Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 18 – Dynamisches Binden

(Stand 14.06.2017)

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.

```
public class UE18Aufgabe1 {
	// 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
		SiebenSpieler[] spieler = new SiebenSpieler[args.length];
		for (int i = 0; i < args.length; i++) {
			spieler[i] = new SiebenSpieler(args[i]);
		}
		// Spiel durchfuehren
		spielen(spieler);
	}

	// Durchfuehrung eines kompletten Spiels mit den
	// uebergebenen Spielern
	public static void spielen(SiebenSpieler[] 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(SiebenSpieler[] 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 SiebenSpieler {
      private String name;

      private boolean ausgeschieden;

      public SiebenSpieler(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;
	}
}
```

Aufgaben:

- Implementieren Sie die Methode *check*
- Leiten Sie von der Klasse *SiebenSpieler* eine Klasse *SiebenProgramm* ab, die die Methode *naechsteZahl* so überschreibt, dass Objekte der Klasse *SiebenProgramm* 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 UE18Aufgabe2 {

	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 fehlenden 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;

	public X() {
		i = 5;
		this.print();
	}

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

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

class Y extends X {
	int i = 9;

	public Y() {
		i = 8;
		this.print();
	}

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

class UE18Aufgabe3 {

	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 UE18Aufgabe4 {

	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.

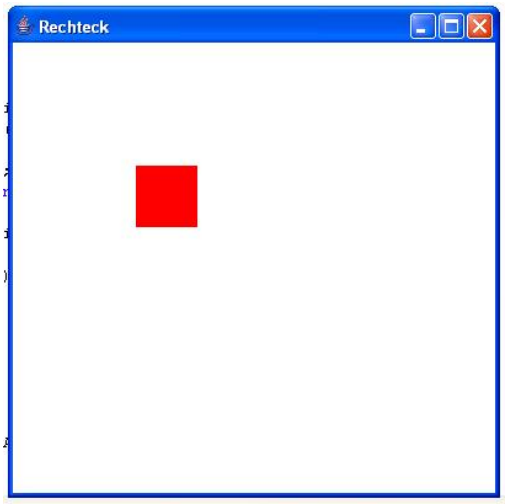
Achten Sie auf Erweiterbarkeit, d.h. es soll möglich sein, weitere Längenmaßklassen wie bspw. Landmeilen oder Yards zu definieren und ins Hauptprogramm zu integrieren, ohne die existierenden Klassen ändern zu müssen!

Aufgabe 5:

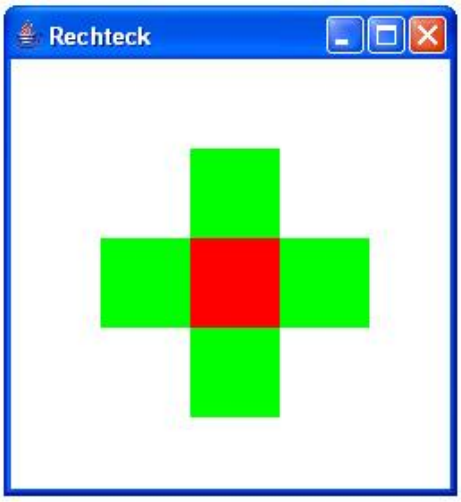
Schauen Sie sich das Programm an:

```
import java.awt.*;

class UE18Aufgabe5 {
      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, der Polymorphie und dem dynamischen Binden. 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 NimmSpieler {
     public int liefereZahl(int aktZahl) {
           return IO.readInt("1 oder 2 eingeben:");
     }
}

class UE18Aufgabe6 {
```

```
	// Hauptprogramm
	public static void main(String[] args) {
		NimmSpieler spielerA = new NimmSpieler();
		NimmSpieler spielerB = new NimmSpieler();
		NimmSpiel spiel = new NimmSpiel(spielerA, spielerB,
				IO.readInt("Startzahl: "));
		spiel.go();
	}
}

class NimmSpiel {

	// Attribute
	NimmSpieler spielerA, spielerB, aktSpieler;
	int zahl;

	public NimmSpiel(NimmSpieler spielerA, NimmSpieler 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;
	}
}

public class UE18Aufgabe7 {
	int zahl;

	ZahlenSpieler a, b;

	public UE18Aufgabe7(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 = ...
		UE18Aufgabe7 spiel = new UE18Aufgabe7(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 ZahlenSpieler 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:

```
public class UE18Aufgabe8 {

    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:

Gegeben sei das folgende Java-Programm:

```
public class UE18Aufgabe10 {

	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 11:

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 UE18Aufgabe11 {

	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 12:

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 UE18Aufgabe12 {
	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); // 6.0 Ellen + 3.216 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 13:

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 14:

Bei dieser Variante des Nimm-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 in Anlehnung an die Implementierung des Nimm-Spiels in Aufgabe 6 diese Variante des Nimm-Spiels in Java, so dass menschliche und/oder Computer-Spieler gegeneinander antreten können.

Aufgabe 15:

Gegeben Sei folgendes Java-Programm:

```
      // leeres Sparschwein erzeugen
      Sparschwein schwein = new Sparschwein();

      // beliebige Muenzen ins Sparschwein einwerfen
      schwein.einwerfen(new ZweiCentMuenze());
      schwein.einwerfen(new EinCentMuenze());
      schwein.einwerfen(new ZweiCentMuenze());
      schwein.einwerfen(new EinEuroMuenze());
      schwein.einwerfen(new ZweiCentMuenze());
   // ...

      // aktuellen Wert des Sparschweins ermitteln (in Cent)
      int wert = schwein.liefereWertInCent();
      System.out.println("Im Schwein befinden sich " + (wert / 100.0) + "
Euro");

      // alle Muenzen eines bestimmten Typs aus dem Sparschwein entfernen;
   // hier alle Zwei-Cent-Muenzen
      schwein.entfernen(ZweiCentMuenze.class);
```

```
     // alle Muenzen aus dem Sparschwein entfernen und ihren Wert (in
Cent)
  // liefern
     wert = schwein.leeren();
     System.out.println("Im Schwein befanden sich " + (wert / 100.0) + "
Euro");
```

Das Programm simuliert die Benutzung eines Sparschweins. Es nutzt dazu eine Klasse *Sparschwein*. In ein Sparschwein können Münzen eingeworfen und entnommen werden (entsprechende Münzen-Klassen) und es kann der Gesamtwert aller Münzen im Sparschwein ermittelt werden.

Definieren Sie die fehlenden Klassen, so dass sich das Programm kompilieren lässt. Ein Aufruf des obigen Programms sollte dann folgende Ausgabe erzeugen:

```
Im Schwein befinden sich 1.07 Euro
Im Schwein befanden sich 1.01 Euro
```

Wichtig: Achten Sie aber darauf, dass die Klasse *Sparschwein* erweiterbar ist, d.h. so implementiert wird, dass neben EinCent-, ZweiCent- und EinEuro-Münzen auch beliebige weitere Münzen ins Sparschwein geworfen werden können, ohne dass die Klasse *Sparschwein* selbst geändert werden muss.

Hinweise zum Implementieren der Methode *entfernen*:

- Im Paket *java.lang* gibt es eine Klasse namens *Class*. Für jede Klasse in Java existiert automatisch genau ein sogenanntes Klassenobjekt, dass eine Instanz dieser Klasse *Class* ist.
- Sei X der Name einer Klasse, dann liefert der Ausdruck *X.class* das Klassenobjekt der Klasse X.
- Die Klasse *Object* besitzt eine Methode *public Class getClass()*, die das Klassenobjekt der Klasse des Objektes liefert, für das diese Methode aufgerufen wird. Es gilt also: `(new X()).getClass() == X.class`

Aufgabe 16:

Gegeben sei folgendes Java-Programm:

```
class Player {

  private boolean isA;

  public Player(boolean isA) {
    this.isA = isA;
  }

  public int calculate(int currentNumber) {
    int number = IO.readInt("Spieler " + (this.isA ? "A" : "B")
        + "! Bitte Zahl eingeben: ");
    return number;
  }

