Technische Hochschule Rosenheim
Technical University of Applied Sciences

Objektorientierte Programmierung
Kapitel 6 – Ausgewählte Klassen, Datentypen und Interfaces

Prof. Dr. Kai Höfig

Inhalt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Arrays**
- Strings
- Basisklasse Object
- Java-Klassenbibliothek
  - Primitive Datentypen und Referenztypen
  - Packages
- Weitere Klassen
  - Wrapper-Klassen
  - Klasse System
  - Klasse Class

Literatur: C. Ullenboom, Java ist auch eine Insel
Kapitel 4, Kapitel 3.6 und 3.7

Arrays: Wiederholung

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Grundschritte
  - Deklarieren von Arrays:
  - Initialisierung von Arrays:
  - Zugriff auf Arrays:

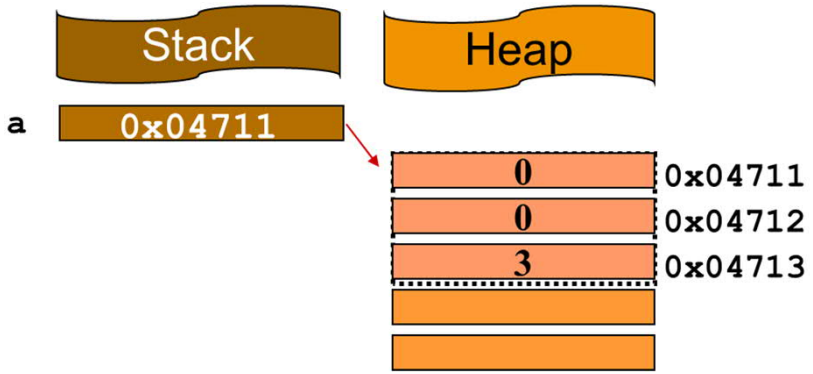
```
int[] a;
a = new int[10];
a[2] = 3;
```

- Direkte Belegung mit festen Werten

```
int[] a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- Arrayinhalt:
  - Primitiver Datentyp oder *beliebiges* Objekt.

- Speicherort
  - Variable/Bezeichner = Referenz (Zeiger) auf Stack
  - Arrayinhalt liegt auf Heap



Arbeiten mit Arrays

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- ***Länge*** eines Arrays
  - `a.length`
- ***Kopieren*** ganzer Arrays
  - `targetArray = sourceArray.clone();` → siehe später!
  - **Nicht!!!**: `targetArray = sourceArray;`
- `java.util.Arrays`: Statische Hilfsmethoden für Arrays.
  - Sortieren und Durchsuchen von Arrays
  - Umwandeln in `Collection`, siehe später: `Arrays.asList()`
- `java.lang.System`: Statische Methode `arraycopy()` um Teile eines Arrays zu kopieren.

```
char[] ca = {'H', 'a', 'l', 'l', 'o'};
char[] cb = {'S', 'e', 'r', 'v', 'u', 's'};
System.arraycopy(ca, 2, cb, 1, 3);
System.out.println(Arrays.toString(cb));
```

?

Ist ein Array ein Objekt?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Arrays verhalten sich ähnlich wie Objekte
  - Ein Array hat ein Attribut → `length`
  - Für ein Array muss Speicherplatz reserviert werden → `new`
  - Ein Array ist ein Referenzdatentyp.

- Dennoch sind Arrays speziell in Ihrer Notation
  - `int[] a = new int[10];`

- **Achtung!**
  - "==" vergleicht Identität und nicht ob Zustand von Objekten gleich ist.

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};
if (a == b) {
}
```

Inhalt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

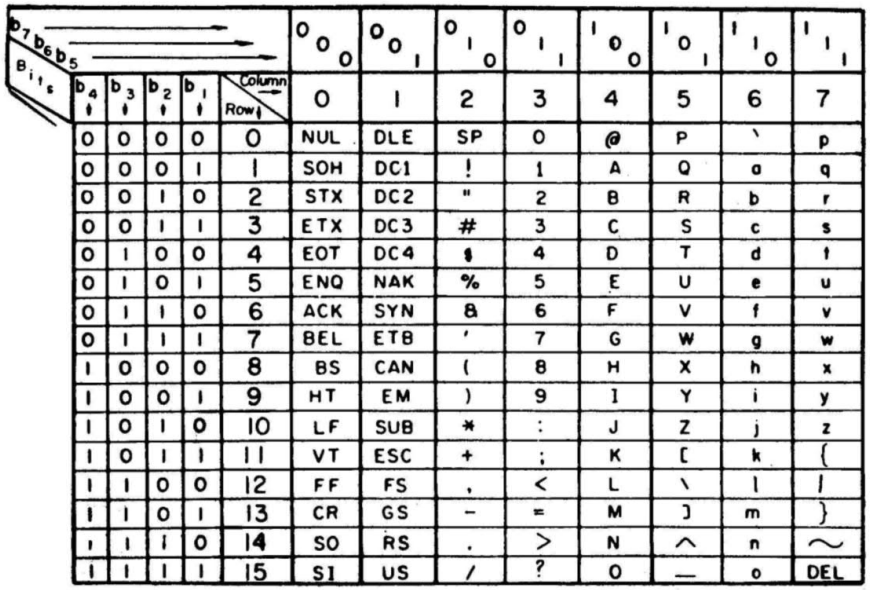
- Arrays
- **Strings**
- Basisklasse Object
- Java-Klassenbibliothek
  - Primitive Datentypen und Referenztypen
  - Packages
- Weitere Klassen
  - Wrapper-Klassen
  - Klasse System
  - Klasse Class

Literatur: C. Ullenboom, Java ist auch eine Insel
Kapitel 4, Kapitel 3.6 und 3.7

Zeichenkodierungen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Zeichen müssen durch Bits kodiert werden.
- ***ASCII:***
  - 7-Bit
- ***ISO/IEC 8859-1:***
  - 8-Bit
  - + Umlaute
- ***Unicode***: 4 Byte
  - Weitere Zeichen
  - Die meisten Zeichen kommen fast nie vor
  - UTF-8 kodiert häufige Zeichen in 1 Byte
  - Die restlichen Zeichen in 16 und 32 Byte

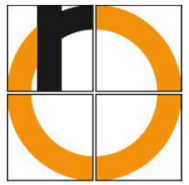


Bits b4 b3 b2 b1	Row	000	001	010	011	100	101	110	111
	Column	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

ASCII Tabelle aus dem Original-Standard [1]

Klasse `Character`

Technische Hochschule Rosenheim
Technical University of Applied Sciences



- Zahlreiche statische Methoden, die im Umgang mit Zeichen interessant sind.

