Institut für Informatik
der Universität München
Prof. Dr. M. Hofmann
Dr. R. Matthes, M. Lange

WS 02/03

19. Dezember 2002

Practical 3 zur Vorlesung

Einführung in die Informatik: Programmierung und Software-Technik

Abgabe: spätestens 20.01.2003, 21:00 Uhr

Gegenstand dieses Practicals ist der Aufbau einer eigenen Hierarchie von Klassen sowie einer grafischen Benutzerschnittstelle (GUI = Graphical User Interface). Konkret stellen wir uns eine Situation aus dem E-Commerce vor: Ein Pizza-Service möchte seine Pizzen im Internet anbieten. Der Kunde soll im Wesentlichen durch Mausklicks seine Wünsche angeben können. Was wir hier nicht modellieren: Netzwerkfähigkeit, gleichzeitige Bedienung mehrerer Kunden, grafische Benutzerschnittstelle für die Angebotserstellung, Kundendatenverwaltung, Speicherung und Weiterleitung der Aufträge. Zudem verzichten wir auf die Bestellung von Getränken, Salaten, Nachspeisen etc. und kümmern uns nicht um Sicherheitsaspekte und die Plausibilität der Aufträge. Auch ist die Bedienerführung keineswegs auf Flexibilität hin optimiert.

Zur Bearbeitung brauchen Sie die teilweise programmierte GUI in Form der Klasse `Pizzeneingabe`. Diese können Sie sich von der WWW-Seite der Vorlesung herunterladen. Dazu finden Sie Rohdaten zu Pizzen in der Datei `rohdaten.txt`.

Damit das elektronische Abgabesystem automatisch überprüfen kann, ob Ihr Programm fehlerfrei kompiliert, muss Ihre Klasse einen bestimmten Namen haben. Für dieses Practical ist dies erwartungsgemäß

```
Practical3
```

Das bedeutet, dass in dem JAR-Archiv, welches Sie abgeben, eine Datei namens `Practical3.java` vorhanden sein muss. Diese Datei muss dann auch die `main`–Methode zum Ausführen Ihres Programmes enthalten. Im Übrigen ist der Name des Archivs, das Sie abgeben, egal.

Achten Sie darauf, dass *alle* Dateien außer den Standardbibliotheken, die von Ihrem Programm benutzt werden, in dem `jar`-Archiv vorhanden sind, also alle Klassen, die Sie für dieses Practical schreiben sollen. Das elektronische Abgabesystem erwartet, dass die Dateien im Archiv in keinen Unterverzeichnissen liegen. Ist dies der Fall, so kann es die Dateien nicht finden, und Sie erhalten eine Fehlermeldung bei der Abgabe.

Beachten Sie auch, dass über `import` nur Java-Bibliotheksklassen eingebunden werden und nicht solche Klassen, die bereits im gleichen Verzeichnis wie `Practical3.java` liegen.

Geben Sie nur über das elektronische Abgabesystem ab. Schicken Sie Ihre JAR–Archive **nicht** per Mail an uns. Schicken Sie überhaupt nichts an uns, denn auch wir können nur JAR-Archive über die Eingabemaske des elektronischen Abgabesystems dort eintragen.

Bei Problemen wenden Sie sich an die studentischen Hilfskräfte, die die Rechnerraumbetreuung durchführen. Deren Termine im CIP-Pool Sibirien bzw. Gobi ändern sich ab dem 21.12.2002. Bitte konsultieren Sie die WWW-Seite der Vorlesung vor einem Besuch. Beachten Sie, dass es keine Betreuung während der Weihnachtsferien gibt.

Auf der WWW-Seite der Vorlesung finden Sie einige Bilder, die das Aussehen der GUI während des Programmablaufs beispielhaft illustrieren.

