# NKT
# Tag 1: Datentypen, Bedingungen und Arrays

Sven Hoops, Jonas Goudarzi

10.03.2025

# Gliederung

# Variablendeklaration

Heinrich Heine
Universität Düsseldorf

In Java benutzen wir häufig Variablen, um Daten zu speichern. Um eine Variable anzulegen benötigen wir einen Datentypen und einen Namen. Das Anlegen einer Variable heißt Deklaration:

```
1    int f;
2    float g;
3    String msg;
```

primitive Variablen, die nur deklariert wurden, kriegen einen Standardwert zugewiesen. Bei *int* ist dieser beispielsweise 0

# Variableninitialisierung

Heinrich Heine Universität Düsseldorf

Bei der Initialisierung von Variablen weisen wir diesen nun einen Wert zu:

```
1    int f = -2;
2    float g = 42.69;
3    String msg = "Hello World";
```

Referenzen gelten hier auch als Werte, die wir zuweisen können:

```
1    String importantMessage = msg + "!";
```

# Primitive Datentypen

Heinrich Heine
Universität Düsseldorf

Welche primitiven Datentypen gibt es?

- boolean
- byte
- char
- short
- int
- long
- float
- double

# Nicht primitive Datentypen

Heinrich Heine Universität Düsseldorf

- Objektreferenzierung
  - speichert anstelle des Objektes nur die Referenz auf dem Stack
  - erlaubt die Einschränkung von Zugriff auf Attribute und Variablen des Objektes
  - ist häufig nötig, da Stack Platz sehr begrenzt
- Beispiele
  - String
  - java.awt.Color
  - (Arrays)
  - eigene Klassen

# Die String Klasse

- ein String Objekt enthält eine Zeichenkette in festgelegter Reihenfolge
- die Klasse String verfügt über Methoden, die auf dem Objekt selber aufgerufen werden können

```
1    String msg = "dies␣ist␣ein␣String!";
2    String msg2 = msg.toUpperCase();
```

# Verzweigung

```
1    if(x){                    //kann aleine genutzt werden
2        //do X
3    } else if(y) {            //falls spezieller Fall y
4        //do Y
5    } else {                  //default Fall
6        //do Z
7    }
```

- falls viele *else if's* auftreten sollten wir eher ein *switch statement* verwenden

# Schleifen - While

```
1    //while Schleife
2    while(x){                    //solange die Bedingung x zutrifft
3        //do X
4    }
5    //do-while Schleife
6    do{
7    } while(x)
```

- Achtung vor Endlosschleifen!

# Schleifen - *for*

```
1  for(int i = 0; i < 5; i++){
2      //do X
3  }
```

- häufig benutzt, um Collections durchzugehen
- *for-each* Schleife:

```
1  for (int itVar : array){
2      //do X
3  }
```

- hierbei bitte aufpassen. Mit der *for-each* Schleife kann nicht der Inhalt des Arrays geändert werden. Dies hat Performance Gründe

# Logische Operatoren

Heinrich Heine Universität Düsseldorf

Logische Operatoren werden in Schleifenköpfen benötigt. Die Bedingung dafür, wie lange die Schleife läuft, ist also ein komplexer logischer Operator:

- ||
- &&
- ==
- < oder >
- <= oder >=
- |
- &

```
1    if(v + 42 <= 69){
2        // do X
3    }
```

# Idiomatisch

Heinrich Heine Universität Düsseldorf

For Schleife ⇔ While Schleife
Wenn wir ein Array mit einer *while* Schleife ausgeben wollen geht dies auch:

```
1    int count = 0;
2    while(count <= array.length){
3        System.out.println(array[count]);
4        count++;
5    }
```

Eine *for* Schleife ist hier jedoch deutlich sinnvoller

# Arrays

Was ist ein Array?

