Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 2 SoSe 2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof R. Lorenz, L. Petrak, J. Linne, V. Le Claire, P. Schalk 12.07.2024

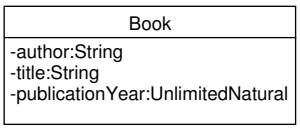
Lösungsvorschlag zu Übungsblatt 11

- Dieses Übungsblatt ist **unbewertet** und wird nicht korrigiert.
- Es ist also nicht abzugeben.
- Alle Aufgaben dieses Übungsblatts sind **klausurrelevant**.

Aufgabe 41 * *(XML und JSON, 21 Minuten)*

In dieser Aufgabe sollen Sie mit den Formaten XML und JSON arbeiten bzw. diese besser kennen lernen.

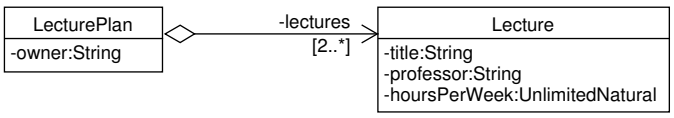
a) *(*, 6 Minuten)*
Erstellen Sie ein gültiges XML-Dokument mit mindestens zwei unterschiedlichen Elementen zu folgender UML-Klasse:



Verwenden Sie ein Element mit dem Namen `Library` als Wurzelelement.
Lösung:

```xml
<Library>
        <Book>
                <author>Christian Ullenboom</author>
                <title>Java ist auch eine Insel</title>
                <publicationYear>2001</publicationYear>
        </Book>
        <Book>
                <author>Brandon Sanderson</author>
                <title>The Final Empire</title>
                <publicationYear>2006</publicationYear>
        </Book>
</Library>
```

b) *(*, 6 Minuten)*
Erstellen Sie ein gültiges JSON-Dokument mit mindestens zwei Vorlesungsplänen und jeweils mindestens zwei Vorlesungen zu folgendem UML-Klassendiagramm:



Speichern Sie die Vorlesungspläne in einem Array.
Lösung:

```
{"lecturePlans": [
                { "owner":"Isabell Informatikerin", "lectures":[
                        { "title":"Informatik 2", "professor":"Robert Lorenz",
                        ↪  "hoursPerWeek":3 },
                        { "title":"Einführung in die theoretische Informatik",
                        ↪  "professor":"Kirstin Peters", "hoursPerWeek":4.5 }
                ]},
                { "owner":"Peter Physiker", "lectures":[
                        { "title":"Physik II", "professor":"Andreas Hörner", "hoursPerWeek":3
                        ↪  },
                        { "title":"Mathematische Konzepte II", "professor":"Sergey
                        ↪  Mikhailov", "hoursPerWeek":4.5 },
                ]},
        ]
}
```

c) *(*, 9 Minuten)*
Überführen Sie das dargestellte JSON-Dokument in ein gültiges XML-Dokument.

```
{"allTrips":[
        { "from":"Augsburg", "to":"Berlin", "when":19.07.2024 },
        { "from":"Strassbourg", "to":"Bratislava", "when":20.08.2024 },
        { "from":"Rome", "to":"Birmingham", "when":03.09.2024 },
]}
```

Lösung:

```
<allTrips>
        <trip>
                <from>Augsburg</from>
                <to>Berlin</to>
                <when>19.07.2024</when>
        </trip>
        <trip>
                <from>Strassbourg</from>
                <to>Bratislava</to>
                <when>20.08.2024</when>
        </trip>
        <trip>
                <from>Rome</from>
                <to>Birmingham</to>
                <when>03.09.2024</when>
        </trip>
</allTrips>
```

Aufgabe 42 ** *(Textdateien, 20 Minuten)*

a) *(**, 8 Minuten)*
Implementieren Sie eine Programmklasse, die einen Dateiauswahl-Dialog zum Laden einer Datei öffnet. Wenn der Nutzer eine Datei auswählt, soll deren Inhalt zeilenweise auf der Kommandozeile ausgegeben werden.

