Christian Bunse & Jöran Pieper
E-Mail: Joeran.Pieper@fh-stralsund.de
Büro: Haus 4/306c/331

Aufgaben zur Lehrveranstaltung
Laborpraktikum Software
SMSB3300, SMIB3300

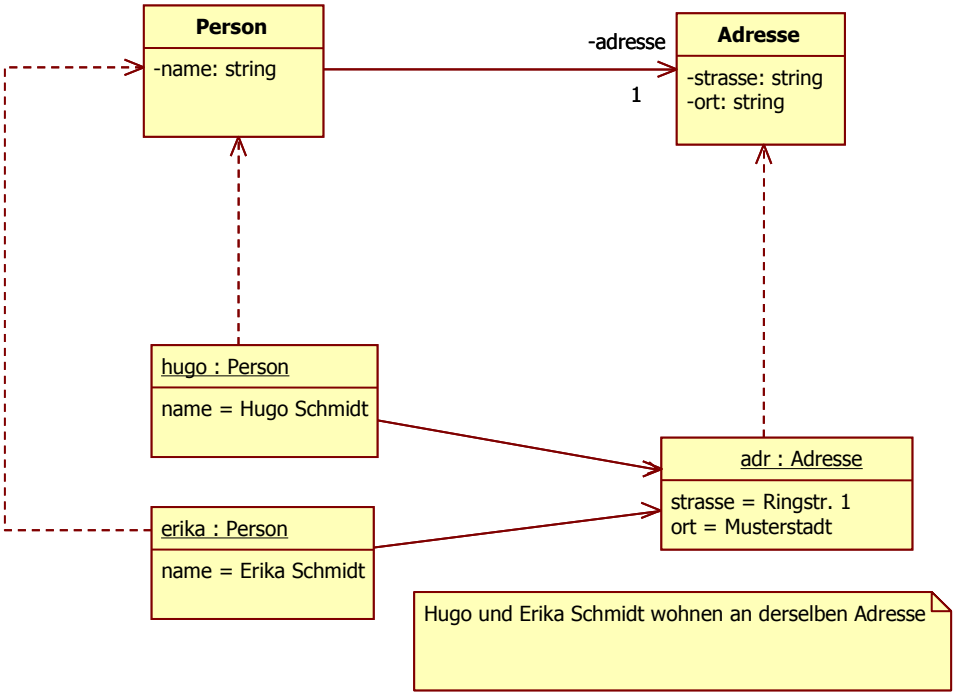
C. Aufgabenblatt

- *C.1 Präsenzaufgabe (Serialisierung mit dem BinaryFormatter)*
- *C.2 Präsenzaufgabe (`ArrayList`, `Enum`, `Exception`)*
- *C.3 Präsenzaufgabe (`IEnumerable`, Datenkapselung)*
- *C.4 Hausaufgabe (`ArrayList`, `IEnumerator`)*
- *C.5 Hausaufgabe (Sortieren, `IComparable`)*
- *C.6 Hausaufgabe (Interfaces, Polymorphie, Serialisierung, File-I/O, Exceptions)*
- *C.7 Hausaufgabe (Generics)*

Die Präsenzaufgaben dieses Aufgabenblattes werden im Labor ab dem 23.10.2018 bearbeitet. Die Hausaufgaben dieses Aufgabenblattes sind **bis Montag, 12.11.2018, 8:00 Uhr abzugeben (siehe Veranstaltungsplan)**. Verspätete Abgaben werden nicht berücksichtigt.

C. Aufgabenblatt

C.1 Präsenzaufgabe (Serialisierung mit dem ByteArrayOutputStream)



Schauen Sie sich die Vorlesungsfolien zur Serialisierung (Kapitel 8.2) an. Insbesondere das Beispiel mit Hugo und Erika Schmidt. Schreiben Sie ein Programm, in dem sie eine `ArrayList` erzeugen und in diese mit Hugo und Erika Schmidt zusammen mit ihrer Adresse einfügen. Anschließend soll die Liste unter Nutzung des `ObjectOutputStream` serialisert anschließend analog deserialisiert werden. Überlegen Sie sich anschließend, wie Sie nun testen können, ob die Adresse tatsächlich nur einmal deserialisiert wurde. Implementieren Sie diesen Test.

