Programmierkurs Java

UE 10 – Klassen und Objekte II

Dr.-Ing. Dietrich Boles

Gliederung

- Klassendefinition (Erweiterung)
- Objekte
- Nutzung von Klassen
- Beispiel Gewicht
- Definitionen

Klassendefinition (Erweiterung)

```
<klassen-def> ::= ["public"] "class" <bezeichner> "{"
                    { <attr-def |
                      <konstruktor-def> |
                      <methoden-def> | ...
                    }
                  "}"
<attr-def>        ::= <variablen-def>
<methoden-def>    ::= <funktionen-def> (ohne static)
<konstruktor-def> ::= <methoden-def> (ohne Funktionstyp)
```

Anmerkungen:

- Bezeichner: **Klassenname** (neuer Typ!)
- **Klasse = Zusammenfassung von Attributen und Funktionen/Methoden**

Klassendefinition / Attribute

- Gültigkeitsbereich: der gesamte Klassenblock
- Lebensdauer: entspricht Lebensdauer des Objektes
- jedes Objekt einer Klasse hat dieselben Attribute
- die Wert der Attribute können jedoch verschieden sein!

```
class Mensch {

  // Attribute
  boolean maennlich;
  int groesse; // in cm
  double gewicht; // in kg
```

- spezielle Funktion/Methode zur Initialisierung der Attribute eines Objektes
- Funktionsname = Klassenname
- kein Funktionstyp (auch nicht `void`)
- wird bei der Erzeugung eines Objektes aufgerufen

```
// Konstruktor
Mensch(boolean mann, int groe, double gew) {
  maennlich = mann;
  groesse = groe;
  gewicht = gew;
}
```

Klassendefinition / Methoden (1)

- wie Prozeduren / Funktionen, jedoch ohne `static`
- können auf Attribute und (andere) Methoden zugreifen

```
// Methoden
void wachsen(int cm) {
  groesse += cm;
}

void zunehmen(double kg) {
  gewicht += kg;
}

boolean hatOrdentlichesGewicht() {
  return gewicht <= berechneNormalgewicht() &&
         gewicht >= berechneIdealgewicht();
}
```

Klassendefinition / Methoden (2)

```
double berechneNormalgewicht() {
  return groesse - 100;
}

double berechneIdealgewicht() {
  if (maennlich) {
    return berechneNormalgewicht() / 100.0 * 90.0;
  } else {
    return berechneNormalgewicht() / 100.0 * 85.0;
  }
}
```

```
<objectvar-def> ::= <klassen-bezeichner>
                    <bezeichner>
                    { "," <bezeichner } 
                    ";"
```

Anmerkungen:

- Klassenbezeichner muss Name einer gültigen Klasse sein
- Bezeichner sind Objektvariablen ("Namen für Objekte")
- Objektvariablen sind Referenzvariablen
- Objektvariablen speichern Referenzen auf Objekte

Beispiele:

```
Mensch paul;
Mensch willi, maria, heike;
Mensch franz = null;  // expl. Initialisierung
```

```
<object-erz> ::= "new" <bezeichner>
                 "(" { <parameter-liste> } ")"
```

Anmerkungen:

- Bezeichner muss Name einer gültigen Klasse sein
- es muss ein "passender" Konstruktor existieren

Semantik:

- Es wird ein Objekt der entsprechenden Klasse "erzeugt" und ggfl. ein Konstruktor ausgeführt
- reserviert Speicherplatz für Objektattribute auf dem Heap
- Konstruktor: initialisiert Objektattribute
- liefert Adresse (Referenz) auf den Speicherbereich

Beispiele:

```
new Mensch(true, 181, 72.0)
new Mensch(false, 169, 55.0)
new Frau()                  // Fehler (unbekannte Klasse)
new Mensch(181)             // Fehler (ungültiger Konstruktor)
```

Objekte / Objektvariablen und Objekterzeugung

- häufig erfolgt die Objektvariablendefinition und die Objekterzeugung in einer Anweisung
- Zuweisungen möglich
- Typ der Objektvariablen muss gleich Typ des Objektes sein (Ausnahme: Polymorphie!)

Beispiele:

```
Mensch paul = new Mensch(true, 181, 72.5);
Mensch karl = new Mensch(true, 192, 83.0);
paul = new Mensch(true, 176, 68.3);
karl = paul;
```

```
class Tier { . . . }
Tier mops = new Mensch(false, 168, 54.0); // Fehler!
```

```
<meth-aufruf> ::= <bezeichner> "." <funktionsaufruf>
```

Anmerkungen:

- Bezeichner muss gültige Objektvariable sein
- Objektvariable muss auf ein Objekt verweisen
- "passende" Funktion muss in der Klasse des Objektes vorhanden sein