- Beispiele
  - ***Testmethoden:***
    - `isDigit(.)?`
    - `isLetter(.)?`
    - `isWhitespace(.)?`
  - ***Konvertieren:***
    - `toLowerCase(.)`
    - `toUpperCase(.)`

- Dokumentation
  - https://docs.oracle.com/javase/8/docs/api/

Zeichenfolgen: Überblick

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- ***Zeichenfolgen***
  - Sammlung von Characters, die geordnet im Speicher abgelegt werden.
- ***Zeichen-Literal***
  - Konstante Zeichenkette
  - Beispiel: `System.out.println("Programmieren 2");`
- ***Strings***
  - Objekt/Klasse, die Zeichenkette als "Character-Array" kapselt.
  - Klassen `String, StringBuffer und StringBuilder`
  - Beispiel: `String a = new String("Programmieren 2");`

	Immutable	**Mutable**
Threadsicher	`String`	`StringBuffer`
Nicht threadsicher		`StringBuilder`

Verfügen über gemeinsame Schnittstelle: `CharSequence`

Wann werden String-Objekte erzeugt?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Strings sind Objekte der Klasse `java.lang.String`

- Erzeugen eines String-Objekts auf zwei Arten
  - ***Literal-Methode***: `String str1 = "Hallo";`
    - JVM erzeugt automatisch String-Objekt, das auf konstante Zeichenkette verweist.
    - *Konstantenpool*: Jede konstante Zeichenkette existiert nur einmal.
  - ***Konstruktor-Methode***: `String str2 = new String("Hallo");`
    - Erzwingt, dass ein neues String-Objekt angelegt wird.

- Frage: Wie viele String-Objekte existieren am Ende?

```
String s1 = "Hallo";
String s2 = s1;
String s3 = "Hallo";
String s4 = new String ("Hallo");
```

String: Unveränderbarkeit

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- String-Objekte sind **immutable.**
  - Einfügen, Austauschen, Entfernen von Zeichen nicht möglich.
  - Stattdessen werden immer neue Strings erzeugt.

- Beispiel: **Konkatenation** durch '**+**'-Operation
  - `String a;`
  - `String b;`
  - `System.out.println(a + b) //→` **3. / neues String-Objekt**

- Strings sind also spezielle Objekte

```
String h1 = "Hans"; // h1 ist Referenz auf Stringobjekt mit Inhalt "Hans"
String h2 = "Hans"; // h2 ist Referenz auf dasselbe Objekt wie h1
h1 += "Meier";      // h1 zeigt auf ein neues Objekt (Ergebnis von '+')
```

String: Wichtige Grundmethoden

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- `length()`
  - Liefert die Länge des Strings.
- `boolean isEmpty()`
  - Liefert true zurück, falls String die Länge 0 hat.
- `charAt(int)`
  - Liefert einzelnes Zeichen aus String.
- `boolean equals(Object)`
  - Vergleicht zeichenweise auf ***Gleichheit***.
- `boolean contains(CharSequence)`
  - Testet ob Teil-String in Zeichenkette vorkommt.
- `int indexOf(char)` oder `int indexOf(String)`
  - Liefert die Fundstelle eines Zeichens bzw. Teil-Strings.

- Übung
  - `String a = "Programmieren";`
  - Wie greift man auf das letzte Zeichen des Strings a zu?
  - `char c = ???`

String: Vergleiche

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- ***Gleichheit:*** Inhaltlicher Vergleich zweier String-Objekte
  - Methode `boolean equals(String)`
  - Zahlreiche weitere Varianten: `equalsIgnoreCase(…)`
  - **Übung:** `equals` vs. "==" und *Konstantenpool*

```
String input = new Scanner(System.in).nextLine();  // user enters "abc"
String expected ="abc";
boolean b1 = input.equals("abc");  // Wert von b1 ??
boolean b2 = (input == "abc");     // Wert von b2 ??
boolean b3 = (expected == "abc");  // Wert von b3 ??
```

- **Lexikografische Vergleiche**
  - `s1.compareTo(s2)`
  - Vergleicht ob String `s2` lexikographisch vor String `s1` kommt.
  - Ergebnis:
    - `< 0` falls `s1` im Alphabet vor `s2` Bsp: `"hello".compareTo("java")`
    - `= 0` falls `s1` wertgleich zu `s2` Bsp: `"hello".compareTo("hello")`
    - `> 0` falls `s1` im Alphabet hinter `s2` Bsp: `"hello".compareTo("compiler")`

String: Extrahieren von Teilen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Extrahieren einzelner Zeichen: `charAt(int)`
- Extrahieren an Positionen: `substring(…)`:
  - `String substring (int from, int to)`
    Teil-Zeichenkette ab Index `from` bis (ausschließlich) Index `to`
  - `String substring (int from)`
- Zerlegen von Zeichenketten
  - `split(..)`: Zerlegung unter Berücksichtigung von Trennzeichen
    - Beispiel: `String[] segs = "abtcd".split("t")`
    - Es können reguläre Ausdrücke verwendet werden.
  - `Scanner`:
    - Erlaubt Einlesen von Zeilen (Trennzeichen "\n") mit Methode `nextLine()`
    - Erlaubt schrittweises Einlesen von Datentypen, z.B. Methode `nextint();`
- Zusammenfügen von Zeichenketten mit Trennzeichen: `StringJoiner`

```
StringJoiner sj = new StringJoiner(", ");
sj.add("1").add("2").add("3");
```

Übung

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wie testet man in Java ob eine String-Referenz null ist oder ob die Zeichenkette leer ist?
  - Gesucht Methode: boolean `isNullOrEmpty(String s)`

StringBuffer

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Unterstützt veränderbare (***mutable***) Zeichenketten
  - Modifiziert Zielobjekt, erzeugt bei Änderungen kein neues Objekt.
  - Kann nur durch Konstruktor erzeugt werden!

- `StringBuilder` vs. `StringBuffer`
  - `StringBuilder` ist etwas schneller.
  - `StringBuffer` ist *thread-safe*.

- Auswahl Methoden
  - `void` **`append`** `(`**`char`** `c)`
    Fügt Zeichen `c` hinten an Zeichenkette an.
  - `void` **`insert`** `(`**`int`** `at,` **`char`** `c)`
    Schiebt Zeichen `c` an Indexstelle `at` ein; Rest rutscht nach hinten.
  - `void` **`deleteCharAt`** `(`**`int`** `at)`
    Löscht Zeichen an Indexstelle `at`; Rest rutscht nach vorne.
  - `String` **`toString()`**
    Erzeugt aus der aktuellen Zeichenkette ein String-Objekt.

Beispiel: StringBuffer

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Geben Sie die Ausgabe des Programms an!

```
StringBuffer sb = new StringBuffer("Programmieren2" );
sb.append(',');
sb.append(" Mathe2, ").append("GDI2");
System.out.println(sb.toString());
```

Inhalt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Arrays
- Strings
- **Basisklasse Object**
- Java-Klassenbibliothek
  - Primitive Datentypen und Referenztypen
  - Packages
- Weitere Klassen
  - Wrapper-Klassen
  - Klasse System
  - Klasse Class

Literatur: C. Ullenboom, Java ist auch eine Insel
Kapitel 4, Kapitel 3.6 und 3.7

Alles in Java ist ein Objekt!

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- *"Wir befinden uns im Jahre 2016. Alles in Java ist ein Objekt … Wirklich alles?"*

- **Java**
  - Alles ist Unterklasse von der Basisklasse `Object`.
  - `https://docs.oracle.com/javase/8/docs/api/`

- **Ausnahmen**
  - Primitive Datentypen
  - Klasse `Array`
  - Klasse `String`.

Welche Datentypen existieren in Java?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Primitive Datentypen**
  - 8 verschiedene Typen
  - `boolean, char, byte, short, int, long, float, double`
  - Alle anderen Datentypen sind Objekt-ähnlich.

- **Referenztypen**
  - Beliebige Klassen und Objekte
    - Selbstdefiniert oder aus Klassenbibliothek
  - Strings sind spezielle Objekte
  - Arrays sind Spezielle Objekte
  - Vordefinierte Konstante `null` (leere Referenz)

- ***Fazit:*** In Java ist (fast) alles ein Objekt!

