Wiederholungsaufgaben

## Aufgabe 1

**Fülle die Tabelle mit den vorgegebenen Datentypen und ergänze den Rest.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| int | String | double | boolean | Array | short |
| char | float | byte | long | Car (class) | ArrayList |

Hilfe: <https://stackoverflow.com/questions/5389200/what-is-a-java-strings-default-initial-value>

|  |  |  |
| --- | --- | --- |
| **Primitiver Datentyp** | **Wrapper-Klasse** | **Standardwert** |
| int | Integer | 0 |
| char | Character | '\u0000' |
| float | Float | 0.0 |
| double | Double | 0.0 |
| byte | Byte | 0 |
| boolean | Boolean | false |
| long | Long | 0 |
| short | Short | 0 |
| **Referenzdatentyp** | **Standardwert** | |
| String | null | |
| Array | null | |
| Car | null | |
| ArrayList | null | |

**Was ist der Unterschied zwischen primitiven Datentypen und Referenzdatentypen?**

Primitive Datentypen speichern ihre „eigenen“ Werte (literale) also Zahlen, Zeichen oder Wahrheitswert, wohingegen Referenzdatentypen (Objektvariablen) die Referenz zum Objekt (numerisch) abspeichern oder auf den Wert null verweisen, wenn keine Referenz hergestellt wurde.

**Wofür brauchen wir die Wrapper-Klassen?**

Wrapperklassen dienen dazu primitiven Daten(typen) die Funktionalität und Kompatibilität von Objekten zu verleihen. Man kann sie als Objekte übergeben, wo notwendig und Methoden auf diese Anwenden, um eine veränderte Fassung zu erhalten oder andere Informationen aus Ihnen abzuleiten. Man kann von oder zu einem String konvertieren.

**Wann muss man einen primitiven Datentyp initialisieren und wann wird der Standardwert benutzt?**

Instanzvariablen (einem Objekt gehörend) und statische Variablen (der Klasse gehörend) müssen nicht explizit initialisiert werden. Sie werden implizit mit ihren Standardwerten belegt, wenn nicht anders festgelegt (siehe Tabelle von eben).

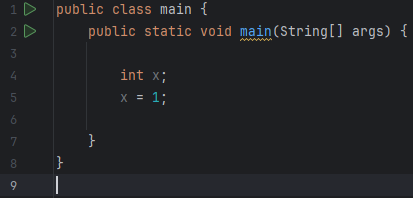
Lokale Variablen hingegen, also einer Methode oder Blockanweisung zugehörig, müssen konkret und explizit initialisiert werden.   
Beispiel Schleifenvariable: Eine in einer Schleife zu überprüfende oder zu verändernde Variable muss zuvor initialisiert (und somit auch deklariert) werden. So kann zum Beispiel nicht ein Boolean angelegt und implizit der Standardwert false angenommen werden.

## Aufgabe 2

In welcher **Zeile** sind folgende Variablen initialisiert?

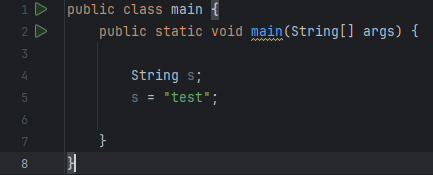
Schreibe auch eine **kurze Begründung** und erkläre die **Lebensdauer** der einzelnen Initialisierungen.

*Beispiel:*



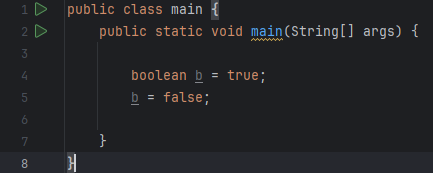
In Zeile:***5, da in Zeile 4 nur der Variablenname gespeichert wird und bei lokalen Variablen kein Standardwert hinterlegt wird. Die Variable x existiert nur in der Main-Methode. Vor und nach Ausführung dieser geht die Initialisierung und die Variable verloren.***

1.



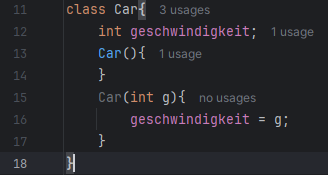
In Zeile: 5, da eine lokale Referenzvariable ohne Initialisierung nicht implizit auf null verweist. Nur innerhalb der main-Methode. (Siehe oben, Beispiel).

