Arbeiten mit dem Pad-Modul

1. Einführung grafische Benutzeroberflächen

Die Ausführung aller bisherigen Programme hat lediglich eine Ausgabe auf der sogenannten „Konsole“ möglich gemacht. Dort können aber nur Texte ausgegeben werden, eine Formatierung ist nur mit einfachsten Mitteln (Abstand, Leerzeilen, …) möglich.

Heutige Programme (wie z.B. Word) arbeiten aber üblicherweise mit grafischen Elementen. So wird jedes Programm in einem „Fenster“ geöffnet und bietet dem Benutzer eine Vielzahl von Möglichkeiten. Auch Java kennt die Programmierung von grafischen Benutzeroberflächen. Sie ist jedoch sehr komplex und Stoff der dritten Klasse.

Das (von Hr. Prof. Buchberger) implementierte Pad-Modul erlaubt es jedoch mit einfachen Mitteln zumindest eine grafische Ausgabe zu realisieren und damit farbige, grafikorientierte Ergebnisse zu präsentieren.

1. Pad\_Modul – Grundlagen (siehe ShowPad\_1)

Das Pad-Modul besteht aus einer Klasse „Pad“, die in den Programmen verwendet werden kann. Dazu muss sie jedoch mit dem eigenen Programm „verbunden“ werden. Da sie nicht zu den Standardklassen in Java gehört (wie z.B. Strings) ist sonst eine Verwendung nicht möglich.

* 1. Einbindung in JGrasp

Die einfachste Methode das Pad-Modul in ein mit JGrasp-erstelltes Programm einzubinden liegt außerhalb der DIE. Folgende Schritte sind dazu notwendig:

* Anlegen eines Ordners, in den das Programm später gespeichert wird.
* Kopieren der Datei „Pad.java“ in diesen Ordner
* Öffnen von JGrasp und Anlegen einer „Java“-Datei
* Beim Speichern den Ordner mit der Pad-Datei wählen

Beim Ausführen muss man darauf achten, dass nicht die Pad-Datei ausgeführt wird, sondern das eigentliche Programm!

* 1. Voraussetzung im Programm schaffen

Da das Pad-Modul Komponenten von Java verwendet müssen diese dem Programm bekannt gemacht werden. Dazu ist ein „import java.awt.\*“ am Beginn der Datei notwendig.

***Beispiel:***

/\* MyFirstWindow

\* Musterlösung Mag. Danner

\* Mai 2020

\*/

import java.awt.\*

public class MyFirstWindow{

…

* 1. Ein Fester erzeugen (siehe ShowPad\_1)

Will man nun am Bildschirm ein Fenster erzeugen, so geschieht dies durch das Anlegen eines neuen „Pad“, also mit Hilfe des Statements „Pad window = new Pad(„Mein erstes Fenster“). Der dabei übergebene String dient als Überschrift über dem Fenster und kann entfallen. Startet man das Programm, so entsteht ein Standardfenster im linken oberen Eck des Bildschirms. Allerdings kann man das Fenster nicht sehen, ohne dass man den Befehl gegeben hat es sichtbar zu machen.

***Beispiel (siehe ShowPad\_1)***

Folgendes Programm:

import java.awt.\*

public class ShowPad\_1{

public static void main(String[] args){

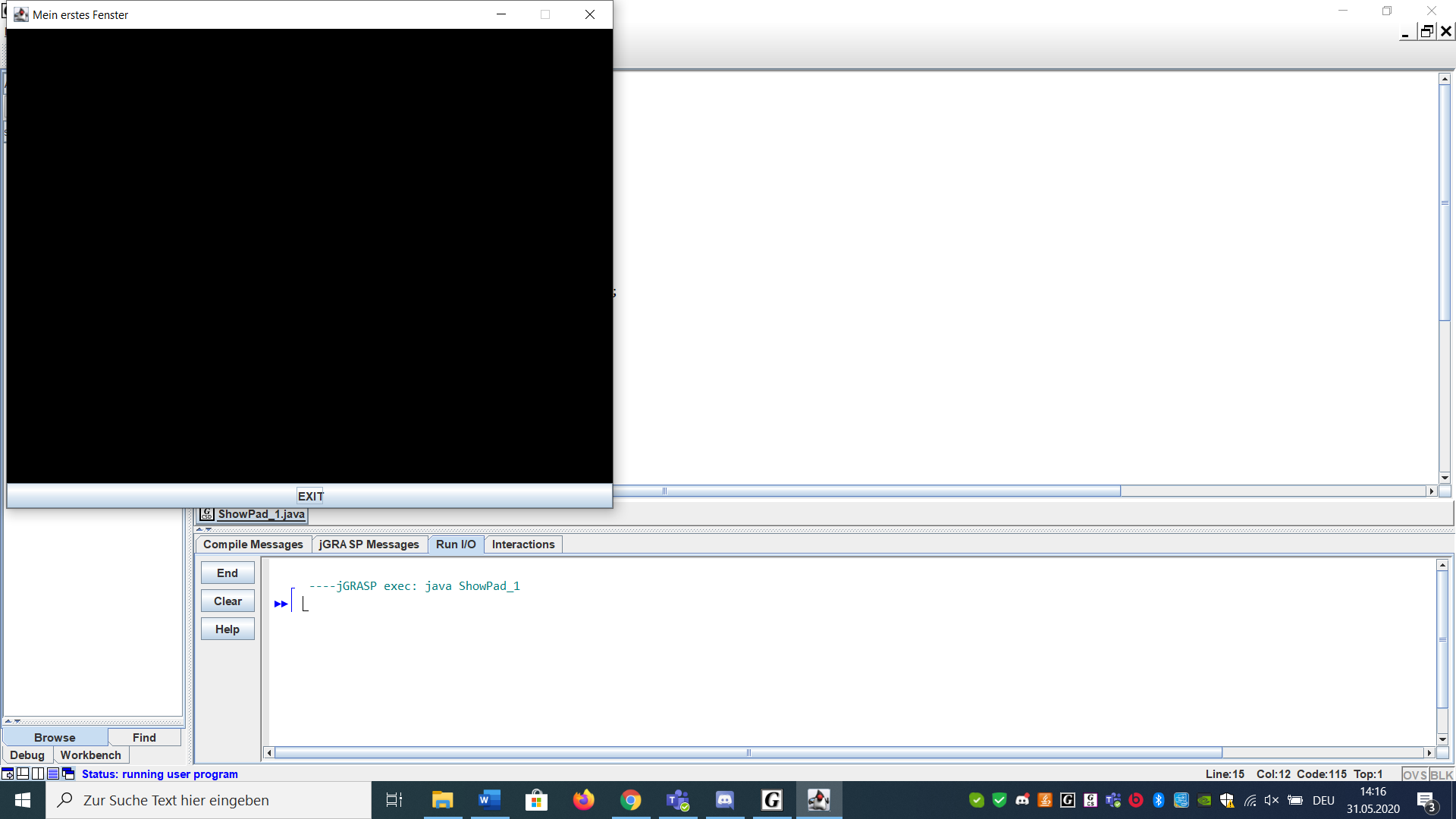
Pad myWindow = new Pad(„Mein erstes Fenster“);

myWindow.setVisible(true);

}

}

erzeugt folgendes Bildschirmbild:

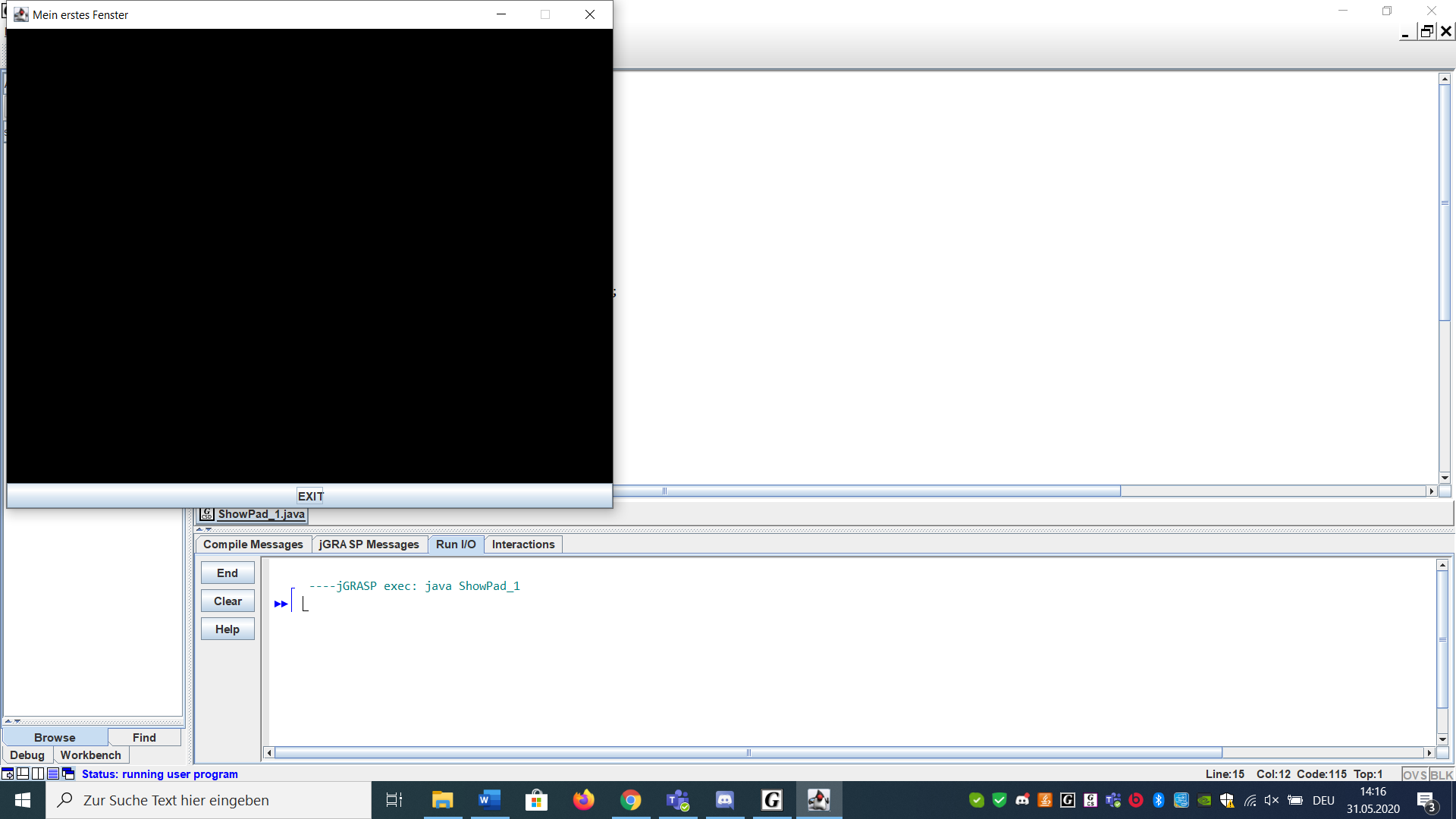


* 1. Standardaufbau des Fensters

b

1

a



2

3

c

Wie in dem Beispiel zu sehen, öffnet sich ein Standardfenster, wie wir es von Windows gewöhnt sind.

Im Wesentlichen besteht es aus der Kopfzeile (1), der Zeichenfläche (2) und der Fußzeile (3). Die Unterscheidung „Fenster“ und „Zeichenfläche“ wird vor allem bei den Positionsangaben noch wichtig werden.

Der Text erscheint linksbündig in der Kopfzeile (a). Ebenfalls enthält das Fenster die drei bekannte Symbole

* „Hinunterlegen“
* „Vollbild, bzw. Verkleinern“
* „Beenden“

wobei jedoch nur das erste und dritte bereits funktionstüchtig sind (b)

Zusätzlich enthält das Fenster in der Fußzeile einen Button „Exit“, der zum Beenden des Programms verwendet werden kann (c).

* 1. Anwendung beenden

Um das Programm zu beenden und das Fenster zu schließen sind zwei Möglichkeiten gegeben:

* Verwendung des standardmäßigen „X“ am rechten oberen Rand
* Klick auf den „Exit“-Button am unteren Rand.

