Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE32-DynamischesBinden
(Stand 16.07.2012)

Aufgabe 1:

Bei einem Spiel namens "Sieben" geht es darum, dass eine Menge an Spielern reihum Zahlen aufsagen und zwar auf folgende Art und Weise: Begonnen wird mit der Zahl 0. Der jeweils nächste Spieler, der an der Reihe ist, muss dann die nächst höhere Zahl nennen, die keine Ziffer 7 enthält und nicht durch 7 teilbar ist. Ansonsten scheidet er aus. Sieger ist/sind der/die Spieler, wer als letzter Spieler nicht ausgeschieden ist oder beim Erreichen der Zahl 10000 noch nicht ausgeschieden ist. Mit Hilfe des folgenden Programms lässt sich das Spiel "Sieben" am Computer spielen.

```
class SpielSieben {
	// Hauptprogramm
	public static void main(String[] args) {
		// ueber die Argumente werden die Spielernamen uebergeben
		if (args.length < 2) {
			IO.println("Bitte mindestens zwei Spielernamen uebergeben");
			return;
		}
		// Spieler erzeugen
		Spieler[] spieler = new Spieler[args.length];
		for (int i = 0; i < args.length; i++) {
			spieler[i] = new Spieler(args[i]);
		}
		// Spiel durchfuehren
		spielen(spieler);
	}

	// Durchfuehrung eines kompletten Spiels mit den
	// uebergebenen Spielern
	public static void spielen(Spieler[] spieler) {
		int anzahlAusgeschieden = 0;
		int zahl = 0; // begonnen wird mit der Zahl 0
		hauptschleife: while (zahl < 10000) {
			// ab 10000 wird noch eine Runde gespielt
			// alle Spieler kommen nacheinander an die Reihe
			for (int i = 0; i < spieler.length; i++) {
				if (!spieler[i].istAusgeschieden()) {
					int ergebnis = spieler[i].naechsteZahl(zahl);
					if (!check(zahl, ergebnis)) {
						spieler[i].ausscheiden();
						IO.println("Falsch! " + spieler[i]
								+ " ist ausgeschieden!");
						anzahlAusgeschieden++;
						if (anzahlAusgeschieden == spieler.length -
1) {
							// alle bis auf einen Spieler sind
ausgeschieden
							break hauptschleife; // Spiel zuende
						}
```

```
                                } else { // eingegebene Zahl war korrekt
                                        zahl = ergebnis; // zahl hochsetzen
                                }
                        }
                }
        }
        gibSiegerBekannt(spieler);
    }

    // Bekanntgabe der/des Siegers (wer nicht ausgeschieden ist)
    public static void gibSiegerBekannt(Spieler[] spieler) {
        for (int i = 0; i < spieler.length; i++) {
            if (!spieler[i].istAusgeschieden()) {
                IO.println(spieler[i] + " ist Sieger!");
            }
        }
    }

    /**
     * Ueberpruefung eines Spielzugs
     *
     * @param aktuelleZahl
     *            die aktuelle Zahl des Spiels
     * @param gelieferteZahl
     *            die vom Spieler angegebene Zahl
     * @return genau dann true, wenn gelieferteZahl die naechst hoehere Zahl
     *         nach aktuelleZahl ist, die keine Ziffer 7 enthaelt und nicht
     *         durch 7 teilbar ist
     */
    public static boolean check(int aktuelleZahl, int gelieferteZahl) {

    }
}

class Spieler {
    private String name;

    private boolean ausgeschieden;

    public Spieler(String name) {
        this.name = name;
        this.ausgeschieden = false;
    }

    public String toString() {
        return this.name;
    }

    public int naechsteZahl(int vorherigeZahl) {
        return IO.readInt(this + ": Zahl nach " + vorherigeZahl + ": ");
    }

    public void ausscheiden() {
        this.ausgeschieden = true;
    }

    public boolean istAusgeschieden() {
        return this.ausgeschieden;
    }
}
```

