Handout zum Java Kurs

Als Ergänzung zu den Folien

Von

Sönke Schmidt

Inhaltsverzeichnis

1 Einführung

1.1 IDE und Voraussetzungen

1.1.1 DrJava

Wie ihr auf den Folien schon gesehen habt, verwenden wir erstmal DrJava. Das findet ihr unter: http://www.drjava.org/

DrJava ist für Windows quasi eine portable App, man muss also nichts installieren und es kann direkt ausgeführt werden, ihr braucht es nur runterladen. Ihr könnt auch die .jar Datei herunterladen, diese Funktioniert auf jedem System (sofern das JRE installiert ist, dazu gleich mehr). Diese ist direkt in Java kompiliert. Ja genau, richtig erkannt, DrJava wurde selbst mit Java geschrieben und kann nun mit Java ausgeführt werden.

1.1.2 Voraussetzungen

Damit man unter Java ein Programm kompilieren kann, muss – wie schon in der Präsentation erwähnt – ein JDK (Java Development Kit) installiert sein. Ist kein JDK installiert, kann man ein Programm nicht kompilieren und somit auch nicht ausführbar machen. Auf den Rechnern hier an der Uni ist das JDK schon installiert, falls ihr es zu Hause auch noch braucht findet ihr es hier:
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
(Oder googelt einfach nach „JDK“)

Damit das Programm ausgeführt werden kann benötigt man die JRE (Java Runtime Environment). Das ist die Java Laufzeitumgebung und kann das Programm auf den Rechner ausführen. Es ist also die virtuelle Java Maschine. Auf den Uni Rechnern hier ist es installiert, generell ist es auf vielen Rechnern bereits installiert. Falls es euch zu Hause fehlt, findet ihr es unter: https://java.com/de/download/
(Oder googelt einfach nach „JRE“)

1.1.3 Geany

Nachher verwenden wir auch noch Geany (https://geany.org/), aber dazu später mehr. Auch wieso wir später noch eine andere IDE verwenden, wird dann erklärt. Ihr braucht es jetzt noch nicht downloaden, denn ich werde euch später eine vorkonfigurierte portable Version mit den richtigen Einstellungen geben.

1.1.4 Weitere IDE's

Wie auf den Folien schon gezeigt, gibt es noch viele andere IDE's. Was einem gefällt ist eigentlich Geschmackssache, einige sind besser und schlechter. Manche sind komplizierter als andere oder haben spezielle Features die andere nicht haben. Ich möchte euch nicht auf eine IDE beschränken, ihr könnt zu Hause gerne noch weitere Testen und sehen, welche euch gefällt. Für den Anfang denke ich aber sind die beiden bereits erwähnten IDE's (DrJava und Geany) schon sehr gut, da sie nicht überladen und somit nicht kompliziert sind.

Wenn du gerne noch weitere IDE's sehen möchtest kannst du zu Hause danach googeln oder dir zu Hause die folgenden mal genauer ansehen:

IntelliJ IDEA (https://www.jetbrains.com/idea/)
Falls du dich für IntelliJ entscheidest kannst du die kostenlose Community Version nehmen. Studenten können intelliJ übrigens kostenlos bei jetbrain bekommen.

Eclipse (https://www.eclipse.org/downloads/)

Beide IDE’s sind für größere Projekte gut geeignet und empfehlenswert. Für kleinere Projekte geht wie gesagt aber locker DrJava oder Geany.

2 Das erste Programm

Wie ein Hello-World Programm aussieht und man etwas auf die Konsole ausgibt, habt ihr bereits gesehen. Auch wie man Funktionen aufruft und weitere wichtige Sachen, habt ihr bereits kennengelernt. Es wird also nun Zeit, dass wir unser eigenes Programm schreiben und uns mit Java mehr vertraut machen.

Hast du DrJava schon heruntergeladen und es ausgeführt? Wenn nicht, ist das der erste Schritt. Benutz du die .jar Version von DrJava? Wenn es nicht richtig starten will, könnt ihr es auch von der Konsole aus starten. Dafür öffnest du einfach eine Eingabeaufforderung, am schnellsten geht das mit „Win + r“, tippst dort „cmd“ ein und bestätigst mit Enter. Nun muss man zu dem Ordner navigieren, wo die .jar Datei liegt, das geht mit dem Befehl „cd“. Bei mir wäre es zum Beispiel dieser Pfad:

```
cd C:\PortableApps\DrJava\neu
```

Nun kann man mit Java diese .jar Datei ausführen, das geht folgendermaßen:

```
java -jar drjava-beta-20160913-225446.jar
```

Jetzt sollte DrJava starten. Vielleicht fragt euch beim Starten, die Windows Firewall noch um Erlaubnis, wenn es geht, dann könnt ihr das ruhig erlauben. Und wenn es nicht geht, ist es auch kein Problem 😉

DrJava ist nun hoffentlich gestartet?! Wenn nicht sag mir bitte Bescheid und wir lösen das zusammen.
Ansonsten könnt ihr jetzt eine neue Java Datei erstellen und sie als „Test.java“ speichern. Den nachfolgenden Code kannst du gleich abtippen. Noch als Hinweis, speichere regelmäßig ab und spiel auch gerne mit dem Code rum. Drücke zwischendurch mal „Compile“ um es zu kompilieren und um zu schauen, ob Fehler im Code sind. Drücke auch ruhig mal auf „Run“, um den Quellcode auszuführen.

Du siehst „Compile“ oder „Run“ oben in der Leiste gar nicht? Dann vergrößere am besten mal das Fenster von DrJava und ziehe es links oder rechts etwas zur Seite. Alternativ findest du unter „Tools“ ganz oben in der Leiste auch im nun aufpoppenden Dropdown Menü das compile und run.

Gebt nun den folgenden Code ein:

```
import java.util.Scanner;
              //Importanweisung für den Scanner, damit wir diesen benutzen können
              //Wir brauchen den Scanner zum einlesen von der Konsole erst später

//Der Name unserer Klasse. So heißt dann auch diese Java Datei (also "Test.java")
public class Test
{

       //Unser Haupteinstiegspunkt. Bei main beginnt das Programm immer:
       public static void main (String[] args)
       {

              String name;   //String Variable "name" Initialisieren
              //Methode "VonKonsoleTextEinlesen" aufrufen,
              //dabei übergeben wir einen Text,
              //das Ergebnis wird in die Variable "name" gespeichert.
```

```
            name = VonKonsoleTextEinlesen("Bitte geben Sie ihren Namen ein: ");
            //Nun geben wir das Ergebnis, was in "name" steht auf die Konsole aus:
            System.out.println("Hallo " + name + " :) ");
            //Wir machen einen Extra Umbruch und setzen den Namen in
            //Anführungsstriche, wir müssen dafür das " Escapen:
            //Anschließend machen wir nochmal einen Extra umbruch mit \n machen
            System.out.println("\n" + "Ist \""+name+ "\" wirklich korrekt? \n");

            //Diesmal erzeugen wir gleich die Variable und speichern in
            //ihr das Ergebnis aus der Methode:
            String alter = VonKonsoleTextEinlesen("Wie alt bist du?");
            System.out.println("So so, du bist also " + alter + " Jahre alt.");

            System.out.println( VonKonsoleTextEinlesen("Was ist dein ↵
            Lieblingseis?") + " ist also dein Lieblingseis!");

      }

      /* Diese Methode heißt VonKonsoleTextEinlesen und bekommt einen
       * String übergeben der in der Variable "Text" gespeichert ist.
       * Wir geben außerdem einen String zurück (mit return).
       * Das wir einen String zurückgeben geben erkennen wir an
       * "String" das vor VonKonsoleTextEinlesen steht.
       * Diese Funktion gibt auf der Konsole den übergebenen Text aus
       * und liest dann von der Konsole
       * alles ein, was der Benutzer eingibt, bis er ein Enter drückt.
       * Diese Eingabe wird dann von der Methode zurückgegeben.
       */
      public static String VonKonsoleTextEinlesen(String Text)
      {
            //Den übergebenen Text auf die Konsole ausgeben
            System.out.println(Text);
            // neues Scanner-Objekt "sc"
            Scanner sc = new Scanner(System.in);
            //Von der Konsole einlesen und in dem String "input" speichern
            String input = sc.nextLine();

            return input; //Die Eingabe wird mit return zurückgegeben
      }


}   //Ende von: public class Test
```

Achtung:

Bei einem `System.out.println` ist der Text auf die nächste Zeile umgebrochen und geht dort einfach weiter. Das der Text umgebrochen wurde, aber eigentlich in derselben Zeile steht, erkennst du an dem Zeichen „↵“.
Im Editor von DrJava solltet ihr das hintereinanderschreiben, sonst gibt es Fehlermeldungen! Will man auf mehreren Zeilen weiterschreiben, müsste man den String unterbrechen und mit einem Plus am Ende konkatenieren. Wie das genau aussieht mache ich in dem Code im Kapitel „Operatoren und Exceptions“

Spiel ein bisschen mit dem Code rum und versteh ihn. Speicher den Quellcode ab und drücke auf „Compile“ um zu kompilieren. Klappt alles? Wenn es nicht richtig kompiliert und es Fehlermeldungen ausgibt, hast du irgendwo einen Fehler drin. Vermutlich hast du dann einen Tippfehler gemacht oder etwas übersehen. Kontrolliere am besten nochmal deinen Code mit dem Quellcode, der hier steht. Wenn alles erfolgreich kompiliert, kannst du nun auf

„Run“ drücken, um es auszuführen.
Hast du verstanden, wie man die Funktion jeweils aufgerufen hat und was dort in der main-Methode passiert?

