
Diagrammeditor-Übung Teil 0

Aufgabe 0.0*: Aufgaben mit *

Folgt der Aufgabennummer ein *, so müssen Sie die Aufgabe nicht zwingend bearbeiten, sondern nur, wenn Sie noch genügend Zeit haben. *-Aufgaben sind für das entstehende Projekt nicht notwendig.

Aufgabe 0.1: Allgemeine Hinweise

Unter Windows kann es sein, dass `java` und `javac` nicht auf Anhieb funktionieren. Sie können das Problem beheben, indem sie die folgende Zeile eingeben:

```
set PATH=C:\Programme\jdk122\bin;%PATH%
```

Aufgabe 0.2: Code ist nicht heilig

Trauen Sie sich, Code wieder zu löschen. Die in Code investierte Arbeit ist nicht das Tippen, sondern das Denken.

Aufgabe 0.3: Mut zu Notlösungen (hacks)

Es ist nicht verwerflich, unfertige Programmteile zu implementieren, selbst wenn Sie nicht zufrieden sind mit der Lösung. Viel schlimmer ist es, an diesen Teilen festzuhalten (siehe 0.2). Manchmal findet man die richtige Lösung erst, nachdem man mehrere "falsche" geschrieben hat

Aufgabe 0.4: Ein wenig Disziplin

Halten Sie sich an die vorgegebenen Klassen- und Methodennamen. Dies erleichtert die Diskussion über die Lösungen, das Übernehmen von fremden Code ("abschreiben"), die Vorgabe neuer Klassen und die Integration ganzer Programmteile.

Aufgabe 0.5: Lesen bildet!

Lesen Sie jeden Übungszettel zunächst komplett durch — alle Aufgaben bauen aufeinander auf. Jeder Übungszettel bildet aber für sich eine Einheit, sie sollten keine Informationen vom nächsten Zettel benötigen.

Aufgabe 0.6*: Musterlösungen

Musterlösungen sind nach und nach unter <http://www.inf.fu-berlin.de/lehre/WS00/java> erhältlich.

Diagrammeditor-Übung Teil 1

Aufgabe 1.1: Schreiben Sie ein “Hello-World”-Applet.

Erzeugen Sie eine Unterklasse von `java.awt.applet`, und überschreiben Sie die `paint()`-Methode. Nutzen Sie das `Graphics`-Objekt und die Methode `drawString`.

Aufgabe 1.2: Testen Sie die Methoden von Graphics

Benutzen Sie die verschiedenen Methoden von `Graphics` (siehe Script) in Ihrem Hello-World-Applet, um weitere grafische Elemente auszugeben.

Aufgabe 1.3: Verändern Sie die Schriftart

Erzeugen Sie ein `Font`-Objekt und setzen Sie es als Default-font in `Graphics`.

Aufgabe 1.4: Verändern Sie die Farben

Benutzen Sie die `setColor`-Methode von `Graphics`, um ihre Testobjekte verschieden einzufärben.

Aufgabe 1.5: Zeichnen Sie eine Textbox

Zeichnen Sie nun ein Rechteck genau (mit 10 Pixeln Abstand) um einen Text. Finden Sie die Grösse des Textes mit Hilfe eines `FontMetrics`-Objektes heraus. Färben Sie das Rechteck schwarz und den Text weiss ein, und füllen Sie das Rechteck in rot.

Aufgabe 1.6: Zeichnen Sie drei Textboxen

Überlegen Sie, wie Sie drei solche Textboxen zeichnen würden. Kopieren Sie den Code? Schreiben Sie eine Methode? Schreiben Sie eine Klasse?

Diagrammeditor-Übung Teil 2

Aufgabe 2.1: Schreiben Sie eine `TextBox`-Klasse

Benutzen Sie Ihre Erfahrungen aus Teil 1, um eine Klasse `TextBox` zu schreiben. Diese soll einen Konstruktor `TextBox(String s, int x, int y)` und eine Methode `draw(Graphics g)` besitzen. Der Aufruf von `draw` soll eine Textbox wie in Aufgabe 1.5 beschrieben mit *Mittelpunkt* (x,y) und Text `s` auf dem übergebenen `Graphics`-Objekt zeichnen.