Im folgenden wird beschrieben, welche Datenstrukturen "hinter der GUI" verwendet werden sollen, d. h., wie die in der GUI repräsentierten Informationen in Klassen zu organisieren sind. Dazu wird beschrieben, wie das Programm auf die Benutzereingaben in die GUI reagieren soll. Wie Sie dieses Verhalten genau realisieren, bleibt Ihnen überlassen. Wichtig ist aber, dass Sie sich dabei an die vorgegebenen Datenstrukturen halten und dass die GUI dem in der Datei `Pizzeneingabe.java` vorgegebenen Aufbau folgt. Konzentrieren Sie Ihre Kreativität auf die Programm-Logik, d. h. die schlüssige Reaktion der GUI auf die Benutzereingaben. Eine opulente Ausgestaltung der GUI (etwa mit Programmgeneratoren) würde Sie voraussichtlich den Bezug zu Ihrem Quellcode verlieren lassen. Im Gegensatz zu GUI's aus der Realität des E-Commerce sollten Sie diejenige in Ihrer Abgabe in jedem Detail verstehen können.

Ignorieren Sie auf keinen Fall den Abschnitt über die Kommentare!

1 Aufgabenstellung

1.1 Präambel

Die folgenden Beschreibungen sind nur ein Weg, die "Realität" in Java abzubilden. Halten Sie sich aber bitte an diese Vorgaben. Ansonsten wird Teamarbeit praktisch unmöglich (stellen Sie sich Ihre Korrektoren und die Übungsleiter als Teammitglieder vor).

1.2 Eine Klassenhierarchie für Pizzasorten

Ein `Pizzaprodukt` hat einen Namen und zwei Preise: für die große und die kleine Ausführung (siehe die gängigen Prospekte von Heimservices). Deklarieren Sie die entsprechenden Instanzvariablen als `private` und implementieren Sie daher passende `get-` und `set`-Methoden. Dazu noch die Methode `public String toString()`, die nur den Namen liefern soll (wird später gebraucht für die Beschriftung der `JComboBox`). Ein Konstruktor soll nicht implementiert werden, da auch keine Objekte der Klasse `Pizzaprodukt` gebildet werden sollen. Sie dient damit nur als Oberklasse der im Folgenden beschriebenen Klassen.[1]

Eine `Auflagenart` ist ein `Pizzaprodukt` (erbt also von `Pizzaprodukt`). Eine Auflagenart kann aus einem Namen und den beiden Preisen (Auflage auf eine große bzw. eine kleine Pizza) konstruiert werden.

Eine `Pizzaart` ist ein `Pizzaprodukt`, das zusätzlich eine Instanzvariable vom Typ `Auflagenart[]` hat, also einen Array von Pizza-Auflagen. Dieser soll `private` sein. Daher braucht es Methoden

```
public int getAuflagenzahl()
public Auflagenart getAuflagenart(int index)
public boolean addAuflagenart(Auflagenart auflagenart)
```

Diese sollen ausgeben, wieviele Auflagen bereits gespeichert sind bzw. die zum `index` gehörende Auflage ausgeben. Der Einfachheit halber nehmen wir an, dass immer die ersten `getAuflagenzahl()` Einträge des Arrays unsere gespeicherten Auflagenarten enthalten. `addAuflagenart` soll nur Auflagen abspeichern, die noch nicht in dem Array vorkommen und dann `true` zurückgeben, sonst aber `false`. (Falls der Array schon voll ist, genügt es, ebenfalls `false` zurückzugeben. Sie können aber auch verfahren wie in der Übung zur Implementierung von `LIFO`.) Weiter gebraucht wird eine Methode `public String alleAuflagen()`, die eine komma-separierte Liste aller gespeicherten Namen der Auflagen bildet. Der Konstruktor muß natürlich den Array initialisieren (mit einer konstanten Länge, z. B. 50, die als Konstante passend zu deklarieren ist). abstrakt (wie schon `Pizzaprodukt` selbst) Wir wollen aber nur Objekte der folgenden Unterklassen bilden: `BasisPizzaart`, die vorgegebene Pizzaarten darstellt und `KomponiertePizzaart` für vom Kunden zusammengestellte Pizzaarten.

[1] Man könnte auch `Pizzaprodukt` als abstrakte Klasse markieren mittels `public abstract class Pizzaprodukt` anstatt des üblichen `public class Pizzaprodukt`, siehe Horstmann Seiten 366f.