3 Operatoren und Exceptions

Wir erweitern nun unsere Klasse und fügen in die Klasse (also innerhalb von dem Bereich der geschweiften Klammern der Klasse) noch weitere folgende Methoden ein:
(Am besten unter/nach der Methode „VonKonsoleTextEinlesen“ die folgenden Methoden einfügen)

```
/* Diese Methode wandelt einen String in eine Ganzezahl (int) um.
 * Bekommt einen String.
 * Gibt einen int zurück.
 */
public static int StrToInt(String ZahlAlsString)
{
	try  //Versuche das folgende, wenn es nicht geht, springe zu catch
	{
		return ( Integer.parseInt(ZahlAlsString) );
	}
	catch(Exception e) //Umwandlung nicht möglich? Dann mache folgendes:
	{
		System.out.println("Es ist ein Problem aufgetreten! "+
				"Der String scheint keine natürliche Zahl zu"+
				"sein und konnte nicht umgewandelt werden! \n"+
				"Die Fehlermeldung lautet: "+e);
	 System.out.println("Programm wird nun wegen Fehlern zwangsbeendet!");

		//Die 1 gibt an, dass das Programm nicht erfolgreich beendet
		//wurde. Möchte man das Programm normal und ohne Probleme
		//beenden bzw dem System anzeigen, dass die Terminierung so
		//gewollt war, kann man den Exitcode 0 angeben.

		System.exit(1);

		//Java erwartet Tatsächlich ein return für diesen Zweig.
		//Ansonsten kompiliert es nicht. Jedoch werden wir
		//dieses return nie ausführen können, da wir vorher
		//schon mit exit das Programm terminiert haben.
		return 0;
	}

}



/* Diese Methode wandelt einen int in einen String um.
 * Bekommt eine Ganzezahl (als int).
 * Gibt einen String zurück.
 */
public static String IntToStr(int ZahlAlsInt)
{
	return ( Integer.toString(ZahlAlsInt) );
}
```

In die main-Methode schreiben wir nun hinter

```
System.out.println( VonKonsoleTextEinlesen("Was ist dein Lieblingseis?") + " ist
also dein Lieblingseis!");
```

folgendes:

```
//Den String von alter in einen int umwandeln, damit wir
//damit rechnen können.
int alter_int = StrToInt(alter);
//Jetzt können wir das Geburtsjahr berechnen
int Geburtsjahr = 2019 - alter_int;
//Wir geben die Variable Geburtsjahr aus, welches vom
//Typen int ist, diesen müssen wir bei einer Ausgabe auf
//die Konsole nicht nochmal explizit zu einem String
//umwandeln, das macht Java automatisch:
System.out.println("Dein Geburtsjahr muesste also "
                                + Geburtsjahr + " sein.");
```

Denkt dran, dass die main-Methode auch eine geschlossene geschweifte Klammer hat bzw. braucht!

Wir können nun also mit verschiedenen Sachen rechnen und mit dem Benutzer interagieren, indem wir die Berechnungen wieder auf der Konsole ausgeben lassen.

4 Zweiter Teil vom Script - Kontrollstrukturen

Jetzt hatten wir gerade auf den Folien die Kontrollstrukturen durchgenommen. Dann lasst uns gleich mal einige Sachen dazu programmieren und ausprobieren. Gebt dazu den folgenden Code in die Main Methode ein, am besten direkt hinter

```
System.out.println("Dein Geburtsjahr muesste also "
                                + Geburtsjahr + " sein.");
```

diesen Code.

Vergleiche gehen besonders gut mit Zahlen, darum verwenden wir für das Überprüfen eine Ganzezahl (int) und nicht den String. Wir haben zwar auch gelernt, wie man Strings miteinander vergleichen kann, aber das nützt uns hier nichts. Würden wir zwei Strings miteinander vergleichen, können wir nur sagen, ob der String, welcher ja nur eine Zeichenkette ist, miteinander identisch ist. Da wir aber vergleichen wollen, ob das alter kleiner als 6 ist, müssen alle anderen Werte darunter (also 0, 1, 2, 3, 4, 5) abfragbar sein. Als int kann man simple sagen, dass alles kleiner als 6 unseren gewünschten Fall ausführt. Dahingegen kann man bei String nur auf Gleichheit prüfen.

```
if ( alter_int < 6) //Wenn das alter kleiner als 6 Jahre ist, dann:
{
        System.out.println("Kannst du ueberhaupt lesen oder ⏎
        schreiben?");
}


System.out.println("Lass uns deine Geburtstage zaehlen: ");
//Wir gehen in der Schleife von 1 bis zum angegebenen alter:
for ( int i= 1 ; i <= alter_int; i++) // i++  entspricht i = i +1
{
        System.out.println( i );
}
System.out.println("WOOOW So viele Geburtstage hast du schon ⏎
gefeiert!!!");
```

Beachte hier bitte wieder den Hinweis für „⏎“, wie bei Achtung beschrieben.

Probiert damit herum und versteht, was ihr gemacht habt.

Ihr seht ich habe die Umlaute (ä,ü,ö) gemieden und dafür lieber die alternative geschrieben, so wird ä zu ae, ü zu ue und ö zu oe bei mir. Aber warum habe ich das gemacht?
In der internen Konsole von DrJava werden die Umlaute richtig dargestellt und angezeigt. Lässt man nun aber den Code in der Eingabeaufforderung (also der Windows Konsole) ausführen, kann es bei älteren Windows Versionen (Win 7 und älter) dazu kommen, dass die Zeichen nicht korrekt dargestellt werden, weil die Eingabeaufforderung teilweise noch mit einer alten Textcodierung arbeitet und Umlaute nicht erkennt. In diesem Fall werden die Umlaute sehr seltsam dargestellt. Bei mir persönlich klappt es aber ab Win 10 ohne Probleme. Probiert es also einfach mal aus und wenn seltsame Textanzeige Fehler kommen, behaltet die Umlaute oder andere Sonderzeichen, wie z.B. „ß“, als Problemverursacher im Hinterkopf.

Lass uns die boolesche Logik, die wir an der Tafel gemacht haben, gleich mal Testen. Wir gehen davon aus, dass Kinder unter 6 Jahren noch nicht lesen können und wir gehen davon aus, dass Leute über 95 nicht mehr richtig lesen können bzw. Probleme mit dem lesen haben, weil sie zu alt sind. Wie würdest du es machen? Überlege erstmal selber, bevor du umblätterst. Verwende dafür die bereits vorhandene „if“ Kontrollstruktur.

Hast du es ausprobiert und getestet? Funktioniert es?

Die Aufgabe war etwas kniffelig und fies, denn im Text habe ich ein „und“ verwendet. Wenn man es aber als Programm umsetzen möchte, muss man es folgendermaßen Aussprechen: Wenn das alter unter 6 Jahren ist oder das alter über 95 Jahre ist, soll ein Text ausgegeben werden. Wir müssen also das „logische oder“ verwenden.
Sobald eines dieser beiden Aussagen erfüllt wurde (entweder die erste Aussage mit „kleiner als 6“ oder die andere Aussage „größer als 95“) wird die Gleichung als Wahr ausgewertet und führt das was im „if-Fall“ steht aus.

Wir haben also das if folgendermaßen geändert:

```
if ( ( alter_int < 6) || (alter_int > 95) )
```

Das `||` steht dabei für das „oder“, siehe dafür nochmal in die Folien.

5 Simpler Mailclient

Ganz genau, wir wollen einen Mailclienten bauen. Ist das nicht zu schwer oder zu krass für mich, wirst du dich vielleicht nun fragen. Aber ich kann dich beruhigen, wir werden einige fertige Komponenten bzw. eine sogenannte Library verwenden und dadurch wird es recht easy. Wir können uns also auf den spaßigen Teil konzentrieren und müssen uns nicht mit den Protokollen oder der Übertragung rumschlagen.

5.1 Disclaimer und Vorsicht

Wie hieß es schon bei Spiderman: „Aus großer Kraft folgt große Verantwortung“. Nehmt euch das bitte auch zu Herzen, denn mit dem Wissen das du hier erhälst, könnte man ziemlich viel Blödsinn anstellen. Man kann aber auch echt viel Cooles und Gutes damit machen. Ich möchte es aber nochmal explizit ansprechen: Bitte sieh davon ab Spam-Mails oder Massen-Emails zu versenden, gefälschte Email Absender zu benutzen oder sonstiges schlechtes damit zu machen. Hinzukommt, dass du zwar viel mehr Macht hast, als in einem gewöhnlichen Emailclienten oder einer Weboberfläche, aber alle Aktionen noch vom Emailserver (GoogleMail, Hotmail, Web.de, etc) ausgeführt werden müssen. Da du dich dort aber verifizieren, genauer gesagt also Anmelden musst, können die Serverbetreiber auch genau deine Aktionen zu Dir zurück nachverfolgen. Das alles auf dich zurückverfolgbar ist, liegt auch daran, dass nicht nur der Username gespeichert wird, sondern auch die IP und die Uhrzeit sowie Datum bei den Servern geloggt werden. Die Server haben meist auch Automatismen eingebaut, die solche Vorgänge automatisch erkennen können, so dass sie dann den Benutzer daraufhin zur Rechenschafft ziehen können.

5.2 Vorbereitungen

Bevor wir loslegen können, müssen wir noch einiges machen. Wir brauchen einmal die Library der JavaMail, das ist einfach die Bibliothek, die schon alles wichtige implementiert hat. Dann müssen wir den Classpath beim kompilieren und beim Ausführen anpassen. Der Classpath ist der Pfad, der zu den Libraries (sozusagen den Bibliothekeinträgen) führt. Zusätzlich müssen wir die Serverdaten bei unserem Mail Anbieter herausfinden und ggf. noch weitere Einstellungen vornehmen. Wenn das alles erledigt ist, können wir anfangen unseren Code zu schreiben.

Wichtig: Ich habe für euch die meisten Sachen schon eingestellt und werde euch eine passende IDE geben, damit ihr direkt loslegen könnt. Diese Infos sind also für euch als Zusatz gedacht und damit ihr es auch zu Hause weiter machen könnt. Wir werden ab nun Geany als unsere IDE verwenden. Hier ist alles bereits von mir eingestellt und vorkonfiguriert.

Die von mir bereits vorkonfigurierte Geany Portable App findest du auf:
https://www.magentacloud.de/share/tqa-.2.pvj
Wer das nicht alles abtippen möchte hat hier einen Kurzlink zu dieser Adresse, damit geht es etwas schneller: https://t1p.de/7qym

Das Passwort um in diesen Ordner zu kommen lautet: `Sommeruni2019`

Geh bitte auf die Seite, gib das Passwort ein und drücke Login, du wirst nun auf die Webseite geleitet, wo du „GeanyPortable.7z“ herunterladen kannst. Während du weiterliest, kann es so schon mal herunterladen und von dir entpackt werden. Das entpacken geht mit 7-Zip (https://www.7-zip.de) ganz einfach. 7-Zip ist an der Uni auch schon installiert. Drück nach dem downloaden einfach rechtsklick auf die Datei, gehe zu „7-Zip“ -> „Hier entpacken“.

