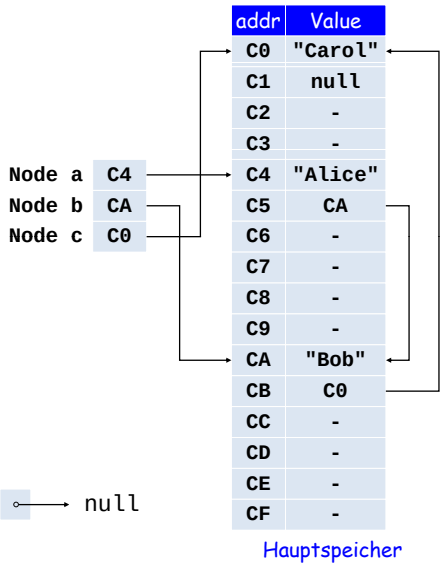
Übung zur VL „Grundlagen der Programmierung“

9. Übung

Dr. Zubow

Verkettete Liste

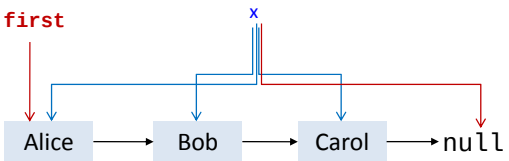
```
Node c  = new Node();
c.item  = "Carol";
c.next  = null;

Node b = new Node();
b.item = "Bob";
b.next = c;

Node a  = new Node();
a.item  = "Alice";
a.next  = b;
```



Traversierung einer verketteten Liste

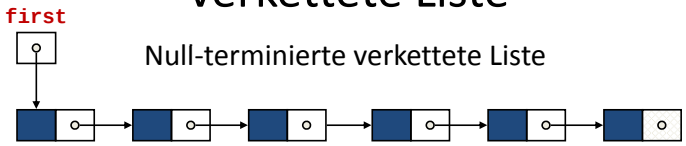
Im Falle einer null-terminierten verketteten Liste:

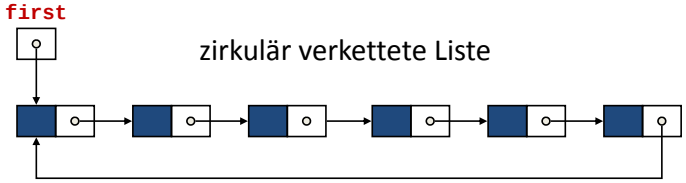
```
for (List x = first; x != null; x = x.next) {
   System.out.println(x.name);
}
```




```
% java List
Alice
Bob
Carol
```

Null-terminierte und zirkulär verkettete Liste

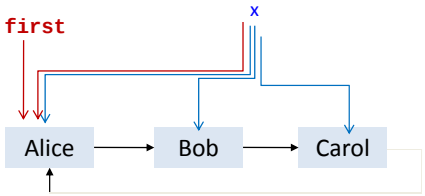




Traversierung einer verkettete Liste

Im Falle einer zirkulär verketteten Liste

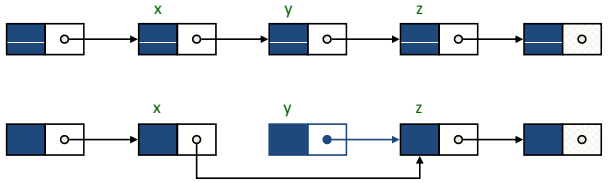
```
if (first != null) {   // if list is not empty
   x = first;
   do {
      System.out.println(x.name);
      x = x.next;
   } while (x != first);
}
```



