Karlsruher Institut für Technologie

Programmieren WS 2012/2013

Forschungsgruppe: Verifikation trifft Algorithmik
http://verialg.iti.uka.de/prog_ws12.php
Dr. Carsten Sinz, Florian Merz, Markus Iser

Übungsblatt 5
(v1.0)

Ausgabe:	17.12.2012
Abgabe:	14.01.2013, 13Uhr
Besprechung:	21.01.2013 - 25.01.2013

Hinweise

Beachten Sie bei Ihrer Abgabe inbesondere auf das Einhalten ...

- der maximale Zeilenlänge von 120 Zeichen,
- der Kompilierbarkeit,
- der Whitespace-Regeln[1],
- der JavaDoc-Regeln[2] und

Beachten Sie auch, dass die Einhaltung der Regeln vom Praktomaten automatisiert geprüft wird. Besteht eine Abgabe eine dieser Prüfungen nicht, so wird sie hiedurch automatisch ungültig und nicht in eine Bewertung einbezogen. Fangen Sie daher frühzeitig mit dem Lösen Ihrer Aufgaben an und testen Sie die Abgabe im Praktomaten.

Neben den spezifisch für die Aufgaben gestellten Anforderungen gelten für dieses Blatt die folgenden Erwartungen, deren Nichtbeachtung zu Punktabzügen führt.

- Klassen, Methoden und Attribute erhalten sinnvoll gewählte Sichtbarkeiten, Namen und Argumente.
- Im normalen Programmlauf soll Ihre Abgabe keine zusätzlichen Ausgaben, beispielswiese zum Debugging, produzieren. Dies ist für dieses Blatt besonders wichtig, da der Praktomat die Ein- und Ausgabe für Tests benutzt.
- Die mit diesem Blatt neu eingeführten Checkstyle Regeln sollten eingehalten werden.

Implementieren Sie Ihre Lösung nicht im Paket `programmieren`, dieses ist reserviert für Dateien die von uns bereitgestellt wurden. Modifizieren Sie die bereitsgestellten Dateien aus dem Paket `programmieren` nicht.

Denken Sie daran Ihre Lösung selbständig zu testen, um unnötige Punktabzüge für funktionale Fehler zu vermeiden. Klären Sie offene Fragen zur Aufgabenstellung im Forum des Praktomaten.

[1] `http://baldur.iti.uka.de/programmieren/whitespace-checks.xml`
[2] `http://baldur.iti.uka.de/programmieren/javadoc-checks.xml`

A Vererbung und Interfaces

Fangen wir zunächst mit einigen Fragen zum Thema Vererbung an:

A.1 Fragen

Geben Sie Ihre Antworten auf die Fragen in einer `.txt` Datei im dafür vorgesehenen Task des Praktomaten ab. Geben Sie keine Dateien anderer Aufgabenteile in diesem Task ab.

Frage A.1.1 Erklären Sie mit Ihren eigenen Worten den Unterschied zwischen *Überschreiben* und *Überladen* von Methoden.

Frage A.1.2 Welche funktionalen Probleme verursacht die folgende Implementierung einer `equals`-Methode? Begründen Sie Ihre Antwort und zeigen Sie mit einem Beispiel auf wann Probleme auftreten.

```
public class Something {
    private int attribute;

    public boolean equals(Something other) {
        if (other == null) { return false; }
        if (other == this) { return true; }
        if (other.attribute == this.attribute) { return true; }
        return false;
    }
}
```

Für die folgenden Fragen seien diese beiden Klassen als Grundlage gegeben:

```
class GeneralType {
    String attribute = "general foobar";

    public void printAttribute() {
        System.out.println(attribute);
    }
}

class SpecificType extends GeneralType {
    String attribute = "specific foobar";

    @Override
    public void printAttribute() {
        System.out.println(attribute);
    }
}
```

Frage A.1.3 Wie lautet die Ausgabe von folgendem Programm und warum?

```
public class InheritanceTest {
    public static void generalPrintAttribute(GeneralType thing) {
        thing.printAttribute();
    }

    public static void main(String[] args) {
        SpecificType thing = new SpecificType();
        generalPrintAttribute(thing);
    }
}
```