2 Teilaufgaben:

1. Implementieren Sie die Methode *check* der Klasse SpielSieben

2. Leiten Sie von der Klasse *Spieler* eine Klasse *Programm* ab, die die Methode *naechsteZahl* so überschreibt, dass Objekte der Klasse *Programm* immer als

Sieger des Spiels hervorgehen würden. Ändern Sie die Methode *main* (und nur die!) so ab, dass immer ein Programm als Spieler mitspielt

Aufgabe 2:

Das folgende Java-Programmfragment

```
public class Aufgabe2 {

	public static void main(String[] args) {
		Name n1 = new Name("Mehl");
		Name n2 = new AbkName("Messerspitze", "Msp");
		Name n3 = new AbkName("Tasse");
		n1.schreib();
		n2.schreib();
		n3.schreib();
	}
}
```

erzeugt die Ausgabe

```
Mehl
Msp.: Messerspitze
Tasse
```

Implementieren Sie die beiden Klassen *Name* und *AbkName*. Die Klassen sollen private Attribute und die für das obige Programmfragment und die gezeigte Ausgabe notwendigen Konstruktoren und Methoden besitzen. Die Bildschirmausgabe soll dabei nicht bereits im Konstruktor sondern in der Methode *schreib* erfolgen. Die Klasse *Name* speichert einen im Konstruktor übergebenen Namen. Die Klasse *AbkName* speichert neben einem Namen (Konstruktor mit einem Parameter) zusätzlich unter Umständen noch eine Abkürzung für den Namen (Konstruktor mit zwei Parametern).

Aufgabe 3:

Welche Ausgabe produziert folgendes Java-Programm! Begründen Sie Ihre Entscheidung kurz!

```
class X {
	int i = 5;

	public X() {
		this.print();
	}
```

```
	public void print() {
		System.out.println("X" + i);
	}

	public void call() {
		this.print();
	}
}

class Y extends X {
	int i = 8;

	public Y() {
		this.print();
	}

	public void print() {
		super.print();
		System.out.println("Y" + i);
	}

	public static void main(String[] args) {
		X obj = new X();
		obj.call();
		obj = new Y();
		obj.call();
	}
}
```

Aufgabe 4:

Implementieren Sie Klassen, die es erlauben, beliebige Längenmaße miteinander addieren zu können. Konkret sollen die Klassen so definiert werden, dass sich folgendes Programm übersetzen und mit dem angegebenen Ergebnis ausführen lassen kann:

```
public class Aufgabe4 {

	public static void main(String[] args) {
		Meter meter = new Meter(2);
		meter.print(); // Ausgabe: 2.0 Meter

		Fuss fuss = new Fuss(4);
		fuss.print(); // Ausgabe: 4.0 Fuss

		Elle elle = new Elle(6);
		elle.print(); // Ausgabe: 6.0 Ellen

		meter.addiere(fuss); // 2.0 Meter + 4.0 Fuss
		meter.print(); // Ausgabe: 3.216 Meter
```

```
            elle.addiere(meter); // 6.0 Ellen + 3.216 Meter
            elle.print(); // Ausgabe: 10.872726 Ellen

      }
}
```

Dabei gilt: 1 Fuß = 0.304 Meter und 1 Elle = 0.66 Meter.

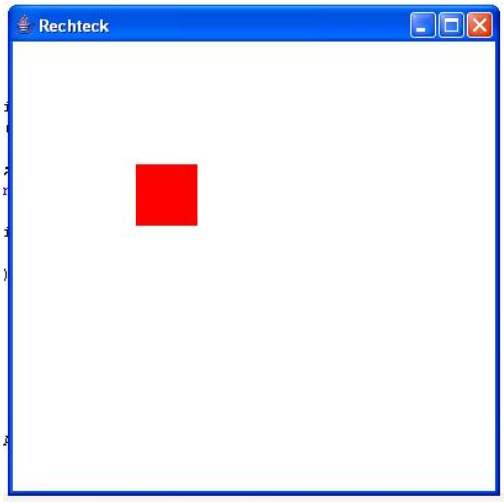
Achten Sie auf Erweiterbarkeit, d.h. es soll möglich sein, weitere Längenmaßklassen wie bspw. Landmeilen oder Yards zu definieren und ins Hauptprogramm zu integrieren, ohne die existierenden Klassen ändern zu müssen!