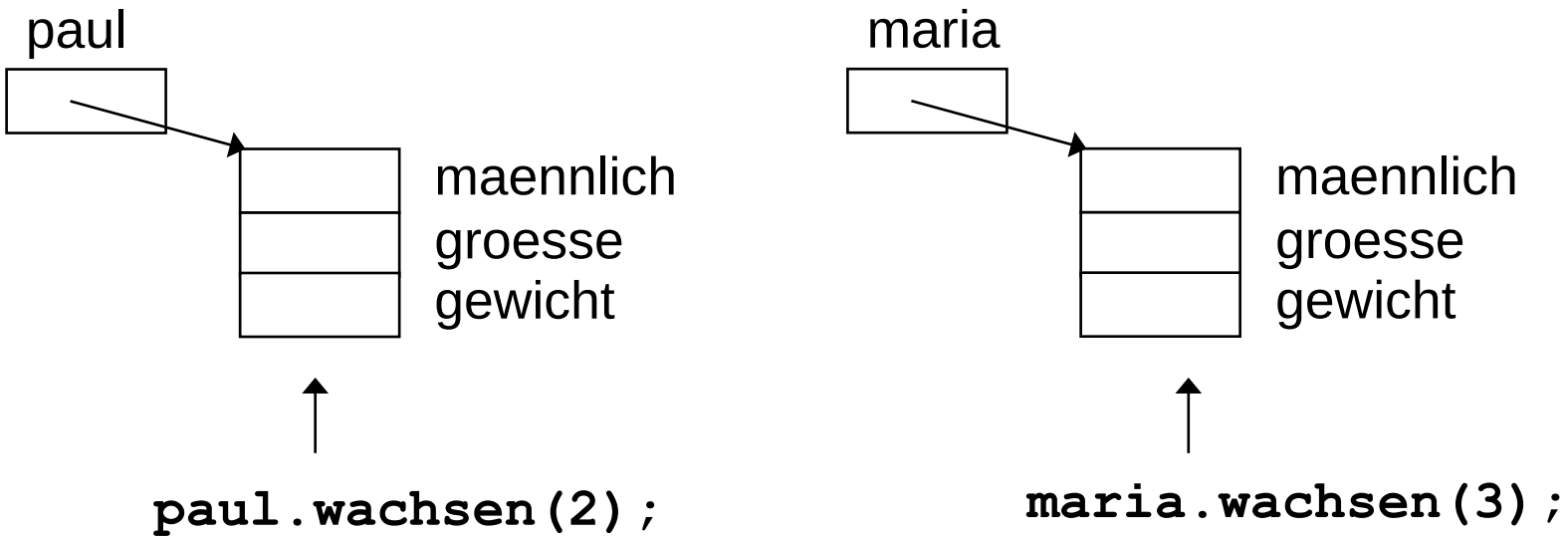
Semantik:

- die Funktion ("für das Objekt") wird aufgerufen
- sie "operiert" auf den Attributen des Objektes

Beispiele:

```
Mensch paul = new Mensch(true, 181, 72.0);
if (paul.hatOrdentlichesGewicht()) paul.zunehmen(2.3);

Mensch karl;
karl.wachsen(2);     // Laufzeitfehler (kein Objekt)
paul.schrumpfen();   // Fehler (fehlende Funktion/Methode)
paul.wachsen(3.0);   // Fehler (fehlende Funktion/Methode)
```

```
Mensch paul = new Mensch(true, 181, 72.0);
Mensch maria = new Mensch(false, 169, 54.5);
paul.wachsen(2);
maria.wachsen(3);
paul.zunehmen(2.5);
maria.zunehmen(1.0);
```



Nutzung von Klassen

Ab jetzt gilt das Prinzip der **Datenkapselung**:

- Von außerhalb einer Klassendefinition darf nur noch auf die Methoden eines Objektes zugegriffen werden, nicht mehr auf die Attribute!
- Ggfls. **Getter**/**Setter**-Methoden definieren

```
class Mitarbeiter {
   double gehalt;
   double getGehalt() { return gehalt; }
   void setGehalt(double geh) { gehalt = geh; }
}

class Firma { ...
   void streik() {
        for (Mitarbeiter m : mitarbeiter) {
            m.setGehalt(m.getGehalt() + 100);
            m.gehalt += 100; // verboten
   }   }
}
```

Problemstellung:

Das so genannte Normalgewicht berechnet sich nach der Formel

"Körpergröße (in cm) minus 100". Das Idealgewicht beträgt bei Männern 90% und bei Frauen 85% des Normalgewichts.

Schreiben Sie ein (objektorientiertes) Java-Programm, welches nach Eingabe von Größe, Gewicht und Geschlecht ausgibt, ob ein Mensch zu dick oder zu dünn ist, oder ob er/sie zwischen Ideal- und Normalgewicht liegt.