Frage A.1.4 Warum lässt sich das folgende Programm nicht übersetzen?

```
public class InheritanceTest {
    public static void specificPrintAttribute(SpecificType thing) {
        thing.printAttribute();
    }

    public static void main(String[] args) {
        GeneralType thing = new GeneralType();
        specificPrintAttribute(thing);
    }
}
```

Frage A.1.5 Wie lautet die Ausgabe des folgenden Programms? Begründen Sie Ihre Antwort.

```
public class InheritanceTest {

    public static void generalPrintAttribute1(GeneralType thing) {
        thing.printAttribute();
    }

    public static void generalPrintAttribute2(GeneralType thing) {
        System.out.println(thing.attribute);
    }

    public static void specificPrintAttribute1(SpecificType thing) {
        thing.printAttribute();
    }

    public static void specificPrintAttribute2(SpecificType thing) {
        System.out.println(thing.attribute);
    }

    public static void main(String[] args) {
        SpecificType thing = new SpecificType();
        generalPrintAttribute1(thing);
        generalPrintAttribute2(thing);
        specificPrintAttribute1(thing);
        specificPrintAttribute2(thing);
    }
}
```

A.2 Medienbibliothek

Sie sollen in dieser Teilaufgabe zunächst den Kern einer Medienbibliothek entwerfen. Die Medienbibliothek soll dabei unterschiedliche Medientypen (Audio und Video) sowie unterschiedliche Arten von Urhebern unterstützen. Verwenden Sie in Ihrer Implementierung die Konzepte Vererbung und Interfaces, die Sie in der Vorlesung kennengelernt haben.

Beschränken Sie sich bei Ihrer Implementierung auf das nötigste zum Lösen der Aufgaben, achten Sie jedoch immer auf eine saubere Ausführung. Die Verwendung von Klassen und Interfaces aus `java.util` und `java.lang` ist für diese Aufgabe ausdrücklich erlaubt. Verwenden Sie beispielswiese die Klasse `java.util.ArrayList` als Container-Klasse. Die Verwendung andere Pakete ist nicht erlaubt.

Sie sind für diese Aufgabe frei in Ihrer Wahl der Klassen-, Methoden und Attributsbezeichnungen. Auch die Struktur des Programms ist Ihnen nicht vorgegeben. Dadurch sind Praktomat-JUnit-Tests für diese Aufgabe natürlich nicht möglich. Der Praktomat wird daher für seine Tests die Kommandozeilenschnittstelle aus dem zweiten Aufgabenteil benutzen. Achten Sie daher besonders darauf keine Ausgaben zu produzieren die nicht in der Aufgabenstellung gefordert wurden.

Fügen Sie allen Klassen die Sie schreiben eine `toString`-Methode hinzu, die eine textuelle Darstellung der Klasse als Zeichenkette zurückgibt. Überschreiben Sie außerdem in jeder Klasse `clone()`, `equals()` und `hashCode()` enstsprechend der Java-Konventionen.

A.2.1 Urheber

Die Medienbibliothek soll drei Arten von Urheber unterstützen: Einfache Personen, Künstler[3] und Bands.

Definition 1 (Person)**.** Eine **Person** *hat einen* Vornamen und einen Nachnamen. Eine einfache **Person** wird textuell dargestellt als "`<vorname>␣<nachname>`"[4].

Beispiel Eine **Person** mit Vornamen Larry und Nachnamen Mullen würde als "`Larry Mullen`" dargestellt.

Definition 2 (Künstler)**.** Ein **Künstler** *ist eine* **Person** mit einem zusätzlichen Künstlernamen. Ein **Künstler** wird textuell dargestellt als "`<vorname>␣"<künstlername>"␣<nachname>`"[5].

Beispiel U2s Frontmann Bono der mit Vornamen Paul David und mit Nachnamen Hewson heißt würde dargestellt als "`Paul David "Bono" Hewson`".

Definition 3 (Band)**.** Eine **Band** *hat einen* Bandnamen und *hat mehrere* Mitglieder. Nur **Person**en können Mitglieder von **Band**s sein. Eine **Band** wird textuell dargestellt als "`bandname␣(<member1>,␣<member2>,␣...)`". Wobei `<memberX>` für die textuelle Darstellung des Mitglieds steht.