`BasisPizzaart` ist eine `Pizzaart`, die einen Konstruktor genau wie `Auflagenart` hat, also einen Namen und zwei Preise als Argumente verlangt.

`KomponiertePizzaart` erweitert ebenfalls `Pizzaart`. Wir modellieren, dass ein Kunde auf eine vorgegebene Pizzaart (eben ein Objekt der Klasse `BasisPizzaart`) noch weitere Auflagen wünschen kann—und die entstehenden Pizzaarten als Objekte der Klasse `KomponiertePizzaart`. Daher gibt es einen Konstruktor

```
public KomponiertePizzaart(BasisPizzaart basis)
```

Dabei sollen alle Auflagen der `basis` in den Array von Auflagen kopiert werden, ebenso die Preise. Der Name soll modifiziert werden, z. B. durch Anhängen der Zeichenkette " `mit:`". (Also erlauben wir dem Benutzer nicht, Fantasienamen für seine Pizza-Kreationen zu vergeben. Wir ernüchtern ihn auch nicht durch einen etwaigen Hinweis, dass seine Pizza dieselben Auflagen hat wie eine aus unserem Standard-Angebot, die dann üblicherweise billiger käme.) Wesentlich ist die neue Funktionalität der Methode

```
public boolean addAuflagenart(Auflagenart auflagenart)
```

Zusätzlich zu dem bei `Pizzaart` Gesagten soll der Name die neue Auflagenart enthalten, z. B. durch Anhängen deren Namens (passend mit einem Komma abgetrennt), und die beiden Preise sollen um den Preisaufschlag der neuen Auflagenart erhöht werden. (Bei `BasisPizzaart` handelte es sich ja immer um Festpreise, die sich nicht aus den einzelnen Auflagen errechnen.)

Bemerkung: Die uns vorliegende Implementierung der beschriebenen fünf Klassen hat (inklusive knapper Dokumentation) etwa 3800 Zeichen. Zum Testen empfiehlt sich eine Klasse, z. B. `Practical3`, mit einer Methode `main`, in die man dann z. B. Befehle aus der Datei `rohdaten.txt` schreibt.

1.3 Zwei Klassen für Bestellvorgänge

Ein `Bestellposten` "besteht aus" (d. h., hat als Instanzvariablen) einem Objekt der Klasse `Pizzaart` (der bestellten Pizzaart—egal, ob vorgegeben oder selbst zusammengestellt), zwei booleschen Variablen, die anzeigen, ob die große Variante der Pizza bestellt wurde bzw. ob man die Pizza als Calzone zubereitet haben möchte (mindestens ein Pizza-Service in München liefert alle seine Pizzen laut Prospekt auch in dieser Form) sowie der Anzahl, die man davon möchte. (Sofern die Anzahl 1 ist, kann man sich dies als eine "konkrete" Pizza vorstellen.) Der Konstruktor setzt all diese Instanzvariablen. Eine Methode `public double preis()` soll den Preis bestimmen, und `public String toString()` eine Darstellung dieses Postens in Klartext, die man in die Liefer-Rechnung übernehmen kann.

Ein `Warenkorb` wird konstruiert aus einer Kundennummer vom Typ `int`, d. h. der Konstruktor sollte die Signatur `public Warenkorb(int kundennummer)` haben. Die wichtigste Instanzvariable ist ein Array von Objekten aus `Bestellposten`, der im Konstruktor in passender Länge (man vergleiche die Behandlung in `Pizzaart`) instantiiert wird. Implementieren Sie Methoden

```
public boolean addBestellposten(Bestellposten posten)
public String toString()
```

Der boolesche Wert soll `false` sein, falls `posten` nicht mehr in den Array gepaßt hat (sofern Sie den Array nicht verlängern wollen wie bei `LIFO` in den Übungen), sonst `true`. Die Ausgabe von `toString()` soll `toString()` für jeden Bestellposten aufrufen (und Zeilenvorschub vornehmen) und schließlich den Gesamtpreis ausgeben.

