

Eigene Klassen und Objekte in Processing (Java)

Didaktische Hinweise

Zielgruppe

Die Materialien richten sich an Schüler*innen in der Qualifikationsphase. Der erste Teil bis einschließlich Aufgabe 7 eignet sich sowohl für Kurse auf grundlegendem als auch auf erhöhtem Anforderungsniveau. Der zweite Teil zu den Klassendiagrammen und weiteren Konzepten der objektorientierten Programmierung (OOP) fördert Kompetenzen, die in Niedersachsen nur für Kurse auf erhöhtem Anforderungsniveau verpflichtend sind (vgl. [2]).

Voraussetzungen

Es wird davon ausgegangen, dass die Lernenden mit dem algorithmischen Problemlösen in Processing¹ unter Verwendung von algorithmischen Grundbausteinen, eigenen Methoden, Variablen sowie Reihungen vertraut sind. Außerdem ist es hilfreich, wenn sie bereits eine Bibliothek zum Erstellen von Benutzeroberflächen (GUI) verwendet haben. In den Beispielen kommt die Bibliothek *G4P* von Peter Lager mit dem entsprechenden Tool *G4P GUI Builder*² zum Einsatz. Ggf. kann die Einführung von Benutzeroberflächen aber auch anhand des Einstiegsbeispiels thematisiert werden. Bei der Betrachtung weiterer Konzepte der OOP wie z. B. der Vererbung werden die Zeichenfunktionen von Processing genutzt, da sich Beispiele aus dem Bereich Grafik gut eignen, um die Konzepte praktisch umzusetzen.

Lernziele

Anhand des vorliegenden Leitfadens können sich die Schüler*innen Einsatzmöglichkeiten eigener Klassen und Objekte sowie den Umgang damit erarbeiten. Dabei geht es sowohl um die Verwendung einer gegebenen Klasse als auch um den Entwurf und die Implementierung eigener Klassen. Da in diesem Zusammenhang verschiedene Kompetenzen aus dem Bereich „Grundlagen der Algorithmik“ benötigt werden, werden diese ebenfalls vertieft und weiter gefestigt.

Im Fokus stehen die folgenden Kompetenzen für die Qualifikationsphase aus dem Lernfeld *Algorithmisches Problemlösen* des niedersächsischen Kerncurriculums für die Sekundarstufe II (s. [2]):

Die Schülerinnen und Schüler

- unterscheiden zwischen primitiven Datentypen und Objektreferenzen.
- entwerfen und implementieren Algorithmen unter Verwendung von gegebenen und eigenen Klassen/Objekten.
- entwerfen Klassen und deren Beziehungen (Assoziation, Vererbung) und stellen diese durch Klassendiagramme dar. (Erweiterung eA)

Zu beachten ist, dass sich die Materialien zwar am niedersächsischen Kerncurriculum für die gymnasiale Oberstufe orientieren, jedoch keinen Anspruch auf Vollständigkeit hinsichtlich der für die Abiturprüfung erwarteten Kompetenzen erheben. Darstellungen und Schreibweisen orientieren sich an

¹ Die Programmierumgebung Processing wurde 2001 von Ben Fry und Casey Reas initiiert. Nähere Informationen finden Sie unter <https://processing.org/>

² Peter Lager. GUI Builder Tool. <http://www.lagers.org.uk/g4ptool/index.html> [Datum des Zugriffs: 07.05.2025]

den ergänzenden Hinweisen zum Kerncurriculum Informatik für die Sek II (s. [3]). Sie können aber ggf. von den in der Abiturprüfung verwendeten Darstellungen und Schreibweisen abweichen. Verbindlich für das Abitur in Niedersachsen sind allein das niedersächsische Kerncurriculum für die gymnasiale Oberstufe (s. [2]) sowie die ergänzenden Hinweise (s. [3]) in der jeweils aktuellen Fassung. Es obliegt daher den jeweiligen Fachlehrer*innen, den Unterricht so zu gestalten, dass die Schüler*innen umfassend auf das Abitur vorbereitet werden. Die vorliegenden Materialien stellen somit nur eine Anregung dar, die an die individuellen Bedürfnisse der Lerngruppe angepasst werden müssen.