Beispiel Die **Band** U2 würde dargestellt als "`U2 (Paul David "Bono" Hewson, David Howell "The Edge" Evans, Larry Mullen, Adam Clayton)`".

[3] Wie jeder weiß ist mann nur dann ein Künstler, wenn man auch einen Künstlernamen hat!

[4] Beachten Sie, dass das Zeichen ␣ für ein normales Leerzeichen steht und die Anführungszeichen (" und ") nicht Teil der textuellen Darstellung sind.

[5] Beachten Sie, dass hier die *inneren* Anführungszeichen durchaus Teil der textuellen Darstellung sind.

Definition 4 (Urheber)**.** Sowohl **Person**en (und damit auch **Künstler**) also auch **Band**s können **Urheber** von **Mediendatei**en sein.

Sie können das **Urheber**-Konzept wahlweise als abstrake Klasse oder als Interface umsetzen. Im Rahmen dieser Aufgabenstellung empfehlen wir die Verwendung einer abstrakten Klasse, da dies später bei der Implementierung der `match`-Methode Arbeit erspart.

A.2.2 Mediendateien

Ihre Medienbibliothek soll zwei Arten von Mediendateien verwalten können: Audio und Video.

Definition 5 (Mediendatei)**.** Eine **Mediendatei** hat einen URI (uniform ressource identifier) und einen Urheber. Eine Mediendatei ist immer entweder eine **Audio**- oder eine **Video**-Datei.

Kodieren Sie den URI als Zeichenkette und den Urheber als Referenz auf einen solchen.

Definition 6 (Audiodatei)**.** Eine **Audiodatei** *ist eine* **Mediendatei** mit einer zusätzlichen ganzzahligen Länge in Sekunden. Die textuelle Repräsentation einer Audiodatei ist "`<uri>␣by␣<urheber>,␣(<länge>s)`".

Definition 7 (Videodatei)**.** Eine **Videodatei** *ist eine* **Mediendatei** mit den zusätzlichen ganzzahligen Attributen Breite und Höhe. Die textuelle Repräsentation einer Videodatei ist "`<uri>␣by␣<urheber>,␣(<width>x<height>)`".

A.3 Suche

Nun sollen Sie das System so erweitern, dass der Benutzer im System nach Objekten suchen kann. Er soll dabei mit ein- und derselben Funktion sowohl nach Urhebern als auch nach Mediendateien suchen können. Hierfür ist es notwendig die Objekte mit einer einheitlichen Schnittstelle ansprechen zu können.

Realisieren Sie diese Schnittstelle über ein Interface `Matchable`, welches die Methode `boolean match(String str)` enthält. Das Argument `str` der Methode `match` ist dabei der String nach dem der Benutzer sucht und die Methode gibt `true` zurück, wenn das Objekt "genau genug" mit der Sucheingabe übereinstimt und `false` andernfalls. Wann genau ein Objekt "genau genug" mit der Sucheingabe übereinstimmt kann hierdurch in jedem Objekt anders berechnet werden. Dennoch gilt für alle Objekt, dass es dann "genau genug" mit einer Sucheingabe übereinstimmt, wenn mindestens eines seiner Attribute mit der Anfrage "genau genug" übereinstimmt.

Beispiel Der **Künstler** "Paul David "Bono" Hewson" soll sowohl bei der Suche nach "Paul David", "Bono", als auch "Hewson" gefunden werden. Gleichzeitig soll jedoch auch die **Band** U2, in der Bono Mitglied ist, bei diesen Suchbegriffen gefunden werden. Um die Implementierung für diese Aufgabe einfach zu halten soll Bono bei der Suche nach "Paul" (ein Substring des Vornamens) oder "Paul David Hewson" (der Kombination aus Vorname und Nachname) nicht gefunden wird, da dies nur Substrings von Attributen sind.

