

# Objektorientiertes Programmieren 1

## 2D-Grafiken

Stephan Kessler, BSc in Systems Engineering

Kontakt: [stephan.kessler@edu.teko.ch](mailto:stephan.kessler@edu.teko.ch)

# Programmbibliothek



- In Java haben Sie die Möglichkeit auf Programmbibliotheken zuzugreifen.
- Diese Bibliotheken bestehen aus vorgefertigte Klassen.
- Java besitzt eine sehr grosse Anzahl an vorgefertigten Klassen, die in der Grundinstallation (JRE / JDK) schon implementiert sind (Standardbibliothek).
- Die Programmbibliotheken in Java sind Betriebssystemunabhängig (Ausnahme: Native-Varianten) und werden mehrheitlich zur Laufzeit geladen.

# Programmbibliothek

```
public class Hauptklasse
{
    public static void main(String[] args){
        javax.swing.JFrame frame = new javax.swing.JFrame();
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

Um eine Klasse eindeutig zu identifizieren, wird ein sogenannter **Namensraum** verwendet. Z.B. können Sie die Klasse *JFrame*, womit Sie ein Fenster erzeugen können, aus der Standardbibliothek mit dem vollständigen Namensraum erreichen: `javax.swing.JFrame` .

# Programmbibliothek

```
import javax.swing.JFrame;
```

```
public class Hauptklasse  
{  
    public static void main(String[] args){  
        JFrame frame = new JFrame();  
        frame.setSize(300, 200);  
        frame.setVisible(true);  
    }  
}
```

Um jedoch diese lange Schreibweise zu verkürzen, können Sie das Schlüsselwort ***import*** verwenden. Im oberen Beispiel ist diese Verwendung dargestellt. Das Schlüsselwort ***import*** gibt Java an, dass bei der Verwendung der Klasse *JFrame*, eigentlich die Klasse *javax.swing.JFrame* gemeint ist.

# Programmbibliothek

```
import javax.swing.JFrame;

public class Hauptklasse
{
    public static void main(String[] args){
        JFrame frame = new JFrame();
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

Entsprechend kann bei der weiteren Verwendung der Klasse *JFrame* die kurze Schreibweise (ohne *javax.swing...*) verwendet werden.

# Programmbibliothek

```
import javax.swing.JFrame;

public class Hauptklasse
{
    public static void main(String[] args){
        JFrame frame = new JFrame();
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

In Java gibt es einen Teil der Programmbibliothek, welcher standardmässig in jedem Programm schon importiert ist: *java.lang*

Dort sind Klassen wie *System* oder *Math* hinterlegt. Daher konnten Sie die Methode *System.out.println()* ohne einen Import verwenden.

# Programmbibliothek

```
import javax.swing.JFrame;
```

```
public class Hauptklasse  
{  
    public static void main(String[] args){  
        JFrame frame = new JFrame();  
        frame.setSize(300, 200);  
        frame.setVisible(true);  
    }  
}
```

Oftmals werden die Schlüsselwörter ***extends*** und ***import*** miteinander verwechselt. Daher bietet sich folgende Unterscheidung:

- ***import***: Ermöglicht den direkten Zugriff (kürzere Schreibweise) auf eine externe Klasse.
- ***extends***: Vererbt / Erweitert eine Klasse mit einer anderen Klasse.



# Aufgabe

- Programmieren und führen Sie den vorherigen Code aus.
- Verändern Sie den Wert in `setVisible()`. Was verändert sich?
- Verändern Sie die Werte in `setSize()`. Was verändert sich?



# Java Dokumentation

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#)    Detail: [Field](#) | [Constr](#) | [Method](#)

javax.swing

**Class JFrame**

java.lang.Object  
  java.awt.Component  
    java.awt.Container  
      java.awt.Window  
        java.awt.Frame  
          javax.swing.JFrame

**All Implemented Interfaces:**  
  ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants

---

public class **JFrame**  
  extends `Frame`  
  implements `WindowConstants`, `Accessible`, `RootPaneContainer`

An extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture.  
Frames.

Im Internet finden Sie meistens zu den Programm-Bibliotheken passende Dokumentationen (Javadocs), welche den Aufbau der Klassen beschreiben.

# Java Dokumentation

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

javax.swing

**Class JFrame**

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
```

**All Implemented Interfaces:**

ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants

```
public class JFrame
  extends Frame
  implements WindowConstants, Accessible, RootPaneContainer
```

An extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture.  
Frames.

