Graphen von Polynomen

Sven Eric Panitz

3. Dezember 2022

Inhaltsverzeichnis

1 Einleitung

In dieser Übungsaufgabe soll eine Applikation entwickelt werden, die Polynome als Graphen auf dem Bildschirm darstellt.

Für einen Teil der Aufgaben gibt es Tests, die Subato durchführt. Für den graphischen Teil sind Bilddateien zu erzeugen, die visuell als Tests dienen können.

Im Zuge dieses Übungsblatts werden Sie ein paar Klassen und Methoden aus dem Swing API kennenlernen, sowie auch das Konzept der abstrakten Klassen und der inneren Klassen.

Laden Sie sich die Vorlage für die Lösung in einen Editor. Öffnen Sie parallel eine Session der JShell. In dieser Session können Sie, nachdem Sie Änderungen an der Lösungsdatei vorgenommen haben, diese neu in die JShell laden und dort erste Testaufrufe der neu umgesetzten Funktionalitäten machen. Hierzu nutzen Sie das Kommando `/open` der JShell.

Shell

```
Polynom$ jshell
|  Welcome to JShell -- Version 19.0.1
|  For an introduction type: /help intro

jshell> /open solution/Poly.java

jshell>
```

Die Applikation verwendet viele Klassen aus anderen Paketen von Java. Das sind insbesondere die GUI-Bibliothek Swing und Pakete zur Erzeugung von Bilddateien. Diese werden somit in dem Programm importiert.

Java: Poly

```
import javax.swing.*;
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.ImageIO;
```

Die Applikation, die in dieser Übung entsteht, besteht aus mehreren Klassen. Diese befinden sich alle innerhalb einer Schnittstelle. Damit haben wir die komplette Appliakation in einer Datei. Dieses ist in Java eher unüblich, macht aber eine Abgabe in Subato als eine Datei möglich und erleichtert es, mit einer Quelltextdatei schneller in der JShell Tests durchzuführen.

Java: Poly

```
public interface Poly{
```

2 Eine abstrakte Klasse zum Zeichnen

In diesem Abschnitt wird eine abstrakte Klasse entwickelt, die es erlaubt, Graphiken im 2-dimensionalen Raum zu programmieren, diese in einem Fenster anzuzeigen und sie auch als png-Datei ins Dateisystem zu schreiben.

Die Klasse `PaintBoard` hat zwei Felder, um die Größe der Graphik zu bezeichnen. Beide Felder werden in einem Konstruktor initialisiert.

Da es sich um eine innere Klasse der Schnittstelle `Poly` handelt, ist der komplette Name dieser Klasse: `Poly.PaintBoard`.

Java: Poly

```
abstract class PaintBoard extends JPanel {
  int width;
  int height;
  PaintBoard(int width, int height){
    this.width= width;
    this.height=height;
  }
```

Neben dem Konstruktor ist zur Verwendung dieser Klasse die folgende abstrakte Methode `paintWith` entscheidend. Abstrakt bedeutet, dass die Methode noch gar keine Implementierung hat. Sie hat nur eine Signatur. Es gibt keinen Rumpf, der in geschweiften Klammern eingeschlossen ist. Deshalb musste auch die Klasse selbst mit dem Attribut `abstract` versehen werden. Das hat zur Folge, dass man kein Objekt dieser Klasse mit `new` erzeugen kann. Wäre dieses möglich, dann erzeugte man ein Objekt, das noch keine Implementierung der Methode `paintWith` hat. Dieses Objekt müsste passen, wenn die Methode aufgerufen wird. Damit das nicht passieren kann, ist es nicht möglich, Objekte einer abstrakten Klasse zu erzeugen.