Damit es schneller geht, hier ein teilfertiges Programm:

```
public class Adresse implements Serializable
{
	private String strasse;
	private String ort;

	public String getStrasse() { return this.strasse; }
	public void setStrasse(String strasse) { this.strasse = strasse; }

	public String getOrt() { return ort; }
	public void setOrt(String ort) { this.ort = ort; }

	public String toString()
	{
		return new StringBuilder().append(strasse).append(", ")
			.append(ort).toString();
	}
}

public class Person implements Serializable
{
	private String name;
	private Adresse adresse;

	public String getName() { return name; }
	public void setName(String name) { this.name = name; }

	public Adresse getAdresse() { return adresse; }
	public void setAdresse(Adresse adresse) { this.adresse = adresse; }

	public String toString()
	{
		return new StringBuilder()
			.append(name).append(", ").append(adresse.toString())
			.toString();
	}
}

public class Program
{
	public static void main(String[] args)
	{
		Adresse adresse = new Adresse();
		adresse.setStrasse("Ringstr. 1");
		adresse.setOrt("Musterstadt");

		Person hugo = new Person();
		hugo.setName("Hugo Schmidt");
		hugo.setAdresse(adresse);

		Person erika = new Person();
		erika.setName("Erika Schmidt");
		erika.setAdresse(adresse);

		// Hier fehlt Ihr Code
	}
}
```

C.2 Präsenzaufgabe (`ArrayList`, `Enum`, `Exception`)

Implementieren Sie eine Klasse `Familie` (s. Beispielcode unten). Diese Klasse soll die Namen aller Familienmitglieder in einem Feld vom Typ `ArrayList` enthalten. Die Klasse `Familie` hat keine weiteren Felder! Die `ArrayList` ist geordnet: zuerst kommt die Mutter, dann der Vater, dann die Kinder. Implementieren Sie den Konstruktor (zwei Parameter für Vater und Mutter) und eine Methode `addKind` zum Hinzufügen von Kindern zur Familie. Der Konstruktor sowie die Hinzufüge-Methode sollen eine Exception vom Typ `ArgumentException` erzeugen, wenn ein „leerer" Name übergeben wird. Dies kann entweder über eine „eigene" Exception oder die Java „IllegalArgumentException" geschehen. Stellen sie eine sinnvolle und hilfreiche Fehlerausgabe sicher.

Implementieren Sie schließlich eine Methode `getMitglied`, die als Parameter einen `enum` entgegennimmt. Den `enum` realisieren Sie mit folgenden Werten: `Vater`, `Mutter`, `Kinder`. Die Rückgabe der Methode `getMitglied` ist wie folgt umzusetzen:

- Bei Angabe des enum-Wertes `Vater` bzw. `Mutter` wird der Name des Vaters bzw. der Mutter zurückgegeben.
- Bei Angabe des enum-Wertes `Kinder` werden die Namen aller Kinder (durch Kommata getrennt) zurückgegeben.

Die Methode `getMitglied` soll einen leeren String zurückgeben, wenn das betreffende Familienmitglied nicht existiert, d. h. bei einem kinderlosen Ehepaar muss der **Zugriff auf Array-Elemente ab Position 2 verhindert** werden.

```
import java.util.ArrayList;

public class Familie {

	private ArrayList mitglieder;

	public Familie(String vater, String mutter){
		// Ihr Code hier
	}

	public void addKind(String kind){
		// Ihr Code hier
	}

	public enum Familienmitglied{
		// Ihr Code hier
	}

	// implementieren Sie hier die Methode "getMitglied(...)
}
```

Schreiben Sie ein Hauptprogramm (bitte separat in einer eigenen Klasse `Program`). Das Hauptprogramm soll eine `Familie` mit zwei Kindern instanziieren und ein weiteres kinderloses Ehepaar instanziieren. Anschließend soll das Hauptprogramm durch mehrfachen Aufruf von `getMitglied` zuerst den Vater, dann die Mutter und schließlich die Kinder ermitteln und auf der Konsole ausgeben.

Testen Sie Ihr Programm einmal durch Instanziierung einer kinderreichen Familie und ein weiteres Mal durch Instanziierung eines kinderlosen Ehepaares. Probieren Sie auch aus, was passiert, wenn Sie eine Familie mit leerer Vater-Zeichenkette oder leerer Mutter-Zeichenkette (`"" bzw.null`) instanziieren. Fangen Sie die entstehende `Exception` und geben die Fehlermeldung auf der Konsole aus.

