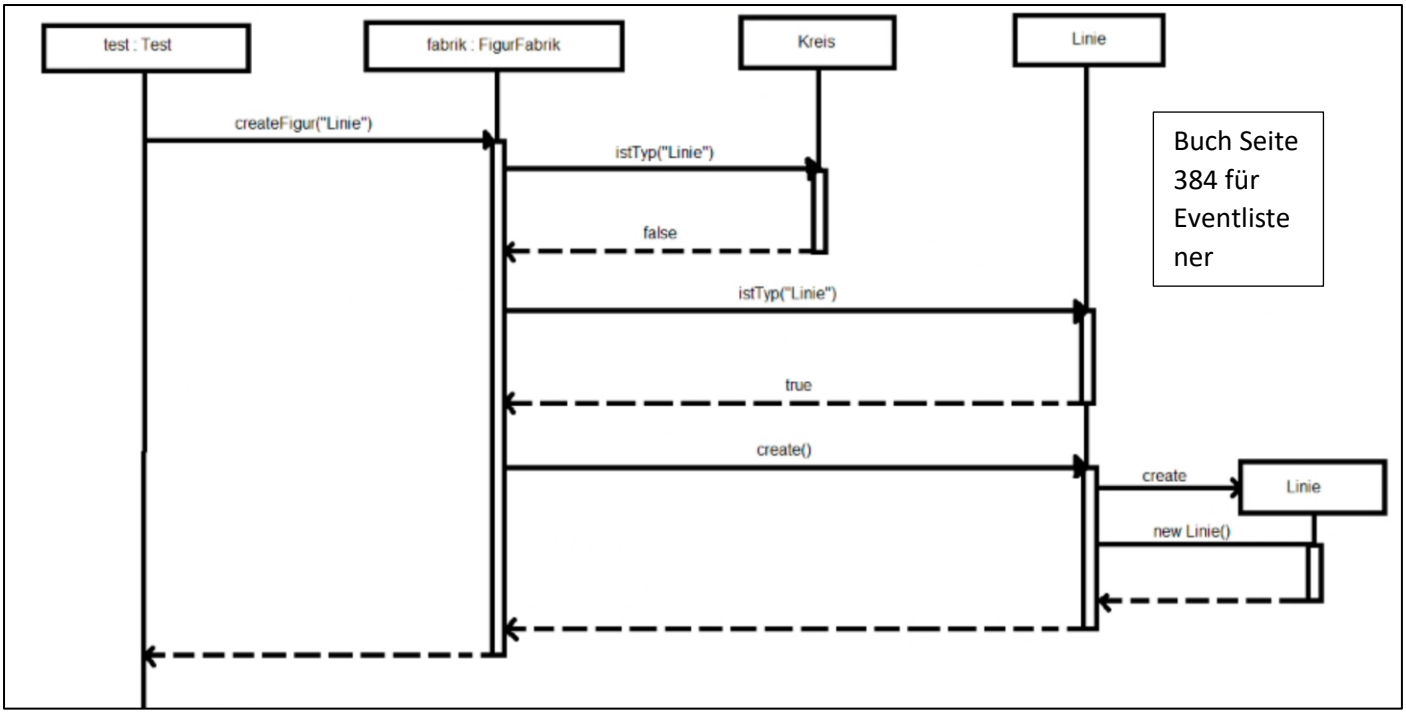



```java
package Editor;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class EditorKeyAdapter extends KeyAdapter {
    private EditorControl editorControl;

    public EditorKeyAdapter(EditorControl editorControl) { this.editorControl = editorControl; }

    @Override
    public void keyReleased(KeyEvent e){
        editorControl.setFigurTyp(e.getKeyChar());
    }
}
```

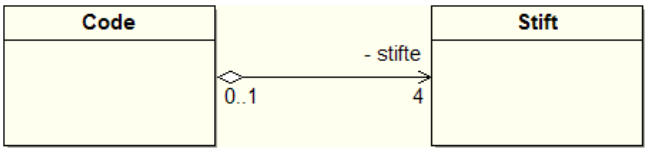
```java
package Editor;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class EditorMouseAdapter extends MouseAdapter{

    private EditorControl editorControl;

    public EditorMouseAdapter(EditorControl editorControl) { this.editorControl = editorControl; }

    @Override
    public void mouseClicked(MouseEvent e) {
    }

    @Override
    public void mousePressed(MouseEvent e) {
        editorControl.setErsterPunkt(e.getPoint());
    }

    @Override
    public void mouseReleased(MouseEvent e) { editorControl.erzeugeFigurMitZweitemPunkt(e.getPoint()); }

    @Override
    public void mouseEntered(MouseEvent e) {

    }

    @Override
    public void mouseExited(MouseEvent e) {

    }
}
```

```java
public void zeichneFiguren(Graphics g) {
    for (Figur f : figuren) {
        f.zeichne(g);
    }
}
```

<- ist Poly-morp

Aggregation

Die Aggregation ist eine Form der Assoziation, welche eine „Ganzes zu Teil“-Beziehung ausdrückt. Im folgenden Beispiel besteht eine Aggregation zwischen der Klasse `Code` (Ganzes) und der Klasse `Stift` (Teil). Zur Laufzeit steht ein `Code`-Objekt mit genau 4 `Stift`-Objekten in Verbindung.

