# Auftrag 4: Polymorphismus und Delegation

## Aufgabe 1: Die Vorteile von Polymorphismus und Delegation verstehen

*Einleitung:*

Objektorientiertes Software-Design steht für Wartbarkeit, Erweiterbarkeit und Robustheit. Mit Hilfe von Polymorphismus und Delegation kann dies erreicht werden.

*Aufgabe:*

Im Screencast „Einführung (aus früherem 226).mp4“ und der dazugehörigen Präsentation „Polymorphismus und Delegation.ppt“ werden die Prinzipien und Vorteile von Polymorphismus und Delegation am Beispiel eines Ausschnitts aus dem Programm Grafikeditor erklärt. Schauen Sie sich den Screencast und die Präsentation an.

## Aufgabe 2: Polymorphismus und Delegation einsetzen

*Einleitung:*

K 1.4

Sie haben gesehen, wie mit Hilfe der Prinzipien Delegation und Methoden überschreiben (Polymorphismus) erweiterbarer Code geschrieben werden kann. Wobei „erweiterbarer Code“ hier zu verstehen ist, dass bei Erweiterungen bestehender Code nicht angetastet wird; sondern die Erweiterungen lediglich durch neue Klassen realisiert werden.

*Aufgabe:*

Der nachfolgende Codeausschnitt zeigt die Klassen eines Bibliothekprogramms.

* Die Klasse Bibliothek enthält einen Katalog, in welchem alle verfügbaren Medien als Objekte gespeichert sind.
* Medien sind Bücher, CDs und DVDs.
* Jedes Medium verfügt über ein Attribut sample, das einen Ausschnitt des tatsächlichen Inhaltes birgt.
* Die Klasse Bibliothek hat eine Methode zur Ausgabe des Samples (showSample(Medium m)), wobei je nach Art des Mediums ein Player-Objekt verwendet wird, das in der Lage ist Sound und Video abzuspielen.

**public** **class** Bibliothek {

**private** Medium[] katalog;

**public** **void** showSample(Medium m) {

**if** (m **instanceof** Buch) {

System.*out*.println(((Buch) m).sample);

} **else** **if** (m **instanceof** CD) {

Player p = **new** Player();

p.playSound(((CD) m).sample);

} **else** **if** (m **instanceof** DVD) {

Player p = **new** Player();

p.playVideo(((DVD) m).sample);

}

}

}

**public** **abstract** **class** Medium { }

**public** **class** Buch **extends** Medium {

String sample;

}

**public** **class** CD **extends** Medium {

Sound sample;

}

**public** **class** DVD **extends** Medium {

Video sample;

}

**public** **class** Sound { // ... }

**public** **class** Video { // ... }

**public** **class** Player {

**public** **void** playVideo(Video v) {

// Video anzeigen ...

}

**public** **void** playSound(Sound s) {

// Sound spielen ...

}

}

Die Implementation der Methode showSample()ist ein typisches Beispiel für eine nicht erweiterbare Methode (siehe Polymorphismus und Delegation.ppt, Seite 5).

Lösen Sie folgende Aufgabe in einem neuen Eclipse-Projekt.

1. Verbessern Sie diese Methode durch Anwendung von Delegation und Polymorphismus, so dass das Bibliotheksprogramm später um weitere Medien (z.B. Game) *erweitert* werden kann.
2. Schreiben Sie die nötigen Änderungen im Code (Klassen, Methoden) in Java auf (inklusive der geänderten Methode showSample() der Klasse Bibliothek.

## Aufgabe 3: Polymorphismus und Delegation im Programm Grafikeditor (Diese Aufgabe kann nicht für die Kompetenz 1.4 verwendet werden, ist aber Voraussetzung für die Kompetenz 3.4, die nach Lektion 5 abgegeben werden kann.)

*Einleitung:*

In der Aufgabe 1 haben Sie gesehen, dass für die Realisierung des Grafikeditors *Zeichnung* als neue Klasse identifiziert wurde. Ein Zeichnungsobjekt dient als Container für alle Figurobjekte, die zu einer bestimmten Zeichnung gehören. Jetzt sollen Sie diese Klasse gemäss dem besprochenen Design implementieren.

*Aufgabe:*

Implementieren Sie in Ihrem Grafikeditor-Projekt eine Klasse Zeichnung.Sie können sich an dem Klassendiagramm auf S. 11 von „Polymorphismus und Delegation.ppt“ orientieren:

* Die Klasse Zeichnung hat einen *Array* oder einen *Vector* (siehe Buch Abschnitt 5.4.1) als Instanzvariable. Hier werden die *Referenzen* auf die verschiedenen Figurobjekte der Zeichnung gespeichert.
* Die Klasse Zeichnung soll einen Konstruktor haben. Dieser verfügt über einen Parameter mit dem selben Typ, wie obige Instanzvariable. Beim Aufruf wird die Instanzvariable mit dem übergebenen Array oder Vector initialisiert.
* Die Klasse Display soll nichts mehr mit dem Zeichnen von Figuren zu tun haben. Sie delegiert dies an die Klasse Zeichnung.
  + Die Methode zeichneFiguren(Graphics g) war bisher in der Klasse Display. Die Methode muss jetzt in die Klasse Zeichnung verschoben werden.
  + Alle anderen Methoden, die mit Figuren zusammenhängen (hinzufügen, löschen, etc.), müssen auch verschoben werden.
  + Damit die Delegation funktioniert, braucht die Klasse Display ein Referenz auf das Zeichnungs-Objekt als Instanzvariable.
  + In der Klasse braucht es neu eine Methode zum Setzen des zugehörigen Zeichnungs-Objekt (**public** **void** setZeichnung(Zeichnung zeichnung)
* Auch die Zeichnungsklasse zeichnet die Figuren nicht selber, sondern delegiert dies an die einzelnen Figurobjekte. Dieser Schritt wir exakt in der Präsentation „Polymorphismus und Delegation.ppt“ gezeigt. Führen Sie die beschriebenen Ergänzungen und Änderungen in den Klassen Zeichnung und Figur und den von Figur abgeleiteten Klassen durch.
* Das Testprogramm muss so ergänzt werden, dass ein Zeichnungsobjekt mit einigen Figuren erzeugt wird. Dieses muss dem Display mit der setZeichnung()-Methode zur Anzeige übergeben werden.

