Teilaspekt der Objektorientierung in Java

In diesem Dokument möchte ich einen Teilaspekt der Objektorientierung in Java erläutern, der meiner Meinung nach in Büchern und im Internet nicht ausführlich genug erklärt wird und daher oftmals für Verständnisprobleme sorgt. Bei mir hat es somit relativ lange gedauert, bis ich diesen Teil verstanden habe.

Fangen wir mit einem einfachen kleinen Programm an, das ein Fenster öffnet und drei Buttons erstellt. Zwei davon sind rot:

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

public class myProgram {
  public static void main(String[] args) {

    JFrame f = new JFrame("My Window");
    JPanel p = new JPanel();
    JButton b1 = new JButton("Button 1");
    JButton b2 = new JButton("Button 2");
    JButton b3 = new JButton("Button 3");
    b1.setBackground(new Color(255,0,0));
    b3.setBackground(new Color(255,0,0));
    p.add(b1);
    p.add(b2);
    p.add(b3);
    f.add(p);
    f.pack();
    f.setVisible(true);

  }
}
```

Durch „`new`“ wird ein neues Objekt erstellt, das eine Kopie des Hauptobjekts ist.
Spielen Sie nun ein wenig mit obigem Programm herum. Schreiben Sie z.B. die Befehle mal in einer anderen Reihenfolge, um zu sehen, an welcher Stelle die Reihenfolge von Bedeutung ist und an welcher Stelle nicht.
Das obige Programm sollten Sie verstanden haben, wenn Sie dem Rest dieses Dokuments folgen wollen.

Betrachten wir uns nun `JFrame`. Man kann in Java eine neue Klasse erstellen (sofern die Klasse, von der sie sich ableitet nicht „`private`“ ist), die sämtliche Eigenschaften der Klasse, von der sie abgeleitet wurde, erbt.

In folgendem Programm habe ich nun eine Klasse `myFrame` erstellt, die mittels `extends` von der Klasse `JFrame` alle Eigenschaften erbt:

```
import javax.swing.*;

public class myFrame extends JFrame{
  myFrame(String name) {
    super(name);
  }
}
```

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

public class myProgram {
  public static void main(String[] args) {

    myFrame f = new myFrame("My Window");
    JPanel p = new JPanel();
    JButton b1 = new JButton("Button 1");
    JButton b2 = new JButton("Button 2");
    JButton b3 = new JButton("Button 3");
    b1.setBackground(new Color(255,0,0));
    b3.setBackground(new Color(255,0,0));
    p.add(b1);
    p.add(b2);
    p.add(b3);
    f.add(p);
    f.pack();
    f.setVisible(true);

  }
}
```

Im Programm `myProgram` wird nun `myFrame` auch genau so genutzt und behandelt wie zuvor `JFrame`. Und das Ergebnis ist ebenfalls gleich.
In myProgram wird an `myFrame` genau ein Argument übergeben: Der String `"My Window".`
Die Anzahl und Art der Argumente muss im Konstruktor von der `myFrame`-Klasse auch vorkommen: `myFrame (String name)`.
Und damit dieses Argument wiederum auch an das eigentliche `JFrame` übergeben wird, steht dort `super(name)`.

`myFrame` ist ein neues Objekt. Ich kann ihm sämtliche Eigenschaften meines Programms geben:

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

public class myFrame extends JFrame{
  myFrame() {
    super("My Window");
    JPanel p = new JPanel();
    JButton b1 = new JButton("Button 1");
    JButton b2 = new JButton("Button 2");
    JButton b3 = new JButton("Button 3");
    b1.setBackground(new Color(255,0,0));
    b3.setBackground(new Color(255,0,0));
    p.add(b1);
    p.add(b2);
    p.add(b3);
    this.add(p);
    this.pack();
    this.setVisible(true);
  }
}
```

```
public class myProgram {
  public static void main(String[] args) {

    myFrame f = new myFrame();

