Primitive Variablen

Programmbeispiel:

```
class BspInt
{

  public static void main ( String[] args )
  {
     int zahl;      // 32 Bit großer Speicherbereich für ganze Zahl reserviert

     zahl = 18234; // Zuweisung: Abbildung eines bestimmten Bitmusters
                   // in dem reserviertem Speicherbereich

     System.out.println(zahl);
  }
}
```

Objekte

⇒ groß, komplex, variieren im Umfang

⇒ enthalten Daten und Methoden

```
class BspString
{
  public static void main ( String[] args )
  {
    String text;          // Deklaration eines Namens für ein zukünftiges Objekt

    text = new String(``Mein Text'');  // Initialisierung = Objekt erzeugt

    System.out.println(text);
  }
}
```

Zuweisungsoperatoren

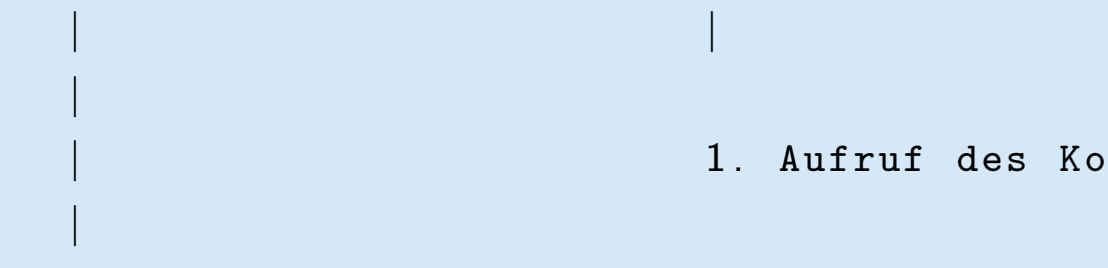
⇒ zwei Schritte:

1. Auswertung des Ausdrucks rechts von "="
2. Ergebnis der Auswertung wird der Variablen links von "=" zugewiesen

ABER:

```
String text;     // Name für ein zukünftiges Objekt

text =       new String( "Mein Text" );
```



1. Aufruf des Konstruktors erzeugt das Objekt.

2. Referenz, Speicherplatz, Weg, wie das Objekt zu finden ist, wird in der Variablen text gespeichert.

Zusammenfassung

Typ der Variablen	**Information, die sie enthält**	**Wenn sie sich links vom -"befindet**
primitive Variable	Enthält die aktuellen Daten.	Vorherige Daten werden durch die neuen Daten ersetzt.
Referenzvariable	Enthält Informationen über den Speicherbereich	Die alte Referenz wird durch die neue Referenz ersetzt.

Beispiel:

```
System.out.println(text);
```

Referenz in text wird verwendet, um

1. den Inhalt (Daten) der Variable zu finden
2. den Inhalt auszugeben

```
System.out.println(zahl);
```

Primitiver Datentyp:
Daten dieser Variable werden direkt verwendet.

Objektreferenzen

Anweisung	Aktion
str = new String(Text");	Das **ERSTE** Stringobjekt erzeugen. Eine Referenz auf das Objekt in str speichern.
System.out.println(str);	Daten des **ERSTEN** Objekt holen und ausgeben.
str = new String(neuer Text");	Ein **ZWEITES** Objekt erzeugen. Ab dieser Stelle gibt es keine Referenz mehr auf das **ERSTE** Objekt.
System.out.println(str);	Daten des **ZWEITEN** Objektes holen und ausgeben.

Hinweis: jeder **new**-Aufruf erzeugt ein neues Objekt !!!!!

Auf Gleichheit prüfen

```
String text = new String(``heute'');
String text2 = new String(``morgen'');
String text3 = text;
String text4 = new String(``morgen'');
```

⇒ (text **==** text2) → Ergebnis: false

- Prüft ob die Referenz (Speicherbereich) der beiden Objekte identisch ist
- (text == text3) →
- (text2 == text4) →

⇒ text.**equals**(text2) → Ergebnis: false

- Prüft ob die Inhalt der beiden Objekte identisch ist
- text2.equals(text4) →

Auf Gleichheit prüfen

```
String text = new String(``heute'');
String text2 = new String(``morgen'');
String text3 = text;
String text4 = new String(``morgen'');
```

⇒ (text **==** text2) → Ergebnis: false

- Prüft ob die Referenz (Speicherbereich) der beiden Objekte identisch ist
- (text == text3) → true (zwei Referenzen zeigen auf das gleiche Objekt)
- (text2 == text4) →

⇒ text.**equals**(text2) → Ergebnis: false

- Prüft ob die Inhalt der beiden Objekte identisch ist
- text2.equals(text4) →

Auf Gleichheit prüfen

```
String text = new String(``heute'');
String text2 = new String(``morgen'');
String text3 = text;
String text4 = new String(``morgen'');
```

⇒ (text **==** text2) → Ergebnis: false

- Prüft ob die Referenz (Speicherbereich) der beiden Objekte identisch ist
- (text == text3) → true (zwei Referenzen zeigen auf das gleiche Objekt)
- (text2 == text4) → false (verschiedene Objekte)

⇒ text.**equals**(text2) → Ergebnis: false

- Prüft ob die Inhalt der beiden Objekte identisch ist
- text2.equals(text4) →

Auf Gleichheit prüfen

```
String text = new String(``heute'');
String text2 = new String(``morgen'');
String text3 = text;
String text4 = new String(``morgen'');
```

⇒ (text **==** text2) → Ergebnis: false

- Prüft ob die Referenz (Speicherbereich) der beiden Objekte identisch ist
- (text == text3) → true (zwei Referenzen zeigen auf das gleiche Objekt)
- (text2 == text4) → false (verschiedene Objekte)

⇒ text.**equals**(text2) → Ergebnis: false

- Prüft ob die Inhalt der beiden Objekte identisch ist
- text2.equals(text4) → true (Inhalt ("morgen") gleich)

Übung 3.2

Ein Bruch kann als Paar positiver ganzer Zahlen $zähler$ und $nenner$ dargestellt werden. Entwerfen Sie eine Klasse Bruch, die eine Methode enthält, die den Bruch, so dass Zähler und Nenner teilerfremd sind (z.B. $48/256 = 3/16$ - 3 und 16 sind teilerfremd).

Eine ganze Zahl k ist ein gemeinsamer Teiler von Zähler und Nenner, falls gilt:

$$zähler \ \% \ k == 0$$
$$nenner \ \% \ k == 0$$

Wenn ein gemeinsamer Teiler gefunden worden ist, soll $zähler$ und $nenner$ durch $zähler \ / \ k$ und $nenner \ / \ k$ ersetzt werden.

Übung 3.2 (cont.)

⇒ Schreiben Sie einen $Konstruktor$ der einen Bruch initialisiert und die obere Methode aufruft.

⇒ Schreiben Sie Methoden, die die Grundrechenarten (Addition/Subtraktion/Multiplikation/Division) realisieren.

⇒ Schreiben Sie eine Methode $equals()$, wobei $a/b = c/d$ gilt, wenn $a * d = b * c$

⇒ Schreiben Sie eine Methode $print()$, die den Bruch ausgibt.