04-Objekte-3-Vererbung-2

Objektorientierte Programmierung | Matthias Tichy

Software Engineering
Programming Languages

universität ulm

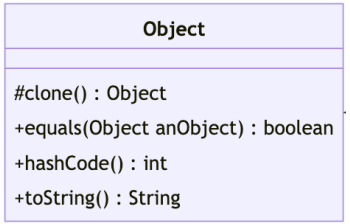
Lernziele

- Methoden überschreiben
- Virtuelle Methodenbindung
- Interfaces
- Abstrakte Klassen
- instanceof

Methoden überschreiben und virtuelle Methodenbindung

Die Klasse Object

- Alle Java Klassen erben (automatisch) von Object

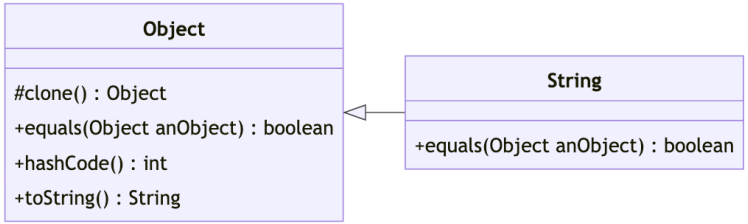



```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Methoden können in der erbenden Klasse „geändert“ werden

Die Klasse Object

- Alle Java Klassen erben (automatisch) von Object




```
public boolean equals(Object obj) {
    return (this == obj);
}
```

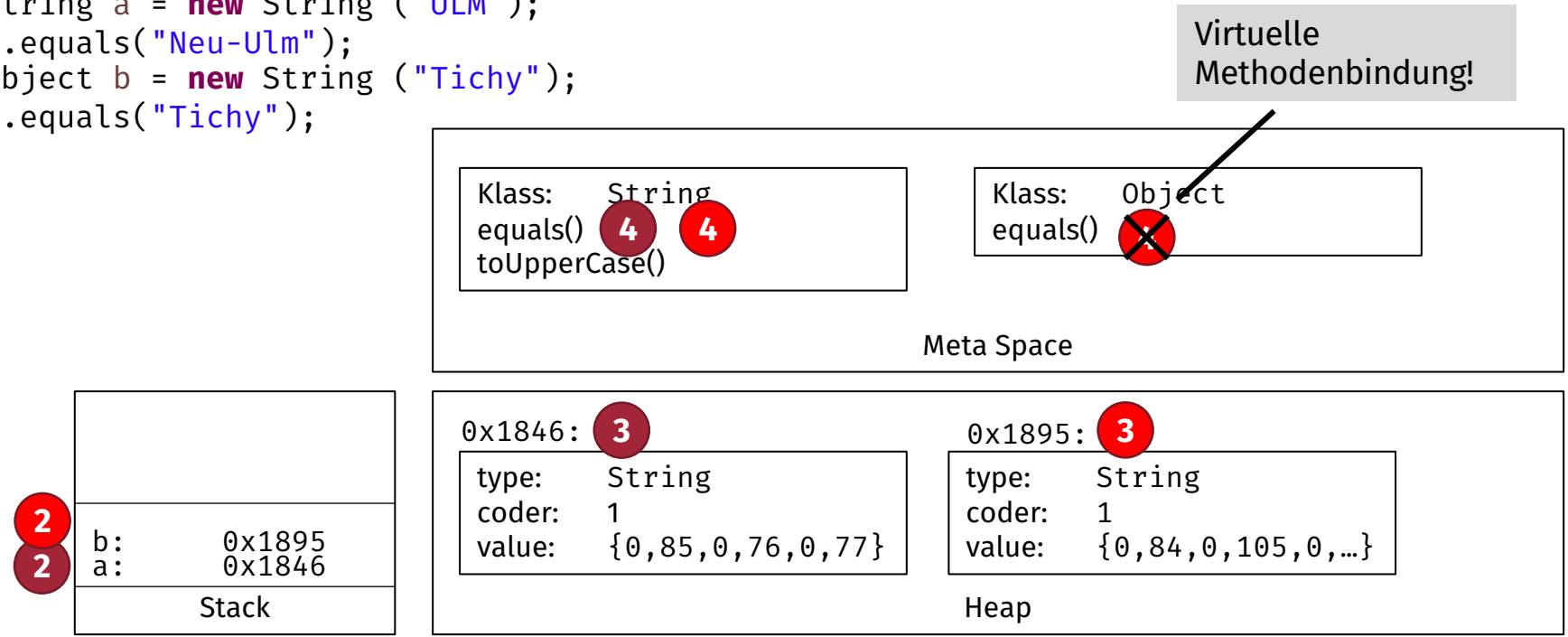
- Methoden können in der erbenden Klasse „geändert“ werden → String überschreibt equals()

