C

Dateien

Datei öffnen:

```
FILE *fopen (const char *filename, const char *mode)
```

Mode: "**r**" read; "**w**" write, "**a**" attach

In Datei schreiben:

```
int fprintf (FILE *stream, const char *format, ...)
```

Aus Text Datei formatiert lesen:

```
int fscanf (FILE *stream, const char *format, ...)
```

Flush

```
fflush (stream)
```

fflush schreibt noch nicht geschriebene Daten für stream, wirft noch nicht gelesene, gepufferte Eingaben weg.

Prüfe ob am Ende des Files amgekommen:

```
int feof (FILE *stream );
```

Datei schliessen:

```
int fclose(FILE *stream)
```

fclose schreibt noch nicht geschriebene Daten für *stream*, wirft noch nicht gelesene, gepufferte Eingaben weg, gibt automatisch angelegte Puffer frei und schliesst den Datenstrom

Präprozessor

Übersetzung von C/C++ in 3 Phasen

1. Phase: Präprozessor (rein textuelle Substitutionen)
   Präprozessor Direktiven alle mit #
2. Phase: Übersetzung in Maschinencode
   obj-Dateien: (linkbare Einheit)
3. Phase: Linken der Teile zu ausführbarem Programm

Der Präprozessor hat folgende Aufgaben:

- Dateien in andere einfügen #include
  ```
  #include <Dateiname>
  #include "Dateiname"
  ```
- Konstanten definieren #define
  ```
  #define ANZAHL 10
  ```
  Rein textuelle Ersetzung
- Makros zur Verfügung stellen #define
  ```
  #define MAKRO(Arg1, Arg2, ...) Ersetzungstext
  ```
  Rein textuelle Ersetzung
- Teile des Quelltextes selektieren #ifdef, #ifndef
  Mehrfache Übersetzung/Includes verhindern (Headerfiles):
  ```
  #ifndef MEINEDATEI_H

  #define MEINEDATEI_H
  // Code
  #endif
  ```

Dynamischer Speicher

`calloc(n, size)` Alloziert mit 0 initialiesierten Speicher von n mal «size»

`malloc(n)` Alloziert uninitialiesierten Speicher von n bytes

Listen

First, next →

AdressOf Operator

```
int i = 5;
int* pi = &i;
printf("%d", *pi); //Value of i: 5
printf("%d", i); //Value of i: 5
printf("%d", pi); //Adress of i
```

Java

Basics

```
class RobotPose {
    public float x;
    public float y;
    public float alpha;

    public float calculateDistance() {
        return (float)Math.sqrt(x*x+y*y);
    }

    public float calculateDistance(RobotPose pose) {
        return (float)Math.sqrt((pose.x-x)*(pose.x-x)+(pose.y-y)*(pose.y-y));
    }
}
```

Klasse

Daten (Variablen)

Methoden

```
RobotPose pose = new RobotPose();      // kreiert ein Objekt
RobotPose otherPose = new RobotPose();
…

pose.x = 10.3;    // zuweisen von numerischen Werten
pose.y = 17.4;

float distance = pose.calculateDistance();    // Aufruf der Methode

float distanceToOther = pose.calculateDistance(otherPose);
```

Klasse Objekt

Datentypen

Typenhierarchie: double (höchster Typ)
float
long
int
short
byte (niedrigster Typ)

Automatische Umwandlung: Bei gemischten Datentypen wird automatisch in den höheren umgewandelt

Umwandlung in niedrigeren Datentyp: `float d = 4.0; int i = (int)d;`

Referenzdatentypen

Arrays

Als Parameter: Werden immer als Pointer übergeben

Deklaration:

```
int[] a; // deklariert a
a = new int[5]; // reserviert Platz fuer 5 int auf Heap
```

Länge wird mit Objekt gespeichert. Kann mit `arrayname.length` abgerufen werden

Strings

Verkettung:

```
String str3 = str1 + str2;
String str1 = "Hallo " + 1 + 2;//"Hallo 12"
```

Test auf Gleichheit: `if (s.equals(t)) {} // teste auf Gleichheit von Wert`

Reihenfolge, Sortierung: `int compareTo(String str);`

```
int n = a.compareTo(b);
n == 0: a ist gleich b
n < 0: a steht vor b "ameise" < "biene"
n > 0: a steht nach b
Sonderzeichen < Zahlen < Grossbuchstaben < Kleinbuchstaben
```

Methoden

Instanzmethoden sind an ein Objekt gebunden, static-Methoden an eine Klasse.

Zugriffsmodifikator (access control modifier)

- `public` überall sichtbar (auch für andere Klassen; main Methode muss public sein)
- `protected` nur innerhalb derselben und abgeleiteter Klasse sichtbar
- `private` nur innerhalb derselben Klasse sichtbar
- `static` Methode können direkt Aufgerufen werden, ohne dass man eine Instanz erstellt.

