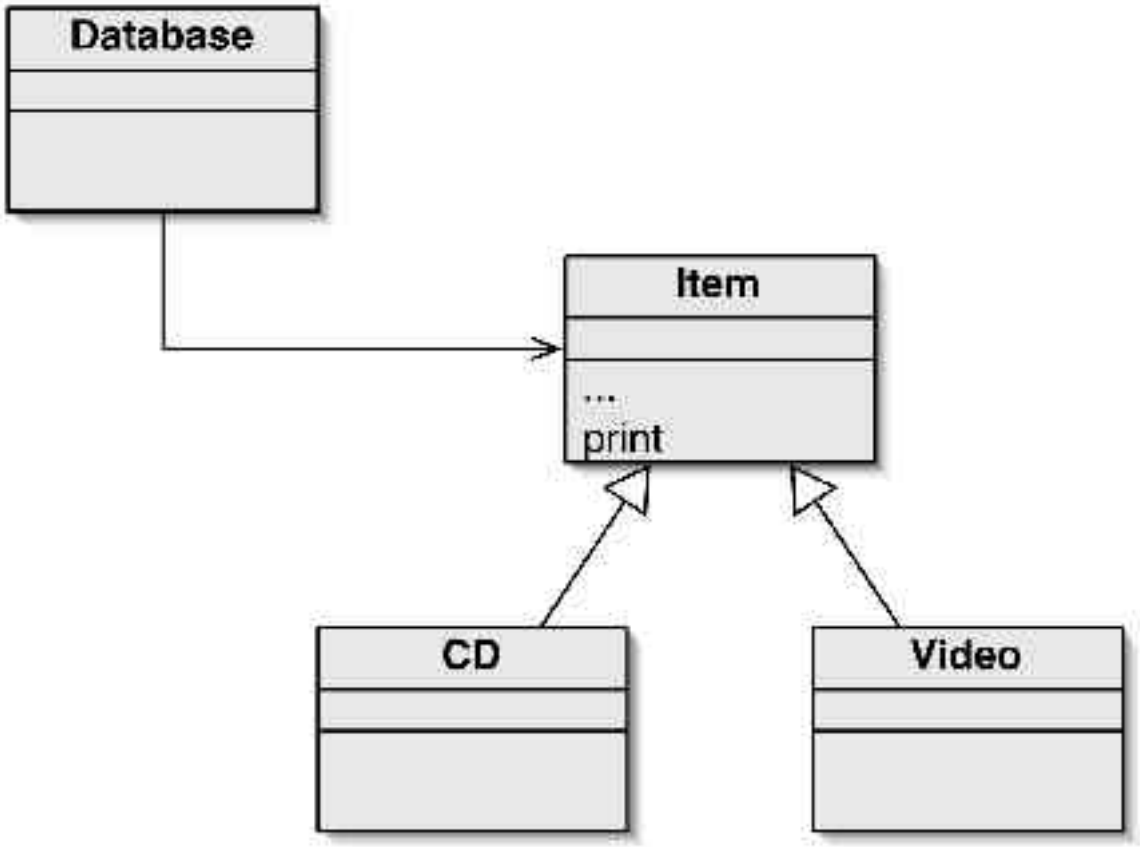
Mer om arv

Att utforska polymorfismen

Fler begrepp

- polymorfa metoder
- statiska och dynamiska typer
- överskuggning (overriding)
- dynamiskt metodval
- skyddad åtkomst

DoMe arvhierarki



Problem vid utskrift


```
CD: A Swingin' Affair (64 mins)*
    Frank Sinatra
    tracks: 16
    my favourite Sinatra album

video: O Brother, Where Art Thou? (106 mins)
    Joel & Ethan Coen
    The Coen brothers' best movie!
```

