Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, II

Aufgabe 5: Die Klasse `MeinPark`

In deinem ersten Projekt mit GPanel-Grafik soll ein Park mit Blumen, einer Sonne und einem Schneemann dargestellt werden. Dazu sind bereits zwei Klassen vorgegeben: Die Klasse `MeinPark` und die Klasse `Blume`. Betrachten wir zunächst die Klasse `MeinPark`. Die meisten Befehle sind bereits aus der Klasse `Zeichenfenster` bekannt. Jedoch besitzt die Klasse `MeinPark` ein weiteres Attribut in Zeile 8: Ein Blumenobjekt namens `flower`.

Dieses Objekt wird in Zeile 23 mit dem Befehl `new Blume` erzeugt, wobei drei Parameter übergeben werden: die x- bzw. y-Position der Blume (d.h. genauer des unteren Endes des Blumenstängels) und das GPanel auf dem die Blume gezeichnet werden soll. (Was durch den `new`-Befehl genau passiert, wird in Aufgabe 6 beschrieben).

In Zeile 32 wird `flower` gezeichnet, indem die Methode `zeichne()` der Klasse `Blume` aufgerufen wird. Diese sorgt dafür, dass eine Blume an die übergebene Position auf das übergebene GPanel gezeichnet wird. Die Farbe der Blüte wird dabei zufällig aus den Farben rot, blau, gelb und orange gewählt. Wie genau das geschieht, schauen wir uns später an.

a) Erweitere die Klasse `MeinPark` derart, dass nicht nur eine sondern fünf Blumen an verschiedene Positionen verteilt über das Fenster gezeichnet werden. Die Klasse `Blume` brauchst du dazu nicht verändern. Wenn du es hinbekommst, kannst du ein Blumenarray verwenden, anstelle von 5 einzelnen Blumen!

b) Verändere das Programm so, dass die Positionen der 5 Blumen nun zufällig gewählt werden, wobei darauf zu achten ist, dass sie sich nicht zu nahe am Rand befinden sollten.
   Eine ganzzahlige Zufallszahl zwischen z.B. 0 und 99 (jeweils einschließlich) erhältst du durch:
   ```
   int zufall = (int) (Math.random()*100);
   ```
   Um Zufallszahlen passend zu deiner Fensterbreite bzw. -höhe zu erhalten, brauchst du nur den Wert 100 anzupassen.

Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, II

Aufgabe 6: Die Klasse Blume

Blume
gp: GPanel xPos: Zahl yPos: Zahl farbe: Color
zeichne()

Schauen wir uns nun die Klasse Blume anhand des Entwurfsdiagramms rechts genauer an:

Jede Blume besitzt vier Attribute (Code Zeile 19 – 22):

- das **GPanel**, auf das sie gezeichnet werden soll,
- die **x**- und die **y**-Position und die
- **Farbe** der Blüte.

Außerdem verfügt eine Blume über die Methode **zeichne()** (Code Zeile 53 – 56), die dafür sorgt, dass sie an die entsprechende Position gezeichnet wird.

Aber wie gelangen nun Blumen in den Park? Betrachte dazu noch einmal den Code, mit dem ein Blumenobjekt in der Klasse `MeinPark` erzeugt wurde:

```
flower = new Blume(800,500,myGPanel);
```

Hier wird der `new`-Operator verwendet, um ein neues Exemplar der Klasse `Blume` zu erzeugen. Die dabei aufgerufene Methode heißt ebenfalls `Blume` und wird **Konstruktor** genannt. Der dazugehörige Code befindet sich in den Zeilen 29 bis 48 der Klasse `Blume` und hat die Aufgabe, dafür zu sorgen, dass die Attribute mit den passenden Werten belegt werden;

- In den Zeilen 30 und 31 findet eine Zuweisung der x- und der y-Koordinate statt.
- In Zeile 32 wird dem GPanel-Attribut der entsprechende Wert zugewiesen. Da hier Attribut und Parameter dieselbe Bezeichnung (nämlich `gp`) haben, sorgt das Schlüsselwort `this` dafür, das zwischen dem Attribut (`this.gp`) und dem Parameter (`gp`) unterschieden werden kann.
- Das Attribut `farbe` schließlich erhält seinen (hier zufälligen) Wert in der `switch`-Anweisung.

Konstruktor:

Der Konstruktor eine Klasse ist eine besondere Art Methode, die immer automatisch ausgeführt wird, wenn ein Objekt dieser Klasse mit `new` erzeugt wird. Ein Konstruktor besitzt keinen Rückgabetyp (auch nicht `void`), der zwischen dem Schlüsselwort `public` und dem Namen stehen würde. Der Name des Konstruktors lautet immer genau so wie die Klasse selbst. Der Code im Konstruktor wird automatisch bei der Erzeugung des entsprechenden Objektes ausgeführt. Im Konstruktor stehen alle Befehle, die nötig sind, um das entsprechende Objekt bei seiner Erzeugung in den gewünschten Anfangszustand zu versetzen.

Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, II

a) Implementiere einen weiteren Konstruktor der Klasse `Blume`, der zusätzlich zu den drei bisher vorhandenen Parametern auch die Farbe der Blume als Parameter besitzt. Eine Blume, die mit diesem Konstruktor erzeugt wird, soll also keine zufällige, sondern eine vom aufrufenden Programm vorgegebene Farbe haben.

b) Ändere dein Hauptprogramm (also die Klasse, in der sich die `main`-Methode befindet, d.h. die Klasse `MeinPark`) derart ab, dass nun fünf Blumen an zufälliger Position in fünf unterschiedlichen Farben gezeichnet werden, indem du den Konstruktor aus Aufgabenteil a) verwendest.

   (Welche Farben zur Verfügung stehen hast du ja bereits in Aufgabe 1c untersucht.)

Aufgabe 7: Privat oder öffentlich?

Bisher waren alle Methoden, die du geschrieben hast, `public`, d.h. öffentlich. Wie du an den Methoden `zeichneStaengel()` und `zeichneBluete()` der Klasse `Blume` erkennen kannst, können Methoden aber auch `private` sein. **Private Methoden sind nur innerhalb der Klasse sichtbar, in der sie deklariert wurden.** Sie werden verwendet, um die Struktur des Codes zu verbessern.

public/private:
Wenn Methoden von außerhalb der Klasse aufgerufen werden sollen, dann müssen sie `public` sein. Wenn Methoden nur von anderen Methoden derselben Klasse aus aufgerufen werden sollen (wie es z.B. bei der Methode `zeichneStaengel()` der Fall ist), dann könne sie als `private` deklariert werden.

Private Methoden sind außerhalb der Klasse nicht sichtbar und erlauben auch keinen Zugriff. Es hat sich bewährt, Methoden, die nur für den internen Gebrauch bestimmt sind, als `private` zu deklarieren. Das trägt dazu bei, Fehler zu vermeiden, und weist deutlicher auf den Zweck der Methode hin.

Wie du erkennen kannst, sind auch die Attribute der Klasse `Blume` private. Damit befolgen wir das bei der objektorientierten Programmierung übliche **Geheimnisprinzip**. Dieses Prinzip besagt, dass ein Objekt möglichst alle seine Attribute vor außenstehenden Objekten verbergen soll. Nur das Objekt selbst soll alle seine Attribute kennen und darauf zugreifen dürfen. Wenn ein Zugriff auf Attributwerte oder eine Veränderung von Attributwerten notwendig ist, soll dies durch den Aufruf von Methoden des Objekts geschehen (kommt später genauer). So behält das Objekt stets die Kontrolle darüber, welche Werte seine Attribute annehmen.

Objektorientiertes Programmieren mit GPanel-Grafik

Das Projekt MeinPark, II

Dass dieses Geheimnisprinzip sinnvoll ist, zeigt unser Beispiel: Einmal gezeichnet kann z.B. die Farbe der Blume nicht mehr geändert werden – das macht in unserem Projekt durchaus Sinn.

a) Im Programm `TestPublicPrivate` wird in Zeile 33 versucht, die Farbe von `flower` zu ändern, um sie dann neu zu zeichnen. Compiliere das Programm. Wie lautet die entsprechende Fehlermeldung?

b) Ändere nun (zu Testzwecken und nur vorübergehend!) die Sichtbarkeit des Attributs `farbe` in der Klasse `Blume` von `private` auf `public` und versuche erneut, das Programm `TestPublicPrivate` zu compilieren und auszuführen. Was beobachtest du?

c) Was passiert, wenn du nach der Änderung aus Aufgabe b) in Zeile 34 nicht die Methode `zeichne()` sondern nur `zeichneBluete()` aufrufst?

d) Mache die Änderung aus b) rückgängig.

Aufgabe 8: Die Klasse Sonne

Als nächstes soll im Park die Sonne scheinen.

Sonne
gp: GPanel xPos: Zahl yPos: Zahl radius: Zahl
zeichne()

a) Implementiere die Klasse Sonne anhand des rechts stehenden Entwurfsdiagramms. Dem Konstruktor solltest du das GPanel, die x- und y-Position des Mittelpunkts sowie den Radius übergeben.

b) Erweitere dein Hauptprogramm `MeinPark` aus Aufgabe 5 derart, dass bei Mausklick nun nicht nur die Blumen gezeichnet werden, sondern auch eine Sonne in Form eines gelben Kreises mit einem Radius von 40 Pixeln im linken oberen Teil des GPanels erscheint.

c) Verändere den Code in `MeinPark` nun noch so, dass die fünf Blumen an zufälligen Positionen gezeichnet werden, die mindestens 100 Pixel weit vom Mittelpunkt der Sonne entfernt sind.