Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 19 – Abstrakte Klassen

(Stand 22.12.2015)

Grundlage der Aufgaben 1 – 4:

Es soll eine Prüfung durchgeführt werden. Die Prüfung besteht aus einem Quiz, das eine Menge an Prüflingen zu absolvieren haben. Ein Quiz besteht aus einer Menge an Fragen. Bei den Fragen handelt es sich um Fragen unterschiedlichen Typs (Wahr/Falsch, MultipleChoice, ...). Bei einer richtigen Antwort bekommt der Prüfling eine der Frage zugeordneten Punktzahl. Eine Prüfung läuft so ab, dass zunächst das Quiz vorbereitet wird, d.h. es wird eine Menge an Fragen eingegeben. Anschließend wird die Prüfung durchgeführt, d.h. die Prüflinge müssen der Reihe nach die Fragen beantworten. Danach wird eine Rangliste nach den erreichten Punkten erzeugt und die Ergebnisse der Prüfung werden ausgegeben.

Das folgende Programm realisiert eine solche Prüfung:

```
abstract class Frage {

	String text; // Fragetext

	int punkte; // zu erreichende Punktzahl

	Frage(String text, int punkte) {
		this.text = text;
		this.punkte = punkte;
	}

	// Frage auf den Bildschirm ausgeben
	void frageStellen() {
		IO.println(this.text);
	}

	// Frage beantworten durch Prüfling, Antwort auswerten
	// und Punkte vergeben
	abstract void frageBeantworten(Pruefling person);

	int getPunkte() {
		return this.punkte;
	}
}

// Klasse, die Wahr/Falsch-Fragen realisiert
class WahrFalschFrage extends Frage {
	boolean richtig; // richtig oder falsch

	WahrFalschFrage(String text, int punkte, boolean richtig) {
		super(text, punkte);
```

```
            this.richtig = richtig;
      }

      // Frage beantworten durch Prüfling, Antwort auswerten
      // und Punkte vergeben
      void frageBeantworten(Pruefling person) {
            boolean ant = IO.readChar("Wahr o. Falsch (w/f)?") == 'w';
            if (ant == this.richtig) {
                  IO.println("Richt. Antw.: " + punkte + " Punkte");
                  person.neuePunkte(this.punkte);
            } else {
                  IO.println("Falsche Antwort: 0 Punkte");
            }
      }
}

// Klasse, die Multiple-Choice-Fragen realisiert
class MCFrage extends Frage {
      String[] antworten; // moegliche Antworten

      int richtigIndex; // Index der richtigen Antwort

      MCFrage(String text, int punkte, String[] antworten, int
richtigIndex) {
            super(text, punkte);
            this.antworten = antworten;
            this.richtigIndex = richtigIndex;
      }

      // Frage auf den Bildschirm ausgeben
      void frageStellen() {
            super.frageStellen();
            for (int f = 0; f < this.antworten.length; f++) {
                  IO.println("(" + f + "): " + this.antworten[f]);
            }
      }

      // Frage beantworten durch Prüfling, Antwort auswerten
      // und Punkte vergeben
      void frageBeantworten(Pruefling person) {
            int antwort = IO.readInt("Auswahl: ");
            if (antwort == this.richtigIndex) {
                  IO.println("Richtige Antwort: " + this.punkte + "
Punkte");
                  person.neuePunkte(this.punkte);
            } else {
                  IO.println("Falsche Antwort: 0 Punkte!");
                  IO.println("Richtig Antwort ist " + this.richtigIndex);
            }
      }
}

class Quiz {
      Frage[] fragen; // Menge an Fragen

      String titel; // Titel des Quizes

      int aktuellerIndex; // aktuelle Anzahl an Fragen-1

      int naechsterIndex; // Schleifenvariable

      // Konstruktor
      Quiz(String titel, int maxFragen) {
```

```
            this.titel = titel;
            this.aktuellerIndex = -1;
            this.fragen = new Frage[maxFragen];
            for (int i = 0; i < this.fragen.length; i++)
                  this.fragen[i] = null;
            this.naechsterIndex = -1;
      }

      String getTitel() {
            return this.titel;
      }

      // Frage hinzufuegen
      void neueFrage(Frage f) {
            if (this.aktuellerIndex < this.fragen.length - 1)
                  this.fragen[++this.aktuellerIndex] = f;
      }

      // liefert zyklisch die naechste Frage oder null,
      // falls keine (mehr)vorhanden ist
      Frage liefereNaechsteFrage() {
            if (this.fragen.length == 0)
                  return null;
            Frage f = null;