```
public class String extends Object {
  public boolean equals(Object anObject) {
      if (this == anObject) { return true; }
      return (anObject instanceof String aString)
              && (!COMPACT_STRINGS || this.coder == aString.coder)
              && StringLatin1.equals(value, aString.value);
  }
}
```

Virtuelle Methodenbindung

```
String a = new String ("ULM");
a.equals("Neu-Ulm");
Object b = new String ("Tichy");
b.equals("Tichy");
```

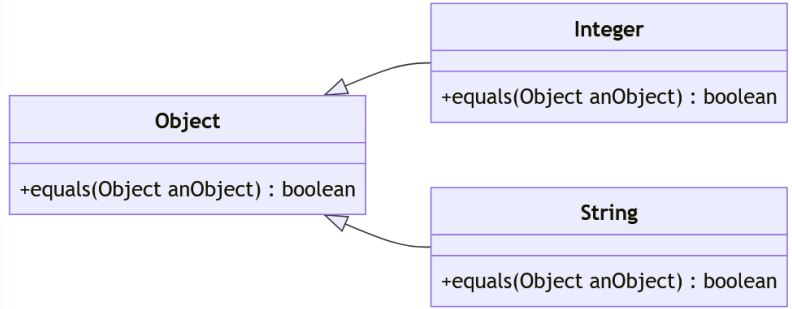
1
1



Virtuelle Methodenbindung

> Bei der virtuellen Methodenbindung wird die auszuführende Methode zur Laufzeit auf Basis des Objekts und der Argumente ausgewählt.

```
Object a;
if (Math.random()>0.5) {
  a = new String ("ULM");
} else {
  a = new Integer(42);
}
a.equals("Neu-Ulm");
```



- Ursache: Überschreiben und Überladen von Methoden

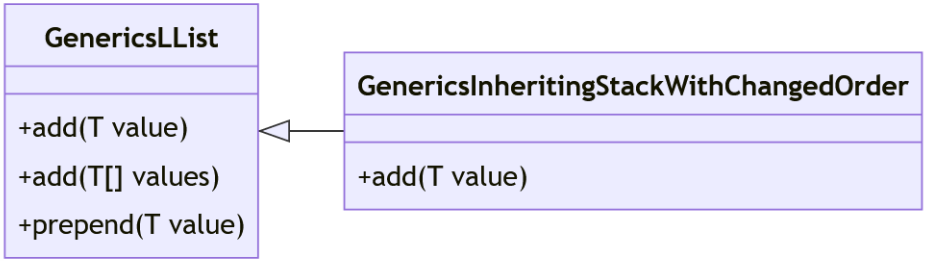
Virtuelle Methodenbindung

```
public class GenericsLList<T> {
  public void prepend(T value) {
    if (head == null) {
      var elem = new GenericsListElement<T>(value);
      head = elem;
    } else {
      var newHead = new GenericsListElement<T>(value);
      newHead.setNext(head);
      this.head = newHead;
    }
  }
  public void add(T[] values) {
    for (T t : values) {
      this.add(t);
    }
  }
}
```

```
public class GenericsInheritingStackWithChangedOrder<T>
extends GenericsLList<T> {
  @Override
  public void add(T value) {
    super.prepend (value);
  }
  public Optional<T> peek () {
    return get(0);
  }
}
```

Optionale Annotation, dass die Methode etwas überschreibet

Virtuelle Methodenbindung




```
public class GenericsLList<T> {
  public void add(T value) {
  ...
  }
  public void add(T[] values) {
    for (T t : values) {
      this.add(t);
    }
  }
}
```

