IT I: Heute

- abstrakte Methoden und Klassen
- Interfaces
- Interfaces *List*, *Set* und *Collection*

Abstrakte Methoden und Klassen

- Eine abstrakte Methode ist die Definition einer Methode ohne Implementierung:
-

```
abstract String getDetails();
```

- Eine Klasse mit einer abstrakten Methode muss selbst abstrakt sein:
-

```
abstract class Disk
{
    …
}
```

Abstrakte Methoden und Klassen

- Von einer abstrakten Klasse können keine Objekte erzeugt werden (da u.U. manche Methoden nicht implementiert sind).
- Trotzdem kann abstrakte Klasse einen Konstruktor haben! Dieser kann aber nicht zur Erstellung eines Objekts verwendet werden.
- In einer nicht-abstrakten Unterklasse einer abstrakten Klasse muss es für alle abstrakten Methoden Implementierungen geben.

Java *interfaces*

- *Interfaces* sind eine besondere Form von abstrakten Klassen:
  - Alle Methoden sind *public* und *abstract*.
  - Es gibt keine Konstruktoren.
  - Es gibt keine Attribute außer Konstantendefinitionen, die *public*, *static* und *final* sind.
- D.h., dass eine nicht-abstrakte Unterklasse eines Interfaces ***alle*** Methoden des Interfaces implementieren muss.

Java *interfaces*

- Interface ist also eine Spezifikation, die angibt, über welche Methoden ein Objekt dieses Typs verfügt.

- **Neuerung:**

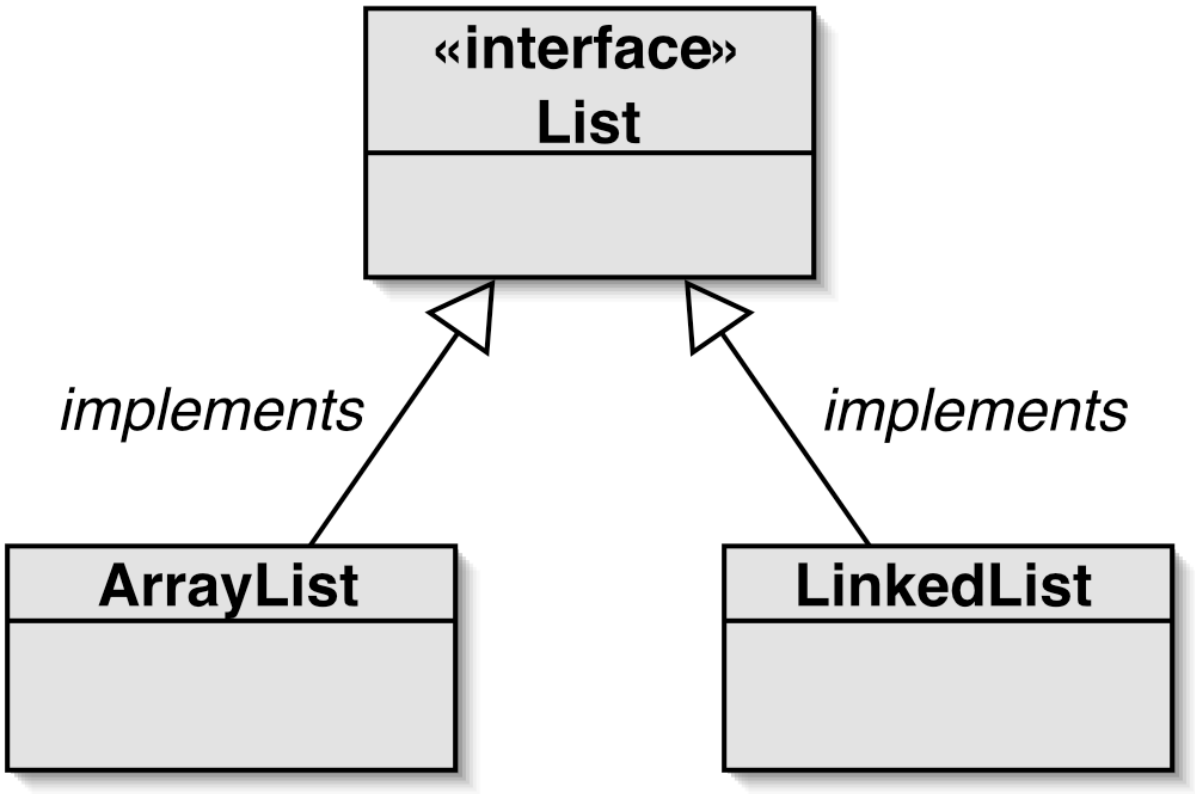
  Seit Java 8 sind sogenannte *Default-Implementierungen* von Methoden in einem Interface erlaubt.