Aufgabe 2.2: Denken Sie über Diagramm-Objekte nach

Welche graphischen Objekte sollte ein Diagrammeditor zur Verfügung stellen? Was fällt Ihnen zusätzlich zu `Box`, `FilledBox`, `TextBox`, `Line`, `Arrow` ein?

Aufgabe 2.3: Finden Sie die Abhängigkeiten

Welche Zusammenhänge bestehen zwischen den Objekten in einem Diagrammeditor?

Aufgabe 2.4: Finden Sie eine Hierarchie

Welche Eigenschaften von Diagrammobjekten lassen sich vererben?

Aufgabe 2.5: Finden Sie die Wurzel

Was ist allen Diagramm-Objekten gemeinsam? Überlegen Sie sich die Methoden und Variablen einer gemeinsamen Oberklasse `DiagramObject`.

Aufgabe 2.6: Implementieren Sie die Teilhierarchie `DiagramObject` bis `TextBox`.

Nutzen Sie Vererbung, um möglichst wenig Code schreiben zu müssen. Sie benötigen eine `setSize`-Methode für `Box`. Implementieren Sie alle `Box`-Objekte so, dass Sie über `setPos` und `getPos` Zugriff auf den Mittelpunkt haben (das heisst, es wird nicht die linke obere Ecke als Referenz benutzt). Verwenden sie keine `public`-Instanzvariablen.

Aufgabe 2.7: Testen Sie Ihre Objekte

Schreiben Sie ein Test-Applet `DiagramApplet` und erzeugen Sie in Ihrer `init`-Methode ein paar Test-Objekte, die sie in `paint` zeichnen.

Diagrammeditor-Übung Teil 3

Aufgabe 3.1: Verbindungen

Bisher können Sie nur rechteckartige Elemente zeichnen. Überlegen Sie sich, wie ein Objekt `Line` aussehen könnte, welches zwei Rechtecke verbindet.

Aufgabe 3.2: Line-Objekte

Implementieren Sie eine Klasse `Line`, Unterklasse von `DiagramObject`, welche eine Anfangsbox `startbox` und eine Endbox `endbox` als Instanzvariablen besitzt. `Line` soll über einen Konstruktor `Line(Box start, Box end)` initialisiert werden. Die `draw`-Routine soll eine Linie vom Mittelpunkt der einen `Box` zum Mittelpunkt der anderen `Box` zeichnen.

Aufgabe 3.3: Testen der Linien

Testen Sie ihren Code, indem sie mehrere Objekte erzeugen und diese mit Linien verbinden. Was fällt Ihnen auf?

Aufgabe 3.4: Vorausplanung

Worauf müssen Sie achten, damit Verbindungsgeraden nicht über die Boxen gemalt werden? Merken Sie sich dies für morgen!

Aufgabe 3.5*: Mehr Objekte

Wenn Sie heute noch Zeit und Lust haben, können Sie weitere Diagramm-Objekte implementieren. Leiten Sie alle Objekte entweder von `Box` oder `Line` ab.

Aufgabe 3.6*: Mehr Eigenschaften

Sofern Sie es noch nicht getan haben, können Sie den Objekten mehr Eigenschaften verpassen: Farbe, Liniendicke, Font, ...

Benutzen Sie keine `public`-Variablen und stellen Sie für jede Eigenschaft `get-/set`-Paare zur Verfügung (damit sind Ihre Objekte *zukunftsorientiert!*).

Diagrammeditor-Übung Teil 4

Die Aufgaben 4.1 bis 4.3 sollten gleichzeitig, aber schrittweise, bearbeitet werden.

In diesem Teil sollen die Grundlagen für ein Userinterface für den Diagrammeditor geschaffen werden. Da wir diverse Eingabekomponenten benötigen, müssen wir zunächst aus dem monolithischen Applet eine “Diagramm-Komponente” machen.

Aufgabe 4.1: Ein Canvas

Trennen Sie die Zeichenroutine aus dem Applet heraus, indem Sie eine Unterklasse `DiagramCanvas` von `Canvas` schreiben. Setzen Sie das Layout des Applets auf `BorderLayout` und fügen Sie eine Instanz des `DiagramCanvas` mit `add("Center", ...)` dem Applet (als `Container`) hinzu.

