Ein kleines Cheat Sheet mit den wichtigsten Klausurinhalten, ohne Anspruch auf Korrektheit und Vollständigkeit. Hat sich während der Klausur als sehr nützlich erwiesen :)

Inhalt

1. Standard

1.1. Konstruktor

```
public ClassName(type one, type two ){
   this.one = one;
   this.two = two;
}
```

1.2. Getter & Setter

```
public type getThing(){
return this.thing;
}
public void setThing(type thing){
this.thing = thing;
}
```

1.3. Exception erstellen und werfen

a) Neue Exception erstellen

```
package myPackage;

public class MyException extends Exception {
    public MyException(String message) {
         super(message);
    }
}
```

b) Exception werfen

```
public type myMethod() throws MyException{
   if( condition ){
         throw new MyException("my text");
   }
}
```

```
public type otherMethod() {
   try {
               myMethod();
         } catch (MyException e1) {
               JOptionPane.showMessageDialog(null,
         "Please enter a number!",
         "Error",
         JOptionPane.ERROR_MESSAGE);
   }
}
```

1.4. Schleifen

a) for

```
for(int i = start; x <= end; i++ ){
```

```
    // do something
}
```

b) while

```
while(condition){
    //do thing
}
```

c) while mit Zähler

```
int i = start;
while(i<end){
    //do thing
    i++;
}
```

d) Schleife zum ergänzen einer Menge

```
HashSet<Integer> numbers = new HashSet<>();
int i = 0;
while (i < amount) {       //amount of objects to be added
    int number = i + 69;
    if (numbers != null) {
          if (!numbers.contains(number)) {
                numbers.add(number);
                i++;
          }
    } else {
          numbers.add(number);
          i++;
    }
}
```

1.5. Try-Catch

```
try {
// Try doing this
}
catch(Exception e) {
//  Block of code to handle errors
} finally {
// always do this
}
```

1.6. Random

```
//gets random Int between min(inclusive) and max(exclusive)
ThreadLocalRandom.current().nextInt(min, max);
```

2. Enum

```
public enum Aufzählungstyp {
    FIRST("first", 1),
    SECOND("second", 2),
    THIRD("second", 3),
    FOURTH("second", 4);

    private String text;
    private int number;


    /**
     * Constructor
     * @param text
     * @param number
     */
    private Aufzählungstyp(String text, int number) {
        this.text = text;
        this.number = number;
    }


    /**
     * @return the text
     */
    public String getText() {
        return text;
    }


    /**
     * @param text the text to set
     */
    public void setText(String text) {
        this.text = text;
    }

    /**
     * @return the number
     */
    public int getNumber() {
        return number;
```

```
    }


    /**
     * @param number the number to set
     */
    public void setNumber(int number) {
        this.number = number;
    }

}
```