Beide Aktionen schließen nicht nur das Fenster, sondern beenden auch das Programm korrekt, was durch die Ausgabe von

----jGRASP: operation complete.

in der Konsole bestätigt wird. Sind mehrere Fenster offen, so ist es egal, in welchem Fenster die Aktion ausgeführt wird. Sie schließt immer alle Fenster und beendet die Anwendung.

* 1. Methoden aufrufen

Ähnlich wie bei Strings können auch für Fenster, die mit Pad erzeugt wurden, unterschiedlichste Methoden aufgerufen werden. Die Art des Aufrufs entspricht dem von Strings bereits bekannten – die sogenannte Punkt-Notation.

Zur Erinnerung: Methoden werden dabei immer in der Form

*Variablenname****.****Methodenname(Parameterliste)*

aufgerufen.

* 1. Standardnamensgebung

Viele der Methoden folgen einer Standardnamensgebung in Java. Wichtig dabei ist zu wissen, dass

* Methoden, die zum Verändern eines Zustandes dienen immer mit „set“ beginnen, wie z.B. setVisible
* Methoden, die dazu dienen einen Zustand abzufragen immer mit „get“ beginnen, wie z.B. getHeight
  1. Farbangaben

Um im Pad-Modul eine Farbe zu bestimmen gibt es zwei Möglichkeiten

* + 1. Standardfarben

Im Pad-Modul sind einige Farben vordefiniert. Sie können mit „Pad.Farbname“ angegeben werden. Dabei sind folgende Farben verfügbar:

|  |  |  |
| --- | --- | --- |
| Aufruf | Bezeichnung | Farbe |
| black | schwarz |  |
| blue | blau |  |
| green | grün |  |
| magenta | rosa |  |
| red | rot |  |
| white | weiß |  |
| yellow | gelb |  |
| lightBlue | hellblau |  |
| mediumBlue | mittelblau |  |
| lightYellow | hellgelb |  |

* + 1. Selbstdefinierte Farben

Durch die Angabe eines RGB-Codes können Farben auch selbst gestaltet werden. Der Aufruf im setColor erfolgt dann mit drei Int-Werten, die einen Farbcode angeben.

Beispiel:

setColor(150,20,37) erzeugt folgenden Farbton

|  |
| --- |
|  |

1. Verändern des Fensters (siehe ShowPad\_2)

Die nachfolgend beschriebenen Methoden verändern das Standardfenster, ohne etwas darauf auszugeben.

* 1. Sichtbarkeit ändern

Die Sichtbarkeit gibt an, ob ein Fenster gerade angezeigt werden soll oder nicht. Mit „setVisible(true)“ wird ein Fenster sichtbar gemacht, mit „setVisible(false)“ unsichtbar. Wichtig dabei ist zu wissen, dass die Standardeinstellung „false“ ist, d.h. ein Fenster muss nach dem Anlegen bewusst sichtbar gemacht werden. Das Statement setVisible(true) muss immer das letzte Statement beim Anlegen des Fensters sein.

