Bedingte Anweisungen
steuern unter Verwendung von Vergleichsoperationen den Programmablauf über Verzweigungen innerhalb eines Programms. Sie haben folgende Struktur:

```
if (boolscher ausdruck) {
        anweisungen ...
}
else {
        anweisungen...
}
```

Das Ergebnisses eines Vergleiches im boolschen ausdruck (z.B. x<y) wird evaluiert. Liefert der Vergleich in Abhängigkeit von den aktuellen Werten seiner Variablen das Resultat true (wahr) werden die Anweisungen direkt hinter dem Schlüsselwort if ausgeführt, anderenfalls verzweigt der Programmablauf direkt zu den Anweisungen hinter dem Kennwort else. Fehlt dieser zweite Teil, so wird im Ergebnis false (falsch) der geklammerte Anweisungsblock nach dem Schlüsselwort if übersprungen.
Mit der Kombination else if, bei der die else-Anweisung ihrerseits weitere if-Anweisungen enthält, ist eine Verschachtelung bedingter Anweisungen möglich, die eine Kette von Abfragen startet.

Zählschleifen
werden benutzt, um Teile eines Programms mehrfach auszuführen. Die entsprechende Anweisung wird mit dem Schlüsselwort for gestartet und hat folgende Struktur:

```
for (startwert; bedingung; inkrement) {
        anweisungen...
}
```

Der erste Ausdruck in der Parameterliste definiert einen Startwert zum Schleifenindex; dieser erhöht sich nach jedem Durchlauf um das Inkrement, bis die Bedingung in Form eines boolschen Ausdrucks, die vor Abarbeitung der Anweisungen geprüft wird, im Ergebnis false zum Abbruch der Schleife führt.

```
int i;
for (i=1; i <= 10; i = i ++) {
        System.out.print(i)
}
```

Bedingungsschleifen

while-Schleife:
Im Gegensatz zu Zählschleifen, bei welcher der aktuelle Wert eines Laufindex den Durchlauf steuert, wird eine while-Schleife so lange wiederholt, bis ein logischer Ausdruck das Resultat false liefert. Bedingungsschleifen haben folgende Struktur:

```
while (boolscher ausdruck) {
        anweisungen...
}
```

do-Schleife:
Anders als die while-Schleife, welche ihre Abruchbedingung am Anfang der Schleife prüft, findet bei der do-Schleife der Abruchtest am Ende statt; die Schleife wird also mindestens einmal durchlaufen. Sie hat folgende Struktur:

```
do {
        anweisungen...
} while (ausdruck);
```

z.B.

```
int i = 0;
do {
        System.out.print(i++);
} while (i <= 10);
```

Schleifen können mehrfach hierarchisch verschachtelt sein.

Mehrfachverzweigungen:

Eine switch-Anweisung ermöglicht die Verzweigung zu verschiedenen, unterschiedlich markierten Anweisungen. Sie hat folgende Struktur:

```
switch (ausdruck) {
        case konstante1:
                anweisungen1...
                break;
        case konstante2:
                anweisungen2...
                break;
        ...
        default:
                anweisungen...
}
```

Zunächst wird der Ausdruck vom Typ char, byte, short oder int hinter dem Schlüsselwort switch evaluiert. Stimmt dessen aktueller Wert mit einem der Werte überein, die den konstanten Ausdrücken hinter den case-Anweisungen entsprechen, wird das Programm an der Anweisung fortgeführt, die der zutreffenden case-Anweisung folgt. Diese Verzweigung läuft bis zum Ende der switch-Anweisung oder bis zum nächsten break; letztere Anweisung unterbricht den Ablauf durch Verlassen der switch-Anweisung. Wenn keiner der Werte übereinstimmt, verzweigt der Algorithmus auf die Anweisung hinter dem Schlüsselwort default.; fehlt diese, setzt der Programmablauf hinter der switch-Anweisung fort.