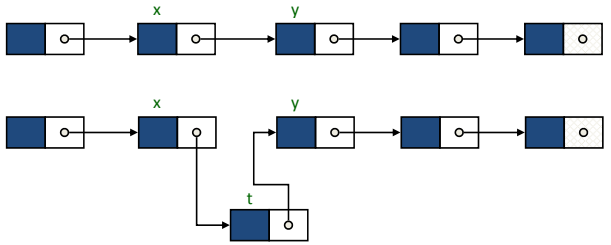

```
% java List
Alice
Bob
Carol
```

Löschen & Einfügen

Löschen

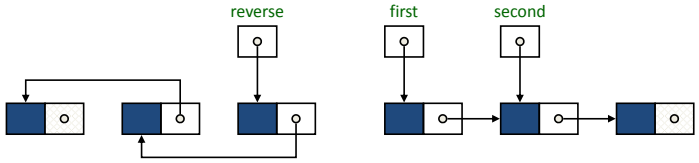


Einfügen



Invertieren einer verketteten Liste

Invertieren

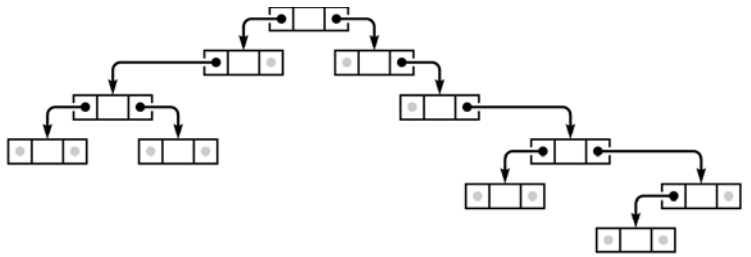


Binärer Baum in Java

Java Implementierung eines binären Baums von Strings

- Eine Referenz auf einen `String`.
- Eine Referenz auf den linken `Tree`.
- Eine Referenz auf den rechten `Tree`.

```
public class Tree {
   private String s;
   private Tree left;
   private Tree right;
}
```



Verkettete Strukturen

- Jede der 3 Methoden f(), g() bzw. h() ist dazu geeignet, eine oder mehrere der Datenstrukturen A,B,C,D nach einem Wert key zu durchsuchen.
- Bestimmen sie zu jeder der Datenstrukturen A,B,C,D jeweils die Funktion (f(), g(), oder h()),die zum Durchsuchen dieser Datenstruktur und dem Auffinden des Wertes key geeignet sind.

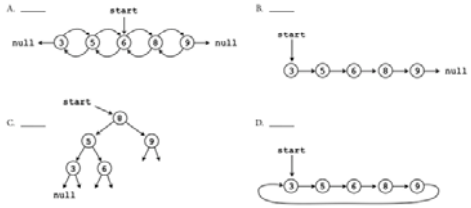
Der Hilfs-Typ Node
ist wie folgt definiert:

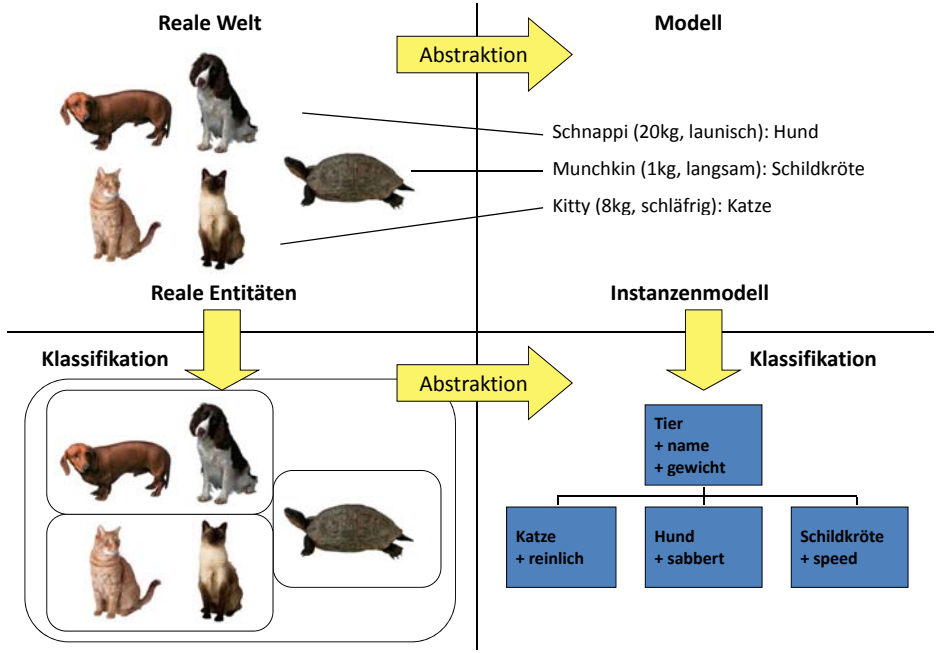
```
public class Node {
  public int key;
  public Node left, right;
}
```

```
public boolean f(int key)
{
  Node x = start;
  while (x != null) {
    if (x.key == key)
      return true;
    x = x.right;
  }
  return false;
}
```

