Script generated by TTT

Title: groh: profile1 (03.06.2015)

Date: Wed Jun 03 17:52:08 CEST 2015

Duration: 103:21 min

Pages: 157

Klassen und Objekte in Java

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

42

Klassen und Objekte in Java

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

42

Klassen und Objekte in Java

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

Attribute (Zustand)

Methoden (Verhalten)

Klassen-Definition

41

Klassen und Objekte in Java

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

Attribute (Zustand)

Methoden (Verhalten)

Klassen-Definition

Klassen und Objekte in Java

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```


Klassen und Objekte in Java

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

42

Klassen und Objekte in Java

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

42

Interfaces

Idee: Spezifiziere nur *welche* Methoden eine Klasse haben muss, die das Interface implementieren will (Analogie: einzuhaltender Vertrag)

```
interface IBicycle {
       void changeCadence(int newValue);

       void changeGear(int newValue);

       void speedUp(int increment);

       void applyBrakes(int decrement);
}
```

```
class Bicycle implements IBicycle {
   // remainder of this class implemented as before
   // except that above methods must be public
}
```


Interfaces

Idee: Spezifiziere nur *welche* Methoden eine Klasse haben muss, die das Interface implementieren will (Analogie: einzuhaltender Vertrag)

```
interface IBicycle {
       void changeCadence(int newValue);

       void changeGear(int newValue);

       void speedUp(int increment);

       void applyBrakes(int decrement);
}
```

```
class Bicycle implements IBicycle {
   // remainder of this class implemented as before
   // except that above methods must be public
}
```


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 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);

51

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);

51

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);

51

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);

51

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);

51

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);

51

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);

51

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);

51

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);

51

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);

51

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);

51

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

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

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

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

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

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

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

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}

53

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}

53

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}

53

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;
```

$= -345545.34534 * 10^{-12}$

$= 3245343455.555 * 10^{67}$

53

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;
```

$= -345545.34534 * 10^{-12}$

$= 3245343455.555 * 10^{67}$

53

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;
```

$= -345545.34534 * 10^{-12}$

$= 3245343455.555 * 10^{67}$

53

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ (byte, short, int, long) $\in \mathbb{Q}$ „$\approx \mathbb{R}$“ (float, double)

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]	[+/- $\approx 1.4*10^{-45}$, +/- $\approx 3.4*10^{38}$]	[+/- $\approx 4.9*10^{-324}$, +/- $\approx 1.8\ 10^{308}$]

```
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

$\in \mathbb{Z}$ (byte, short, int, long) $\in \mathbb{Q}$ „$\approx \mathbb{R}$“ (float, double)

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]	[+/- $\approx 1.4*10^{-45}$, +/- $\approx 3.4*10^{38}$]	[+/- $\approx 4.9*10^{-324}$, +/- $\approx 1.8\ 10^{308}$]

```
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;
```

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

54

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ (byte, short, int, long) $\in \mathbb{Q}$ „$\approx \mathbb{R}$“ (float, double)

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]	[+/- $\approx 1.4*10^{-45}$, +/- $\approx 3.4*10^{38}$]	[+/- $\approx 4.9*10^{-324}$, +/- $\approx 1.8\ 10^{308}$]

```
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;
```

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

54

Primitive Typen: Numerische Typen

$\in \mathbb{Z}$ (byte, short, int, long) $\in \mathbb{Q}$ „$\approx \mathbb{R}$“ (float, double)

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]	[+/- $\approx 1.4*10^{-45}$, +/- $\approx 3.4*10^{38}$]	[+/- $\approx 4.9*10^{-324}$, +/- $\approx 1.8\ 10^{308}$]

```
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;
```

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

54

Darstellung / Approximation von reellen Zahlen

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

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

- 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:

$(2^{24})^{-24} = 2$

```
double e = Math.pow(2.0d,24.0d); //1.6777216E7
double f = Math.pow(e,-24.0d);   //4.043174611952195E-174
```

- → 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;
```

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

- 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:

$(2^{24})^{-24} = 2$

