1. Aufgabe

Definieren Sie jeweils eine Variable der Typen **long: l1**, **int: i1**, **short: s1**, **double: d1**, **float: f1** und **char: c1**. Die Werte der Variablen l1, i1, s1, f1, d1 und c1 sollen eingelesen werden. Bilden Sie folgende Ausdrücke:

A1: **double** ergebnis = l1 + i1 + s1 + f1 + d1 + c1;
A2: String sErgebnis1 = "" + l1 + i1 + s1 + f1 + d1 + c1;
A3: String sErgebnis2 = l1 + i1 + s1 + f1 + d1 + c1 + "";
A4: String sErgebnis3 = l1 + i1 + s1 + "" + f1 + d1 + c1;
A5: **int** iErgebnis = (**int**)l1 + i1 + s1 + (**int**)f1 + (**int**)d1 + c1;
A6: **float** fErgebnis = l1 + i1 + s1 + f1 + (**float**)d1 + (**int**)c1;

Testen Sie mit verschiedenen Eingaben. Geben Sie die jeweiligen Ergebnisse aus. Geben Sie dabei auch jeweils den kompletten Ausdruck mit den jeweiligen Werten aus, mit dem das Ergebnis zustande kam, damit Sie die Rechnung und das Ergebnis nachvollziehen können. Versuchen Sie die Ergebnisse zu verstehen. Was passiert, wenn Sie die sog. *Casting*-Operatoren weglassen?

Hinweise:

- Zeichen vom Typ **char** werden in zusammengesetzten arithmetischen Ausdrücken automatisch in den Typ **int** konvertiert (entsprechend ihrem UNI-Code).
- Zeichenketten sind mit doppelten Anführungszeichen umrahmt. Die Konkatenation von Zeichenketten erfolgt durch den +-Operator. Ist wenigstens einer der beiden Operanden in a + b ein String, wird der gesamte Ausdruck als String-Konkatenation ausgeführt.

2. Aufgabe

Ausdrücke in Java, die aus mehr als nur einem elementaren Datentyp zusammengesetzt sind, werden je nach Verwendung durch implizite oder explizite Typkonversion ausgewertet. Gegeben sind nun folgende Variablendeklarationen:

```
long l1 = 5L, l2 = 123L;
int i1 = 9, i2 = 5877;
short s1 = 6;
byte b1 = 99, b2 = 2;
float f1 = 2.0f;
double d1 = 0.222, d2 = 17.0;
char c1 = '!', c2 = 'a';
String str1 = "123";
```

a) Welchen Typ und welchen Wert haben die Ergebnisse der folgenden Ausdrücke? Begründen Sie Ihre Entscheidung kurz. Erst nachdenken, dann probieren.
   Beispiel i1 + i2: das Ergebnis von ist vom Typ **int**, mit Wert 5886, da i1 und i2 vom Typ **int** sind!
   (b1 * i1) / (f1 * 3.0f)
   "1 + 2 + 3 = " + (i1 - 3)
   d1 / f1 + i1
   c1 * c2
   i1 + str1 + l2
   s1 + b1
   l1 + f1 + l2

b) Begründen Sie, warum nachfolgender Ausdruck einen Kompilierfehler ergibt. Wie könnte man ihn beheben?

```
byte b3 = b1 + b2;
```

3. Aufgabe

Entwickeln Sie eine Java- Anwendung, in der Sie eine Variable auf drei verschiedenen Wegen - sprich mit drei unterschiedlichen Operatoren - um den Wert 1 erhöhen. Geben Sie nach jeder Erhöhung um 1 den Wert der Variablen zur Kontrolle aus.

4. Aufgabe

Entwickeln Sie eine Java-Anwendung, die ein beliebiges Zeichen (Character) einliest und die drei folgenden Zeichen ausgibt (Beispiele: Eingabe von A erzeugt die Ausgabe B, C und D; Eingabe von '3' erzeugt die Ausgabe '4', '5' und '6').