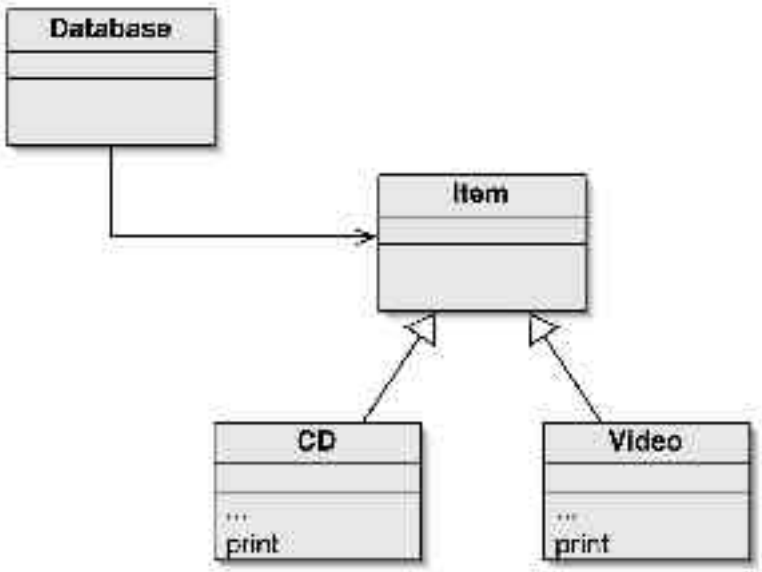

```
title: A Swingin' Affair (64 mins)*
    my favourite Sinatra album

title: O Brother, Where Art Thou? (106 mins)
    The Coen brothers' best movie!
```

Problemet

- **`print`**-metoden i **`Item`** skriver bara ut de gemensamma fälten.
- Arv går bara åt ett håll:
  - En subklass ärver superklassens fält och metoder.
  - Superklassen känner inte till subklassernas fält och metoder.

Hur löser man problemet?



- Placera **`print`** i den klass där alla fält är åtkomliga.
- Varje subklass får sin egen version.
- **`Men: Item:`**s fält är privata. De kan inte refereras från sub-klasserna
- **`Alltså: Vi skulle behöva en print-metod i Item också.`**

Statiska och dynamiska typer

- En mer komplex typhierarki kräver nya begrepp för att kunna beskriva vad som gäller.
- Ny terminologi:
  - statisk typ
  - dynamisk typ
  - metod dispatch/lookup

Statisk och dynamisk typ

Vilken typ har c1?

```
Car c1 = new Car();
```

Vilken typ har v1?

