Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, I

Aufgabe 1: Die Java-Klassenbibliotheken

Einer der Gründe, die der Programmiersprache Java zu ihrem Erfolg und ihrer Verbreitung verholfen haben, ist die Existenz einer standardisierten Klassenbibliothek mit einer Vielzahl von nützlichen vordefinierten Klassen für unterschiedlichste Zwecke.

Pakete:
Da diese Bibliotheken Tausende von Klassen enthalten, wurden sie in Paketen (logisch verwandte Gruppen von Klassen) zusammengefasst.

Wenn ein Klassenname Punkte beinhaltet, wie z.B. **java.lang.Math**, ist nur der letzte Teil Name der Klasse und die Teile davor bilden den Namen des Pakets. Die Klasse **`Math`** befindet sich also im Paket **`java.lang`**. Auch die Klasse **`System`**, die du bereits vom Befehl `System.out.println()` kennst, findet man in diesem Paket.

Glücklicherweise gibt es eine Möglichkeit, nachzuschlagen, welche vordefinierten Klassen es gibt und über welche Methode sie verfügen:

a) Öffne die Java-Dokumentation

https://docs.oracle.com/javase/8/docs/api/

und wähle links oben das Paket `java.lang` und darin die Klasse `Math` aus.

b) Suche die Methode `max(int a, int b)` der Klasse `Math`. Welche Aufgabe hat sie?

Handelt es sich um eine Funktion oder eine Prozedur?

c) Finde mit Hilfe der Klasse `Color` aus dem Paket `java.awt` heraus, welche vordefinierten Farbkonstanten es gibt.

Exkurs: RGB-Farbmodell:
Beim RGB-Farbmodell werden die drei Grundfarben Rot, Grün und Blau gemischt. Alle drei Farben zusammen ergeben Weiß. Jede der drei Farben kann mit einer Intensität von 0 bis 255 zugemischt werden (8 Bit pro Farbe). 0 bedeutet, dass die Farbe nicht eingemischt wird. Damit können 256*256*256 = 16.777.216 (16,7 Mio) Farben erzeugt werden.
Oft werden die einzelnen Farbanteile im Hexadezimalsystem wiedergegeben, statt 0 bis 255 wird also 0 bis FF verwendet. "Reines" Rot hätte damit den Code #FF0000, "reines" Blau den Code #0000FF. Schwarz wäre #000000, Weiß #FFFFFF. Grautöne erhält man, wenn alle drei Farben den gleichen Anteil haben (z.B. #555555 oder #A1A1A1). Das RGB-Farbmodell spielt im Bereich der Computergrafik eine zentrale Rolle.