5.2.1 IMAP, SMTP, POP3, SSL, TLS – wie bitte?

Um mit dem Mailserver zu kommunizieren und auch Daten auszutauschen, benötigen wir ein bzw. besser noch verschiedene Protokolle. Wie diese Protokolle genau Funktionieren und wie sie eine Verbindung mit dem Mailserver aufbauen ist für uns eigentlich erstmal nicht so wichtig. Denn wir werden eine bereits vorhandene Implementierung verwenden. Dafür greifen wir auf eine Bibliothek zurück. Mehr dazu findest du im Kapitel JavaMail. Ich möchte hier aber dennoch ganz, ganz, wirklich gaaaaaanz kurz auf die Eigenschaften der Protokolle eingehen, damit du weißt was eigentlich passiert.
Als erstes fange ich mit POP3 (Post Office Protocol, in der Version 3) an. Es dient dazu die Mails vom Mailserver zu holen und dann bei Dir lokal zu verarbeiten. Klassisch wird dafür eigentlich die Mail vom Server abgefragt, zu Dir gesendet und dann beim Server gelöscht. Ist also nicht unbedingt so gut, wenn man eigentlich nur etwas herum probieren möchte und plötzlich sind die Mails auf dem Server gelöscht und man hat sie selber nicht abgespeichert. Hinzukommt, dass POP3 recht wenig Funktionalität hat und uns neben dem abholen der Mails sowie deren Auflisten nur noch das löschen zur Verfügung steht. Wir werden darum IMAP (Internet Message Access Protocol) verwenden, um die Mails vom Server zu holen. IMAP bietet sehr viele Funktionen und kann in einem „Read only“ Modus benutzt werden. Durch diesen Modus werden die Mails auf den Server belassen, aber wir können dennoch ohne Probleme alle Mails vom Server auslesen (sie aber dabei auf dem Server belassen), Ordnerstrukturen haben/benutzen, Mails verwalten usw… Natürlich kann man mit IMAP auch Mails vom Server löschen, wenn man will. Was ist nun der Vorteil von IMAP gegenüber POP3? Abgesehen, dass IMAP moderner ist und mehr Funktionen anbietet, ist IMAP inzwischen zum Standard geworden, wenn man verschiedene Rechner mit Mailclienten verwendet. Sowas passiert schnell, wenn man z.B. einen Mailclienten auf dem Smartphone verwendet, zusätzlich zu Hause einen Clienten verwendet und vielleicht noch woanders - wie auf der Arbeit - einen benutzt und alle greifen auf denselben Account zu. Der Vorteil an IMAP ist nämlich, es belässt alle Mails auf dem Server und wir verwalten unsere Mails quasi nur von dem Rechner aus. POP3 geht nicht mit unterschiedlichen Rechnern/Geräten, da hier alle Mails vom Server auf den Rechner lokal heruntergeladen werden und anschließend vom Server gelöscht werden. Die Mails können nun nur lokal verwaltet werden. Bedeutet loggt man sich von einem anderen Rechner mit einem Mailclienten über POP3 ein, wird man keine Mails mehr in diesem Clienten sehen, da auf dem Server bereits alle Mails gelöscht wurden und keine mehr heruntergeladen werden können (denn sie wurden ja alle auf dem anderen PC heruntergeladen).

Wir verwenden also IMAP um die Mails auszulesen, aber was würde man verwenden, wenn man Mails senden möchte? Dazu braucht man SMTP (Simple Mail Transfer Protocol). Hiermit werden die zu versendenden Mails an unseren Mailserver gesendet und von dort in das System eingespeist. Der Mailserver kümmert sich dann um das Versenden an die richtige Adresse. Wir müssen also nichts Großartiges beachten.

Da die Protokolle nicht sonderlich gesichert sind und man nicht nur deine Mails abhören könnte sondern auch gleich dein Passwort, welches du an den Server sendest, müssen wir alles sicher machen. Damit uns keiner belauschen kann und Daten stehlen kann, sollte man ein Verschlüsselungsprotokoll verwenden, das die Datenübertragung im Internet sicher macht. Einige Server verlangen darum schon von Anfang an einen Verbindungsaufbau mit SSL (Secure Sockets Layer) und verweigern den Zutritt zu jeglichen Diensten, wenn man die Verbindung nicht verschlüsselt. Einige Server wollen auch lieber schon die neuere Verschlüsselungsversion TLS (Transport Layer Security) verwenden. TLS ist dabei die Weiterentwicklung von SSL und somit moderner.

Damit eure Daten (Mails, Infos, Username, Passwort) nicht abgehört oder gar geklaut werden können, rate ich euch auch dazu immer ein Verschlüsselungsprotokoll zu verwenden.

5.2.2 JavaMail

Damit wir uns die weitere Vertiefung der Protokolle sparen können und nicht noch tiefer eintauchen müssen, können wir eine fertige Bibliothek verwenden, die diese Protokolle schon implementiert hat. Neben der Zeitersparnis ist ein weiterer Vorteil, dass wir darauf vertrauen können, dass alles korrekt programmiert wurde und wir dort keine Fehler mehr machen können.

Eines dieser Bibliotheken ist JavaMail (siehe auch https://de.wikipedia.org/wiki/JavaMail), es ist kostenlos und bereits bei der Java Enterprise Edition integriert, es kann aber auch ganz einfach in der normalen Java Standard Edition als zusätzliches Package nachgerüstet und benutzt werden. Es hat eine einfache API (application programming interface, bedeutet: Programmierschnittstelle) mit der wir die Protokolle und sonstige Befehle recht leicht umsetzen können. Die API ist für uns also die Schnittstelle zu den Protokollen und dadurch können wir unsere Mails leicht abfragen. Wie die Befehle genau aussehen, gehe ich später drauf ein. Erstmal möchte ich noch erklären, wo ihr JavaMail herbekommt und es benutzen könnt. All diese Sachen braucht ihr an der Uni nicht zu machen, da es schon alles in der Geany Version von mir eingestellt ist. Die Infos sind also eher dafür da, wenn ihr es zu Hause machen wollt oder mit einer anderen IDE ausprobieren wollt.

Als erstes braucht man JavaMail, das bekommt man direkt bei Oracle (den Entwickler von Java): https://www.oracle.com/technetwork/java/javamail/index.html

Auf der rechten Seite steht „Downloads“ und man kann es dort herunterladen. Hier befindet sich auch die Dokumentation dazu. Wenn ihr also ein neues Feature einbauen wollt und nicht genau wisst, wie ihr dafür die API ansteuern müsst, dann findet ihr in der Dokumentation vielleicht Hilfe dazu. Oder ihr googelt es und findet jemanden der schon mal ein ähnliches Problem hatte…

Hat man JavaMail nun heruntergeladen und entpackt, kann man die mail.jar und die jar-Dateien aus dem Ordner „lib“ nun in den Ordner kopieren, in dem man das Mail programm schreiben möchte. Es gibt hier viele verschiedene Wege, besonders wenn man eine „größere“ IDE wie Eclipse verwendet. Aber wir gehen jetzt von einem sehr einfachen Fall aus. Jetzt kann man diese Bibliothek verwenden, es muss nur beim kompilieren und ausführen auf diese Bibliothek hingewiesen werden, damit Java weiß, wo es die Schnittstellen findet. Wie das geht, verrate ich Dir im Kapitel Classpath einbinden.

5.2.3 Classpath einbinden

Damit beim Kompilieren und später beim Ausführen die JavaMail Bibliothek gefunden wird, müssen wir den Pfad dorthin angeben. Das geschieht über den Classpath.

Achtung: In der Geany Version die ich euch gegeben habe ist das schon erledigt. Wenn ihr es aber zu Hause machen wollt, dann geht es folgendermaßen:

In Geany könnt ihr unter „Erstellen“ -> „Kommandos zum Erstellen konfigurieren“ folgendes schreiben:

Beim kompilieren:

```
..\..\jdk1.8.0_60\bin\javac.exe -cp "%d\*;" "%f"
```

Der erste Teil führt dabei zum Java kompilierer, nämlich zu „javac.exe“. Ich habe euch das aktuelle JDK 8 schon gleich mit in Geany eingefügt, damit ihr es nicht downloaden müsst oder es sonstige Komplikationen gibt. Der Pfad führt also zum JDK, was bei Geany liegt. Machst du es zu Hause, musst du den Pfad zu deinem JDK ggf. anpassen. Nach javac.exe wird der Classpath mit „-cp“ angegeben. Um in dasselbe Verzeichnis, in dem wir die auszuführende Java Datei haben, zu kommen, schreiben wir „%d“. Dadurch wird beim kompilieren dann „%d“ mit dem aktuellen Pfad ersetzt. Mit dem „*“ dahinter geben wir an, dass wir alle Dateien in diesem Verzeichnis anbieten und benutzen können. So werden alle benötigten Bibliotheksimporte selber gefunden. Mit „%f“ wird dann noch der aktuelle Dateiname inklusive der Dateiendung (zum Beispiel „Email.java“) beim kompilieren automatisch eingefügt, so wird unabhängig wie wir die Datei nennen diese dort verwendet.

Bei Ausführen kann man folgendes schreiben:

```
java "-classpath" "%d\*;" "%e"
```

Wir führen nun mit der bereits installierten Java Umgebung unser Programm aus. Da Java sich einen Verweis zurechtlegt, brauchen wir nur Java schreiben und werden automatisch auf Java.exe im Programme Ordner verwiesen. Anschließen müssen wir noch den Classpath Weg zu den Bibliotheksimporten angeben. Diesmal muss classpath ausgeschrieben werden, der Pfad ist aber wieder mit „%d“ abkürzbar. Diesmal wird die kompilierte „.class“ zum ausführen verwendet und nicht die „.java“ Datei, welche den Quellcode beinhaltet. Da man beim Ausführen aber nie die Dateiendung .class mit anhängt, gibt man nur den Namen an (ohne Dateiendung). Das geschieht hier automatisch mit „%e“.

5.2.4 Ausführbar machen mit einer .bat Datei

Man kann das Programm auch recht simple mit einem kleinen Script starten, so braucht man nicht immer in die Konsole/Eingabeaufforderung zu gehen um Java aufzurufen oder gar eine IDE starten, um dann dort auf Ausführen/run zu klicken.

Unter Windows können wir recht simple mit einer „.bat“-Datei solch ein ausführbares Script erstellen. Unter Linux geht das mit einem Shell Script. Wer von euch Linux verwendet, weiß das vermutlich aber eh schon, darum gehe ich hier nur auf die Windows Version mit .bat ein.

Um ein bat Script zu erstellen, öffnest du am besten erst einmal Notepad oder einen anderen beliebigen Editor. Dort gibst du folgendes ein:

```
java "-classpath" "%~dp0*;" "Email"

pause
```

Und speicherst es als „start-script.bat“ in dem Ordner, in dem auch dein Java Mail Client liegen soll. Achte dabei unbedingt darauf, dass die Dateiendung .bat ist, damit es später auch ausgeführt werden kann! Willst du dieses Skript irgendwann nochmal verändern, kannst du es im Editor einfach öffnen und bearbeiten.