            if (this.naechsterIndex < this.aktuellerIndex)
                  f = this.fragen[++this.naechsterIndex];
            else
                  this.naechsterIndex = -1;
            return f;
      }
}

class Pruefling {
      String name; // Name der Person

      int punkte; // bisher erzielte Punkte

      Pruefling(String name) {
            this.name = name;
            this.punkte = 0;
      }

      String getName() {
            return this.name;
      }

      int getPunkte() {
            return this.punkte;
      }

      void neuePunkte(int anzahl) {
            this.punkte += anzahl;
      }
}

class Pruefung {

      // Hauptprogramm
      public static void main(String[] args) {
            Pruefung klausur = new Pruefung();
            klausur.vorbereiten();
            klausur.durchfuehren();
            klausur.ergebnisseBekanntgeben();
```

```
	}

	Quiz pruefung;

	Pruefling[] studenten;

	Pruefung() {
		this.pruefung = null;
		this.studenten = null;
	}

	void vorbereiten() {
		IO.println("Fragen eingeben");
		IO.println("----------------");
		String titel = IO.readString("Titel des Quizes: ");
		int anzahl = IO.readInt("Anzahl Fragen: ");
		this.pruefung = new Quiz(titel, anzahl);
		// Fragen eingeben
		for (int i = 0; i < anzahl; i++) {
			Frage f = this.frageErzeugen(i + 1);
			this.pruefung.neueFrage(f);
		}
	}

	public Frage frageErzeugen(int nummer) {
		// spaeter Factory-Pattern
		int typ = IO.readInt("Fragetyp: Wahr/Falsch (1), Multiple
Choice (2)?");
		switch (typ) {
		case 1:
			return erzeugeWahrFalschFrage(nummer);
		case 2:
		default:
			return erzeugeMCFrage(nummer);
		}
	}

	private Frage erzeugeWahrFalschFrage(int nummer) {
		String text = IO.readString("Frage " + (nummer) + ": ");
		boolean wahr = IO.readChar("Wahr/falsch(w/f)?") == 'w';
		int punkte = IO.readInt("Erreichbare Punkte: ");
		return new WahrFalschFrage(text, punkte, wahr);
	}

	private Frage erzeugeMCFrage(int nummer) {
		String text = IO.readString("Frage " + (nummer) + ": ");
		int anzahl = IO.readInt("Anzahl an Antworten: ");
		String[] antworten = new String[anzahl];
		for (int i = 0; i < anzahl; i++) {
			antworten[i] = IO.readString("Antwort " + i + ": ");
		}
		int richtigIndex = IO.readInt("Index der richtigen Antwort: ");
		int punkte = IO.readInt("Erreichbare Punkte: ");
		return new MCFrage(text, punkte, antworten, richtigIndex);
	}

	void durchfuehren() {
		IO.println("Pruefung");
		IO.println("--------");
		int anzahl = IO.readInt("Anzahl Prueflinge: ");
		this.studenten = new Pruefling[anzahl];
		// alle Prueflinge abfragen
		for (int i = 0; i < anzahl; i++) {
```

```
                IO.println("Pruefling " + (i + 1) + " ist an der Reihe");
                this.studenten[i] = new Pruefling(IO.readString("Name:
"));
                Frage f = null;
                // alle Fragen der Pruefung stellen
                while ((f = this.pruefung.liefereNaechsteFrage()) !=
null) {
                      f.frageStellen();
                      f.frageBeantworten(this.studenten[i]);
                }
          }
     }

     void ergebnisseBekanntgeben() {
          this.ranglisteErstellen();
          IO.println("Pruefungsergebnisse");
          IO.println("--------------------");
          IO.println("Quiz: " + this.pruefung.getTitel());
          for (int i = 0; i < this.studenten.length; i++) {
                IO.println("Platz " + (i + 1) + ": " +
this.studenten[i].getName()
                           + " mit " + this.studenten[i].getPunkte() + "
Punkten");
          }
     }

     private void ranglisteErstellen() {
          // Bubblesort nach erreichten Punkten
          boolean veraendert = false;
          do {
                veraendert = false;
                for (int i = 0; i < this.studenten.length - 1; i++) {
                      if (this.studenten[i].getPunkte() <
this.studenten[i + 1]
                                  .getPunkte()) {
                            Pruefling help = this.studenten[i];
                            this.studenten[i] = this.studenten[i + 1];
                            this.studenten[i + 1] = help;
                            veraendert = true;
                      }
                }
          } while (veraendert);
     }
}
```