Referenztypen: Zuweisungen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Prinzipiell identische Verwendung wie primitive Datentypen
- Referenztypen sind ***Verweise*** → Semantik anders!
- **Beachte: Zuweisung erstellt keine Kopie eines Objekts**
  - Es wird der Verweis (=Zeiger) auf das jeweilige Objekt kopiert.
  - Das Objekt selbst wird nicht kopiert.
  - Methode `clone()` erforderlich für echte Kopie, siehe späteres Kapitel.

```
Example e1 = new Example();
e1.attr = "wert1";
Example e2 = new Example();
e2.attr = "wert2";
e2 = e1;
e2.attr = "wert3";
```

Welchen Wert hat am Ende `e1.attr`?

Referenztypen: Testen auf Identität und Gleichheit

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Identität (==)**
  - Liegen 2 Objekte an ***identischer Position*** im Speicher?
  - Testen mit: `if (object1 == object 2)`

- **Gleichheit (`equals`)**
  - Testen ob 2 Objekte ***inhaltlich gleich*** sind.
  - 2 Objekte können gleich sind, auch wenn sie nicht identisch sind, z.B. falls es sich um Kopien handelt.
  - Testen mit: if (object1.equals(object 2)
  - Details folgen bald.

```
Example e1 = new Example();
e1.attr = "wert";
Example e2 = new Example();
e2.attr = "wert";
boolean b = (e1 == e2);
```

Welchen Wert hat am Ende die Variable b?

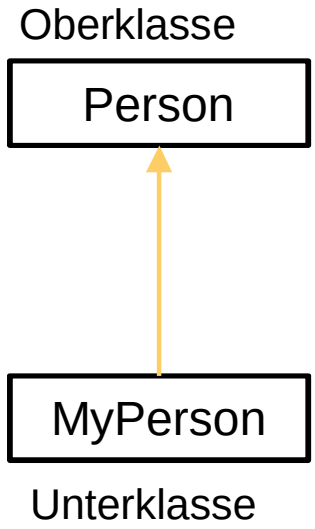
Operator `instanceof`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Prüft, ob Variable einen bestimmten Datentyp besitzt.
- Schlüsselwort in Java.

- **Beispiel:**
  - Ein Variable vom Typ `MyPerson` hat den Datentyp `MyPerson` und den Datentyp `Person`.

```
MyPerson p = new MyPerson("Müller", 29);
boolean b1 = p instanceof MyPerson;
boolean b2 = p instanceof Person;
```



Welchen Wert hat `b1` und `b2`? true oder false?

Welche Datentypen existieren in Java?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Primitive Datentypen**
  - 8 verschiedene Typen
  - `boolean, char, byte, short, int, long, float, double`
  - Alle anderen Datentypen sind Objekt-ähnlich.

- **Referenztypen**
  - Beliebige Klassen und Objekte
    - Selbstdefiniert oder aus Klassenbibliothek
  - ***Arrays*** sind spezielle Objekte
    - "Speziell": `int[] a = new int[10];`
  - ***Strings*** sind Spezielle Objekte
    - "Speziell": String s = new "Hallo";
  - Vordefinierte Konstante `null` (leere Referenz)

Beliebige Objekte, Strings und Arrays → Referenz-semantik

Bei Zuweisungen und Testen auf Gleichheit beachten!

Wichtige Interfaces der Java Standard Library

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- `Cloneable`
  - Markierungsschnittstelle
  - Objekte unterstützen Klonen mit der Methode: `public Object clone()`

- `Comparable`
  - Objekt kann **sich** mit einem anderen Objekt vergleichen.
  - `public int compareTo(Object obj)`

- `Comparator`
  - Vergleicht 2 Objekte miteinander.
  - `public int compare(Object obj1, Object obj2)`

- `Runnable`
  - Objekte enthalten *run*-Methode, die parallelisiert in Thread ausgeführt werden kann.

- `Serializable:`
  - Objekte können in Byte-Strom geschrieben werden, z.B. zum Speichern in Datei oder zur Datenübertragung.
  - `writeObject( ), readObject( )`

Interfaces: `Comparable` und `Comparator`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Natürliche Ordnung**: `Comparable`
  - `public int compareTo(Object obj)`
  - Implementiert Klasse diese Schnittstelle, so sind Objekte der Klasse vergleichbar.
  - Programmierer der Klasse legt jedoch fest, "wie" verglichen wird.
  - Beispiel: Ein Raum wird mit einem anderen Raum bzgl. Anzahl Sitzplätze verglichen.

- **Weitere Ordnung:** `Comparator`
  - `public int compare(Object obj1, Object obj2)`
  - Nötig, falls es ***mehrere verschiedene Vergleichskriterien*** für Objekte gibt.
  - Beispiel: "Räume" sollen einmal nach Anzahl Sitzplätze und einmal nach Quadratmetergröße sortiert werden.

- **Ergebnis** jeweils:
  - <0, wenn aktuelles bzw. linkes Objekt kleiner ist.
  - >0, wenn aktuelles bzw. linkes Objekt größer ist.
  - 0 bei "Gleichheit".

Objekte liegen also in der richtigen Reihenfolge vor

```
"A".compareTo("B") = -1
```

- Generischer Code
  - Beispiel: Sortieralgorithmen funktionieren auf allen Klassen, die Schnittstelle `Comparable` umsetzen.

Implementierung von `Comparable`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- `Dog` soll nach seiner Laufgeschwindigkeit (= `speed`) verglichen werden.
- **Lösung**: Implementiere Interface `Comparable`, **2 Varianten**
  - Mit *Raw Types* (vor Java 5)
  - Mit *Generics* (nach Java 5): siehe Kapitel "Collections".
- **Regel:** `compareTo(…) == 0` ↔ `equals(…) == true`

Raw Types, ohne Generics

```
public class Dog extends Pet
implements Trainable, Comparable
. . .

@Override
public int compareTo(Object o) {
  Dog other = (Dog) o;
  if (this.speed < other.speed)
    return -1;
  else if (this.speed > other.speed)
    return +1;
  else
    return 0;
}
```

Mit Generics

```
public class Dog extends Pet
implements Trainable, Comparable<Dog>


@Override
public int compareTo(Dog o) {
 if (this.strength < o.strength)
   return -1;
 else if (this.strength > o.strength)
   return +1;
 else
   return 0;
}
```

Beispiel: Implementierung von `Comparator`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- *Weitere Ordnung:* Ordne `Dog` (=`Pet`) nach Namen.
- **Lösung**: Implementiere Interface `Comparator` in eigener Klasse; wieder **2 Varianten:**
  - Mit *Raw Types* (vor Java 5)
  - Mit *Generics* (nach Java 5): siehe nächstes Kapitel.

```
public class PetByNameComparator
implements Comparator {

 @Override
 public int compare(Object o1, Object o2) {
    Pet p1 = (Pet) o1;
    Pet p2 = (Pet) o2;
    // re-use compareTo method from String
    return p1.getName().
               compareTo(p2.getName());
 }
}
```

Raw Types, ohne Generics

Mit Generics

```
public class PetByNameComparator
implements Comparator<Pet> {

 @Override
 public int compare(Pet p1, Pet p2) {
   // re-use compareTo method from string
   return p1.getName().
              compareTo(p2.getName());
 }
}
```

Comparable und Comparator

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Anwendung**
  - Comparable:

```
d1.compareTo(d2);
```