```
public class GenericsInheritingStackWithChangedOrder<T> {
extends GenericsLList<T> {
  @Override
  public void add(T value) {
    super.prepend (value);
  }
}
```

```
private static GenericsLList<Integer> createGenericsInheritingStack()
{
  var l = new GenericsInheritingStackWithChangedOrder<Integer>();
  l.add(new Integer[] {12, 14, 42});
  return l;
}
```

- Was passiert?

Interfaces und abstrakte Klassen

Ähnliche Methoden

Stack

- Methoden:
  - push
  - pop
  - peek
  - size

List

- Methoden:
  - add
  - remove
  - get
  - size

- Interfaces deklarieren eine Menge zusammenhängender Methoden – **ohne deren Implementierung und ohne Attributdefinition**
- Ziel: Nutzer muss nicht wissen, wie etwas implementiert ist, nur dass ein Objekt (bzw. dessen Klasse) die Schnittstelle implementiert

Murmelgruppe

Interfaces

> Welche Methoden würden Sie minimal erwarten, um Mengen zu verarbeiten?

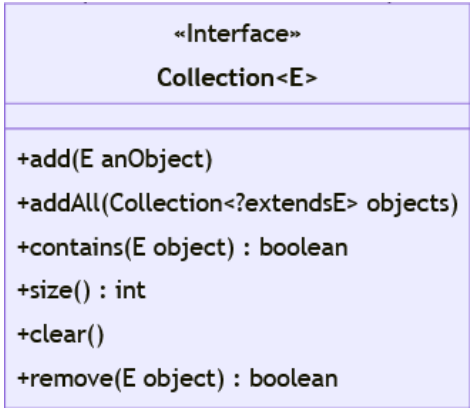
- Diskutieren Sie mit ihrem/r Sitznachbar/in 2-3 Minuten
- Wir sammeln und gruppieren diese Methoden danach im Plenum.

Collection<E>

Erweiterung des Iterable<E> Interfaces

Interfaces

```
public interface Collection<E> extends Iterable<E> {
  boolean add(E e);
  boolean addAll(Collection<? extends E> c);
  boolean contains(E o);
  int size();
  void clear();
  boolean remove(E o);
  ...
}
```



- „A collection represents a group of objects, […] its elements.”
- Keine Aussage in Bezug auf Ordnung oder Duplikate.

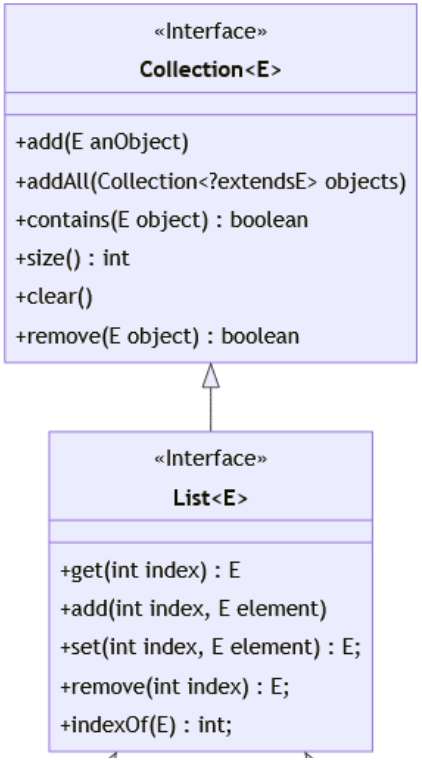
https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/Collection.html

List<E>

Interfaces

Erweiterung des Collection<E> Interfaces

```
public interface List<E> extends Collection<E> {
  // Positional Access Operations
  E get(int index);
  E set(int index, E element);
  void add(int index, E element);
  E remove(int index);
  // Search Operations
  int indexOf(E o);
  int lastIndexOf(E o);
  ...
}
```



- Listen haben eine Ordnung und können typischerweise Duplikate enthalten.

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/List.html

Default Implementierung

Interfaces

```
public interface List<E> extends Collection<E> {
 default void sort(Comparator<? super E> c) {
        Object[] a = this.toArray();
        Arrays.sort(a, (Comparator) c);
        ListIterator<E> i = this.listIterator();
        for (Object e : a) {
            i.next();
            i.set((E) e);
        }
    }
  ...
}
```

- Interfaces enthalten keine Implementierung – außer, wenn eine Implementierung ohne Wissen über die Implementierung möglich ist.

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/List.java

LinkedList<E>

Implementierung