Aufgabe 1:

Oben wurde ein Java-Programm vorgestellt, mit dem eine Prüfung simuliert werden kann. An Fragetypen unterstützt dieses Programm Wahr/Falsch- und MultipleChoice-Fragen. Hierzu wurden entsprechende Klassen von einer abstrakten Klasse `Frage` abgeleitet und die daraus resultierende Polymorphie ausgenutzt.

Erweitern Sie das Programm so, dass als weiterer Fragetyp die **Mehrfachauswahl** unterstützt wird, d.h. zu einer Frage werden mehrere Antworten gegeben, von denen keine, eine oder auch mehrere korrekt sind. Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs "Mehrfachauswahl" gestellt werden können.

Aufgabe 2:

Oben wurde ein Java-Programm vorgestellt, mit dem eine Prüfung simuliert werden kann. An Fragetypen unterstützt dieses Programm Wahr/Falsch- und MultipleChoice-Fragen. Hierzu wurden entsprechende Klassen von einer abstrakten Klasse `Frage` abgeleitet und die daraus resultierende Polymorphie ausgenutzt.

Erweitern Sie das Programm so, dass als weiterer Fragetyp der **Lückenfrage** unterstützt wird, d.h. in einer Aussage fehlt ein Wort, das ein Prüfling zu ergänzen hat. Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs "Lückenfrage" gestellt werden können.

Aufgabe 3:

Oben wurde ein Java-Programm vorgestellt, mit dem eine Prüfung simuliert werden kann. An Fragetypen unterstützt dieses Programm Wahr/Falsch- und MultipleChoice-Fragen. Hierzu wurden entsprechende Klassen von einer abstrakten Klasse `Frage` abgeleitet und die daraus resultierende Polymorphie ausgenutzt.

Erweitern Sie das Programm so, dass als weiterer Fragetyp **Zahlenfolgen-erweiterungen** unterstützt werden, d.h. es werden *n* Zahlen einer Zahlenfolge bekannt gegeben und der Prüfling muss daraus die zugrunde liegende Zahlenfolge identifizieren und die *n+1*-te Zahl der Zahlenfolge eingeben. Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs " **Zahlenfolgenerweiterungen** " gestellt werden können.

Beispiel (Eingaben in grün)**:**

```
Frage 1 (Zahlenfolgen ergaenzen)
Anzahl an vorgegebenen Zahlen: 4
Zahl 0: 1
Zahl 1: 2
Zahl 2: 3
Zahl 3: 4
Korrekte Folgezahl: 5
Erreichbare Punkte: 10

...

Identifizieren Sie die Zahlenfolge und geben Sie die nächste Zahl der
Folge ein!
1 2 3 4 5
Richtige Antwort: 10 Punkte
```

Aufgabe 4:

Oben wurde ein Java-Programm vorgestellt, mit dem eine Prüfung simuliert werden kann. An Fragetypen unterstützt dieses Programm Wahr/Falsch- und MultipleChoice-Fragen. Hierzu wurden entsprechende Klassen von einer abstrakten Klasse `Frage` abgeleitet und die daraus resultierende Polymorphie ausgenutzt.