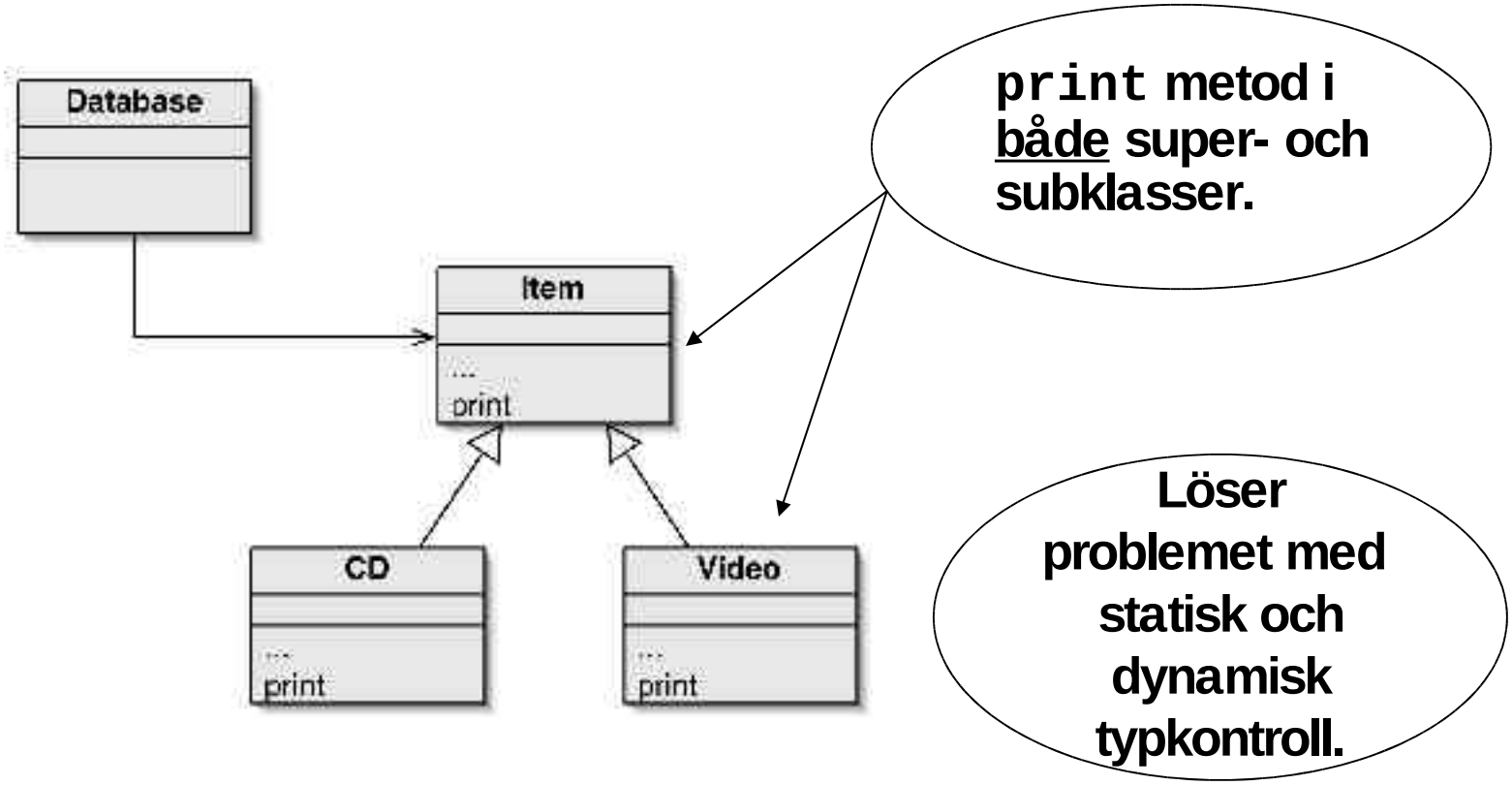
```
Vehicle v1 = new Car();
```

Statisk och dynamisk typ

- Den deklarerade typen för en variabel är dess *statiska typ*.
- Typen på det objekt en variabel refererar är dess *dynamiska typ*.
- Kompilatorn kan upptäcka konflikter som gäller den statiska typen.

```
Item item = (Item) iter.next();
item.print(); // Compile-time error.
```

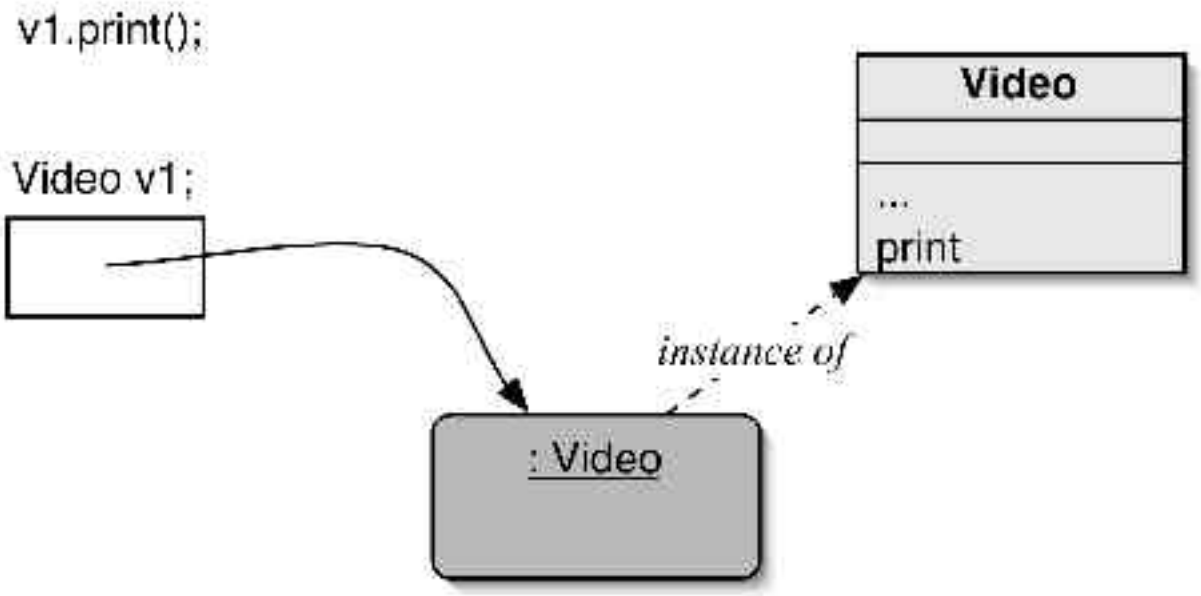
Lösning: Överskuggning!