* 1. Position bestimmen

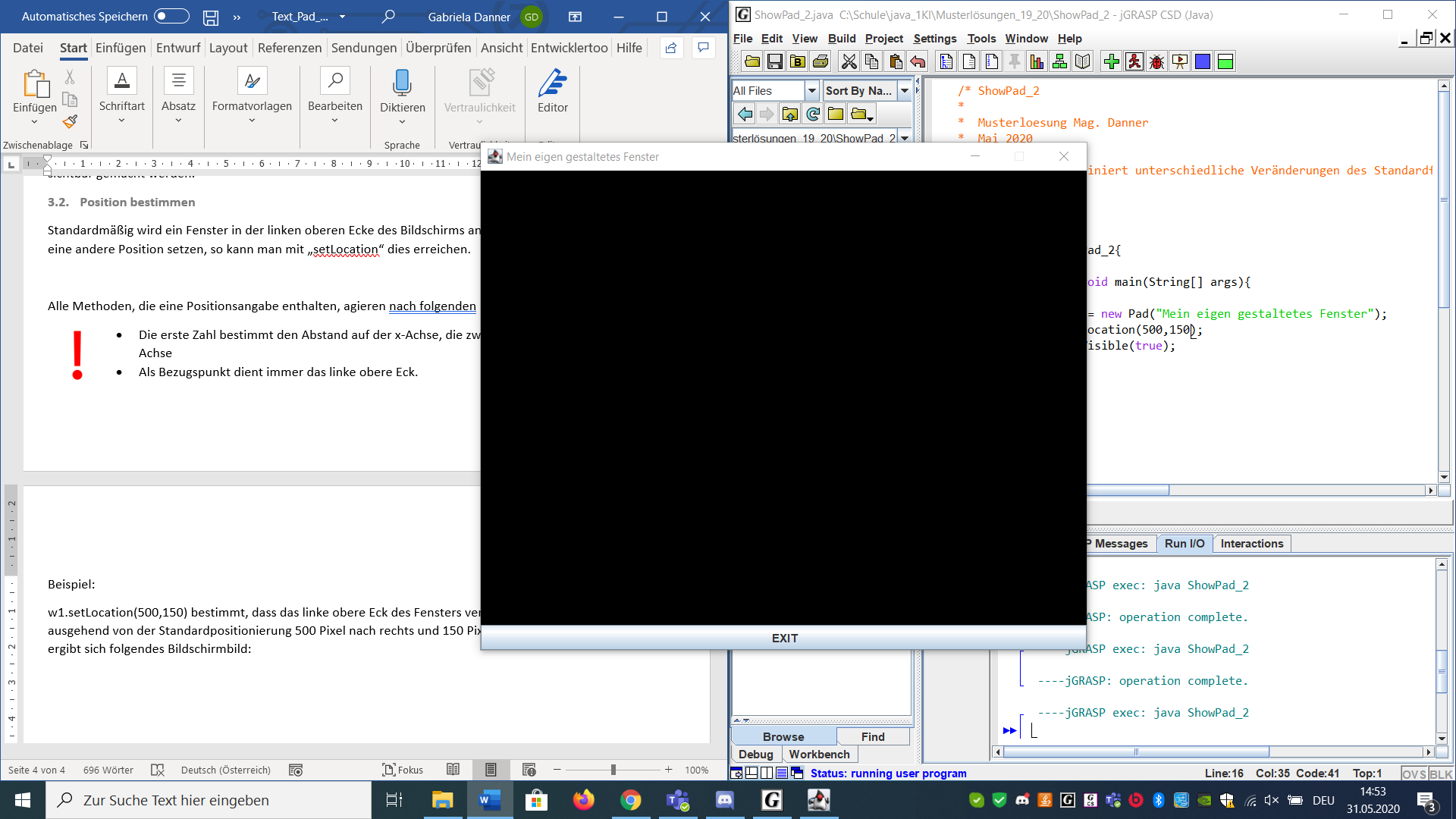
Standardmäßig wird ein Fenster in der linken oberen Ecke des Bildschirms angezeigt. Will man es an eine andere Position setzen, so kann man mit „setLocation“ dies erreichen.

Alle Methoden, die eine Positionsangabe enthalten, agieren nach folgenden Regeln:

* Die erste Zahl bestimmt den Abstand auf der x-Achse, die zweite jenen auf der y-Achse
* Als Bezugspunkt dient immer das linke obere Eck.

***Beispiel:***

w1.setLocation(500,150) bestimmt, dass das linke obere Eck des Fensters verschoben wird und zwar ausgehend von der Standardpositionierung 500 Pixel nach rechts und 150 Pixel nach unten. Dadurch ergibt sich folgendes Bildschirmbild:



Man erkennt gut, dass das Fenster jetzt in der Mitte des Bildschirms positioniert ist.

* 1. Größe bestimmen

Mit „setPadSize(int Breite, int Höhe) lässt sich die Größe eines Fensters bestimmen. Dabei sind alle Größen in Pixel angegeben.

* 1. Hintergrundfarbe

Die Methode „setBackground“ setzt die Hintergrundfarbe auf eine bestimmte Farbe. Dabei können alle in Pad definierten Standardfarben mit „Pad.“ angegeben werden.

