**Aufgabe 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aufgabe** | **Lösung** | **Datentyp** | **Probleme?** | **Erklärung** |
| **A:** 3 + 4 \* 5 | 23 | Integer | Unter Umständen möchte man (aus welchen Gründen auch immer) die Operationen exakt in der Reihenfolge ausführen wie angegeben. Ohne Regeln. | In Java gilt Punkt vor Strich. Also wird zuerst 4\*5 gerechnet (20) und dazu die 3 Addiert. Endergebnis ist ein Integer, da nur Integer Zahlen verwendet wurden. |
| **B:** 3 + 4 / 3 | 4 | Integer | Selbige Problematik wie bei **A**. | Selbiges wie in **A**. |
| **C:** 3 + 4 % -3 | 4 | Integer | Selbige Problematik wie bei **A**. | Modulo (%) wird immer vor Strichrechnungen ausgeführt. Schreibt man das Modulo um erhält man: 4 – (4/-3)\*-3 was ausgerechnet 1 ergibt. Dies Addiert zu 3 ergibt 4. Wieder ist das Ergebnis ein Integer, da nur Integer verwendet wurden. |
| **D:** 3 / -4 \* 4 | 0 | Integer | Man könnte denken, dass Intern immer weiter mit Gleitkomma-zahlen gerechnet wird. | Java castet nach jeder Rechenoperation das Ergebnis direkt zum benötigten Datentyp. So wird zuerst 3 / -4 gerechnet was abgerundet 0 ergibt, und anschließend mit 4 Multipliziert. Gesamtergebnis ist 0, und wieder ein Integer da nur Integer verwendet wurde. |
| **E:** 3 + „true“ + false | „3truefalse“ | String |  | Hier werden zunächst alle Bestandteile der Konkatenation implizit in Strings umgewandelt, und anschließend mit einander von links nach rechts miteinander verknüpft. Das Ergebnis ist logischerweise wieder ein String. |
| **F:** 3 \* 4 < 5 == 5 >= 3 \* 2 | true | Boolean | Man könnte sich mit der Prioritäten vertun. | Hierfür muss man wissen wie die Prioritäten in Java aufgebaut sind. Zuerst wird multipliziert/dividiert, anschließend addiert/subtrahiert, nun mit Größer und kleiner verglichen und als letztes auf Gleichheit geprüft. Rechnet man das alles miteinander aus erhält man false == false was wiederum true ergibt. Denn beide Booleans besitzen den gleichen Wert. Logischerweise bekommen wir hier dann auch einen Boolean heraus. |
| **G:** „text“ + 3 + 4 | „text34“ | String |  | Selbiges wie in **E**. |
| **H:** 3 + 4 + „text“ | „7text“ | String | Man könnte vermuten, dass es hier wie in **E** und **G** abläuft. | Hier findet vor der Konkatenation eine Addition statt. Man sollte wissen, dass Java von links nach rechts alle Operationen durchgeht, und erst sobald ein Integer/Boolean mit einem String verknüpft sind die Konkatenation eingeleitet wird. Im Klartext heißt dies, dass Java zuerst 3 +4 Addiert und anschließend die 7 mit „text“ konkateniert. |
| **I:** 3 > 4 + „text | **Fehler!** | **Fehler!** | Allerdings | Schaut man sich nochmals die Prioritäten aus **F** an so merkt man, dass hier vor dem Vergleich von 3 > 4 zuerst die 4 mit dem Text konkateniert wird. Das heist, dass hier ein Integer mit einem Text verglichen wird (3 > „4text“ ). Wie nun ein Vergleich eines Integers in mit einem String funktionieren soll ist in Java nicht definiert. Deshalb resultiert hieraus ein Fehler. |
| **J:** (3 + 4) \* 5 + „text“ | „35text“ | String |  | Ähnlich wie bei **H**. Erst wird die Rechenoperation am Anfang durchgeführt ((3 + 4) \* 5) und das Ergebnis mit „text“ konkateniert. |
| **K:** (3 + 4 != 5 \* 6 ) + „“ != „text“ | true | Boolean |  | Die Große Klammer braucht man im Grunde nicht beachten. Denn das Ergebnis hieraus wird in jedem Fall mit dem String „“ konkateniert. Dieser neue String wird nun auf nicht-gleichheit mit „text“ untersucht. Nun sollte man wissen, dass bei einem Vergleich 2er Strings mithilfe von „==“ oder „!=“ **immer** die Speicheradresse verglichen wird! Daher kann bei dieser Ausführung nur true herauskommen. Denn beide Strings können nicht die gleiche Adresse haben. |
| **L:** true > 5 | **Fehler!** | **Fehler!** | Aufjedenfall! | Ähnlich wie **I**. Der Vergleich von Boolean mit Integer ist nicht definiert. In anderen Sprachen ginge das, jedoch ist Java hier sehr strikt. |
| **M:** false < (4 != 5) | **Fehler!** | **Fehler!** | Oh, ja! | Ähnlich wie **I** und **l**. Der Größer/Kleiner Vergleich 2er Booleans ist ebenfalls nicht definiert. Ist auch schwierig 2 Wahrheitswerte auf ihre Größe zu untersuchen. Jedenfalls ist Java hierbei wieder ähnlich wie bei **L** sehr strikt, sodass nur auf Gleichheit geprüft werden kann. |