F9 - Polymorfism

ID1004 Objektorienterad programmering

Fredrik Kilander fki@kth.se

Polymorfism - flerformighet

- Vi vet vad metoden heter (signaturen)
- Men vi vet inte vid anropet exakt vilken metod som faktiskt kommer att köras
- Detta beror på vilken klass objektet tillhör

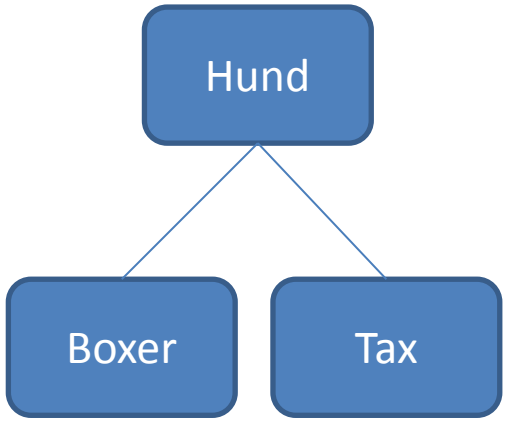
```
Object o = ... ; // Något objekt

String str1 = o.toString();
```

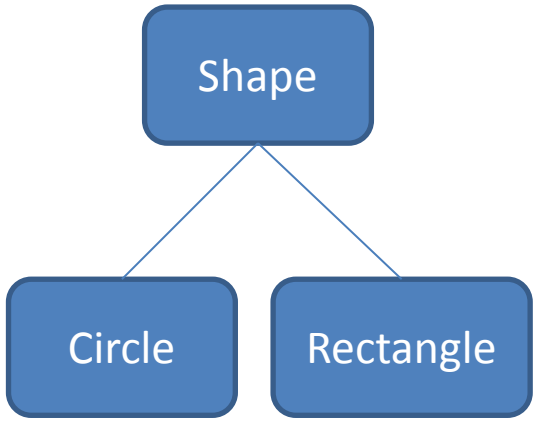
Polymorfism - flerformighet

- Polymorfism genom arv
  - Subklasser ärver och åsidosätter metoder
- Polymorfism genom interface
  - Subklasser har metoder med samma namn (signatur) men eget beteende

Polymorfism genom arv

- Vi betraktar ett objekt som lite mer generellt
- t ex: Boxer är en Hund
- t ex: Circle är en Shape

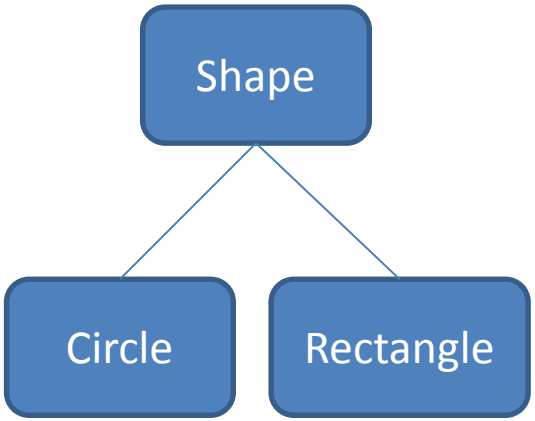




Polymorfism genom arv

- En variabel av typen Shape kan referera till både Circle och Rectangle

```
Shape myShape = null;
...
myShape = new Circle();
...
myShape = new Rectangle();
```