* 1. Titel vergeben

Wird nicht schon beim Anlegen eines Fensters eine Bezeichnung (ein Titel) mit angegeben, so kann dieser nachträglich mit „setTitle(String title)“ gesetzt werden.

* 1. Informationen abfragen

Um später Elemente korrekt positionieren zu können muss man die Höhe und Breite des Fensters kennen. Diese Informationen können mit den Methoden „getAreaHeight“ und „getAreaWidth“ abgefragt werden. Der Rückgabewert ist in beiden Fällen vom Typ „int“ und gibt die Anzahl der Pixel an.

1. Zeichnen auf der Zeichenfläche (siehe ShowPad\_3)

Die folgenden Methoden zeigen, wie einzelne Elemente auf der Zeichenfläche gezeichnet werden können. Dabei gelten für alle Positionsangaben die in 3.2 erklärten Regeln.

* 1. Grundsätzliches

Viele der Zeichenmethoden stehen in mehreren Varianten zur Verfügung. Dabei gelten folgende Gesetzmäßigkeiten:

* + 1. „Draw“-Methoden

Methoden, die mit der Silbe draw beginnen, zeichnen von der darin beschriebenen Figur nur den Umfang.

***Beispiel:***

Die Methode drawRect liefert ein nicht ausgefülltes Rechteck wie z.B.

* + 1. „Fill“-Methoden

Methoden, die mit der Silbe fill beginnen, zeichnen die darin beschriebene Figur ausgefüllt.

***Beispiel:***

Die Methode fillCircle liefert einen ausgefüllten Kreis wie z.B.

* + 1. Positionsangaben mit zwei int-Werten

Praktisch alle Methoden brauchen eine Angabe wohin das Element gezeichnet werden soll. Dazu werden bei dieser Methode zwei int Werte übergeben, wobei der erste immer den x-Wert, der zweite den y-Wert beschreibt. Angegeben ist immer die linke obere Ecke, des Rechtecks, in die die Figur passen würde.

***Beispiel:***

Der Aufruf fillCircle(50,50,20) bedeutet, dass ein Kreis mit 20 Pixel Radius so gezeichnet werden soll, dass die mit „A“ bezeichnete Ecke auf der Position 50/50 zu stehen kommt.

A

* + 1. Positionsangaben mit „Point“

Für die Angabe der Position kann auch ein Element der Klasse „Point“ (auch eine Standardklasse von Java) verwendet werden. Ein solches Element kann ganz einfach mit new erzeugt werden. Als Parameter werden wieder ein x und ein y Wert übergeben. Der Vorteil dieser Methode ist, dass man den Punkt im Laufe der Programmierung öfter verwenden kann.

***Beispiel:***

Folgende Codeteile ergeben die gleiche Ausgabe:

w1.fillCircle(50,40,10); Point p1 = new Point(50,40);

w1.fillCircle(p1,10);

* 1. Farbgebung

Für alle Zeichenelemente gilt, dass sie in der Farbe gezeichnet werden, die gerade gesetzt ist. Will man die Farbe wechseln, so erfolgt dies über das Statement: w1.setColor(Pad……). Alle nachfolgenden Elemente werden solange in dieser Farbe gezeichnet, bis eine neue bestimmt wird.

* 1. Zeichenfläche leeren

Sollen alle Elemente, die gerade auf der Zeichenfläche vorhanden sind, gelöscht werden reicht eine einzige Anweisung. w1.clear() setzt das Fenster wieder in den Ausgangszustand zurück, d.h. nur mit der Hintergrundfarbe.

* 1. Methoden für das Zeichnen von Formen
     1. Rechtecke

Alle vorher beschriebenen Möglichkeiten gibt es zum Zeichnen von Rechtecken. Bei allen Aufrufen steht zuerst die Positionsangabe (x/y oder Point), dann die Breite des Rechtecks (Ausdehnung auf der X-Achse) und zum Schluss die Höhe des Rechtecks (Ausdehnung auf der Y-Achse.

***Beispiel:***

Die Angabe w1.fillRect(30,30,100,20) erzeugt also ein ausgefülltes Rechteck mit einer Breite von 100 Pixel und einer Höhe von 20 Pixel.