C.3 Präsenzaufgabe (`Iterator`, Datenkapselung)

Leiten Sie die Klasse `Familie` von den Interfaces `Iterator` oder `Iterable` ab und implementieren die entsprechenden Methode(n).

Folgender Code sollte nun im Hauptprogramm funktionieren:

```
Familie familie = ...

for (String name : familie) {
      System.out.println(name);
}
```

Überlegen Sie, ob Sie nun das Prinzip der Datenkapselung verletzt haben. Kann fremder Code nun die Mitglieder der Familie verändern, überschreiben oder gar löschen? Begründen Sie.

C.4 Hausaufgabe (`ArrayList`, `Iterator`)

[3 Punkte]

Der Bibliothekar hat die Konsolen-Ausgabe seines Zettelkastens realisiert und ist schon recht zufrieden. Er steht jedoch vor folgendem Problem. Die Verwendung eines Arrays für die Sammlung aller Medien ist unzweckmäßig. Wenn neue Medien dazu kommen, muss er das Array immer wieder von neuem anlegen. Ebenso kann es vorkommen, dass er Medien aus seinem virtuellen Zettelkasten entfernen muss.

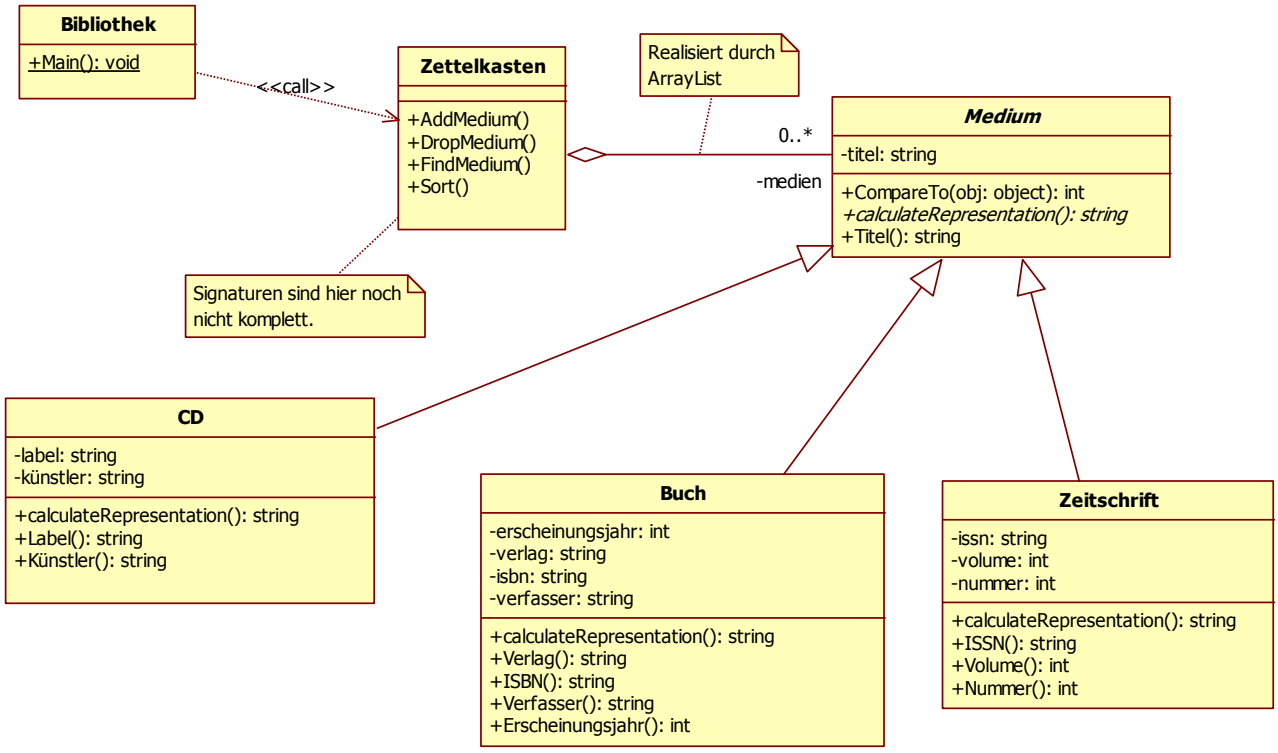


Abbildung 1: UML-Modell der Bibliothek

Zur Lösung des Problems denkt der Bibliothekar an die Verwendung der Klasse `ArrayList`. Lesen Sie das entsprechende des Online-Buches „Java ist auch eine Insel“ . Passen Sie das Hauptprogramm `Bibliothek.main` so an, dass die Medien „Duden“, „Beatles-CD“ und „Spiegel“ nicht mehr als Array, sondern als `ArrayList` gesammelt werden.

