Script generated by TTT

Title: groh: profile1 (26.05.2016)

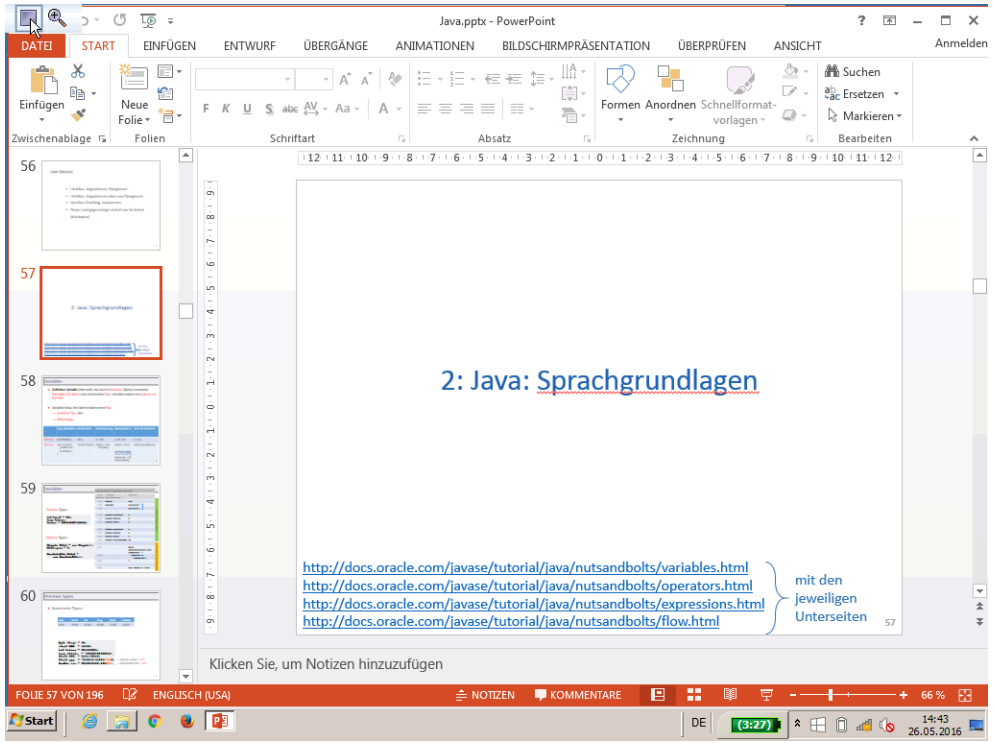
Date: Thu May 26 14:43:54 CEST 2016

Duration: 93:17 min

Pages: 128

Live-Demos:

- LittleBee, AngryHornet, FlyingInsect
- LittleBee, AngryHornet erben von FlyingInsect
- Interface ICanSting, implements
- Flower und gegenseitiger Aufruf von Verhalten (Methoden)



Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleichheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht! ←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleichheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht! ←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleichheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht! ←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleichheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht! ←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.
- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!←→ genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

58

Variablen

- **Definition Variable** (informell): mit einem Bezeichner (Name) versehener Platzhalter für Werte eines bestimmten Typs. Variablen haben eine Adresse im Speicher.

- Variablen (bzw. ihre Werte) haben einen Typ:
  - primitiver Typ oder
  - Referenztyp

	(Typ-)Definition	Deklaration	Instantiierung	Manipulation	Test auf Gleicheit
Primitiv	(vordefiniert)	int a;	a = 117;	a = b + 42;	a == b;

Referenz	class Student {
	// Fields and
	// methods ...
	}
		Student heiner;	heiner = new Student();	heiner = horst;
				je nach Betrachtungsweise auch (Vorsicht!<--> genaue Definition! Siehe Folie 58ff.):
				heiner.age = 21;
				heiner.yawn();
					heiner.equals(horst);

Variablen

Primitive Typen:

```
int horst = 101;
long heiner;
heiner = 5638465837465845;
```

Referenz Typen:

```
Bicycle bike1 = new Bicycle();
bike1.gear = 3;

MountainBike bike2 =
    new MountainBike();
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt	
1123	horst	101	Daten (bspw. Attribute)
1124	heiner	56384658	
1125		37465845	
...	...	...	
1150	bike1.cadence	0	
1151	bike1.speed	0	
1152	bike1.gear	3	
...	...	...	
1330	bike2.cadence	0	
1331	bike2.speed	0	
1332	bike2.gear	1	
1333	bike2.seatHeight	15	
...	...	...	
4027		void changeCadence(int newValue) {	Instruktionen der Methoden
4028		cadence = newValue;	
4029		}	
...	...	...	
4035		int horst = 101;	

Variablen

Primitive Typen:

```
int horst = 101;
long heiner;
heiner = 5638465837465845;
```

Referenz Typen:

```
Bicycle bike1 = new Bicycle();
bike1.gear = 3;

MountainBike bike2 =
    new MountainBike();
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt	
1123	horst	101	Daten (bspw. Attribute)
1124	heiner	56384658	
1125		37465845	
...	...	...	
1150	bike1.cadence	0	
1151	bike1.speed	0	
1152	bike1.gear	3	
...	...	...	
1330	bike2.cadence	0	
1331	bike2.speed	0	
1332	bike2.gear	1	
1333	bike2.seatHeight	15	
...	...	...	
4027		void changeCadence(int newValue) {	Instruktionen der Methoden
4028		cadence = newValue;	
4029		}	
...	...	...	
4035		int horst = 101;	

Variablen

Primitive Typen:

```
int horst = 101;
long heiner;
heiner = 5638465837465845;
```