* + 1. Kreise

Der Zusatz „Circle“ erzeugt auf der Ausgabe einen Kreis. Auch diese Gruppe enthält alle vorher beschriebenen Kombinationen. Zusätzlich zu den Positionsangaben wird jeweils noch ein Radius angegeben.

* + 1. Ellipsen

Der Zusatz „Oval“ erzeugt die Ausgabe einer Ellipse. Dabei beschreibt der erste Parameter nach den Positionsangaben die Ausdehnung auf der X-Achse, der zweite die Ausdehnung auf der Y-Achse. Sind beide gleich wird wieder ein Kreis erzeugt.

* + 1. Striche

Natürlich können Striche nicht ausgefüllt sein. Es gibt sie daher nur in der Version „draw“. Sie zeichnen eine Linie von einem Punkt zu einem zweiten. Diese beiden können entweder durch x/y-Werte oder durch Point-Angaben beschrieben sein (Allerdings nur entweder - oder, nicht gemischt). Die Art der Verbindungslinie kann durch das Statement setStroke bestimmt werden. Zur Auswahl stehen dabei

* Pad.stroke: relativ dünner Standardstrich
* Pad.wideStroke: dicker Strich
* Pad.dashed: Unterbrochener Strich
  + 1. Polygone

Um Vielecke oder gebogene Linien zu erzeugen gibt es die Funktion „drawPolygon“. Als Polygon wird dabei eine Reihe von Punkten verstanden, die verbunden werden, wobei auch eine Verbindung zwischen dem letzten und ersten Punkt gezogen wird. Die Angabe der Punkte kann dabei wieder durch x/y-Werte oder durch Point erfolgen. Im ersten Fall sind zwei int-Arrays gleicher Länge notwendig, im zweiten Fall ein Point-Array.

Die Verbindung wird dabei in der zuvor gewählten Farbe und Strichstärke gezeichnet.

* + 1. Textausgabe

Über die Methode „drawString“ können Text auf der Ausgabe platziert werden. Dabei kann der auszugebende „Text“ den Typ String, long oder double haben.

Neben dem Text erhalten sie natürlich auch eine Positionsangabe:

Im Gegensatz zu allen anderen Methoden steht bei der Verwendung von x/y-Werten zur Positionsangabe in drawString **zuerst der Text und dann die Positionsangabe.** Verwendet man einen Point zur Positionsangabe, so wir wieder die gewohnte Reihenfolge von Position und restliche Parameter eingehalten.