Erweitern Sie das Programm so, dass als weiterer Fragetyp die **Ordnungsfrage** unterstützt wird. Bei der Ordnungsfrage wird dem Prüfling eine Aufgabe und eine Menge mit n Antworten gegeben, die der Benutzer dann in die korrekte Reihenfolge

bringen muss (bekannt durch die Spielerauswahlfrage bei „Wer-wird-Millionär?“). Leiten Sie eine entsprechende Klasse von der Klasse `Frage` ab und erweitern Sie die Methode `frageErzeugen` der Klasse `Pruefung`, so dass auch Fragen des Typs "Ordnungsfrage" gestellt werden können.

Beispiel (Eingaben in grün):

```
Frage 1: Ordnen Sie die folgenden deutschen Städte von Nord nach Süd!
Anzahl an Antworten: 4
Antwort 0: Oldenburg
Antwort 1: Kiel
Antwort 2: München
Antwort 3: Frankfurt
1. Index: 1
2. Index: 0
3. Index: 3
4. Index: 2
Erreichbare Punkte: 20

...

Ordnen Sie die folgenden deutschen Städte von Nord nach Süd!
(0): Oldenburg
(1): Kiel
(2): München
(3): Frankfurt
1. : 1
2. : 0
3. : 3
4. : 2
Richtige Reihenfolge: 20 Punkte
```

Aufgabe 5:

In dieser Aufgabe geht es um den Aufbau von Pipelines für die Ausführung von Filter-Operationen auf Strings, wie Streichen von führenden und abschließenden Leerzeichen (Trimmen), Umwandeln von Klein- in Großbuchstaben, Ersetzen von ‚ß’ („sz“) durch „ss“, Verschlüsseln oder ähnliches; Beispiel:

„ Fuß“ → „Fuß“ → „Fuss“ → „FUSS“ → ...
^ ^ ^

Trimmen SZ-Ersetzen UpperCase

Die String-Operationen der Pipeline sollen dabei beliebig erweiterbar sein, die Reihenfolge ihrer Ausführung soll beliebig angeordnet werden können und die Pipeline soll auch beliebig lang werden können.

Realisiert werden kann eine solche Pipeline dadurch, dass es für jede String-Filter-Operation eine Klasse gibt, die in einer entsprechenden Methode `verarbeiten` die Operation auf einen als Parameter übergebenen String durchführt, Beispiel:

```
class Trimmer {
  public String verarbeiten(String str) {
      // liefere getrimmten str
  }
}
```

Allerdings sind zum Aufbau einer solchen Pipeline weitere Dinge notwendig, über die Sie sich Gedanken machen sollen. Letztendlich soll es möglich sein, mit den von Ihnen zu entwickelnden Klassen Programme folgender Gestalt zu schreiben:

```
Filter pipeline = new ToUpperCaser(new SZErsetzer(new Trimmer()));
while (true) {
     String eingabe = IO.readString("String: ");
     System.out.println(pipeline.verarbeiten(eingabe));
}
```

In diesem Beispiel wird (bei der Definition entsprechender Klassen `ToUpperCaser`, `SZErsetzer` und `Trimmer`) die obige Pipeline aufgebaut und beim Aufruf von pipeline.verarbeiten aktiviert.

Andere mögliche Pipelines wären (bei entsprechenden Klassen):

```
Filter pipeline = new Trimmer(new ToUpperCaser(new Caesar(1)));
Filter pipeline =
    new Caesar(2, new ToLowerCaser(new Trimmer(new UmlautErsetzer())));
```

Aufgabe: Überlegen Sie aufbauend auf den Konzepten der Polymorphie und des dynamischen Bindens ein Konzept zur Realisierung obiger Pipelines.

Implementieren Sie konkret die Klassen `ToUpperCaser`, `SZErsetzer` und `Trimmer`, so dass sich das Beispielprogramm übersetzen lässt und die gewünschte Pipeline realisiert.