  - Comparator:

```
PetByNameComparator p = new PetByNameComparator();
p.compare(p1, p2);
```

- Jede Klasse, die Comparable (oder Comparator) implementiert, hat auch den Datentyp Comparable (oder Comparator) .
  - Schreiben von generischen (Sortier)algorithmen möglich, z.B.:
  - `public static void bubbleSort(Comparable[] objects)`

- **Raw Types oder Generics**
  - Ab Java 5 wird die Generics-Variante empfohlen, da etwas kürzer
  - Compilerwarnungen bei Verwendung von Raw Types lässt sich durch Annotation unterdrücken: `@SupressWarnings("rawtypes")`

Wieso ist `compare()` nicht `static`?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **https://stackoverflow.com/questions/21817/why-cant-i-declare-static-methods-in-an-interface?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa**

- **Kurze Antwort: weil `Comparator` alt ist.**

Inhalt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Arrays
- Strings
- Basisklasse Object
- **Java-Klassenbibliothek**
  - Primitive Datentypen und Referenztypen
  - Packages
- Weitere Klassen
  - Wrapper-Klassen
  - Klasse System
  - Klasse Class

Literatur: C. Ullenboom, Java ist auch eine Insel
Kapitel 4, Kapitel 3.6 und 3.7

Java Klassenbibliothek

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Programmiersprache Java besteht aus
  - Grammatik, Syntax, Schlüsselwörter
  - Klassenbibliothek: Mehr als 208 Pakete und 2000 Klassen

- **Ziele der Klassenbibliothek**
  - Bereitstellen wichtiger Hilfsfunktionen
  - Plattformunabhängigkeit: Funktionen abstrahieren plattformspezifische Eigenschaften

- **Beispiele**
  - Datenstrukturen → Collections
  - Ein- und Ausgabe (von Dateien)
  - Grafikprogrammierung → Java FX, Swing
  - Netzwerkprogrammierung
  - …

- Dokumentation unter:
  - https://docs.oracle.com/javase/8/docs/api/
  - IntelliJ: Cursor auf Klassenname und Quick Documentation (Ctrl+Q)

Aufbau von Java-Dateien

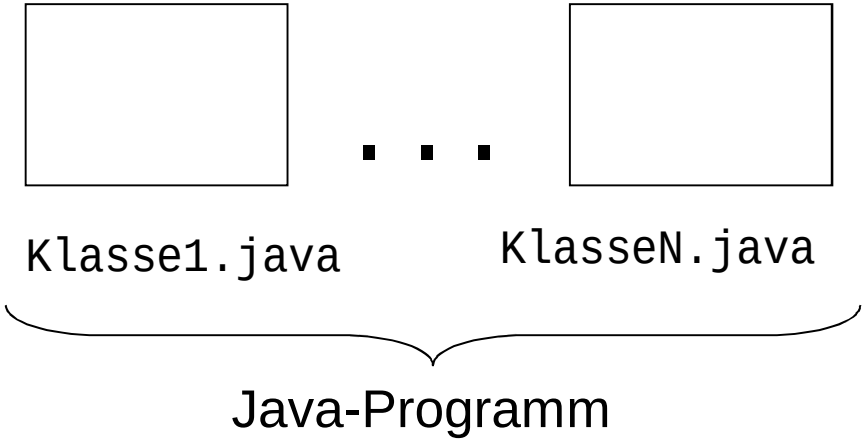
Technische Hochschule Rosenheim
Technical University of Applied Sciences

Aufbau einer `.java` Datei:

➔ Optional: `package`-Anweisung

➔ Optional: `import`-Anweisung

➔ Klassendefinitionen:
- Maximal eine Klassendefinition (Konvention)
- Dateiname identisch mit Name von Klasse
- Kann `main`-Methode enthalten:
  - Einstiegsklasse, Klasse als Applikation ausführbar
  - Startpunkt des Programms
  - koordiniert weitere Programmausführung



Packages

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wozu dient die **`package`**-Anweisung?
  - Inhaltliche Strukturierung und Abgrenzung der Sichtbarkeit
  - Alle Java-Dateien mit gleicher `package`-Anweisung gehören zum gleichen Paket.
  - Bei fehlender `package`-Anweisung: →`default`-package
  - Zusammenhang mit Ordnerstruktur im Dateisystem

- Wozu dient die **`import`**-Anweisung?
  - Mitteilung an Compiler, dass Klassen eines anderen Pakets benutzbar sein sollen.
  - Beispiel

```
import java.io.IOException
```

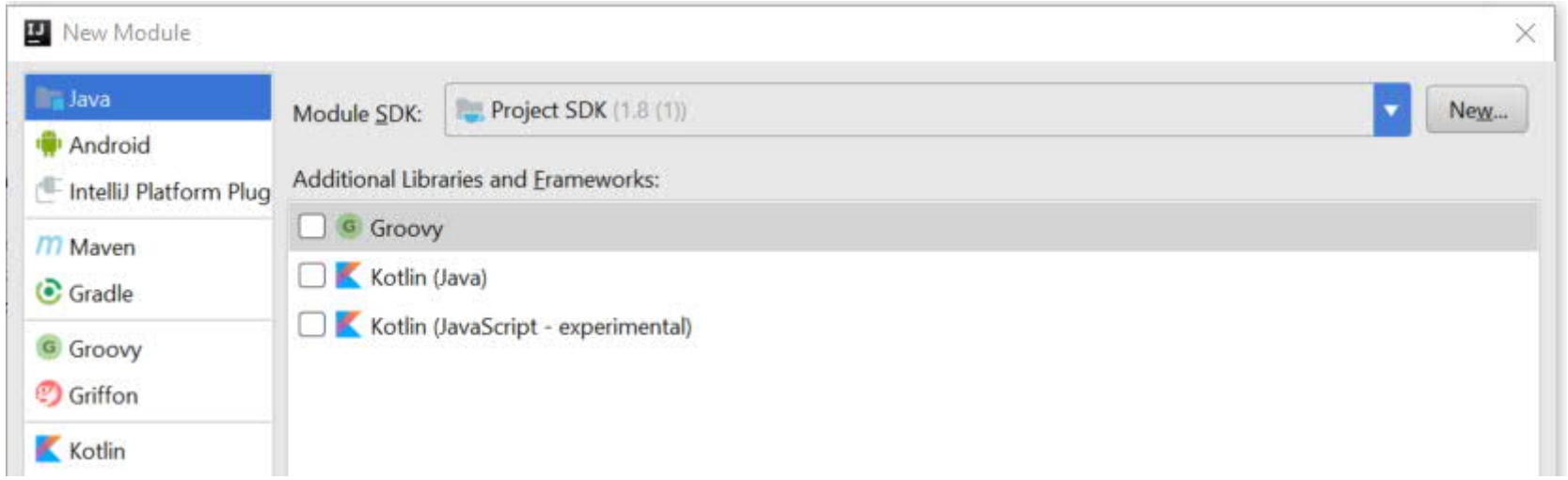
package Klasse

  - Import von einzelnen Klassen oder ganzer Pakete (`import java.util.*`)

Module (IntelliJ)

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Module
  - Gruppe von Java-Dateien, die man unabhängig voneinander kompilieren, testen und debuggen kann.
  - Module werden häufig in mehreren Projekten verwendet.
- Module == Konzept von IntelliJ
- Module ≠ Packages



Basisklasse `Object`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- `java.lang.Object` ist Oberklasse **aller** Klassen in Java.
  - Lässt man bei eigener Klasse `extends` weg, so wird implizit aus `Object` abgeleitet.
  - Es ist möglich Instanzen von Object zu erzeugen.