Bemerkung: Die uns vorliegende Implementierung der beiden Klassen umfaßt etwa 1800 Zeichen. Setzen Sie Ihre Tests kontinuierlich fort, damit Sie nicht am Ende zuviele Fehlerquellen auf einmal in Betracht ziehen müssen.

1.4 Die grafische Benutzerschnittstelle

Besorgen Sie sich die Datei `Pizzeneingabe.java` von der WWW-Seite zur Vorlesung. Sie enthält vor allem eine Klasse `Pizzeneingabe`, die von `JFrame` erbt, also ein Fenster ist. Zudem implementiert sie das Interface `ActionListener`. Das bedeutet, dass eine Methode

```
public void actionPerformed(ActionEvent ereignis)
```

bereitsteht, die auf alle beobachteten Ereignisse reagiert. Allerdings ist diese Implementierung zunächst leer. Dazu weiter unten mehr.

Weiter befinden sich in der Datei zwei Hilfsklassen: `Beschliesser` und `Textausgabezone`. Die erste ist schlicht das Mittel der Wahl, damit man das Fenster in üblicher Weise schließen kann. Die zweite Klasse dient lediglich dazu, etwas weniger Code schreiben zu müssen, als wenn man direkt die Swingklasse `JTextArea` verwenden würde.

Der Konstruktor von `Pizzeneingabe` wird aufgerufen mit je einem Objekt des Typs `Auflagenart[]` bzw. `BasisPizzaart[]`, die die Vorgaben des Pizza-Service enthalten. Wenn Sie den Abschnitt 1.2 noch nicht umgesetzt haben, können Sie so ein Fenster auf der Basis leerer Implementierungen von `Auflagenart` und `BasisPizzaart` testweise erzeugen.

Achtung, Fortgeschrittene sollten den folgenden Abschnitt gleich bearbeiten, Unsicherere machen dies besser erst nach Bearbeitung des Rests der Aufgabe: Praktisch die gesamte GUI ist vorbereitet. Es fehlt nur eine Eingabemöglichkeit für die Anzahl, die am besten mit einer `JComboBox` realisiert wird, der schon die Zahlen von 1 bis etwa 9 vorgegeben werden: Methode `addItem`. Dazu sollte man mit der Methode `setEditable` eine optionale freie Eingabe erlauben. Dann soll das ganze Fenster, d. h. `this`, als Listener (= "Lauscher") gesetzt werden sowie die `JComboBox` zusammen mit einem `JLabel` in ein `JPanel` gesteckt und dies in das `JPanel mitte` eingefügt werden. Damit da Platz ist, muß das Layout anders gesetzt werden, nämlich `new GridLayout(8,1)` statt `new GridLayout(7,1)` an der betreffenden Stelle stehen. **Solange dies nicht realisiert ist, kann man in der Implementierung von `actionPerformed` hilfsweise die Anzahl immer auf 1 setzen.**

Die eigentliche Aufgabe ist es, die Methode `actionPerformed` so zu definieren, dass die Benutzereingaben sinnvoll interpretiert werden. Dazu werden Sie weitere Instanzvariablen in `Pizzeneingabe` brauchen und weitere Anweisungen in deren Konstruktormethode.

Das Verhalten soll etwa sein wie folgt: Zuallererst wird die Kundennummer verlangt (muß mit der Enter-Taste abgeschlossen werden). Da die ganze Oberfläche bereitsteht, wird dies dadurch angezeigt, dass mindestens die Auswahl der Basispizza inaktiv gemacht wird (von der Klasse `java.awt.Component` geerbte Methode `setEnable` für die entsprechende `JComboBox basisWahl` mit dem Argument `false` aufrufen; mit dem Argument `true` würde man die Auswahl aktivieren). Zudem soll auf alle anderen Ereignisse als die Eingabe der Kundennummer nicht reagiert werden. Dazu verwendet man z. B. eine Variable vom Typ `boolean`, die erst auf `true` gesetzt wird, wenn die Kundennummer registriert wurde. Man erhält sie selbst mittels der von `javax.swing.text.JTextComponent` geerbten Methode `getText()` von `JTextField kundenEingabe` durch Umwandlung eines Strings in eine Zahl. Dann kann ein `Warenkorb` konstruiert werden und die Auswahl der Basispizza wieder aktiviert werden. (Bemerkung: Manchmal kann es nötig sein, eine Komponente explizit neu zeichnen zu lassen mittels der von `java.awt.Component` ererbten Methode `repaint()` des entsprechenden Objekts. In unserer Muster-Implementierung war dies jedoch nicht erforderlich.)