Hinweise zu einzelnen Aspekten und Aufgaben

Einstiegsaufgabe

Als Einstiegsaufgabe bietet sich eine Problemstellung an, bei der mehrere Objekte der gleichen Art verwaltet werden müssen. Möglicherweise wurden im Zusammenhang mit ein- und zweidimensionalen Reihungen bereits entsprechende Probleme betrachtet. So ist ein Vokabeltrainer beispielsweise auch als Übungsaufgabe im Materialpaket zu den ein- und zweidimensionalen Reihungen enthalten. Sollen zu einem Objekt Eigenschaften mit verschiedenen Datentypen gespeichert werden, sind verschiedenen Reihungen notwendig und die Verknüpfung bzw. Zuordnung zu einem Objekt kann nur über gleiche Indizes hergestellt werden. Hieraus ergibt sich eine Motivation für eigene Klassen, da diese eine komfortablere und zuverlässigere Lösung des Problems bieten.

Erkunden des Beispielsprogramms

In Aufgabe 1 und 2 setzen sich die Schüler*innen mit dem Beispielsprogramm Vokabeltrainer auseinander. Die vorhergehenden Erläuterungen können genutzt werden, um die entsprechenden Anpassungen und Ergänzungen umzusetzen. Da der Entwurf eigener Klassen in der Regel erst dann sinnvoll erscheint, wenn viele gleichartige Objekte benötigt werden, erfolgt die Verwaltung der Variablen hier in einer Reihung. Dadurch kommt zu dem neuen Konzept des Erzeugens und Verwendens von Objekten jedoch der Umgang mit der Reihung hinzu. Es kann daher sinnvoll sein, zunächst das Erzeugen einzelner Objektvariablen zu üben, wie beispielsweise in Aufgabe 2a)[2] oder indem das Erzeugen des Objekts als einzelne Objektvariable und das Speichern in der Reihung in einzelne Schritte zerlegt werden.

Sichtbarkeit von Attributen und Operationen

Da das Prinzip der Kapselung ein zentrales Konzept der OOP ist (vgl. z. B. [4]), wird in dem vorliegenden Leitfaden konsequent zwischen privaten und öffentlichen Attributen unterschieden und von außen nur über `get`- und `set`-Methoden auf Attribute einer Klasse zugegriffen. Fällt Schüler*innen in Kursen auf grundlegendem Anforderungsniveau der direkte Zugriff auf die Attribute leichter, sind die Problemstellungen in der Regel hinreichend überschaubar, um von diesem Vorgehen abzuweichen und mit öffentlichen Attributen zu arbeiten.

Bei der Arbeit mit Processing ist zu beachten, dass eigene Klassen als innere Klassen des Hauptprogramms erzeugt werden. Das führt dazu, dass Attribute, die in den eigenen Klassen als privat deklariert sind, von der Hauptklasse aus trotzdem direkt zugänglich sind, so als wären sie öffentlich. Bei Bedarf wäre entsprechend zu thematisieren, warum es trotzdem sinnvoll ist, das Prinzip der Kapselung zu verwenden.

Objekte als Referenzvariablen

Im Umgang mit Objekten ist es wichtig, sich den Unterschied zu primitiven Datentypen bewusst zu machen. Ansonsten kann es beispielsweise beim Vergleichen und Kopieren von Objekten zu unerwünschten Effekten kommen. In diesem Zusammenhang ist auch zu beachten, dass Java bei der Übergabe einer Variablen als Parameter an eine Methode nach dem Prinzip „call by value“ arbeitet. Das heißt, dass eine Kopie des Inhalts der Variablen übergeben wird. Im Falle von Objekten wäre das also die Referenz auf das Objekt im Speicher. Veränderungen an den Attributen des Objektes wirken sich daher sowohl lokal auf den Parameter als auch auf die ursprüngliche Variable außerhalb der Methode aus, da beide auf dasselbe Objekt zeigen. Wird ein neues Objekt erzeugt und dem Parameter zugewiesen, ändert sich hingegen nichts an der ursprünglichen Variablen außerhalb der Methode, da diese weiterhin auf das alte Objekt zeigt. Nähere Erläuterungen findet man z. B. unter [1].