Die erste Darstellung nach dem Klassen-Namen zeigt die gesamte Vererbung an. In diesem Beispiel erbt *JFrame* von *Frame*, dieses wiederum von *Window*, dieses wiederum von *Container* usw.

# Java Dokumentation

## Field Summary

### Fields

Modifier and Type	Field and Description
protected <a href="#">AccessibleContext</a>	<a href="#">accessibleContext</a> The accessible context property.
static int	<a href="#">EXIT_ON_CLOSE</a> The exit application default window close operation.
protected <a href="#">JRootPane</a>	<a href="#">rootPane</a> The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	<a href="#">rootPaneCheckingEnabled</a> If true then calls to add and setLayout will be forwarded to the contentPane.

### Fields inherited from class [java.awt.Frame](#)

[CROSSHAIR\\_CURSOR](#), [DEFAULT\\_CURSOR](#), [E\\_RESIZE\\_CURSOR](#), [HAND\\_CURSOR](#), [ICONIFIED](#), [MAXIMIZED\\_BOTH](#), [MAXIMIZED\\_HORIZ](#), [MAXIMIZED\\_VERT](#), [MOVE\\_CURSOR](#), [N\\_RESIZE\\_CURSOR](#), [NE\\_RESIZE\\_CURSOR](#), [NORMAL](#), [NW\\_RESIZE\\_CURSOR](#), [S\\_RESIZE\\_CURSOR](#), [SE\\_RESIZE\\_CURSOR](#), [SW\\_RESIZE\\_CURSOR](#), [TEXT\\_CURSOR](#), [W\\_RESIZE\\_CURSOR](#), [WAIT\\_CURSOR](#)

Das Feld *Field Summary* zeigt alle Eigenschaften / Variablen der Klasse auf (auch die vererbten Eigenschaften / Variablen → „... inherited from...“).

# Java Dokumentation

## Field Summary

### Fields

Modifier and Type	Field and Description
protected <a href="#">AccessibleContext</a>	<a href="#">accessibleContext</a> The accessible context property.
static int	<a href="#">EXIT_ON_CLOSE</a> The exit application default window close operation.
protected <a href="#">JRootPane</a>	<a href="#">rootPane</a> The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	<a href="#">rootPaneCheckingEnabled</a> If true then calls to add and setLayout will be forwarded to the contentPane.

### Fields inherited from class [java.awt.Frame](#)

[CROSSHAIR\\_CURSOR](#), [DEFAULT\\_CURSOR](#), [E\\_RESIZE\\_CURSOR](#), [HAND\\_CURSOR](#), [ICONIFIED](#), [MAXIMIZED\\_BOTH](#), [MAXIMIZED\\_HORIZ](#), [MAXIMIZED\\_VERT](#), [MOVE\\_CURSOR](#), [N\\_RESIZE\\_CURSOR](#), [NE\\_RESIZE\\_CURSOR](#), [NORMAL](#), [NW\\_RESIZE\\_CURSOR](#), [S\\_RESIZE\\_CURSOR](#), [SE\\_RESIZE\\_CURSOR](#), [SW\\_RESIZE\\_CURSOR](#), [TEXT\\_CURSOR](#), [W\\_RESIZE\\_CURSOR](#), [WAIT\\_CURSOR](#)

Das Schlüsselwort ***protected*** ist ein weiterer Zugriffsmodifikator (wie public) und bedeutet in Java, dass man auf dieses Element **nur** in dieser Klasse oder **in einer Klasse, welche von dieser Klasse erbt, zugreifen kann**.

Z.B:

- Man instanziiert ein Objekt *frame* von *JFrame* in der Hauptklasse. Der Zugriff *frame.rootPane* ist dabei nicht möglich.
- Man erstellt eine neue Klasse und lässt diese von der Klasse *JFrame* erben. Innerhalb dieser Klasse ist der Zugriff auf *rootPane* möglich.

# Java Dokumentation

## Constructor Summary

### Constructors

#### Constructor and Description

**JFrame()**

Constructs a new frame that is initially invisible.

**JFrame(GraphicsConfiguration gc)**

Creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.

**JFrame(String title)**

Creates a new, initially invisible Frame with the specified title.

**JFrame(String title, GraphicsConfiguration gc)**

Creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Das Feld *Constructor Summary* zeigt alle Konstruktoren für die Instantiierung auf.