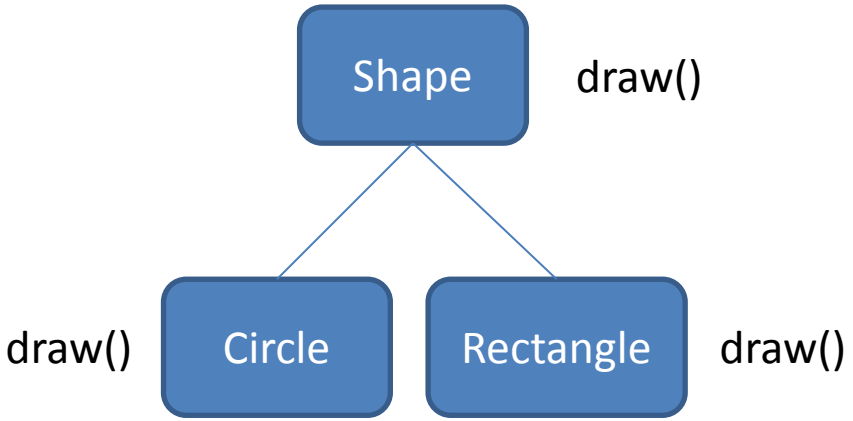
Polymorfism genom arv

- Med en referens till en generellare typ kan typens subklasser behandlas uniformt
- Vi kan använda samma kod, oavsett om det är en Circle, Rectangle, Triangle, etc.
- Men, bara använda det som syns i den generella typen

Polymorfism genom arv

- Alla Shape har en egen version av draw()
- Vilken som anropas beror på instansens klass

```
Shape myShape = null;
...
myShape = new Circle();
myShape.draw();
...
myShape = new Rectangle();
myShape.draw();
```



Polymorfism med interface

- En klass kan implementera ett (eller flera) interface
- I och med deklarationen så måste alla metoder i interfacet vara med

```
class Person implements Comparable {
  ...
  public int compareTo(Person p) {
    ...
  }
  ...
}
```

```
class Car implements Comparable {
  ...
  public int compareTo(Car c) {
    ...
  }
  ...
}
```

Polymorfism med interface

- Med en referens till en interface-typ kan alla implementerande klasser behandlas uniformt
- Vi kan använda samma kod för alla
- Men, bara använda det som syns i interfacet

```
void sort(Comparable [] ar) {
  ...
  if (ar[i].compareTo(ar[j]) < 0) {

  }
  ...
}
```

Kort om sortering

- *Valet av sorteringsmetod beror på:*
- Vilka implementationer som finns
- Hur mycket data som ska sorteras
- Går det att sortera i minnet eller måste man ut på disk?
- Hur ofta måste det sorteras?
- Hur lång tid får det ta?

Bubble sort

- Byt plats på intilliggande element tills dess att inga byten har gjorts
- $O(n^2)$
- Pinsamt ineffektiv

Selection sort

- För varje element *e* i listan, finn det minsta elementet *g* i resten av listan, och om g<e byt plats på *e* och *g*

- $O(n^2)$

- I praktiken effektivare än bubble sort.
- Endast för små datamängder pga $O(n^2)$

Insertion sort

- Tag ut varje element ur listan och stoppa in det på rätt plats i den sorterade dellistan som växer från början
- $O(n^2)$
- Effektiv för korta listor (n<=10)

Quicksort

- Välj ett pivot-värde p (t ex det mittersta elementets värde)
- flytta alla element mindre än p till vänster om p, och element större än p till höger om p
- anropa Quicksort(vänster)
- anropa Quicksort(höger)
- O(n log n)

Sortering i Java

- Använd inbyggda rutiner
  - java.util.Arrays.sort(E [])
  - java.util.Collections.sort(List<T> list)

- Använd Collections (samlingar) som håller objekten sorterade, t ex:
  - java.util.TreeSet
  - java.util.TreeMap

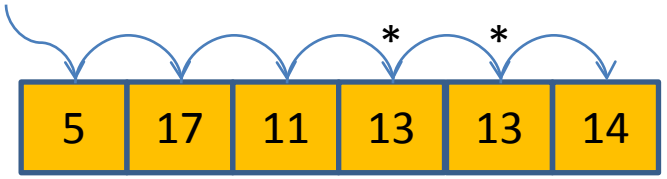
Sökning

- Att hitta ett visst element i en datastruktur
- En oordnad datastruktur tvingar oss att inspektera alla element
- Ibland finns det bara ett rätt element
- Ibland räcker det att hitta det första
- Ibland vill man hitta alla matchande element
- Ibland finns inget matchande element

Linjär sökning – alla träffar

- Titta i alla element

```
List<Integer> indexOf(int [] nums, int key) {

  ArrayList<Integer> hits = new ArrayList<Integer>();

  for (int i = 0; i < nums.length ; i++) {
    if (key == nums[i]) {
      hits.add(i);
    }
  }
  return hits;
}
```