Överskuggning (Overriding)

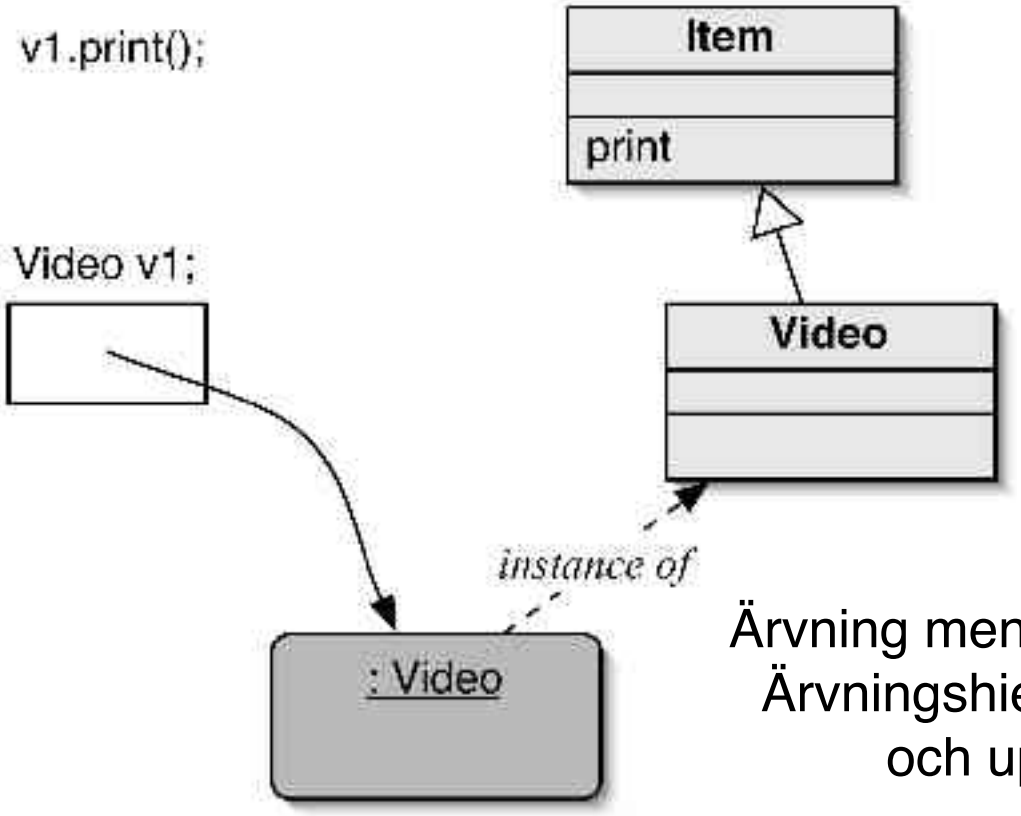
- Superklass och subklass definierar metoder med samma signatur.
- Var och en av metoderna kommer åt fälten i sin klass.
- Metodens förekomst i superklassen innebär att den statiska typkontrollen kan utföras av kompilatorn.
- Subklassmetoden kommer att anropas under exekveringen – den *överskuggar* superklassens version.
- Vad händer då med superklassens version?

Metodval under exekvering



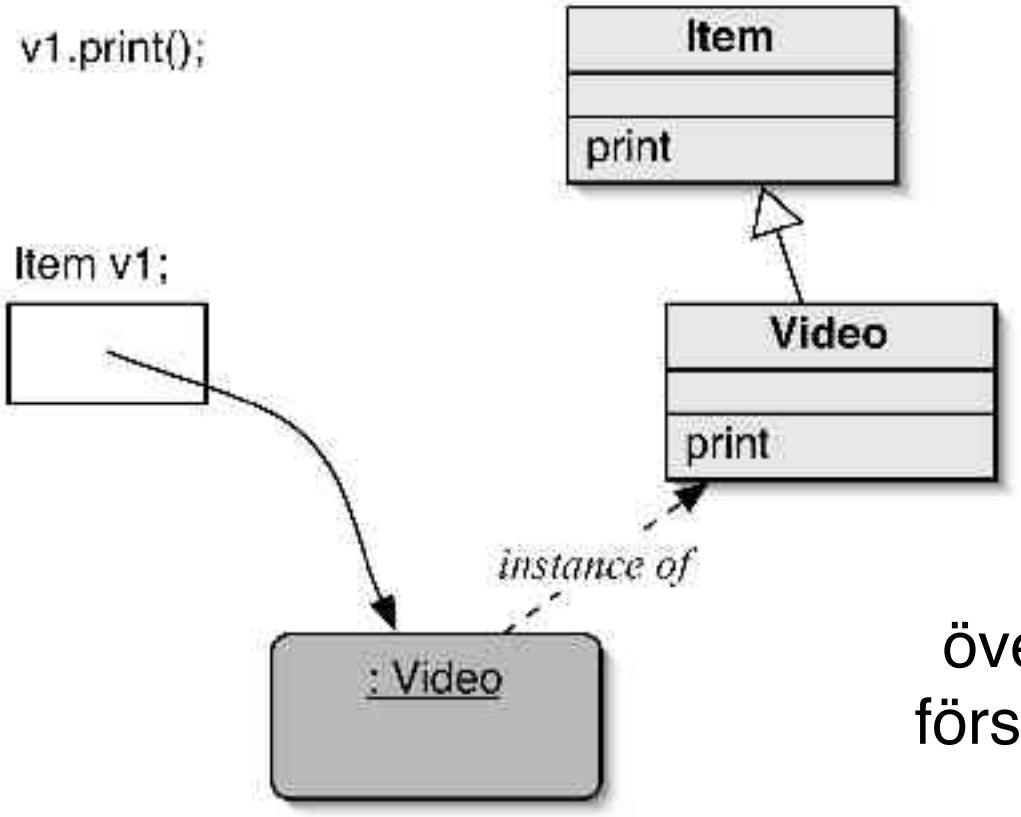
Ingen ärvning eller polymorfism.
Det finns bara en metod att välja.