Aufgabe 4.2: Speichern der Objekte

Die überschriebene `paint`-Methode des Applets wandert nun in das `DiagramCanvas`. Wie weiss nun das `DiagramCanvas` von den in `init` erzeugten Objekten? Schreiben Sie Methoden `addObject` und `removeObject` für `DiagramCanvas`, mit denen Sie zu zeichnende `Diagramm`-Objekte hinzufügen können und entfernen können. Die Objekte soll das `DiagramCanvas` speichern, zum Beispiel in einem `java.util.Vector` (welcher natürlich `private` deklariert ist).

Aufgabe 4.3: Anpassen von paint

Die `paint`-Routine muss nun natürlich angepasst werden: Durchlaufen Sie den `Vector` und zeichnen Sie jedes einzelne Objekt mit einem `draw`-Aufruf. Achtung: Dafür ist ein `cast` nach `DiagramObject` notwendig (jetzt ist klar, warum wir diese Klasse benötigten). Bedenken Sie auch, wie Sie Ihre Erkenntnis aus 3.4 umsetzen können.

Aufgabe 4.4*: Testen mehrere DiagramCanvas-Objekte

Probieren Sie mit einem `GridLayout`-Objekt aus, ob sie mehrere `DiagramCanvas`-Instanzen erzeugen und darstellen können. Wie verhalten sich diese? Können Sie ein `DiagramObject` mehrmals verwenden?

Diagrammeditor-Übung Teil 5

In diesem Teil werden Events eingeführt.

Aufgabe 5.1: Wer ist denn da?

Um interaktiv in ein Diagramm eingreifen zu können, muss es möglich sein, einem Mausklick an einer Stelle der `DiagramCanvas` ein Objekt, welches sich dort befindet, zuzuordnen. Da wir zunächst nur `Box`-Objekte bewegen möchten, muss diese Identifizierungsmöglichkeit für jedes `Box`-Objekt bestehen, genauer: Jedes `Box`-Objekte muss in der Lage sein, auf die Frage "Bist Du an der Stelle (x,y)" mit ja oder nein zu antworten. Schreiben Sie eine entsprechende Methode `public boolean isHot(int x, int y)`. Wie oft müssen Sie diese schreiben? Bemerkung: Es ist OK, wenn ein Objekt zumindest einmal gezeichnet wurde, bevor es `isHot`-Anfragen korrekt beantwortet.

Aufgabe 5.2: Identifikation von Objekten

Da das `DiagramCanvas` als einzige Klasse Informationen über die dargestellten Objekte hat, wir aber später auch von anderen Klassen Zugriff auf die Objekte an der Mausposition haben wollen, benötigt das `DiagramCanvas` eine Methode `public DiagramObject isHot(int x, int y)`, die ein Objekt zurückliefert, welches sich an der Stelle (x,y) befindet. Liefert kein Objekt an dieser Stelle `true` für seine `isHot`-Methode, soll `null` zur zurückgegeben werden.

Aufgabe 5.3: Reagieren auf Mausklicks

Schreiben Sie nun zum Testen Ihrer Methoden eine Klasse `Identify`, die `MouseListener` (implementiert `MouseListener` mit leeren Methoden.) erweitert, und überschreiben Sie die Methode `public void mouseClicked(MouseEvent me)` so, dass das Objekt an der Mausposition mittels `System.out.println` ausgegeben wird. Hinweis: Die Mausposition erhalten Sie über die Methoden `getX()` und `getY()` von `Mausevent`. Vergessen Sie nicht, `java.awt.event.*` zu importieren!

Wenn Sie die Klasse implementiert haben, dann testen Sie sie, indem Sie im Applet eine Instanz von `Identify` erzeugen und mittels `addMouseListener` für das `DiagramCanvas` registrieren.

Aufgabe 5.4*: Überschreiben von `toString()`

Wenn Ihnen die Ausgabe von `System.out.println()` nicht gefällt, dann überschreiben Sie die Methode `public String toString()` von `Box`, `FilledBox` und `TextBox`.

Diagrammeditor-Übung Teil 6

Aufgabe 6.1: Bewegen Sie was!