Wozu ist dann eine abstrakte Klasse gut? Der Trick ist, dass Unterklassen einer abstrakten Klasse definiert werden können. Wenn diese die abstrakten Methoden überschreiben, haben diese Implementierungen aller versprochenen Methoden. Von diesen Unterklassen können dann auch Objekte erzeugt werden. Daher benötigt die abstrakte Klasse auch einen Konstruktor, damit man im Konstruktor der Unterklasse mit `super()` auch den entsprechenden Konstruktor der Oberklasse aufrufen kann.

Betrachten wir also die Signatur der abstrakten Methode:

Java: Poly

```
abstract void paintWith(Graphics2D g);
```

Es ist eine `void`-Methode, die ein Objekt der Standard-Klasse `java.awt.Graphics2D` als Argument erhält. Wer die Dokumentation der Klasse `Graphics2D` durchsieht, findet dort eine Reihe von Methoden zum Erzeugen graphischer Elemente auf diesem Objekt, wie dem Zeichnen von Linien, Vierecken, Ovalen aber auch von Texten.

Die Methode `paintWith` werden wir in Unterklassen überschreiben und darin definieren, wie unsere Graphik aussehen soll.

2.1 Anzeige der Graphik auf dem Bildschirm

Damit wir unsere Graphik auf dem Bildschirm anzeigen können, definieren wir eine Funktion `showMe()`, die ein Fenster öffnet, dessen Inhalt die definierte Graphik ist.

Java: Poly

```
public void showMe(){
  var f = new JFrame();
  f.add(this);
  f.setResizable(false);
  f.pack();
  f.setVisible(true);
}
```

Das Fenster wird mit einer fixen Fenstergröße geöffnet und zeigt die in `paintWith` definierte Graphik an.

2.2 Speichern der Graphik in eine Datei

Als zweite sinnvolle Funktion für den Anwender wird eine Funktion bereitgestellt, die die Graphik auch in eine Datei speichern kann. Als Dateiformat wird PNG verwendet.

Java: Poly

```
public void saveToPNG(String fileName){
  var img
   = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
  var g2d = img.createGraphics();
  paintComponent(g2d);
  g2d.dispose();
  var file = new File(fileName);
  try{ImageIO.write(img, "png", file);}catch(Exception e){}
}
```

2.3 Dimension der Graphik

Damit die beiden vorhergehenden Methoden korrekt funktionieren ist die Funktion, die die optimale Größe der Graphik zurückgibt, mit unseren Dimensionen zu überschreiben:

Java: Poly

```
@Override
public Dimension getPreferredSize(){
  return new Dimension(width,height);
}
```

2.4 Zeichnen der Graphik

Damit auch wirklich unsere in `paintWith` definierte Graphik angezeigt wird, ist die Methode `paintComponent` zu überschreiben.

In dieser wird zunächst die Hintergrundfarbe auf weiß gesetzt, dann durch den Aufruf von `super.paintComponent(g)` die gesamte Fläche mit dem Hintergrund ausgefüllt, auf dem dann mit der spezifischen Methode `paintWith` die Graphik gezeichnet wird.

Java: Poly

```
  @Override
  protected void paintComponent(Graphics g){
    setBackground(Color.WHITE);
    super.paintComponent(g);
    paintWith((Graphics2D)g);
  }
}
```

2.5 Beispielgraphik

Wir haben eine abstrakte Klasse zum Definieren von Graphiken. Es ist an der Zeit, diese in einem Beispiel zu verwenden. Hierzu ist eine Unterklasse zu definieren.

Java: Poly

```
class PaintABit extends PaintBoard{
```

Diese Klasse braucht einen Konstruktor, der mit `super` den Konstruktor der Oberklasse aufruft. Für unser Beispiel nehmen wir einfach eine konstante Größe von 400 mal 400 Pixel als Größe an.

Java: Poly

```
PaintABit(){
  super(400,400);
}
```

Nun gilt es die abstrakte Methode `paintWith` mit unserer eigenen Graphik zu überschreiben. Wir setzen beispielsweise jeweils für die eizelnen Figuren die Farbe und zeichnen Ovale, eine Linie und einen Text.