3. GUI

3.1. GUI Template

```
package coronaWarn;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.*;
import javax.swing.*;

public class TestGUI {
    private int one;
    private double two;
    private String three;

    /**
     * @param one
     * @param two
     * @param three
     */
    public TestGUI(int one, double two) {
        this.one = one;
        this.two = two;
        this.three = String.format("one: %d \t two: %.2f", one, two);

        // Create frame with Panels
        JFrame frame = new JFrame("Title");

        JPanel top = new JPanel();
        JPanel center = new JPanel();
        JPanel bottom = new JPanel();

        // Create Components
        JLabel label = new JLabel(three);

        JButton btn1 = new JButton("1");
```

```
        JButton btn2 = new JButton("2");
        JButton btn3 = new JButton("3");
        JButton btn4 = new JButton("4");
        JButton[] grid = { btn1, btn2, btn3, btn4 };

        // Edit the Frame
        frame.setLayout(new BorderLayout());

        /*
         * Options for Layout:
         *
         * frame.setLayout(new BorderLayout()); - platz zwischen komponenten
setzbar
         * frame.setLayout(new BoxLayout(frame, BoxLayout.PAGE_AXIS)); - axe
         * setzbar(x,y,seite etc.) frame.setLayout(new CardLayout()); - platz
zwischen
         * komponenten setzbar frame.setLayout(new FlowLayout()); - ausrichtung
und
         * platz zwischen komponenten setzbar frame.setLayout(new
GridBagLayout());
         * frame.setLayout(new GridLayout()); - zeilen, spalten und platz
zwischen
         * komponenten setzbar frame.setLayout(new GroupLayout(frame));
         * frame.setLayout(new SpringLayout());
         *
         */

        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setMinimumSize(new Dimension(400, 400));

        // Add and Edit the Panels
        frame.add(top, BorderLayout.NORTH);
        frame.add(center, BorderLayout.CENTER);
        frame.add(bottom, BorderLayout.SOUTH);

        top.setLayout(new BorderLayout());
        top.setForeground(new Color(0, 0, 255));
        center.setLayout(new GridLayout(2, 2));
        center.setPreferredSize(new Dimension(400, 300));
        bottom.setLayout(new FlowLayout());

        // Edit the Top Panel
        top.add(label);
        label.setHorizontalAlignment(SwingConstants.CENTER);
        // Edit the Center Panel - grid
        int i = 0;
        for (int y = 0; y < 2; y++) {
              for (int x = 0; x < 2; x++) {
                    center.add(grid[i]);
                    i++;
              }
        }
```

```
            // Edit the Bottom Panel
            bottom.add(new JLabel("Buttons: "));
            bottom.add(new JButton("do"));
            bottom.add(new JButton("don't"));

            // Action Listener

            ActionListener listen = new ActionListener() {
                  @Override
                  public void actionPerformed(ActionEvent e) {
                        if (e.getSource() == btn1) {
                              System.out.println("one was pressed");
                        } else if (e.getSource() == btn2) {
                              System.out.println("two was pressed");
                        } else if (e.getSource() == btn3) {
                              System.out.println("three was pressed");
                        } else if (e.getSource() == btn4) {
                              System.out.println("four was pressed");
                        }

                  }
            };

            btn1.addActionListener(listen);
            btn2.addActionListener(listen);
            btn3.addActionListener(listen);
            btn4.addActionListener(listen);

            Runnable player = new Runnable() {

                  @Override
                  public void run() {
                        try {
                              // do something
                              Thread.sleep(500); // 0,5 seconds
                              System.out.println("this is the other thread");
                        } catch (InterruptedException e) {
                        }
                  }
            };
            new Thread(player).start();

            frame.pack();
            frame.setVisible(true);

      }
      public static void main(String[] args) {
            new TestGUI(1, 1.2345);
      }

}
```

3.2. Dialogfenster anzeigen

```
// default title and icon
JOptionPane.showMessageDialog(f1,
"Eggs aren't supposed to be green.");
// custom title, warning icon
JOptionPane.showMessageDialog(f1,
"Eggs aren't supposed to be green.",
"Warning",
JOptionPane.WARNING_MESSAGE);
// custom title, error icon
JOptionPane.showMessageDialog(f1,
"Eggs aren't supposed to be green.",
"Error",
JOptionPane.ERROR_MESSAGE);
// custom title, no icon
JOptionPane.showMessageDialog(f1,
"Eggs aren't supposed to be green.",
"A plain message",
JOptionPane.PLAIN_MESSAGE);
```

3.3. Dynamisch Raster mit Elementen erstellen

```
// Beispiel: JButtons erstellen, speichern und hinzufügen
// Array zum speichern der Buttons, die Länge ist die breite * höhe vom
Grid Panel

grid.setLayout(new GridLayout(height, width));
JButton[] buttons = new JButton[width * height];

int i = 0;
	for (int y = 0; y < height; y++) {
		for (int x = 0; x < width; x++) {
			JButton current = new JButton("text");
			 buttons[i] = current;
			 grid.add(current);
			 i++;
		}
	}
```

4. Read & Write

4.1. Datei erstellen und speichern

```
public void saveFile() {
	String filename = “name.txt”
```

```
        File file = new File(filename);
        try {
              if (!file.exists()) {
                    file.createNewFile();
              }
              FileWriter writer = new FileWriter(filename, false);
              for (int i = 0; i<5; i++) {
                    String line = “inhalt einer hübschen zeile”;
                    writer.write(line);
              }
              writer.close();

              if (file.exists() && file.length() > 0) {
                    JOptionPane.showMessageDialog(null, filename + " was
saved successfully.", "Success",
                                JOptionPane.INFORMATION_MESSAGE);
              } else {
                    JOptionPane.showMessageDialog(null, filename + "
could not be saved. Please try again.", "Error",
                                JOptionPane.ERROR_MESSAGE);
              }

        } catch (IOException ex) {
              ex.printStackTrace();
        }
    }
```