Schreiben Sie eine neue Klasse `MoveListener`, welche `MouseListener` erweitert und dadurch das Interface `MouseListener` implementiert. Erweitern Sie nun `mousePressed` und `mouseReleased` so, dass beim Drücken der Maus das aktuelle Objekt gesucht (über `isHot`) und abgespeichert (in einer Instanzvariable) wird und beim Loslassen dieses Objekt die neue Mausposition erhält (über `setPos`). Um `isHot` benutzen zu können, müssen Sie wissen, auf welcher Komponente der `MouseEvent` stattfand, dies erfahren Sie mit `getComponent`, dessen Rückgabewert dann auf `DiagramCanvas` gecastet werden kann.

Aufgabe 6.2: Testen Sie das Bewegen

Nun versuchen Sie etwas zu bewegen! Tut sich etwas? Wenn nicht, dann liegt das wahrscheinlich daran, dass das `DiagramCanvas` nicht über die Veränderung informiert wird. Ändern Sie dies, indem Sie an der richtigen Stelle einen Aufruf der `repaint`-Methode des `DiagramCanvas` einfügen.

Aufgabe 6.3: Verbessern Sie das Bewegen

Es ist unbefriedigend, dass man während der Bewegung nicht den aktuellen Zustand der Zeichnung sieht. Erweitern Sie die Klasse `MoveListener` um die Methoden

```
public void mouseDragged(MouseEvent me) und
```

```
public void mouseMoved(MouseEvent me).
```

Dadurch wird das Interface `MouseMotionListener` implementiert. In der Methode `mouseDragged` können Sie nun zusätzlichen Code einfügen, der den aktuellen Zustand der Zeichnung zeigt — er unterscheidet sich nur geringfügig von dem Code in `mouseReleased`!

Aufgabe 6.4: Testen Sie erneut!

Vergessen Sie nicht, den `MoveListener` auch als `MouseMotionListener` zu registrieren, sonst sehen Sie Ihre Änderungen nicht!

Aufgabe 6.5*: Double-Buffering

Wenn Ihnen das Flackern nicht gefällt, dann implementieren Sie ein Double-Buffering.

Diagrammeditor-Übung Teil 7

Aufgabe 7.1: Erzeugen von Boxen

Schreiben Sie eine Klasse `AddBox`, welche ähnlich wie `MouseListener` die beiden Interfaces `MouseListener` und `MouseMotionListener` implementiert. Allerdings soll ein `mousePressed`-Event ein `Box`-Objekt an der Mausposition erzeugen, welches in `mouseDragged` und `mouseReleased` über `setSize` in der Grösse angepasst wird.

Aufgabe 7.2: Erzeugen von `FilledBox` und `TextBox`

Überlegen Sie, was sich am Code von `AddBox` ändert, wenn Sie nicht ein `Box`-Objekt erzeugen wollen, sondern ein `FilledBox`-Objekt. Ändern Sie `AddBox` so ab, daß der `new Box()`-Aufruf durch einen Aufruf der Methode `getObject()` ersetzt wird. Diese Methode schreiben Sie ebenfalls, sie liefert ein neues `Box`-Objekt. Jetzt können Sie `AddFilledBox` und `AddTextBox` mit wenigen Zeilen als Unterklasse von `AddBox` schreiben. Stören Sie sich zunächst nicht daran, dass der Text der `TextBox` statisch festgelegt werden muss. Dies ändern wir später.

Aufgabe 7.3: Testen

Probieren Sie die beiden neuen Listener aus, indem Sie die `addXListener`-Aufrufe ändern. Was ist das Problem beim Hinzufügen einer `TextBox`?

Aufgabe 7.4: Ändern des Verhaltens der `TextBox`

Eine `TextBox` hat eine feste Grösse, daher soll sie beim Hinzufügen verschoben werden statt in der Grösse verändert. Wie können Sie das mit minimalem Aufwand erreichen?

