Johannes Gutenberg-Universität Mainz
Institut für Informatik
Einführung in die Softwareentwicklung
Prof. Dr. Michael Wand
Dr. Christian Ali Mehmeti-Göpel

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Einführung in die Softwareentwicklung

ÜBUNGSBLATT 04: Vektorgraphik Teil 2 – OOP

26. Mai 2025

Über dieses Übungsblatt

Auf diesem Übungsblatt programmieren wir weiter an dem vektor-orientierten Zeichenprogramm, das wir auf Blatt 03 begonnen haben. Der Fokus liegt diesmal darauf, objektorientierte Entwurfsprinzipien umzusetzen, sowie auch weitere Features einzubauen.

Außerdem gibt es zum Einstieg eine kleine Übung zu OOP in SCALA, damit wir das nicht völlig aus dem Auge verlieren.

Auch hier wieder die gleichen Tipps wie letztes Mal:

- Nutzen Sie die Gelegenheit, Erfahrungen mit Softwareentwurf zu machen (nicht nur Punkte sammeln)
- Lesen Sie alle Teilaufgaben (hier insbesondere Aufgabe 2 und 3) durch, bevor Sie mit der Bearbeitung beginnen – damit Sie sich eine gute Architektur überlegen können!

Abgabe: (Abgabe 9. SW)

Für dieses Übungsblatt sind wieder zwei Wochen Bearbeitungszeit vorgesehen. Das Übungsblatt muss entsprechend bis zum **8. Juni 2025, 23:59h** in LMS abgegeben werden. Laden Sie dazu den Quellcode aller Aufgaben hoch. Die Ergebnisse müssen in den Übungen zwischen dem 9.-13. Juni 2025 vorgestellt werden.

Aufgabe 0: Design (unbewertet)

Lesen Sie sich auch dieses Mal zuerst alle Aufgaben durch, und überlegen Sie sich, wie Sie Ihre Programme strukturieren möchten. Wie kann man dafür sorgen, dass es leicht erweiterbar bleibt?

Dieses Mal ist in Aufgabe 1 etwas anderes zu tun (kleine Übung für OOP & Scala) als in Aufgabe 2 und 3 (Fortsetzung vom letzten Blatt). Wichtig ist besonders, das Vorgehen für Aufgabe 2 und 3 gut zu durchdenken. Aufgabe 1 wird auf den folgenden Übungsblättern nicht weiter ausgebaut, das Zeichenprogramm aus Aufgabe 2 und 3 aber schon.

Aufgabe 1: OOP in Python und Scala (20+10 = 30 Punkte)

Zur Vorbereitung wollen wir die in der Vorlesung besprochenen OOP-Konzepte in SCALA und Python einmal ausprobieren. Dazu gibt es eine relativ einfache Aufgabe für einen Softwareentwurf.

Sie sollen eine Klasse programmieren, die ein Datum speichert. Ein Datum besteht immer aus drei Bestandteilen:

- Tag (`int`)
- Monat (`int`)
- Jahr (`int`)

Implementieren Sie eine Oberklasse „`Datum`“, die (prinzipiell) diese drei Komponenten speichern kann. Eine Prüfung der Parameter ist nicht nötig (der Benutzer wird schon keinen 42. Dezember des Jahres -32786 anlegen). Die Klasse sollte abstrakt sein, und zwei abstrakte Methoden vorsehen, die in Unterklassen implementiert werden:

- `print_date()` – gibt das Datum ausführlich aus (z.B. 24. Dezember 2024)
- `print_date_short()` – gibt das Datum kurz aus (z.B. 24.12.24)

Leiten Sie nun zwei konkrete Klassen „`DatumUS`“ und „`DatumDE`“ ab, die diese beiden Methoden einmal nach US-amerikanischer Konvention (12/24/24 bzw. December 24 2024) und im anderen Fall nach deutscher Konvention (24.12.24 bzw. 24. Dezember 2024) ausgibt.

Implementieren Sie dies einmal in Scala (20 Punkte) und einmal in Python (10 Punkte). Testen Sie Ihren Code mit einem kurzen Testprogramm, dass alle vier Varianten für ein Beispiel auf der Konsole ausgibt.

Details (z.B. dataclasses vs. explizites __init__ in Python, case-classes oder nicht in Scala, wo werden die Daten konkret gespeichert – in Unterklassen oder der Oberklasse?) können Sie frei festlegen.

Anmerkung: Das ist eine Übungsaufgabe, das vorgegebene OO-Design ist nicht grandios :-)

Aufgabe 2: Vektorgraphik mit OOP (30 Punkte)

Nutzen Sie nun Ihre Lösung von Übungsblatt 03*) und überarbeiten Sie diese so, dass die verschiedenen Zeichenprimitive des Vektorgraphikprogramms durch eine gemeinsame abstrakte Oberklasse `Shape` sowie davon abgeleitete konkrete Klassen `Rectangle`, `Circle`, `Star` (die Namen der Klassen dürfen gerne von den Vorgaben abweichen) beschrieben werden.

Die abstrakte Oberklasse sollte mindestens eine abstrakte Methode für das Zeichnen der Primitive (Sterne, Kreise, Rechtecke) mit Hilfe eines `QPainter` (oder der Entsprechung in AWT/Swing) bereitstellen, die dann in den Unterklassen überschrieben (und ausgearbeitet) wird.

