Gartenarchitektur

Anmerkungen

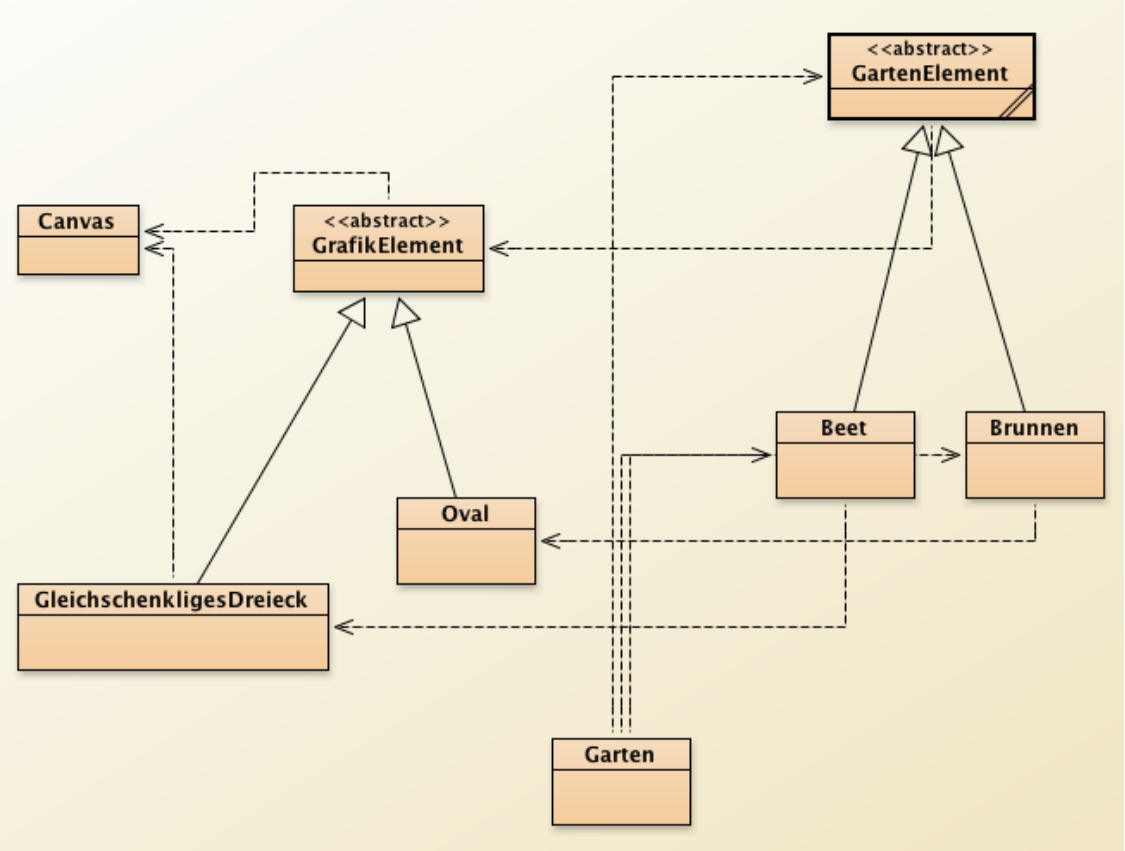
- erhöhtes Anforderungsniveau
- vorgesehene Bearbeitungszeit: 150 min

Aufgabe

Eine kleine Softwarefirma hat den Auftrag, eine Grafikanwendung zu entwickeln, die die Erstellung kleiner Karten, beispielsweise zur Anlage eines Parks oder eines Gartens, ermöglicht. Der Kunde liefert die nebenstehende Skizze eines Parks, die er mit dem Programm erstellen möchte.

Die Gartenanlagen sollen aus Beeten bzw. Wegen bestehen, deren Begrenzungen klaren Linien folgen. Außerdem soll es möglich sein, Büsche, Bäume und Hecken darzustellen. Im Park soll es außerdem Brunnen geben. Dabei wird vom Auftraggeber ausdrücklich darauf hingewiesen, dass nur ein geringes Budget zur Verfügung steht. Deshalb werden einfache Darstellungen akzeptiert. Das Programm soll nur streng geometrische Formen wie Dreiecke oder Ovale erlauben.