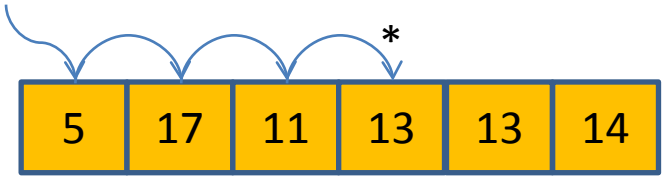


Antal jämförelser: N, N = antal element att söka igenom

Linjär sökning – första träffen

- Titta i alla element tills det blir träff

```
int indexOf(int [] nums, int key) {

  for (int i = 0; i < nums.length ; i++) {
    if (key == nums[i]) {
      return i;
    }
  }
  return -1;
}
```

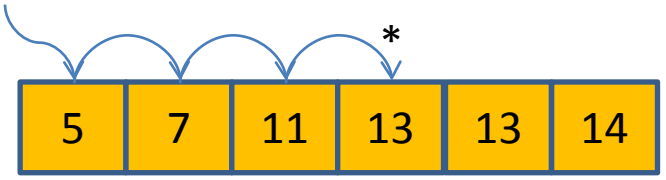


Förväntat antal jämförelser: N/2 (funnen) N (ej funnen), N = antal element att söka igenom

Linjär sökning – sorterad data

- Sluta söka när ingen träff kan finnas

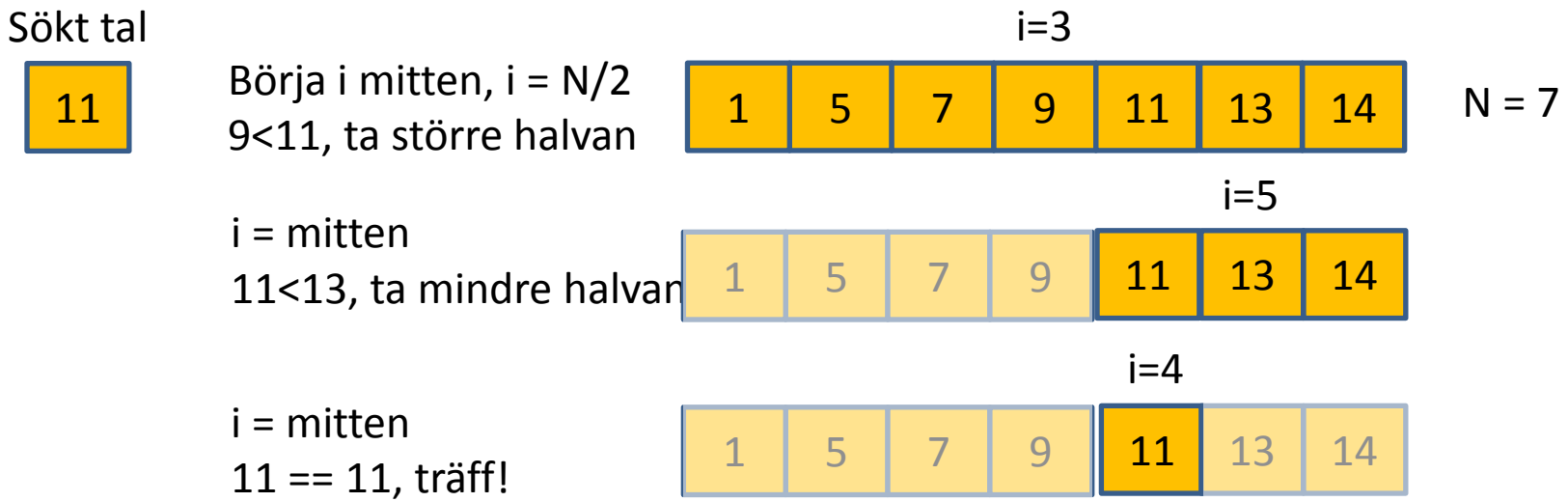
```
int indexOf(int [] nums, int key) {

  for (int i = 0; i < nums.length ; i++) {
    if (key < nums[i]) {
      continue; // För litet, nästa
    } else if (key == nums[i]) {
      return i; // Träff
    } else {
      return -1; // För stort, finns inte
    }
  }
  return -1;
}
```