2.



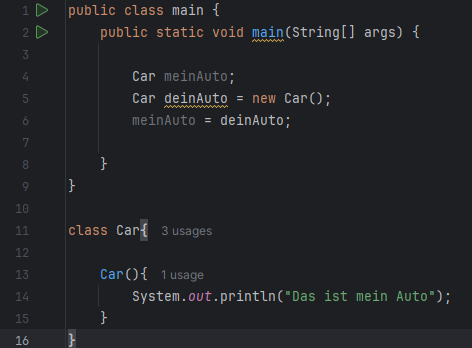
In Zeile: 4 mit true, (überschrieben dann in Zeile 5). Nur innerhalb der main-Methode.

3.



In Zeile: 13, da es eine Instanzvariable ist und der Standardkonstruktor verwendet wird.  
Wird bei Instanziierung initialisiert und nach Löschung des Objektes (nicht nur der Referenz) (durch den Garbage Collector?) wieder zerstört.

4.



meinAuto in Zeile: 6, erhält die Referenz von deinAuto.

deinAuto in Zeile: 5, erzeugt ein neues Objekt und erhält eine neue Referenz.

Jeweils lokale Variablen und nur innerhalb der main-Methode.

## Aufgabe 3

Schau in der letzten Teilaufgabe (4.) von Aufgabe 2 nach. Warum reicht Zeile 4 nicht aus, um ein Objekt der Klasse Car zu erzeugen? Warum müssen wir **= new Car()** schreiben?

Die Referenzvariable meinAuto ist dort keiner Referenz zugewiesen und auch nicht auf null, weil es eine lokale Variable ist. Sie ließe sich so nicht verwenden. Das *new Car()* ruft den Konstruktor auf, der das Objekt überhaupt erst erzeugt, wobei der Variable die Referenz, die zusammen mit dem Objekt entstanden ist, zugewiesen wird.

## Aufgabe 4

Schreibe ein Testprogramm mit welchem du die Funktion von **==, equals** und **compareTo** testen kannst. **Teste es bei allen in Aufgabe 1 genannten Datentypen**. Was sind die Unterschiede zwischen diesen Vergleichsoperatoren und wann sollte man welchen verwenden? (Strukturelle Vorgaben sehr gewöhnungsbedürftig oder täusche ich mich?)

Anmerkung: Ich bin bei der Aufgabenerfüllung davon ausgegangen, dass mit Nutzung der in Aufgabe1 genannten Datentypen die primitiven und NICHT ihre Wrapper gemeint waren. Dies hat Auswirkung auf die Ergebnisse der Vergleiche. (Es gibt eine Gegenprobenklasse.)

Bei welchen Datentypen funktioniert **==**?

Und schreibe eine kurze Erklärung, warum es nur bei ihnen funktioniert.

Funktioniert bei allen, weil == die Referenz oder Literale vergleicht. Eine Referenz wird als numerischer Wert wie auch primitive Daten gespeichert (theoretisch ist eine Referenz somit ein primitives Datum, nicht aber das dazugehörende Objekt). Aus diesem Grund können nicht nur Referenzen, sondern auch Literale verglichen werden. Primitive Datentypen enthalten ihre Daten selbst, während Objektvariablen eine numerisch codierte Referenz enthalten, wobei dann mit == die Zahlen dieser Referenzen verglichen werden. Bei String wird auch die Referenz des Literals im String Pool des Heaps berücksichtigt, sofern der String irgendwo hartcodiert wurde, andernfalls werden die wie normale Objekte behandelt.

Bei welchen Datentypen funktioniert **equals**?

Und schreibe eine kurze Erklärung, warum es nur bei ihnen funktioniert.

Funktioniert nicht mit primitiven Datentypen, sondern nur mit Objekten (und somit auch den Wrappern primitiver Datentypen). equals vergleicht den Inhalt eines Objekts (nicht die Referenz oder alleinstehende Literale). (Auto-Boxing mit Casting ergibt wohl Ungleichheit).

Bei welchen Datentypen funktioniert **compareTo**?