Aufgabe 5:

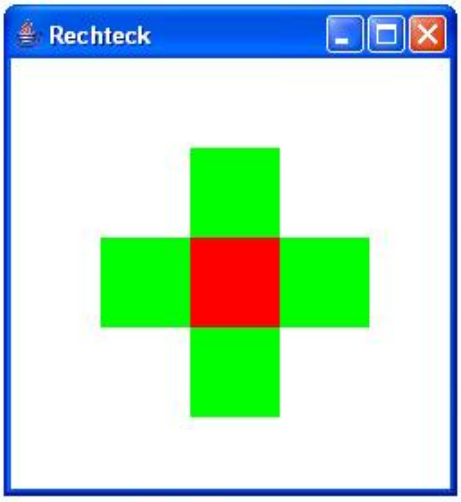
Schauen Sie sich das Programm Rechteck.java an:

```
import java.awt.*;

class Rechteck {
      static Frame f;
      static Zeichenflaeche flaeche;

      public static void main(String[] args) {
            f = new Frame("Rechteck");
            f.setSize(400, 400);
            flaeche = new Zeichenflaeche();
            hinzufuegen(flaeche);
            f.setVisible(true);
      }

      public static void hinzufuegen(Zeichenflaeche flaeche) {
            f.add(flaeche);
      }

}

class Zeichenflaeche extends Canvas {

      public void paint(Graphics g) {
            g.setColor(Color.red);
            g.fillRect(100, 100, 50, 50);
      }
}
```

Der Aufruf des Programms bewirkt folgende Ausgabe:



Schauen Sie sich das Programm an und erweitern Sie es, so dass um das rote Rechteck 4 grüne Rechtecke gezeichnet werden:



Achtung: Arbeiten Sie dabei mit den Konzepten der Vererbung und der Polymorphie. Sie dürfen in dem bereit gestellten Programm nur die dritte Zeile der main-Funktion ändern. Leiten Sie daher von der Klasse `Zeichenflaeche` eine neue Klasse ab, die die Methode `paint` überschreibt.

Aufgabe 6:

Schauen Sie sich das folgende Java-Programm an:

```
class Spieler {
	public int liefereZahl(int aktZahl) {
		return IO.readInt("1 oder 2 eingeben:");
	}
}

class NimmSpiel {

	// Hauptprogramm
	public static void main(String[] args) {
		Spieler spielerA = new Spieler();
```

```
		Spieler spielerB = new Spieler();
		NimmSpiel spiel = new NimmSpiel(spielerA, spielerB, IO
				.readInt("Startzahl: "));
		spiel.go();
	}

	// Attribute
	Spieler spielerA, spielerB, aktSpieler;

	int zahl;

	public NimmSpiel(Spieler spielerA, Spieler spielerB, int zahl) {
		this.spielerA = spielerA;
		this.spielerB = spielerB;
		this.aktSpieler = spielerA;
		this.zahl = zahl;
	}

	public void go() {
		IO.println("Spieler A ist an der Reihe");
		while (this.zahl > 0) {
			IO.println("Zahl: " + this.zahl);
			int spielerZahl = this.aktSpieler.liefereZahl(this.zahl);
			if (!(spielerZahl == 1 || spielerZahl == 2)) {
				IO.println("Verloren: ungueltige Zahl!");
				return;
			}
			this.zahl -= spielerZahl;
			if (this.zahl > 0) {
				if (this.aktSpieler == spielerA) {
					this.aktSpieler = spielerB;
					IO.println("Spieler B ist an der Reihe");
				} else {
					this.aktSpieler = spielerA;
					IO.println("Spieler A ist an der Reihe");
				}
			}
		}
		if (this.aktSpieler == spielerA)
			IO.println("Spieler A hat gewonnen!");
		else
			IO.println("Spieler B hat gewonnen!");
	}
}
```

Mit diesem Programm können zwei menschliche Spieler gegeneinander das Nimm-Spiel spielen. Beim Nimm-Spiel müssen zwei Spieler abwechselnd von einem Haufen mit Steinen jeweils 1 oder 2 Steine entfernen. Es gewinnt derjenige Spieler, der die letzten Steine entnimmt.