Metodval under exekvering



Ärvning men ingen överskuggning.
Ärvningshierarkin avsöks nerifrån och upp efter en metod som motsvarar signaturen.

Metodval under exekvering



Polymorfism och överskuggning. Den först funna versionen används!

Summering av metodval under exekvering

- Vi har en variabel som refererar ett objekt.
- Objektet som variabeln refererar letas upp.
- Klassbeskrivningen för objektet letas upp.
- Klassen genomsöks efter metoden.
- Om metoden inte hittas fortsätter sökningen i superklassen.
- Upprepas tills metoden hittas eller vi har genomsökt den översta superklassen.
- Innebär att om det finns en överskuggande metod i en subklass så är det denna som väljs under exekveringen.

Super-anrop i metoder

- Överskuggade metoder i en superklass är inte direkt åtkomliga för anrop ...
- ... men det kan finnas anledning att vilja använda dem.
- Lösning: En överskuggad metod får anropas via **`super`** från den metod som överskuggar den.
  - `super.method(...)`
  - Jämför med användning av `super` i konstruktorer.

print-exemplet

```
public class CD
{
    ...
    public void print()
    {
        super.print();
        System.out.println("    " + artist);
        System.out.println("    tracks: " +
                           numberOfTracks);
    }
    ...
}
```

Anrop av `Items` `print`-metod!

Metodpolymorfism

- Det vi har diskuterat kallas *metodpolymorfism* (eller, alternativt, polymorfiskt metodval).
- En polymorfisk variabel kan lagra objekt av olika typ.
- Metodpolymorfism:
  - Den metod som väljs vid anropet beror på den dynamiska objekttypen.

Metoder i klassen Object

- De metoder som definierats i `Object` ärvs av alla klasser.
- Dessa metoder kan överskuggas.
- Metoden `toString` är ett exempel på en metod som ofta överskuggas:
  - `public String toString()`
  - Returnerar en strängrepresentation av objektet.

Exempel: Överskuggning av toString

```
public class Item
{
    ...

    public String toString()
    {
        String line1 = title +
                       " (" + playingTime + " mins)");
        if(gotIt) {
            return line1 + "*\n" + "    " +
                   comment + "\n");
        } else {
            return line1 + "\n" + "    " +
                   comment + "\n");
        }
    }
    ...
}
```

