Axel Tetzlaff
8. Februar 2016

fhwedel
UNIVERSITY OF APPLIED SCIENCES

---

Aufgaben zur Klausur Objektorientierte Programmierung im WS 2015/16

Zeit: 90 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.
Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 9 Seiten.

Die Klasse `Klausur` enthält ein Hauptprogramm, das eine Datenstruktur zur Definition von Noten nutzt um das Lied 'Alle meine Entchen' abzuspielen.

```
class Klausur {
  public static void main(String[] args) throws InterruptedException {
    //                    Tonhöhe    Tonlänge
    Playable c4 = new Note(Pitch.C, Duration.QUARTER);
    Playable d4 = new Note(Pitch.D, Duration.QUARTER);
    Playable e4 = new Note(Pitch.E, Duration.QUARTER);
    Playable f4 = new Note(Pitch.F, Duration.QUARTER);
    Playable g2 = new Note(Pitch.G, Duration.HALF);
    Playable g1 = new Note(Pitch.G, Duration.WHOLE);
    Playable a4 = new Note(Pitch.A, Duration.QUARTER);
    Playable e2 = new Note(Pitch.E, Duration.HALF);
    Playable c1 = new Note(Pitch.C, Duration.WHOLE);

    // 'Alle meine Entchen'
    Playable beginning =
      Sequence.create(new Playable[]{c4,d4,e4,f4,g2,g2});

    // 'schwimmen auf dem See'
    Playable middle =
      Sequence.create(new Playable[]{a4,a4,a4,a4,g1});

    // 'Köpfchen in das Wasser, Schwänzchen in die Höh'
    Playable ending =
      Sequence.create(
          new Playable[]{f4,f4,f4,f4,e2,e2,d4,d4,d4,d4,c1}
      );

    Playable entchenSong = Sequence.create(new Playable[]{
      beginning, new Repeat(middle, 2), ending
    });

    Instrument i = new MidiInstrument();
    entchenSong.play(i);
  }
}
```

Um ein Lied abzuspielen, verwenden wir ein Objekt mit der Schnittstelle `Instrument`

```
public interface Instrument {
  // z.B. beim Klavier: 'Taste nach unten gedrückt halten'
  void noteOn(Pitch pitch, int octave);

  // z.B. bei Klavier: 'Taste wieder loslassen'
  void noteOff(Pitch pitch, int octave);
}
```

Für alle 'Dinge', die man auf einem Instrument abspielen kann, definieren wir eine allgemeine Schnittstelle `Playable`

```
interface Playable {
  public void play(Instrument instrument) throws InterruptedException ;
}
```

Die Klasse `Pitch` definiert die Töne der Tonleiter

```
public final class Pitch {
```

```
  public static final Pitch C  = new Pitch("C",  0);
  public static final Pitch C_ = new Pitch("C#", 1);
  public static final Pitch D  = new Pitch("D",  2);
  public static final Pitch D_ = new Pitch("D#", 3);
  public static final Pitch E  = new Pitch("E",  4);
  public static final Pitch F  = new Pitch("F",  5);
  public static final Pitch F_ = new Pitch("F#", 6);
  public static final Pitch G  = new Pitch("G",  7);
  public static final Pitch G_ = new Pitch("G#", 8);
  public static final Pitch A  = new Pitch("A",  9);
  public static final Pitch B  = new Pitch("B",  10);
  public static final Pitch B_ = new Pitch("Bb", 11);

  public final int pitchNumber;
  public final String name;

  private Pitch(String name, int pitchNumber) {
    this.name = name;
    this.pitchNumber = pitchNumber;
  }

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

# 1 Aufgabe

Die Klasse `Duration` definiert die unterschiedlichen Notendauern. Eine **ganze** Note, ist eine langer Ton. Eine **Viertel** ist ein Ton mit enstprechend reduzierter Länge.