Objektorientiertes Programmieren mit GPanel-Grafik

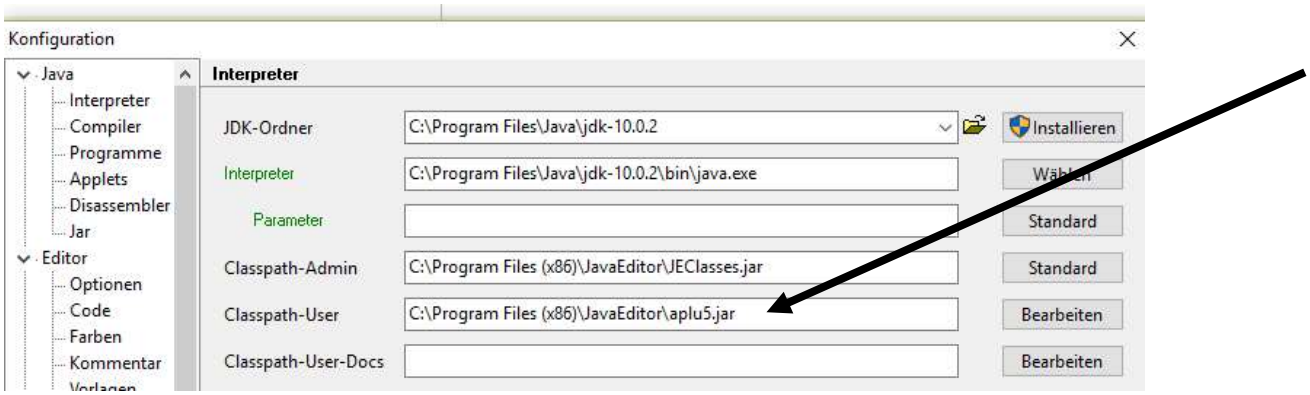
Das Projekt MeinPark, I

Aufgabe 2: Die Bibliothek ch.aplu.util

Für unser Programmierprojekt benötigen wir eine Grafikumgebung, die wir so direkt leider nicht in einer der vordefinierten Klassenbibliotheken finden. Besonders intuitiv und leicht erhältlich ist jedoch die **GPanel-Grafik**, die im Computerlabor der Universität Bern durch **Prof. Dr. Aegidius Plüss** konzipiert wurde. Die Hilfsklassen hierfür befinden sich in der jar-Datei "`aplu5.jar`".

Vorbereitung:

Kopiere die Datei **aplu5.jar** in deinen Java-Editor-Ordner und nimm den entsprechenden Pfad auf (über `Fenster→Konfiguration`):



Die verwendete Bibliothek **ch.aplu.util** stellt in der Datei **aplu5.jar** unzählige Hilfsklassen zur Verfügung, von denen für uns hauptsächlich die Klasse **GPanel** von Bedeutung ist (s.u.). Um dir jedoch einen Überblick über sämtliche Klassen dieser Bibliothek zu verschaffen, öffne die Datei `index.html` im Ordner `Dokumentation der aplu-Klassen`. Links erkennst du alle Klassen der Bibliothek.

Öffne die Klasse `BaseTimer`.

a) Wie viele Methoden besitzt diese Klasse? _____

b) Wie viele dieser Methoden besitzen einen Parameter? _____

c) Wie viele Methoden liefern einen Wert zurück? _____

d) Wozu dient die Methode `resume()`? _____

Auf eine vollständige Auflistung der Methoden der verschiedenen Hilfsklassen soll hier verzichtet werden - für einfache Zwecke reichen die nachfolgend beschriebenen Methoden der Klasse **GPanel** aus:

Objektorientiertes Programmieren mit GPanel-Grafik

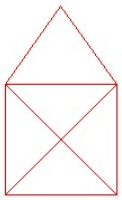
Das Projekt MeinPark, I

Methode	Erklärung
bgColor(java.awt.Color color)	Set the background color.
circle(double radius)	Draw a circle with center at the current graph position and given radius.
color(java.awt.Color color)	Set the given new color.
delay(int time)	Delay execution for the given amount of time (in ms).
draw(double x, double y)	Draw a line from current graph position to given window coordinates and set the graph position to the endpoint.
fillCircle(double radius)	Draw a filled circle with center at the current graph position and given radius.
fillPolygon(double[] x, double[] y)	Draw a filled polygon with given corner coordinates.
fillRectangle(double width, double height)	Draw a filled rectangle with center at the current graph position and given width and height
fillTriangle(double x1, double y1, double x2, double y2, double x3, double y3)	Draw a filled triangle with given corner coordinates.
font(java.awt.Font font)	Select the given font for all following text operations.
getKeyCode()	Return the keycode associated with last key pressed.
move(double x, double y)	Set the current graph position to given window coordinates (without drawing anything).
point(double x, double y)	Draw a single point at the given window coordinates.
polygon(double[] x, double[] y)	Draw a polygon with given corner coordinates.
text(double x, double y, java.lang.String str)	Draw a string at the given position.
title(java.lang.String title)	Set the title in the window's title bar.
triangle(double x1, double y1, double x2, double y2, double x3, double y3)	Draw a triangle with given corner coordinates.
window(double xmin, double xmax, double ymin, double ymax)	Set window coordinates.

Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, I

Aufgabe 3: Zeichnen mit der GPanel-Grafik



Öffne die bereits existierend Datei `Zeichenfenster.java` und führe das Programm aus. Noch passiert nicht viel:

- Um die Klassen, die im Paket `ch.aplu.util` liegen, außerhalb dieses Pakets nutzen zu können, müssen sie dem Compiler mit der gesamten Paketangabe bekannt gemacht werden. Hierzu dient die **import**-Deklaration in Zeile 1. Das * zeigt dabei an, dass alle Klassen des Pakets importiert werden sollen.

- In Zeile 5 wird das (bislang einzige) **Attribut** deklariert: Ein `GPanel`-Objekt namens `myGPanel`.