Fehlt eine break-Anweisung, so bedeutet dies, dass mehrere Werte des ausdrucks auf die gleiche Anweisung führen.

z.B.

```
switch (ausdruck) {
        case konstante1:
        case konstante2:
                anweisungen12...
                break;
        case konstante3:
        case konstante4:
                anweisungen34...
                break;
        default:
                anweisungen...
}
```

Das Schlüsselwort break kann auch genutzt werden, um for, while oder do-Schleifen zu verlassen.
Das Schlüsselwort continue dagegen, bewirkt, dass innerhalb einer Schleife an deren Ende gesprungen wird, um danach den nächsten Durchlauf zu starten.
break- und continue-Anweisungen können auch mit einer Markierung (Label) versehen werden, die anzeigt wohin der Sprung gehen soll; das Programm wird an der Anweisung fortgesetzt, die mit dem Label markiert ist.

z.B.

```
for (startwert; bedingung; inkrement) {
        anweisungen...
        continue markierung1;
        anweisungen...
}
anweisungen
markierung1:
        anweisung...
```

Abstract Windowing Toolkit (AWT) / Swing

Java stellt zwei Bibliotheken zur Oberflächenprogrammierung bereit (AWT und Swing). Benutzungsoberflächen (Graphical User Interface, GUI) sind eine Kombination von Komponenten (Dialogelemente, Schaltflächen, Kontrollkästchen, Beschriftungs-, Texteingabefelder, Menüs, Listen usw.) sowie Containern (Fenster, Panels). Die Steuerung von Komponenten durch den Anwender erfolgt i Allg. über Tastatur und Maus.
Im Paket AWT sind nur plattformunabhängige Basiskomponenten definiert. Die konkrete Darstellung am aktuellen Rechner nutzt spezifische Systemkomponenten (Peers), die das der Plattform (Windows, MacOS, Unix) entsprechende Erscheinungsbild erzeugen.
Ein Nachteil dieser Vorgehensweise besteht darin, dass sich das Erscheinungsbild einer Applikation auf verschiedenen Computersystemen in den Details unterscheiden kann.
Die Swing Bibliothek ist eine Erweiterung des AWT; sie ermöglicht wesentlich mehr Variationen im Design aller grafischen Komponten als das Grafik-Paket AWT. Swing stellt zusätzlich zu den Peer-Komponenten auch Methoden bereit, mit denen man Applets und Applikationen so entwerfen kann, dass jede Anwendung plattformübergreifend ein eigenes Erscheinungsbild (LookAndFeel) hat. Swing-Komponenten erlauben es also, grafische Benutzeroberflächen zu erzeugen, welche den Stil des Java-Entwicklungsumgebung repräsentieren, oder die den Komponenten des Betriebssystems auf dem aktuellen Rechner entsprechen. Methoden dazu stellt die zuständige Klasse javax.swing.UIManager zur Verfügung.

Das Erscheinungsbild des ausführenden Computers erzeugt der Algorithmus:

```
String laf = UIManager.getSystemLookAndFeelClassName( );
UIManager.setLookAndFeel(laf);
```

Ein plattformübergreifender Java-Stil (Metal) basiert auf den Programmzeilen:

```
UIManager.setLookAndFeel(
        UIManager.getCrossPlatformLookAndFeelClassName( ));
```

oder

```
UIManager.setLookAndFeel(
        " javax.swing.plaf.metal.MetalLookAndFeel");
```

Für eine Oberfläche im MacOS-Stil steht der Befehl:

```
UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.mac.MacLookAndFeel");
```

Obwohl es technisch möglich wäre, kann ein MacOS-Erscheinungsbild aus Copyright-Gründen nicht auf einem Windows-Rechner verwendet werden.

Ein wesentlicher Vorteil beider Pakete (AWT, Swing) ergibt sich aus der Bereitstellung von Klassen, die selbst die Programmierung äußerst komplexer Prozesse wie z.B. die Öffnung von Farbpaletten (JColorChooser) zur Farbwahl oder von Dialogfenstern zur Auswahl von Dateien (JFileChooser) mit wenigen Programmzeilen erlauben.