Und schreibe eine kurze Erklärung, warum es nur bei ihnen funktioniert.

Funktioniert in diesem Szenario nur mit String. Ansonsten ginge es auch mit den Wrapper-Klassen und allgemeiner gefasst mit allen Klassen, die das Interface Comparable beinhalten. Dabei wird eine numerische oder lexigrafische Verschiebung anhand einer eines Integers zurückgegeben. Bei – liegt es davor, bei 0 auf gleicher Stufe, bei + dahinter.   
Auch Boolean kann damit verglichen werden, aber das sollte man im Detail nachlesen.

## Aufgabe 5

Erweitere dein Testprogramm aus Aufgabe 4, sodass du auch ein **double** mit einem **int** vergleichen kannst. Nutze dazu **parsen** (also das Überführen in einen anderen Datentyp).

Meint ihr mit parsen das Konvertieren wie (double) integer-Wert oder tatsächlich die parse-Methoden der Wrapper (parseInt(Zahl))?

Anmerkung: Da hier von parsen die Rede ist, muss ich davon ausgehen, dass Wrapper gemeint sind. Wenn ich dies versuche, brauche ich für die parse-Methoden aber einen String und müsste einen Umweg gehen, der erst in der zweiten Erweiterung benötigt wird. Demnach nutze ich für die erste das Casting. Wenn dies nicht gewollt war, bitte nächstes Mal genauere Vorgaben machen. Dies ist ja eh ein konstruiertes Beispiel.

Zweite Erweiterung: einen **String** mit einem **int** vergleichen.

Hier konnte ich parsen. Oben war ohne String das Casten sinnvoller.

(War das wieder so eine Aufgabe alla: Wenn du was Besseres hast, mach stattdessen das?)

## Aufgabe 6

Erweitere dein Testprogramm, sodass es einen Fehler beim **Parsen** abfängt und das Programm neu gestartet wird mit anderen Werten. (Neustart mit anderen Werten?)  
**Starte beim String mit einem Wort und beim erneuten Start ersetzt das Wort mit einer Zahl, sodass es zu keinem Fehler kommt.**

Versuche das Programm **ohne Scanner** (also ohne Benutzereingaben während der Laufzeit) zu schreiben. (Also sollte bis hierhin mit Scanner oder wie?)

Ich gehe mal davon aus nicht das ganze Programm soll neustarten, sondern nur der Teil / die Methode wo der Parsing-Fehler auftritt. Wenn erst ein Wort und dann eine Zahl in den String soll, dann muss ich das wohl hardcoden und das kann ich nicht durch einen Neustart bei Fehler verändern. Sehr verwirrend geschrieben. Ich löse das mit einer Schleife.

## Aufgabe 7

An welche Exceptions müssen wir denken, wenn wir von einem Benutzer eine Eingabe erwarten?

Nenne 6 Beispiele. (Ihr müsst **nicht** den Namen einer Exception angeben, sondern was kann alles schief gehen?)

*z.B der Benutzer hat nichts eingegeben (leere Eingabe)*

1. Buchstaben eingegeben
2. Sonderzeichen eingegeben
3. falsches Trennzeichen bei Kommazahlen
4. Zahlenraum des Zieldatentyps überschritten (z.B. byte größer 128 oder kleiner -127)
5. Eingabe leer (wie oben genannt)
6. Außerhalb der erwarteten Parameter (z.B. 6 bei Menü von 1-5)

## Aufgabe 8

Schreibe ein kleines Satzbau Programm

Nimm zufällig aus 3 ArrayList’s je ein Satzstück und gebe dann den kompletten Satz aus.

Beispiel:

**Der Bauer** *springt* Seifenblasen.

**Teil 1: sollte ein Nomen sein**

*Teil 2: sollte ein Verb sein*

Teil 3: sollte wieder ein Nomen sein

Sodass dann ein halbwegs grammatikalisch richtiger Satz herauskommt.

Lasst eurer Kreativität freien Lauf und habt Spaß :)

*Schöne Feiertage und ein guter Rutsch ins neue Jahr!*

*Wünschen euch eure Dozentin und Tutoren*

*Alexandra, Dmytro, Luca und Severin*