```
public final class Duration {
  public static final Duration WHOLE     = new Duration(1);
  public static final Duration HALF      = new Duration(2);
  public static final Duration QUARTER   = new Duration(4);
  public static final Duration EIGTH     = new Duration(8);
  public static final Duration SIXTEENTH = new Duration(16);

  private final int denominator;

  private Duration(int denominator) {
    this.denominator = denominator;
  }

  @Override
  public String toString() {
    return "1/" + this.denominator;
  }

  public int inMillis() {
    return 1500 / denominator;
  }
}
```

### 1.1 Aufgabe

Der Konstruktor der Klasse `Duration` ist als **private** deklariert. Welche Konsequenzen hat dies und warum ist das in diesem Fall sinnvoll?

### 1.2 Aufgabe

Die Klasse selbst ist als **final** deklariert. Was wird damit verhindert und warum ist das für diese Klasse sinnvoll?

## 2 Aufgabe

Die Klasse `Note` definiert einen Typen um konkrete Noten darzustellen. Eine Note besteht aus Tonhöhe (`pitch`, `octave`) und Länge (`duration`). Die 'Oktave' beschreibt in welcher 'Lage' ein Ton gespielt wird, also z.B. ob es sich um ein 'hohes' oder 'tiefes' C handelt.

```
1  class Note implements Playable {
2    private final Pitch pitch;
3    private final int octave;
4    private final Duration duration;
5
6    public Note(Pitch pitch, int octave, Duration duration) {
7      this.pitch = pitch;
8      this.octave = octave;
9      this.duration = duration;
10   }
11
12
13     ..........................................................
14
15     ..........................................................
16
```

```
17      ....................................................
18
19
20    @Override
21    public String toString() {
22      return "O" + this.octave + "-" + this.pitch + "-" + this.duration;
23    }
24
25    public void play(Instrument instrument) throws InterruptedException {
26      instrument.noteOn(this.pitch, this.octave);
27
28      // durch den Aufruf von sleep werden die folgenden Aufrufe
29      // um die übergebene Anzahl Millisekunden verzögert
30      Thread.sleep(duration.inMillis());
31
32      instrument.noteOff(this.pitch, this.octave);
33    }
34  }
```

### 2.1 Aufgabe

Im Hauptprogramm werden die Noten-Objekte wie folgt instanziiert:

```
1     Playable c8 = new Note(Pitch.C, Duration.QUARTER);
```

Ergänzen Sie den fehlenden Konstruktor im Code der `Note` Klasse so, dass Noten ohne Angabe der Oktave mit dem `octave` Wert 0 erzeugt werden. Vermeiden Sie Verdoppelung von existierendem Code.

## 3 Aufgabe

Um aufeinander folgende Töne zu Melodien kombinieren zu können definieren wir ein Klasse `Sequence` wie folgt.

```
1  class Sequence implements Playable {
2
3    private static final Playable EMPTY = new Playable() {
4
5        ....................................
6
7        ....................................
8
9        ....................................
10     }
11   };
12
13   private final Playable current;
14   private final Playable next;
15
16   private Sequence(Playable current, Playable next) {
17     this.current = current;
18     this.next = next;
19   }
20
```

```
21   public static Playable create(Playable[] playables) {
22     Playable head = EMPTY;
23     for (int i = playables.length; i > 0; --i) {
24       Playable current = playables[i-1];
25       head = new Sequence(current, head);
26     }
27     return head;
28   }
29
30   @Override
31   public void play(Instrument instrument) throws InterruptedException  {
32
33       ....................................
34
35       ....................................
36   }
37
38 }
```

### 3.1 Aufgabe

Implementieren sie die fehlenden Elemente in der Klasse `Sequence` an den markierten Stellen so, dass wie im Hauptprogramm dargestellte Melodien in der richtigen Reihenfolge abgespielt werden und das Abspielen am Ende der Melodie terminiert.

### 3.2 Aufgabe

Die Variable `Playable EMPTY` wurde als `private static final` deklariert. Welche Auswirkungen hat dies und warum ist das in diesem Fall sinnvoll?

## 4 Aufgabe

Der Mitteilteil in 'Alle meine Entchen' wird einmal wiederholt (*'Schwimmen auf dem See, Schwimmen auf dem See'*). Um uns bei der Definition der Melodie nicht unnötig wiederholen zu müssen, verwenden wir eine Klasse `Repeat` mit der sich Teile wiederholt abspielen lassen. Implementieren Sie diese Klasse, so dass der Aufruf im Hauptprogramm kompiliert und funktioniert.