Förväntat antal jämförelser: N/2, N = antal element att söka igenom

Binärsökning – sorterad data

- Halvera sökrymden vid varje jämförelse



Antal jämförelser: log2(N), N = antal element att söka igenom

Binärsökning – sorterad data

- java.util.Collections.binarySearch(list, key)

- Nyckeln och elementen måste vara jämförbara med *varandra*

```
Integer.compareTo(Integer) // Helt ok
String.compareTo(String)   // Helt ok
Integer.compareTo(String)  // Går inte!
```

De måste vara ömsesidigt jämförbara *(mutually comparable)*

Designa för polymorfism

- Polymorfism kan ge generell kod
- Kod som kan användas till flera klasser underlättar återanvändning
- Onödiga detaljer och egenheter kapslas in
- Renare design och tydligare struktur

”... a consistent approach to inconsistent behaviour.”
Lewis & Loftus, 6th ed, pg 546

Designa för polymorfism

- Medvind:
  - Enkla operationer och algoritmer
  - Liknande egenskaper
- Motvind:
  - Komplexa algoritmer, många specialfall
  - Heterogenitet

Var finns polymorfismen?

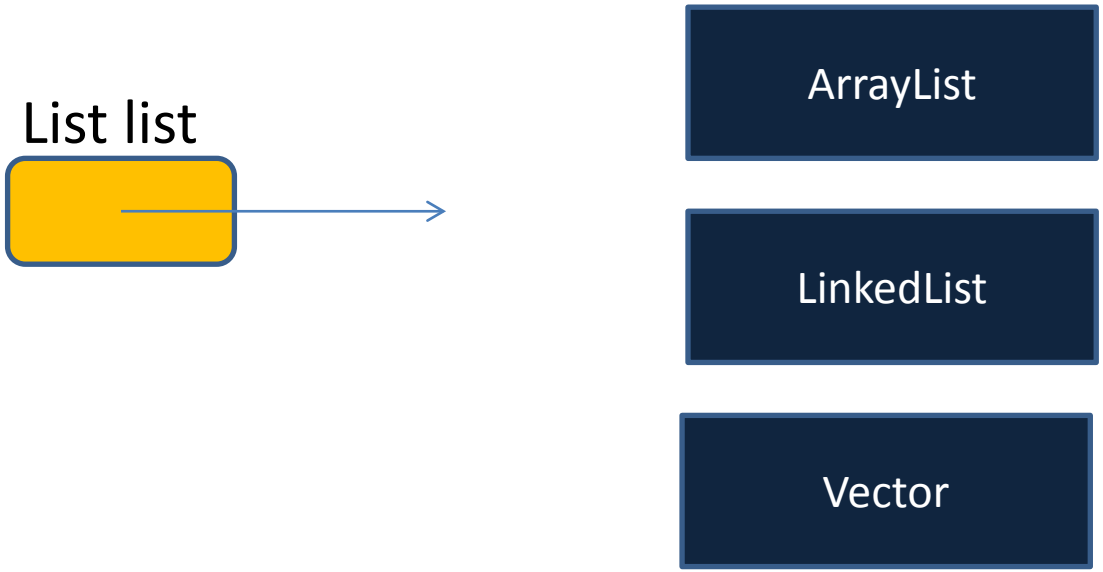
- Olika typer av fordon rör sig på olika sätt

Var finns polymorfismen?