“Utskrift” av ett objekt

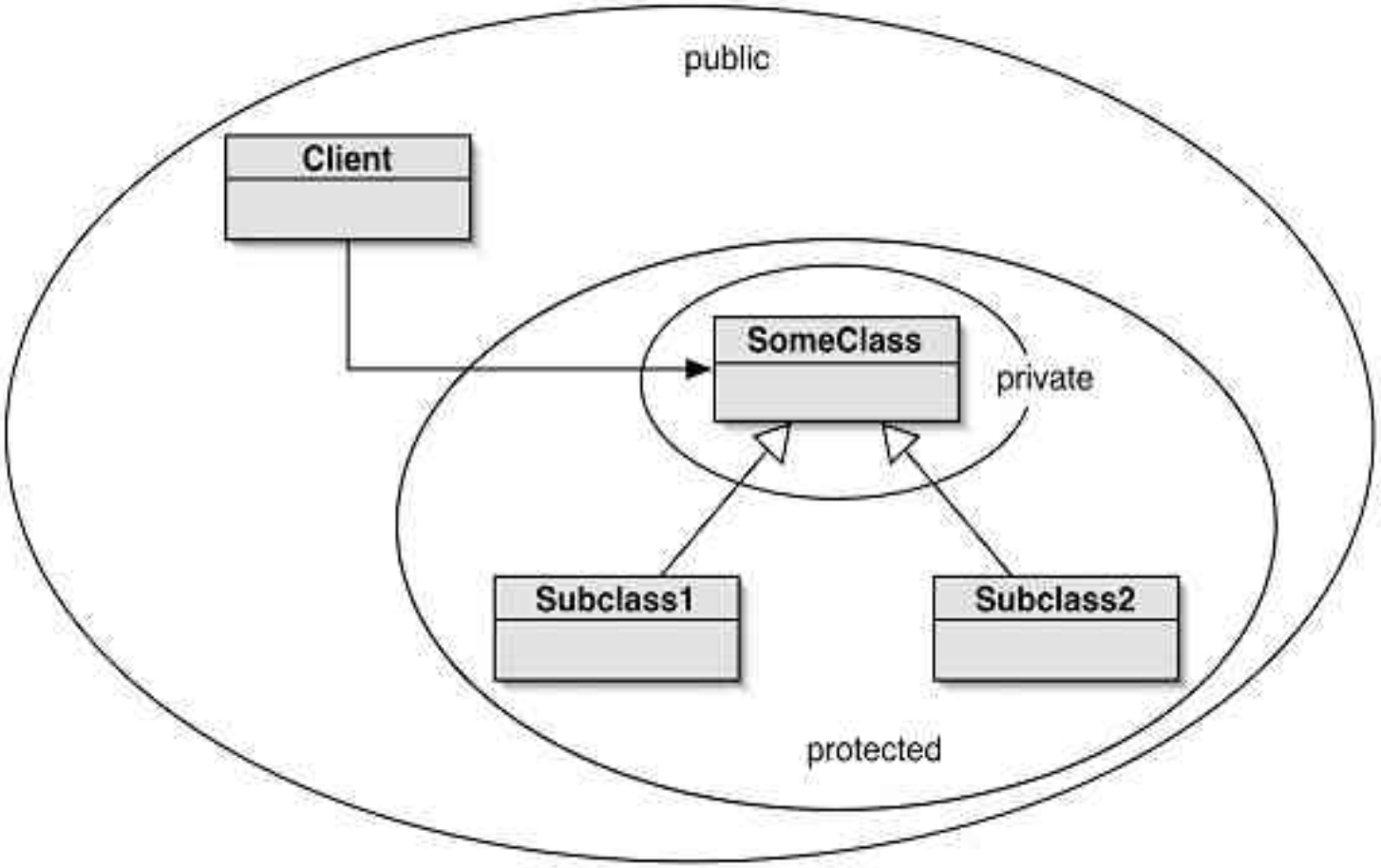
- Ett anrop av **`println`** med ett objekt som parameter innebär att **`toString`** för objektet anropas:

  - **`System.out.println(item);`**

Skyddad åtkomst

- Det är ibland alltför restriktivt att deklarera fält *private* i en klass.
  Man kan vilja att subklasserna (men inga andra klasser) skall kunna använda fälten.
- Åstadkommes genom att deklarera fälten *protected*.
- Men: Rekommendationen är att man deklarerar fälten *private* och i stället deklarerar fältens åtkomst- och ändringsmetoder *protected* (eller *public*).

Åtkomstnivåer



Summering

- En variabels deklarerade typ kallas dess statiska typ.
  - Kompilatorn kan kontrollera statiska typer.
- Om en variabel refererar ett objekt så bestämmer objektets typ variablens dynamiska typ.
  - Den dynamiska typen är alltså något som bara är relevant under exekvering.
- Metoder kan överskuggas (*overriding*) i en subklass.
- När en metod anropas under exekvering letar interpretatorn efter den nerifrån och upp i det objekt som refereras. Innebär att en överskuggande metod i en subklass kommer att väljas.
- *protected* anger att ett fält eller en metod endast kan refereras inom klassen eller från en subklass.