Java: Poly

```
  @Override
  void paintWith(Graphics2D g){
    g.setColor(Color.RED);
    g.fillOval(100,100,50,60);
    g.setColor(Color.BLUE);
    g.fillOval(200,150,70,30);
    g.setColor(Color.GREEN);
    g.drawLine(0,0,width,height/2);
    g.setColor(Color.BLACK);
    g.setFont(new Font("Helvetica", Font.BOLD, 25));
    g.drawString("Hello World!",100,300);
  }
}
```

Jetzt haben wir mit `PaintABit` eine Klasse, von der ein Objekt erzeugt werden kann und das dann auf dem Bildschirm angezeigt werden und in eine Datei gespeichert werden kann.

Shell

```
jshell> /open solution/Poly.java

jshell> var bild = new Poly.PaintABit()

jshell> bild.showMe()

jshell> bild.saveToPNG("bild1.png")
```

Probieren Sie es selbst mit den obigen Aufrufen auf der JShell.

Sie müssten das Bild aus Abbildunge 1 sehen können.

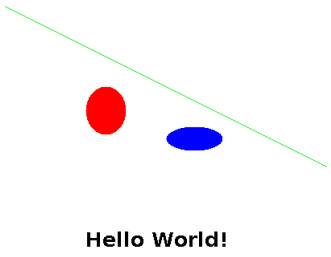

Abbildung 1: Ausgabe eines PaintABit Objektes.

Aufgabe 1 Machen Sie eine eigene Unterklasse von `PaintBoard`. Machen Sie darin eine eigene Graphik, in der unter Anderem auch Ihr Name als String angezeigt wird. Speichern Sie diese Graphik in eine Datei.

3 Polynomverarbeitung

3.1 Monome

Ein Monom ist ein Term über eine freie Variable x der folgenden Form: $b * x^e$. Dabei ist b der Koeffizient und e der Exponent.

Hierfür lässt sich direkt eine Datenklasse definieren, die Koeffizient und Exponent als Felder hat:

Java: Poly

```
record Monom(double b, int e){
```

Aufgabe 2 Überschreiben Sie für die Klasse `Monom` die Methode `toString`, sodass der Term in möglichst einfacher Form gelesen werden kann.

Aufgabe 3 Ergänzen Sie die Klasse Monom um die Methode, die für ein Argument x den Wert des Monoms errechnet.

Java: Poly

```
double eval(double x){
  return 0; /*TODO*/
}
```

Aufgabe 4 Ergänzen Sie die Methode, die das Monom, das die erste Ableitung des Monoms ist, errechnet.

Java: Poly

```
Monom derivation(){
  return null; /*TODO*/
}
```

Soweit die Klasse der Monome.

Java: Poly

```
}
```

Wir können uns nun im interaktiven Modus schnell drei Monome erzeugen und ein paar Methoden für diese aufrufen:

Shell

```
jshell> var m1 = new Poly.Monom(2,1)
m1 ==> 2.0*x^1

jshell> var m2 = new Poly.Monom(0.3,2)
m2 ==> 0.3*x^2

jshell> var m3 = new Poly.Monom(-0.1,3)
m3 ==> -0.1*x^3

jshell> m3.eval(2)
$10 ==> -0.8

jshell> m3.derivation()
$11 ==> -0.30000000000000004*x^2
```

3.2 Polynome

Ein Polynom ist die Summe einer endlichen Anzahl von Monomen. Auch hierfür definieren wir eine Datenklasse.

Java: Poly

```
record Polynom(Monom... ms){
```

Dabei verwenden wir die Notation für eine variable Parameteranzahl. Bei dieser werden nach dem Typnamen drei Punkte gesetzt. Damit wird angezeigt dass die Methode (das ist in diesem Fall der kanonischen Konstruktor) mit einer beliebigen Anzahl von Parametern diesen Typs aufgerufen werden kann. Das in diesem Fall definierte Feld `ms` ist dann vom Typ `Monom[]` also eine Reihung.