- Ett hotell behöver planera restaureringen av alla rum

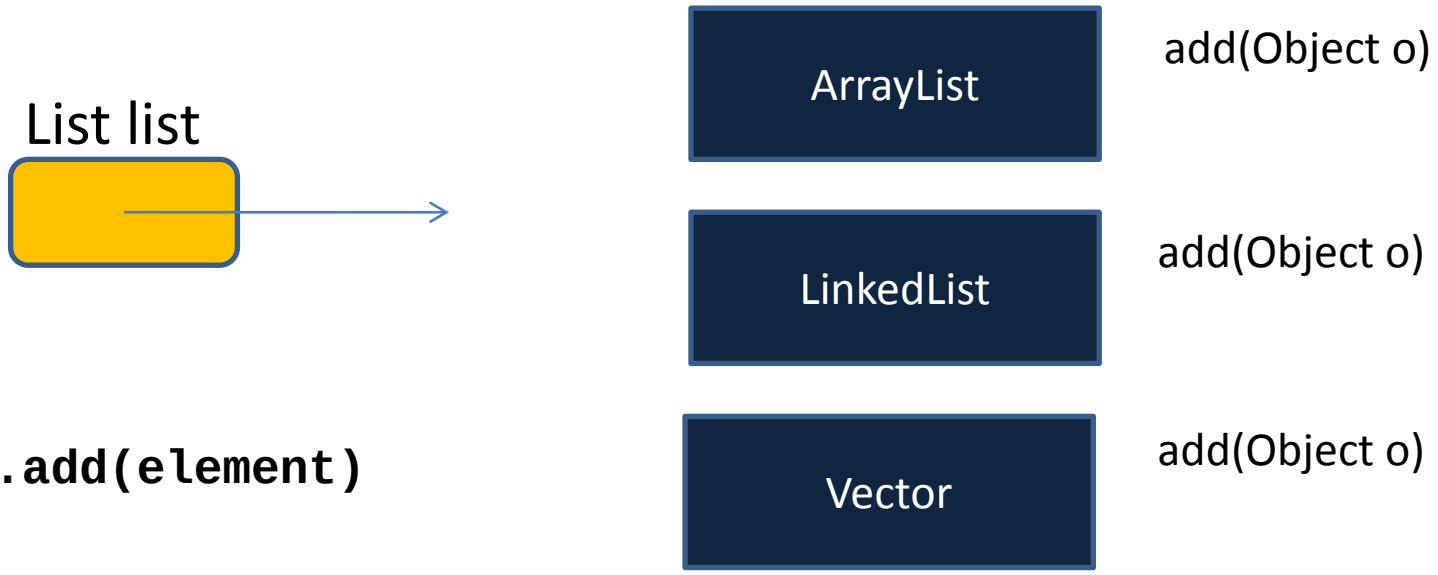
Sammanfattning

- En polymorfisk referens kan referera till objekt av olika typ under programmets exekvering