1. Die folgende Abbildung zeigt den ersten Klassenentwurf als vereinfachtes UML-Diagramm. Die Anlage 1 stellt die Methoden und Attribute der einzelnen Klassen im Detail dar.
   Erläutern Sie die Beziehungen zwischen den Klassen.



2. Ein Objekt der Klasse `Garten` wurde dazu genutzt, das nebenstehende Bild zu zeichnen.

Nennen Sie die vier Objekte, die von der Klasse `Garten` erzeugt werden.

Die Klasse `Garten` enthält die Methode `zeichne()`.
Beschreiben Sie ausführlich, welche Objekte betroffen sind und wie sie agieren, damit dieses Bild auf der Zeichenfläche erscheint.

3. Es soll ein Konstruktor der Klasse `Beet` ergänzt werden, bei dem die Größe und die Farbe des Gartenelements als Parameter übergeben werden können.

   Erläutern Sie, welche Aufgabe Konstruktoren in der objektorientierten Programmierung haben. Begründen Sie, weshalb häufig mehrere Konstruktoren verwendet werden.

   Erläutern und implementieren Sie den neuen Konstruktor. Der aktuelle Konstruktor ist in Anlage 2 enthalten.

4. Der Kunde wünscht eine Erweiterung des Programmes, sodass auch Beete in der Form eines rechtwinkligen Dreiecks dargestellt werden können.

   Entwickeln Sie eine Klasse `RechtwinkligesDreieck`. Es ist dabei ausreichend, wenn Sie für die Position des rechten Winkels nur einen einfachen Spezialfall berücksichtigen.
   Erläutern Sie Ihren Entwurf und implementieren Sie diese Klasse.

5. Damit Beete und Brunnen vergrößert und verkleinert werden können, soll eine Methode `skaliere` implementiert werden.

   Begründen Sie, dass es am einfachsten ist, die Methode `skaliere` in der Klasse `GartenElement` zu implementieren.

   Implementieren Sie diese Methode.

6. Mit der Anwendung sollen auch Baumreihen dargestellt werden können. Dazu werden der Anfangspunkt, der Endpunkt und die Anzahl der Bäume in der Baumreihe angegeben.