Referenz Typen:

```
Bicycle bike1 = new Bicycle();
bike1.gear = 3;

MountainBike bike2 =
    new MountainBike();
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
1123	horst	101
1124	heiner	56384658
1125		37465845
...	...	...
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1330	bike2.cadence	0
1331	bike2.speed	0
1332	bike2.gear	1
1333	bike2.seatHeight	15
...	...	...
4027		void changeCadence(int newValue) {
4028		cadence = newValue;
4029		}
...	...	...
4035		int horst = 101;

Daten (bspw. Attribute)

Instruktionen der Methoden

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}

= 3245343455.555 * 10^{67}

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}

= 3245343455.555 * 10^{67}

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}

= 3245343455.555 * 10^{67}

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

60

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

60

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

60

Primitive Typen

- Numerische Typen:

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ $\in \mathbb{Q}$ „$\approx \mathbb{R}$“

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit
[-2^7, 2^7-1] = [-128, 127]	[-2^{15}, 2^{15}-1] = [-32768, 32767]	[-2^{31}, 2^{31}-1] = [-2147483648, 2147483647]	[-2^{63}, 2^{63}-1] = [-9223372036854775808, 9223372036854775807]	[+/- ≈ 1.4*10^{-45}, +/- ≈ 3.4*10^{38}]	[+/- ≈ 4.9*10^{-324}, +/- ≈ 1.8 10^{308}]

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ $\in \mathbb{Q}$ „$\approx \mathbb{R}$“

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit
[-2^7, 2^7-1] = [-128, 127]	[-2^{15}, 2^{15}-1] = [-32768, 32767]	[-2^{31}, 2^{31}-1] = [-2147483648, 2147483647]	[-2^{63}, 2^{63}-1] = [-9223372036854775808, 9223372036854775807]	[+/- ≈ 1.4*10^{-45}, +/- ≈ 3.4*10^{38}]	[+/- ≈ 4.9*10^{-324}, +/- ≈ 1.8 10^{308}]

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ $\in \mathbb{Q}$ „$\approx \mathbb{R}$“

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit
[-2^7, 2^7-1] = [-128, 127]	[-2^{15}, 2^{15}-1] = [-32768, 32767]	[-2^{31}, 2^{31}-1] = [-2147483648, 2147483647]	[-2^{63}, 2^{63}-1] = [-9223372036854775808, 9223372036854775807]	[+/- ≈ 1.4*10^{-45}, +/- ≈ 3.4*10^{38}]	[+/- ≈ 4.9*10^{-324}, +/- ≈ 1.8 10^{308}]

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ $\in \mathbb{Q}$ „$\approx \mathbb{R}$“

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit
$[-2^7, 2^7-1]$ = [-128, 127]	$[-2^{15}, 2^{15}-1]$ = [-32768, 32767]	$[-2^{31}, 2^{31}-1]$ = [-2147483648, 2147483647]	$[-2^{63}, 2^{63}-1]$ = [-9223372036854775808, 9223372036854775807]	[+/- ≈ $1.4*10^{-45}$, +/- ≈ $3.4*10^{38}$]	[+/- ≈ $4.9*10^{-324}$, +/- ≈ $1.8\ 10^{308}$]

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000}
2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity  (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004  (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ $\in \mathbb{Q}$ „$\approx \mathbb{R}$“

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit
$[-2^7, 2^7-1]$ = [-128, 127]	$[-2^{15}, 2^{15}-1]$ = [-32768, 32767]	$[-2^{31}, 2^{31}-1]$ = [-2147483648, 2147483647]	$[-2^{63}, 2^{63}-1]$ = [-9223372036854775808, 9223372036854775807]	[+/- ≈ $1.4*10^{-45}$, +/- ≈ $3.4*10^{38}$]	[+/- ≈ $4.9*10^{-324}$, +/- ≈ $1.8\ 10^{308}$]

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ $\in \mathbb{Q}$ „$\approx \mathbb{R}$“

byte	short	int	long	float	double
8 bit	16 bit	32 bit	64 bit	32 bit	64 bit
$[-2^7, 2^7-1]$ = [-128, 127]	$[-2^{15}, 2^{15}-1]$ = [-32768, 32767]	$[-2^{31}, 2^{31}-1]$ = [-2147483648, 2147483647]	$[-2^{63}, 2^{63}-1]$ = [-9223372036854775808, 9223372036854775807]	[+/- ≈ $1.4*10^{-45}$, +/- ≈ $3.4*10^{38}$]	[+/- ≈ $4.9*10^{-324}$, +/- ≈ $1.8\ 10^{308}$]

```
byte flags = 63;
short bbb  = 10133;
int heiner = 234103234;
long lilalo  = -83628735682345;
float fff  = 5464.00345;
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000}
2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity   (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004   (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000}
2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity   (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004   (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

62

Darstellung / Approximation von Zahlen

- mit beschränkter Anzahl Bits: Nur Darstellung von Zahlen bis zu einer bestimmten Größe möglich
- Folgen: möglicherweise Überlauf:
  - Bsp.: Annahme: 4 Bit zur Darstellung von positiven natürlichen Zahlen x vorhanden → $x \in [0,15]$: `0000, 0001, 0010, … , 1111`
  - Operation $15 + 1$: `1111 + 0001 = 10000.`
    Es sind aber nur 4 Stellen vorhanden! → $15 + 1 = 0$
- Konsequenz: Größenbereiche der Zahltypen nicht überschreiten!

Literatur: Wikipedia Artikel: „ IEEE floating point” , „Signed number representations” oder
http://www.math.kit.edu/ianm2/lehre/progjava2009w/seite/aktuelles/media/zahldarstellung.handout.pdf (URL, Okt. 2014)

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000}
2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity   (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004   (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

62

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\}\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000} 2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity   (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004   (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\}\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000} 2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity   (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004   (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

62

Darstellung / Approximation von reellen Zahlen

```
float ggg  = -345545.34534E-12f;
double sss = 3245343455.555E67;
```

= -345545.34534 * 10^{-12}
= 3245343455.555 * 10^{67}
$\}\in \mathbb{Q}$ „$\approx \mathbb{R}$“

- mit beschränkter Anzahl Bits: Nur Approximation von reellen Zahlen (bspw. π) bzw. rationalen Zahlen mit nicht abbrechender Dezimalbruchentwicklung (bspw. 1/3) möglich. (Standard: IEEE 754).
- Folgen: Numerische Fehler möglich. Bsp:

2^{1000} 2^{5000}

```
double e = Math.pow(2.0d,1000.0d); //1.0715086071862673E301
double f = Math.pow(2.0d,5000.0d); //Infinity   (Zahlbereichsüberschreitung)
double g = 0.1d + 0.1d + 0.1d; //0.30000000000000004   (Rundungsfehler)
```

siehe auch: http://introcs.cs.princeton.edu/java/91float/

- → Test auf Gleichheit zweier float oder double Werte x, y nicht mit $x == y$ sondern mit $|x - y| < \epsilon$;
  viele weitere Konsequenzen → numerische Mathematik
- Es gibt in Java auch Möglichkeiten mit beliebiger Genauigkeit zu rechnen (bspw. mit Klasse BigDecimal)

62

Darstellung / Approximation von Zahlen

- mit beschränkter Anzahl Bits: Nur Darstellung von Zahlen bis zu einer bestimmten Größe möglich
- Folgen: möglicherweise Überlauf:
  - Bsp.: Annahme: 4 Bit zur Darstellung von positiven natürlichen Zahlen x vorhanden → $x \in [0,15]$: `0000, 0001, 0010, … , 1111`
  - Operation $15 + 1$: `1111 + 0001 = 10000`.
    Es sind aber nur 4 Stellen vorhanden! → $15 + 1 = 0$
- Konsequenz: Größenbereiche der Zahltypen nicht überschreiten!

Literatur: Wikipedia Artikel: „ IEEE floating point” , „Signed number representations” oder
http://www.math.kit.edu/ianm2/lehre/progjava2009w/seite/aktuelles/media/zahldarstellung.handout.pdf (URL, Okt. 2014)

Darstellung / Approximation von Zahlen

- mit beschränkter Anzahl Bits: Nur Darstellung von Zahlen bis zu einer bestimmten Größe möglich
- Folgen: möglicherweise Überlauf:
  - Bsp.: Annahme: 4 Bit zur Darstellung von positiven natürlichen Zahlen x vorhanden → $x \in [0,15]$: `0000, 0001, 0010, … , 1111`
  - Operation $15 + 1$: `1111 + 0001 = 10000`.
    Es sind aber nur 4 Stellen vorhanden! → $15 + 1 = 0$
- Konsequenz: Größenbereiche der Zahltypen nicht überschreiten!

Literatur: Wikipedia Artikel: „ IEEE floating point" , „Signed number representations" oder http://www.math.kit.edu/ianm2/lehre/progjava2009w/seite/aktuelles/media/zahldarstellung.handout.pdf (URL, Okt. 2014)

Darstellung / Approximation von Zahlen

- mit beschränkter Anzahl Bits: Nur Darstellung von Zahlen bis zu einer bestimmten Größe möglich
- Folgen: möglicherweise Überlauf:
  - Bsp.: Annahme: 4 Bit zur Darstellung von positiven natürlichen Zahlen x vorhanden → $x \in [0,15]$: `0000, 0001, 0010, … , 1111`
  - Operation $15 + 1$: `1111 + 0001 = 10000`.
    Es sind aber nur 4 Stellen vorhanden! → $15 + 1 = 0$
- Konsequenz: Größenbereiche der Zahltypen nicht überschreiten!

Literatur: Wikipedia Artikel: „ IEEE floating point" , „Signed number representations" oder http://www.math.kit.edu/ianm2/lehre/progjava2009w/seite/aktuelles/media/zahldarstellung.handout.pdf (URL, Okt. 2014)

Darstellung / Approximation von Zahlen

- mit beschränkter Anzahl Bits: Nur Darstellung von Zahlen bis zu einer bestimmten Größe möglich
- Folgen: möglicherweise Überlauf:
  - Bsp.: Annahme: 4 Bit zur Darstellung von positiven natürlichen Zahlen x vorhanden → $x \in [0,15]$: `0000, 0001, 0010, … , 1111`
  - Operation $15 + 1$: `1111 + 0001 = 10000`.
    Es sind aber nur 4 Stellen vorhanden! → $15 + 1 = 0$
- Konsequenz: Größenbereiche der Zahltypen nicht überschreiten!

Literatur: Wikipedia Artikel: „ IEEE floating point" , „Signed number representations" oder http://www.math.kit.edu/ianm2/lehre/progjava2009w/seite/aktuelles/media/zahldarstellung.handout.pdf (URL, Okt. 2014)

Primitive Typen: Boolean und Char

boolean	char
1 bit	16 bit
{ true, false }	{ … !, „, §, $, %, &, …., a, b, c, …., ∭,…,∰,…,⇳, ガ,キ,ギ,…,㋗,㋘,㋙,…, 樵,槅, 皾,… 얙,얬,얜…,ششش, …,→,↓,■, …}

```
char ccc    = 'm';
char ccc2   = '\n';

boolean isCool = true;
```

`\n` means "new line"

Primitive Typen: Boolean und Char

boolean	char
1 bit	16 bit
{ true, false }	{ … !, „, §, $, %, &, …., a, b, c, …., ∭,….,∰,….,≎, ガ,キ,ギ,….,㋗,㋘,㋙,…., 樵,樶, 樷,… 액,앳,앤…,شش, …,→,↓,■, …}

```
char ccc   = 'm';
char ccc2  = '\n';

boolean isCool = true;
```

\n means "new line"

Primitive Typen: Boolean und Char

boolean	char
1 bit	16 bit
{ true, false }	{ … !, „, §, $, %, &, …., a, b, c, …., ∭,….,∰,….,≎, ガ,キ,ギ,….,㋗,㋘,㋙,…., 樵,樶, 樷,… 액,앳,앤…,شش, …,→,↓,■, …}

```
char ccc   = 'm';
char ccc2  = '\n';

boolean isCool = true;
```

\n means "new line"

Primitive Typen: Boolean und Char

boolean	char
1 bit	16 bit
{ true, false }	{ … !, „, §, $, %, &, …., a, b, c, …., ∭,….,∰,….,≎, ガ,キ,ギ,….,㋗,㋘,㋙,…., 樵,樶, 樷,… 액,앳,앤…,شش, …,→,↓,■, …}

```
char ccc   = 'm';
char ccc2  = '\n';

boolean isCool = true;
```

\n means "new line"

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
      new Bicycle();
Bicycle bike2 =
      new Bicycle();
```

```
boolean c;
c = bike1.equals(bike2);
   // c == true
c = (bike1 == bike2);
   // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
…	…	…
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	1
…	…	…
1327	bike2	<1405>
…	…	…
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
…	…	…

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();



boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	1
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();



boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	1
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

65

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();



boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	1
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

65

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();

bike1.gear = 3;



boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	**bike1.gear**	**3**
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

66

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();

bike1.gear = 3;

bike1 = bike2;



boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == true
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();

bike1.gear = 3;

bike1 = bike2;



boolean c;
c = bike1.equals(bike2);
    // c == true
c = (bike1 == bike2);
    // c == true
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

67

Referenztypen – spezieller Wert `null`

der spezielle Wert **`null`** ist eine Referenz auf nichts (Variable zeigt auf nichts mehr, besteht aber als Variable weiter).