## Aufgabe 4: Eine neue Figurklasse einfügen

*Einleitung:*

Das Design Ihres Programms Grafikeditor sollte nun so sein, dass Sie die Vorteile von Polymorphismus und Delegation ausnützen können. Testen Sie dies, indem Sie eine neue Art von Figur (RechteckRund: Rechteck mit abgerundeten Kanten, Dreieck, Trapez, oder irgendeine andere Figur Ihrer Wahl) in das Programm aufnehmen.

K 1.4

*Aufgabe:*

Überlegen Sie sich eine weitere Figur-Form, die Sie in Ihr Programm integrieren möchten. Schreiben Sie die dazugehörige Klasse, fügen Sie ein Objekt in die Zeichnung ein und geben Sie es aus.

Achten Sie darauf, dass an den Klassen Display und Zeichnung keine Änderungen vorgenommen werden dürfen, damit die Prinzipien von Polymorphismus und Delegation erfüllt sind.

## Vertiefungs- und Ergänzungsaufgaben

Die folgenden Aufgaben dienen Ihrer individuellen Vertiefung. Sie können diese nach Belieben bearbeiten und dadurch auch Zusatzkompetenzen erlangen. Die Aufgaben sind bewusst offen formuliert und Sie werden für die Lösung im Web recherchieren müssen.

#### Speichern von Figuren in Datei

Die Vertiefungsaufgabe 2 des Arbeitsblatts 03 dreht sich um das Speichern und Laden von Figur-Objekten in eine, respektive aus einer Datei.  
  
Wir betrachten vorerst das Speichern.Wenn Sie diese Aufgabe gelöst haben, dann werden Sie dort vermutlich eine weitere switch-artige Situation angetroffen haben. Je nach Typ des zu speichernden Figur-Objektes muss etwas anderes in die Datei geschrieben werden. Eliminieren Sie jetzt diese switch-artige Situation durch Anwendung von Delegation und Polymorphismus. Ergänzen Sie hierzu die Klasse Zeichnung mit einer save()-Methode.

K 1.4

K 3.5

#### Laden von Figuren aus Datei I

*Vorbemerkung: Die folgenden zwei Vertiefungsaufgaben sind „Igelaufgaben“ und verlangen entsprechend Durchhaltevermögen.*

Nun schauen wir noch das Laden der Figur-Objekte aus einer Datei an. Auch hier haben wir es mit einer switch-artigen Situation zu tun. Auf Grund des aus der Datei gelesenen Textes, oder Teilen davon, muss entschieden werden, welche Art von Figur erzeugt werden soll. Können Sie diese Situation auch eliminieren?

K 1.5

K 3.5

Als Lösungsansatz kann folgende Überlegung dienen: Ein Rechteck-Objekt wäre geeignet, um festzustellen, ob ein bestimmter aus der Datei gelesener Textteil für ein Rechteck-Objekt steht. Dazu müsste die Klasse Rechteck eine neue Methode erhalten, welche eine solche Überprüfung des gelesenen Textteils durchführt und mit true oder false antwortet. Ein Kreis-Objekt könnte dann seinerseits feststellen, ob ein Textteil für ein Kreis-Objekt steht und ein Linie-Objekt könnte dasselbe für Linien tun. Immer indem bei den Klassen Kreis und Linie entsprechende Methoden hinzugefügt werden.

Beim Laden aus der Datei müssten also die gelesenen Textteile der Reihe nach einem Rechteck-, einem Kreis- und einem Linien-Objekt zur Prüfung vorgelegt werden. Dasjenige Objekt, welches den Text erkennt kann dann ein neues Objekt des eigenen Typs erzeugen und dieses neue Objekt gemäss dem Textteil initialisieren. Dazu braucht es eine weitere Methode in den Klassen Rechteck, Kreis und Linie. Diese trägt oft den Namen clone() und macht eine Kopie des eigenen Objektes.

Damit das funktioniert, braucht es aber von Anfang an einen Satz von Figur-Objekten, welche nicht zu einer Zeichnung gehören, sondern nur dazu dienen, weitere Objekte ihres eigenen Typs zu erzeugen. Solche Objekte nennt man *Prototypen*. Das zu Grunde liegende Design-Pattern heisst *Prototype-Pattern*.

Ergänzen Sie also Ihre Zeichnungsklasse mit einer load()-Methode.

#### Laden von Figuren aus Datei II

Mit dem Prototype-Pattern aus der vorigen Aufgabe verwandt ist das *Factory-Method-Pattern*. Die Idee ist, dass nicht mehr die Figur-Objekte selber neue Objekte erzeugen, sondern dass hierzu eigene Factory-Klassen mit create()-Methoden gemacht werden. Können Sie Ihren Code so umbauen?

K 1.5

K 3.5