Dieses Script ruft Java auf und gibt den Classpath zum aktuellen Pfad, in dem diese bat-Datei gerade liegt an. „%~dp0“ gibt also den kompletten Pfad an. Diesmal ist hier sogar ein Backslash schon mit vorhanden, so dass wir kein Backslash mehr ergänzen müssen, wie es noch bei Geany in den Einstellungen nötig war. Mit dem Sternchen geben wir wieder an, dass sämtliche Bibliotheksimporte aus diesem Ordner verwendet werden können und

brauchen so nicht jede einzelne .jar Datei aufzuschreiben. „Email“ ist hier mein Programm, was wir später erstellen werden. Achte darauf, dass hier nicht „.class“ oder „.java“ steht! Das Programm heißt bei mir also „Email.java“, darum schreibe ich hier „Email“. Du kannst hier natürlich auch einen anderen Namen für ein anderes Java Programm angeben. Passe es dann ggf. an.

Wenn du nun dein Java Programm geschrieben hast, die richtige Datei in der Bat Datei zu stehen hast, es kompiliert hast (so dass eine .class Datei von deinem Programm im Ordner existiert), dann kannst du nun die Bat-Datei Doppelklicken und so das Java Programm starten.

Um ein Java Programm zu starten, gibt es natürlich viele Wege. Dies ist einer von vielen. Wie erwähnt geht auch die Eingabeaufforderung, es geht aber auch, dass du dein Programm in eine Ausführbare Jar-Datei umwandelst. Das geht am einfachsten mit größeren IDE’s wie Eclipse oder IntelliJ. Dort ist es nur ein simpler Export, den man aufrufen muss. Falls dich das interessiert, dann spiel gerne zu Hause ein bisschen mit diesen IDE’s rum und probiere es aus.

5.2.5 Server Adressen finden

Bevor wir nun endlich loslegen können, müssen wir noch die jeweilige Server Adresse herausfinden. Wir wollen dabei erstmal nur die IMAP Adressen erfahren, da wir erstmal nur Mails vom Server auslesen wollen. Wie man diese herausfindet ist bei jedem Anbieter unterschiedlich. Um Dir etwas Arbeit abzunehmen, zeige ich es hier für Google Mail bzw. Gmail und Hotmail (bzw. Outlook.de). Möchtest du es bei einem anderen Anbieter herausfinden kann man meist nach „IMAP“ suchen und findet dann die benötigten Daten.

5.2.5.1 Gmail IMAP Daten

Für Google finden wir die Einstellungen auf folgender Seite:
https://support.google.com/mail/answer/7126229

Dort steht:

Posteingangsserver (IMAP):	imap.gmail.com SSL erforderlich: Ja Port: 993

Wie der SMTP Server lautet steht dort auch, interessiert uns aber erstmal weniger.

5.2.5.2 Hotmail (bzw. Outlook.de/outlook.com) IMAP Daten

Für Hotmail (bzw. Outlook.de/Outlook.com) müssen wir uns erst einloggen und auf die Einstellungen gehen (rote 1) wie im Bild bei „Abbildung 1: Hotmail Einstellungen aufrufen“ zu sehen und dann auf „Alle Outlook-Einstellungen anzeigen“ (bei der roten 2) klicken.

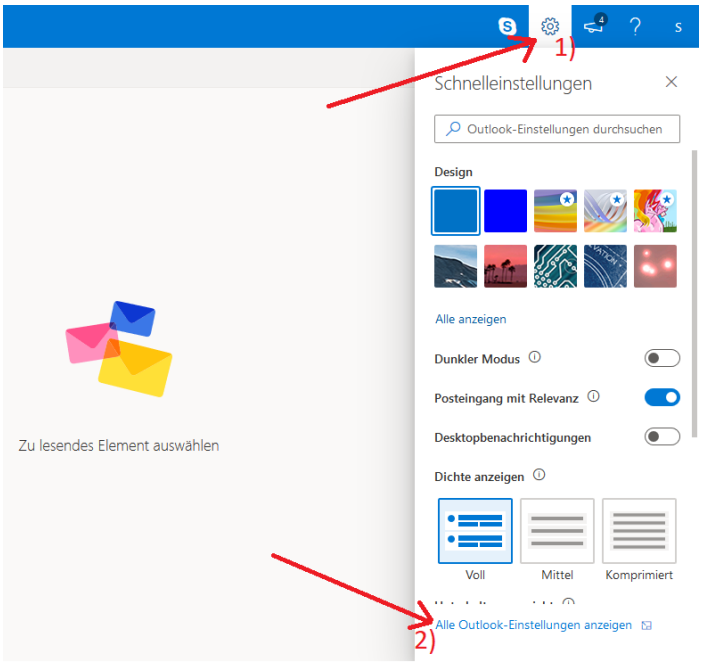

Abbildung 1: Hotmail Einstellungen aufrufen

Nun kann man bei „Emai synchronisieren“ klicken (rote 1) und sieht dann, wenn man nach unten scrollt, wie in „Abbildung 2: Hotmail IMAP Einstellungen“ zusehen, die IMAP Einstellungen.

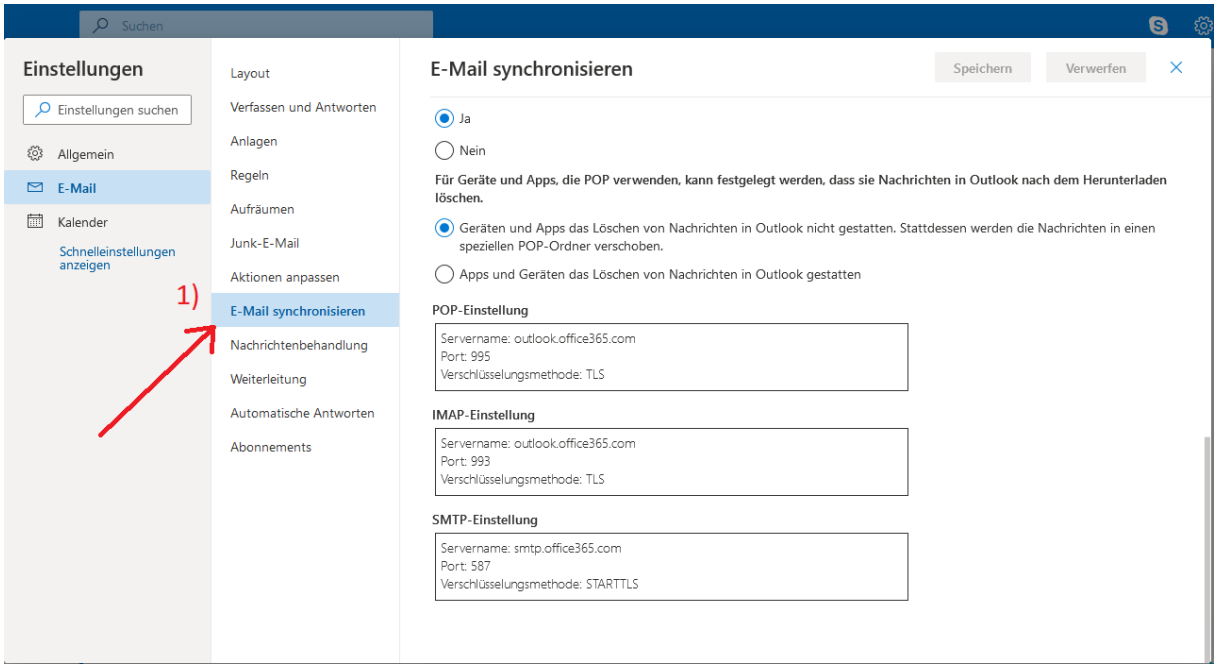

Abbildung 2: Hotmail IMAP Einstellungen

Die IMAP Einstellungen lauten also:

Posteingangsserver (IMAP):	Servername: outlook.office365.com Port: 993 Verschlüsselungsmethode: TLS

5.2.6 Gmail Postfach für externe Anwendungen freigeben

Google legt besonders viel Wert auf Sicherheit oder will seine eigenen Programme besser vermarkten, jedenfalls müssen wir um Gmail Accounts verwenden zu können, noch einiges vorher einstellen.

Wir müssen den Zugriff via IMAP erstmal erlauben. Dazu musst du dich erstmal in deinen Gmail Account mit einem Browser einloggen. Dann gehst du auf das kleine Zahnrad rechts oben (siehe rote 1) wie in „Abbildung 3: Gmail Einstellungen aufrufen“ zu sehen. Anschließend klickst du auf „Einstellungen“ (rote 2).

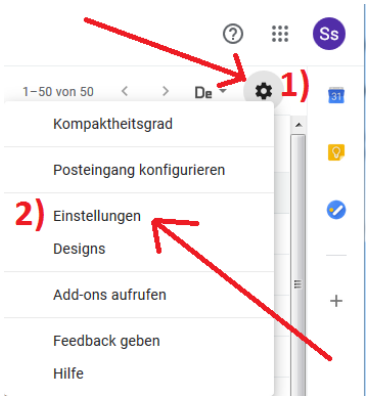

Abbildung 3: Gmail Einstellungen aufrufen

Wenn du das gemacht hast, öffnet sich die Einstellungen und du gehst wie in „Abbildung 4: Gmail IMAP Einstellungen“ zu sehen oben auf „Weiterleitung & POP/IMAP“ (rote 1). Hier musst du dann im „IMAP-Zugriff“ Bereich „IMAP aktivieren“ auswählen (siehe rote 2).