Der Speicherplatz von Objekten, auf die niemand mehr zeigt (in diesem Fall das ursprüngliche `bike1` Objekt (in 1150, 1151, 1152)) werden vom Garbage Collector irgendwann für Anderes freigegeben

```
bike1.gear = 3;

bike1 = bike2;

bike2 = null;

boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	null
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

poof!

Referenztypen – spezieller Wert `null`

der spezielle Wert **`null`** ist eine Referenz auf nichts (Variable zeigt auf nichts mehr, besteht aber als Variable weiter).

Der Speicherplatz von Objekten, auf die niemand mehr zeigt (in diesem Fall das ursprüngliche `bike1` Objekt (in 1150, 1151, 1152)) werden vom Garbage Collector irgendwann für Anderes freigegeben

```
bike1.gear = 3;

bike1 = bike2;

bike2 = null;

boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	null
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

poof!

68

Referenztypen – spezieller Wert `null`

der spezielle Wert **`null`** ist eine Referenz auf nichts (Variable zeigt auf nichts mehr, besteht aber als Variable weiter).

Der Speicherplatz von Objekten, auf die niemand mehr zeigt (in diesem Fall das ursprüngliche `bike1` Objekt (in 1150, 1151, 1152)) werden vom Garbage Collector irgendwann für Anderes freigegeben

```
bike1.gear = 3;

bike1 = bike2;

bike2 = null;

boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	null
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

poof!

Referenztypen – spezieller Wert `null`

der spezielle Wert **`null`** ist eine Referenz auf nichts (Variable zeigt auf nichts mehr, besteht aber als Variable weiter).

Der Speicherplatz von Objekten, auf die niemand mehr zeigt (in diesem Fall das ursprüngliche `bike1` Objekt (in 1150, 1151, 1152)) werden vom Garbage Collector irgendwann für Anderes freigegeben

```
bike1.gear = 3;

bike1 = bike2;

bike2 = null;

boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	null
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

poof!

68

Referenztypen – spezieller Wert `null`

der spezielle Wert **`null`** ist eine Referenz auf nichts (Variable zeigt auf nichts mehr, besteht aber als Variable weiter).

Der Speicherplatz von Objekten, auf die niemand mehr zeigt (in diesem Fall das ursprüngliche `bike1` Objekt (in 1150, 1151, 1152)) werden vom Garbage Collector irgendwann für Anderes freigegeben

```
bike1.gear = 3;

bike1 = bike2;

bike2 = null;

boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

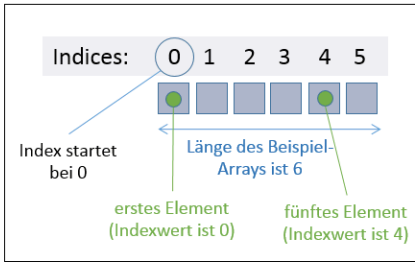
Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	null
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

poof!

68

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp



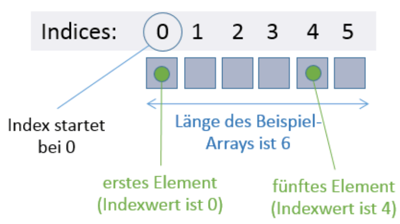

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp



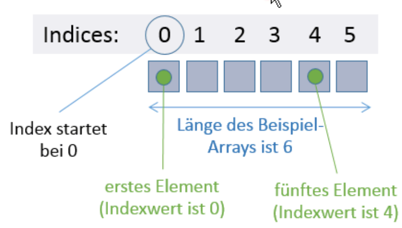

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp




```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

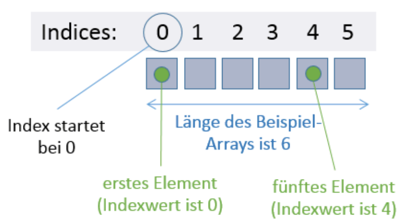
Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

69

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp




```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

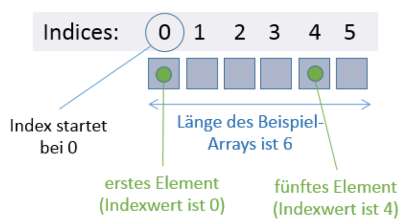
Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

69

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp




```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

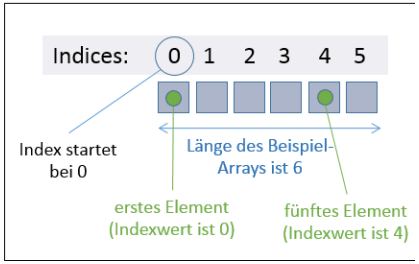
Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

69

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp




```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp

Indices: 0 1 2 3 4 5
Index startet bei 0
Länge des Beispiel-Arrays ist 6
erstes Element (Indexwert ist 0)
fünftes Element (Indexwert ist 4)

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

String[] someOtherArray;
someOtherArray = new String[30];
someOtherArray[17] = "bla bla";

AnyClass[] thirdArray;
thirdArray = new AnyClass[45];
thirdArray[44] = new AnyClass();
thirdArray[22 * 2].someMethod();
```

Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

69

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp (hier int[])

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	**someArray**	<1150>
1150	someArray[0]	0
1151	someArray[1]	0
1152	someArray[2]	7
...	...	...
1327	**anotherArray**	<1328>
1328	anotherArray[0]	0
1329	anotherArray[1]	8
1330	anotherArray[2]	0
...	...	...
...	...	...