Generell kann immer abgefragt werden, welche Basispizza bzw. Auflagenart gerade gewählt ist mit der Methode `getSelectedItem()` der Klasse `JComboBox`. Diese liefert allerdings nur ein `Object` zurück, das explizit in den richtigen Typ konvertiert werden muß. Auch für die Anzahl (die erst implementiert werden muß) gilt das.

Ebenfalls erhält man immer, ob die grosse Pizza gewünscht wird, mit der Methode `isSelected()` von `JRadioButton grossKnopf`, die von der Klasse `javax.swing.AbstractButton` geerbt wird. Gleiches gilt für die Wahl der Calzone und `JCheckBox calzoneKnopf`. Daher können Sie nach jeglicher Eingabe einen `Bestellposten` konstruieren und seine Textrepräsentation in `Textausgabezone`

`textFuerPosten` mittels der von `javax.swing.text.JTextComponent` geerbten Methode `setText` ausgeben, damit immer klar ist, welche Bestellung nun mittels des Knopfes mit Aufschrift "Pizza in den Warenkorb" vorgemerkt würde (Wegnehmen aus dem Warenkorb wollen wir hier nicht erlauben). Eine kleine Schikane: Der Aufruf von `toString()` wird für Objekte der Klasse `Bestellposten` erzwungen, indem man den leeren String mit diesem Objekt verkettet. Tut man dies nicht, so würde `setText` einen `Bestellposten` als Argument erhalten, was zu einem Compilerfehler führt.

Auch der aktuelle Warenkorb kann so immer ausgegeben werden in `Textausgabezone textFuerKorb`.

Wird eine Auflage ausgewählt, so muß die Basispizza festgehalten werden, also deren Auswahl inaktiv gemacht werden. Damit läßt sich dann eine `KomponiertePizzaart` konstruieren. Das sollte man aber nur bei der ersten Wahl einer Auflage tun, also sich in einer booleschen Variablen merken, ob man sie schon konstruiert hat und es dann nicht nochmals tun. Abhängig von der Ausgabe von `addAuflagenart` kann man dann in `Textausgabezone auflagenText` ausgeben, dass diese Auflage schon zur Pizza gehört. Wenn nicht, dann läßt sich der neue Name dieser Pizzaart ausgeben.

Wann immer eine Basispizza gewählt wird, kann man in `Textausgabezone basisText` die Liste der Namen der vorgegebenen Auflagen ausgeben.

Der Stornoknopf und der Knopf zum Einfügen in den Warenkorb lösen fast dieselbe Reaktion aus: Beim Einfügeknopf wird der aktuelle `Bestellposten` in den `Warenkorb` gelegt. Bei beiden Knöpfen wird die Komposition einer eigenen Pizzaart beendet und die Auswahl einer Basispizza aktiviert. Weiter sollen die zugehörigen Textfelder gelöscht werden (Methode `loesche()` in `Textausgabezone`) und (falls implementiert) die gewünschte Anzahl mittels der Methode `setSelectedItem` für die zugehörige `JComboBox` auf 1 zurückgesetzt werden.

Wird der Knopf "Bestellung abschließen" gedrückt, so sollen die Auswahlen deaktiviert und die zugehörigen Textfenster gelöscht werden. In `Textausgabezone textFuerKorb` soll eine vollständige Auftragsbestätigung erscheinen unter Nennung der Kundennummer. Natürlich soll das Fenster dann nicht sofort geschlossen werden. Aus `JPanel fussleiste` (in der die Knöpfe stehen) können mit der von `java.awt.Container` geerbten Methode `remove` die beiden Knöpfe `stornoKnopf` und `inKorbKnopf` entfernt werden. Der verbleibende Knopf kann mit der von `javax.swing.AbstractButton` geerbten Methode `setText` eine neue Aufschrift "Beenden" erhalten. Dass diese Umbenennung stattgefunden hat, sollte in einer booleschen Variablen gespeichert werden und bei erneuter Aktivierung dieses Knopfes sofort `System.exit(0)` aufgerufen werden.