Die Swing-Klassen (Paket javax.swing) werden von älteren Java-Versionen nicht unterstützt. Basiskomponenten, für die es analoge Elemente auch im Paket java.awt gibt, sind:

- JFrame (nur Applikationen)
  ein Hauptfenster mit Rahmen und Titelleiste, welches vom Benutzer maximiert, minimiert oder geschlossen werden kann und das der Anzeige von Oberflächen dient
  Konstruktor:
  JFrame( )
  JFrame(java.awt.GraphicsConfiguration)
  JFrame(String)
  JFrame(String, java.awt.GraphicsConfiguration)
  Während in AWT-basierten Programmen Komponenten direkt in ein Frame eingefügt werden können, verwenden Swing-Programme einen Hauptcontainer (JPanel), welcher mit der Methode setContentPane() definiert wird und der in das Rahmenfenster JFrame zu integrieren ist. Dieser Hauptcontainer beinhaltet alle Komponenten u.U. auch in weiteren Container-Rangfolgen.
  Alle Inhalte werden erst mit dem Befehl setVisible(true) auf dem Desktop angezeigt.

- JPanel
  rahmenloser Container (Panel) innerhalb eines Fensters oder eines anderen Panels zur Zusammenfassung von Komponenten
    Unterteilung eines Fensters oder Applets in mehrere Bereiche, in denen weitere Komponenten angeordnet werden können
    vor Einfügung in einen übergeordneten Bereich (Container) ist Füllung mit Komponenten erforderlich
    zur Anzeige auf dem Desktop müssen die Panels in ein Frame bzw. Hauptcontainer eingefügt werden
  Konstruktoren:
  JPanel( )
  JPanel(java.awt.LayoutManager)
  JPanel(java.awt.LayoutManager, boolean)
  JPanel(boolean)

- JLabel
  Benutzerkomponente mit Text oder Bild(Icon)
  Konstruktoren:
  JLabel( )
  JLabel(String)
  JLabel(String, int)
  Jlabel(String, javax.swing.Icon, int)

- JButton
  interaktive Schaltfläche, Aktionen per Mausklick
  Konstruktor:
  JButton(String)
  JButton(String, javax.swing.Icon)
  Jbutton(javax.swing.Action)
  Jbutton(javax.swing.Icon)

- JCheckBox
  Kästchen mit oder ohne Label, 2-wertig (Haken bei Selektion, sonst ohne)
  Konstruktoren:
  JCheckBox( )
  JCheckBox(String)
  JCheckBox(String, javax.swing.Icon)
  JCheckBox(String, javax.swing.Icon, boolean)
  JCheckBox(String, boolean)
  JCheckBox(javax.swing.Action)
  JCheckBox(javax.swing.Icon)
  JCheckBox(javax.swing.Icon, boolean)

- JRadioButton
  Kreise mit Punkt bei Selektion, sonst ohne
  Konstruktoren:
  JRadioButton( )
  JRadioButton(String)
  JRadioButton(String, javax.swing.Icon)
  JRadioButton(String, javax.swing.Icon, boolean)
  JRadioButton(String, boolean)
  JRadioButton(javax.swing.Action)
  JRadioButton(javax.swing.Icon)
  JRadioButton(javax.swing.Icon, boolean)

- Drop-Down-Liste
  auswählbare Einträge aus einer Liste
  Die Methode addItem(Object) fügt der Liste Einträge hinzu.
  setEditable(true) macht aus der Liste eine Combo-Box, bei welcher der Nutzer einen Text ins Feld eintippt, anstatt einen Eintrag aus der Liste zu wählen.
  Konstruktoren:
  JComboBox( )
  JComboBox(java.util.Vector)
  JComboBox(javax.swing.ComboBoxModel)
  JComboBox(Object[ ])