# Java Dokumentation

## Method Summary

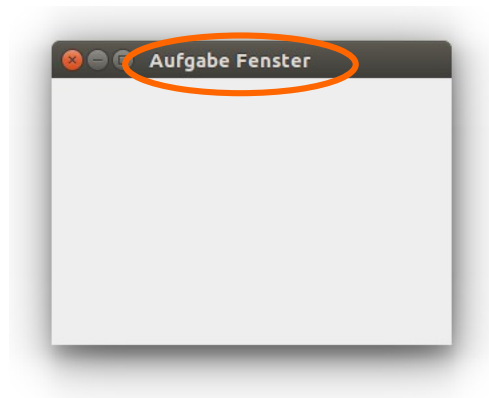
Methods	
Modifier and Type	Method and Description
protected void	<b>addImpl</b> ( <b>Component</b> comp, <b>Object</b> constraints, int index) Adds the specified child Component.
protected <b>JRootPane</b>	<b>createRootPane</b> () Called by the constructor methods to create the default rootPane.
protected void	<b>frameInit</b> () Called by the constructors to init the JFrame properly.
<b>AccessibleContext</b>	<b>getAccessibleContext</b> () Gets the AccessibleContext associated with this JFrame.
<b>Container</b>	<b>getContentPane</b> () Returns the contentPane object for this frame.
int	<b>getDefaultCloseOperation</b> () Returns the operation that occurs when the user initiates a "close" on this frame.
<b>Component</b>	<b>getGlassPane</b> () Returns the glassPane object for this frame.

Das Feld *Method Summary* zeigt alle Methoden auf (auch die vererbten Methoden → „inherited from...“).



# Aufgabe

- Suchen Sie im Internet nach der Dokumentation von *JFrame* (Suchwort: „Java JFrame docs“)
- Schauen Sie sich die Konstruktoren von *JFrame* an. Versuchen Sie mit dem Wissen einen Titel (z.B. „Aufgabe Fenster“) für das Fenster der letzten Aufgabe zu setzen:



- Suchen Sie die Methode *setSize()*. Was können Sie sonst noch für Parametern anstelle von *int width* und *int height* verwenden?

# 2D-Grafiken Vorbereitung

```
import javax.swing.*;  
import java.awt.*;
```

```
public class Hauptklasse  
{  
    public static void main(String[] args){  
        JFrame frame = new JFrame();  
        frame.setSize(300, 200);  
        frame.setVisible(true);  
    }  
}
```

Damit wir die 2D-Grafiken nutzen können, benötigen wir weitere Klassen aus den Programmbibliotheken. Um nicht jedes einzelne Element mit dem Schlüsselwort *import* einfügen zu müssen, nutzen wir beim *import*-Befehl das Zeichen: \*

Für die 2D-Grafiken brauchen wir mehrere Klassen aus *swing* und *awt*. Mit dem Zeichen \* haben wir nun den direkten Zugriff auf all diese Elemente aus diesen Bereichen.



# 2D-Grafiken Vorbereitung

```
import javax.swing.*;
import java.awt.*;

public class MeinFenster extends JFrame
{
}
}
```

```
public class Hauptklasse
{
    public static void main(String[] args){
        MeinFenster frame = new MeinFenster();
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

Sie können natürlich auch die Klasse *JFrame* auf eine eigene Klasse vererben und diese Klasse dann instantiieren.

# 2D-Grafiken Vorbereitung

```
import javax.swing.*;
import java.awt.*;

public class MeinFenster extends JFrame
{
    public void paint(Graphics g){

    }
}
```

```
public class Hauptklasse
{
    public static void main(String[] args){
        MeinFenster frame = new MeinFenster();
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

Dadurch haben Sie, wie Sie bereits wissen, auch die Möglichkeit Methoden von *JFrame* zu **überschreiben**.

Um 2D-Grafiken zeichnen zu können, haben wir die Möglichkeit die Methode *paint(Graphics g)* zu überschreiben. Diese Methode wird nach dem Instantiieren von *JFrame* bzw. *MeinFenster* automatisch aufgerufen. Das Objekt *Graphics g* steht uns dabei bei jedem Aufruf zur Verfügung. Mit diesem Objekt können wir dann die Figuren zeichnen.