4.2. Datei einlesen

```
package sinem_klausur.chorona;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Reader {
    private String filepath;

    /**
     * @param filepath
     */
    public Reader(String filepath) {
```

```
        this.filepath = filepath;
    }

    public List<String> getLinesWithBufferedReader(String filepath)
throws IOException {

 //ab hier kopieren
       BufferedReader reader;
       List<String> lines = new ArrayList<>();
       String line;
       try {
             reader = new BufferedReader(new FileReader(filepath));
             line = reader.readLine();
             while (line != null) {
                   System.out.println(line);
                   lines.add(line);
                   // read next line
                   line = reader.readLine();
             }
             reader.close();
       } catch (IOException e) {
             e.printStackTrace();
       }
//bis hier kopieren

       return lines;
    }

    public List<String> getLinesWithScanner(String filepath) throws
IOException {

// ab hier kopieren
       List<String> lines = new ArrayList<>();
       Scanner scanner = new Scanner(new File(filepath));
       while (scanner.hasNextLine()) {
             String line = scanner.nextLine();
             lines.add(line);
             // System.out.println(line);
       }
//bis hier kopieren

       return lines;
    }

}
```

5. Anmerkungen

5.1. String format specifier

Format Specifier	Data Type	Output
%a	floating point (except *BigDecimal*)	Returns Hex output of floating point number.
%b	Any type	"true" if non-null, "false" if null
%c	character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer
%e	floating point	decimal number in scientific notation
%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%h	any type	Hex String of value from hashCode() method.
%n	none	Platform-specific line separator.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.
%x	integer (incl. byte, short, int, long, bigint)	Hex string.

5.2. String builder

```
/ Java code to illustrate StringBuilder

import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class GFG1 {
    public static void main(String[] argv)
        throws Exception
    {

        // create a StringBuilder object
        // using StringBuilder() constructor
        StringBuilder str
            = new StringBuilder();

        str.append("GFG");

        // print string
```

```
        System.out.println("String = "
                           + str.toString());

        // create a StringBuilder object
        // using StringBuilder(CharSequence) constructor
        StringBuilder str1
            = new StringBuilder("AAAABBBCCCC");

        // print string
        System.out.println("String1 = "
                           + str1.toString());

        // create a StringBuilder object
        // using StringBuilder(capacity) constructor
        StringBuilder str2
            = new StringBuilder(10);

        // print string
        System.out.println("String2 capacity = "
                           + str2.capacity());

        // create a StringBuilder object
        // using StringBuilder(String) constructor
        StringBuilder str3
            = new StringBuilder(str1.toString());

        // print string
        System.out.println("String3 = "
                           + str3.toString());
    }
}
```

Ein StringBuffer als veränderbarer String

Java-Strings sind unveränderbar. Bei der Neubelegung einer String-Variablen wird somit jedes Mal ein neues String-Objekt erzeugt - bei häufigen Änderungen eine überflüssige Speicherbelastung.
Die Klasse *java.lang.StringBuffer* stellt eine Character-Sequenz bereit, die beliebig verändert und schließlich in einen String gewandelt werden kann.

Ein *StingBuffer*-Objekt kann durch vier überladene Konstruktoren erzeugt werden

- `StringBuffer()`
- `StringBuffer(CharacterSequenz cs)`
- `StringBuffer(int capacity)`
- `StringBuffer(String s)`

stellt aber selbst kein String-Objekt dar, sondern muss - wenn gewünscht - durch die Methode *toString()* in einen solchen gewandelt werden.
Die wesentlichen Routinen zur Manipulation eines *StringBuffer*-Objektes sind die vielfach überladenen Methoden `insert()` und `append()`, mit deren Hilfe jeder beliebige Wert entweder an einem als Parameter übergebenen Index eingefügt oder ans Ende angehängt werden kann.

```
public class StrBuff {

    public static void main(String[] args) {
        StringBuffer buff = new StringBuffer("sch\u00E4fer");
        System.out.println(buff);
        buff.insert(3, 'l');
        System.out.println(buff);
        buff.insert(0, 7);
        System.out.println(buff);
        buff.append("h\u00F6hle");
        System.out.println(buff);
    }
}
```

Das Beispiel demonstriert dies durch das Einfügen des Characters '`l`' an Position 3 und der String-Repräsentation des int-Wertes 7 am Anfang des Objektes.
Darüber hinaus kann erkannt werden, dass und wie in Java Character-Typen auch durch Unicode-Werte dargestellt werden können.