Beispiel Gewicht / Klassendefinition (1)

```
class Mensch {

  // Attribute
  boolean maennlich;
  int groesse; // in cm
  float gewicht; // in kg

  // Konstruktor
  Mensch(boolean mann, int groe, float gew) {
    maennlich = mann;
    groesse = groe;
    gewicht = gew;
  }
```

Beispiel Gewicht / Klassendefinition (2)

```
// Methoden
void wachsen(int cm) {
  groesse += cm;
}

void zunehmen(float kg) {
  gewicht += kg;
}

boolean hatUebergewicht() {
  return gewicht > berechneNormalgewicht();
}

boolean hatUntergewicht() {
  return gewicht < berechneIdealgewicht();
}

boolean hatOrdentlichesGewicht() {
  return gewicht <= berechneNormalgewicht() &&
         gewicht >= berechneIdealgewicht();
}
```

Beispiel Gewicht / Klassendefinition (3)

```
  // Hilfsmethoden

  float berechneNormalgewicht() {
    return groesse - 100;
  }

  float berechneIdealgewicht() {
    if (maennlich) {
      return berechneNormalgewicht() / 100.0f * 90.0f;
    } else {
      return berechneNormalgewicht() / 100.0f * 85.0f;
    }
  }
}
```

Beispiel Gewicht / Hauptprogramm

```
public class OOGewicht {
  public static void main(String[] args) {
    boolean isMann = IO.readChar("Maennlich (m/w)?") == 'm';
    int groesse = IO.readInt("Groesse (cm) eingeben: ");
    float gewicht = IO.readFloat("Gewicht (kg) eingeben: ");

    Mensch person = new Mensch(isMann, groesse, gewicht);
    if (person.hatUebergewicht()) {
      IO.println("Alter Fettsack!");
    } else if (person.hatUntergewicht()) {
      IO.println("Na, du Hungerhaken!");
    } else {
      IO.println("Idealer Gewichtsbereich!");
    }
  }
}
```

Definitionen (1)

Klasse:

beschreibt

- Eigenschaften (Attribute)
- Struktur (Subobjekte)
- Verhalten (Methoden)

einer Gruppe von gleichartigen Objekten (→ Datentyp)

Objekte (**Instanzen**):

- werden durch Klassen beschrieben
- setzen sich demnach zusammen aus
  - Datenelementen (Attribute) → Eigenschaften / Struktur / Zustand
  - den auf den Attributen ausführbaren Operationen → Verhalten
- Objekte einer Klasse haben gleiche Attribute und gleiche Funktionen; sie unterscheiden sich nur in den Werten ihrer Attribute

Definitionen (2)

Attribut (Instanz-Variable):

- Variable, für die in jedem Objekt Speicherplatz reserviert ist
- Menge der Attribute eines Objektes repräsentiert Zustand eines Objektes

Methode:

- realisieren die auf Objekten einer Klasse anwendbaren Operationen

Konstruktor:

- spezielle Methode zur Initialisierung von Objekten

Instantiierung:

- Erzeugung von Objekten

Protokoll einer Klasse:

- Menge der von außen aufrufbaren Methoden einer Klasse

Zusammenfassung

- Klasse: Bauplan für gleichartige Dinge (Objekte)
- Eine Klasse definiert einen neuen Typ
- Methoden: Eine Klasse definiert u. a. Funktionen (Methoden), die für Objekte der Klasse aufgerufen werden können
- Objekte: Von einer Klasse können Objekte erzeugt werden
- Objektvariablen: Der Methodenaufruf erfolgt via der Punktnotation über Objektvariablen, die Referenzen auf Objekte speichern können
- OO-Programm: Aufruf von Methoden von/für Objekte
- Datenkapselung: Zugriff auf Attribute von außerhalb verboten

Programmierkurs Java

UE 10b – GUI-Programmierung

Dr.-Ing. Dietrich Boles

Gliederung

- Übersicht
- Aufbau einer GUI
- GUI-Klassen des Java-AWT
- Beispielprogramm

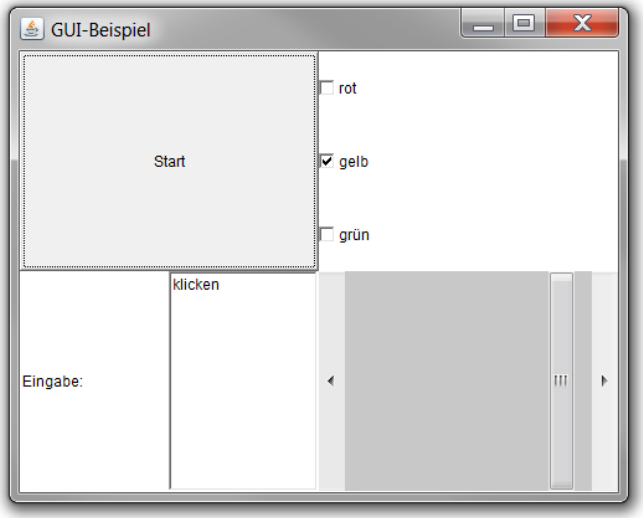
Übersicht

- **GUI**: Graphical User-Interface
- **GUI-Komponenten**: Fenster, Buttons, Scrollbars, …
- **GUI-Anwendung**: Anwendung mit einer GUI
- **GUI-Framework**: erweiterbare Klassenbibliothek für die Entwicklung von GUI-Anwendungen
- Java SE 12 besitzt 3 GUI-Frameworks:
  - **AWT** (Abstract Window Toolkit)
  - Swing
  - JavaFX (separat)
- Ablaufmodell der **invertierten Programmierung**:
  - Automatischer Start eines **Event-Dispatcher-Threads** (EDT)
  - EDT erkennt Events (Mausklicks, Tastatureingaben, …) auf GUI-Komponenten
  - EDT führt zugeordneten Code aus