Aufgabe 6:

Schauen Sie sich das folgende Programm an:

```
import java.util.Calendar;
import java.util.GregorianCalendar;

class AppManager {

     public void ausfuehren() {
           while (true) {
                String appName =
                   IO.readString("Welche App soll ausgeführt werden: ");
                if (appName.equals("A")) {
                     ausfuehrenAppA();
                } else if (appName.equals("B")) {
                     ausfuehrenAppB();
                }
           }
     }

     void ausfuehrenAppA() { // Taschenrechner
           int zahl1 = IO.readInt("Zahl 1: ");
           int zahl2 = IO.readInt("Zahl 2: ");
           System.out.println(zahl1 + " + " + zahl2 + " = " +
                                (zahl1 + zahl2));
     }

     void ausfuehrenAppB() { // Uhranzeige
           Calendar kal = new GregorianCalendar();
           System.out.println("Es ist " +
                                kal.get(Calendar.HOUR_OF_DAY) + "Uhr");
```

```
    }
}

class Smartphone {
    public static void main(String[] args) {
        AppManager manager = new AppManager();
        manager.ausfuehren();
    }
}
```

Hintergrund dieses Programms ist ein Smartphone, auf dem ein Programm (`AppManager`) läuft, das die Auswahl und Ausführung von Apps steuert. Problem des AppManagers ist seine fehlende Erweiterbarkeit ohne Sourcecode-Änderungen. Immer wenn eine neue App hinzukommen soll, muss der Sourcecode des AppManagers geändert werden.

Setzen Sie die Konzepte der abstrakten Klassen, der Polymorphie und des dynamischen Bindens ein, um dieses Problem zu beheben. Konkret soll durch folgendes Smartphone-Programm der neue AppManager (Klasse `AppManagerOO`) gestartet werden können:

```
class SmartphoneOO {

    public static void main(String[] args) {
        AppManagerOO manager = new AppManagerOO();
        manager.appHinzufuegen(new AppA("A"));
        manager.appHinzufuegen(new AppB("B"));
        manager.ausfuehren();
    }
}
```

Apps werden hierbei durch geeignete Klassen repräsentiert. Eine Erweiterung des AppManagers ist in diesem Fall ohne Sourcecode-Änderungen der Klasse `AppManager` möglich.

Konkrete Aufgaben:

Definieren Sie die fehlenden Klassen `AppManagerOO`, `AppA` und `AppB` so, dass sich das Programm `SmartphoneOO` fehlerfrei compilieren lässt und bezogen auf die Ein- und Ausgabe dasselbe tut, wie das Programm `Smartphone`.

Die Klasse `AppManagerOO` soll dabei ohne Sourcecode-Änderungen erweiterbar sein, d.h. auch andere neue hinzugefügte Apps auswählen und ausführen können.

Die Klasse `AppA` soll konkret den oben angedeuteten Taschenrechner implementieren.

Die Klasse `AppB` soll konkret die oben angedeutete Uhranzeige implementieren.

Aufgabe 7:

In dieser Aufgabe geht es um den Aufbau von Pipelines für die Ausführung von Filter-Operationen in der Bildverarbeitung. Als Filter-Operationen sollen dabei Operationen betrachtet werden, die Bilder als eine Folge von Bytes übergeben bekommen, die entsprechende Operation auf den Bytes anwenden und das veränderte Bild wieder als Folge von Bytes zurückliefern. Beispiele sind der Gauß-

Filter, der Laplace-Filter, der Median-Filter oder der Binomial-Filter. Pipelines von Filter-Operationen sind dabei mehrere unmittelbar hintereinander ausgeführte Filter-Operationen, bspw. ein Bild wird zunächst über einen Gauß-, dann über einen Laplace- und dann über einen Binomial-Filter verändert.

Die Filter-Operationen der Pipeline sollen dabei beliebig erweiterbar sein, die Reihenfolge ihrer Ausführung soll beliebig angeordnet werden können und die Pipeline soll auch beliebig lang werden können.