- `Object` hat ***keine* Attribute**

- **Methoden** der Klasse `Object`
  - `public String toString()`
  - `public boolean equals(Object obj)`
  - `public int hashCode()`
  - `protected Object clone()`
  - `public final Class getClass()`

"Alles in Java ist ein Objekt. Alle Objekte haben einen gemeinsamen Vorfahren: `Object`

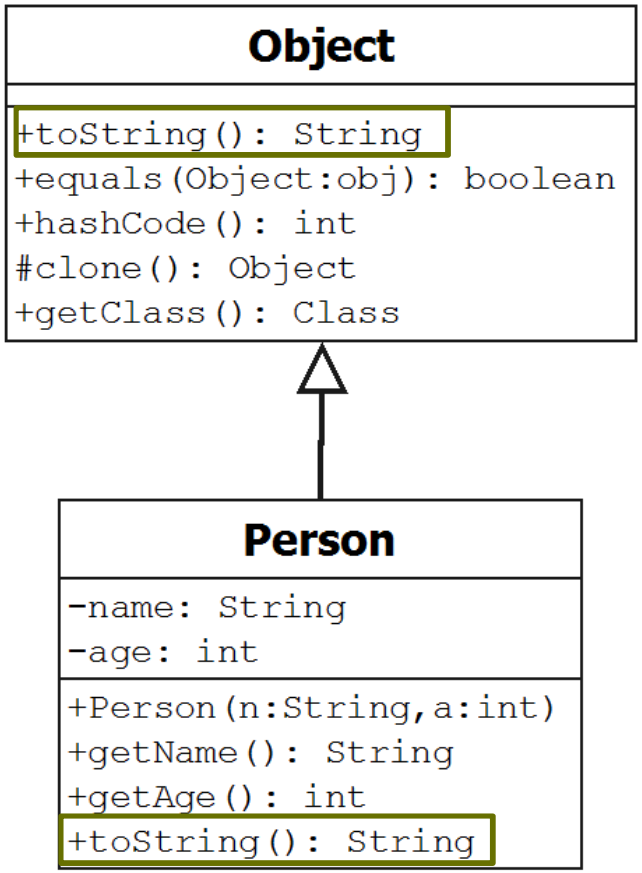
Methode `toString()` der Klasse `Object`

- `public String toString()`
  - Erzeugt eine String-Repräsentation des Objekts.
  - Liefert Zeichenkette, die Objekt in lesbarer Form beschreibt

- **Implementierung** in Klasse `Object`:
  - `getClass().getName() + '@' + Integer.toHexString(hashCode())`
  - <Klassenname>@<Hashwert/ID des Objekts>

- **Empfehlung:**
  - Für eigene Klassen: Methode überschreiben!

```
@Override
public String toString() {
    return name + "-" + phone;
}
```



Methode `equals(.)` der Klasse `Object`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

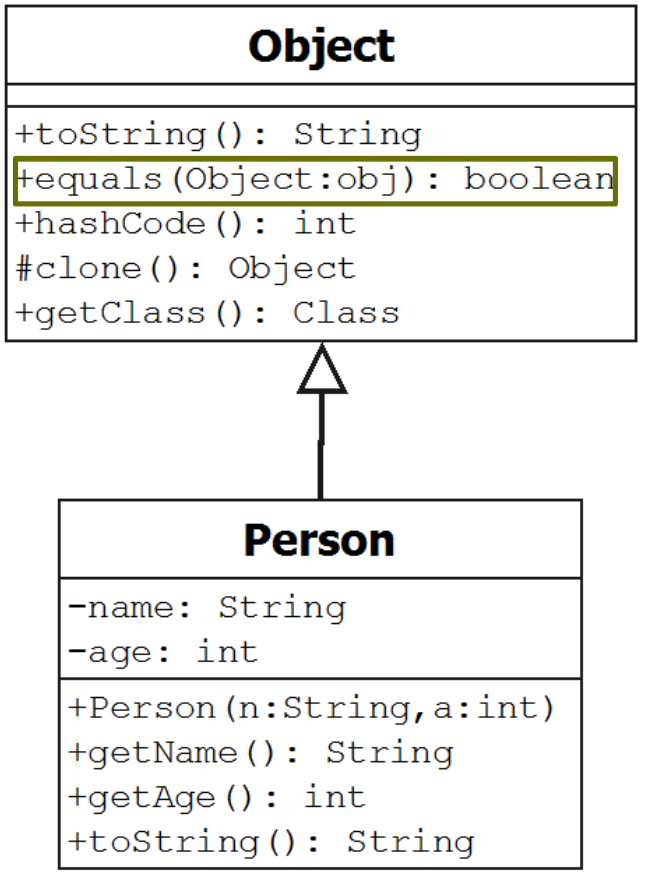
- Überprüfen der **Identität**: "==" bzw. "!="
  - Bei *primitiven Datentypen* gleichbedeutend mit Übereinstimmung der Integer-Werte.
  - Bei *Referenzdatentypen* (Objekte) bedeutet das nur, dass Referenzen (= Adressen im Speicher) übereinstimmen.

- **Gleichheit** der Objekte, auf die Referenztypen verweisen**:** `equals`

- Standardimplementierung in Klasse Object:

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

- **Empfehlung:**
  - Für eigene Klassen: Methode überschreiben.
  - Wann 2 Objekte gleich sein sollen, hängt von Anwendung ab.



Methode `equals(.)` der Klasse `Object`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- ***Annahme:***
  - Klasse `Example` überschreibt Methode `equals()` nun"vernünftig".
  - "Vernünftig" bedeutet: 2 Objekte sind "gleich", wenn deren Attribute übereinstimmen.

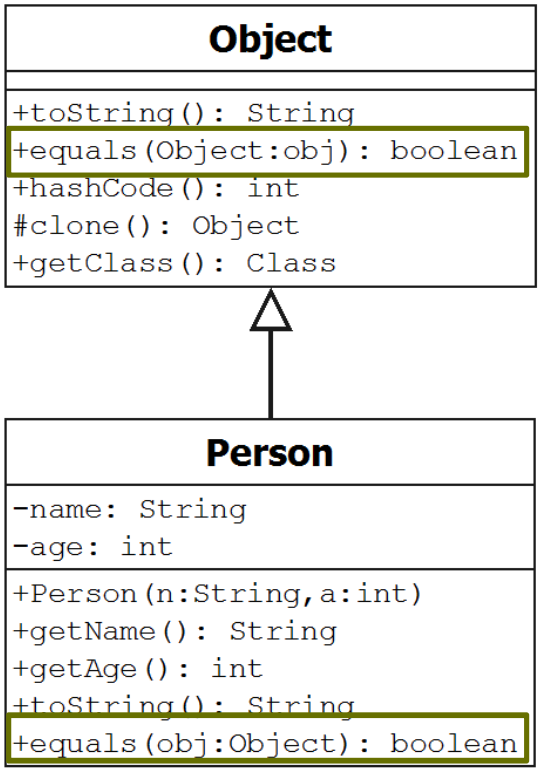
- ***Übung:***
  - Welchen Wert hat am Ende die Variable `b1`?
  - Welchen Wert sollte am Ende die Variable `b2` haben
    - Annahme: `equals` wurde in Klasse `Example` überschrieben

```
Example e1 = new Example();
e1.attr = "wert";
Example e2 = new Example();
e2.attr = "wert";
boolean b1 = (e1 == e2);
boolean b2 = (e1.equals(e2));
```

Schema: Überschreiben der Methode `equals()`

- **Anforderungen**:
  `boolean equals(Object obj)`
  1. Ergebnis des Vergleichs ist `false`, falls `obj null` ist.
  2. Prüfe auf Identität durch Vergleich mit `this` (optional).
  3. Teste ob `obj` den ***gleichen Datentyp*** hat wie Objekt, auf dem Methode aufgerufen wird.
  4. Evtl.: Methode `equals()` der Oberklasse aufrufen.
  5. Vergleiche Attribute der Objekte, abhängig von Anwendung.