Aufbau einer GUI

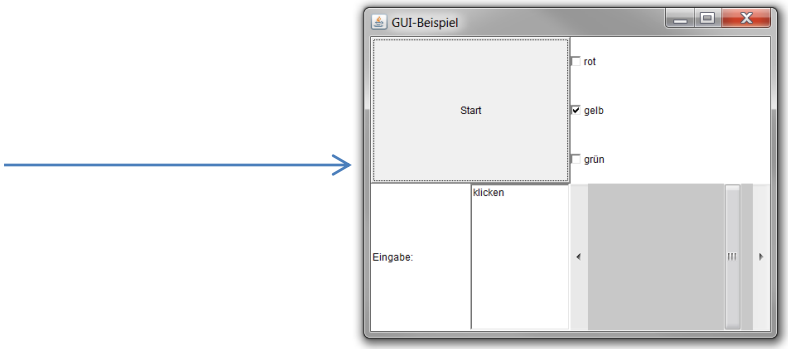
- Fenster (Frame) können aufnehmen
  - GUI-Komponenten (Component)
    - Button
    - Checkbox
    - Label
    - TextField
    - Scrollbar
  - Container: unsichtbare GUI-Komponenten, die andere UI-Komponenten aufnehmen können
    - Panel
- LayoutManager: kümmern sich um die Anordnung von GUI-Komponenten in Fenstern und Containern
  - FlowLayout: hintereinander
  - **GridLayout**:

    tabellenartig, Komponentengröße identisch (wird angepasst)



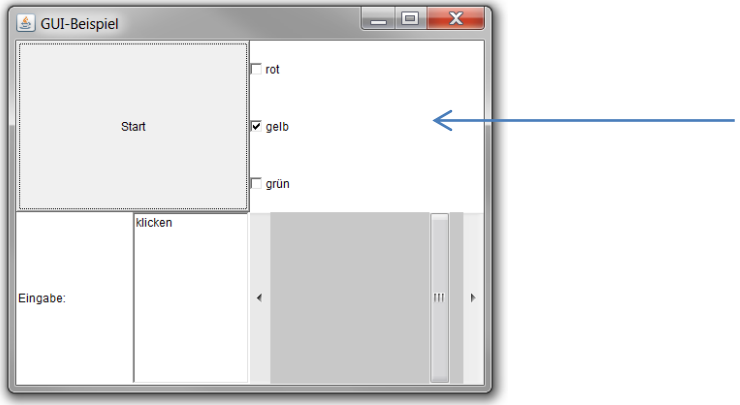
Klasse Frame

```
class Frame {  // realisiert ein Fenster auf dem Bildschirm
  Frame(String title)

  // setzen von Groesse und Bildschirmposition
  void setSize(int width, int height)
  void setLocation(int x, int y)

  // zuordnen eines Layout-Managers
  void setLayout(GridLayout layoutManager)

  // Methoden, mit denen dem Fenster bestimmte GUI-Komponenten zugeordnet
  // werden koennen
  void add(Panel comp) // Container
  void add(Label comp)
  void add(Button comp)
  void add(Checkbox comp)
  void add(TextField comp)
  void add(Scrollbar comp)

  // muss mit true aufgerufen werden, um das Fenster sichtbar zu machen
  void setVisible(boolean visible)
}
```