Realisiert werden kann eine solche Pipeline dadurch, dass es für jede Filter-Operation eine Klasse gibt, die in einer entsprechenden Methode `transformieren` die entsprechende Operation auf ein als Parameter übergebenes byte-Array durchführt, und das resultierende byte-Array zurückliefert; Beispiel:

```
class MedianFilter {
     public byte[] transformieren(byte[] bytes) {
           // Durchfuehrung der Median-Filter-Operationen
      }
}
```

Allerdings sind zum Aufbau einer solchen Pipeline weitere Dinge notwendig, über die Sie sich Gedanken machen sollen. Letztendlich soll es möglich sein, mit den von Ihnen zu entwickelnden Klassen Programme folgender Gestalt zu schreiben:

```
public class Bildverarbeitung {

     public static void main(String[] args) {
           Filter pipeline = new GaussFilter(
                           new LaplaceFilter(new MedianFilter()));
           while (true) {
                 byte[] pictureData = readFile("holiday.gif");
                 show(pipeline.transformieren(pictureData));
           }
     }

     static byte[] readFile(String dateiname) {
           return null; // hier unwichtig
     }

     static void show(byte[] picture) {
     }

}
```

In diesem Beispiel wird (bei der Definition entsprechender Klassen `GaussFilter`, `LaplaceFilter` und `MedianFilter`) eine Pipeline aufgebaut und beim Aufruf von `pipeline.transformieren` aktiviert, d.h. auf dem Bild (`pictureData`) wird zunächst der Median-Filter, dann der Laplace-Filter und dann der Gauß-Filter angewendet.

Eine andere mögliche Pipeline (erst der Binomial- und dann der Laplace-Filter) wäre (bei entsprechenden Klassen):

```
Filter pipeline = new LaplaceFilter(new BinomialFilter());
```

Aufgabe:

Überlegen Sie aufbauend auf den Konzepten der Polymorphie und des dynamischen Bindens ein Konzept zur Realisierung obiger Pipelines.

Implementieren Sie neben u.U. weiteren notwendigen Klassen die konkreten Klassen `GaussFilter`, `LaplaceFilter` und `MedianFilter`, so dass sich das erste Beispielprogramm übersetzen lässt und die gewünschte Pipeline realisiert. Ersetzen Sie die eigentlich korrekte Implementierung der entsprechenden Filter-Operationen dadurch, dass die 3 Filter die der Methode `transformieren` übergebenen Bytes einfach jeweils um den Wert 1 bzw. 2 bzw. 3 erhöhen.

Aufgabe 8:

Schauen Sie sich das folgende Programm an:

```
class WebBrowser {

	public void starten() {
		while (true) {
			String url = IO.readString("URL: ");
			String webseite = getWebSeite(url);
			if (webseite.contains("flash")) {
				startePlugInFlash(webseite);
			}
			if (webseite.contains("pdf")) {
				startePlugInPDF(webseite);
			}
			darstellen(webseite);
		}
	}

	private String getWebSeite(String url) {
		String seite = IO.readString("HTML-Code: ");
		return seite;
	}

	void startePlugInFlash(String webseite) { // Flash Player
		System.out.println("Flash Player gestartet");
	}

	void startePlugInPDF(String webseite) { // Adobe Reader
		System.out.println("Adobe PDF-Reader gestartet");
	}

	void darstellen(String webseite) {
		System.out.println(webseite);
	}
}

class WebBrowserSimulation {
	public static void main(String[] args) {
		WebBrowser browser = new WebBrowser();
		browser.starten();
	}
}
```

Hintergrund dieses Programms ist eine einfache Simulation eines WebBrowers, der Webseiten laden (hier einlesen) und darstellen (hier ausgeben) kann. Wird zur Darstellung einer Webseite ein Plug-In benötigt, wird dieses zuvor gestartet.

Problem des WebBrowsers ist seine fehlende Erweiterbarkeit ohne Sourcecode-Änderungen. Immer wenn ein neues Plug-In hinzukommen soll, muss der Sourcecode der Klasse `WebBrowser`, insbesondere die Methode `starten`, geändert werden.