- In Zeile 7 beginnt der so genannte **Konstruktor** (hierauf wird später genauer eingegangen), der in Zeile 15 in der `main`-Methode aufgerufen wird.

- Durch den Befehl
  ```
  myGPanel = new GPanel(new Size(1200,700));
  ```
  in Zeile 8 wird das 1200*700 **Pixel** große Zeichenfenster `myGPanel` erzeugt.

- Durch den Befehl
  ```
  myGPanel.window(0, 1200, 0, 700);
  ```
  werden die **Koordinaten** des Fensters gesetzt, d.h. der Koordinatenursprung befindet sich in der linken unteren Ecke, die rechte obere Ecke hat die Koordinaten (1200|700).

- Durch den Befehl
  ```
  myGPanel.title("Das Haus vom Nikolaus");
  ```
  erhält das Fenster einen **Titel**.

Die Methoden `window` und `title` in den Zeilen 9 und 10 sind nicht in der Klasse `Zeichenfenster` implementiert, sondern in der Klasse `GPanel`. Damit sie gefunden werden, muss die Punktnotation verwendet werden:

Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, I

Punktnotation:
Ist eine Methode in einer anderen Klasse definiert, müssen wir die Klasse angeben, in der sich die Methode befindet, oder das Objekt nennen, für das die Methode aufgerufen werden soll, gefolgt von einem Punkt, dem Methodennamen und eventueller Parameter.

a) Implementiere die Methode `zeichne`, durch die ungefähr in der Mitte von `myGPanel` das „Haus vom Nikolaus“ in angemessener Größe unter Verwendung der Methoden `move` und `draw` der Klasse `GPanel` gezeichnet wird.

b) Ändere dein Programm so ab, dass dein Haus in einer anderen Farbe gezeichnet wird. Möchtest du es zum Beispiel in rot zeichnen, benötigst du dazu den Befehl `color(Color.RED)` der Klasse `GPanel`.
Achtung: Hierzu benötigst du die Java-Klasse `java.awt.Color`, die du zunächst importieren musst.

Aufgabe 4: Reaktion auf Mausklicks

Beim Arbeiten mit grafischen Oberflächen interagiert der Benutzer mit Komponenten. Er bewegt z.B. die Maus im Fenster, klickt eine Schaltfläche an oder verschiebt einen Rollbalken. Das grafische System beobachtet die Aktionen des Benutzers und informiert die Applikation über die anfallenden Ereignisse. Dann kann das laufende Programm entsprechend reagieren.

Bei unserem Programm soll z.B. das Haus vom Nikolaus erst gezeichnet werden, wenn der Benutzer mit der Maus ins das Zeichenfenster klickt. Durch einen weiteren Mausklick soll das Programm beendet werden.

Wir betrachten also das Ereignis „Mausklick“. Dafür müssen wir einen so genannten **MouseListener** implementieren. Hierbei handelt es sich sozusagen um einen „Horcher“, der während des gesamten Programmablaufs darauf achtet, ob ein Mausereignis eintritt und, falls dies der Fall ist, eine entsprechende Aktion auslöst.

Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, I

a) Füge folgenden Programmcode hinter den `title`- und vor den `zeichne`-Befehl in deiner Klasse `Zeichenfenster` ein:

```
myGPanel.addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent evt) {
    mausWurdeGedrueckt(evt);
  }
});
```

b) Der Programmcode aus Aufgabenteil 3a) sorgt dafür, dass bei Mausklick die Methode

```
public void mausWurdeGedrueckt(MouseEvent evt)
```

aufgerufen wird. Sie soll bewirken, dass nicht schon zu Beginn, sondern erst beim ersten Klick das Haus vom Nikolaus gezeichnet und beim zweiten Klick das Programm beendet wird.

Implementiere diese Methode!

Tipps:

- Du benötigst hierfür außer `myGPanel` ein weiteres Attribut (Typ: `boolean`), welches angibt, ob die Maus bereits geklickt wurde.
- Der Aufruf der Methode `zeichne()` innerhalb des Konstruktors entfällt nun offensichtlich, denn er erfolgt ja in dieser Methode.
- Die Ereignisklassen befinden sich im Paket **java.awt.event**, welches durch
  `import java.awt.event.*;`
  importiert werden muss.
- Der Befehl zum Beenden des Programms und schließen des Fensters lautet
  `System.exit(0);`