Schließlich möchte der Bibliothekar sein Programm ein wenig aufräumen. Er will das Hauptprogramm fast vollständig leer räumen und dessen Inhalte in andere Klassen und Funktionen verlagern.

Dazu programmiert er eine neue Klasse `Zettelkasten`. In dieser Klasse implementiert er die `ArrayList` mit all seinen enthaltenen Medien. Die Klasse `Zettelkasten` soll folgende Operationen besitzen:

- `addMedium`: Fügt Medien zur Liste hinzu. Wichtig dabei ist das nur korrekte (Buch mit gültiger ISBN, und vollständige (z.B. Titel besteht aus mindestens einem Zeichen,etc) in die Liste aufgenommen werden, ansonsten soll eine Fehlermeldung ausgegeben werden.
- `dropMedium`: löscht ein Medium anhand eines, als Parameter übergebenen, Titels. Falls der Titel nicht gefunden wird, wird nichts gelöscht und eine entsprechende Meldung ausgegeben.
- `findMedium`: findet ein Medium anhand eines als Parameter übergebenen Titels. Falls der Titel nicht gefunden wird, wird null zurückgegeben.
- `sort`: s. Aufgabe C.5

Außerdem möchte der Bibliothekar der Klasse `Zettelkasten` die Funktionalität geben, dass ein außenstehender Programmcode durch alle enthaltenen Medien mit einem einfachen foreach-Statement hindurch laufen kann. Dazu ergänzt er die Klasse `Zettelkasten` so, dass sie das Interface `Iterator` bzw.`Iterable` realisiert. Das `main`-Programm soll nachher nur noch folgenden Code enthalten:

```
public class Bibliothek {

	public static void main(String[] args) {
		Zettelkasten zettelkasten = new Zettelkasten();
		zettelkasten.addMedium(new CD("Live At Wembley",
				"Queen", "Parlophone (EMI)"));

		// ... und hier noch 2 weitere Statements
		//     zum Hinzufügen von Medien

		zettelkasten.sort(); // für Aufgabe C.5

		for (Medium medium : zettelkasten) {
			System.out.println(medium.calculateRepresentation());
		}
	}
}
```

C.5 Hausaufgabe (Sortieren, `IComparable`)

[2 Punkte]
Wenn Sie diese Aufgabe bearbeiten, können Sie sie in Gestalt eines Programms zusammen mit Aufgabe C.4 abgeben.

Der Bibliothekar möchte die Medien in der `ArrayList` nach Titel sortieren.

Dazu soll die Klasse `Medium` das Interface `Comparable` realisieren. Implementieren Sie eine Methode `compareTo(…)` in der abstrakten Klasse `Medium`. Sortieren Sie anschließend die Medien in der `ArrayList`.

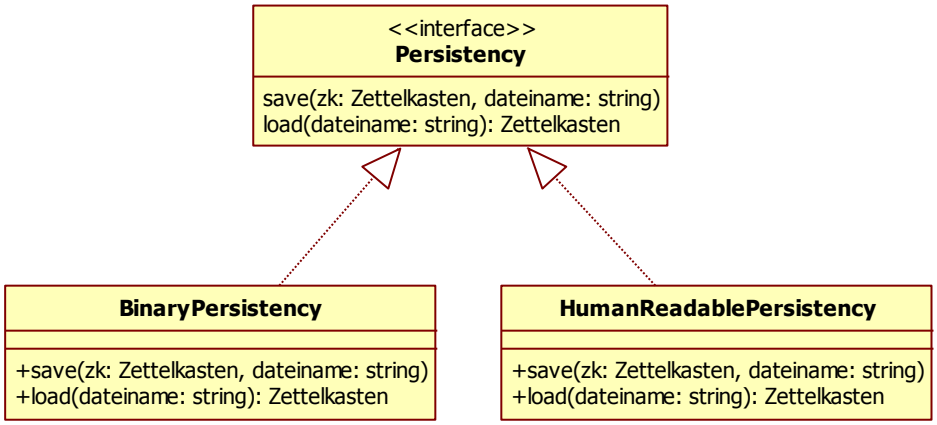
Implementieren Sie in der Klasse `Zettelkasten` die Methode `sort()`. Diese soll, per Parameter wählbar, die enthaltenen Medien absteigend („A“→ „Z“) oder aufsteigend („Z“→ „A“) nach Titel sortieren. Implementieren Sie bitte keinen eigenen Sortieralgorithmus, sondern nutzen einfach eine geeignete Methode der Klasse `ArrayList`.