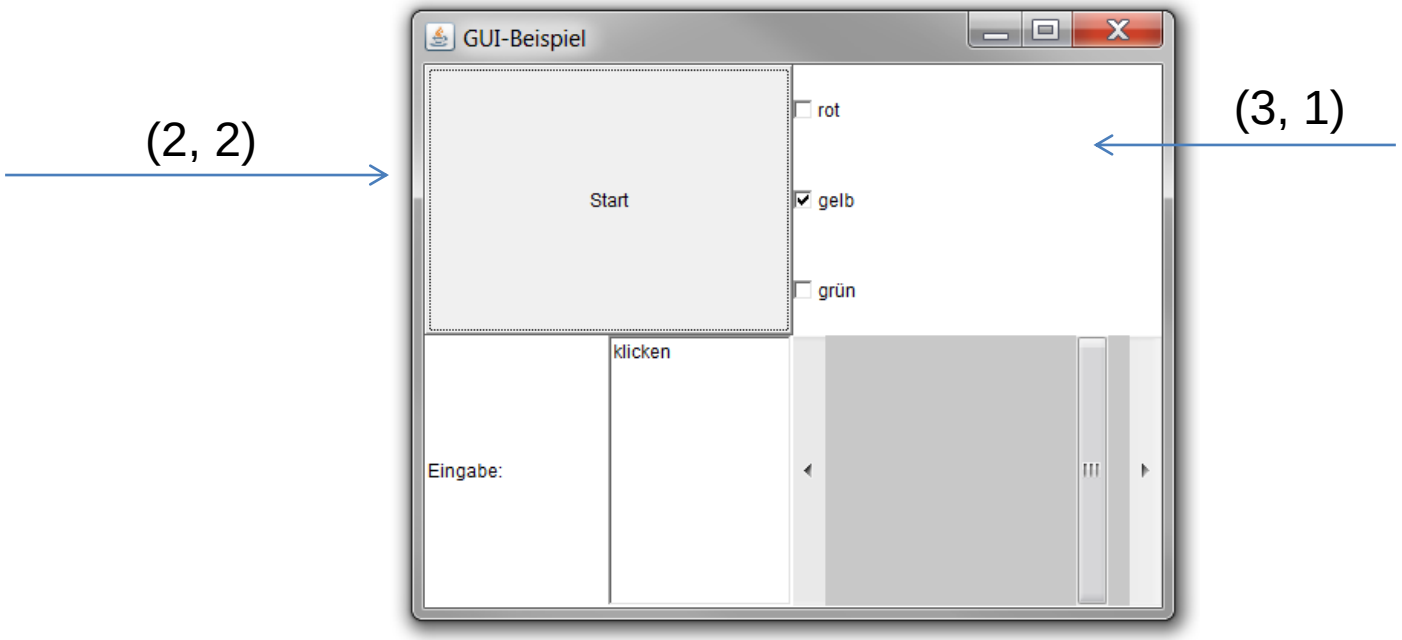
Klasse Panel

```
// realisiert einen nicht sichtbaren Container, der
// GUI-Komponenten aufnehmen und verwalten kann
class Panel {
  Panel()

  // zuordnen eines Layout-Managers
  void setLayout(GridLayout layoutManager)

  // Methoden, mit denen dem Panel bestimmte GUI-Komponenten
  // zugeordnet werden koennen
  void add(Panel comp)  // Container (hierarchischer Aufbau!)
  void add(Label comp)
  void add(Button comp)
  void add(Checkbox comp)
  void add(TextField comp)
  void add(Scrollbar comp)
}
```



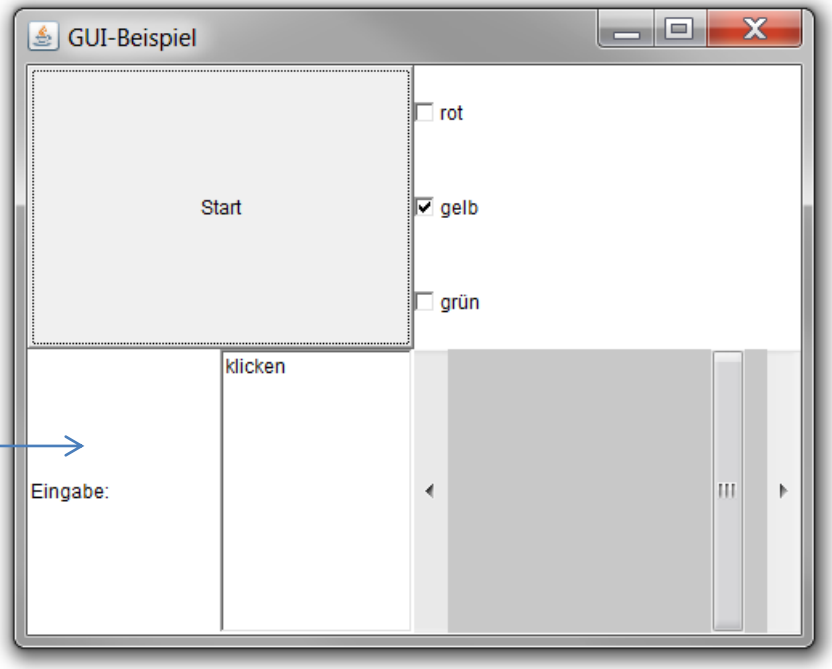
Klasse GridLayout

```
// realisiert einen speziellen Layout-Manager,
// der für eine tabellenartige
// Ausrichtung der Elemente sorgt
class GridLayout
    GridLayout(int rows, int cols)
}
```