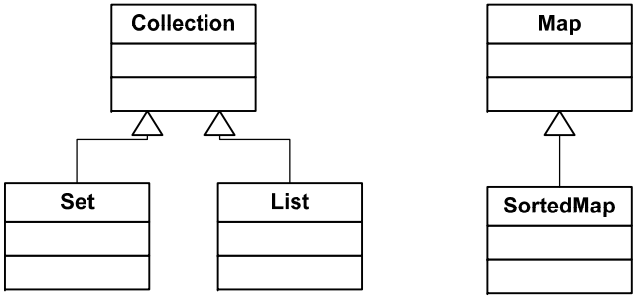
```
public boolean g(int key)
{
  Node x = start;
  while (x != null) {
    if (x.key == key) return true;
    if (x.key > key) x = x.left;
    else x = x.right;
  }
  return false;
}
```

```
public boolean h(int key)
{
  Node x = start;
  while (true) {
    x = x.right;
    if (x.key == key) return true;
    if (x == start) break;
  }
  return false;
}
```





Java Collections



Collection = represents a group of objects, known as its elements.

Set = A collection that contains no duplicate elements.

List = An ordered collection (also known as a sequence).

Map = An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

SortedMap = A map that further guarantees that it will be in ascending key order, sorted according to the natural ordering of its keys.

Objekte in Java

- Beziehungen zwischen Klassen
  - Vererbung (inheritance relationships)
  - Schnittstellen (interfaces)
  - Packaging
  - Innere Klassen (inner classes)
- Subklassen und Vererbung
  - Klassen in Java liegen in einer Klassenhirarchie vor
  - Ein Klasse kann als Subklasse einer anderen Klassen mit Hilfe des Schlüsselwortes **`extends`** deklariert werden
  - Eine Subklasse erbt Variablen und Methoden der Superklasse und kann diese verwenden, als wären sie selbst in der Subklasse deklariert

Objekte in Java

- Subklassen und Vererbung

```
class Animal {
  float weight;
  void eat() { ... }
}
class Mammal extends Animal { // Säugetier
  int heartRate;
  // inherits weight
  ...
  void breathe() { // atmen
    ...
  }
  // inherits eat()
}
```

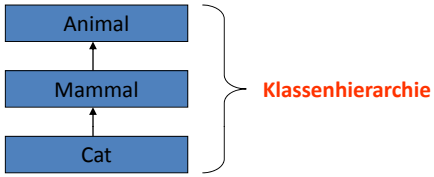
Objekte in Java

- Subklassen und Vererbung
  - Eine Klasse kann max. eine andere erweitern (single inheritance) → Lösung über Interfaces (später)
  - Eine Klasse kann ferner weiter erweitert werden (Vererbung als Spezialisierung)

```
class Cat extends Mammal {
  boolean longHair;
  // inherits weight and heartRate
  ...
  void purr() { // schnurren
    ...
  }
  // inherits eat() and breathe()
}
```

Objekte in Java

- Subklassen und Vererbung
  - Eine Katze ist vom Typ Säugetier, welches wiederum vom Typ Tier ist
  - Objekte vom Typ Katze erben alle Charakteristiken von Objekten des Typs Säugetier und Tier
  - Eine Katze besitzt jedoch zusätzliche Eigenschaften (**`purr()`**-Methode und **`longHair`**-Variable)