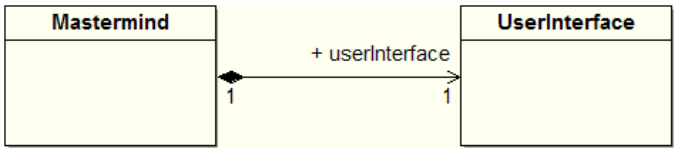


Die Aggregation ist eine schwache Form der „Ganzes zu Teil“-Beziehung. Wird das Ganze aufgelöst, bestehen die Teile weiter. Konkret: Wird das `Code`-Objekt zerstört, leben die `Stift`-Objekte weiter.

Aggregation wird durch einen leeren Rhombus auf der Seite des Ganzen angezeigt.

Komposition

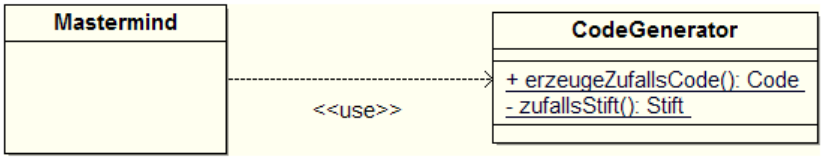
Die Komposition ist eine stärkere Form der Aggregation. Hier ist die Lebensdauer der Teilobjekte an die Lebensdauer des Ganzen gebunden. Stirbt das Ganze, so werden auch die Teilobjekte zerstört. Im folgenden Beispiel existiert das `UserInterface`-Objekte nur so lange, wie das `Mastermind`-Objekt existiert.



Komposition wird durch einen ausgefüllten Rhombus auf der Seite des Ganzen angezeigt. Zusätzlich darf die Multiplizität auf der Seite des Ganzen nie grösser als 1 sein. Ein Teilobjekt kann immer nur zu einem Ganzen gehören.

Abhängigkeiten

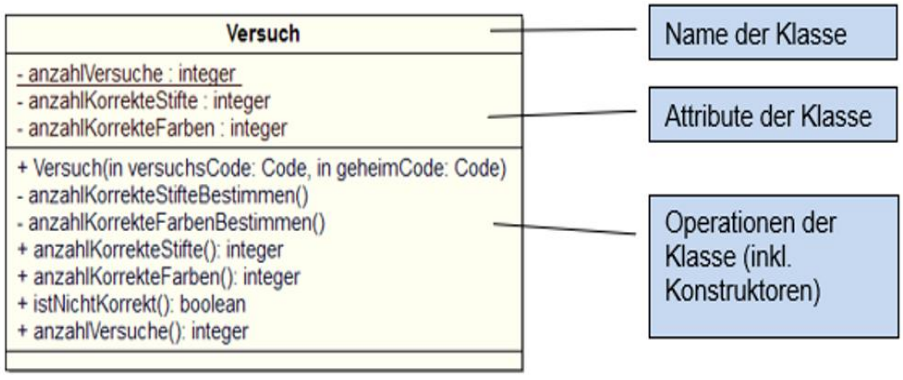
Eine Abhängigkeit drückt aus, dass eine Klasse eine andere Klasse in irgendeiner Form benötigt.



In obigem Beispiel geht es darum, dass die Klasse `Mastermind` für das Erzeugen des Geheimcodes die Methode `erzeugeZufallsCode`() der Klasse `CodeGenerator` aufruft. Dadurch wird die Klasse `Mastermind` von `CodeGenerator` abhängig. Wenn an letzterer etwas geändert wird, so muss überprüft werden, ob erstere noch korrekt arbeitet.

Abhängigkeiten werden durch gestrichelte Pfeile dargestellt. Es gibt viele verschiedene Formen von Abhängigkeiten. Diese werden durch das beigefügte Schlüsselwort in << >> unterschieden. Im Beispiel geht es um eine *Usage*-Abhängigkeit.

Normalerweise wird man in einem Diagramm nur die wichtigsten Abhängigkeiten darstellen, da das Diagramm sonst überladen wird.



Abstrakte Klassen und Methoden: Schief oder {abstract} am Ende