5. Aufgabe

Entwickeln Sie eine Java- Anwendung, die eine Ganzzahl einliest. Gehen Sie davon aus, dass diese Zahl vierstellig ist. Machen Sie mit dieser Zahl folgendes:

- Zerlegen Sie die Zahl in einzelne Ziffern und geben Sie diese aus.
- Berechnen Sie die Quersumme der Ziffern und geben diese aus – so soll zum Beispiel für die Zahl 1234 die Quersumme 10 errechnet werden.
- Drehen Sie die Zahl um und geben diese aus. So wird zum Beispiel aus der Zahl 1234 die Zahl 4321.

6. Aufgabe

Lesen Sie die unteren Abschnitte mit den Titeln „**Vergleichsoperatoren“, „Logische Operatoren“** und **„Präzedenz-Tabelle und Prioritäten“.** Erstellen Sie zu den jeweiligen Beispielen (blassblaue Schattierung) jeweils ein lauffähiges Programm, welches die jeweiligen Werte der Variablen **„r“** bzw. **„ii“** ausgibt. Lösen Sie dann (mit Ihrem neuen Wissen) die weiteren Aufgaben. Erstellen Sie die Anwendungen so, dass Testwerte über die Konsole eingegeben werden können. Für das Eingabe-/Ausgabeprinzip siehe Aufgabe 6.

7. Aufgabe

Entwickeln Sie eine Java- Anwendung und definieren Sie drei Variable vom Typ boolean namens *Licht*, *Strom* und *Schalter*. Erstellen Sie eine derartige Verknüpfung, dass die Variable *Licht* nur dann auf true gesetzt ist, wenn die Variable *Strom* und die Variable *Schalter* auf true gesetzt ist. Geben Sie den Wert der Variablen *Licht* bzw. das Ergebnis der Verknüpfung in geeigneter Weise aus.

Eingabe-/Ausgabeprinzip (Beispiel):

```
Strom> true                 //Start Eingabe
Schalter> false
das Licht ist aus           //Ausgabe
```

8. Aufgabe

Entwickeln Sie eine Java- Anwendung und definieren Sie drei Variablen vom Typ boolean namens *Fahren*, *Zündschlüssel* und *Handbremse*. Erstellen Sie eine derartige Verknüpfung, dass die Variable *Fahren* nur dann auf true gesetzt ist, wenn die Variable *Zündschlüssel* auf true und die Variable *Handbremse* auf false gesetzt ist. Geben Sie den Wert der Variablen *Fahren* bzw. das Ergebnis der Verknüpfung in geeigneter Weise aus.
Eingabe-/Ausgabeprinzip (Beispiel):

```
Zündschlüssel> true                    //Eingabe
Handbremse> true                       //Eingabe

das Auto fährt nicht                   //Ausgabe
```

9. Aufgabe

Entwickeln Sie eine Java- Anwendung und definieren Sie zwei Variablen vom Typ boolean namens *Raketenstart* und *Freigabe* und eine Variable vom Typ int namens *Countdown*. Erstellen Sie eine derartige Verknüpfung, dass die Variable *Raketenstart* nur dann auf true gesetzt ist, wenn die Variable *Freigabe* auf true und die Variable *Countdown* auf 0 gesetzt ist. Geben Sie den Wert der Variablen *Raketenstart* bzw. das Ergebnis der Verknüpfung in geeigneter Weise aus.
Eingabe-/Ausgabeprinzip (Beispiel):

```
Freigabe> true                         //Eingabe
Countdown> 0                           //Eingabe

die Rakete startet!                    //Ausgabe
```