```
public class LinkedList<E>
  extends AbstractSequentialList<E>
  implements List<E> {

  transient int size = 0;
  transient Node<E> first;
  transient Node<E> last;
```

```
public boolean add(E e) {
  linkLast(e);
  return true;
}

void linkLast(E e) {
  final Node<E> l = last;
  final Node<E> newNode = new Node<>(l, e, null);
  last = newNode;
  if (l == null)
    first = newNode;
  else
    l.next = newNode;
  size++;
}
```

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/LinkedList.java

ArrayList<E>

Implementierung

```
public class ArrayList<E>
  extends AbstractList<E>
  implements List<E> {

  private int size;
  transient Object[] elementData;
```

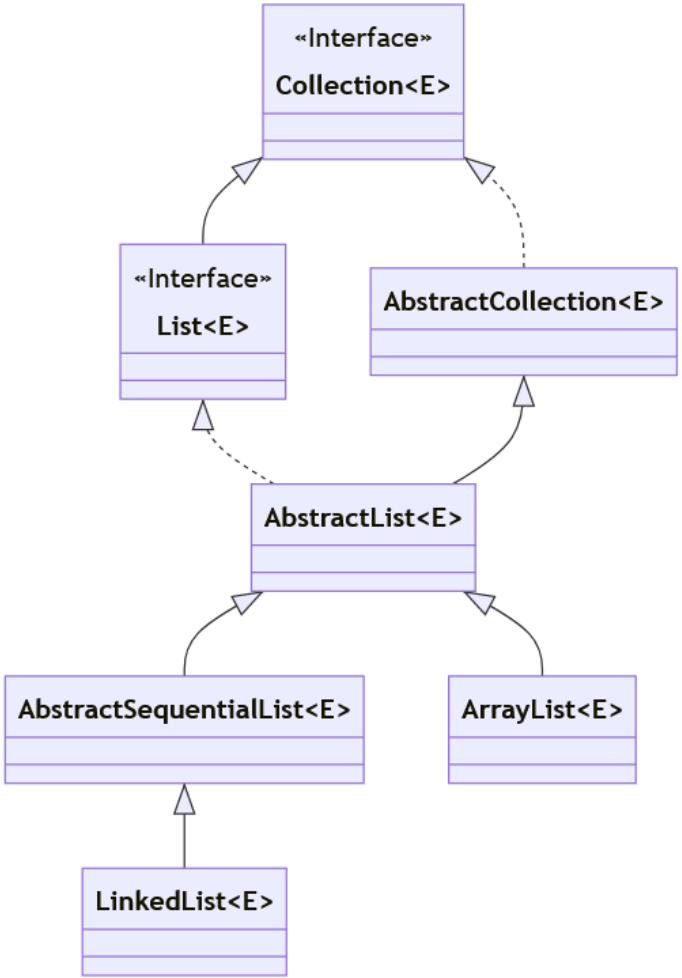
```
public boolean add(E e) {
  modCount++;
  add(e, elementData, size);
  return true;
}

private void add(E e, Object[] elementData, int s)
{
  if (s == elementData.length)
    elementData = grow();
  elementData[s] = e;
  size = s + 1;
}
```

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/ArrayList.java

Vererbungshierarchie

Ausschnitt



AbstractCollection<E>

Implementierung

```
public abstract class AbstractCollection<E> implements Collection<E> {

  public abstract Iterator<E> iterator();
  public abstract int size();