```
double e = Math.pow(2.0d,24.0d); //1.6777216E7
double f = Math.pow(e,-24.0d);   //4.043174611952195E-174
```

- → 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)

55

Darstellung / Approximation von reellen Zahlen

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

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

- 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:

$(2^{24})^{-24} = 2$

```
double e = Math.pow(2.0d,24.0d); //1.6777216E7
double f = Math.pow(e,-24.0d);   //4.043174611952195E-174
```

- → 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)

55

Darstellung / Approximation von reellen Zahlen

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

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

- 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:

$(2^{24})^{-24} = 2$

```
double e = Math.pow(2.0d,24.0d); //1.6777216E7
double f = Math.pow(e,-24.0d);   //4.043174611952195E-174
```

- → 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)

55

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:

$(2^{24})^{-24} = 2$

```
double e = Math.pow(2.0d,24.0d); //1.6777216E7
double f = Math.pow(e,-24.0d);   //4.043174611952195E-174
```

- → 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:

$(2^{24})^{-24} = 2$

```
double e = Math.pow(2.0d,24.0d); //1.6777216E7
double f = Math.pow(e,-24.0d);   //4.043174611952195E-174
```

- → 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 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,樶, 韄,… 왝,왯,왠…,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, 韄,… 왝,왯,왠…,ششش, …,→,↓,■, …}

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

boolean isCool = true;
```

`\n` means "new line"

57

Primitive Typen: Boolean und Char

boolean	char
1 bit	16 bit
{ true, false }	{ … !, „, §, $, %, &, …., a, b, c, …., ∭,…,∰,…,≎, ガ,キ,ギ,…,㋗,㋘,㋙,…, 樵,樶, 韄,… 왝,왯,왠…,ششش, …,→,↓,■, …}

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

boolean isCool = true;
```

`\n` means "new line"

57

Primitive Typen: Boolean und Char

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

```
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();
```

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
…	…	…

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


Referenztypen

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

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

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
…	…	…

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


Referenztypen

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

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

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
…	…	…

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

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
...	...	...

58

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
...	...	...

58

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
...	...	...

58

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
...	...	...

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;



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
...	...	...

59

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
...	...	...

60

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
...	...	...

60

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

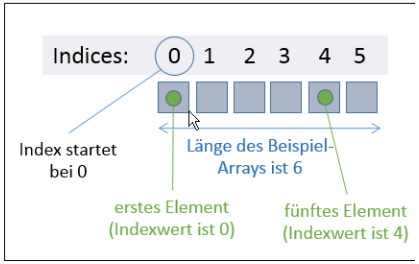
Zellnr (Adresse)	Zellname (Variablenname)	Zellinhalt
...	...	...
1149	bike1	<1405>
1150	bike1.cadence	0
1151	bike1.speed	0
1152	bike1.gear	3
...	...	...
1327	bike2	null
...	...	...
1405	bike2.cadence	0
1406	bike2.speed	0
1407	bike2.gear	1
...	...	...

poof!

61

Arrays

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



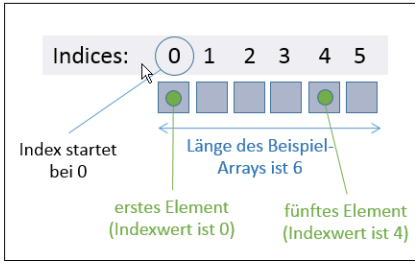

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

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();
```

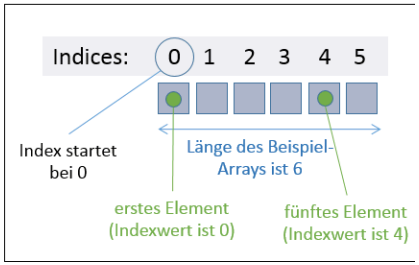
Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

62

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();
```

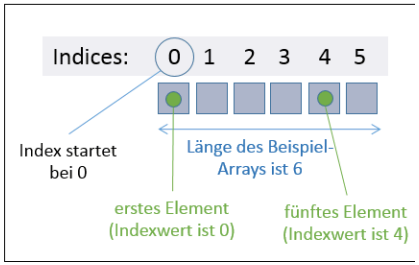
Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

62

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();
```