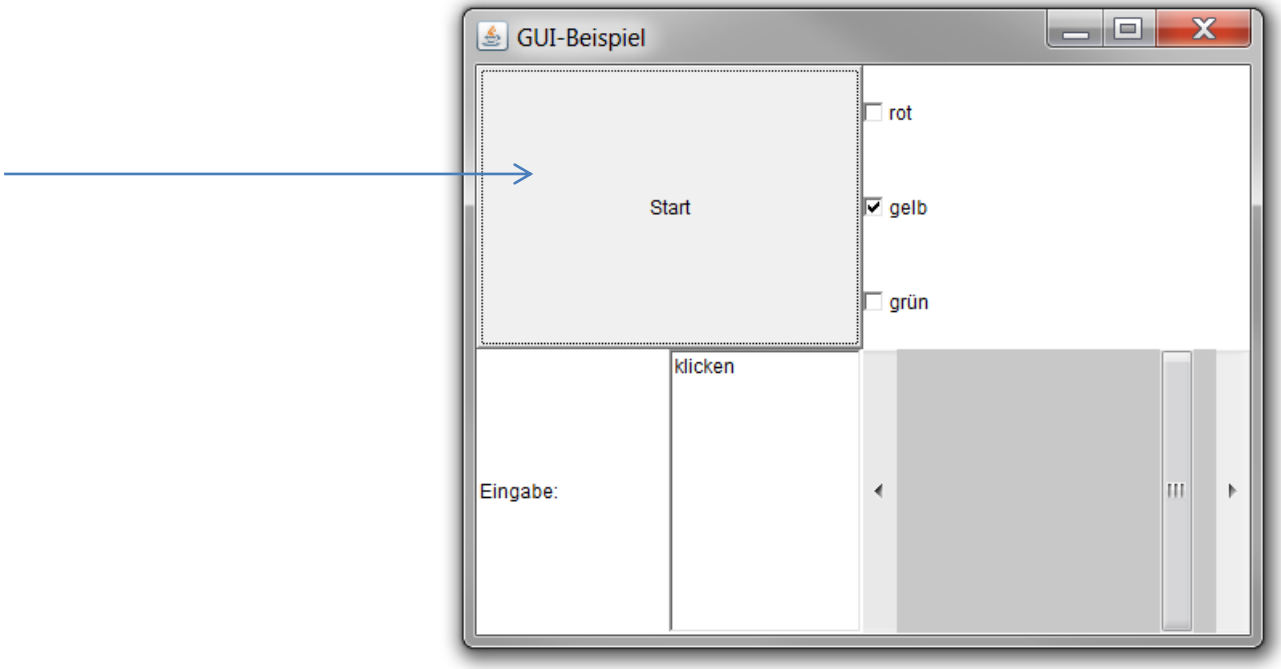
Klasse Label

```
// zur Darstellung von Texten
class Label {
  Label()
  Label(String label)
  void setLabel(String label)
  String getLabel()
}
```



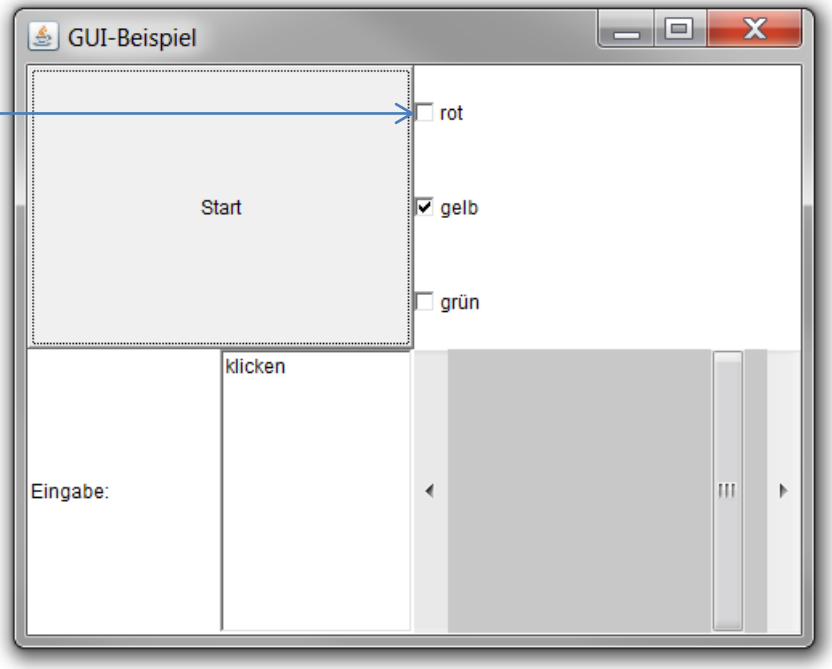
Klasse Button

```
// zur Darstellung von Buttons
class Button {
  Button()
  Button(String label)
  void setLabel(String label)
  String getLabel()
}
```