  }
}
```

Somit wird nun nur `myFrame` aufgerufen. In `myFrame` ist alles drin enthalten.
Man muss es wie eine Komponente sehen. Das Programm sieht zwar beim Ausführen wieder genau gleich aus, aber vom Programmaufbau ist es nun etwas anders. Ich kann z.B. nun `myFrame` als Komponente in andere Programme integrieren. Befehle wie `JButton` brauche ich in dem anderen Programm dann nicht zu schreiben.

Die `main()`-Methode in `myProgram` kann man in jede beliebige Klasse schreiben. So kann man sie auch direkt in `myFrame` schreiben:

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

public class myFrame extends JFrame{
  myFrame() {
    super("My Window");
    JPanel p = new JPanel();
    JButton b1 = new JButton("Button 1");
    JButton b2 = new JButton("Button 2");
    JButton b3 = new JButton("Button 3");
    b1.setBackground(new Color(255,0,0));
    b3.setBackground(new Color(255,0,0));
    p.add(b1);
    p.add(b2);
    p.add(b3);
    this.add(p);
    this.pack();
    this.setVisible(true);
  }

  public static void main(String[] args) {

    myFrame f = new myFrame();

  }
}
```

Das hat den enormen Vorteil, dass ich in Java in mehren Klassen `main()`-Methoden haben kann. Ich kann dann das Programm entweder von der eigentlichen Klasse aus starten oder aber auch einfach die `main()`-Methode in einer der Komponenten. In letzterem Fall hat man dann z.B. ein Programm, das nur die eine Komponente anzeigt. Somit muss man nicht immer das ganze Programm starten, wenn man nur mal kurz Veränderungen an einer Komponente testen will.

Genauso wie wir nun `myFrame` erstellt haben, können wir auch eine Klasse namens `RedButton` erstellen, dessen Aufgabe es ist einen roten Button zu erstellen:

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

public class RedButton extends JButton {
  RedButton(String name) {
    super(name);
    this.setBackground(new Color(255,0,0));
  }
}
```

```
import javax.swing.*;

public class myProgram {
  public static void main(String[] args) {

    myFrame f = new myFrame("My Window");
    JPanel p = new JPanel();
    RedButton b1 = new JButton("Button 1");
    JButton b2 = new RedButton("Button 2");
    RedButton b3 = new JButton("Button 3");
    p.add(b1);
    p.add(b2);
    p.add(b3);
    f.add(p);
    f.pack();
    f.setVisible(true);

  }
}
```

An dieser Stelle sieht man auch besser den Sinn der abgeleiteten Klassen. Nun wurden zwei

JButtons direkt zu RedButtons. Wenn man bedenkt, dass sämtliche Swing-Komponenten wie JButton gezeichnet werden, so ist es eine erhebliche Arbeitserleichterung, dass das Programm dahinter nur einmal geschrieben wurde und dann immer wieder genutzt werden kann.

Zu guter Letzt nun auch noch mal die Klasse RedButton in Kombination mit der Klasse myFrame:

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

public class RedButton extends JButton {
  RedButton(String name) {
    super(name);
    this.setBackground(new Color(255,0,0));
  }
}
```

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

public class myFrame extends JFrame{
  myFrame() {
    super("My Window");
    JPanel p = new JPanel();
    RedButton b1 = new JButton("Button 1");
    JButton b2 = new RedButton("Button 2");
    RedButton b3 = new JButton("Button 3");
    p.add(b1);
    p.add(b2);
    p.add(b3);
    this.add(p);
    this.pack();
    this.setVisible(true);
  }

  public static void main(String[] args) {

    myFrame f = new myFrame();

  }
}
```

Ich hoffe, dieses Dokument bringt bei dem einen oder anderen etwas mehr Licht ins Dunkel der Objektorientierung von Java. Es erklärt halt nur einen Teilaspekt, wenngleich doch einen sehr wichtigen.