- **Diskussion**
  - Erst im letzten Schritt kommen Eigenheiten der Anwendung zum Tragen.



Beispiel: Überschreiben der Methode `equals()`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
public boolean equals(Object obj) {

    if (obj == null) return false;    // (1) Vergleich mit 0


    if (obj == this) return true;     // (2) Prüfe auf Identität

    // (3) Teste ob gleicher Datentyp
    if (!this.getClass().equals(obj.getClass())) {
        return false;
    }

    // (4) Nur falls equals() in Oberklasse überschrieben wurde;
    //     Hier nicht sinnvoll, nur zur Vollständigkeit angegeben
    if (super.equals(obj) == false) return false;

    // (5) Vergleich der Datenelemente, abhängig von Anwendung
    Person other = (Person) obj;      // Typecast
    if (this.name.equals(other.name) && this.age == other.age)
        return true;
    return false;
}
```

Übung

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Überschreiben Sie für die Klasse String die Methode `equals`!
  - 2 Strings sind dann gleich, wenn alle Ihre Zeichen übereinstimmen.
  - Greifen Sie hier vereinfachend auf die Methode `compareTo` zurück!

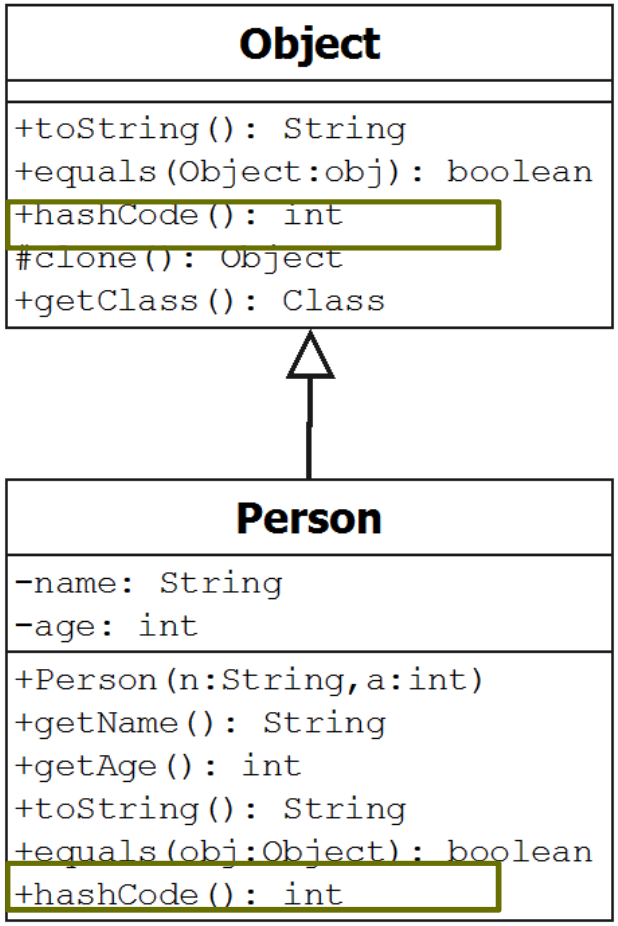
Methode `hashCode()` der Klasse `Object`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- `public int hashCode()`
  - Liefert zu jedem Objekt einen Hashwert, der das Objekt möglichst eindeutig identifiziert.
  - Berechnung durch eine *Hashfunktion.*
  - Datenstrukturen, die nach dem Hashing-Verfahren arbeiten (z.B. `java.util.HashMap`) benötigen solche Hashwerte.

- **Standardimplementierung**
  - Stützt sich auf Speicheradresse des Objekts.
  - `public native int hashCode();`

- **Richtlinien**
  - Objekt bleibt unverändert → Hashwert bleibt unverändert.
  - 2 Objekte sind gemäß `equals` gleich → Rückgabe des gleichen Hashwertes.



Schema: Überschreiben der Methode `hashCode()`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Überschreibt man `equals()`, sollte auch `hashCode()` überschrieben werden.
- Muster zur Berechnung des Hashwertes, falls mehrere Attribute h1, h2, h3, … ein Objekt identifizieren:

```
int result = h1.hashCode;
result = 31 * result + h2.hashCode();
result = 31* result + h3.hashCode();
```

- ***Beispiel:***

- Primzahl!
- *31 schnell zu implementieren

```
class Person {
    public int hashCode() {
        int result = name.hashCode(); // String implementiert hashCode()
        result = 31 * result + age;   // int: kein Aufruf von hashCode() nötig
        return result;
    }
    . . .
}
```

Methode `clone(.)` der Klasse `Object`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
Example e1 = new Example();
e1.attr = "wert1";
Example e2 = new Example();
e2.attr = "wert2";
e2 = e1;
```

Keine Kopie einer Instanz! e1 und e2 zeigen danach auf das **identische** Objekt im Speicher.

- **Probleme** beim Kopieren
  - Zuweisung eines Objektes ist nur Kopieren eines "Zeigers".
  - Objekte können selbst Attribute besitzen, die alle kopiert werden müssen.

- 2 Möglichkeiten für ***echte Kopie / Replikation***
  - **Copy-Konstruktor:**
    - Konstruktor bekommt als Parameter Instanz eines bereits vorhandenen Objektes und überträgt dessen Zustand auf das neu anzulegende Objekt.
  - Überschreiben der Methode: `protected Object clone()`

Object: clone() – Flache vs. Tiefe Kopie

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Flache Kopie** (Standardfall)
  - Es werden alle Attribute, die primitive Datentypen sind, kopiert.
  - Falls Attribut ein Objekt ist, wird nur die Referenz kopiert.