Bei Ausnahmen soll die zugehörige Fehlernachricht auf Kommandozeile ausgegeben werden. Am Ende sollen alle Ressourcen wieder freigegeben werden.

Lösung:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

import javax.swing.JFileChooser;

public class MyFirstFileChooser {

    public static void main(String[] args) {
                JFileChooser loadChooser = new JFileChooser();
                int valChooser = loadChooser.showSaveDialog(null);
                if (valChooser == JFileChooser.APPROVE_OPTION) {
                        File file = loadChooser.getSelectedFile();
                        try (BufferedReader reader = new BufferedReader(new
                        ↪  InputStreamReader(new FileInputStream(file.getPath())))) {
                                String line = reader.readLine();
                                while (line != null) {
                                        System.out.print(line);
                                        line = reader.readLine();
                                }
                        } catch (IOException e) {
                                System.out.println(e.getMessage());
                        }
                }
    }

}
```

b) *(**, 12 Minuten)*
Implementieren Sie eine Programmklasse, die bei jedem Aufruf eine Benutzereingabe liest und diese in einer Datei speichert, sofern diese nicht bereits in der Datei steht. Das Programm soll

- Eine Zeile vom Benutzer über die Kommandozeile einlesen,
- versuchen, die Datei `"a.txt"` zu öffnen,
- den Inhalt der Datei zeilenweise lesen und mit der Benutzereingabe vergleichen,
- wenn die Benutzereingabe gefunden wird, sich diesen Umstand merken und das Lesen der Datei beenden,
- ansonsten den Zeileninhalt in einer `ArrayList` speichern,
- den Lesevorgang der Datei beenden,
- wenn die Benutzereingabe gefunden wurde, den Benutzer darüber informieren und nichts weiter tun und
- wenn die Benutzereingabe nicht gefunden wurde,
  - den Benutzer darüber informieren,
  - die Benutzereingabe in der ersten Zeile der Datei speichern und
  - alle alten Inhalte in derselben Reihenfolge wie zuvor danach in der Datei speichern.

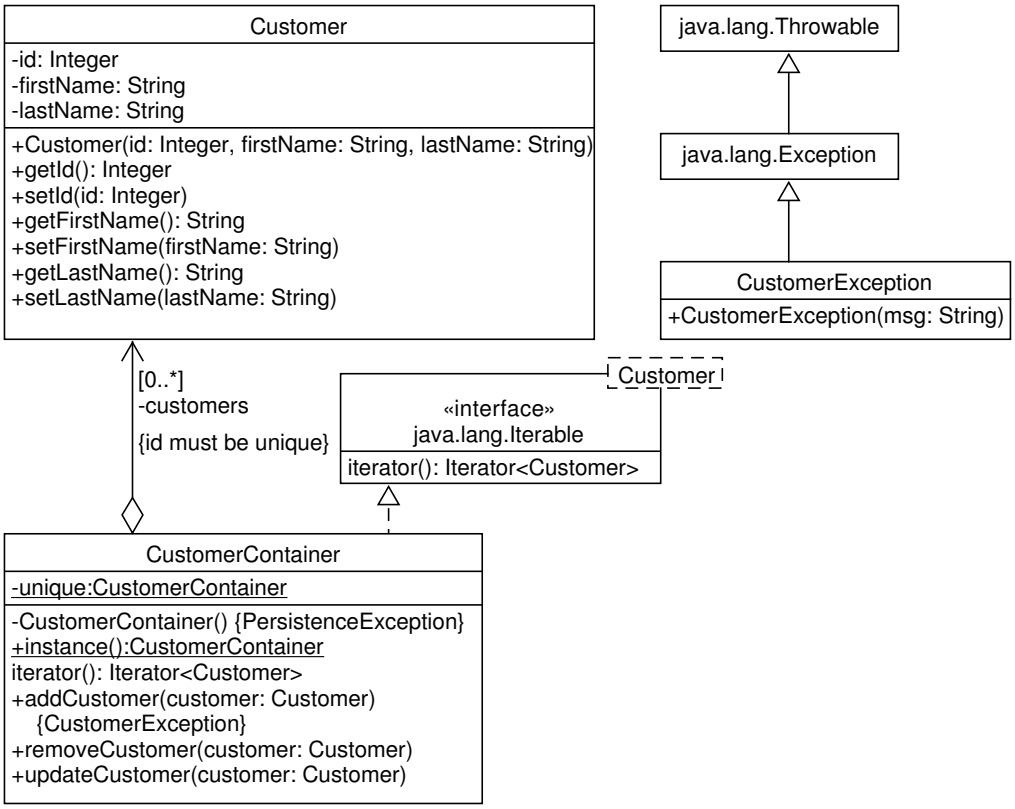
Auftretende Ausnahmen sollen unterdrückt werden. Am Ende sollen alle Ressourcen wieder freigegeben werden.

Lösung:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Scanner;

public class SimpleStringSaver {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        Path path = Paths.get("a.txt");
        boolean found = false;
        ArrayList<String> lines = new ArrayList<>();
        try (BufferedReader reader = Files.newBufferedReader(path)) {
            String line = reader.readLine();
            while (line != null) {
                if (line.equals(input)) {
                    found = true;
                    break;
                }
                lines.add(line);
                line = reader.readLine();
            }
        } catch (IOException e) {

        }
        if (!found) {
            System.out.println(input + " not found, adding to a.txt");
            lines.add(0, input);
            try {
                Files.write(path, lines);
            } catch (IOException e) {

            }
        } else {
            System.out.println(input + " found, leaving a.txt unchanged");
        }
    }
}
```