Damit die Suchfunktion toleranter auf Tippfehler in der Eingabe des Benutzers reagiert, sollen Sie die Eingabe nicht auf Gleichheit mit den Attributen des Objekts vergleichen (mittels `equals`), sondern die Editierdistanz nach Levenshtein, die Sie im dritten Übungsblatt bereits kennengelernt haben, benutzen[6]. Sie finden auf `http://baldur.iti.kit.edu/programmieren` eine Klasse, die Ihnen

[6] Wir meinen hier natürlich die reguläre Levenshtein-Distanz, nicht die h-Variante.

die Levenshtein-Distanz zwischen zwei Zeichenketten berechnet. Implementieren Sie auf dieser Basis die Methode `getNormalizedLevenshteinDistance()`, die die normalisierte Levenshtein-Distanz zurückliefert. Die normalisierte Levenshtein-Distanz lev' liefert im Gegensatz zur Levenshtein-Distanz lev immer Werte zwischen 0 und 1, indem die reguläre Levenshtein-Distanz durch die Länge des längeren der beiden Wörter geteilt wird.

$$\text{lev}'(p, a_i) = \frac{\text{lev}(p, a_i)}{\max(\text{len}(p), \text{len}(a_i))}$$

Definition 8 (Matchable.match)**.** Für eine Suchanfrage mit dem `String` `str` soll die Methode `match`

- `true` zurückgegeben, wenn die normalisierte Levenshtein-Distanz zwischen `str` und einem String Attribut des Objekts kleiner als 0.25 ist *(direct match)*,
- `true` zurückggeben, wenn die `match`-Methode eines der Attribute[7] die das Interface `Matchable` implementieren `true` zurückgibt *(indirect match)* und
- `false` in allen anderen Fällen zurückgeben.

B Kommandozeilenbasierte Benutzerschnittstelle

In dieser Teilaufgabe sollen Sie nun eine sehr einfache Datenbank und eine Kommandozeilenschnittstelle für Ihre Medienbibliothek implementieren.

B.1 Datenbank

Implementieren Sie nun eine "Datenbank"-Klasse, die in erster Linie dazu dient Referenzen auf alle bisher vom Benutzer ins System eingegeben **Urheber** und **Mediendatei**en zu speichern. Sie soll die Möglichkeit bieten neue **Urheber** und **Mediendateien** hinzuzufügen und darüber hinaus die Möglichkeit bieten nach Objekten in der Datenbank zu suchen. Hier ist es sinnvoll drei (sehr ähnliche) Suchfunktionen zu implementieren:

(a) eine Methode die eine Liste aller **Urheber** zurückgibt, die auf eine Sucheingabe `matchen` und

(b) eine Methode die eine Liste aller **Person**en zurückgibt, die auf eine Sucheingabe `matchen`.

(c) Eine Methode die eine Liste aller Objekte zurückgibt, die auf eine Sucheingabe `matchen` passen,

B.2 Kommandozeile

Benutzen Sie die auf `http://baldur.iti.kit.edu/programmieren` bereitgestellte Klasse `MyTerminal.java` und deren Methoden `askString` und `askInt` um Eingaben von der Kommandzeile einzulesen.

Um Ihre Abgabe automatisiert testen zu können ist es wichtig, dass Ihre Abgabe genau den Vorgaben entspricht. Insbesondere ist es wichtig, dass keine zusätzlichen Ausgaben angezeigt werden, dass Sie sich genau an die hier gezeigte Schreibweise halten und dass Sie die Reihenfolge des Programmablaufs genau einhalten.

[7] Auch dann, wenn es indirekt in einer Liste steckt.

Bei Programmstart wird der Benutzer mit folgendem Prompt aufgefordert einen Befehl einzugeben:
`Enter␣command␣(add|search|quit)>␣`

Um ein Objekt in die Datenbank hinzuzufügen kann er `add` eingeben, um ein Objekt zu suchen kann er `search` eingeben und um das Programm zu beenden kann er `quit` eingeben. Gibt er einen anderen Befel ein, so soll `Invalid␣input` ausgegeben werden. Nach erfolgreichem Abschluss einer Eingabe sowie nach dem Auftreten von Eingabefehlern soll das Programm immer an dieser Stelle fortgesetzt werden.