Weitere Konzepte der objektorientierten Programmierung

Die Erarbeitung weiterer Konzepte der objektorientierten Programmierung erfolgt am Beispiel von Klassen für gezeichnete Figuren. Beim Zeichnen von geometrischen Figuren entstehen Pixelgrafiken, sodass eine nachträgliche Veränderung nicht möglich ist. Hier bietet sich daher das Erstellen entsprechender Klassen für die gezeichneten Objekte an. Anhand der Beispiele lässt sich auch der Unterschied zwischen Pixelgrafiken und einer geometrischen Beschreibung von Bildern, wie sie bei Vektorgrafiken verwendet wird, thematisieren.

OOP-Konzepte wie die Vererbung sind für Software-Projekte in der Schule häufig nicht notwendig, da sie nicht hinreichend komplex sind. Geometrische Figuren mit ihren Gemeinsamkeiten und Unterschieden bieten hier ein Anwendungsfeld, in dem Vererbung und Polymorphie sinnvoll erscheinen. In diesem Zusammenhang werden auch Problemstellungen aus dem Materialpaket „Reihungen in Processing und Java“ wieder aufgegriffen. Sie können aber auch unabhängig davon bearbeitet werden.

Lösungen

Die Implementierungen im Lösungsordner stellen lediglich eine mögliche Lösung dar und erheben nicht den Anspruch besonders effizient oder dergleichen zu sein.

Ausblick

Eine genauere Betrachtung des Einstiegsbeispiels eines Vokabeltrainers zeigt, dass eine Reihung fester Größe zur Verwaltung der Vokabel-Objekte nicht optimal ist. Zunächst müssen leere Speicherplätze vorgehalten werden und wenn die Reihung gefüllt ist, können keine weiteren Objekte hinzugefügt werden. Zudem muss der nächste freie Index in einer Variablen gespeichert werden.

Als Verbesserung können hier daher eigene Klassen, die spezielle Datenstrukturen implementieren, eingeführt werden. Eine dynamische Reihung löst das Problem der flexiblen Anzahl an Vokabeln. Aber auch eine Schlange bietet sich als Datenstruktur an. So können die Vokabeln beispielsweise in zwei Schlangen verwaltet werden. Zunächst befinden sich alle Vokabeln in einer Schlange. Es wird jeweils das vorderste Element entnommen und abgefragt. Wird die Vokabel richtig übersetzt, wird sie an die zweite Schlange angehängt. Wird sie falsch übersetzt, wird sie wieder hinten an die erste Schlange angefügt.

Zu den für die Qualifikationsphase vorgegebenen Datenstrukturen wird ebenfalls ein Materialpaket zur Verfügung gestellt.

Literatur

- [1] Czeschla, J. (2023). javabeginners. Grundlagen – Call by Value.
https://javabeginners.de/Grundlagen/Call_by_Value.php [Datum des Zugriffs: 12.04.2023]
- [2] Niedersächsisches Kultusministerium (Hrsg.) (2017) Kerncurriculum für das Gymnasium - gymnasiale Oberstufe, die Gesamtschule – gymnasiale Oberstufe, das Kolleg. Informatik. Hannover: unidruck
- [3] Niedersächsisches Kultusministerium (Hrsg.) (2018) *Ergänzende Hinweise zum Kerncurriculum Informatik für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie für das Kolleg.* <https://cuvo.nibis.de/cuvo.php?p=download&upload=174> [Datum des Zugriffs: 01.09.2020]
- [4] G. Saake & K.-U. Sattler (2006). Algorithmen und Datenstrukturen. Eine Einführung in Java. (3. überarb. Aufl.) Heidelberg: dpunkt

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International Lizenz](#).

Für die korrekte Ausführbarkeit der beiliegenden Quelltexte wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.