Mit dem so definierten Konstruktor können wir also zum Beispiel ein Polynom aus drei Monomen erzeugen:

Shell

```
jshell> var pol = new Poly.Polynom(m1,m2,m3)
pol ==> Polynom[ms=[LPoly$Monom;@4b553d26]
```

Aufgabe 5 Überschreiben Sie für die Klasse `Polynom` die Methode `toString`, sodass der Term in möglichst einfach gelesen werden kann.

Aufgabe 6 Schreiben Sie jetzt die Methode, die für ein Argument x den Wert des Polynoms berechnet.

Java: Poly

```
double eval(double x){
  return 0.0; /*TODO*/
}
```

Aufgabe 7 Schreiben Sie die Methode, die für ein Polynom die erste Ableitung errechnet.

Java: Poly

```
Polynom derivation(){
  return null; /*TODO*/
}
```

Aufgabe 8 Die für die Datenklasse generierte Methode `equals` macht keinen Vergleich über die Elemente der Reihung für die variable Parameteranzahl. Überschreiben Sie deshalb die Methode `equals`.

Soweit die Klasse für Polynome.

Java: Poly

```
}
```

4 Zeichnen von Graphen

Wir haben eine Datenstruktur für Polynome und eine Klasse, mit der Graphiken erstellt werden können. Dieses wollen wir nun verbinden, indem wir eine Klasse schreiben, die den Graphen einer Polynomfunktion und die Graphen ihrer ersten zwei Ableitungen in einem Koordinatensystem darstellt.

Hierzu ist eine Unterklasse der Klasse `PaintBoard` zu erstellen.

Java: Poly

```
class PolyBoard extends PaintBoard{
```

Die Klasse braucht folgende Informationen:

- das darzustellende Polynom

- das Itervall für die x-Werte, das dargestellt wird
- das Itervall für die y-Werte, das dargestellt wird

Hierfür sehen wir die folgenden 5 Felder vor:

Java: Poly

```
Polynom poly;
double minX;
double maxX;
double minY;
double maxY;
```

Zusätzlich ist es wichtig zu wissen, welchen Bereich ein Pixel in der Bilddarstellung im Zahlenbereich entspricht. Dieses kann in x und in y Richtung unterschiedlich sein. Für diese Auflösung sind für beide Dimensionen Felder definiert.

Java: Poly

```
double xRes;
double yRes;
```

Aufgabe 9 Vervollständigen Sie den Konstruktor für diese Klasse. Es soll insbesondere aus den Intervallen für x und y und den Werten für die Weite und Höhe, die korrekte Auflösung pro Pixel berechnet werden.

Java: Poly

```
PolyBoard( int width, int height
         , double minX, double maxX, double minY, double maxY
         , Polynom poly){
  super(width, height);
  /*TODO*/
}
```

Jetzt können Sie bereits ein Objekt zum Zeichnen eines speziellen Polynomgraphen anlegen.

Shell

```
jshell> var pb = new Poly.PolyBoard(400,400, -20, 20, -20, 20, pol)
pb ==> Poly$PolyBoard[,0,0,0x0,invalid,layout=java.awt.F ...
↪  nimumSize=,preferredSize=]
```

Aufgabe 10 Vervollständigen Sie die zwei Hilfsfunktionen die für eine Pixelposition den jeweiligen x bzw. y Wert des Pixels an dieser Stelle errechnen. Beachten Sie, dass die Pixel der y-Achse von oben nach unten steigend sind.

Java: Poly

```
double pxToX(int px){
  return 0; /*TODO*/
}
double pxToY(int py){
  return 0; /*TODO*/
}
```

Aufgabe 11 Jetzt geht es um die umgekehrte Richtung. Für einen Wert auf der x bzw. auf der y-Achse soll das dazugehörige Pixel errechnet werden.