Erweitern Sie das Programm derart, dass Menschen gegen Programme und Programme gegen Programme das Nimm-Spiel spielen können, wobei die gegebene Methode `go` die Spieldurchführung übernimmt. Dabei dürfen Sie jedoch bis auf die ersten zwei Anweisungen der main-Prozedur keine weiteren Anweisungen ändern! Sie dürfen lediglich eine neue Klasse `Programm` hinzufügen. Nutzen Sie also die Konzepte der Vererbung in Verbindung mit Polymorphie und dynamischem Binden!

Hinweis: Gewinnen kann man dadurch, dass man versucht, eine durch 3 teilbare Anzahl an Steinen auf dem Steinhaufen zu belassen. Implementieren Sie für die Klasse `Programm` diese Gewinnstrategie!

Aufgabe 7:

Gegeben sei folgender Source-Code:

```
class ZahlenSpieler {
     public int naechsteGeradeZahl(int aktuelleZahl) {
          return aktuelleZahl;
     }
}

class ZahlenSpiel {
     int zahl;

     ZahlenSpieler a, b;

     public ZahlenSpiel(ZahlenSpieler a, ZahlenSpieler b) {
          this.a = a;
          this.b = b;
          this.zahl = 0;
     }

     public void spielen() {
          ZahlenSpieler akt = a;
          while (true) {
               int erg;
               if ((erg = akt.naechsteGeradeZahl(this.zahl)) !=
                              this.zahl + 2) {
                    System.out.println("Falsch; verloren!");
                    break;
               }
               this.zahl = erg;
               System.out.println(this.zahl);
               if (akt == a)
                    akt = b;
               else
                    akt = a;
          }
     }

     public static void main(String[] args) {
          ZahlenSpieler a = ...
          ZahlenSpieler b = ...
          ZahlenSpiel spiel = new ZahlenSpiel(a, b);
          spiel.spielen();
  }

}
```

In diesem Spiel sollen zwei Spieler jeweils abwechselnd die nächste gerade Zahl berechnen. Leiten Sie zwei Klassen von der „falschen“ Klasse Spieler ab; zum ersten für einen menschlichen Spieler, der die Zahl über die Tastatur eingibt, und zum zweiten für ein Programm, das die nächste gerade Zahl berechnet. Ersetzen Sie die

Pünktchen in der main-Funktion, so dass ein Mensch gegen ein Programm spielen kann.

Aufgabe 8:

Schauen Sie sich zunächst folgendes Hauptprogramm an:

```
class SprachenTest {

	public static void main(String[] args) {
		Franzoesisch f = new Franzoesisch("je");
		Deutsch d = new Deutsch("bin");
		Englisch e = new Englisch("happy");

		Deutsch satz1 = new Deutsch(f);
		satz1.append(d);
		satz1.append(e);

		Englisch satz2 = new Englisch(f);
		satz2.append(d);
		satz2.append(e);

		Franzoesisch satz3 = new Franzoesisch(f);
		satz3.append(d);
		satz3.append(e);

		Deutsch satz4 = new Deutsch(e);
		satz4.append(d);
		satz4.append(f);

		System.out.println(satz1);
		// Ausgabe: "ich bin gluecklich"

		System.out.println(satz2);
		// Ausgabe: "i am happy"

		System.out.println(satz3);
		// Ausgabe: "je suis heureux"

		System.out.println(satz4);
		// Ausgabe: "gluecklich bin ich"
	}
}
```

Implementieren Sie geeignete Klassen, so dass das Hauptprogramm die vorgegebenen Ausgaben liefert. Der Wortschatz sowie die notwendige Übersetzung seien dabei auf die drei vorkommenden Begriffe beschränkt. Realisieren Sie die Klassen so, dass bspw. eine weitere Klasse Hessisch hinzugefügt und im Hauptprogramm benutzt werden kann, ohne dass die anderen existierenden Klassen geändert werden müssen. Nutzen Sie dazu insbesondere die Möglichkeiten der Polymorphie und des dynamischen Bindens.