- JTextField
  editierbares, einzeiliges Textfeld, Eingaben über Tastatur
  Konstruktoren:
  JTextField( )
  JTextField(int)
  JTextField(String)
  JTextField(String, int)
  JTextField(javax.swing.text.Document, String, int)

- JTextArea
  editierbares, mehrzeiliges Feld, Eingaben über Tastatur
  Konstruktoren:
  JTextArea(int, int)
  JTextArea(String)
  JTextArea(String, int, int)
  JTextArea(javax.swing.text.Document)
  JTextArea(javax.swing.text.Document, String, int, int)

- JOptionPane
  quittierbares Dialogfenster mit Hinweis, Warn- oder Fehlermeldung
  Konstruktoren:
  JOptionPane( )
  JOptionPane(Object)
  JOptionPane(Object, int)
  JOptionPane(Object, int, int)
  JOptionPane(Object, int, int, javax.swing.Icon)
  JOptionPane(Object, int, int, javax.swing.Icon, Object[ ])
  JOptionPane(Object, int, int, javax.swing.Icon, Object[ ], Object)
  Standarddialogfenster sind:
    - ConfirmDialog – Frage, Buttons Yes, No, Cancel
      Anzeige: showCofirmDialog(Component, Object)
    - InputDialog – Texteingabe
      Anzeige: showInputDialog(Component, Object)
    - MessageDialog – Nachricht
      Anzeige: showMessageDialog(Component, Object)

- JFileChooser
  Menü zur Auswahl einer Datei
  Konstruktoren:
  JFileChooser( )
  JFileChooser(java.io.File)
  JFileChooser(java.io.File, javax.swing.filechooser.FileSystemView)
  JFileChooser(String)
  JFileChooser(String, javax.swing.filechooser.FileSystemView)
  JFileChooser(javax.swing.filechooser.FileSystemView)

- JMenuBar, JMenu, JMenuItem, JPopupMenu
  Generierung von Menü-Leisten, Haupt- oder Untermenüs, Aufklappmenüs sowie von auswählbaren Menübefehlen

- Scrollbar (AWT)
  horizontale oder vertikale Bildlaufleisten
  Orientierung über die Klassenvariablen:
  Scrollbar.HORIZONTAL Scrollbar.VERTICAL

  Konstruktoren:
  Scrollbar( )
  Scrollbar(int)
  Scrollbar(int, int, int, int, int)
  Die Klasse ScrollBar von Swing kann nicht direkt aufgerufen werden; sie wird mittels JScrollPane, JSlider, JProgressBar initiert.

- JScrollPane
  scrollbarer Container, der beliebige Komponenten enthalten kann
  Konfiguration der Scrollleisten über
  die alternativen Klassenvaraiblen der Schnittstelle ScrollPaneConstants:
  VERTICAL_SCROLLBAR_ALWAYS
  VERTICAL_SCROLLBAR_AS_NEEDED
  VERTICAL_SCROLLBAR_NEVER
  Analoge Konstanten existieren für horizontale Scrollleisten.

  Konstruktoren:
  JScrollPane( )
  JScrollPane(int, int)
  JScrollPane(java.awt.Component)
  JScrollPane(java.awt.Component, int, int)

- JSlider
  Regler
  Konstruktoren:
  JSlider( )
  JSlider(int)
  JSlider(int, int)
  JSlider (int, int, int)
  JSlider (int, int, int, int)
  JSlider(javax.swing.BoundedRangeModel)

- JToolBar
  Werkzeugleiste
  Konstruktoren:
  JToolBar( )
  JToolBar(int)
  JToolBar(String)
  JToolBar(String, int)

- JTabbedPane
  Registerdarstellung (Container mit Registerseiten)
  Konstruktoren:
  JTabbedPane( )
  JTabbedPane(int tabPlacement)
  JTabbedPane(int tabPlacement, int tabLayoutPolicy)

Neben diesen Basiskomponenten existieren zusätzliche, komplexe Swing-Komponenten wie: JTree (Baumstrukturen zur Darstellung von Hierarchien) und JTable (Tabellen).