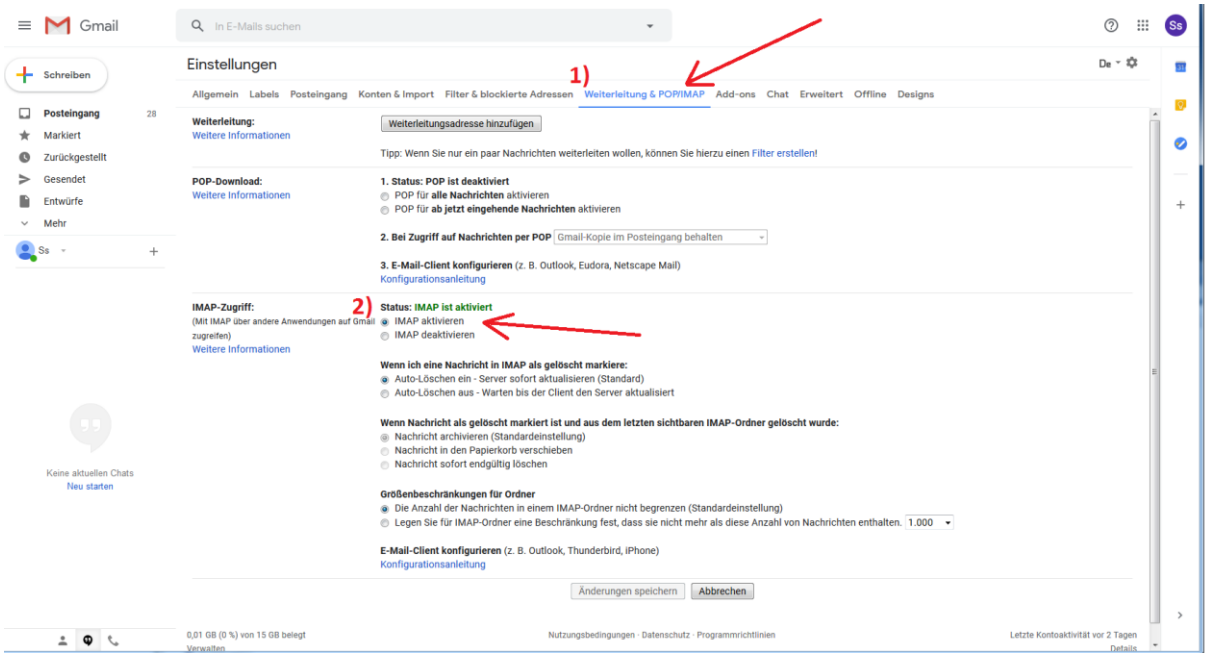

Abbildung 4: Gmail IMAP Einstellungen

Ok, nun ist IMAP zwar aktiviert, aber wenn wir das Programm später ausführen würden, würden wir dennoch keine Mails von Google bekommen. Das liegt daran, dass Google nur Programme erlaubt, die es kennt (also z.B. Outlook, Thunderbird, Apple Mail, o.ä.) oder eine

zusätzliche spezielle Authentifizierung namens OAuth haben möchte. Diese werden wir aber nicht implementieren, da es alles erstmal komplizierter machen würde. Keine Sorge um eure Sicherheit, wir verwenden für die Übertragung TLS bzw. SSL und haben so eine verschlüsselte Datenübertragung.

Es muss explizit gesagt werden, dass wir andere Programme den Zugriff auf Gmail erlauben wollen. Damit wir unser Java-Programm also benutzen können, wenn wir auf unsere Mails bei Gmail zugreifen wollen, müssen wir das noch einstellen. Dafür geh einfach oben rechts auf deinen Benutzer Icon (siehe rote 1) so wie in „Abbildung 5: Gmail Benutzer Icon - Google-Konto“ zu sehen. Anschließend klickst du auf „Google-Konto“ (rote 2).

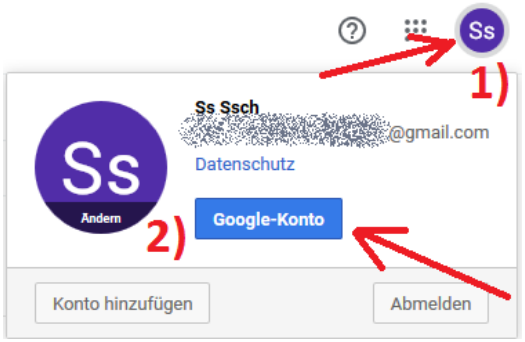

Abbildung 5: Gmail Benutzer Icon - Google-Konto

Es öffnen sich nun die Google Konto Einstellungen. Wie in „Abbildung 6: Google Konto Einstellungen“ zu sehen, klickst du nun erst auf „Sicherheit“ (rote 1) und dann nach dem runterscrollen bei „Zugriff durch weniger sichere Apps“ auf „Zugriff erlauben (nicht empfohlen)“.

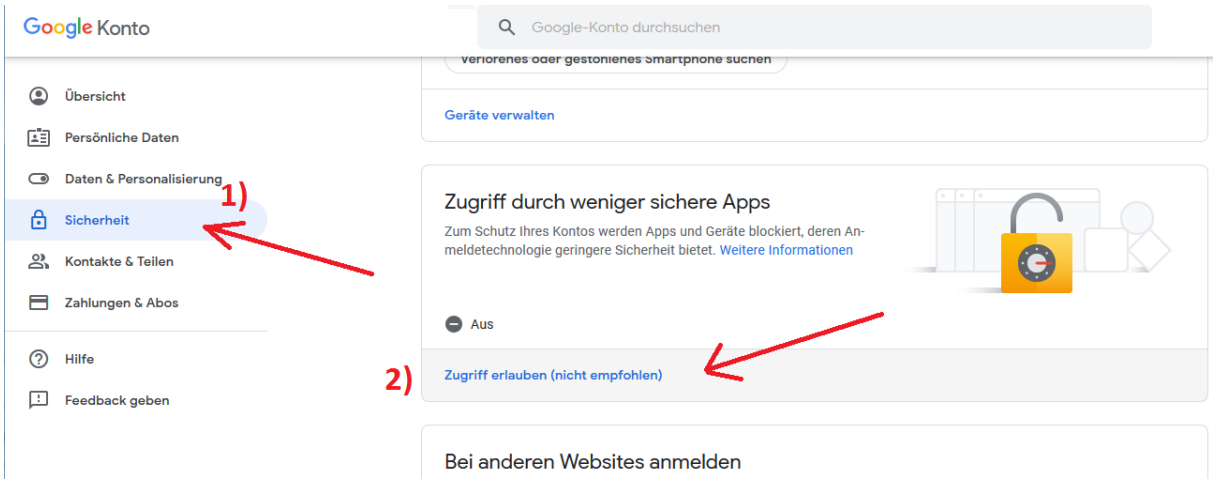

Abbildung 6: Google Konto Einstellungen

Es öffnet sich eine neue Seite, dort ziehst du den Regler einfach zur anderen Seite, so dass dort bei „Weniger sichere Apps zulassen“ von „Aus“ auf „An“ geschaltet wird. Das Umschalten sieht man bei Abbildung 7, so dass es dann aktiviert ist und so aussieht wie in Abbildung 8.

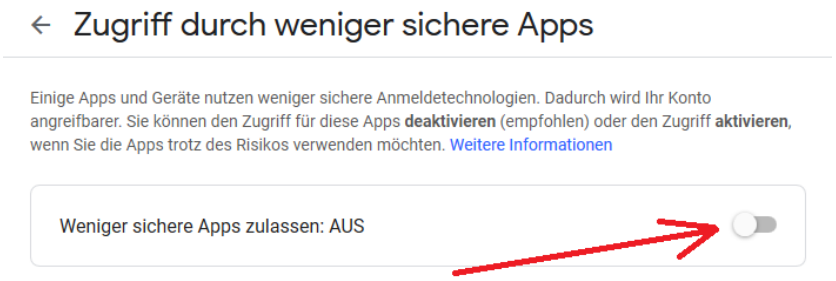

Abbildung 7: Gmail - Weniger sichere Apps zulassen ist auf AUS

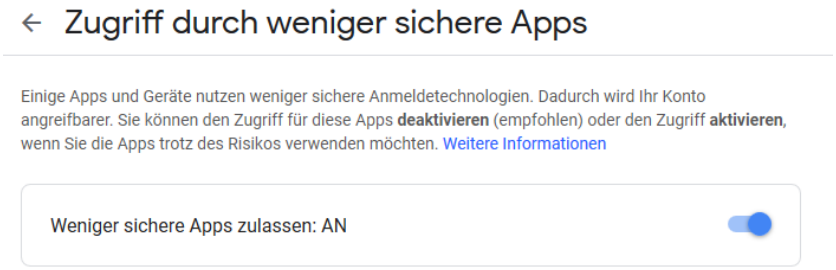

Abbildung 8: Gmail - Weniger sichere Apps zulassen ist auf AN

Wenn man nun zurück geht, sollte im Bereich „Zugriff durch weniger sichere Apps“ nun ein „An“ stehen. Es sollte also so aussehen, wie in Abbildung 9.

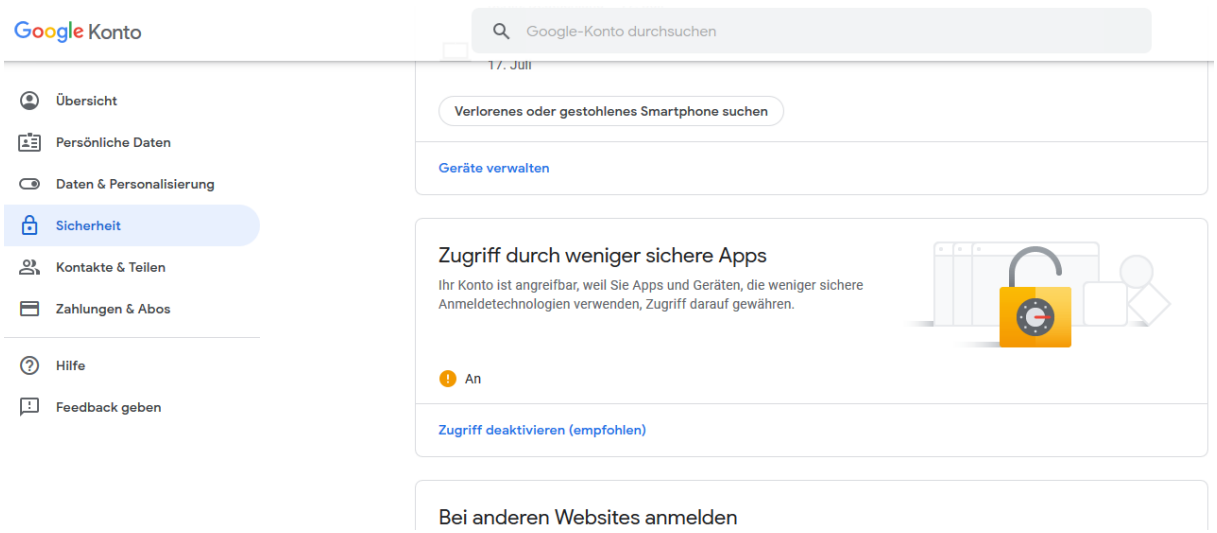

Abbildung 9:Google Konto Einstellungen - Zugriff weniger sicherer Apps ist nun "An"

Ab nun kann man auch eigene Programm dazu verwenden, um auf diesen Gmail Account zuzugreifen und die Mails von dort via IMAP zu bekommen.