  public boolean isEmpty() {
    return size() == 0;
  }
  public boolean contains(Object o) {
    Iterator<E> it = iterator();
    [...]
    while (it.hasNext())
      if (o.equals(it.next()))
        return true;
    return false;
  }
```

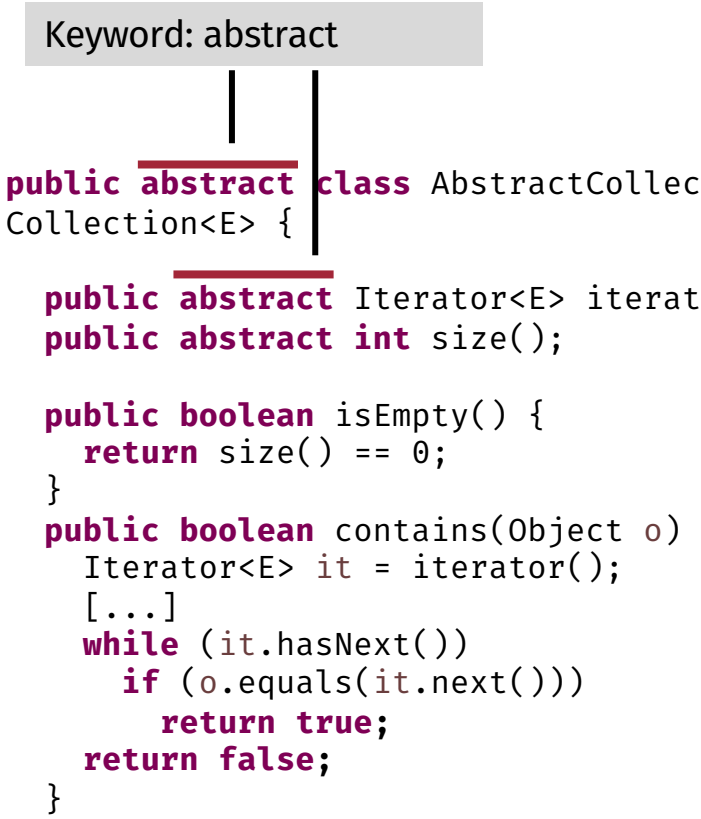
Abstrakte Methoden werden nur deklariert

Methode ist definiert

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/AbstractCollection.java

Abstrakte Klassen

- Beinhalten eine Teilimplementierung
- Einige Methoden: implementiert
- Einige Methoden: nicht implementiert
- Keine Instanziierung möglich
- Ziel: Wiederverwendung allgemeiner Programmlogik für mehrere Subklassen



Murmelgruppe

Interfaces

> Finden Sie die Bedeutung der Interfaces Comparable<T>, RandomAccess heraus.

- Recherchieren Sie gemeinsam mit ihrem/r Sitznachbar/in.
- Diskutieren Sie die gefunden Inhalte.
- Wir sammeln und diskutieren danach im Plenum.

Comparable<T>

Interfaces

> „This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's natural ordering, and the class's code compareTo method is referred to as its natural comparison method.”

```
public interface Comparable<T> {
 public int compareTo(T o);
}

public final class String
  implements Comparable<String>
{
 ...
```

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/

```
jshell> String a = new String ("ULM");
a ==> "ULM"
jshell> String b = new String ("Tichy");
b ==> "Tichy"
jshell> a.compareTo(b);
$3 ==> 1
jshell> b.compareTo(a);
$4 ==> -1
```

compareTo<T>

Gesetze

- “Compares this object with the specified object for order.
- Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
- The implementor must ensure signum(x.compareTo(y)) == -signum(y.compareTo(x)) for all x and y.
- The implementor must also ensure that the relation is transitive: (x.compareTo(y) > 0 && y.compareTo(z) > 0) implies x.compareTo(z) > 0.
- Finally, the implementor must ensure that x.compareTo(y)==0 implies that signum(x.compareTo(z)) == signum(y.compareTo(z)), for all z.”
- Zusammenhang mit equals()?

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/Comparable.html

RandomAccess

Interfaces

> „Marker interface used by List implementations to indicate that they support fast (generally constant time) random access. The primary purpose of this interface is to allow generic algorithms to alter their behavior to provide good performance when applied to either random or sequential access lists.”

```
public interface RandomAccess {
}
```

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/util/RandomAccess.html

Collections.shuffle()

Nutzung eines „Marker“-Interfaces

```
public static void shuffle(List<?> list, Random rnd) {
  int size = list.size();
  if (size < SHUFFLE_THRESHOLD || list instanceof RandomAccess) {
    for (int i=size; i>1; i--)
      swap(list, i-1, rnd.nextInt(i));
  } else {
    Object[] arr = list.toArray();
    for (int i=size; i>1; i--)
      swap(arr, i-1, rnd.nextInt(i));
      ListIterator it = list.listIterator();
      for (Object e : arr) {
        it.next();
        it.set(e);
      }
    }
  }
}
```

Implementiert list das Interface RandomAccess?

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/Collections.java

instanceof

instanceof

Ausdruck zur Prüfung ob ein Objekt Instanz eines definierten Interfaces, einer Klasse oder Subklasse ist.