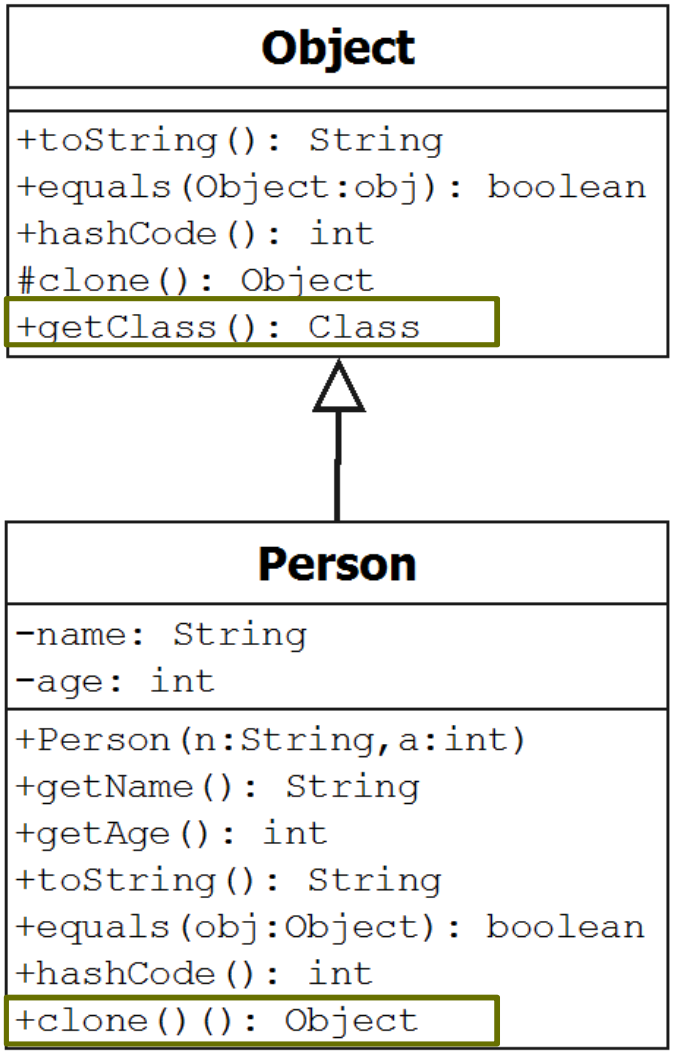
- **Tiefe Kopie** (Handarbeit notwendig)
  - Alle Attribute, die selbst Objekte sind, müssen echt kopiert (dupliziert) werden.
  - Für Objekte x muss gelten:
    - `x.clone() != x`
    - `x.clone().getClass() == x.getClass()`
    - `x.clone().equals(x)`

- 2 Ansätze für tiefe Kopie
  - Manuelles Anlegen eines neuen Objektes, Kopieren aller Attribute und Rückgabe einer Referenz auf das neue Objekt (***Copy-Konstruktor***)
  - Zurückgreifen auf das Laufzeitsystem und damit auf die Methode `clone()`, die bereits in Klasse `Object` implementiert ist (siehe nächste Folien)

Schema: Überschreiben der Methode `clone()`

- **Redefinieren** von `clone()` als `public`.
  - Um zu kennzeichnen, dass Klasse die Methode `clone()` überschrieben hat, diese mit "`implements Cloneable`" kennzeichnen.

- Regeln für Überschreiben von `clone()`
  - Erzeugen einer Kopie des Basisklassen-Objekts durch Aufruf von `super.clone()`
  - Umwandeln der Kopie auf eigenen Typ (***Downcast***)
  - Falls Objekt Attribut enthält, deren Typ selbst Referenztyp ist
    - Tiefe Kopie erzeugen
    - Dafür Attribute mit Referenztyp einzeln mit untergeordneten `clone()`-Aufrufen kopieren
  - Zurückgeben des Duplikats



Object: clone() – Beispiel für Überschreiben

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
class Person implements Cloneable {
    private String name;
    private int age;
    private MyClass attr;

    public Object clone() throws CloneNotSupportedException {
        Person newPers = (Person) super.clone();
        newPers.attr = (MyClass) attr.clone();
        newPers.age = age;
        newPers.name = name;
        newPers.name = name.clone();
        return newPers;
    }
}
```

Fehlermeldung, falls `clone()` auf Objekt einer Klasse aufgerufen wird, die `clone()` nicht implementiert.

```
Person first = new Person(…);
Person second = first.clone()
```

- String ist in Java ein Objekt → Aufruf der Methode `clone()` auf String-Objekt.

Inhalt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Arrays
- Strings
- Basisklasse Object
- Java-Klassenbibliothek
  - Primitive Datentypen und Referenztypen
  - Packages
- **Weitere Klassen**
  - Wrapper-Klassen
  - Klasse System
  - Klasse Class

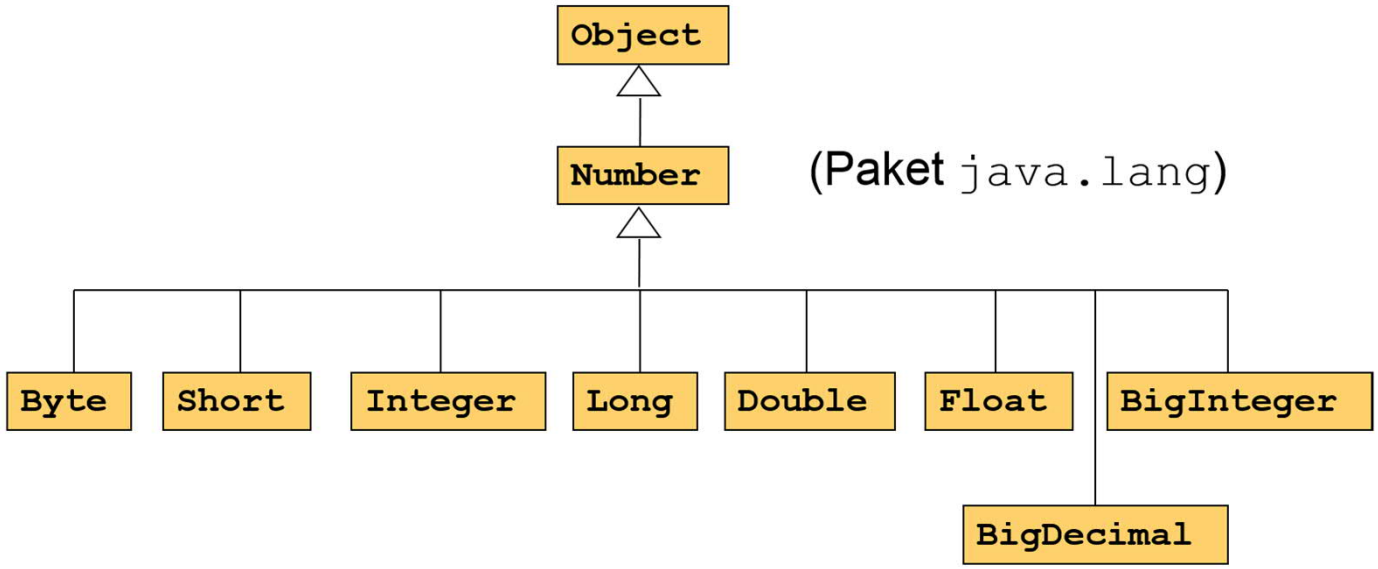
Literatur: C. Ullenboom, Java ist auch eine Insel
Kapitel 4, Kapitel 3.6 und 3.7

Wrapper-Klassen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Für jeden primitiven Datentyp gibt es eine ***Wrapper-Klasse***
  - Kapseln primitive Datentypen in einem Objekt.

- Warum Wrapper-Klassen?
  - Bieten statische Methoden für Konvertierung in Strings und zurück.
  - Notwendig für Datenstrukturen der Klassenbibliothek (Collections), die nur Objekte speichern können.
  - Generics (siehe später) gibt es nur mit Wrapper-Klassen.



Wrapper-Klassen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Erzeugen von Wrapper-Objekten
  - mit Konstruktoren
  - statische `valueOf`-Methoden
  - mittels ***Boxing***
- Alle Wrapper-Klassen überschrieben `equals()`
- Wrapper-Klassen sind **immutable**!
- **Autoboxing**
  - Automatisches Umwandeln zwischen primitiven Datentypen und Wrapper-Objekten
- Operationen ohne Wrapper-Klasse sind teilweise performanter!