Bemerkung: Die uns vorliegende vollständige Implementierung weist ungefähr 2700 Zeichen mehr auf als die Datei `Pizzeneingabe.java` im WWW.

1.5 Das Hauptprogramm

Das Hauptprogramm, d.h. die `main`–Methode, muss sich in der Datei `Practical3.java` befinden. Es soll einen Array von Auflagenarten und ein Objekt der Klasse `Basispizzaart[]` anlegen und damit die Konstruktormethode von `Pizzeneingabe` aufrufen. Damit das interessant wird, müssen diese beiden Arrays genug Daten enthalten, wie man sie z. B. aus der Datei `rohdaten.txt` erhält. Dann soll das erzeugte Fenster eine sinnvolle Titelzeile erhalten und angezeigt werden. Dazu ruft man die von `JFrame` weitervererbten Methoden `setTitle` und `show()` auf. Der eigentliche Ablauf geschieht dann innerhalb des Fensters.

1.6 Kommentare

Versehen Sie Ihr Programm und Ihre selbstgeschriebenen Klassen mit Kommentaren für Konstruktoren, Instanzvariablen und Methoden, so dass eine Dokumentation Ihrer Abgabe mit `javadoc` erzeugt werden kann. Das kann Ihnen selbst dienen, weil Sie dann auch alle ererbten Methoden angezeigt bekommen. Zur Wiederholung: Ein JavaDoc–Kommentar beginnt mit `/**` und endet mit `*/` und steht vor dem dazugehörigen Block.

1.7 Pizzeneingabe.java

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JCheckBox;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;

public class Pizzeneingabe extends JFrame implements ActionListener{

    private Auflagenart[] auflagenarten;
    private BasisPizzaart[] basisarten;

    private Container schreibflaeche;
    private JPanel fussleiste;
    private JButton stornoKnopf;
    private JButton inKorbKnopf;
    private JButton fertigKnopf;
    private JTextField kundenEingabe;
    private Textausgabezone basisText;
    private JComboBox basisWahl;
    private Textausgabezone auflagenText;
    private JComboBox auflagenWahl;
    private JRadioButton grossKnopf;
    private JRadioButton kleinKnopf;
    private JCheckBox calzoneKnopf;
    private Textausgabezone textFuerPosten;
    private Textausgabezone textFuerKorb;

    // hier werden noch Instanzvariablen für den Programmablauf gebraucht

    public Pizzeneingabe(Auflagenart[] auflagenarten,
                         BasisPizzaart[] basisarten){
        // Parameterübergabe
        this.auflagenarten = auflagenarten;
        this.basisarten = basisarten;



        // Grundsätzliches zum Fenster
        final int FENSTERBREITE = 700;
        final int FENSTERHOEHE = 700;
        setSize(FENSTERBREITE, FENSTERHOEHE);
        addWindowListener(new Beschliesser());
        schreibflaeche = getContentPane();

        // die Knöpfe unten
        fussleiste = new JPanel();
```

```
fussleiste.setLayout(new GridLayout(1,3));
stornoKnopf = new JButton("aktuelle Pizza stornieren");
inKorbKnopf = new JButton("Pizza in den Warenkorb");
fertigKnopf = new JButton("Bestellung abschließen");
stornoKnopf.addActionListener(this);
inKorbKnopf.addActionListener(this);
fertigKnopf.addActionListener(this);
fussleiste.add(stornoKnopf);
fussleiste.add(inKorbKnopf);
fussleiste.add(fertigKnopf);
schreibflaeche.add(fussleiste, "South");

JPanel mitte = new JPanel();
mitte.setLayout(new GridLayout(7,1));

