1. **C** (==), weil identische Literale auf dieselbe Referenz im Heap verweisen. Ansonsten haben primitive Daten ohne ihre Wrapper keine Methoden, mit denen man Vergleiche anstellen könnte. Dass sie nicht verglichen werden können, haben wir empirisch bereits widerlegt durch die Anwendung in unseren Arbeiten.

2. **C**: Datenkapselung hat das Ziel Variablen, Methoden und Klassen vor unerwünschten zugriffen von extern zu schützen. Also Kontrolle über Daten.

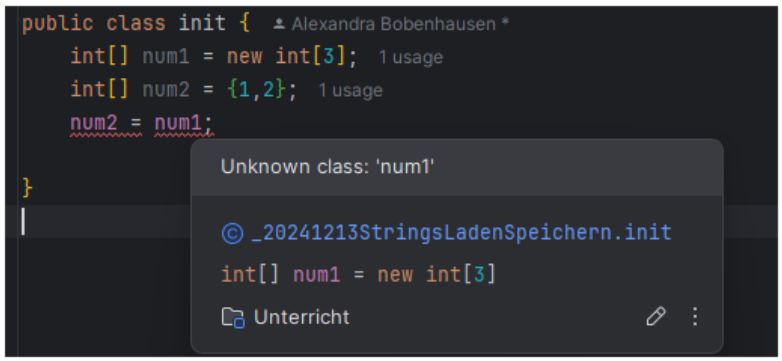
3. **D**, weil nur dort berücksichtigt wird, dass Additionen in Klammern bei einer Konkatenation wieder arithmetisch stattfinden und nicht aneinandergesetzt.

4. **C**, weil das Auftreten einer Exception (hier allgemein in find()) bereits in find() selbst abgefangen und mit einem print out gehandelt wird. Das Try and Catch im Main macht so gar keinen Sinn. Die ArithmeticException kann nicht mehr eintreten, weil in find() keine geworfen wird.

5. **A**, denn B und C gehen schonmal nicht, weil D ist package-private, B ist explizit private und damit können Klassen in sports und player einander nicht sehen. Eine static class wäre mir nicht bekannt. A stellt sicher, dass die packages einander sehen.
6. **A**, weil alle anderen Aussagen true ergeben, aufgrund dessen, dass A negiert ist, denn alle Aussagen in allen Klammern sind wahr und die Negation kehrt sie um.

7. **C**, weil es für jede Plattform eine eigene JDK gibt (!A). Nicht B, weil die JVM teil der JDK ist, nicht umgekehrt und nicht C, weil es nicht explizit Exceptions handelt.

8. **D**, weil Arrays nicht einfach ineinander kopiert bzw. zugewiesen werden können. Außerdem ist num1 zu groß für num2. Ansonsten gäbe es dazu Methoden.
   Der Fragebogen sagt es sei A (3:3), jedoch erlaubt mir die IDE das Zuweisen von num2 auf num1 NICHT. Wäre das jedoch syntaktisch korrekt, so würde das kleinere Array num2 die Länge des Arrays num1 erhalten und dann wäre A richtig.



Kommentiert [DL1]: Ich hatte zuerst A, aber jede Plattform bekommt ihr eigenes JDK.

Kommentiert [DL2]: Geht doch, ich habe die main vergessen.
Es wird aber kein Klon, sondern nur eine Referenz zu num1 hergestellt. Man kann num2 ab dann nicht mehr unabhängig von num1 verändern.

9. **D**, weil beim Aufruf von obj.doWork() alle einer auf 2 erhöht werden. Also 2:2:2. Nun ist es aber so, dass var2 statisch ist und somit keinem Objekt, sondern der Klasse gehört. Wird obj2.work() nun aufgerufen, sind var1 und var3 wieder 1, weil sie jeweils dem Objekt gehören, wohingegen var2 bereits 2 ist, bzw. immer noch 2 vom letzten Aufruf und der Wert in der Klasse verbleibt; nach doWork() also 2:3:2.

10. **C** (3), weil string1 2 Objekte kreiert: sich selbst (wegen new) und das Literal dazu, welches im StringPool des Heaps abgelegt wird, wohingegen string1 im allg. Heap liegt. string2 erzeugt kein neues Objekt, wer es der Referenz von string1 zugewiesen wird. Und string3 erzeugt einen neuen Wert, den es vorher nicht gab.

11. **B**, weil in der main ein default-Konstruktor aufgerufen wird, welcher in Class A überladen und nicht explizit neu implementiert wurde.

12. **A** (100/300), weil wenn obj.print(100) aufgerufen wird, wird dies direkt nach line n1 gedruckt. Dann wird var1(100) zu var2(200, statisch), aufaddiert. Anschließend, wird die statische Methode print() ausgeführt, die sich in der Signatur von der in line n1 unterscheidet und somit valide überladen ist. Diese druckt var2, welche durch die Addition in der objektbezogenen print(int) 300 ist.

13. **C**, weil eine Skriptsprache ausschließlich interpretiert wird, nicht kompiliert, wie auch Java (wenn auch mit Zwischencode, Bytecode). Strukturiert ist sie auch nicht (Spaghetticode goto xxx), sondern objektorientiert. Low-Level ist sie auch nicht, dazu müsste sie auf einer tieferen Abstraktionsebene sein wie Assembly.

14. **C**, weil capsicum an Index 1 gesetzt wird, wo eben noch cabbage stand, dabei rückt cabbage einen weiter auf, damit capsicum seinen Platz einnehmen kann. Bei einem Einschub mit Index, rutschen also alle Einträge einen auf. Carrot bleibt an Stelle 0, weil der Index 1 die zweite, nicht die erste Stelle (Index 0) adressiert.

15. D, weil auf msg erstmal nur die void-Methode sayHello() ausgeführt wird, also „Hello!“. Da nur hier das „!“ auftaucht, wäre das schon Grund genug. Da aber auch keine Error, keine Null und kein Komma in der Void Methode auftaucht und bei D wie erwartet das Queen auftaucht, ist es richtig.