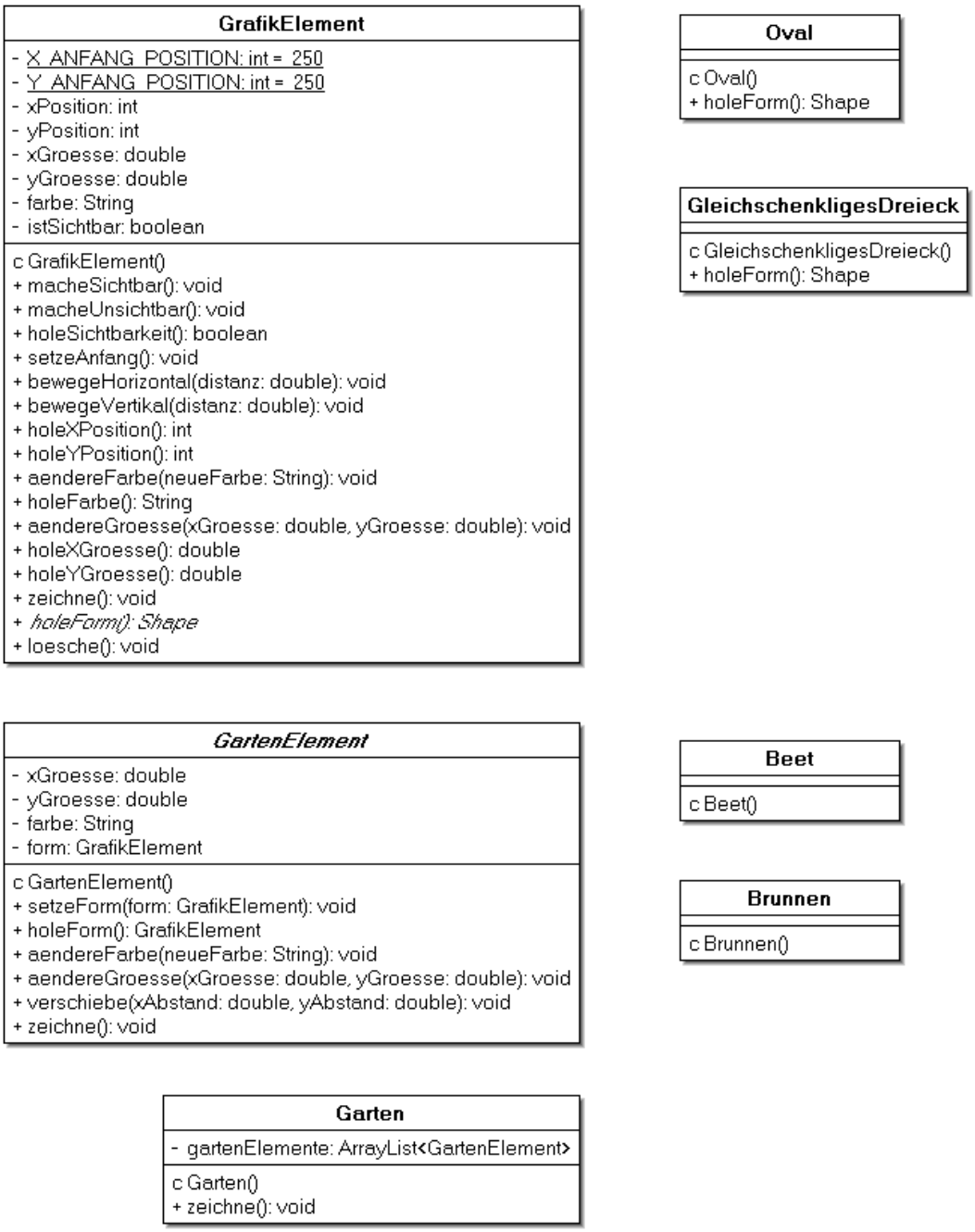
   Erläutern Sie den Entwurf und implementieren Sie die Klassen.

7. Nachdem der Entwurf des Gartens erstellt ist, möchte der Kunde wissen, welche Gesamtfläche alle Beete und Brunnen seines Gartens zusammen belegen.

   Erläutern Sie, wie diese zusätzliche Funktionalität implementiert werden sollte.

   Arbeiten Sie heraus, wie die Berechnung der Gesamtfläche zur Laufzeit konkret erfolgt.

Anlage 1


GrafikElement
- X_ANFANG_POSITION: int = 250
- Y_ANFANG_POSITION: int = 250
- xPosition: int
- yPosition: int
- xGroesse: double
- yGroesse: double
- farbe: String
- istSichtbar: boolean
c GrafikElement()
+ macheSichtbar(): void
+ macheUnsichtbar(): void
+ holeSichtbarkeit(): boolean
+ setzeAnfang(): void
+ bewegeHorizontal(distanz: double): void
+ bewegeVertikal(distanz: double): void
+ holeXPosition(): int
+ holeYPosition(): int
+ aendereFarbe(neueFarbe: String): void
+ holeFarbe(): String
+ aendereGroesse(xGroesse: double, yGroesse: double): void
+ holeXGroesse(): double
+ holeYGroesse(): double
+ zeichne(): void
+ holeForm(): Shape
+ loesche(): void
Oval
c Oval()
+ holeForm(): Shape
GleichschenkligesDreieck
c GleichschenkligesDreieck()
+ holeForm(): Shape
GartenElement
- xGroesse: double
- yGroesse: double
- farbe: String
- form: GrafikElement
c GartenElement()
+ setzeForm(form: GrafikElement): void
+ holeForm(): GrafikElement
+ aendereFarbe(neueFarbe: String): void
+ aendereGroesse(xGroesse: double, yGroesse: double): void
+ verschiebe(xAbstand: double, yAbstand: double): void
+ zeichne(): void
Beet
c Beet()
Brunnen
c Brunnen()
Garten
- gartenElemente: ArrayList<GartenElement>
c Garten()
+ zeichne(): void