Syntax von Java *interfaces (1)*

- interface *Interfacename*
  {
  …
  }

- class Klassenname implements *Interfacename*
  {
  …
  }

ArrayList implementiert Interface *List*



List

- Gegeben: *List<Anlage> liste*;
- kann alle Methoden aus dem Interface *List* verwenden:
  - *get(i)*
  - *for each* – Schleife
  - *size()*
  - etc.

List

- Rückgabewert *List<Anlage>*:

  muss Objekt einer Klasse zurückgeben, das *List* implementiert, z.B. *ArrayList*
- *List<Anlage> liste = new ArrayList<Anlage>();*

  oder
- *ArrayList<Anlage> liste = new ArrayList<Anlage>();*

Interfaces

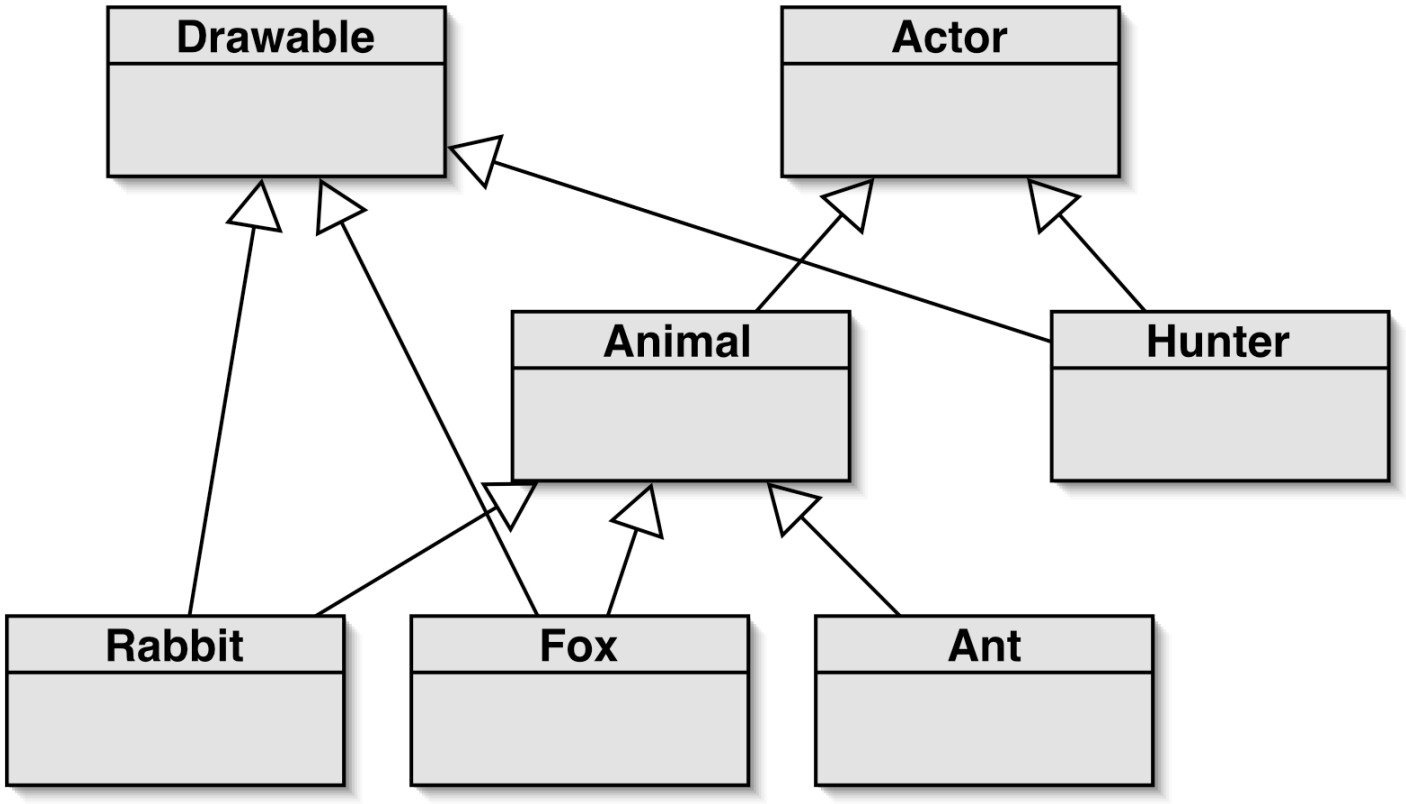
- Interfaces können ganz normal als Datentyp eingesetzt werden, s. Bspe. mit Interface *List* !
- Weiteres Beispiel:

```
interface einInterface
{   public boolean istDoof(); }

einInterface x = ...;
if(x.istDoof()){...}
```