10.Aufgabe

Entwickeln Sie eine Java- Anwendung und definieren Sie vier Variablen vom Typ **boolean** namens *Satt*, *Vorspeise*, *Steak* und *Nachspeise*. Definieren Sie außerdem zwei Variablen vom Typ **int** namens *Getränke* und *Kartoffeln*. Erstellen Sie eine derartige Verknüpfung, dass die Variable *Satt* nur dann auf **true** gesetzt ist, wenn die Variable *Vorspeise* auf **true**, die Variable *Getränke* größer oder gleich 3, die Variable *Kartoffeln* größer oder gleich 2 und die Variable *Steak* auf **true** gesetzt ist.
Statt der Variablen *Steak* kann auch die Variable *Nachspeise* auf **true** gesetzt sein, damit die Variable *Satt* **true** ergibt.
Geben Sie den Wert der Variablen *Satt* bzw. das Ergebnis der Verknüpfung in geeigneter Weise aus.
Eingabe-/Ausgabeprinzip (Beispiel):

```
Vorspeise> false                       //Eingabe
Steak> true                            //Eingabe
Nachspeise> true                       //Eingabe
Getränke> 4                            //Eingabe
Kartoffeln> 4                          //Eingabe

ist er/sie satt? --> false             //Ausgabe: Alternative 1
er/sie ist nicht satt                  //Ausgabe: Alternative 2
```

11. **Aufgabe** – Zum Selbsttest

Betrachten Sie folgendes Programmfragment:

```
int i = 2, j = 3, erg1, erg2;
erg1 = (i++ == j ? 7 : 8) % 3;
erg2 = (++i == j ? 7 : 8) % 2;

boolean la1, la2, la3;
i = 3;
char c = 'n';
la1 = 2 > 3 && 2 == 2 ^ 1 == 1;
la2 = (2 > 3 && 2 == 2) ^ (1 == 1);
la3 = !(i > 0 || c == 'j');
```

Welche Werte erhalten die Variablen `erg1` und `erg2` bzw. die Variablen `la1` bis `la3` ?
Versuchen Sie es erst durch Nachdenken.
Verifizieren Sie Ihre Ergebnisse dann via Programm(ausgaben).

12. **Aufgabe –** zum Selbsttest

Welche der folgenden Zeichenketten sind gemäß der Definition der Programmiersprache Java korrekte Ausdrücke (bitte **ja** oder **nein** ankreuzen)? Geben Sie für jeden korrekten Ausdruck seinen (Ergebnis-)Typ an, d.h. den Typ den eine Variable besitzen muss damit man ihr das Ergebnis des Ausdrucks (ohne Typumwandlung) zuweisen kann. Geben Sie alternativ an warum er nicht korrekt ist. Setzen Sie voraus, dass alle Variablen vom Typ **int** sind und bereits korrekt initialisiert wurden.

Erst nachdenken, dann verifizieren.

Ausdruck	**Korrekt?**		**Typ / Warum nicht korrekt?**
	ja	**nein**	
a++ * b			
a<<2			
((x+y) / z) / (a - b)			
((m - n) + (w-x-z) / (p % q)			
a&b			
a-(b) < -3			
a|b^c			
a++ + ++c			

Vergleichsoperatoren

Mit den Vergleichsoperatoren kann überprüft werden, ob ein Wert gleich, ungleich, größer, größer gleich, kleiner oder kleiner gleich einem anderen Wert ist. Für einen Vergleich werden logischerweise immer zwei Werte benötigt, so dass alle Vergleichsoperatoren binäre Operatoren sind. In Java werden die Vergleichsoperatoren folgendermaßen geschrieben:

==, !=, >, >=, < und <=. (Bei zwei Zeichen kein Zwischenraum)

Das Ergebnis eines Vergleiches ist entweder true oder false - also ein Wahrheitswert. Dies ist insofern nachvollziehbar, als dass ein Vergleich wahr ist oder eben nicht. Betrachten Sie nachfolgende Beispiele.

```
int i = 10, j = 5;
boolean r;

r = i == j;
r = i != j;
r = i > j;
r = i >= j;
r = i < j;
r = i <= j;
```

Es werden nacheinander die Variablen *i* und *j* auf Gleichheit, Ungleichheit, Größer, Größer-Gleich, Kleiner und Kleiner-Gleich überprüft. Das Ergebnis wird in einer Variablen *r* vom Typ boolean gespeichert, da dies der Datentyp zum Speichern von Wahrheitswerten in Java ist.