```
jshell> Object a = new String ("ULM");
a ==> "ULM"
jshell> a instanceof String
$2 ==> true
jshell> a instanceof Object
$3 ==> true
jshell> a instanceof Comparable
$4 ==> true
```

instanceof

Ausdruck zur Prüfung ob ein Objekt Instanz eines definierten Interfaces, einer Klasse oder Subklasse ist.

```
jshell> Object a = new String ("ULM");
a ==> "ULM"
jshell> if (a instanceof String) {
  String str = (String) a;
  System.out.println(str);
}
```

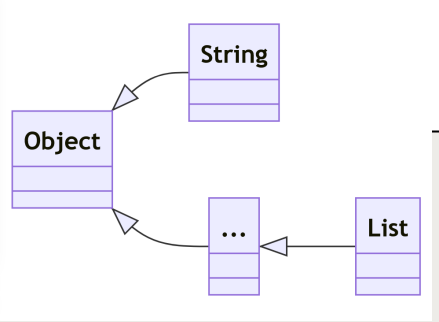
instanceof

Ausdruck zur Prüfung ob ein Objekt Instanz eines definierten Interfaces, einer Klasse oder Subklasse ist.

```
jshell> Object a = new String ("ULM");
a ==> "ULM"
jshell> if (a instanceof String str) {
  String str = (String) a;
  System.out.println(str);
}
```

instanceof

Ausdruck zur Prüfung ob ein Objekt Instanz eines definierten Interfaces, einer Klasse oder Subklasse ist.

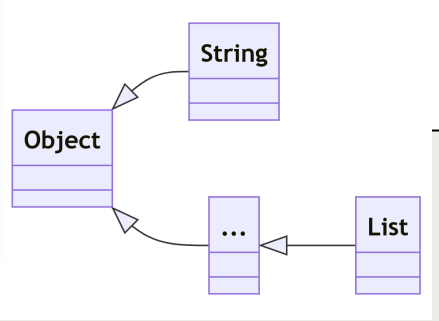



```
jshell> String a = new String ("ULM");
a ==> "ULM"
jshell> a instanceof List
|  Fehler:
|  Inkompatible Typen: java.lang.String kann nicht in java.util.List
konvertiert werden
|  a instanceof List
|  ^
```

instanceof nur hierarchisch nach oben und unten. Alles andere kann sowieso nicht true werden.

Type Cast

Typzusicherung für den Compiler, welchen Typ er für ein Objekt annehmen soll.



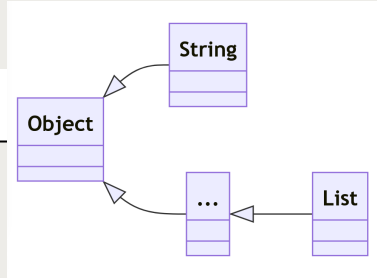

```
jshell> String a = new String ("ULM");
a ==> "ULM"
jshell> ((Object) a) instanceof List
$5 ==> false
```

TypeCast

Type Cast

Typzusicherung für den Compiler, welchen Typ er für ein Objekt annehmen soll.

→ Trotzdem Typprüfung zur Laufzeit




```
jshell> String a = new String ("ULM");
a ==> "ULM"
jshell> List l = (List) ((Object) a)
|  Exception java.lang.ClassCastException: class java.lang.String cannot be
cast to class java.util.List
|        at (#3:1)

jshell>
```

Type Cast

Typzusicherung für den Compiler, welchen Typ er für ein Objekt annehmen soll.

→ Trotzdem Typprüfung zur Laufzeit

- Nutzung:
  - Manchmal geht der wirkliche Typ “verloren” → siehe Generics
  - TypeCast mit instanceof kombinieren

Lernziele

- Methoden überschreiben
- Virtuelle Methodenbindung
- Interfaces
- Abstrakte Klassen
- instanceof