Anlage 2

```
1 /**
2  * Konstruktor der Klasse Beet
3  */
4 public Beet()
5 {
6     setzeForm(new GleichschenkligesDreieck());
7     aendereFarbe("green");
8     aendereGroesse(30.0, 80.0);
9 }
```

```
1  // Die Klasse gleichschenkliges Dreieck
2  import java.awt.*;
3  import java.awt.geom.*;
4
5  public class GleichschenkligesDreieck extends GrafikElement
6  {
7     /**
8     * Erzeugt ein Gleichschenkliges Dreieck.
9     */
10    public GleichschenkligesDreieck()
11    {
12          aendereFarbe("yellow");
13          aendereGroesse(100, 30);
14    }
15
16    /**
17     * Holt den Umriss des gleichschenkligen Dreiecks.
18     *
19     * @return Shape    Umriss des gleichschenkligen Dreiecks.
20     */
21    public Shape holeForm()
22    {
23          GeneralPath form = new GeneralPath();
24          form.moveTo(holeXPosition() - holeXGroesse() / 2.0, holeYPosition());
25          form.lineTo(holeXPosition(), holeYPosition() + holeYGroesse());
26          form.lineTo(holeXPosition() + holeXGroesse() / 2.0 , holeYPosition());
27          form.closePath();
28          return form;
29    }
30 }
```

Lösungshinweise

Aufg.	erwartete Leistungen
1	Der `Garten` setzt sich aus `Brunnen` und `Beeten` zusammen, es handelt sich um eine Teile-Ganzes-Beziehung. `Brunnen` und `Beete` sind Spezialfälle der Klasse `GartenElement`, sie erben von dieser Klasse. `GleichschenkligesDreieck` und `Oval` sind Spezialfälle der Klasse `GrafikElement`, sie erben von dieser Klasse. Das `GartenElement` benutzt `GrafikElement`. Das Beet benutzt `GleichschenkligesDreieck`, der `Brunnen` benutzt `Oval`. Anmerkung: *Die meisten Methoden von `GrafikElement` werden von der allgemeinen Klasse vorgegeben, die abstrakte Methode `holeForm` wird in den speziellen Klassen überschrieben (implementiert). Die von `GartenElement` abgeleiteten Klassen unterscheiden sich nur in den Konstruktoren.*
2	Es gibt einen Brunnen und drei Beete. Es werden folglich diese vier Objekte erzeugt. Wird die Methode `zeichne()` aufgerufen, so muss über die Elemente der ArrayList `GartenElemente` (siehe UML-Diagramm) iteriert werden. Dadurch erhält jedes einzelne `GartenElement` die Nachricht `zeichne()`. Je nachdem, ob ein `GartenElement` ein Beet oder ein Brunnen ist, wird damit deren spezifische Methode `zeichne()` aufgerufen. Diese Objekte reichen die Nachricht `zeichne` an ihre jeweiligen `GrafikElemente`, nämlich `GleichschenkligesDreieck` und `Oval`, weiter. Dabei werden Farbe, Abmessungen und Position über das Attribut `form` kommuniziert.

3	Der Konstruktor hat die Aufgabe, einen gültigen Anfangszustand für ein Objekt sicherzustellen.
	Mehrere Konstruktoren sind sinnvoll, damit Objekte unterschiedlicher Anfangszustände einfach erzeugt werden können.
	Für den neuen Konstruktor `Beet` werden drei Parameter benötigt. Der Parameter für die Farbe muss vom Typ String sein und die Abmessungen des Beetes (`xGroesse` und `yGroesse`) vom Typ `double`. *Dieser Satz kann in der Erläuterung fehlen, wenn dies im Programmtext des Konstruktors richtig implementiert ist.*
	Da die Zustandsgrößen in der übergeordneten Klasse `private` angelegt sind, kann der Konstruktor die Werte nicht direkt setzen, sondern muss dies über die „Setter“ der abstrakten Klasse tun.
	