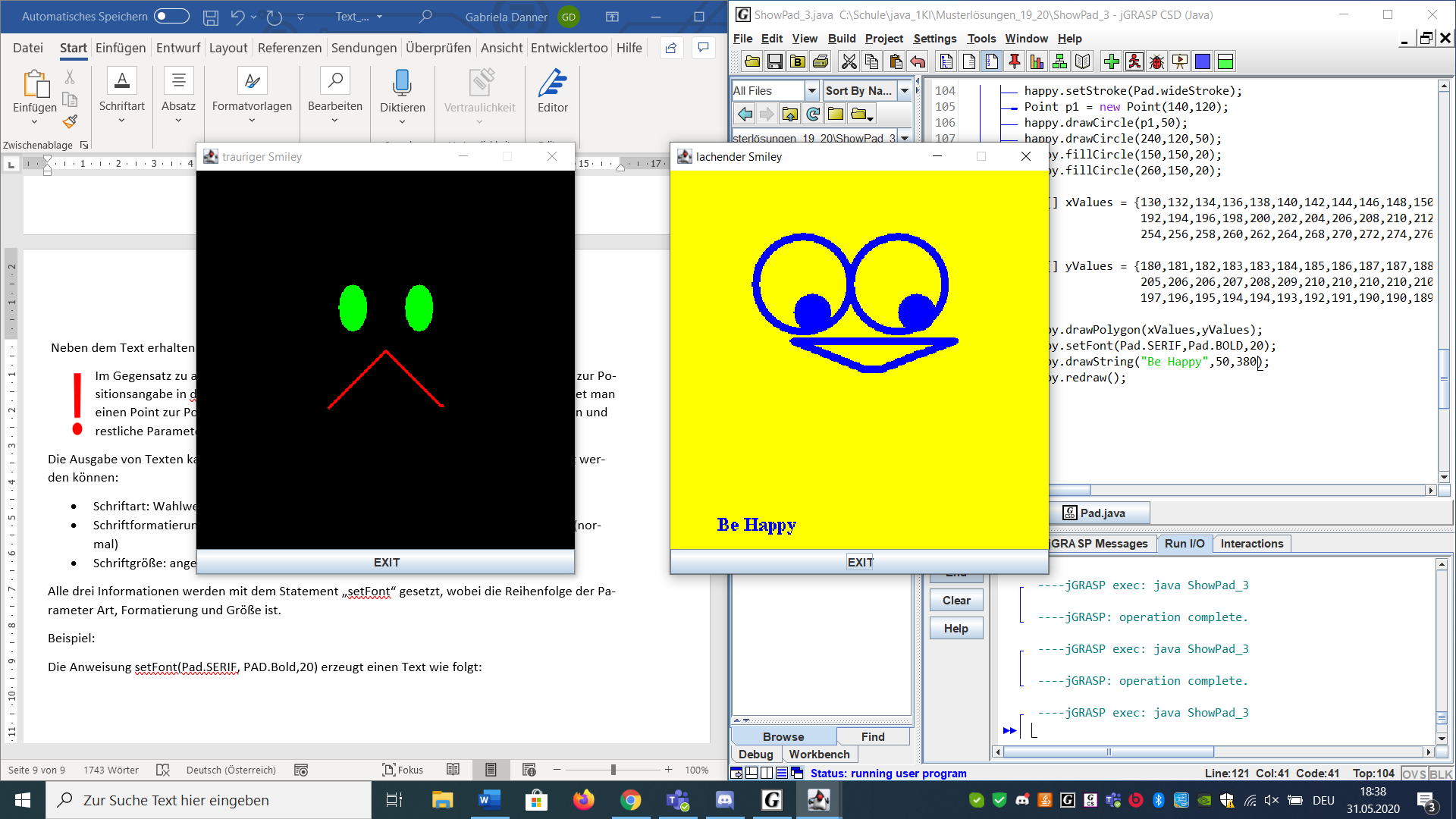
Die Ausgabe von Texten kann vor dem eigentlichen Aufruf noch formatiert werden. Bestimmt werden können:

* Schriftart: Wahlweise Pad.SANSSERIF, SERIF, FIXED oder DIALOG
* Schriftformatierung: Wahlweise PAD.BOLD (fett), PAD.ITALIC (kursiv) und PAD.PLAIN (normal)
* Schriftgröße: angegeben durch Pixel.

Alle drei Informationen werden mit dem Statement „setFont“ gesetzt, wobei die Reihenfolge der Parameter Art, Formatierung und Größe ist.

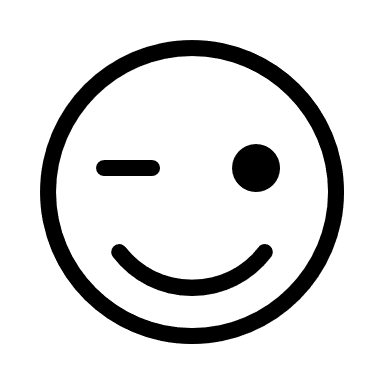
***Beispiel:***

Die Anweisung setFont(Pad.SERIF, PAD.Bold,20) erzeugt einen Text wie folgt:



* + 1. Zusätzliche Methoden

Das Pad-Modul enthält noch weitere Methoden, die hier nicht näher beschrieben werden. Sie sind aber in Source-Code ausreichend dokumentiert, um deren Funktion zu verstehen. Besonders gilt dies für einige „get“-Funktionen, die Auskunft über Abstände, Zeichengrößen etc. liefern.

** *Wie für alle grafischen Benutzeroberflächen gilt aber: Probieren geht über Studieren!! Einfach mal schauen, was die einzelnen Methoden bewirken erklärt sie am besten!*

* 1. Zeichenanweisungen sichtbar machen

Um die angegebenen Elemente auf dem Fenster auch sichtbar zu machen muss ein Neuzeichnen angestoßen werden. Dies geschieht mit der Anweisung w1.redraw()!!