Aufgabe 9:

Implementieren Sie eine Funktion, die überprüft, ob beliebige (als Parameter übergebene) Funktionen von `int` nach `int` im Intervall von `von` bis `bis` eine Nullstelle haben. `von` und `bis` werden auch als Parameter übergeben.

Aufgabe 10:

Bei einem Spiel namens "Sieben" geht es darum, dass eine Menge an Spielern reihum Zahlen aufsagen und zwar auf folgende Art und Weise: Begonnen wird mit der Zahl 1. Der jeweils nächste Spieler, der an der Reihe ist, muss dann die nächst höhere Zahl nennen, die keine Ziffer 7 enthält und nicht durch 7 teilbar ist. Ansonsten scheidet er aus. Sieger ist/sind der/die Spieler, wer beim Erreichen der Zahl 10000 noch nicht ausgeschieden ist.

Implementieren Sie das Spiel (auf eine objektorientierte Art und Weise),so dass Menschen und/oder Programme das Spiel „Sieben“ spielen können.

Beispiel für einen möglichen Programmablauf mit 2 menschlichen Spielern dibo und Hans sowie einem Roboter/Programm (Eingaben in <>):

```
dibo: Naechste Zahl eingeben (aktuelle Zahl = 0): <1>
Hans: Naechste Zahl eingeben (aktuelle Zahl = 1): <2>
Roboter: Ermittelte Zahl = 3
dibo: Naechste Zahl eingeben (aktuelle Zahl = 3): <4>
Hans: Naechste Zahl eingeben (aktuelle Zahl = 4): <5>
Roboter: Ermittelte Zahl = 6
dibo: Naechste Zahl eingeben (aktuelle Zahl = 6): <6>
dibo ist ausgeschieden!
Hans: Naechste Zahl eingeben (aktuelle Zahl = 6): <7>
Hans ist ausgeschieden!
Roboter: Ermittelte Zahl = 8
Roboter: Ermittelte Zahl = 9
Roboter: Ermittelte Zahl = 10
Roboter: Ermittelte Zahl = 11
Roboter: Ermittelte Zahl = 12
Roboter: Ermittelte Zahl = 13
Roboter: Ermittelte Zahl = 15
Roboter: Ermittelte Zahl = 16
Roboter: Ermittelte Zahl = 18
```

Aufgabe 11:

Gegeben sei das folgende Java-Programm:

```
public class Einkauf {

     public static void main(String[] args) {
          Warenkorb korb = new Warenkorb();
          einkaufen(korb);
          bezahlen(korb);
     }

     static void einkaufen(Warenkorb korb) {
          korb.hineinlegen(new Marmelade());
```

```
            korb.hineinlegen(new Quark());
            korb.hineinlegen(new Butter());
            // ...
      }

      static void bezahlen(Warenkorb korb) {
            double kosten = 0.0;
            for (Ware ware : korb) {
                  kosten += ware.getPreis();
            }
            System.out.println("Zu zahlen = " + kosten + " EUR");
      }

}
```

Das Programm simuliert einen Einkauf. In einem ersten Schritt werden Waren in einen Warenkorb gelegt. An der Kasse werden anschließend die Gesamtkosten des Einkaufs ermittelt und ausgegeben.

Definieren Sie fehlende Klassen, so dass sich das Programm kompilieren lässt. Legen Sie selbst Preise für die entsprechenden Waren fest. Achten Sie darauf, dass das Programm erweiterbar gestaltet werden soll, d.h. dass neben Marmelade, Quark und Butter weitere Waren existieren und in den Warenkorb gelegt werden können und natürlich auch an der Kasse bei der Kostenermittlung berücksichtigt werden sollen.