Wie bei den arithmetischen Operatoren verändern auch Vergleichsoperatoren die Operanden nie selber. Um das Ergebnis eines Vergleichs zu speichern muss also ebenfalls wieder mit dem Zuweisungsoperator = das Ergebnis, nämlich einer der beiden Wahrheitswerte true oder false, in einer geeigneten Variablen gespeichert werden.

Beachten Sie, dass = der Zuweisungsoperator ist, der Vergleichsoperator == jedoch aus zwei Gleichheitszeichen besteht. Vor allem Anfängern passiert es immer wieder, das zweite Gleichheitszeichen zu vergessen und anstatt einem Vergleich versehentlich eine Zuweisung zu programmieren.

Kommen Ihnen obigen Code-Zeilen ein wenig merkwürdig vor, weil das Vergleichen zweier Werte an sich nicht ganz sinnvoll zu sein scheint, so werden Sie im nächsten Kapitel **Kontrollstrukturen** kennen lernen, die fast ständig mit Vergleichsoperatoren arbeiten.

Logische Operatoren

So wie arithmetische Operatoren das Rechnen mit Zahlen ermöglichen können Wahrheitswerte mit logischen Operatoren verrechnet werden. Java bietet drei binäre und einen unären logischen Operator an, die wie folgt definiert sind.

Das logische UND && verknüpft zwei Wahrheitswerte in der Art, dass das Ergebnis der Verknüpfung dann true ist, wenn genau beide Operanden true sind. In allen anderen Fällen ist das Ergebnis der Verknüpfung false.

Das logische ODER || verknüpft zwei Wahrheitswerte in der Art, dass das Ergebnis der Verknüpfung dann true ist, wenn einer der beiden Operanden oder auch beide Operanden true sind. Das Ergebnis ist dann false, wenn also genau beide Operanden false sind.

Java bietet im Gegensatz zu C und C++ ein logisches EXKLUSIVES-ODER an. Das ^ verknüpft zwei Wahrheitswerte in der Art, dass das Ergebnis der Verknüpfung dann true ergibt, wenn genau einer der beiden Operanden true und der andere Operand false ist. Das Ergebnis ist dann false, wenn beide Operanden true oder beide Operanden false sind.

Das logische NICHT ! gibt als Ergebnis true zurück, wenn der Operand false ist, und gibt als Ergebnis false zurück, wenn der Operand true ist. Das logische NICHT ist der unäre Operator und erwartet nur einen Operanden, der hinter ! angegeben werden muss. Daraufhin wird einfach der Wahrheitswert des Operanden umgedreht und zurückgegeben.

```
boolean i = true, j = false, r;

r = i && j;
r = i || j;
r = i ^ j;
r = !i;
```

Nachdem logische Operatoren Wahrheitswerte verknüpfen, sind im obigen Beispiel die Variablen *i* und *j* vom Typ boolean. Das Ergebnis einer logischen Verknüpfung ist ebenfalls wieder ein Wahrheitswert, so dass auch die Variable *r*, die die Ergebnisse speichert, den Datentyp boolean besitzt.

Beachten Sie, dass auch logische Operatoren niemals den Wert eines Operanden ändern. Logische Operatoren geben als Ergebnis einen Wahrheitswert zurück. Soll das Ergebnis später weiterverwendet werden, dann muss es wieder in einer geeigneten Variablen gespeichert werden.