Tippt der Benutzer `add` ein, so wird er aufgefordert den Typ des hinzuzufügenden Objekts anzugeben:
`Enter␣entry␣type␣(person|artist|band|audio|video)>␣`

Je nachdem ob ein **Urheber** oder eine **Mediendatei** hinzugefügt werden soll verhält sich das Programm unterschiedlich. Wird keine der akzeptierten Befehle eingegeben, so soll `Invalid␣input` ausgegeben werden.

B.2.1 Hinzfügen von Urhebern

Soll eine **Urheber** hinzugefügt werden (`person`, `artist`, `band`), so müssen Sie nun aus folgender Liste die passenden Abfragen durchführen:
`Enter␣given␣name>␣`
`Enter␣family␣name>␣`
`Enter␣pseudonym>␣`
`Enter␣band␣name>␣`
`Enter␣member␣or␣'quit'>␣`

Beispiel Ablaufbeispiel für das Hinzufügen von **Künstler** “Vorname "Pseudonym" Nachname”:
`Enter␣command␣(add|search|quit)>␣add`
`Enter␣entry␣type␣(person|artist|band|audio|video)>␣artist`
`Enter␣given␣name>␣Vorname`
`Enter␣family␣name>␣Nachname`
`Enter␣pseudonym>␣Pseudonym`
`Enter␣command␣(add|search|quit)>␣`

Eine Besonderheit stellt dabei die Eingabe von Bandmitgliedern dar. Hierbei wird der Benutzer solange nach neuen Mitgliedern gefragt, bis der Benutzer `quit` eintippt. Die Eingabe des Benutzers wird hierbei als Suchanfrage nach einer **Person** die Datenbank weitergereicht. Diese gibt eine Liste von **Person**en zurück, die auf die Suchanfrage `matchen`. Besteht diese Liste aus genau einer Person wird diese **Person** der **Band** als Mitglied hinzgefügt. Andernfalls wird `No unique match found` ausgegeben, kein Mitglieder der **Band** hinzugefügt und mit der Eingabe von weiteren Bandmitgliedern fortgefahren.

B.2.2 Hinzufügen von Mediendateien

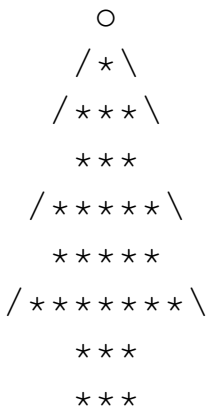
Soll anstelle eines **Urheber**s eine **Mediendatei** (`audio` oder `video`) hinzugefügt werden, so sind aus folgender Liste die jeweils passenden Abfragen durchzuführen:
`Enter␣URI>␣`
`Enter␣creator>␣`
`Enter␣duration␣(as␣integer)>␣`
`Enter␣width␣(as␣integer)>␣`
`Enter␣height␣(as␣integer)>␣`

Wie die Eingabe der Bandmitglieder ist auch die Eingabe des **Urheber**s über eine Suche in der Datenbank realisiert. Enthält die von der Datenbank zurückgegeben Liste an **Urheber**n genau ein Element, so wird dieses Element als **Urheber** der **Mediendatei** gesetzt. Andernfalls wird `No␣unique␣match␣found` ausgegeben und das Hinzufügen eines Mediums abgebrochen.

B.2.3 Suchen von Einträgen

Hat der Benutzer auf die Aufforderung `Enter␣command␣(add|search|quit)>␣` den Befehl `search` eingegeben, so wird er aufgefordert einen Suchbegriff einzugeben:
`Enter␣search␣term>␣`

Diesmal erfolgt die Suche in der kompletten Datenbank, also nach **Urheber**n und **Mediendatei**en. Benutzen Sie `Collections.sort` aus dem Paket `java.util.Collections` um die textuellen Darstellungen der Treffer als alphabetisch sortierte `Strings` anzuzeigen. Wurde kein einziger Treffer gefunden, so soll `No␣matches␣found` ausgegeben werden. Nach erfolgter Suche soll das Programm wieder zur ersten Eingabeaufforderung zurückkehren.



Wir wünschen Ihnen allen ein frohes Fest,
und einen guten Rutsch ins neue Jahr!