	`1 public Beet(String farbe, double xGroesse, double yGroesse)`
	`2 {`
	`3  setzeForm(new GleichschenkligesDreieck());`
	`4  aendereFarbe(farbe);`
	`5  aendereGroesse(xGroesse, yGroesse);`
	`6 }`

4	Dazu muss eine Klasse geschrieben werden, die von der abstrakten Klasse `GrafikElement` abgeleitet wird. In der Klasse wird dann die Methode `holeForm` implementiert. Anmerkung: Für die Lösung ist es ausreichend, wenn der Prüfling ein Dreieck zeichnet, dessen Katheten parallel zur x- bzw. y-Achse liegen. Bei der folgenden Lösung wird ein rechtwinkliges Dreieck so gezeichnet, dass der rechte Winkel oben links positioniert wird.

Aufg.	erwartete Leistungen

	1 **public class** RechtwinkligesDreieck **extends** GrafikElement
	2 {
	3 **public** RechtwinkligesDreieck()
	4 {
	5 aendereGroesse(100, 30);
	6 aendereFarbe("green");
	7 }
	8
	9 **public** Shape holeForm()
	10 {
	11 GeneralPath form = **new** GeneralPath();
	12 form.moveTo(holeXPosition(), holeYPosition());
	13 form.lineTo(holeXPosition(),
	14 holeYPosition() + holeYGroesse());
	15 form.lineTo(holeXPosition() + holeXGroesse(),
	16 holeYPosition());
	17 form.closePath();
	18 **return** form;
	19 }
	20 }

5	Die Methode `skaliere` muss nur einmal in der Klasse `GartenElement` implementiert werden, da die Aufgabe der Skalierung für alle `GartenElement`-Objekte gleich ist. Es müssen die Attribute für die Größe des Gartenelements mit dem Skalierungsfaktor multipliziert werden. Sollte die Methode in der Klasse `Beet` bzw. `Brunnen` implementiert werden, müssen „Getter“ für die bestehenden Größen implementiert werden, da die Sichtbarkeit auf den Wert `private` gestellt wurde.
	Alternativ ließe sich die Methode auch mit Hilfe von `aendereGroesse` realisieren.
	
	Beispiel einer Methode:
	1 /**
	2 * skaliere aendert die Groesse um den angegebenen Faktor
	3 * @param faktor Veränderung
	4 */
	5 **public void** skaliere(**double** faktor)
	6 {
	7 xGroesse *= faktor;
	8 yGroesse *= faktor;
	9 }
	
	Anmerkung: Die Prüflinge könnten auch eine zusätzliche Anweisung
	`form.aendereGroesse(xGroesse, yGroesse);`
	nach Zeile 8 hinzufügen, da für sie nicht erkennbar ist, an welcher Stelle die Attribute `xGroesse` und `yGroesse` des Grafikelementes aktualisiert werden.

6	Zuerst sollte eine Klasse `Baum` erstellt werden, die entsprechend `Brunnen` und `Beet` allein den Konstruktor enthält. Ohne diese Klasse ist die Modellierung unbefriedigend.
	
	1 **public class** Baum **extends** GartenElement
	2 {
	3 **public** Baum()
	4 {
	5 setzeForm(**new** Oval());
	6 aendereFarbe("green");
	7 aendereGroesse(50.0, 50.0);
	8 }

Aufg.	erwartete Leistungen

	9 }
	
	In jedem Fall ist die Klasse `Baumreihe` erforderlich. Bei der hier vorgestellten Lösung werden insgesamt fünf Attribute benötigt, und zwar *Anfangsposition* (x, y)*, Endposition* (x, y), *Anzahl*, deren Werte im Konstruktor definiert werden. Notwendig ist der Einsatz einer Sammlungsklasse für die zugehörigen Baum-Objekte und bei den Zugriffen eine Iteration.
	