Dieser Aufwand ist mir bisher nur bei Google bekannt, andere Mailanbieter scheinen so etwas (bis jetzt zumindest) nicht zu haben.

5.3 Mails empfangen

Jetzt habe ich dich schon ziemlich lange hingehalten und du möchtest sicher endlich losprogrammieren, um endlich Mails zu empfangen. Ok, legen wir mal direkt los.

Wir erstellen uns eine neue Java Datei mit Geany. Speichert diese am besten als „Email.java“ in „java_imap“ im „workspace_test“ Ordner ab. In diesem Ordner sind nämlich schon die ganzen benötigten Mail-Bibliotheken als jar-Datei hinterlegt. Wenn ihr nicht meine vorkonfigurierte Geany IDE verwendet, dann denkt dran wie in Kapitel „5.2.2 JavaMail“ beschrieben die Jar-Dateien in diesen Ordner zu kopieren und bei Geany die Classpath Pfade anzupassen, wie im Kapitel „5.2.3 Classpath einbinden“ erklärt.

In der noch leeren „Email.java“ schreiben wir erstmal eine Passworteingabe. Dabei soll das Passwort auf der Konsole eingegeben werden, aber nicht angezeigt werden, so dass jemand der neben uns sitzt oder jemand der später einen Blick auf die Eingabeaufforderung wirft, nicht unser Passwort sieht. Wir wollen das Passwort auch nicht fest in den Quelltext schreiben, da es hier genau so schnell von jemanden gesehen werden könnte, der vorbeikommt. Unseren Scanner von oben können wir so also nicht verwenden. Dafür gibt es aber über die Konsole die Möglichkeit eine Methode namens „readPasswort“ zu verwenden. Wir geben anschließend auf die Konsole die versteckte Eingabe aus und können so testen, ob alles richtig funktioniert. Später sollte das dann aber nicht mehr ausgegeben werden und ihr solltet unbedingt diese Ausgabe wieder löschen! Also jetzt nur zum Ausprobieren benutzen und mit zufallswerten testen und später wenn ihr diese Ausgabe auf die Konsole gelöscht habt, ist es auch sicher dort ein Passwort einzugeben.

```
import java.io.Console;  //für Passwortabfrage gibt es ab Java 1.6

public class Email
{
	public static void main (String[] args)
	{
		System.out.println("Bitte PW eingeben: ");
		Console console = System.console();
		String password = new String(console.readPassword("Password: "));
		//Das folgende ist nur zum Testen, ob alles funktioniert.
		//Später solltet ihr es löschen, damit nicht euer echtes
		//Passwort angezeigt wird!!!!
		System.out.println("PW bekommen... " +password); //Später löschen!
	}
}
```

Wir können unsere Methode `VonKonsoleTextEinlesen` aus „2 Das erste Programm“ nun verwenden, um dynamisch vom Benutzer abzufragen, von welchem Mail Server wir die Daten holen wollen, also wie die genaue IMAP Server Adresse aussieht und welcher Benutzername verwendet werden soll.

Da wir aber jetzt erstmal viel testen und probieren werden, schreibe ich die Werte erstmal fest in die Variable rein. Später könnt ihr das ja noch ändern, so dass der Benutzer das selber eingeben kann. Ihr müsst in die Variablen dann jeweils eure IMAP Adresse und Benutzernamen reinschreiben.

Tippt nun den folgenden Code ab und verändert die Datei „Email.java“ dementsprechend. Die Kommentare braucht ihr nicht abschreiben, sie dienen euch nur als Info und Erklärung.

```
import java.util.Scanner; //Importanweisung für den Scanner
import java.io.Console;  //für pw   gibt es ab Java 1.6

import java.util.Properties;

import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
//import javax.mail.NoSuchProviderException;
import javax.mail.Session;

import java.util.*;
import java.io.*;
import javax.mail.*;
import javax.mail.event.*;
import javax.activation.*;

import javax.mail.internet.*;

import com.sun.mail.imap.IMAPFolder;



public class Email
{
	public static void main (String[] args)
	{
		//Das Passwort werden wir aus Sicherheitsgründen
		//lieber jedes mal in die Konsole eingeben und speichern
		//es hier NICHT ab!!!!
		System.out.println("Bitte PW eingeben: ");
		Console console = System.console();
		String password = new String(console.readPassword("Password: "));
		//Passwort nicht mehr ausgeben lassen
		//System.out.println("PW bekommen... " +password);


		//Die folgenden Werte (host, user) könnt ihr gerne später,
		//wenn ihr aus der Testphase raus seid, dynamisch anpassen,
		//so dass man es auch jedes mal in die Konsole eingeben kann.

		//Der host (die Server-Adresse)
		//Wenn man google verwendet:
		//String host = "imap.gmail.com";
		//Wenn man hotmail/outlook.com verwendet:
		String host = "outlook.office365.com";//


		//Der Benutzername, mit dem du dich auch im Browser einloggst:
		//String user = "[Benutzername]@gmail.com";
		String user = "[Benutzername]@hotmail.de";

		try
		{

			Properties props = new Properties();
			//Eigenschaften einstellen.
			//Soll SSL oder TLS verwendet werden, braucht man das folgende:
			props.setProperty("mail.imap.ssl.enable", "true");
			//hier kann man
			//props.setProperty("mail.imap.ssl.enable", "true");
			//schreiben um SSL zu verwenden. Oder man benutzt TLS,
			//dafür schreibt man dann:
			//props.setProperty("mail.imap.starttls.enable", "true");
			//Also entweder schreibt man ssl oder starttls.
			//Bei den meisten sollte (das ältere) ssl funktionieren,
```

```
//darum verwenden wir einfach mal das als Einstellung.
//Du kannst das gerne später selber anpassen.
System.out.println("Properties eingestellt...");



//Jetzt kommen zwei Sachen, die für euch nicht
//so wichtig sind, die wir aber brauchen. Ihr braucht das
//nicht verstehen. Wir benutzen es einfach :)
//Ein Session Objekt bekommen:
Session session = javax.mail.Session.getInstance(props);
System.out.println("session wurde erzeugt...");
//Ein Stire Objekt bekommen:
Store store = session.getStore("imap");
System.out.println("store session mit imap wurde erzeugt...");


//Verbinden:
System.out.println("Starte Verbindungsaufbau. Connecting...");

store.connect(host, user, password);
System.out.println("...done");



//Wir versuchen nun den inbox Ordner zu bekommen,
//damit wir ihn gleich öffnen können.
//Der inbox Ordner ist einfach der Posteingang und
//hat somit die aktuellsten Mails. Standardmäßig heißt
//dieser Ordner inbox und beherbergt alle ankommenden
//Nachrichten. Willst du einen anderen Ordner öffnen,
//wie z.B. "gesendet", dann geht das natürlich auch.
//Der Ordner muss lediglich existieren.
//Achte evtl. darauf im Browser, wie die Ordner heißen,
//oft haben sie englische Namen.
System.out.print("Den inbox Ordner öffnen...");
//Achtung: Nicht verwirren lassen, wir nennen die Ordner
//Variable auch "inbox" genau wie den Ordner inbox,
//den wir aufrufen...
//Aber so wissen wir gleich anhand der Variable,
//welchen Ordner wir verwenden und was wir machen.
Folder inbox = store.getFolder("INBOX");
System.out.println("   ... done!");

//Vorsichtshalber noch überprüfen, ob es den
//zu öffnenden Ordner auf dem Server gibt:
if (inbox == null || !inbox.exists())
{
	//Nein, den Ordner gibt es wohl nicht...
	System.out.println("Unbekannter oder falscher Ordner!");
	//Programm beenden...
	System.exit(1);
}

//Den Ordner gibt es und wir machen normal weiter.
//Wir öffnen diesen Ordner nun, aber nur lesend,
//um nichts zu verändern!
//Ansonsten wäre: READ_WRITE zu benutzen
inbox.open(Folder.READ_ONLY);



//Wir holen uns nun alle Nachrichten in ein Message Array:
Message[] messages = inbox.getMessages();
//Wir haben nun alle Nachrichten in diesem Array!
//Dabei ist an der ersten Array stelle (Stelle 0)
//die aller erste Mail im Inbox Ordner und somit
//die älteste Mail. Die letzte Nachricht im messages Array
//ist die allerneuste Nachricht, also die zuletzt herein
```

```
//gekommene Nachricht im Inbox Ordner.

//Die länge dieses Arrays entspricht der Anzahl der Nachrichten
//auf dem Server. Denn wir haben ja alle Nachrichten in das
//messages Array geladen.
//Überprüfen wir das mal:
System.out.println("Wir haben " + messages.length +
" Nachrichten! (Das ist die Länge des messages Array)");

//Nun fragen wir mal den Server ab, wieviele Nachrichten
//er im inbox Ordner hat:
int totalMessages = inbox.getMessageCount();
System.out.println("Total messages = " + totalMessages);
//Die beiden Werte sollten auf der Konsole gleich sein.


//Falls es keine Nachrichten gibt, brauchen wir eigentlich
//nichts mehr machen und können das Programm beenden.
if (totalMessages == 0)
{
      System.out.println("Der Ordner ist leer. Nichts zu
      tun hier. Beende das Programm...");
      inbox.close(false);
      store.close();
      System.exit(1);
}
//Ok, wenn wir bis hier hin kommen, dann war der Ordner
//wohl nicht leer und wir können normal weiter machen.


//Dann fragen wir doch mal ab, wieviele neue Nachrichten es auf
//dem Server gibt:
int newMessages = inbox.getNewMessageCount();
System.out.println("New messages = " + newMessages);


//Ok, jetzt haben wir schon interessantes Zeug gelernt,
//aber nun wollen wir mehr zu den Mails wissen!!!!
System.out.println("-----------------------------------");


//Dafür lassen wir uns einfach mal die letzten
//fünf Mails ausgeben. Man kann sich natürlich auch alle
//ausgeben lassen, aber wenn man zu viele im Ordner hat
//(so wie ich), wird es böse...
//Wir haben gelernt, ein Array geht von 0 los und läuft bis
//länge des Arrays -1.
//Es lief bis eins weniger als die Länge des Arrays angibt,
//weil wir ja bereits bei Null anfangen zu zählen.
//Um also das letzte message Array Element zu bekommen
// - und somit die neuste Mail - müssen wir
// "messages.length - 1" schreiben.
//Von dort laufen wir 5 mails weiter runter. Aber vorher
//sollten wir noch überprüfen, ob wir auch überhaupt mehr
//als 5 Mails im Ordner haben, sonst wird es einen
//Fehler geben! Denn sonst läuft man in
//einen negativen Bereich.

//Um die neuste Mail oben stehen zu haben, beginnen
//wir mit dem zählen von hinten nach vorne.

int letzteNachricht = messages.length - 1;

//Wir zählen initial erstmal bis Null (also bis zur
//ältesten Nachricht) runter:
int zaehlenBis = 0;

//Wenn wir nun aber mehr als 5 Nachrichten auf den Server
//haben, zählen wir nur die letzten 5 neusten Nachrichten
```

```
//(und nicht mehr alles bis 0)!
if ( letzteNachricht > 5)
{
	//Dafür ändern wir also zaehlenBis:
	zaehlenBis = letzteNachricht - 5;
}


//Nun gehen wir in einer Schleife diese Nachrichten durch:
for (int i = letzteNachricht; i > zaehlenBis; i--)
{
	//Aktuelle Mail Nummer ausgeben:
	System.out.println("Mail Nr.: " + i);

	//Wir holen die Nachricht, die im messages Array an
	//dieser Stelle gespeichert ist und schreiben es
	//in message:
	//Achtung: Nicht verwirren lassen:
	//messages (mit einem s am Ende) ist das Array und hat
	//alle Nachrichten. Und message (ohne s am Ende) hat
	//nur eine einzige Nachricht gespeichert!
	Message message = messages[i];

	//Wir holen uns einen Flag und schauen mal nach,
	//ob die Nachricht bereits gelesen wurde:
	boolean isRead = message.isSet(Flags.Flag.SEEN);

	//Wurde die Nachricht bereits gelesen?
	if (!isRead)
	{
		System.out.print("[u]"); //[u] für ungelesen
		(u=unseen)
	}
	else
	{
		System.out.print("[s]"); //[s] für bereits
		gelesen (s=seen)
	}
	//Wir haben ein print verwendet, um keinen Zeilenumbruch
	//zu machen!
	//Wir lassen nun den Betreff direkt dahinter ausgeben:
	//Aber mit Tabulatoren eingerückt.
	System.out.println("\t" + "\t" + message.getSubject());


	// FROM
	//Wir wollen nun die Emailadresse vom Absender auslesen.
	//Da der Absender mehrere Adressen oder Angaben haben
	//kann, müssen wir alle Informationen erstmal wieder
	//in ein Array speichern. Das können wir dann auslesen:

	Address[] a;
	//Wir weisen die Adressen aus der aktuellen message
	//der Variable a zu und überprüfen dabei, ob es
	//überhaupt Daten hat (!=null).
	if ((a = message.getFrom()) != null)
	{
		//Es hat wohl Daten und nun können wir
		//alles ausgeben lassen. Meistens ist es
		//aber nur eine Adresse:
		for (int j = 0; j < a.length; j++)
			System.out.println("FROM: \t" +
			a[j].toString());
	}


	// TO
	//Wir überprüfen beim Empfänger Namen auch direkt bei
```

```
                        //der Zuweisung mit einem if, ob alles geklappt hat
                        //und es Daten gibt
                  if ((a = message.getRecipients(Message.RecipientType.TO))
                                                                  != null)
                        {
                              //die Zuweisung zu a hat geklappt. a wurde nun mit
                              //neuen Werten überschrieben und hat nun die
                              //Adressaten als Werte in sich.
                              //Diese lassen wir nun Ausgeben:
                              for (int j = 0; j < a.length; j++)
                              {
                                    System.out.println("TO: \t" +
                                                      a[j].toString());
                              }
                        }



                        // DATE
                        //Jetzte lesen wir noch das sende Datum aus:
                        Date d = message.getSentDate();
                        //Hier wird nun ein "ternary operator" verwendet
                        //(erkennbar an dem Fragezeichen und dem Doppelpunkt),
                        //der wie eine if-else Anweisung funktioniert. Nur das
                        //es in einer Zeile geschrieben wird. Es bedeutet:
                        //Wenn d ungleich null ist, dann wird d.toString()
                        //ausgeführt, ansonsten wird "UNKNOWN" ausgegeben.
                        System.out.println("Datum: \t" + (d != null ?
                                          d.toString() : "UNKNOWN"));

                        System.out.println("---------------------------------");
                        System.out.println();  //Einen Zeilenumbruch machen.


      //Hier könnte man nun schauen, ob man noch den Body Text mit ausgibt...

                  } //ende der for Schleife.

                  //Wir sind nun fertig mit allem und können alle
                  //Verbindungen schließen
                  inbox.close(false);
                  store.close();

            } //ende von try
            catch (Exception ex)
            {
                  //Falls etwas schief läuft, dann geben wir erstmal
                  //nur folgendes aus.
                  //Du kannst das gerne noch weiter anpassen und
                  //genauere Fehlermeldungen ausgeben
                  ex.printStackTrace();
            }


      }       //ende von main
} //ende der Klasse
```