```
class Repeat ....................................
```

## 5 Aufgabe

Gegeben sei das folgende, zweite Hauptprogramm:

```
1 import java.util.List;
2 import java.util.ArrayList;
3 class Klausur2 {
4   public static void main(String[] args) throws InterruptedException {
5     Pitch c = Pitch.C;
6     Pitch cc = Pitch.C;
7     Playable ccc = Pitch.C;
8
9     Note c1  = new Note(c, Duration.QUARTER);
10    Note cc1 = new Note(cc, Duration.QUARTER);
11    Note c4  = new Note(cc, Duration.QUARTER);
12    Playable c2  = new Note(cc, Duration.QUARTER);
13
14    Playable p1 = new Note(Pitch.C, Duration.QUARTER);
15    Playable p2 = c1;
16
```

```
17      Playable[] array = {c1, c4, c1};
18
19      Sequence melody1 = Sequence.create(array);
20      Playable melody2 = Sequence.create(array);
21      Playable melody3 = Sequence.create(array);
22      Playable melody4 = melody3;
23
24      List<Object> os = new ArrayList<Pitch>();
25      Object o = new ArrayList<Pitch>();
26      List<Playable> list1 = new ArrayList<Note>();
27      List<? extends Playable> list2 = new ArrayList<Note>();
28      List<? extends Playable> list3 = new ArrayList<Pitch>();
29      List<?> list4 = new ArrayList<Pitch>();
30      List<? extends Sequence> list5 = melody3;
31
32      boolean b;
33    }
34  }
```

## 5.1 Aufgabe

Mindestens eine der vorhergegangenen Deklarationen wird vom Compiler abgelehnt. Nennen Sie die Zeilennummern und den Grund warum der Compiler die Zeilen ablehnt.

| Zeilenr. | Begründung |
|---|---|
| ..... | ..... |
| ..... | ..... |
| ..... | ..... |
| ..... | ..... |
| ..... | ..... |
| ..... | ..... |

### 5.2 Aufgabe

Im folgenden werden Zeilen gezeigt, die auf die am Ende des Hauptprogramms boolsche Ausdrücke auf die dort deklarierte Variable `b` zu.

Kreuzen Sie an welche dieser Ausdrücke zu `true` und welche zu `false` ausgewertet beziehungsweise vom Compiler abgelehnt werden.

| Ausdruck | true | false | Compilerfehler |
|---|---|---|---|
| `b = c == cc;` | [_] | [_] | [_] |
| `b = cc.equals(cc);` | [_] | [_] | [_] |
| `b = cc1.equals(cc);` | [_] | [_] | [_] |
| `b = cc == new Pitch("C", 0);` | [_] | [_] | [_] |
| `b = p1 == p2;` | [_] | [_] | [_] |
| `b = melody4 == melody2;` | [_] | [_] | [_] |
| `b = melody4.equals(melody2);` | [_] | [_] | [_] |
| `b = melody4.equals(melody3);` | [_] | [_] | [_] |

## 6 Aufgabe

Die Objekte der Klasse `Pitch` werden intern lediglich durch einen `Integer` und den Namensstring repräsentiert. Ist es sinnvoll dafür die eigene Klasse `Pitch` zu Deklarieren anstatt einfach direkt mit einem `Integer` zu arbeiten.

[_] Ja [_] Nein

Begründung:

---

---

Die Schnittstelle `Playable` ist als Java–Interface deklariert. Ist diese Deklaration von Vorteil gegenüber einer Deklaration als abstrakter Klasse?

[_] Ja [_] Nein

Begründung:

---

---