Aufgabe 12:

Implementieren Sie Klassen, die es erlauben, beliebige Gewichtsmaße miteinander addieren zu können. Konkret sollen die Klassen so definiert werden, dass sich folgendes Programm übersetzen und mit dem angegebenen Ergebnis ausführen lassen kann:

```
public class UE32Aufgabe12 {

      public static void main(String[] args) {
            Gramm gramm = new Gramm(2);
            gramm.print(); // Ausgabe: 2.0 Gramm

            Unze unze = new Unze(4);
            unze.print(); // Ausgabe: 4.0 Unzen

            Pound pount = new Pound(6);
            pount.print(); // Ausgabe: 6.0 lb

            gramm.addiere(unze); // 2.0 Gramm + 4.0 Unzen
            gramm.print(); // Ausgabe: 115.4 Gramm

            pount.addiere(gramm); // 6.0 lb + 115.4 Gramm
            pount.print(); // Ausgabe: 6.254414779867282 lb

      }
```

```
}
```

Dabei gilt: 1 Unze = 28.35 Gramm und 1 Pound lb = 453.59 Gramm.

Achten Sie auf Erweiterbarkeit, d.h. es soll möglich sein, weitere Gewichtsmaße wie bspw. Feinunze oder Karat zu definieren und ins Hauptprogramm zu integrieren, ohne die existierenden Klassen ändern zu müssen!

Aufgabe 13:

Implementieren Sie Klassen, die es erlauben, beliebige Währungen miteinander addieren zu können, die Anfang des Jahrtausends in den Euro überführt wurden. Konkret sollen die Klassen so definiert werden, dass sich folgendes Programm übersetzen und mit dem angegebenen Ergebnis ausführen lassen kann:

```
public class Aufgabe3 {

     public static void main(String[] args) {
          Euro euro = new Euro(2);
          euro.print(); // Ausgabe: 2.0 Euro

          DM dm = new DM(4);
          dm.print(); // Ausgabe: 4.0 DM

          Lire lire = new Lire(600);
          lire.print(); // Ausgabe: 600.0 Lire

          euro.addiere(dm); // 2.0 Euro + 4.0 DM
          euro.print(); // Ausgabe: 4.045168 Euro

          lire.addiere(euro); // 600.0 Lire + 4.045168 Euro
          lire.print(); // Ausgabe: 8439.472868217055 Lire
     }

}
```

Dabei gilt: 1 DM = 0.511292 Euro und 1 (italienische) Lire = 0.000516 Euro.

Achten Sie auf Erweiterbarkeit, d.h. es soll möglich sein, weitere Währungen wie bspw. (französische) Francs oder (spanische) Peseten zu definieren und ins Hauptprogramm zu integrieren, ohne die existierenden Klassen ändern zu müssen (zu den Umrechnungsfaktoren siehe bspw. http://de.wikipedia.org/wiki/Euro).

Aufgabe 14:

Implementieren Sie in Anlehnung an die Implementierung des Nimm-Spiels in Aufgabe 6 das Bachet-Spiel derart, dass Menschen gegen Menschen oder Programme antreten können.

Das Bachel-Spiel ist ein Spiel für 2 Personen. Begonnen wird es mit einer zufällig ermittelten Zahl kleiner als 30. Die Spieler addieren abwechselnd eine selbst

gewählte ganze Zahl zwischen 1 und 10 zu dieser Zahl. Gewonnen hat der Spieler, der als erster 100 oder mehr erreicht.

Hinweis: Implementieren Sie ein Programm, dass die auf der Website http://de.wikipedia.org/wiki/Bachet%E2%80%99sches_Spiel beschriebene Gewinnstrategie implementiert.

Aufgabe 15:

Bei dieser Variante des Nim-Spiel sind n Reihen mit n Streichhölzern vorhanden. Zwei Spieler nehmen abwechselnd Streichhölzer aus einer der Reihen weg. Wie viele sie nehmen, spielt keine Rolle; es muss mindestens ein Streichholz sein und es dürfen bei einem Zug nur Streichhölzer einer einzigen Reihe genommen werden. Derjenige Spieler, der den letzten Zug macht, also die letzten Streichhölzer wegnimmt, gewinnt.

Implementieren Sie diese Variante des Nim-Spiels in Java, so dass menschliche und/oder Computer-Spieler gegeneinander antreten können.