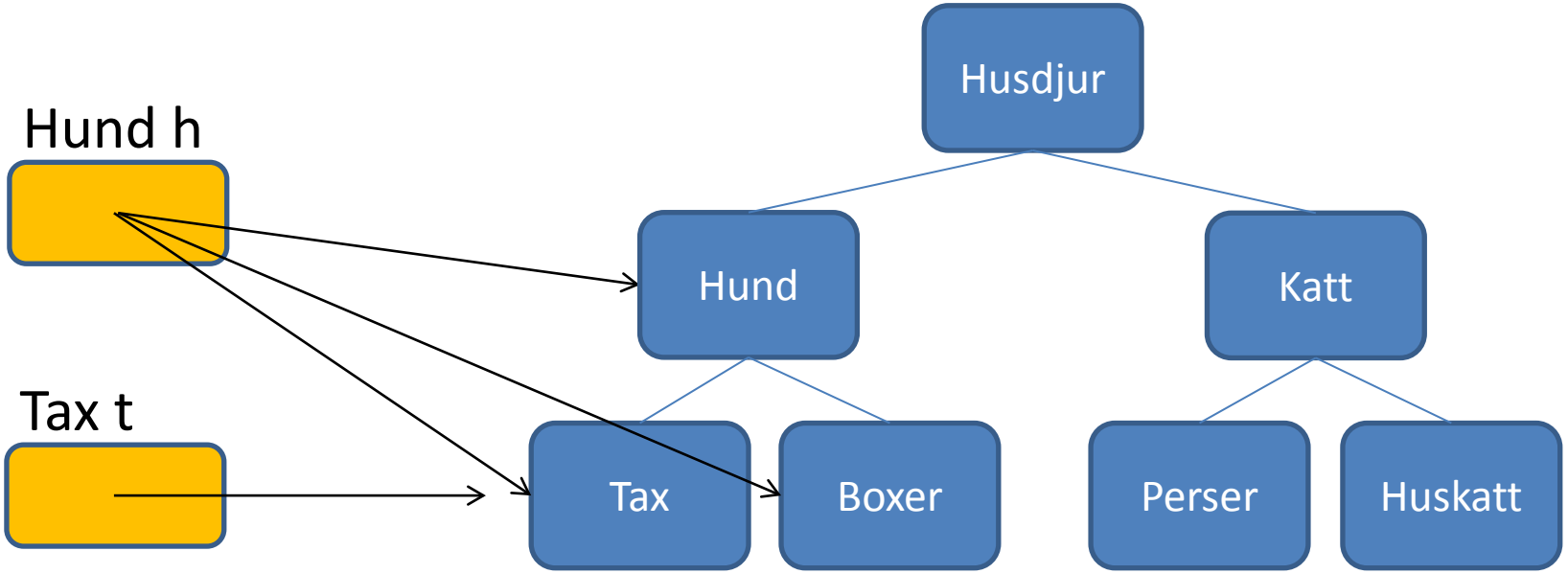
Sammanfattning

- Bindningen mellan anrop och metod sker först vid exekvering



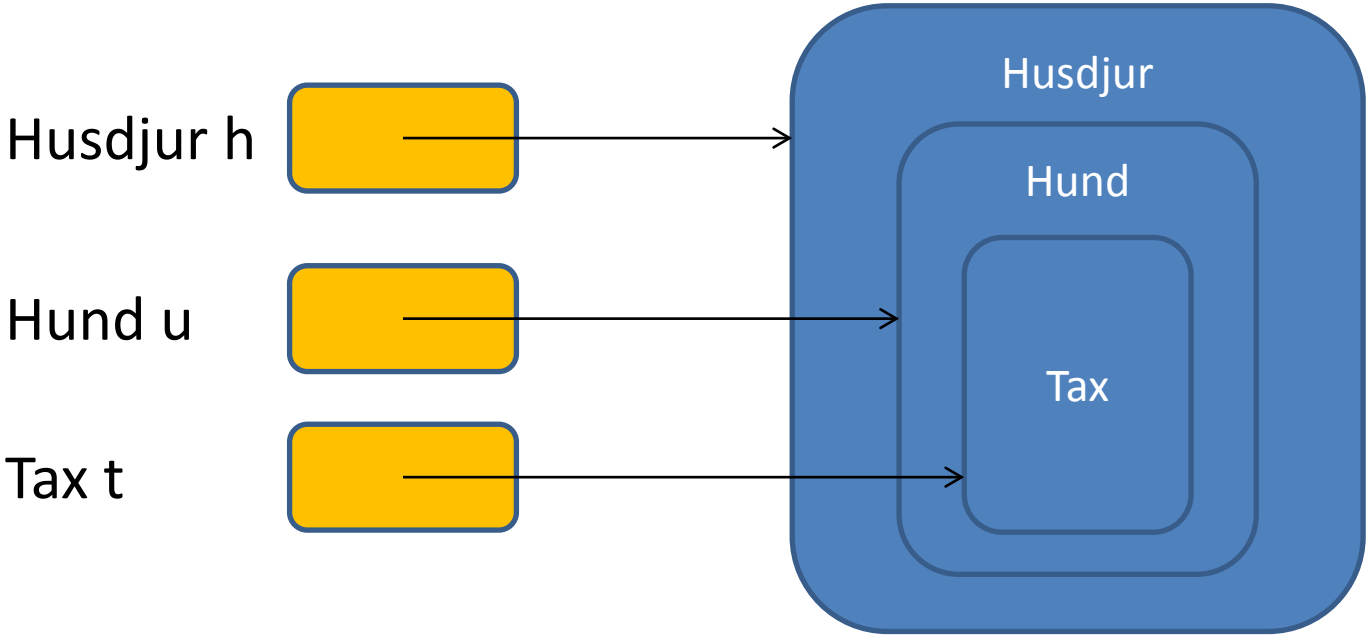
Sammanfattning

- En referensvariabel kan referera till sin typ och alla typens subklasser



Sammanfattning

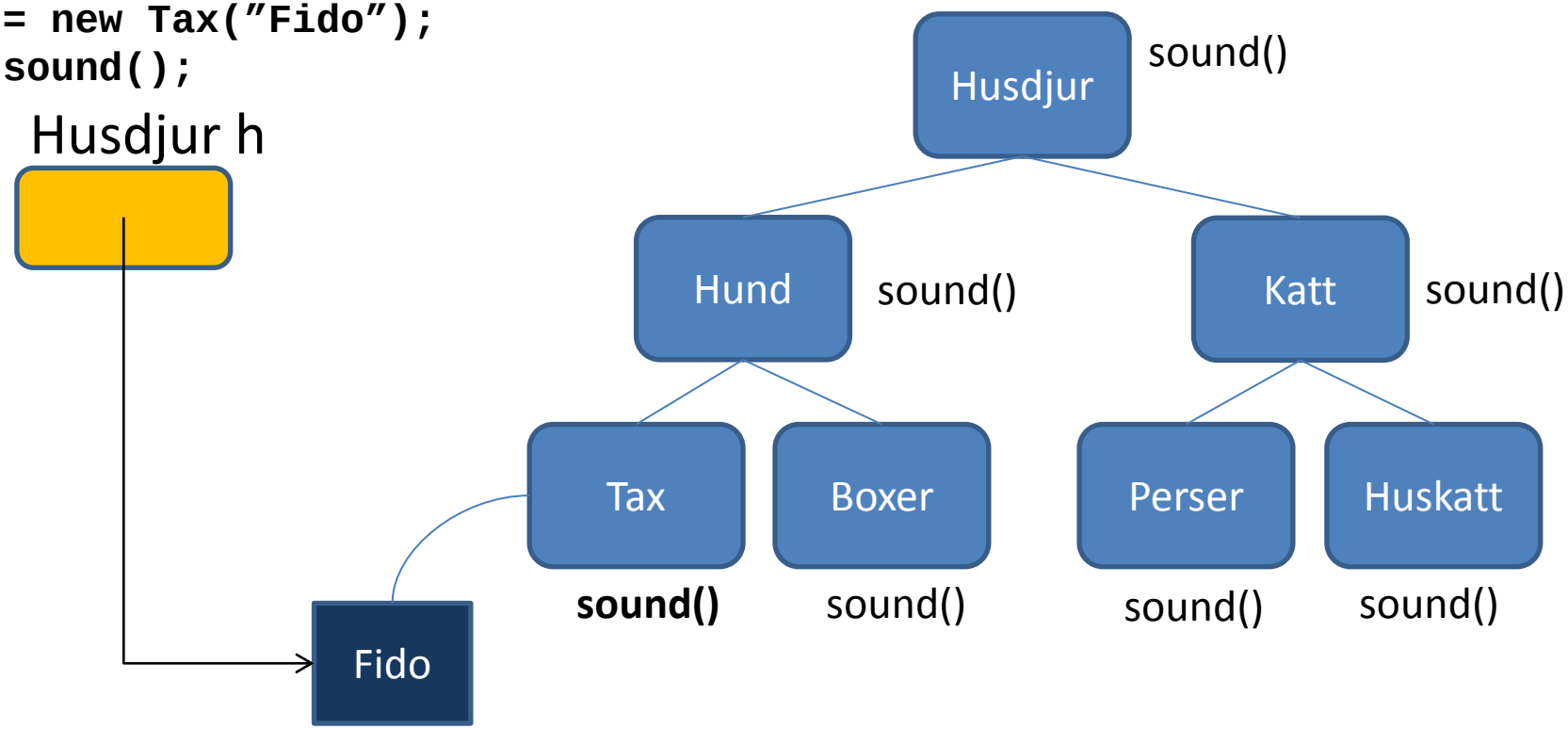
- *Variabelns* typ avgör vilka metoder som *kan* anropas; variabeln når bara sin dekl. typ



Sammanfattning

- *Objektets* typ (inte variabelns) avgör vilken metod som kommer att anropas

```
h = new Tax(”Fido”);
h.sound();
```



Sammanfattning

- Interface är också giltiga datatyper
- En variabel kan deklareras som en interfacetyp
- Den kan då referera till alla objekt som implementerar interfacet
- Variabeln når bara metoder i interfacet

```
List shuffle(List tlist) {

}
```

Sammanfattning

- Polymorfism genom arv eller interface
- Sortera och sök med standardbiblioteket
- Polymorfism är en kraftfull mekanism, och kräver därför god design

polly vill ha sylt

SLUT/THE END/FINE

1 2 3

4 5 6

7 8 9