Java: Poly

```
int yToPx(double y){
  return 0;/*TODO*/
}
int xToPx(double x){
  return 0;/*TODO*/
}
```

4.1 Zeichnen der Graphen

Es ist die abstrakte Methode `paintWith` zu überschreiben. In dieser werden die noch zu implementierende Funktionen `paintCoordinates` und `paintGraph` aufgerufen, die das Koordinatensystem bzw. den Graphen eines Polynoms in das Koordinatensystem zeichnen. Das Polynom wird dabei in blauer, die erste Ableitung in roter und die zweite Ableitung in grüner Farbe gezeichnet.

Java: Poly

```
@Override
void paintWith(Graphics2D g){
  g.setColor(Color.BLACK);
  paintCoordinates(g);
  g.setColor(Color.BLUE);
  paintGraph(poly,g);
  var poly1 = poly.derivation();
  g.setColor(Color.RED);
  paintGraph(poly1,g);
  var poly2 = poly1.derivation();
  g.setColor(Color.GREEN);
  paintGraph(poly2,g);
}
```

Wenn die fehlenden Methoden ergänzt sind, können Sie ein Bild ähnlich der Abbildung 2 erhalten.

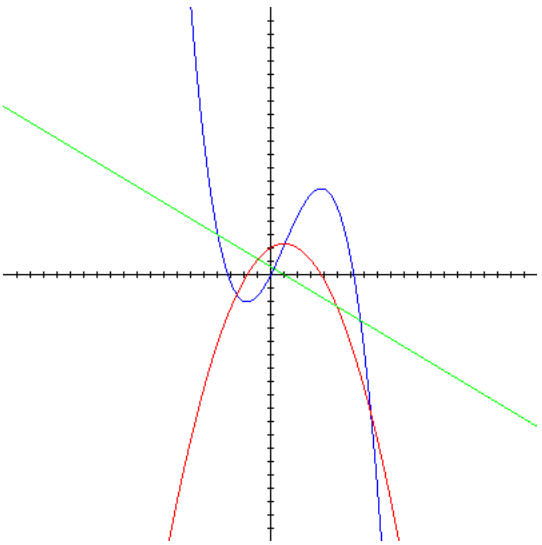

Abbildung 2: Graph für $2x + 0.3x^2 - 0.1x^3$.

Aufgabe 12 Vervollständigen Sie die Funktion, die das Koordinatensystem zeichnet.

Java: Poly

```
void paintCoordinates(Graphics2D g){
  /*TODO*/
}
```

Aufgabe 13 Vervollständigen Sie die Funktion, die den Graphen des übergebenen Polynoms im Koordinatensystem zeichnet. Gehen Sie hierzu die Pixel der x-Koordinate durch, berechnen jeweils den y-Wert im Polynom und zeichnen eine Linie zum y-Wert des nächsten Pixels.

Java: Poly

```
void paintGraph(Polynom p, Graphics2D g){
  /*TODO*/
}
```

Aufgabe 14 Sie haben jetzt ein rudimentäre Tool zum Zeichnen von Funktionsgraphen für Polynome. Erweiteren Sie dieses um ein paar Aspekte, die Sie in persönlicher Kreativität reizen. Dieses könnte ein Beschriftung der Koordinatenachsen sein, oder die Angabe des dargestellten Polynoms. Vielleicht wollen Sie auch Hilflinien als Gitter ins Koordinatensystm zeichnen.

Aufgabe 15 Erstellen Sie mit Ihrem Tool ein paar schöne Funktionsgraphen.

Java: Poly

```
}
```

5 Lernzuwachs

- Variable Parameteranzahl zur Arrayübergabe
- abstrakte Klassen
- Abstraktionsbildung durch Unterklassen
- Die Swing/AWT Klasse `Graphics2D` für Graphiken
- Import von Paketen
- innere (statische) Klassen

Java: Poly

```
}
```