Beim Zeichnen von den drei geometrischen Formen sollen folgende Eigenschaften einstellbar sein:

- Linienstärke des Randes
- Farbe des Randes
- Farbe der Füllung

Beim Zeichnen soll dies berücksichtigt werden (also die richtigen Farben und Randstärke erscheinen auf dem Bild).

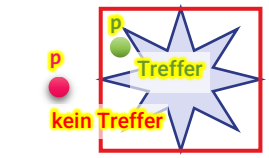
Sie sollten sich vielleicht auch schon überlegen, wie man die Funktionalität, die in Aufgabe 3 gefordert wird, sinnvoll in Ihren Entwurf integrieren kann.

Achtung: Siehe Hinweise unten (nach Aufgabe 3).

Aufgabe 3: Interaktion mit Vektorgraphik (30+10 = 40 Punkte)

Nutzen Sie die gleiche Architektur wie in Aufgabe 2 (abstrakte Oberklasse `Shape` o.ä. mit sinnvoller abstrakter Schnittstelle für alle Zeichenobjekte). Fügen Sie nun weitere abstrakte Methoden hinzu, wie auch konkrete Implementationen in den drei Unterklassen, um die folgende Funktionalität umzusetzen:

- **„Picking" (Treffer prüfen):** Gegeben ein Punkt `p` in Weltkoordinaten soll geprüft werden, ob dieser Punkt `p` innerhalb der Form liegt. Für eine „korrekte" Lösung reicht eine einfache Variante der Prüfung aus, der „Axis-Aligned-Bounding-Box-Test": Man bestimmt die „Axis-aligned Bounding-Box", gegeben durch den kleinsten und größten x-Wert, in dem das Primitiv eng eingeschlossen ist, sowie den kleinsten und größten y-Wert. Wird das so aufgespannte achsenparallele Rechteck vom Punkt `p` getroffen, so gilt das Objekt als getroffen; anderenfalls als verfehlt. Es soll für jede der (bislang drei) Zeichenformen möglich sein, zu prüfen, ob die Form getroffen wurde. Eine Implementation die genau prüft, ob die Form getroffen wurde, ist natürlich auch zulässig (aber bei zwei der drei Primitive etwas komplizierter zu realisieren).



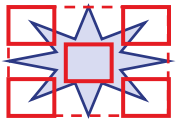
Beispiel: einfacher Bounding-Box Test für „Picking"

- **Geometrische Transformation (Nicht-uniforme Skalierung und Verschiebung):** Es soll möglich sein, Position, Breite und Höhe jedes Zeichenobjektes einzustellen. Wird dies auf

Kreise angewandt, so werden diese zu Ellipsen verzerrt. Bei Sternen können Sie sich aussuchen, ob diese nur entlang eines Ausmaßes (z.B. Breite, Höhe, Diagonale) vergrößert werden, oder ob sie auch verzerrt werden. Das „Picking" (Überprüfung der Treffer) muss aber auch dann richtig funktionieren.

a) Entwerfen Sie eine geeignete Schnittstelle und implementieren Sie die Operationen. Falls Sie Aufgabe b) nicht lösen, sollten Sie Beispiel/Testcode schreiben, der klar zeigt, dass Ihre Implementation funktioniert.

b) Für volle Punktzahl soll es möglich sein, mit der Maus Objekte auszuwählen und zu verschieben und zu skalieren. Zum Beispiel: Einmal anklicken aktiviert das Objekt und die Bounding-Box wird angezeigt, danach kann man an der Box ziehen (linker oberer Teil vergrößert nach links oben, rechter unterer nach rechts unten, etc.; Mitte verschiebt). Wie genau die Maussteuerung umgesetzt und initiiert wird, können Sie sich frei überlegen (z.B. Transformations-Modus mit einem Menüpunkt oder einem Knopf auf der Toolbar; oder Tastarturkürzel, oder immer verfügbar?). Wirklich gelungen ist der Entwurf, wenn er gut mit der Implementation von Aufgabe 04 vom letzten Übungsblatt zusammenarbeitet.



Beispiel: Transformation mit 5 Bereichen zum Anfassen mit der Maus

Die Aufgabe 3b ist recht schwer, daher gibt es nur 10% der Punkte (aber es ist eine echte Herausforderung für einen guten Entwurf, daher zur Übung unbedingt empfohlen).

Achtung: Siehe Hinweise unten.

***) Hinweise zu Aufgaben 2, 3:**

- Die Aufgaben sollten am besten Ihr Programm und Ihren Entwurf vom letzten Übungsblatt weiterentwickeln. Sie können aber auf Wunsch gerne die Musterlösung von Übungsblatt 03 nutzen, und diese weiterentwickeln (es kann ja sein, dass Blatt 3 einfach nicht geklappt hat). Die Musterlösung erscheint mit diesem Aufgabenblatt auf LMS.
- Falls Ihre Lösung von Blatt 3 bereits wie in Anforderungen von Aufgabe 2 (ggf. auch Aufgabe 3, wobei diese in Blatt 3 nicht gefordert war) in Teilen oder insgesamt erfüllt, dann reicht es aus, in der Übungsabgabe lediglich nochmal zu erklären, welches Konzept Sie bei Ihrer früheren Implementation verfolgt haben.