Probiert den Quellcode aus. Ihr könnt den Mailclienten gerne so anpassen, wie ihr es gut und schön findet.

Wenn etwas nicht funktioniert, vergleicht nochmal den Code der hier steht mit eurem Code, ansonsten schaut mal in das Kapitel „5.4 Fehlermeldungen und Probleme – Erste Hilfe“.

Möchtet ihr noch den Email (Body) Text mit ausgeben? Das ist ein bisschen komplizierter. Aber wenn man in die mitgelieferten Demos von JavaMail schaut, findet man schnell in \javamail1_4_5\javamail-1.4.5\demo\msgshow.java
eine passende Lösung.

Wir können diese Lösung fast komplett kopieren und müssen nur am Anfang und am Ende der Methode „dumpPart“ etwas ändern.

Um den Text der Email auszulesen kopiert einfach diese angepasste Funktion in eure Klasse. Um es in die Klasse zu kopieren, kommt es also genau hier hin:

```
      }      //ende von main
/////HIER/////
} //ende der Klasse
```

Schreibt nun die folgenden zwei Methoden ab:

```
        //Wir brauchen noch einige Variablen:
        static boolean showStructure = false;
        static boolean saveAttachments = false;
        static int level = 0;

public static void dumpPart(Part p) throws Exception {
//      if (p instanceof Message)  dumpEnvelope((Message)p);

        /** Dump input stream ..

        InputStream is = p.getInputStream();
        // If "is" is not already buffered, wrap a BufferedInputStream
        // around it.
        if (!(is instanceof BufferedInputStream))
            is = new BufferedInputStream(is);
        int c;
        while ((c = is.read()) != -1)
            System.out.write(c);

        **/

        String ct = p.getContentType();
        try {
            pr("CONTENT-TYPE: " + (new ContentType(ct)).toString());
        } catch (ParseException pex) {
            pr("BAD CONTENT-TYPE: " + ct);
        }
        String filename = p.getFileName();
        if (filename != null)
            pr("FILENAME: " + filename);

        /*
         * Using isMimeType to determine the content type avoids
         * fetching the actual content data until we need it.
         */
        if (p.isMimeType("text/plain")) {
            pr("This is plain text");
            pr("---------------------------");
            if (!showStructure && !saveAttachments)
               System.out.println((String)p.getContent());
        } else if (p.isMimeType("multipart/*")) {
            pr("This is a Multipart");
            pr("---------------------------");
            Multipart mp = (Multipart)p.getContent();
            level++;
            int count = mp.getCount();
```

```
            for (int i = 0; i < count; i++)
               dumpPart(mp.getBodyPart(i));
            level--;
        } else if (p.isMimeType("message/rfc822")) {
            pr("This is a Nested Message");
            pr("---------------------------");
            level++;
            dumpPart((Part)p.getContent());
            level--;
        } else {
            if (!showStructure && !saveAttachments) {
               /*
                * If we actually want to see the data, and it's not a
                * MIME type we know, fetch it and check its Java type.
                */
               Object o = p.getContent();
               if (o instanceof String) {
                   pr("This is a string");
                   pr("---------------------------");
                   System.out.println((String)o);
               } else if (o instanceof InputStream) {
                   pr("This is just an input stream");
                   pr("---------------------------");
                   InputStream is = (InputStream)o;
                   int c;
                   while ((c = is.read()) != -1)
                      System.out.write(c);
               } else {
                   pr("This is an unknown type");
                   pr("---------------------------");
                   pr(o.toString());
               }
            } else {
               // just a separator
               pr("---------------------------");
            }
        }


    }



    /**
     * Print a, possibly indented, string.
     */
    public static void pr(String s) {
            System.out.println(s);
    }
```

Um die Funktion aufzurufen, musst du an der Stelle, wo der Nachrichten Text ausgegeben werden soll einfach nur die Methode mit übergebener message aufrufen, das würde dann so aussehen:

```
dumpPart(message);
```

Ich würde diesen Methodenaufruf an die Stelle einfügen, wo ich bei den Kommentaren folgendes geschrieben hatte:

```
//Hier könnte man nun schauen, ob man noch den Body Text mit ausgibt...
```

Beachte hierbei bitte, dass noch Kommentare über den Mime Typ in der Methode mit ausgegeben werden. Zum testen und herumprobieren ist das sicher gut. Später kann man das aber auskommentieren und nicht ausgeben lassen. Der Mime Typ zeigt uns an, in welchem Format die Mail ist. So kann es nicht nur ein simpler Text sein, sondern auch eine

Nachricht bestehend aus mehreren Teilen sein. Der Inhalt könnte auch eine HTML Datei sein, die hier dann gar nicht interpretiert wird. Man sieht dann nur lauter seltsamer HTML Syntax. Es ist aber auch möglich diese HTML Emails zu filtern und den Text dort auszulesen. Wenn man danach im Internet sucht, findet man schnell eine Lösung dazu. Z.B.: https://stackoverflow.com/questions/11240368/how-to-read-text-inside-body-of-mail-using-javax-mail

Für diese Lösung braucht man aber noch jsoup, das bekommt man unter: https://jsoup.org/download
Einfach die jar-Datei herunterladen und in denselben Pfad wie Email.java packen. Man muss nur noch in Email.java das jsoup importieren. Dafür schreibst du ganz oben in die Java Datei bei den imports das folgende einfach hinzu:

```
import org.jsoup.Jsoup;
```

Da wir alle Bibliotheken im selben Ordner eh schon laden lassen, wird diese Bibliothek aus der jar-Datei beim kompilieren und ausführen auch automatisch benutzt.

5.4 Fehlermeldungen und Probleme – Erste Hilfe

Es kann zu vielen verschiedenen Fehlermeldungen kommen. Es können einerseits schon Probleme beim kompilieren auftreten und es können Fehler (Exceptions) beim Ausführen des Programmes auftreten.