- **NB:** Konkrete Objekte (z.B. rechte Seite der Zuweisung *x=...;*) sind immer von einer Klasse, die das Interface implementiert!

Mehrfachvererbung



Mehrfachvererbung

- Wenn eine Klasse von mehreren Oberklassen erbt, kann es zu Konflikten kommen, wenn dieselbe Methode in mehreren Oberklassen unterschiedlich implementiert ist.
  - Welche Methode wird ausgeführt?

- In Java wird dieser Konflikt vermieden, indem es nur eine echte Oberklasse geben kann, alle anderen Oberklassen müssen *Interfaces* sein.

- Da es in Interfaces keine Implementierung gibt, kommt es zu keinen Konflikten.

Syntax von Java *interfaces (2)*

- interface *Interfacename*
  implements *Interface1*, …, *InterfaceK*
  {
  …
  }

- class Klassenname
  extends Oberklasse
  implements *Interface1*, …, *InterfaceK*
  {
  …
  }

Collections (1)

- Hold groups of objects.
- Increase their capacity as necessary.
- Keep the objects in (some) order.
- Details of how all this is done are hidden.
- We specify:
  - the type of a collection: e.g. `ArrayList`
  - the type of the objects it will contain: e.g. `<String>`
- Collections are defined in *java.util*.

Collections (2)

- Die Gemeinsamkeiten von Collections sind im Interface *Collection* definiert.
- Wichtige Sub-Interfaces von *Collection* sind *List* und *Set*.
- Konkrete (nicht-abstrakte) Unterklassen von *List* sind *ArrayList* und *LinkedList*, sowie *HashSet* und *TreeSet* von *Set*.
- Die wesentlichen Methoden für eine Liste sind im Interface *List* definiert, und daher für *ArrayList* und *LinkedList* gleich.

Sets

- Sets sind Collections:
  - *add()*
  - *size()*
  - *for each* Schleife

  Achtung:
  Für *Set* gibt's keine *get*-Methode!
  (Zugriff nur über *for each* Schleife!)

- *HashSet* implementiert Interface *Set*.

Sets

- Unterschiede zwischen *List* und *Set*:
  - *List* enthält eine Sequenz von Objekten.
  - Objekte können öfters vorkommen.
  - *Set* enthält Objekte in keiner bestimmten Reihenfolge (keine *get*-Methode!)

    Spez. Implementierungen existieren, die eine spezielle Reihenfolge der Objekte einhalten.
  - In *Set* kommt ein Objekt nur einmal vor.