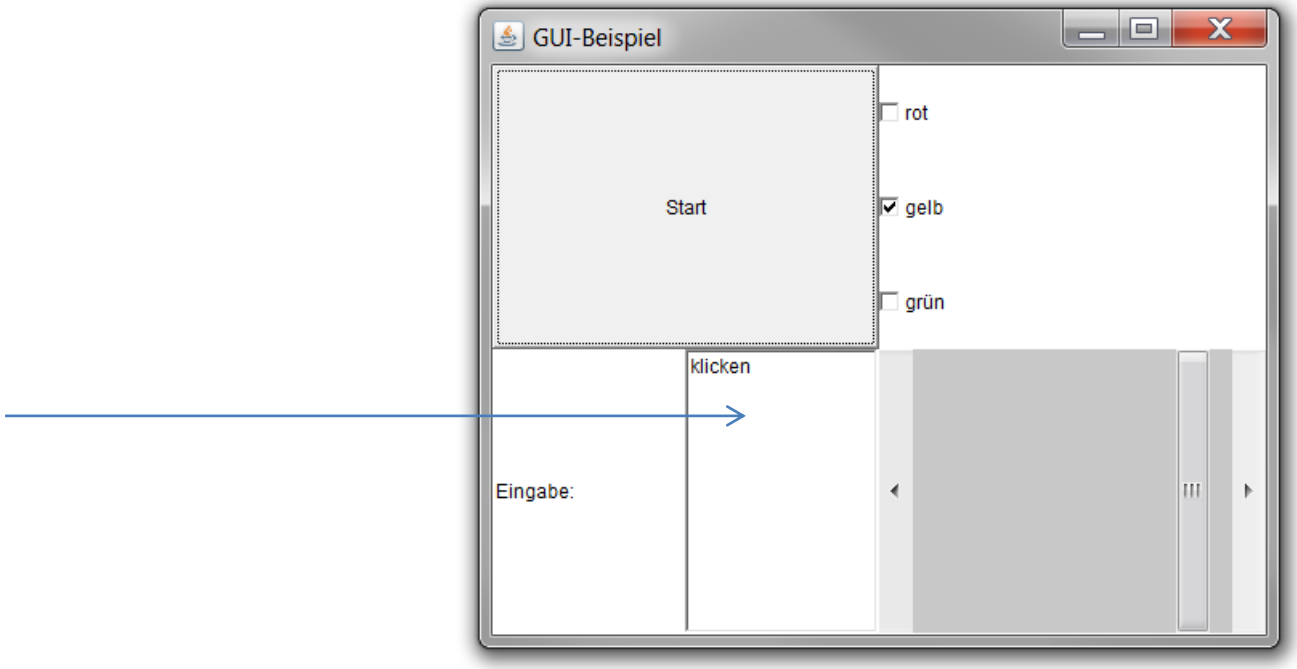
Klasse Checkbox

```
// zur Darstellung von Checkboxen
class Checkbox {
  Checkbox(String label)
  Checkbox(String label, boolean checked)
  void setLabel(String label)
  boolean getState()
}
```



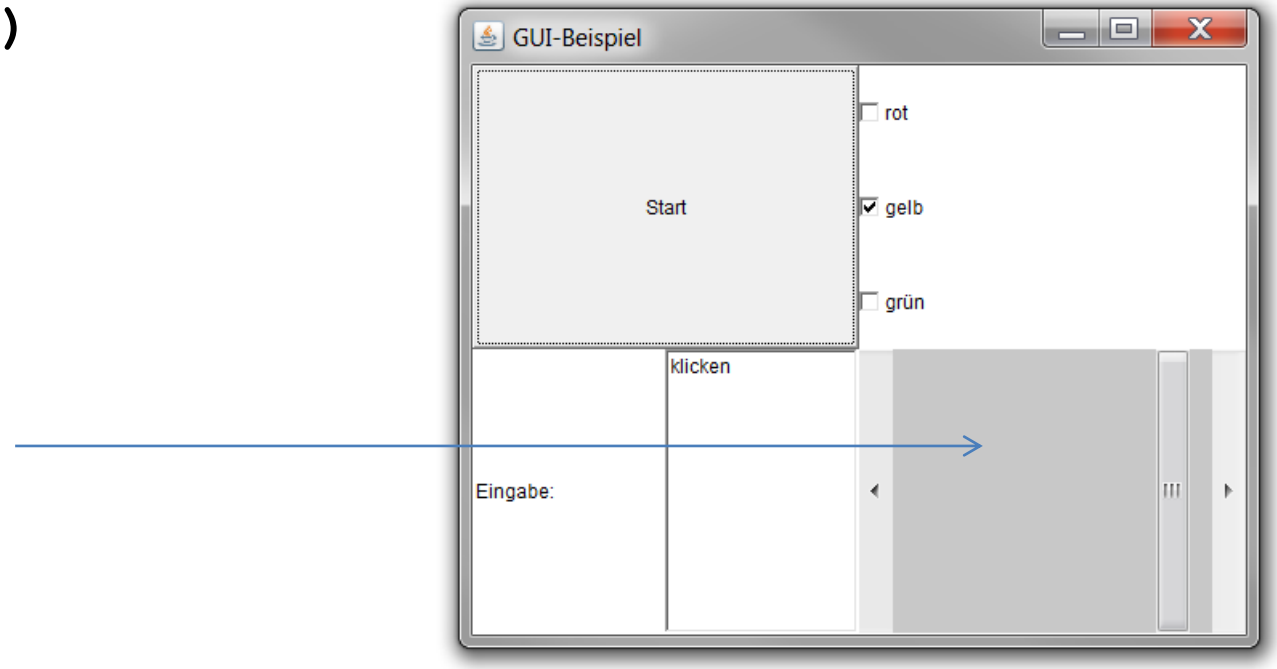
Klasse TextField

```
// zur Darstellung von Texteingabefeldern
class TextField {
  TextField()
  TextField(String text)
  void setText(String text)
  String getText()
}
```