Setzen Sie das Konzept der Polymorphie und des dynamischen Bindens ein, um dieses Problem zu beheben. Konkret soll durch folgendes Programm ein (aus Programmiersicht) besserer WebBrowser (Klasse `WebBrowserOO`) genutzt werden können:

```
class WebBrowserOOSimulation {
     public static void main(String[] args) {
           WebBrowserOO browser = new WebBrowserOO();
           browser.registrieren(new FlashPlugIn());
           browser.registrieren(new PDFPlugIn());
           browser.starten();
     }
}
```

Plug-Ins werden hierbei durch geeignete Klassen repräsentiert, von denen Objekte instanziiert und beim WebBrowser registriert werden. Eine Erweiterung des WebBrowsers in Form der Unterstützung weiterer Plug-Ins ist bei geschicktem Einsatz des Konzeptes des dynamischen Bindens ohne Sourcecode-Änderungen der Klasse `WebBrowserOO` möglich. Zusätzliche Plug-Ins müssen dann lediglich vor dem Starten des WebBrowsers beim WebBrowser registriert werden (Bsp.: `browser.registrieren(new QuickTimePlugIn());`

Konkrete Aufgaben:

Definieren Sie die fehlenden Klassen `WebBrowserOO`, `FlashPlugIn` und `PDFPlugIn` so, dass sich das Programm `WebBrowserOOSimulation` fehlerfrei compilieren lässt und bezogen auf die Ein- und Ausgabe dasselbe tut, wie das Programm `WebBrowserSimulation`.

Die Klasse `WebBrowserOO` soll dabei ohne Sourcecode-Änderungen erweiterbar sein, d.h. auch andere neue registrierte Plug-Ins starten können.

Objekte der Klasse `FlashPlugIn` sollen beim Starten des Plug-Ins den Text „Flash Player gestartet“ ausgeben.

Objekte der Klasse `PDFPlugIn` sollen beim Starten des Plug-Ins den Text „Adobe PDF-Reader gestartet“ ausgeben.

Aufgabe 9:

In dieser Aufgabe geht es um den Aufbau von Funktionenketten. Funktionenketten seien dabei Funktionen, die eine geordnete Menge von mathematischen Funktionen enthalten, die der Reihe nach auf den berechneten Werten der Vorgängerfunktion ausgeführt werden, startend mit einem gegebenen Funktionswert.

Als Funktionen werden dabei in dieser Aufgabe beliebige Funktionen betrachtet, die einen double-Wert auf einen anderen double-Wert abbilden. Die Reihenfolge der Ausführung der Funktionen soll beliebig angeordnet werden können. Die Funktionenkette soll prinzipiell beliebig lang werden können.

Beispiel: Es gebe die Funktionen *f*, *g*, *h* und *p*.

Funktionskette *k1* bestehe aus drei Funktionen in der Reihenfolge *f*, *g* und *h*. Dann gilt: `k1(x) = f(g(h(x)))`

Funktionskette *k2* bestehe aus vier Funktionen in der Reihenfolge *g*, *f*, *g* und *p*. Dann gilt: `k2(x) = g(f(g(p(x))))`

Sie sollen sich in dieser Aufgabe Gedanken darüber machen, wie solche Funktionenketten implementiert werden können. Letztendlich soll es möglich sein, mit den von Ihnen zu entwickelnden Klassen Programme folgender Gestalt zu schreiben:

```
Funktion kette =
  new CosinusFunktion(new WurzelFunktion(new SinusFunktion()));
while (true) {
     double x = IO.readDouble("Wert: ");
     System.out.println(kette.berechnen(x));
}
```

In diesem Beispiel wird (bei der Definition entsprechender Klassen *Funktion*, *CosinusFunktion*, *WurzelFunktion*, *SinusFunktion*) eine Funktionenkette aufgebaut und durch den Aufruf von `kette.berechnen` aktiviert. Berechnet wird der Wert:

`cos(wurzel(sin(x)))` für vom Benutzer eingegebene double-Werte x.

Andere mögliche Funktionenketten wären (bei entsprechenden Klassen):

```
Funktion kette =
  new WurzelFunktion(new WurzelFunktion(new TangensFunktion()));

Funktion kette =
  new SinFunktion(new QuadratFunktion(new IdentitaetsFunktion()));
```

Aufgabe: Überlegen Sie aufbauend auf den Konzepten der Polymorphie und des dynamischen Bindens ein Konzept zur Realisierung obiger Funktionenketten. Implementieren Sie neben der notwendigen Klasse *Funktion* zwei konkrete Klassen *QuadratFunktion* und *SinusFunktion*.