Layout Manager

Die Komponenten JFrame, JPanel, J ScrollPane, JMenu des Pakets javax.swing sind grafische Bereiche (Container), die auf dem Rechner des Benutzer dargestellt werden können. Die Container enthalten i. Allg. weitere Komponenten, unter Umständen auch in mehrfachen Container-Hierarchien. Mit einer Reihe von Klassen, zusammengefasst unter dem Begriff Layout-Manager, legt man fest, wie Komponenten im Container angeordnet werden, um ein gewünschtes Layout zu erreichen. Layout-Manager, deren Klassen im Paket java.awt abgelegt sind, definieren auch die Abmessungen des Containers. Jedes Panel verfügt über einen eigenen Layout-Manager.

- FlowLayout
  Standard-Layout-Manager
  zeilenweise Anordnung, nebeneinander von links nach rechts

- BorderLayout
  Der Container wird in 4 Randbereiche (North, South, East, West) und einen zentralen Bereich (Center) aufgeteilt. Er kann höchstens 5 Elemente (Components), aufnehmen; oft handelt es sich dabei um Container welche ihrerseits einzelne Komponenten enthalten. Die Komponenten werden wie bei allen Layout-Managern mit der Methode add( ) hinzugefügt.
  z. B.
  add ("North", Komponente)
  setzt die Komponente an den oberen Rand des Containers, mit minimaler Höhe und voller Container-Breite
  add ("South", Komponente)
  setzt die Komponente an den unteren Rand des Containers, mit minimaler Höhe und voller Container-Breite
  add ("West", Komponente)
  setzt die Komponente an den linken Rand des Containers, mit minimaler Breite und der vollen Höhe des Bereiches.
  add ("East", Komponente)
  setzt die Komponente an den rechten Rand des Containers, mit minimaler Breite und der vollen Höhe des Bereiches
  add ("Center", Komponente)
  setzt die Komponente in den zentralen Bereich, mit der vollen Höhe und Breite

  Beachten Sie: Es müssen nicht alle 5 Bereiche belegt werden.
  Das Argument Komponente kann auch für einen Container bzw. ein daraus abgeleitetes Objekt stehen.

- GridLayout
  Der Container ist ähnlich einer Tabelle in n Zeilen und m Spalten gleicher Größe aufgeteilt. Die Komponenten werden in der Reihenfolge der add-Befehle Zeile für Zeile in diese Struktur eingesetzt.

- GridBagLayout
  Erweiterung des GridLayout-Managers
  Komponenten über mehrere Zellen des Rasters
  unterschiedliche Proportionen der Reihen oder Spalten
  verschiedene Anordnungen der Komponenten in den Zellen

- CardLayout
  nur Anzeige der aktuellen Komponente analog zur obersten Karte eines Stapels bzw. zum projizierten Dia einer Serie, restliche Komponenten sind unsichtbar
  Normalerweise benutzt man ein Panel pro Karte. Die modifizierte add-Methode

  ```
  add (String, panel)
  ```

  fügt dem Panel eine Karte hinzu. Das erste Argument gibt den Namen der Karte an. Das 2. Argument legt das Panel fest, welches für die Karte steht; dieses Panel muss bereits alle Komponenten enthalten. Die Reihenfolge der add-Befehle ist relevant. Nachdem die Karten den Panels hinzugefügt wurden, werden letztere in den Container gelegt, dem das CardLayout zugeordnet ist.
  Die Methode show ( ) schließlich zeigt eine konkrete Karte des Stapels an. In Fortführung des obigen Beispiels würde der Befehl lauten:

  ```
  cards.show(container, kartenname);
  ```

  Die Anzeige der jeweils nächste Komponente (in der Reihenfolge der add-Befehle) löst der Befehl

  ```
  cards.next(container);
  ```

  aus, die zuvor angezeigte Karte wird verdeckt.
  Die Befehle

  ```
  cards.first(container);
  cards.last(container);
  ```

  zeigen die erste bzw. letzte Komponente des Kartenstapels an.