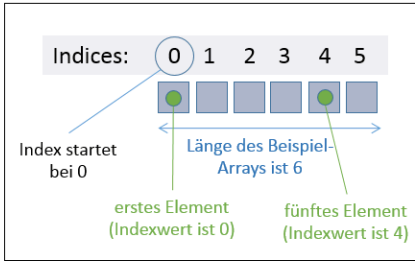
Array von Elementen *primitiven Typs*

Array von Elementen von *Referenztyp* (Objekte)

62

Arrays

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



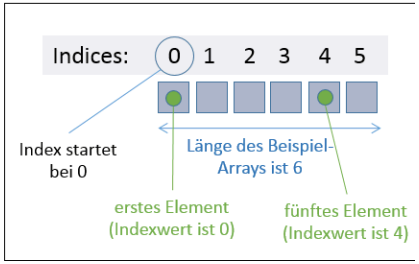

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

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



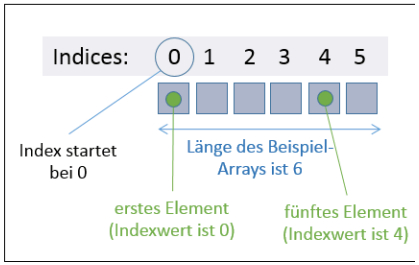

```
int[] someArray;
someArray = new int[6];
someArray[0] = 23;
someArray[1] = 12;
someArray[5] = 4 + someArray[2];

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)

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;
```

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
...	...	...
...	...	...

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
```

- length attribut

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

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
...	...	...
...	...	...

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	

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!)	

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 weiterer Operatoren:

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

logisch ‚impliziert', ‚Implikation' $a \rightarrow b$	b = true	b = false
a = true	true	false
a = false	true	true

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 weiterer Operatoren:

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

logisch ‚XOR', ‚Exklusiv-Oder' $a \oplus b$	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();`

Reference Type Comparison Operator

`instanceof` Ist ein Objekt von einem bestimmten (Referenz-)Typ? d.h. Instanz einer best. Klasse?

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bit-Manipulations-Operatoren

(nicht so interessant für uns. Siehe: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html)

Operatoren

Zuweisungs (Assignment) Operator

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

Reference Type Comparison Operator

`instanceof` Ist ein Objekt von einem bestimmten (Referenz-)Typ? d.h. Instanz einer best. Klasse?

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bit-Manipulations-Operatoren

(nicht so interessant für uns. Siehe: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html)

Operatoren

Zuweisungs (Assignment) Operator

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

Reference Type Comparison Operator

`instanceof` Ist ein Objekt von einem bestimmten (Referenz-)Typ? d.h. Instanz einer best. Klasse?

```
Vector z = new Vector();
boolean b =
    z instanceof Vector;
// b== true;
```

Bit-Manipulations-Operatoren

(nicht so interessant für uns. Siehe: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html)

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();
```

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

76

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

76

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

76

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
```


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

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`

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`

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

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
78

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
78

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
78

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

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

```
if (a != b) {                    // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                                // end block
```

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

```
if (a != b) {                  // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                              // end block
```


Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                  // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                              // end block

System.out.println(c);         // ERROR: c unavailable
```


Kontrollfluss-Statements

Kontrollfluss-Statements erlauben es, von der sequentiellen Abfolge der Abarbeitung der Statements abzuweichen.

- Bedingte Verzweigungen (conditionals): `if, if else, switch`
- Schleifen (loops): `while, do while, for`
- Verzweigungen (branches): `break, continue, return`

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                  // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                              // end block

System.out.println(c);         // ERROR: c unavailable
```

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                            // end block

System.out.println(c);       // ERROR: c unavailable
```


Kontrollfluss-Statements

Kontrollfluss-Statements erlauben es, von der sequentiellen Abfolge der Abarbeitung der Statements abzuweichen.

- Bedingte Verzweigungen (conditionals): `if, if else, switch`
- Schleifen (loops): `while, do while, for`
- Verzweigungen (branches): `break, continue, return`

Kontrollfluss-Statements

Kontrollfluss-Statements erlauben es, von der sequentiellen Abfolge der Abarbeitung der Statements abzuweichen.