Präzedenz-Tabelle und Prioritäten

In jeder Programmiersprache wird die Ausführungsreihenfolge von Operatoren in einer Präzedenz-Tabelle geregelt. Betrachten Sie folgendes Beispiel.

```
int ii;
ii = 10 + 5 * 2;
```

Welches Ergebnis wird in der Variablen *ii* gespeichert 30 oder 20? Wird zuerst addiert oder multipliziert? Wird also zuerst der Operator + oder zuerst der Operator * ausgeführt?

So wie in der Mathematik durch die Regel Punkt-vor-Strich festgelegt ist, dass zuerst multipliziert und dann addiert wird, sind in Präzedenz-Tabellen für eine Programmiersprache Prioritäten für Operatoren definiert. In der Präzedenz-Tabelle für Java besitzt der Operator * natürlich auch eine höhere Priorität als der Operator +, so dass selbstverständlich die Punkt-vor-Strich-Regel auch in Java gilt.

Wenn Sie jedoch folgendes Beispiel betrachten wird es schon recht schwer vorherzusagen, welche Operatoren zuerst ausgeführt werden.

```
boolean r;

r = true != false || true;
```

Wird in der Variablen *r* als Ergebnis true oder false gespeichert? In diesem Fall hilft nur noch der Blick in die Präzedenz-Tabelle von Java. Dort werden alle Operatoren ihrer Priorität nach aufgelistet. Ein Blick in die Präzedenz-Tabelle zeigt, dass der Operator != eine höhere Priorität besitzt als der Operator ||. Demnach wird im obigen Beispiel in der Variablen *r* das Ergebnis true gespeichert.

Wenn Sie sich nicht sicher sind, welcher Operator eigentlich zuerst ausgeführt wird oder nicht, setzen Sie einfach Klammern. Mit Klammern können Sie jederzeit in eine vorgegebene Ausführungsreihenfolge eingreifen. So wird beispielsweise im folgenden Beispiel zuerst die Addition und dann die Multiplikation ausgeführt.

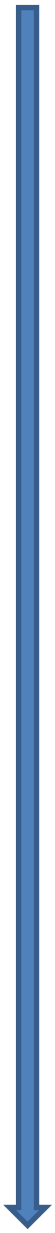
```
int ii;

ii = (10 + 5) * 2;
```

Wie in der Mathematik werden Klammern auch in Java von innen nach außen aufgelöst.

Setzen Sie Klammern, um die Ausführungsreihenfolge von Operatoren zu ändern oder zu bekräftigen. Kein Programmierer sucht nach seiner Präzedenz-Tabelle, um die Ausführungsreihenfolge von Operatoren zu überprüfen. Setzen Sie einfach Klammern, wenn Sie nicht sicher sind, und fertig.

Operatorpräzedenz – Übersicht und Rangfolge

Gruppen von Operatoren mit gleicher Priorität sind durch horizontale Linien begrenzt.



Operator	Anmerkung
.	Komponentenzugriff
[]	Array-Index
( )	Gruppieren von Ausdrücken
++ --	Prä- oder Postinkrement bzw. –dekrement
!	Negation
~	Bit-Komplement
-	Vorzeichenumkehr
new	Objekterzeugung
(Typ)	Typumwandlung (Casting)
* /	Punktrechnung
%	Modulo
+ -	Strichrechnung
+	Stringverkettung
<< >> >>>	Bitweiser Links- und Rechts-Shift
< > <= >=	Vergleichsoperatoren
instanceof	Typprüfung
== !=	Gleichheit, Ungleichheit
&	AND (bitweise; logisch mit unbedingter Auswertung)
^	XOR
|	OR (bitweise; logisch mit unbedingter Auswertung)
&&	Logisches AND (mit bedingter Auswertung)
||	Logisches OR (mit bedingter Auswertung)
? :	Kurzform für if...then...else (Konditionaloperator)
= += -= *= /= %= ^=	Verschiedene Zuweisungen
&= |= <<= >>= >>>=	Weitere Zuweisungen