Steigern Sie die Effizienz des Zettelkastens. Der `Zettelkasten` soll sich merken, ob die darin enthaltenen Medien bereits, bspw. durch einen früheren Aufruf von `sort()`, sortiert sind. In diesem Falle soll die erneute Sortierung entfallen.

C.6 Hausaufgabe (Interfaces, Polymorphie, Serialisierung, File-I/O, Exceptions)

[2 Punkte]
Wenn Sie diese Aufgabe bearbeiten, können Sie sie in Gestalt eines Programms zusammen mit Aufgabe C.4 und C.5 abgeben.

Der virtuelle Zettelkasten existiert zurzeit nur im Hauptspeicher. Der Bibliothekar möchte nun sämtliche Repräsentationen auf der Festplatte sichern. Er stellt sich zwei Varianten der Sicherung vor. Die erste Variante soll lesbar für Menschen sein. D. h. die Repräsentation soll unverändert in eine Datei geschrieben werden. Die zweite Variante soll binär in Form von Objekt-Serialisierung (`ObjectOutputStream`) gesichert werden. Das folgende Diagramm zeigt ein Interface und zwei „Persistency“-Klassen, die der Bibliothekar implementieren möchte.



Implementieren Sie das Interface `Persistency` sowie die beiden Subklassen.
Die Klasse `HumanReadablePersistency` soll die Repräsentation in der folgenden Form in einer Datei als UTF-8-codierten Text abspeichern (Beispiel: Duden):

Titel: Duden 01. Die deutsche Rechtschreibung
Erscheinungsjahr: 2004
Verlag: Bibliographisches Institut, Mannheim
ISBN: 3-411-04013-0
Verfasser: Unbekannt

Die Lade-Operation der Klasse `HumanReadablePersistency` wollen wir im Moment (d. h. für die Lösung dieses Aufgabenzettels) nicht vollständig implementieren. Um sicher zu gehen, dass niemand diese Operation unbedacht aufruft, implementieren Sie bitte die Methode `load` der Klasse `HumanReadablePersistency` indem Sie eine Exception vom Typ `UnsupportedOperationException` generieren.

Die Klasse `BinaryPersistency` soll **beides** (Laden *und* Speichern) unter Zuhilfenahme der Klassen `ObjectInputStream` oder `ObjectOutputStream` umsetzen.

C.7 Hausaufgabe (Erweiterung)

[1 Punkt]
Wenn Sie diese Aufgabe bearbeiten, können Sie sie in Gestalt eines Programms zusammen mit Aufgabe C.4 bis C.6 abgeben.

Der Bibliothekar ist mit unserem Zettelkasten zufrieden, hat aber eine mögliche Fehlerquelle entdeckt. Das Finden und Löschen von Medien erfolgt über den Titel. Diese sind aber nicht immer eindeutig. So kann es z.B. zu einem Buch auch ein Hörspiel gleichen Namens geben. Dies führt zu ungewollten Lösungen bzw. unstimmigen Suchergebnissen.

Ändern Sie daher Ihr Programm zur Behandlung von mehrfachen Einträgen gleichen Titels. Dazu erweitern Sie die

- `findMedium` Operation aus Aufgabe C4 dergestalt, dass bei mehrfachen Einträgen eine generische List von Medien zurückgegeben wird.

- `dropMedium` Operation aus Aufgabe C4 dergestalt, dass bei mehrfachen Einträgen eine `duplicateEntry` Exception zurückgegeben wird. Implementieren Sie zusätzlich eine Variante der `dropMedium` Operation die über geeignete Parameter das Löschen bestimmter oder aller Duplikate eines Eintrags ermöglicht.

- Stellen Sie zusätzlich sicher, dass die Sortierung von Duplikaten nach deren Typ (Zeitschrift, CD, Buch) erfolgt.