}

class Rules {

  public boolean compareNumbers(int oldNumber, int newNumber) {
```

```
    int diff = oldNumber - newNumber;
    if (!(diff == 2 || diff == 3)) {
      return false;
    }
    return true;
  }

}

class Game {

  private Rules rules;

  public Game(Rules rules) {
    this.rules = rules;
  }

  public void play(Player playerA, Player playerB) {
    Player currentPlayer = playerA;
    int number = 20;
    System.out.println("Aktuelle Zahl: " + number);
    while (number > 0) {
      int oldNumber = number;
      number = currentPlayer.calculate(oldNumber);
      System.out.println("Aktuelle Zahl: " + number);
      if (!this.rules.compareNumbers(oldNumber, number)) {
        System.out
            .println("Ungueltige Zahl! "
                + (currentPlayer == playerA ? "Spieler A"
                    : "Spieler B") + " hat verloren!");
        return;
      }
      if (number > 0)
        if (currentPlayer == playerA) {
          currentPlayer = playerB;
        } else {
          currentPlayer = playerA;
        }
    }
    System.out.println("Gewonnen hat "
        + (currentPlayer == playerA ? "Spieler A!" : "Spieler B!"));
  }

}

class TwoThreeGame {

  public static void main(String[] args) {
    Player a = new Player(true);
    Player b = new Player(false);
    Rules rules = new Rules();
    new Game(rules).play(a, b);
  }

}
```

Das Programm `TwoThreeGame` realisiert aufbauend auf den Klassen `Player`, `Rules` und `Game` ein Zahlenspiel, bei dem zwei menschliche Spieler startend bei der 20 abwechselnd eine Zahl eingeben. Die Spielregeln schreiben vor, dass jeweils eine Zahl eingegeben werden muss, die 2 oder 3 kleiner ist als die vorangehende. Wer als erster die 0 oder eine negative Zahl erreicht, hat gewonnen.

Ihre Aufgabe ist es nun, ein ähnliches Programm bzw. Spiel `OneTwoGame` zu implementieren, das sich in zwei Eigenschaften vom obigen Spiel unterscheidet:

- Die Spielregeln sollen geändert werden. Statt einer um die Werte 2 oder 3 kleineren Zahl sollen jeweils Zahlen eingegeben werden müssen, die um 1 oder 2 kleiner sind als die vorangehende Zahl.
- Es soll ein Computer-Programm als Spieler A gegen einen menschlichen Spieler als Spieler B spielen. Dabei soll das Computer-Programm eine Strategie implementieren, so dass es immer gewinnt. Das wird dadurch erreicht, in dem das Computer-Programm immer auf die nächst niedrigere Zahl geht, die durch 3 teilbar ist.

Bei der Programmierung des neuen Spiels sollen Sie so wenig wie möglich neuen Sourcecode schreiben. Insbesondere gelten folgende Einschränkungen:

- Sie dürfen die Klassen `Player`, `Rules` und `Game` nicht ändern! Stellen Sie sich vor, Sie hätten nur den Byte-Code und nicht den Quellcode.
- Sie dürfen den Spielablauf nicht neu implementieren, sondern müssen dazu die obige Klasse `Game` mit ihrer Methode `play` nutzen!

Hinweis: Nutzen Sie die objektorientierten Konzepte Vererbung, Polymorphie und dynamisches Binden zur Realisierung des neuen Spiels!

Aufgabe 17:

Welche Ausgabe wird beim Aufruf des folgenden Programms auf die Konsole ausgegeben:

```
class A {

  int v = 1;

  void call() {
    System.out.print("A");
    System.out.print(f(f(this)).v);
  }

  A f(A obj) {
    System.out.print("B" + obj.v);
    return new A();
  }
}

class B extends A {

  B() {
    super();
    v = 2;
    System.out.print("C");
  }

  A f(A obj) {
    System.out.print("D" + obj.v);
    return new A();
  }
}
```

```
public class Polymorphie {
  public static void main(String[] args) {
    System.out.print("E");
    A obj = new B();
    System.out.print("F");
    obj.call();
    System.out.print("G");
  }
}
```

Aufgabe 18:

Welche Ausgabe wird beim Aufruf des folgenden Programms auf die Konsole ausgegeben:

```
class A {
  A() {
    System.out.print("1");
  }

  void call() {
    System.out.print("2");
    f().f();
  }

  A f() {
    System.out.print("3");
    return new B();
  }
}

class B extends A {

  A f() {
    System.out.print("4");
    return new A();
  }
}

public class Polymorphie {
  public static void main(String[] args) {
    System.out.print("5");
    A obj = new B();
    System.out.print("6");
    obj.call();
    System.out.print("7");
  }
}
```

Aufgabe 19:

Das folgende Java-Programmfragment

```
public class Gruss {

  public static void main(String[] args) {
    Person otto = new Person("Otto", "Mueller");
    Person karl = new Akademiker("Karl", "Schmidt");
    Person maria = new Akademiker("Maria", "Meier", "Dr.");
```

```
    otto.gruessen(karl);
    karl.gruessen(maria);
    maria.gruessen(otto);
  }
}
```

erzeugt die Ausgabe

```
Otto Mueller gruesst Karl Schmidt
Karl Schmidt gruesst Dr. Maria Meier
Dr. Maria Meier gruesst Otto Mueller
```

Implementieren Sie die beiden fehlenden Klassen **`Person`** und **`Akademiker`**, so dass sich die Klasse **`Gruss`** kompilieren lässt und beim Ausführen des Programms die obige Ausgabe erzeugt wird.

Die Klasse **`Person`** speichert einen im Konstruktor übergebenen Vornamen und Nachnamen. Die Klasse **`Akademiker`** speichert neben Vorname und Nachname (Konstruktor mit zwei Parametern) zusätzlich unter Umständen noch einen Titel (Konstruktor mit drei Parametern). Attribute in den beiden Klassen sollen immer als **`private`** deklariert werden. Die Bildschirmausgabe soll nicht bereits im Konstruktor sondern in der Methode **`gruessen`** erfolgen (jeweils eine Zeile).

Tipp: Definieren bzw. überschreiben Sie eine Hilfsmethode **`String getName()`**, die Sie bei der Implementierung der Methode **`gruessen`** dynamisch gebunden nutzen können. Die Methode **`getName`** soll dabei den vollständigen Namen einer Person liefern.

Aufgabe 20:

Gegeben sei folgendes Programm:

```
class A {
    int i;

    A(int v) { i = v; }

    int getI() { return i; }

    void f(A obj) { System.out.println("a" + obj.getI()); }

    void f(B obj) { System.out.println("b" + obj.getI()); }

    void call(A obj1, B obj2) {
        // todo: 5 Anweisungen einfuegen
    }
}

class B extends A {
    B(int v) { super(v); }

    void f(B obj) {
        System.out.println("c" + (obj != null ? obj.getI() : 0));
    }
    static void f() { System.out.println("ff");  }
}
```

```
public class Polymorphie {
    public static void main(String[] args) {
        A o1 = new B(1);
        A o2 = new A(2);
        B o3 = new B(3);
        o1.call(o2, o3);
    }
}
```

Fügen Sie in den Rumpf der Methode **`call`** der Klasse `A` fünf Anweisungen ein, so dass das Programm bei seiner Ausführung die folgenden fünf Konsolenausgaben in der angegebenen Reihenfolge erzeugt: „ff“, „a1“, „c3“, „a2“, „b3“. Markieren Sie die Anweisungen mit (1) bis (5) entsprechend. Berücksichtigen Sie dabei:

- Sie dürfen außerhalb des Rumpfes der Methode **`call`** keine Änderungen und Erweiterungen am gegebenen Programm vornehmen.
- Sie dürfen keine neuen Objekte erzeugen.
- Sie dürfen keinen Typecast-Operator verwenden.
- Sie dürfen keine expliziten Ausgabeanweisungen wie **`System.out.print`** verwenden. Nutzen Sie stattdessen die Möglichkeit, für die vorhandenen Objekte deren Methoden aufzurufen, wie bspw. **`obj1.f(obj2);`**