- speichert eine genaue Anzahl an Variablen desselben Datentypes
- hat eine feste Größe (nicht dynamisch erweiterbar)
- nicht initialiserte Felder kriegen einen Standardwert oder null zugewiesen
- indices für schnellen Zugriff
- wird von Java wie ein Objekt behandelt → liegt auf dem Heap

```
1    int[] myArray = new int[5];
2    int[] mySecondArray = {1,2,3,4,5};
```

# Main Method

Heinrich Heine Universität Düsseldorf

Die Main Method hat als Parameter ein *String* Array.[1] Der Inhalt des Arrays muss beim Aufruf des Programmes mit übergeben werden.

```
1  public static void main (String[] args){
2      String msg = args[0];
3  }
```

Arrays haben Indices. Von 0 bis n-1.[2] Wie können wir nun durch ein Array iterieren?

```
1  public static void main (String[] args){
2      for(int i = 0; i < args.length; i++){
3          System.out.println(args[i]);
4      }
5  }
```

[1] Achtung! Das args Array ist NICHT die Standardeingabe

[2] wenn n die Länge des Arrays ist

# Beispiel Aufgabe

Heinrich Heine
Universität Düsseldorf

Schreiben Sie ein Programm, welches einen Wert n entgegen nimmt und eine Treppe mit Sternen ausgibt. In der M-ten Iteration sollen genau M Sterne in der Zeile ausgegeben werden. Es soll n Iterationen geben. Für n = 4:

```
1   *
2   **
3   ***
4   ****
```

# Standardeingabe

Heinrich Heine Universität Düsseldorf

Wir können auch die Standardeingabe benutzen, um unser Programm mit Daten aufzurufen. Dafür benötigen wir Scanner:

```
1  import java.util.Scanner;
2  public class Wuerfel {
3     public static void main(String[] args) {
4         Scanner stdin = new Scanner(System.in);
5         while(stdin.hasNext()){
6             int number = Integer.parseInt(stdin.nextLine());
7             while(number > 0){
8                 for(int i = 0; i < number; i++){
9                     System.out.print("*");
10                }
11                number--;
12                System.out.println();
```

# Scanner & Dateien

Heinrich Heine Universität Düsseldorf

Wir können Scanner aber auch auf spezifische Dateien setzen. Dafür brauchen wir jedoch den Dateipfad.

```
1    Scanner scanner = new Scanner(dateipfad);
```

Achtung! Wenn an diesem Pfad keine Datei liegt, wird unser Programm abstürzen. Wir sollten deshalb einen *try-catch* Block benutzen:

```
1    try{
2        Scanner scanner = new Scanner("/NKT/wichtigeDatei.txt");
3    } catch(Exception e){
4        e.printStacktrace();
5    }
```

# Mini-Game

Heinrich Heine Universität Düsseldorf

Entwerfen Sie eine kleines Mini-Game. Sie sollen über die Standardeingabe mit dem Spiel interagieren können.

1. Ihr Spieler hat beispielsweise ein Inventar mit 3*9 Slots. Dieses wird ausgeben wenn Sie bspw *inventory* in die Konsole schreiben
2. Implementieren Sie, dass die Gegenstände in das Inventar schreiben können (*add*)
3. Implementieren sie eine Methode *find* <Item>, die ausgibt, an welchem Platz ein Item im Inventar liegt. (Achten Sie darauf, dass nur ein Index gefunden werden kann, wenn der Gegenstand auch wirklich im Inventar liegt)

# Mini Game

Heinrich Heine Universität Düsseldorf

Beispielaufrufe:

```
$ inventory
[Schwert][][][][][Helm][][][]
[][][Bogen][][][][][][Trank]
[][Holz][][][][][Relikt][][]
$ add Stein 2
Stein wurde an index 2 hinzugefuegt!
$ inventory
[Schwert][][Stein][][][Helm][][][]
[][][Bogen][][][][][][Trank]
[][Holz][][][][][Relikt][][]
$ find Helm
Ein Helm befindet sich an index 5!
```

# Beispielaufgabe

Heinrich Heine
Universität Düsseldorf

Für Schnelle: Werden Sie kreativ und überlegen Sie sich Erweiterungen. Ein paar Denkanstöße:

- der Spieler könnte ein Level haben
- Stats (HP, Stamina, etc)
- lasse den Spieler Heiltränke nutzen
- der Spieler kann erkunden gehen und gegen Gegner kämpfen

# Beispielaufrufe

```
$ status
Level: 1
HP: 100
Stamina: 50
$ explore
Du findest einen Heiltrank!
$ add Heiltrank 10
Heiltrank wurde an Slot 10 hinzugefuegt!
$ use Heiltrank
Du hast einen Heiltrank benutzt und 20 HP wiederhergestellt!
$ attack
Du greifst den Goblin an und verursachst 15 Schaden!
Der Goblin greift dich an und verursacht 10 Schaden!
```