5.4.1 Bei Compiler Fehlern

Wenn schon Probleme beim kompilieren auftreten und das Programm erst gar nicht kompilieren will, ist der Quellcode an irgendeiner Stelle nicht korrekt.

Das einfachste ist: Fehlermeldung lesen, das hilft oft weiter und zeigt einem wo das Problem liegt. Mit Doppelklick auf die Fehlermeldung kommt man zu dieser und kann sich ansehen was in diesem Bereich los ist und warum der Compiler motzt. Fangt auch beim ersten Fehler an und nicht beim letzten. Manchmal hat der Fehler Abhängigkeiten und löst ihr den ersten Fehler, verschwinden die anderen darunter.

Beliebte Fehler sind Vertipper und das vergessene Semikolon („;“ am Ende einer Anweisung). Auch falsche Klammerung bzw. vergessene Klammern führen häufiger zu Problemen. Manchmal fehlt auch ein import einer Bibliothek, dessen Methoden man schon verwendet.

Vergleicht an der Stelle wo das Problem auftritt den Quellcode mit meinem Code und schaut an, ob ihr einen Vertipper habt oder etwas vergessen habt. Hilft die Fehlermeldung zum lösen nicht weiter, könnt ihr mich natürlich gerne Fragen und ich versuche euch zu helfen. Wenn ihr auf Fehler zu Hause stoßt die nicht lösbar sind und der Hilfetext vom Compiler nicht weiterhilft, könnt ihr nach der Fehlermeldung bzw. der Beschreibung auch mal googeln und vielleicht findet ihr etwas dazu.

5.4.2 Fehler beim Ausführen des Programmes - Exceptions

Das Programm muss mit dem Internet verbunden sein, um auf die Mails zugreifen zu können. Ist man nicht im Internet, geht das natürlich nicht und es wird eine Exception vom Programm geworfen. Auch eine Firewall kann das Programm behindern ins Internet zu kommen. Dann wird ein Timeout das Porgramm auch irgendwann dazu bringen, eine Exception zu werfen, denn es kann den Server nicht erreichen. Weitere ähnliche Fehlermeldungen entstehen bei falscher Eingabe der Serveradresse, des Benutzernamens oder des Passwortes.

Es kann auch sein, dass ihr wie im Kapitel „5.2.6 Gmail Postfach für externe Anwendungen freigeben“ etwas aktivieren müsst, bevor man sich via IMAP auf den Server einloggen kann.

Überprüft auch, dass ihr keine Logikfehler im Programm habt. Nicht dass ihr eine negative Email-Nummer abfragen wollt oder ähnliches.

Was auch noch passieren kann ist, dass ihr das Programm hier an der Uni geschrieben habt, ausführbar gemacht habt und nun zu Hause testen wollt. An der Uni hat es funktioniert aber zu Hause nicht mehr. Ihr bekommt eine ähnliche Fehlermeldung wie in „Abbildung 10Abbildung 10: Gmail erkennt die IP nicht“ zu sehen ist. Wenn ihr ein Gmail Konto habt, kann es daran liegen, dass er zu Hause die IP nicht erkennt oder noch nicht gespeichert hat für dieses Programm. Auch wenn ihr alle Einstellungen wie oben beschrieben schon in der Uni vorgenommen habt. Loggt euch dann einmal mit dem Browser bei Gmail ein, dann hat Google die Verifizierung und IP, dass weiterhin alles okay ist und ab nun wird auch das Programm bei euch zu Hause funktionieren.

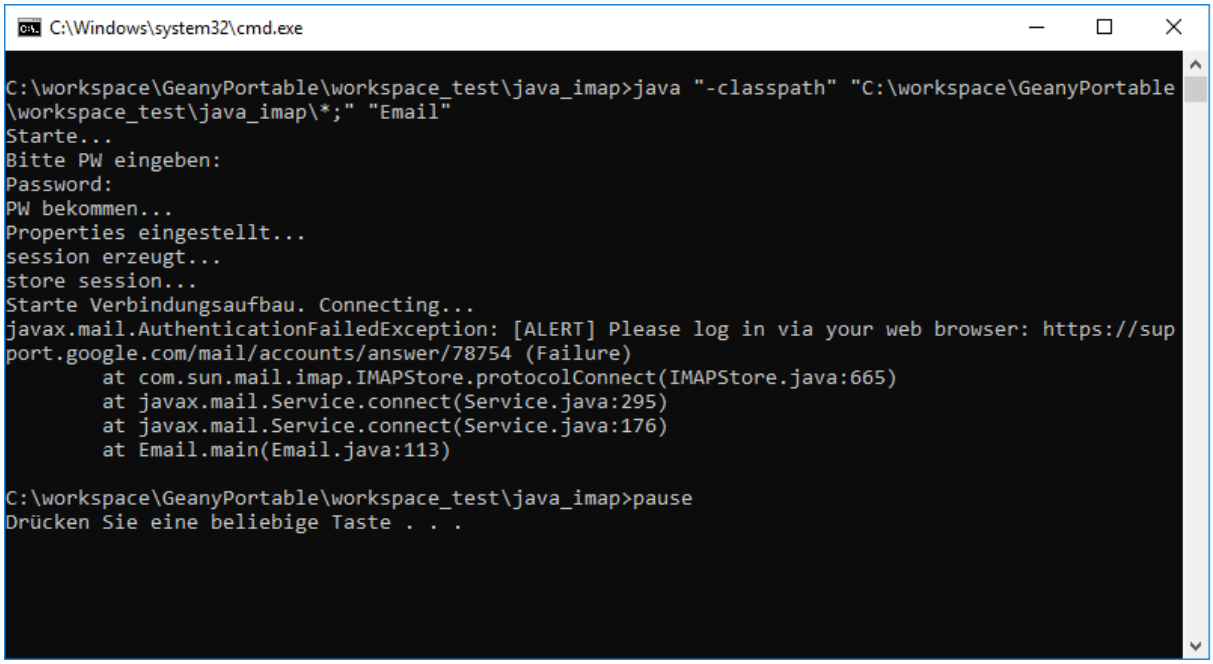

Abbildung 10: Gmail erkennt die IP nicht

Falls deine Exception immer noch nicht gelöst ist, du die Meldung nicht verstehst und du auch sicher bist, dass du keinen Logik Fehler im Programm hast, können wir uns das gerne zusammen anschauen. Alternativ kannst du natürlich auch nach der Excpetion erstmal Googeln und schauen, ob du dazu etwas findest. Was auch oft hilfreich ist, ist das Programm zu debuggen, darauf gehe ich aber nicht sehr genau ein, weil es zu viel wäre und den Rahmen sprengen würde. Du könntest dir aber kleine (Zwischen-)Schritte im Programm und einige Werte einfach als Zusatzinformation auf die Konsole mit rausschreiben (via System.out.println). So kann man die Fehler auch genauer eingrenzen und sieht, ab wann bzw. in welchem Teil des Codes der Fehler auftritt und eine Exception erscheint oder welche Werte nicht mehr in Ordnung sind und zu Fehlern führen. Im Bild „Abbildung 10: Gmail erkennt die IP nicht“ sieht man zum Beispiel, dass er auf die Konsole noch „Starte Verbindungsaufbau. Connecting…“ geschrieben hat und dann eine Exception geworfen wurde. Im Code weiß ich also nun, dass es kurz hinter dieser Ausgabe ein Problem gibt, denn ich habe etwas darunter noch im Code geschrieben, dass nach erfolgreichem Aufbau der Verbindung noch auf die Konsole geschrieben werden soll „…war erfolgreich“. Das wurde aber nicht mehr auf die Konsole geschrieben, sondern nur die Fehlermeldung. So habe ich den Fehler schnell einkreisen können.

5.5 Ausblick und Erweiterungsmöglichkeiten

Du hast nun einen kleinen sehr rudimentären Emailclienten. Der noch nicht viel kann. Du kannst ihn aber massiv erweitern, sofern du möchtest. Ich möchte Dir noch einige Ideen und Inspirationen auf den Weg geben, vielleicht magst du ja das ein oder andere noch realisieren.

Es ist natürlich möglich nicht nur ein Email Postfach abzufragen, sondern alle deine Postfächer abzufragen und die neusten oder relevantesten Mails anzuzeigen.

Man könnte neben dem empfangen noch das senden via SMTP ermöglichen und Mails versenden.

Es wäre möglich eine automatische Antwort an Freunde zu verschicken, wenn etwas sein soll.

Du könntest eine Suche implementieren, bei der jede Mail im Array nach einem Suchwort durchsucht wird. Vielleicht soll nach einem speziellen Absender gesucht werden und alle Mails von einem Freund aufgelistet werden oder vielleicht lässt man sich alle Mails mit dem Betreff „Facebook“ o.ä. ausgeben.

Du kannst auch regelmäßig den Server checken nach neuen Mails (bitte nicht übertreiben und einen zu kleinen Wert fürs checken setzen, um den Server nicht zu flooden und zuzuspammen).

In der Kombination mit Mails checken, kann man natürlich sehr gut auf eine spezielle Mail reagieren. Bedeutet zum Beispiel: Ein Freund sendet dir eine Nachricht, dann lass einen speziellen Sound starten, der dich darauf aufmerksam macht.
Amazon sendet Dir Blitzangebote? Na, dann soll das Java Programm einen Sound abspielen lassen und automatisch den Browser mit dem Link starten lassen.

Oder wie wäre es mit automatischem löschen einer Mail, wenn du eine Mail mit dem Betreff „Du hast gewonnen“ o.ä. bekommst?

Du kannst Dir die neuste Mail auch vorlesen lassen, sobald sie ankommt. Googel einfach mal nach „Text to Voice Java“ oder „Text to Speech Java“. Das wird übrigens besonders lustig, wenn du eine Spammail erhälst ☺

Wie wäre es mit einer eigenen Oberfläche. Wir haben hier nur ein simples Konsolen Programm erstellt, es wäre aber auch möglich eine GUI (graphical user interface) mit Java zu erstellen. Das würde dann mehr nach einem Programm aussehen und nicht nach einer Konsolen-Anwendung. Den Logik Teil des Programmes kann man ja wiederverwenden.

Es gibt sicher noch viele andere Möglichkeiten. Ich hoffe ich konnte Dir schon genug Inspiration liefern und dich motivieren dich weiter mit Java zu beschäftigen.
Auch hoffe ich du hattest Spaß an diesem Kurs und wünsche Dir viel Erfolg beim Programmieren :)