```
// Erzeugung durch Konstruktoren
Boolean   b = new Boolean(true);
Character c = new Character('X');
Byte      y = new Byte(1);
Short     s = new Short(2);
Integer   i = new Integer(3);
Long      l = new Long(4);
Float     f = new Float(3.14f);
Double    d = new Double(3.14);

// Erzeugung mit valueOf
Long l1 = Long.valueOf(1000L)

// Erzeugung mit Boxing
Integer i1 = 42;

int i2 = 4711;
// Boxing -> j = Integer.valueOf(i)
Integer j = i2;
// Unboxing -> k = j.intValue()
int k = j;
```

Klasse `java.math.BigInteger`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Vorteile**
  - Darstellung beliebig großer Zahlen
  - Zahlreiche Zusatzmethoden wie z.B. modulare Arithmetik

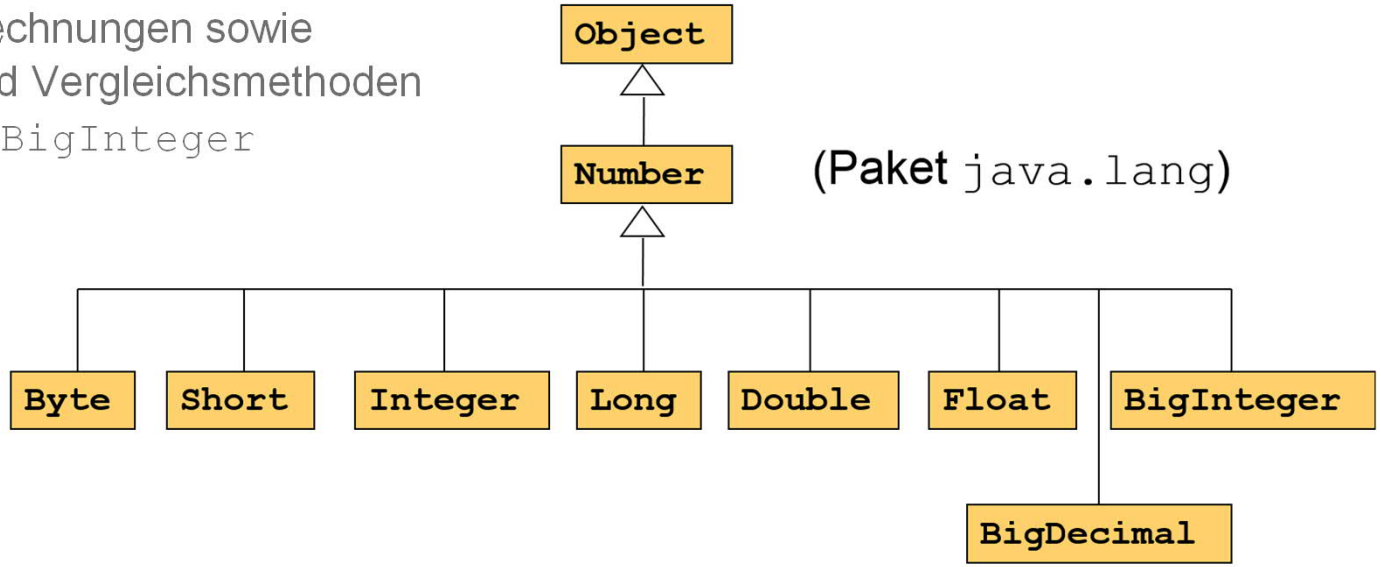
- Objekte der Klasse sind **immutable!**
  - `public BigInteger(String val)`
  - `public BigInteger(String val, int radix)`
  - `static BigInteger valueOf(long val)`

- ***Klassenspezifische Konstanten***
  - `BigInteger.ZERO`
  - `BigInteger.ONE`
  - `BigInteger.TEN`

- Zahlreiche Methoden für arithmetische Berechnungen, Vergleiche, Umwandlung in primitive Datentypen
  - https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html

Klasse `java.lang.BigDecimal`

- Darstellung **beliebig genauer Fließkommazahlen**
  - https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html
- Bestehen aus
  - einer Ziffernfolge (Objekt vom Typ `BigInteger`) und
  - einer Skalierung (Anzahl der Nachkommastellen)
- Objekte der Klasse sind **immutable.**
- Methoden zur Durchführung
  - arithmetischer Berechnungen sowie
  - Konvertierungs- und Vergleichsmethoden
  - Ähnlich der Klasse `BigInteger`



Was ist hier falsch?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
BigInteger  big   = new BigInteger("12345678901234567890123");
BigInteger  small = BigInteger.valueOf(25000);

String  s  = small.toString();  // "25000"
String  t  = small.toString(7); // zur Basis 7: "132613"

big = big.add (small);          // addiert small „zu“ big

BigDecimal bd1 = new BigDecimal(big);
BigDecimal bd2 = new BigDecimal(3.14);
BigDecimal bd3 = new BigDecimal("3.14");

bd2.add(bd3);
```

Klasse `java.lang.System`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

`System` enthält einige nützliche, statische Objekte und Methoden

- Standard Input- (`System.in`) und Output-Streams (`System.out`)
  - Beispiel:

```
System.out.println(„Hello world!“);
```

- Zugriff auf *Umgebungsvariablen* mittels Properties
  - Properties: Eigenschaften, die vom Laufzeitsystem zur Verfügung gestellt werden
  - Zugriff auf einzelne Properties: `getProperty(String key)`
  - Beispiele für Properties.: `java.version`, `java.vm.version`, `java.class.path`, `os.name`, `os.version`, `user.name`, ...
  - Beispiel für Zugriff:

```
String javaVersion = System.getProperty("java.version");
```

- Beenden der JVM: `System.exit(n);`
  - Beispiel:

```
System.exit(1);
```

Klasse `Class`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Repräsentiert Klassen und Interfaces in Java.
  - Laufzeitumgebung legt automatisch für jede Klasse ein Objekt vom Typ `Class` an.
  - Kann nicht direkt instanziiert werden, kein `public`-Konstruktor!

- Class-Objekt enthält Metainformationen zu einer Klasse, z.B.
  - Name der Klasse
  - Verweis auf Oberklasse
  - Zugriff auf Annotationen der Klasse

- Zu einem Objekt enthält man das zugehörige Objekt Class durch Aufruf der Methode ***getClass()***, die von der Klasse Object automatisch geerbt wird.

- Beispiel: Ausgeben des Klassennamens eines Objekts.

```
Person p = new Person("Mueller", 10);
System.out.println(p.getClass().getName());
```

Zusammenfassung

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Primitive Datentypen vs. Referenztypen
  - Referenztypen zeigen nur auf das Objekt im Speicher
- Arrays und Strings sind spezielle Objekte / Referenztypen
- Alle Objekte erben implizit von der Basisklasse `Object`
  - Inhaltliche Gleichheit vs. Identität
  - Überschreiben von zentralen Methoden der Klasse `Object`, z.B. `equals`
- Weitere Klassen
  - Wrapper-Klassen
  - Klasse System
  - Klasse Class
- Die Java-Klassenbibliothek selbst ist ein Paradebeispiel für Objektorientierung.

Quellenverzeichnis

Technische Hochschule Rosenheim
Technical University of Applied Sciences

[1] C. Ullenboom. *Java ist auch eine Insel*, 11. Auflage, Galileo Computing

[2] http://www.comedix.de/lexikon/db/vorwort.php (abgerufen am 12.04.2016)