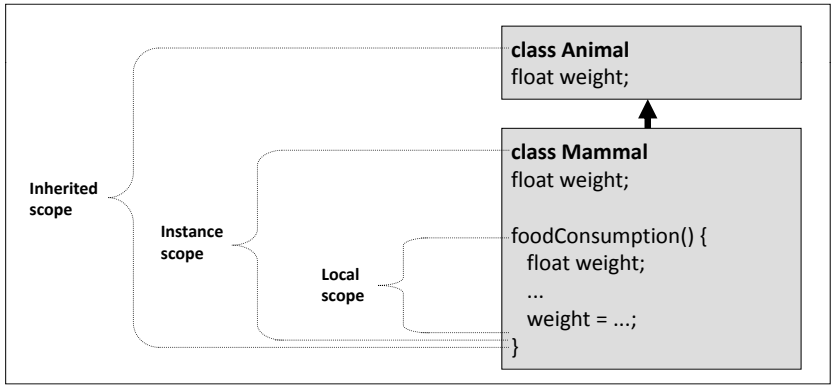
Objekte in Java

- Subklassen und Vererbung
  - Eine Subklasse erbt alle Elemente (Variablen/ Methoden) der Superklasse (Ausnahme: **`private`** deklarierte Elemente)
  - Instanzen des Subtyps können überall dort verwendet werden, wo Instanzen des Supertyps erwartet werden:

```
Cat simon = new Cat();
Animal creature = simon;
```

Objekte in Java

- Subklassen und Vererbung
  - Verdeckte Variablen (shadowed variables)



Objekte in Java

- Subklassen und Vererbung
  - Verdeckte Variablen (shadowed variables)

```
class IntegerCalculator {
  int sum;
}

class DecimalCalculator extends IntegerCalculator {
  double sum;
  ...
  // Zugriff auf sum der Superklasse
  int s = super.sum;
}
```

Objekte in Java

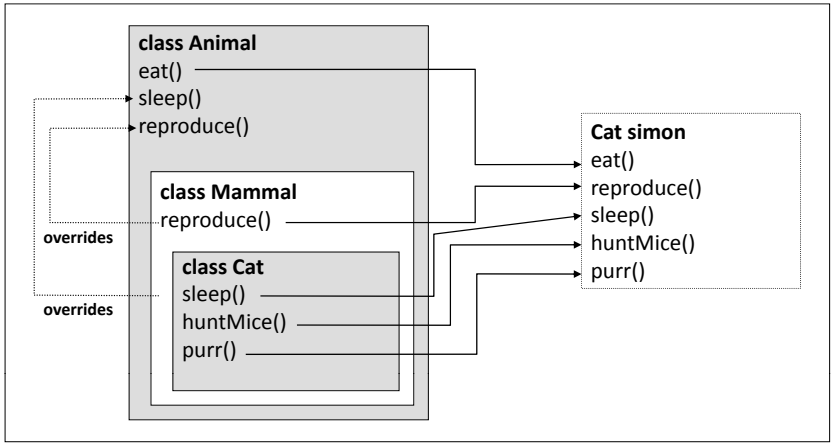
- Subklassen und Vererbung
  - Verdeckte Variablen (shadowed variables)

```
DecimalCalculator dc = new DecimalCalculator();
IntegerCalculator ic = dc;
int s = ic.sum; //Zugriff auf IntegerCalculator
```

  - Überschriebene Methoden (overriding methods)
    - Subklasse kann eine Methode definieren, welche die exakt gleiche Methodensignatur wie eine Methode in der Superklasse hat
    - Die Methode in der Subklasse „überschreibt“ die Methode der Superklasse (ersetzt die Implementierung)
    - Ermöglicht das Verhalten von Objekten zu verändern (→ Subtyp-Polymorphie)

Objekte in Java

- Subklassen und Vererbung



Objekte in Java

- Subklassen und Vererbung
  - Überschriebene Methoden (overriding methods)
    - Existieren mehrere Implementationen einer Methode in der Vererbungshierarchie eines Objektes, dann wird die Methode der speziellsten Klasse gewählt
      - … unabhängig davon, ob über einen weniger speziellen Typ zugegriffen wird

```
Cat simon = new Cat();
Animal creature = simon;
...
creature.sleep(); // accesses Cat sleep()
```

Objekte in Java

- Subklassen und Vererbung
  - Überschriebene Methoden
    - Werden zur Laufzeit (→ überladene Methoden zur Compilezeit) ausgewählt
    - Casting hat einen Einfluss auf die Auswahl von überladenen, nicht aber überschriebenen Methoden
    - Statische Methoden werden nicht dynamisch zur Laufzeit ausgewählt (→ Compilezeit) – eine statische Methode einer Superklasse kann durch eine andere statische M. einer Subklasse überdeckt werden (Ausnahme: **`final`**)
    - Mit Hilfe von **`final`** kann das Überschreiben einer Methode verhindert werden (von **`final`** deklarierten Klassen kann hingegen nicht geerbt werden, z.B. j**`ava.lang.String`**)