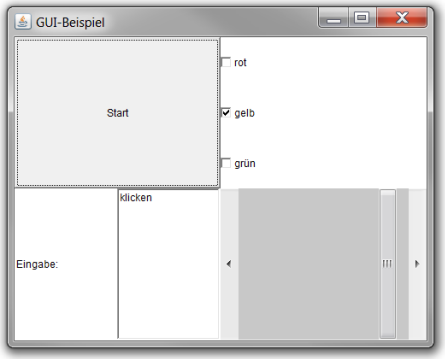


Klasse Scrollbar

```
// zur Darstellung von Scrollbars
class Scrollbar {
  Scrollbar()
  void setOrientation(int orientation)
      // 0 = horizontal, 1 = vertikal
  void setValue(int value) // zwischen 0 und 100
  int getValue()
}
```



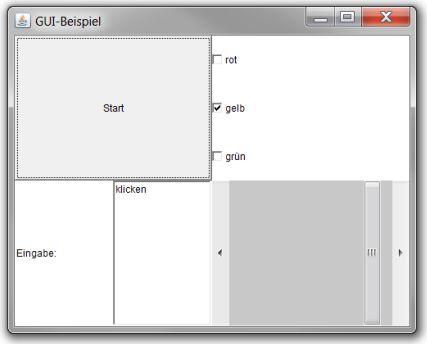
Beispielprogramm (1)




```
import java.awt.*;

public class GUIExample {

  public static void main(String[] args) {

    // Erzeugung eines Fensters
    Frame fenster = new Frame("GUI-Beispiel");

    // Festlegen von Position und Groesse
    fenster.setLocation(10, 20);
    fenster.setSize(500, 400);

    // dem Fenster wird ein Layout-Manager zugeordnet, der die dem
    // Fenster zugeordneten GUI-Komponenten in einer 2x2-Tabelle
    // darstellt
    GridLayout fensterLayout = new GridLayout(2, 2);
    fenster.setLayout(fensterLayout);
```

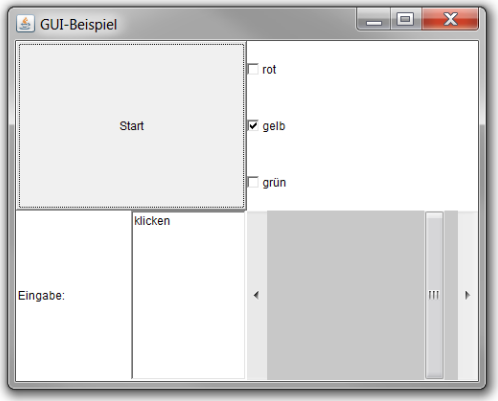
Beispielprogramm (2)

```
// oben links wird dem Fenster ein Button zugeordnet
Button start = new Button("Start");
fenster.add(start);

// oben rechts wird dem Fenster ein Panel mit 3 Checkboxen
// zugeordnet, die untereinander stehen
Checkbox box1 = new Checkbox("rot");
Checkbox box2 = new Checkbox("gelb");
box2.setState(true);
Checkbox box3 = new Checkbox("grün");
Panel checkboxPanel = new Panel();
GridLayout checkboxPanelLayout = new GridLayout(3, 1);
checkboxPanel.setLayout(checkboxPanelLayout);
checkboxPanel.add(box1);
checkboxPanel.add(box2);
checkboxPanel.add(box3);
fenster.add(checkboxPanel);
```



Beispielprogramm (3)

```
// unten links wird dem Fenster ein Panel zugeordnet, das links
// ein Label und rechts ein Texteingabefeld enthaelt
Label label = new Label("Eingabe:");
TextField eingabe = new TextField("klicken");
Panel eingabePanel = new Panel();
GridLayout eingabePanelLayout = new GridLayout(1, 2);
eingabePanel.setLayout(eingabePanelLayout);
eingabePanel.add(label);
eingabePanel.add(eingabe);
fenster.add(eingabePanel);

// unten rechts wird dem Fenster ein horizontaler Scrollbar
// zugeordnet
Scrollbar bar = new Scrollbar();
bar.setOrientation(0);
bar.setValue(83);
fenster.add(bar);
```



Beispielprogramm (4)

```
        // das Fenster wird sichtbar gemacht
        fenster.setVisible(true);
    }

}
```