// Kundennummer
JPanel kundenFeld = new JPanel();
JLabel kundenLabel = new JLabel("Bitte geben Sie zuerst Ihre"
                                +" Kundennummer an:");
kundenEingabe = new JTextField(20);
kundenEingabe.addActionListener(this);
kundenFeld.add(kundenLabel);
kundenFeld.add(kundenEingabe);
mitte.add(kundenFeld);

// Wahl der BasisPizza
JPanel basisFeld = new JPanel();
JLabel basisLabel = new JLabel("Bitte wählen Sie Ihre Pizza:");
basisFeld.add(basisLabel);
basisWahl = new JComboBox(basisarten);
basisWahl.addActionListener(this);
basisFeld.add(basisWahl);
basisText = new Textausgabezone(4,40);
basisFeld.add(basisText);
mitte.add(basisFeld);

// Wahl der zusätzlichen Auflagen
JPanel auflagenFeld = new JPanel();
JLabel auflagenLabel = new JLabel("Möchten Sie weitere Zutaten?");
auflagenFeld.add(auflagenLabel);
auflagenWahl = new JComboBox(auflagenarten);
auflagenWahl.addActionListener(this);
auflagenFeld.add(auflagenWahl);
auflagenText = new Textausgabezone(4,40);
auflagenFeld.add(auflagenText);
mitte.add(auflagenFeld);

// Wahl der Größe
JPanel groessenFeld = new JPanel();
JLabel groessenLabel = new JLabel("Welche Größe?");
groessenFeld.add(groessenLabel);
grossKnopf = new JRadioButton("JUMBO");
kleinKnopf = new JRadioButton("SINGLE",true);
grossKnopf.addActionListener(this);
kleinKnopf.addActionListener(this);
ButtonGroup groessenKnoepfe = new ButtonGroup();
groessenKnoepfe.add(grossKnopf);
groessenKnoepfe.add(kleinKnopf);
JPanel groessen = new JPanel();
groessen.setLayout(new GridLayout(1,2));
```

```
        groessen.add(grossKnopf);
        groessen.add(kleinKnopf);
        groessenFeld.add(groessen);
        mitte.add(groessenFeld);

        // Wahl, ob Calzone
        JPanel calzoneFeld = new JPanel();
        JLabel calzoneLabel =
            new JLabel("Möchten Sie Ihre Pizza als Calzone?");
        calzoneFeld.add(calzoneLabel);
        calzoneKnopf = new JCheckBox("als Calzone");
        calzoneKnopf.addActionListener(this);
        calzoneFeld.add(calzoneKnopf);
        mitte.add(calzoneFeld);

        // Anzahl
        // fehlt leider

        // Ausgabezone für Bestellposten
        JPanel zoneFuerPosten = new JPanel();
        JLabel postenLabel = new JLabel("aktueller Bestellposten");
        zoneFuerPosten.add(postenLabel);
        textFuerPosten = new Textausgabezone(2,40);
        zoneFuerPosten.add(new JScrollPane(textFuerPosten));
        mitte.add(zoneFuerPosten);

        // Ausgabezone für den Warenkorb
        JPanel zoneFuerKorb = new JPanel();
        JLabel korbLabel = new JLabel("aktueller Warenkorb");
        zoneFuerKorb.add(korbLabel);
        textFuerKorb = new Textausgabezone(4,40);
        zoneFuerKorb.add(new JScrollPane(textFuerKorb));
        mitte.add(zoneFuerKorb);

        schreibflaeche.add(mitte, "Center");

        // hier müssen noch einige Variablen für den Programmablauf
        // passend gesetzt werden
    }

    public void actionPerformed(ActionEvent ereignis){

        // hier muß die Logik des Programmablaufs hinein
    }
}
// Nur zum Schließen des Fensters
class Beschliesser extends WindowAdapter{
    public void windowClosing(WindowEvent vergisses){
        System.exit(0);}
}
// kleine Hilfe für die Ausgabe
class Textausgabezone extends JTextArea{
    public Textausgabezone(int x, int y){
        super(x,y);
        setLineWrap(true);
        setWrapStyleWord(true);
        setEditable(false);}
    public void loesche()
        {setText("");}
}
```