- Bedingte Verzweigungen (conditionals): `if, if else, switch`
- Schleifen (loops): `while, do while, for`
- Verzweigungen (branches): `break, continue, return`

Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10)
        speed = speed - 2;
```

Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else {
        speed--;
}
```

EIN statement

Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else {
        if (speed > 0) {
                speed--;
        } else {
                System.err.println("The bicycle has already stopped!");
        }
}
```


Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else {
        if (speed > 0) {
                speed--;
        } else {
                System.err.println("The bicycle has already stopped!");
        }
}
```


Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else {
        if (speed > 0) {
                speed--;
        } else {
                System.err.println("The bicycle has already stopped!");
        }
}
```

Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else if (speed > 0) {
        speed--;
} else {
        System.err.println("The bicycle has already stopped!");
}
```


Schleifen: `while` `do while`

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

`do while`: Ähnlich zu `while`, aber überprüfe Bedingung NACH der Ausführung von etwas (anstelle VOR der Ausführung)

```
int count = 1;
do {
        System.out.print("#:" + count + " ");
        count++;
} while (count < 8)
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: `while` `do while`

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

`do while`: Ähnlich zu `while`, aber überprüfe Bedingung NACH der Ausführung von etwas (anstelle VOR der Ausführung)

```
int count = 1;
do {
        System.out.print("#:" + count + " ");
        count++;
} while (count < 8)
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: `while` `do while`

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

`do while`: Ähnlich zu `while`, aber überprüfe Bedingung NACH der Ausführung von etwas (anstelle VOR der Ausführung)

```
int count = 1;
do {
        System.out.print("#:" + count + " ");
        count++;
} while (count < 8)
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Allgemeine Form:

```
for (initialization; termination; update) {
      statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Allgemeine Form:

```
for (initialization; termination; update) {
      statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

allgemeines `for` ist äquivalent zu `while`:

```
initialization;
while (termination) {
      statements
      update;
}
```

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Verzweigungen: return break continue

- **return:** Beende die gerade ausgeführte Methode und gehe im Kontrollfluss an die Stelle zurück, wo sie aufgerufen wurde. (Mehr Details folgen gleich)
- **break:** Erzwinge die Beendigung einer Schleife
- **continue:** Überspringe die aktuelle Iteration einer Schleife

Man kommt fast immer ohne das aus (und sollte es auch!!)

```
for (int i = 0; i < 10; i++) {
      if (i == 8) {
            break;
      } else if (i % 2 == 0) {
            continue;
      }
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:1 #:3 #:5 #:7

Verzweigungen: return break continue

- **return:** Beende die gerade ausgeführte Methode und gehe im Kontrollfluss an die Stelle zurück, wo sie aufgerufen wurde. (Mehr Details folgen gleich)
- **break:** Erzwinge die Beendigung einer Schleife
- **continue:** Überspringe die aktuelle Iteration einer Schleife

Man kommt fast immer ohne das aus (und sollte es auch!!)

```
for (int i = 0; i < 10; i++) {
      if (i == 8) {
            break;
      } else if (i % 2 == 0) {
            continue;
      }
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:1 #:3 #:5 #:7

Verzweigungen: return break continue

- **return:** Beende die gerade ausgeführte Methode und gehe im Kontrollfluss an die Stelle zurück, wo sie aufgerufen wurde.
  (Mehr Details folgen gleich)

- **break:** Erzwinge die Beendigung einer Schleife
- **continue:** Überspringe die aktuelle Iteration einer Schleife

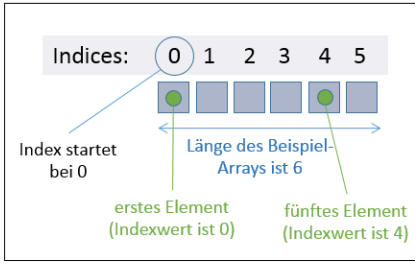
Man kommt fast immer ohne das aus (und sollte es auch!!)

```
for (int i = 0; i < 10; i++) {
      if (i == 8) {
            break;
      } else if (i % 2 == 0) {
            continue;
      }
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: #:1 #:3 #:5 #:7

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)