	`1 public class Baumreihe extends GartenElement`
	`2 {`
	`3 private double xAnfang, yAnfang;`
	`4 private double xEnde, yEnde;`
	`5 private int anzahl;`
	`6 private ArrayList<Baum> baumReihe;`
	`7`
	`8 /**`
	`9 * Konstruktor für Objekte der Klasse Baumreihe`
	`10 */`
	`11 public Baumreihe(double xAnfang, double yAnfang,`
	`12 double xEnde, double yEnde, int anzahl)`
	`13 {`
	`14 GartenElement baum;`
	`15 baumReihe = new ArrayList<Baum>();`
	`16 this.xAnfang = xAnfang;`
	`17 this.yAnfang = yAnfang;`
	`18 this.anzahl = anzahl;`
	`19 double dx = (xEnde - xAnfang) / (anzahl - 1);`
	`20 double dy = (yEnde - yAnfang) / (anzahl - 1);`
	`21 for (int i = 0; i < this.anzahl; i++)`
	`22 {`
	`23 baumReihe.add(new Baum());`
	`24 baumReihe.get(i).verschiebe(xAnfang + i*dx,`
	`25 yAnfang + i*dy);`
	`26 };`
	`27 }`
	`28`
	`29 /**`
	`30 * Zeichnet die Baumreihe`
	`31 */`
	`32 public void zeichne()`
	`33 {`
	`34 for (Baum baum: baumReihe) baum.zeichne();`
	`35 }`
	`36 }`
	
	Da eine Baumreihe ein spezielles Gartenelement ist, sollte die Klasse `Baumreihe` von der Klasse `GartenElement` erben. Dies ermöglicht zudem den einfachen Zugriff in der Klasse `Garten`.
	Für eine vollständige Anwendung müssen noch die „Setter“ von `GartenElement` überschrieben werden.
	Anmerkung: Die letzte Aussage wird von den Prüflingen nicht erwartet. Eine Lösung, bei der die Bäume nur horizontal oder vertikal ausgerichtet gezeichnet werden können, ist zulässig.

7	Dieses kann so realisiert werden, dass eine Methode `berechneGesamtFlaeche` in `Garten` implementiert wird, die über alle Gartenelemente iteriert. In der Klasse `GartenElement` wird eine abstrakte Methode `berechneFlaeche` benötigt, die in den abgeleiteten Klassen `Beet` und `Brunnen` überschrieben wird. Sowohl die Beet- als auch die Brunnenobjekte können über die Methode `holeForm()` auf das Objekt vom Typ `GrafikElement` zugreifen. Mithilfe der Methoden `holeXGroesse()` und `holeYGroesse()` des Grafikelementes können die aktuellen Abmessungen ermittelt und damit die Fläche berechnet werden.

Aufg.	erwartete Leistungen
	Wird die Methode `berechneGesamtFlaeche` der Klasse `Garten` aufgerufen, so wird die Methode `berechneFlaeche` aufgerufen, und zwar in der von dem jeweiligen Objekttyp (Beet- bzw. Brunnenobjekt) überschriebenen Variante und anschließend alle Werte addiert.

Quelle: Freie und Hansestadt Hamburg, Behörde für Schule und Berufsbildung, Abituraufgaben Informatik 2014

Zuordnung zu den Prozess-, Inhalts- und Anforderungsbereichen

Aufg.	Prozessbereiche					Inhaltsbereiche					Bewertungseinheiten in Anforderungsbereichen		
	MI	BB	SV	KK	DI	ID	AL	SA	IS	IMG	I	II	III
1	X				X				X		2	3	
2	X				X				X			3	2
3	X	X		X		X	X		X		5		
4	X			X		X	X		X		6	3	
5	X	X		X	X	X			X			6	2
6	X			X		X	X		X			5	3
7	X	X		X		X			X			5	5
										Summe 50	13	25	12