# 2D-Grafiken Vorbereitung

```
import javax.swing.*;  
import java.awt.*;  
  
public class MeinFenster extends JFrame  
{  
    public void paint(Graphics g){  
        g.drawRect(20, 20, 100, 100);  
    }  
}
```

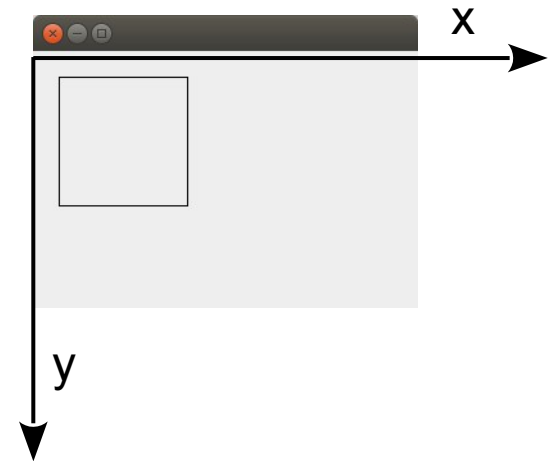
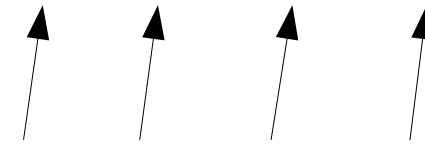


Wir können zum Beispiel mit folgender Zeile ein Rechteck zeichnen.

# 2D-Grafiken Vorbereitung

```
import javax.swing.*;  
import java.awt.*;  
  
public class MeinFenster extends JFrame  
{  
    public void paint(Graphics g){  
        g.drawRect(20, 20, 100, 100);  
    }  
}
```

x      y      width      height



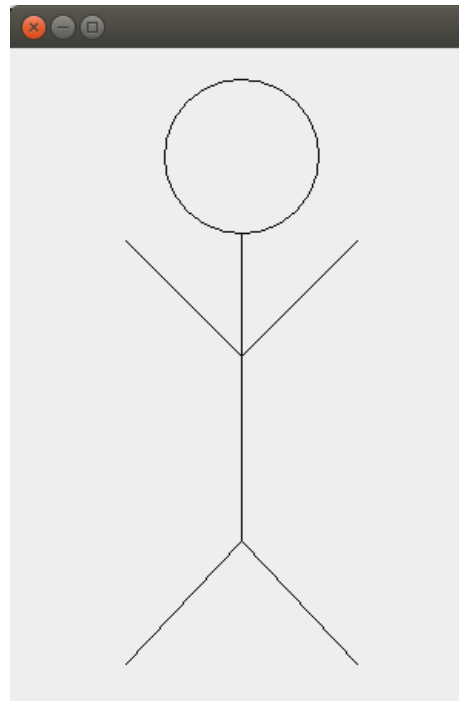
Die ersten beiden Werte geben die Position (x, y) und die zwei Weiteren die Breite (width) und die Höhe (height) des Rechteckes an.

Positionen am Bildschirm werden mehrheitlich mit einem Koordinatensystem gezeichnet, bei der die positive x-Richtung nach rechts und die positive y-Richtung nach unten zeigen.



# Aufgabe

- Suchen Sie im Internet nach den Java-Docs der Klasse *Graphics*.
- Versuchen Sie mit hilfe der Dokumentation von *Graphics* folgende Figur zu zeichnen:





# Standardbibliothek

- **java.lang** Stellt essentielle Klassen zur Verfügung (*String*, *Object*, *Thread* usw.). Ist von Anfang an «importiert».
- **java.util** Stellt viel verwendete Datenstrukturen und Werkzeuge zur Verfügung (*ArrayList*, *Hashtable*, *Scanner* usw.)
- **java.io** Ermöglicht das Lesen und Schreiben von Dateien.
- **java.nio** Alternative zu *java.io*, welcher einen nicht blockierenden Zugriff erlaubt. Geeignet für intensivere Operationen.
- **java.math** Stellt flexible Strukturen bereit, welche an die primitiven Datenstrukturen *float* und *int* anlehnen. (z.B. lassen sich in der Klasse *BigInteger* extrem grosse Zahlen, welche nicht mehr von einem normalen *int* aufgenommen werden können, halten.)
- **java.net** Erlaubt den Zugriff auf das Netzwerk (TCP und UDP).
- **javax.swing** / **java.awt** Stellt Klassen zur Darstellung von grafischen Benutzeroberflächen zur Verfügung.