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp (hier int[])

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	**someArray**	<1150>
1150	someArray[0]	0
1151	someArray[1]	0
1152	someArray[2]	7
...	...	...
1327	**anotherArray**	<1328>
1328	anotherArray[0]	0
1329	anotherArray[1]	8
1330	anotherArray[2]	0
...	...	...
...	...	...

70

Referenztypen

- Referenztyp-Variable „zeigt“ auf ein Objekt
- Typ der Variable ist die Klasse des Objekts

```
Bicycle bike1 =
       new Bicycle();
Bicycle bike2 =
       new Bicycle();

bike1.gear = 3;




boolean c;
c = bike1.equals(bike2);
    // c == false
c = (bike1 == bike2);
    // c == false
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1150>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	**bike1.gear**	**3**
...	...	...
1327	bike2	<1405>
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp (hier int[])

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	**someArray**	<1150>
1150	someArray[0]	0
1151	someArray[1]	0
1152	someArray[2]	7
...	...	...
1327	**anotherArray**	<1328>
1328	anotherArray[0]	0
1329	anotherArray[1]	8
1330	anotherArray[2]	0
...	...	...
...	...	...

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp (hier int[])

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;

someArray = anotherArray;

boolean b = (someArray[1] == 8);
// b == true
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	**someArray**	<1328>
1150	someArray[0]	0
1151	someArray[1]	0
1152	someArray[2]	7
...	...	...
1327	**anotherArray**	<1328>
1328	anotherArray[0]	0
1329	anotherArray[1]	8
1330	anotherArray[2]	0
...	...	...
...	...	...

Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp (hier int[])

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;

someArray = anotherArray;

boolean b = (someArray[1] == 8);
// b == true
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...		...
1149	someArray	<1328>
1150	someArray[0]	0
1151	someArray[1]	0
1152	someArray[2]	7
	...	
1327	anotherArray	<1328>
1328	anotherArray[0]	0
1329	anotherArray[1]	8
1330	anotherArray[2]	0
...	...	...
...	...	...

- length attribut

```
int l = someArray.length;
// l == 3
```


Arrays

- Array: indiziertes Feld (Reihung) einer festen Anzahl von Elementen eines Typs (primitiv oder Referenz)
- ist selbst von Referenztyp (hier int[])

```
int[] someArray = new int[3];
int[] anotherArray = new int[3];

someArray[2] = 7;
anotherArray[1] = 8;

someArray = anotherArray;

boolean b = (someArray[1] == 8);
// b == true
```

Vereinfachtes Speicher-Modell

Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...		...
1149	someArray	<1328>
1150	someArray[0]	0
1151	someArray[1]	0
1152	someArray[2]	7
	...	
1327	anotherArray	<1328>
1328	anotherArray[0]	0
1329	anotherArray[1]	8
1330	anotherArray[2]	0
...	...	...
...	...	...

- length attribut

```
int l = someArray.length;
// l == 3
```


Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
−	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;`
		`float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
−	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;`
		`float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
−	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;` `float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
−	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;` `float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
−	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;` `float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Unäre Operatoren

+	Unärer Plus Operator (nicht so interessant)	`int a = -1; int b = +a;` `// b == -1`
−	Unärer Minus Operator	`int a = -1; int b = -a;` `// b == 1`
++	Inkrement um 1	`int a = 0; a++;` `// a == 1;`
−−	Dekrement by 1	`int a = 1; a--;` `// a == 0;`
!	Negation eines boolean	`boolean b = true; c = !b;` `// c==false;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
−	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;` `float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Unäre Operatoren

+	Unärer Plus Operator (nicht so interessant)	`int a = -1; int b = +a;` `// b == -1`
−	Unärer Minus Operator	`int a = -1; int b = -a;` `// b == 1`
++	Inkrement um 1	`int a = 0; a++;` `// a == 1;`
−−	Dekrement by 1	`int a = 1; a--;` `// a == 0;`
!	Negation eines boolean	`boolean b = true; c = !b;` `// c==false;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Arithmetische Operatoren

+	Addition	`1 + 1` `d + aaa` `cc = b + 1.7d;` `int a = 1 + 1;`
-	Subtraktion	`3 - 7` `int b = c - 9;` `float f = 10.02f - 23.56f;`
*	Multiplikation	`blub = fd * 0.1f;` `double d = z * z;`
/	Division	`int a = 17 / 9;` `// a == 1;`
		`float eee = 13.0f / 2.0f;` `// ee == 6.5f;`
%	Rest	`int a = 17 % 9` `// a == 8;`

Unäre Operatoren

+	Unärer Plus Operator (nicht so interessant)	`int a = -1; int b = +a;` `// b == -1`
-	Unärer Minus Operator	`int a = -1; int b = -a;` `// b == 1`
++	Inkrement um 1	`int a = 0; a++;` `// a == 1;`
--	Dekrement by 1	`int a = 1; a--;` `// a == 0;`
!	Negation eines boolean	`boolean b = true; c = !b;` `// c==false;`

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Gleichheit und andere relationale Operatoren

==	Equal to	`boolean a = (1 == 1);` `// a == true`
!=	Not equal to	`boolean a = (1 != 1);` `// a == false`
>	Greater than	`boolean a = (17 > 12));` `// a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Gleichheit und andere relationale Operatoren

==	Equal to	`boolean a = (1 == 1);` `// a == true`
!=	Not equal to	`boolean a = (1 != 1);` `// a == false`
>	Greater than	`boolean a = (17 > 12));` `// a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Logische Grundoperatoren und Bedingungen

&&	Conditional-AND	`a = false; b = true; c = a && b;` `// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;` `// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)	

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Gleichheit und andere relationale Operatoren

==	Equal to	`boolean a = (1 == 1);` `// a == true`
!=	Not equal to	`boolean a = (1 != 1);` `// a == false`
>	Greater than	`boolean a = (17 > 12));` `// a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Logische Grundoperatoren und Bedingungen

&&	Conditional-AND	`a = false; b = true; c = a && b;` `// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;` `// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)	

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Gleichheit und andere relationale Operatoren

==	Equal to	`boolean a = (1 == 1);` `// a == true`
!=	Not equal to	`boolean a = (1 != 1);` `// a == false`
>	Greater than	`boolean a = (17 > 12));` `// a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Logische Grundoperatoren und Bedingungen

&&	Conditional-AND	`a = false; b = true; c = a && b;` `// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;` `// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)	

Einschub: Aussagenlogik

a && b logisch ‚und', ‚Konjunktion' $a \wedge b$	b = true	b = false
a = true	true	false
a = false	false	false

a || b logisch ‚oder', ‚Disjunktion' $a \vee b$	b = true	b = false
a = true	true	true
a = false	true	false

! a logisch ‚nicht', ‚Negation' $\neg a$	
a = true	false
a = false	true

Zusammenbau komplexerer Ausdrücke:

```
!( (a && b) || !a )
```

$$\neg ((a \wedge b) \vee \neg a)$$

Einschub: Aussagenlogik

a && b logisch ‚und', ‚Konjunktion' $a \wedge b$	b = true	b = false
a = true	true	false
a = false	false	false

a || b logisch ‚oder', ‚Disjunktion' $a \vee b$	b = true	b = false
a = true	true	true
a = false	true	false

! a logisch ‚nicht', ‚Negation' $\neg a$	
a = true	false
a = false	true

Zusammenbau komplexerer Ausdrücke:

```
!( (a && b) || !a )
```

$$\neg ((a \wedge b) \vee \neg a)$$

Einschub: Aussagenlogik

a && b logisch ‚und', ‚Konjunktion' $a \wedge b$	b = true	b = false
a = true	true	false
a = false	false	false

a || b logisch ‚oder', ‚Disjunktion' $a \vee b$	b = true	b = false
a = true	true	true
a = false	true	false

! a logisch ‚nicht', ‚Negation' $\neg a$	
a = true	false
a = false	true

Zusammenbau komplexerer Ausdrücke:

```
!( (a && b) || !a )
```

$$\neg ((a \wedge b) \vee \neg a)$$

Einschub: Aussagenlogik

a && b logisch ‚und', ‚Konjunktion' a ∧ b	b = true	b = false
a = true	true	false
a = false	false	false

a || b logisch ‚oder', ‚Disjunktion' a ∨ b	b = true	b = false
a = true	true	true
a = false	true	false

! a logisch ‚nicht', ‚Negation' ¬ a	
a = true	false
a = false	true

Zusammenbau weiterer Operatoren:

Implikation: $a \rightarrow b == \neg a \vee b$

logisch ‚impliziert', ‚Implikation' a → b	b = true	b = false
a = true	true	false
a = false	true	true

Einschub: Aussagenlogik

a && b logisch ‚und', ‚Konjunktion' a ∧ b	b = true	b = false
a = true	true	false
a = false	false	false

a || b logisch ‚oder', ‚Disjunktion' a ∨ b	b = true	b = false
a = true	true	true
a = false	true	false

! a logisch ‚nicht', ‚Negation' ¬ a	
a = true	false
a = false	true

Zusammenbau weiterer Operatoren:

Exklusiv-Oder (XOR): $a \oplus b == (a \vee b) \wedge \neg(a \wedge b)$

logisch ‚XOR', ‚Exklusiv-Oder' a ⊕ b	b = true	b = false
a = true	false	true
a = false	true	false

Operatoren

Zuweisungs (Assignment) Operator

= `a = b+1;` `boolean ccc = (a==b);` `bike2 = bike1.copy();`

Zwei wichtige Operatoren für Referenztypen

Dereferenzierungs-Operator (dot-operator ".")

- Zugriff auf Attribute, Methodenaufrufe

```
bike1.cadence = 4;
String s1 = s1.concatenate(s2);
bike1.changeGear(5);
```

Objekterzeugungs-Operator

new erzeugt ein Objekt

```
Vector z = new Vector();
Bicycle bike = new Bicycle();
```

Zwei wichtige Operatoren für Referenztypen

Dereferenzierungs-Operator (dot-operator ".")

- Zugriff auf Attribute, Methodenaufrufe

```
bike1.cadence = 4;
String s1 = s1.concatenate(s2);
bike1.changeGear(5);
```

Objekterzeugungs-Operator

`new` erzeugt ein Objekt

```
Vector z = new Vector();
Bicycle bike = new Bicycle();
```


Zwei wichtige Operatoren für Referenztypen

Dereferenzierungs-Operator (dot-operator ".")

- Zugriff auf Attribute, Methodenaufrufe

```
bike1.cadence = 4;
String s1 = s1.concatenate(s2);
bike1.changeGear(5);
```

Objekterzeugungs-Operator

`new` erzeugt ein Objekt

```
Vector z = new Vector();
Bicycle bike = new Bicycle();
```

82

Zwei wichtige Operatoren für Referenztypen

Dereferenzierungs-Operator (dot-operator ".")

- Zugriff auf Attribute, Methodenaufrufe

```
bike1.cadence = 4;
String s1 = s1.concatenate(s2);
bike1.changeGear(5);
```

Objekterzeugungs-Operator

`new` erzeugt ein Objekt

```
Vector z = new Vector();
Bicycle bike = new Bicycle();
```

82

Operatoren

- Es gibt eine festgelegte automatische Präzendenzreihenfolge für Operatoren (Beispiel „Punkt vor Strich“ bei * und +)
- Klammern "(" ... ")" erzwingen allerdings jede gewünschte Präzedenz. Beispiel:

```
int a;
a = 7 + 4 * 8 % 3;        // a == 9
a = ((7 + 4) * 8) % 3;   // a == 1
```

Operatoren

- Es gibt eine festgelegte automatische Präzendenzreihenfolge für Operatoren (Beispiel „Punkt vor Strich“ bei * und +)
- Klammern "(" ... ")" erzwingen allerdings jede gewünschte Präzedenz. Beispiel:

```
int a;
a = 7 + 4 * 8 % 3;        // a == 9
a = ((7 + 4) * 8) % 3;   // a == 1
```


Operatoren

- Es gibt eine festgelegte automatische Präzendenzreihenfolge für Operatoren (Beispiel „Punkt vor Strich“ bei * und +)
- Klammern "(" ... ")" erzwingen allerdings jede gewünschte Präzedenz. Beispiel:

```
int a;
a = 7 + 4 * 8 % 3;        // a == 9
a = ((7 + 4) * 8) % 3;   // a == 1
```


Operatoren

- Es gibt eine festgelegte automatische Präzendenzreihenfolge für Operatoren (Beispiel „Punkt vor Strich“ bei * und +)
- Klammern "(" ... ")" erzwingen allerdings jede gewünschte Präzedenz. Beispiel:

```
int a;
a = 7 + 4 * 8 % 3;        // a == 9
a = ((7 + 4) * 8) % 3;   // a == 1
```


Remark: Type-Conversion

- Variablen eines Typs lassen sich (unter bestimmten Voraussetzungen in Variablen eines anderen Typs umwandeln:

```
int a = 15;
float f = (float)a;  //now f==15.0f
```

- dies funktioniert auch mit Referenztypen:

```
MountainBike mb = new MountainBike();
Bicycle[] bikes = new Bicycle[5];
bikes[3] = mb;     //a MountainBike is a Bicycle
bikes[0] = new Bicycle();
MountainBike newMBReference;
newMBReference = (MountainBike)(bikes[3]);
      //bikes[3] is in principle a reference to a Bicycle
```

Operatoren

Operatoren kombinieren (Werte von) Variablen und Literale (Konstanten) und liefern Werte:

Gleichheit und andere relationale Operatoren

==	Equal to	`boolean a = (1 == 1);` `// a == true`
!=	Not equal to	`boolean a = (1 != 1);` `// a == false`
>	Greater than	`boolean a = (17 > 12));` `// a == true;`
>=	Greater than or equal to	etc.
<	Less than	
<=	Less than or equal to	

Logische Grundoperatoren und Bedingungen

&&	Conditional-AND	`a = false; b = true; c = a && b;` `// c == false;`
||	Conditional-OR	`a = false; b = true; c = a || b;` `// c == true;`
?:	Ternary (shorthand for if-then-else statement, use `if-then-else` instead!)	

Remark: Type-Conversion

- Variablen eines Typs lassen sich (unter bestimmten Voraussetzungen in Variablen eines anderen Typs umwandeln:

```
int a = 15;
float f = (float)a;  //now f==15.0f
```

- dies funktioniert auch mit Referenztypen:

```
MountainBike mb = new MountainBike();
Bicycle[] bikes = new Bicycle[5];
bikes[3] = mb;      //a MountainBike is a Bicycle
bikes[0] = new Bicycle();
MountainBike newMBReference;
newMBReference = (MountainBike)(bikes[3]);
      //bikes[3] is in principle a reference to a Bicycle
```


Remark: Type-Conversion

- Variablen eines Typs lassen sich (unter bestimmten Voraussetzungen in Variablen eines anderen Typs umwandeln:

```
int a = 15;
float f = (float)a;  //now f==15.0f
```

- dies funktioniert auch mit Referenztypen:

```
MountainBike mb = new MountainBike();
Bicycle[] bikes = new Bicycle[5];
bikes[3] = mb;      //a MountainBike is a Bicycle
bikes[0] = new Bicycle();
MountainBike newMBReference;
newMBReference = (MountainBike)(bikes[3]);
      //bikes[3] is in principle a reference to a Bicycle
```


Ausdrücke (Expressions)

Expression: Legale Kombination aus Konstanten, Variablen und Operatoren (inklusive Methodenaufrufe und Objekterzeugung mit `new`)

- Jede Expression hat einen (evaluiert zu einem) Wert der einen bestimmten Typ hat

Beispiel:

gegeben:

```
int a = 73;
Bicycle bike;
```

Expression	evaluiert zu	Typ
`48`	`48`	`int`
`2.0 / 3.0`	`0.6666666666..`	`double`
`true && false`	`false`	`boolean`
`15 / 8`	`1`	`int`
`(17 + (3 * 9)) % 3`	`2`	`int`
`a + 1`	`74`	`int`
`a = 9`	`9`	`int`
`new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`bike = new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`new double[20]`	(Referenz auf Array von `double`)	`double[]`
`bike.cadence`	`0`	`int`

Ausdrücke (Expressions)

Expression: Legale Kombination aus Konstanten, Variablen und Operatoren (inklusive Methodenaufrufe und Objekterzeugung mit `new`)

- Jede Expression hat einen (evaluiert zu einem) Wert der einen bestimmten Typ hat

Beispiel:

gegeben:

```
int a = 73;
Bicycle bike;
```

Expression	evaluiert zu	Typ
`48`	`48`	`int`
`2.0 / 3.0`	`0.6666666666...6`	`double`
`true && false`	`false`	`boolean`
`15 / 8`	`1`	`int`
`(17 + (3 * 9)) % 3`	`2`	`int`
`a + 1`	`74`	`int`
`a = 9`	`9`	`int`
`new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`bike = new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`new double[20]`	(Referenz auf Array von `double`)	`double[]`
`bike.cadence`	`0`	`int`

Ausdrücke (Expressions)

Expression: Legale Kombination aus Konstanten, Variablen und Operatoren (inklusive Methodenaufrufe und Objekterzeugung mit `new`)

- Jede Expression hat einen (evaluiert zu einem) Wert der einen bestimmten Typ hat

Beispiel:

gegeben:

```
int a = 73;
Bicycle bike;
```

Expression	evaluiert zu	Typ
`48`	`48`	`int`
`2.0 / 3.0`	`0.6666666666...6`	`double`
`true && false`	`false`	`boolean`
`15 / 8`	`1`	`int`
`(17 + (3 * 9)) % 3`	`2`	`int`
`a + 1`	`74`	`int`
`a = 9`	`9`	`int`
`new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`bike = new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`new double[20]`	(Referenz auf Array von `double`)	`double[]`
`bike.cadence`	`0`	`int`

85

Ausdrücke (Expressions)

Expression: Legale Kombination aus Konstanten, Variablen und Operatoren (inklusive Methodenaufrufe und Objekterzeugung mit `new`)

- Jede Expression hat einen (evaluiert zu einem) Wert der einen bestimmten Typ hat

Beispiel:

gegeben:

```
int a = 73;
Bicycle bike;
```

Expression	evaluiert zu	Typ
`48`	`48`	`int`
`2.0 / 3.0`	`0.6666666666...6`	`double`
`true && false`	`false`	`boolean`
`15 / 8`	`1`	`int`
`(17 + (3 * 9)) % 3`	`2`	`int`
`a + 1`	`74`	`int`
`a = 9`	`9`	`int`
`new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`bike = new Bicycle()`	(Referenz auf Bicycle Objekt)	`Bicycle`
`new double[20]`	(Referenz auf Array von `double`)	`double[]`
`bike.cadence`	`0`	`int`

85

Expressions: Seiteneffekte

- Manche Expressions haben sogenannte Seiteneffekte (in den meisten Fällen ist dies der einzig wichtige Aspekt)

Beispiel:

gegeben:

```
int a = 48;
int b;
```

Expression	Wert	Seiteneffekt
`a = 84`	`84`	Wert 84 zu a zuweisen
`b = (a = 20)`	`20`	Wert 20 zu a und b zuweisen
`new Bicycle()`	(Referenz auf Bicycle Objekt)	Kreire und initialisiere eine neue Instanz (ein neues Objekt) der Klasse `Bicycle` im Speicher
`new double[20]`	(Referenz auf Array von `double`)	Kreire und initialisiere ein neues Array von 20 `double` Variablen im Speicher
`a++`	`48`	Wert 49 zu a zuweisen
`b = a++`	`48`	Werte 48 an b und 49 an a zuweisen
`++a`	`49`	Wert 49 zu a zuweisen
`b = ++a`	`49`	Werte 49 an b und 49 an a zuweisen

Expressions: Seiteneffekte

- Manche Expressions haben sogenannte Seiteneffekte (in den meisten Fällen ist dies der einzig wichtige Aspekt)

Beispiel:

gegeben:

```
int a = 48;
int b;
```

Expression	Wert	Seiteneffekt
`a = 84`	84	Wert 84 zu a zuweisen
`b = (a = 20)`	20	Wert 20 zu a und b zuweisen
`new Bicycle()`	(Referenz auf Bicycle Objekt)	Kreire und initialisiere eine neue Instanz (ein neues Objekt) der Klasse `Bicycle` im Speicher
`new double[20]`	(Referenz auf Array von `double`)	Kreire und initialisiere ein neues Array von 20 `double` Variablen im Speicher
`a++`	48	Wert 49 zu a zuweisen
`b = a++`	48	Werte 48 an b und 49 an a zuweisen
`++a`	49	Wert 49 zu a zuweisen
`b = ++a`	49	Werte 49 an b und 49 an a zuweisen

Expressions: Seiteneffekte

- Manche Expressions haben sogenannte Seiteneffekte (in den meisten Fällen ist dies der einzig wichtige Aspekt)

Beispiel:

gegeben:

int a = 48;
int b;

Expression	Wert	Seiteneffekt
a = 84	84	Wert 84 zu a zuweisen
b = (a = 20)	20	Wert 20 zu a und b zuweisen
new Bicycle()	(Referenz auf Bicycle Objekt)	Kreire und initialisiere eine neue Instanz (ein neues Objekt) der Klasse Bicycle im Speicher
new double[20]	(Referenz auf Array von double)	Kreire und initialisiere ein neues Array von 20 double Variablen im Speicher
a++	48	Wert 49 zu a zuweisen
b = a++	48	Werte 48 an b und 49 an a zuweisen
++a	49	Wert 49 zu a zuweisen
b = ++a	49	Werte 49 an b und 49 an a zuweisen

86

Expressions: Seiteneffekte

- Manche Expressions haben sogenannte Seiteneffekte (in den meisten Fällen ist dies der einzig wichtige Aspekt)

Beispiel:

gegeben:

int a = 48;
int b;

Expression	Wert	Seiteneffekt
a = 84	84	Wert 84 zu a zuweisen
b = (a = 20)	20	Wert 20 zu a und b zuweisen
new Bicycle()	(Referenz auf Bicycle Objekt)	Kreire und initialisiere eine neue Instanz (ein neues Objekt) der Klasse Bicycle im Speicher
new double[20]	(Referenz auf Array von double)	Kreire und initialisiere ein neues Array von 20 double Variablen im Speicher
a++	48	Wert 49 zu a zuweisen
b = a++	48	Werte 48 an b und 49 an a zuweisen
++a	49	Wert 49 zu a zuweisen
b = ++a	49	Werte 49 an b und 49 an a zuweisen

86

Expressions: Seiteneffekte

- Manche Expressions haben sogenannte Seiteneffekte (in den meisten Fällen ist dies der einzig wichtige Aspekt)

Beispiel:

gegeben:

int a = 48;
int b;

Expression	Wert	Seiteneffekt
a = 84	84	Wert 84 zu a zuweisen
b = (a = 20)	20	Wert 20 zu a und b zuweisen
new Bicycle()	(Referenz auf Bicycle Objekt)	Kreire und initialisiere eine neue Instanz (ein neues Objekt) der Klasse Bicycle im Speicher
new double[20]	(Referenz auf Array von double)	Kreire und initialisiere ein neues Array von 20 double Variablen im Speicher
a++	48	Wert 49 zu a zuweisen
b = a++	48	Werte 48 an b und 49 an a zuweisen
++a	49	Wert 49 zu a zuweisen
b = ++a	49	Werte 49 an b und 49 an a zuweisen

86

Expressions: Seiteneffekte

- Manche Expressions haben sogenannte Seiteneffekte (in den meisten Fällen ist dies der einzig wichtige Aspekt)

Beispiel:

gegeben:

```
int a = 48;
int b;
```

Expression	Wert	Seiteneffekt
`a = 84`	`84`	Wert 84 zu a zuweisen
`b = (a = 20)`	`20`	Wert 20 zu a und b zuweisen
`new Bicycle()`	(Referenz auf Bicycle Objekt)	Kreire und initialisiere eine neue Instanz (ein neues Objekt) der Klasse `Bicycle` im Speicher
`new double[20]`	(Referenz auf Array von `double`)	Kreire und initialisiere ein neues Array von 20 `double` Variablen im Speicher
`a++`	`48`	Wert 49 zu a zuweisen
`b = a++`	48	Werte 48 an b und 49 an a zuweisen
`++a`	`49`	Wert 49 zu a zuweisen
`b = ++a`	49	Werte 49 an b und 49 an a zuweisen