- javax.swing.BoxLayout
  Anordnung verschieden großer Komponenten in Zeilen, Spalten oder Gitter-Strukturen

Erzeugt wird ein Layout-Manager mit dem Operator new;
mit der Methode setLayout ( ) weisen Sie das gewählte Layout einem Container zu.
z. B.

```
public class Starter extends javax.swing.JApplet {
    FlowLayout lm = new FlowLayout( );

    public void init( ) {
        JPanel container = JPanel( )
        container.setLayout(lm);
```

```
                setContentPane(container)
                // Das Panel wird zum Inhaltsbereich des übergeordneten,
                // anzeigenden Applets
        }
}
```

Erst nachdem der Layout Manager definiert wurde, können Komponenten in den zugehörigen Container eingefügt werden.
z. B.

```
JFrame f = JFrame("Titelzeile");
f.setLayout( new FlowLayout( ) );
JButton b = JButton("Bitte drücken");
f.add(b);
```

Der spezifische Befehl setLayout(null) bewirkt, dass kein vorgegebener Layout–Manager verwendet wird; der Programmierer muss alle Komponenten selbst mit den Methoden setLocation(int x, int y) und setSize(int width, int height) oder setBounds(int x, int y, int width, in height) der Klasse java.awt.Component im Container positionieren.

Eine Standardaufgabe ist das Einfügen eines grafischen Objektes in ein vorgegebenes Layout; dies geschieht i. Allg. nach folgendem Muster:

```
...
setLayout(new FlowLayout);
bildanzeige=new(CBildLeinwand);
add(bildanzeige);
...
class CBildLeinwand extends JPanel {
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.drawImage(aktuelleGrafik, Xpos, Ypos, this):
        }
}
```

In Swing-Fenstern erzeugt man Zeichenflächen als Objekte, die von der Klasse JPanel abgeleitet sind. Swing-Klassen nutzen den Konstruktor der Methode paintComponent() stets in der Basisklassenversion (Schlüsselwort super), was die Voraussetzung für eine korrekte Darstellung des Hintergrundes der Komponente ist.

Interaktive Schaltflächen

Ereignisbehandlungen wie Aktionen mit der Maus werden nicht von Komponenten (z. B. Schaltfläche) selbst geleistet sondern von Steuerungsobjekten (Listener), welche den Komponenten zugeordnet werden müssen.

```
public void init( ) {

        JButton bp1 = JButton("push1");
        add(bp1);
        bp1.addActionListener(this);
        JButton bp2 = JButton("push2");
        add(bp2);
        bp2.addActionListener(this);
        public void actionPerformed(ActionEvent evtc) {
                if (evtc.getSource( )==bp1) {
                anweisungen1 ...
                }
                if (evtc.getSource( )==bp2) {
                anweisungen2 ...
                }
         }
}
```

Die Methode add.ActionListener(this) macht die Schaltflächen (bp1, bp2) aktiv. Das Schlüsselwort this verweist in diesem Zusammenhang auf das aktuelle Objekt, für das die Methode aufgerufen wurde. Dieses Vorgehen ist dann notwendig, wenn im Quelltext kein spezifischer Empfänger in Form einer Objektvariablen benannt wurde; die aktuelle Komponente wird als Empfänger registriert. this ist in diesem Fall eine synonyme Adresse für den Listener; es handelt sich um eine Referenzvariable, die beim Anlegen eines Objekts automatisch erzeugt wird. Die Methode actionPerformed( ) der Schnittstelle ActionListener definiert die Reaktionen, die das Ereignis (Mausklick) auslösen soll.
Die Identität der Schaltfläche, auf die geklickt wurde, ermittelt die Methode getSource( ) der Klasse ActionEvent.