Objekt

Polymorphismus

Ein Objekt behält immer die Identität der Klasse, aus der es erzeugt wurde.

Wenn eine Methode auf ein Objekt aufgerufen wird, wird immer die Methode verwendet, die mit der Klasse des Objekt verbunden ist

Klasse

Oberklasse: Klasse die vererbt (Superclass/Baseclass)

Unterklasse: Klasse die erbt (Subclass/abgeleitete Klasse)

Von der Oberklasse werden in der Unterklasse folgende Elemente übernommen:

- alle Attribute
- alle Methoden

die Unterklasse kann

- zusätzliche Attribute definieren
- zusätzliche Methoden definieren
- geerbte Methoden mit `@Override` überschreiben,d.h. ihre Funktionalität anpassen

Eine Methode der Oberklasse kann explizit mit `super()` aufgerufen werden

Abstrakte Klassen und Methoden

können nicht instanziert werden

abstrakte Methoden müssen in abgeleiteten Klassen implementiert werden

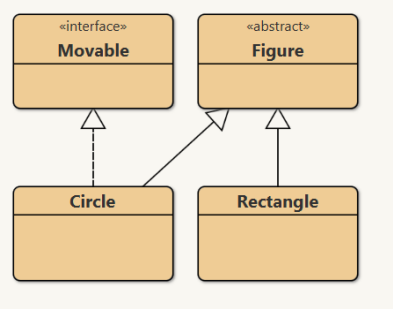
Nur abstrakte Klassen dürfen abstrakte Methoden enthalten.

Konstruktor

«Methode zum Kreiren von Objekten». Ohne Rückgabewert.

Bei abgeleiteten Klassen wird zuerst der Konstruktor der Oberklasse aufgerufen.

Vererbung




```
public interface Movable {…}
public abstract class Figure {…}
public class Circle extends Figure
implements Movable{…}
public class Rectangle extends
Figure{…}
```

Exceptions

Exception «werfen»

```
public double msSqrt(double d) throws Exception {
	if (d < 0)
		throw new Exception("Argument negativ");
	else
		return Math.sqrt(d);
}
```

Exception «fangen»

```
try {
	double d = mySqrt(p);
	result = "Wurzel: " + d;}
catch (Exception ex) {
	result = ex.getMessage();}
```

Eigene Exception Klassen

```
public class MyException extends Exception {
	public MyException (String message) {
		super(message);
		}
}
```

```java
import javax.swing.*; // Import Swing components
import java.awt.*; // Import AWT for Color and Graphics
import java.awt.event.*; // Import events like ActionListener

// Class extends JFrame to create a window
// Implements ActionListener to handle button clicks
public class GUI extends JFrame implements ActionListener {
    // GUI components and variables
    private JButton btnColor, btnSize; // Two buttons for interaction
    private Color currentColor = Color.BLUE; // Current color of the rectangle
    private int x = 100, y = 100, size = 200; // Position and size of the rectangle

    // Initializes the layout and adds components (buttons)
    public void initComponents() {
        JPanel panel = (JPanel) this.getContentPane(); // Get the content panel of the window
        panel.setBackground(Color.WHITE); // Set background color of the window
        panel.setLayout(new FlowLayout()); // Simple layout

        // Create the "Change Color" button
        btnColor = new JButton("Change Color");
        panel.add(btnColor); // Add button to the window
        btnColor.addActionListener(this); // Register this class to handle button click

        // Create the "Change Size" button
        btnSize = new JButton("Change Size");
        panel.add(btnSize);
        btnSize.addActionListener(this);
    }

    // Custom drawing method — draws the rectangle
    @Override
    public void paint(Graphics g) {
        super.paint(g); // Call the parent method to ensure proper drawing
        g.setColor(currentColor); // Set the drawing color
        g.fillRect(this.x, this.y, this.size, this.size); // Draw the rectangle at (x, y) with given size
    }

    @Override
    public void actionPerformed(ActionEvent e) { // This method handles button clicks
        if (e.getSource() == btnColor) {
            // Toggle color between BLUE and RED
            currentColor = (currentColor == Color.BLUE) ? Color.RED : Color.BLUE;
        } else if (e.getSource() == btnSize) {
            // Enlarge square by 10
             this.size += 10;
        }

        repaint(); // Redraw the window with updated color or size
    }

    // Entry point of the program
    public static void main(String[] args) {
        // Optional: Try to set the system's look and feel
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            e.printStackTrace(); // Print error if setting the look and feel fails
        }

        // Create the window (GUI instance)
        GUI window = new GUI();
        window.setTitle("Rectangle Demo"); // Set the window title
        window.setSize(400, 400); // Set window dimensions
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Exit when closed
        window.initComponents(); // Add buttons and layout
        window.setLocationRelativeTo(null); // Center the window on the screen
        window.setVisible(true); // Make the window visible
    }
}

```