Diagrammeditor-Übung Teil 8

Aufgabe 8.1: Wählen des Listeners

Kopieren Sie sich die Dateien `MyCheckbox.java` und `MyCheckboxGroup.java` von <http://www.inf.fu-berlin.de/lehre/WS00/java>. Erzeugen Sie in Ihrem Applet ein `Panel`, dem Sie drei Instanzen von `MyCheckbox` hinzufügen, welche jeweils einen der drei Listener aus Teil 7 im Konstruktor übergeben bekommen. Erzeugen Sie eine Instanz von `MyCheckboxGroup`, die als Argument im Konstruktor die `DiagramCanvas` erhält, und setzen Sie die `CheckboxGroup` der drei Checkboxes auf diese Gruppe. Fügen Sie das Panel in den Norden, Süden, Westen oder Osten Ihres Applets ein. Entfernen Sie alle `addXListener`-Aufrufe im Applet.

Aufgabe 8.2: Testen Sie die verschiedenen Listener!

Können Sie nun Boxen aller Art hinzufügen? Wenn ja, dann erweitern Sie Ihr User-Interface um den schon geschriebenen `MoveListener` mit einer weiteren `Checkbox`.

Aufgabe 8.3: Hinzufügen von Linien

Nun sind sie soweit, dass Sie Linien zwischen den Objekten hinzufügen können. Schreiben Sie einen Listener `AddLine`, der diese Funktionalität bietet, und erweitern Sie das User-Interface. Wie gehen Sie mit Linien um, die nicht an `Box`-Objekten beginnen und enden? Behelfen Sie sich mit Boxen der Größe (0,0). Sie benötigen wahrscheinlich auch Zugriffsroutinen auf Start- und End-Box einer Linie.

Aufgabe 8.4*: Entfernen von Objekten

Wie kann ein Entfernen von Objekten in das bestehende Design eingefügt werden? Überlegen Sie, was mit Linien passiert, wenn ihre Boxen gelöscht werden, und wie diese Konsequenzen implementiert werden können. Wie kann man Linien direkt löschen? Wenn Sie die Problematik überdacht haben, implementieren Sie einen `DeleteListener` — Achtung, sie müssen eventuell recht viel bestehenden Code ändern.

Diagrammeditor-Übung Teil 9

Aufgabe 9.1: Setzen von Texten

Fügen Sie Ihrem Panel ein `TextField` hinzu. Dieses soll benutzt werden, um den Text von erzeugten Boxen zu setzen. Dazu muss der `AddTextBox-Listener` über das `TextField` informiert werden, ergänzen Sie einen entsprechenden Konstruktor! Dann ändern Sie die `getObject`-Methode dahingehend ab, dass das `TextField` nach seinem Text gefragt wird, um den Text einer gerade erzeugten `TextBox` festzulegen.

Aufgabe 9.2*: Ändern von Texten

Warum kann man nicht die `ActionEvents` des `TextField` benutzen, um den Text einer `TextBox` zu verändern. Überlegen Sie sich eine andere Möglichkeit!

Aufgabe 9.3*: Selektieren von Objekten

Eine Methode zum Ändern von Objekteigenschaften ist ein Selektionsmechanismus. Dazu gehört, dass selektierte Objekte anders am Bildschirm dargestellt werden. Wie können Sie das bisherige Programm abändern, um dies zu erreichen? Wenn Sie dies implementiert haben, dann lösen Sie Aufgabe 9.2 eleganter!

Aufgabe 9.4: Editierfenster**

Eine andere Methode zum Ändern von Objekteigenschaften ist ein Editiermodus: Wenn ein Objekt angeklickt wird, erscheint ein neues Fenster, in dem die Eigenschaften des Objektes eingestellt werden können. Implementieren Sie solch einen Editiermodus, indem Sie jedem `Box`-Objekt eine `editProperties`-Methode verschaffen, die ein Fenster öffnet, welches die Eigenschaften des Objekts darstellt und editieren lässt. Der `EditListener` muss dann nur noch bei einem `mouseClicked` die Routine des richtigen Objektes aufrufen. Wenn der `EditListener` nur auf `Doppelclicks` (`me.getClickCount()`) reagiert, kann er global registriert werden und ist dann stets verfügbar.

Aufgabe 9.5: Speichern/Laden**

Schreiben Sie Routinen, die `Object-Serialization` verwenden, um ein Diagramm abzuspeichern.