Objekte in Java

- Subklassen und Vererbung
  - Spezielle Referenzen: **`this`** und **`super`**
    - Zugriff auf die Bestandteile des aktuellen Objektes bzw. der Superklasse
    - **`super`** um auf Bestandteile der Superklasse zuzugreifen
    - Häufiger Anwendungsfall: Überschreibende Methode (der Subklasse) führt einige vorgelagerte Aufgaben durch, bevor sie die auf die überschriebene Methode (der Superklasse) verweist

```
class Animal {
  void eat(Food f) throws InedibleException {
    // consume food
  }
}
class Herbivore extends Animal {
  void eat(Food f) throws InedibleException {
    // check if edible
    super.eat(f);
  }
}
```

Objekte in Java

- Subklassen und Vererbung
  - Spezielle Referenzen: **`this`** und **`super`**
    - **`super`** veranlasst die Suche nach Variablen/Methoden in der unmittelbaren Superklasse zu beginnen
    - Die geerbte Variable/Methode kann sich in der unmittelbaren bzw. einer weiter entfernten Superklasse befinden
  - Casting
    - Mit Hilfe eines **`cast`** wird dem Compiler explizit mitgeteilt, dass der Typ einer Objektreferenz verändert werden soll
    - Wird zur Compile- und Laufzeitzeit überprüft (→ **`ClassCastException`**)
    - Lediglich das „casten“ von Objekten der selben Vererbungshierarchie ist erlaubt (→ Interfacekonzept)
    - Ein cast hat lediglich Einfluss auf die Referenz – die jeweiligen Objekte bleiben unverändert (es wird lediglich die Sicht des Compilers bzw. der Laufzeit auf ein Objekt verändert)

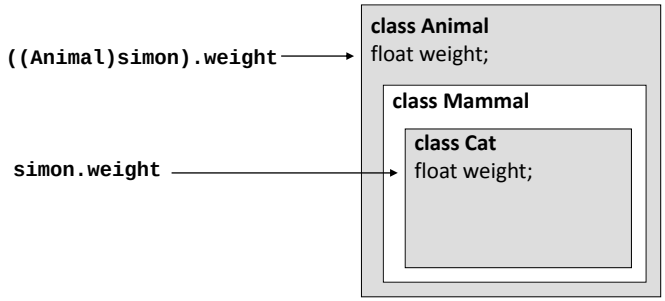
Objekte in Java

- Subklassen und Vererbung
  - Casting
    - Wird benötigt, um den Typ einer Referenz spezieller zu machen:

```
Animal creature = ...
Cat simon = ...
creature = simon; //OK (-> implicit upcasting)
//simon = creature; //Failed, incompatible types
simon = (Cat) creature; //OK (-> downcasting)
```

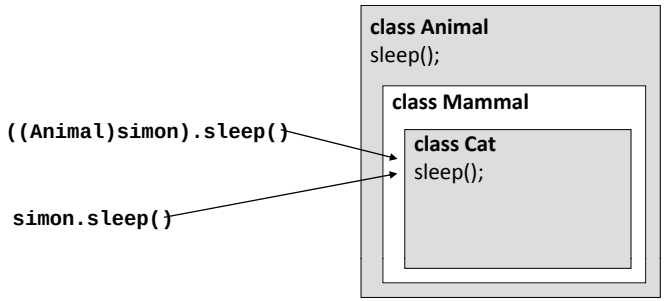
Objekte in Java

- Subklassen und Vererbung
  - Casting - Einfluss auf Variablen und überladene Methoden



Objekte in Java

- Subklassen und Vererbung
  - Casting – **kein** Einfluss auf überschrieben Methoden



Objekte in Java

- Subklassen und Vererbung
  - Superklassen-Konstruktoren
    - Anweisung **`super()`** ruft explizit den Konstruktor der Superklasse auf
    - Notwendig, da Java implizit nur den Standardkonstruktor aufruft

```
class Person {
  Person(String name) {...}
}
class Doctor extends Person {
  Doctor(String name, String speciality) {
    super(name);
    // process additional work
  }
}
```