Aufgabe 43 ** *(Datenhaltung mit Textdateien, 30 Minuten)*

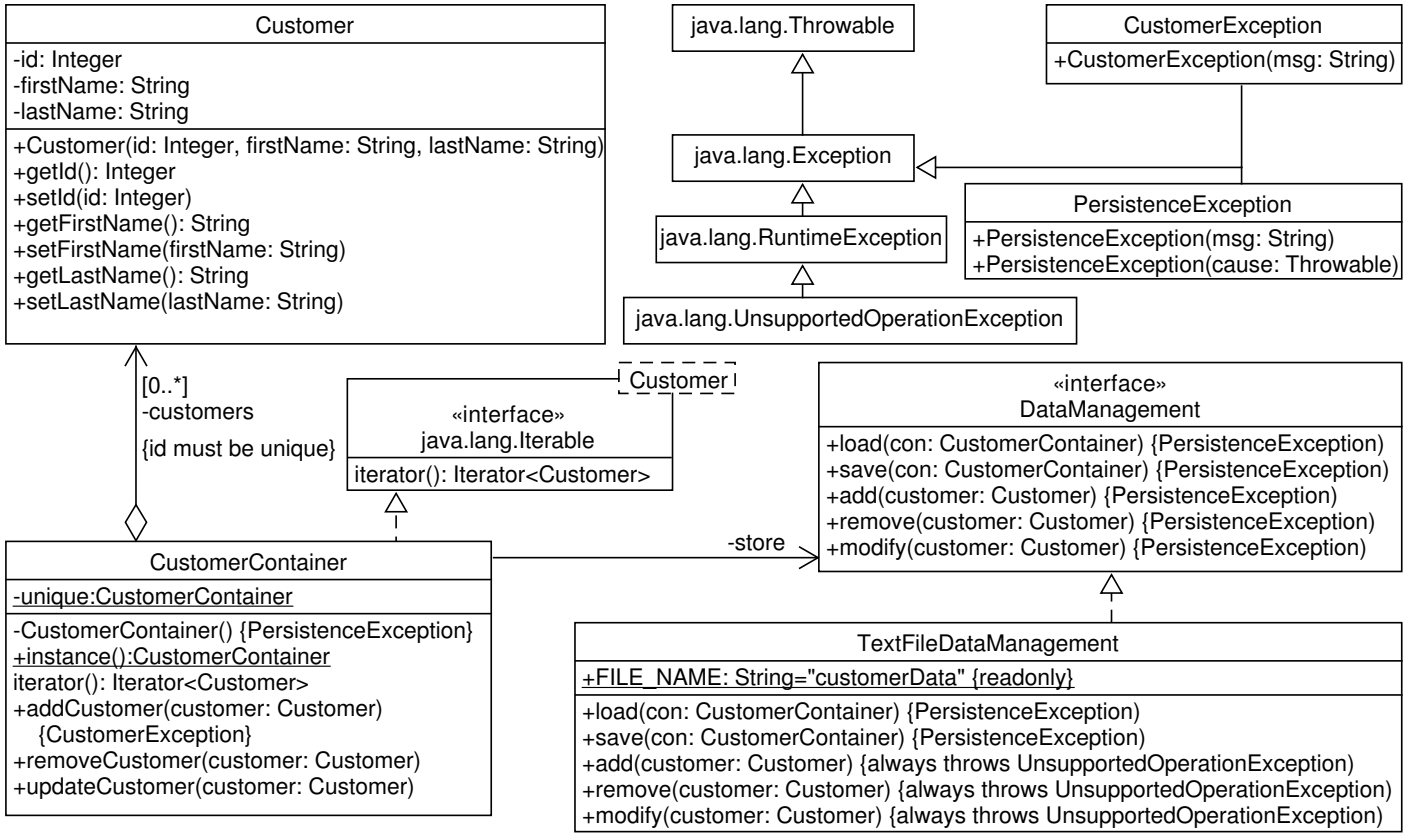
a) *(**, 13 Minuten)*
Betrachten Sie das folgende UML-Klassendiagramm:



Modellieren Sie eine Datenhaltungsschicht mit einer Schnittstelle `DataManagement` mit allen Operationen für globales **und** inkrementelles Speichern. Jede der Operationen reagiert mit dem Werfen einer geprüften Ausnahme `PersistenceException` auf das Auftreten von Fehlern. Eine `PersistenceException` kann sowohl mit einem `Throwable`-Objekt als `cause` auch mit einer Zeichenkette als Nachricht erzeugt werden. Modellieren Sie auch die Ausnahmeklasse. `CustomerContainer`-Objekte sollen über eine Referenz auf ein `DataManagement`-Objekt verfügen.

Modellieren Sie außerdem eine UML-Klasse `TextFileDataManagement`, welche die Schnittstelle `DataManagement` implementiert. Diese verfügt über ein konstantes Klassenattribut, das den Dateinamen `"customerData"` beinhaltet. Die Operationen `add`, `remove` und `modify` sollen immer eine `UnsupportedOperationException` werfen, weil das inkrementelle Speichern in Textdateien nicht sinnvoll ist.
Hinweis: Die .uxf-Datei des gegebenen UML-Klassendiagramms finden Sie bei der Angabe dieses Übungsblatts.

Lösung:



b) *(**, 17 Minuten)*
Implementieren Sie nun die Klasse `TextFileDataManagement` in Java. Dabei soll die Datei in folgendem Format abgespeichert werden:

- Vor den Daten jedes Kunden steht das Schlüsselwort `NEW` in einer eigenen Zeile.
- Die Attribute der Objekte werden jeweils in eigener Zeile in der Reihenfolge abgespeichert, in der sie im UML-Klassendiagramm dargestellt sind.
- Die Textdatei endet mit `END` in einer eigenen Zeile.
- Insbesondere darf auch nach `END` keine weitere Zeile folgen.

Ein Beispiel einer Textdatei könnte wie folgt aussehen:

```
NEW
1
Max
Mustermann
NEW
2
Jane
Doe
END
```

Lösung:

```
import data.Customer;
import data.CustomerContainer;
import data.CustomerException;
import store.PersistenceException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;

public class TextFileDataManagement implements DataManagement {
    private static final String FILE_NAME = "customerData.txt";

    @Override
    public void load(CustomerContainer container) throws PersistenceException {
        try (BufferedReader reader = Files.newBufferedReader(Paths.get(FILE_NAME))) {
            String line = reader.readLine();
            while (!line.equals("END")) {
                if (!line.equals("NEW")) {
                    throw new PersistenceException("Illegal File Format");
                }
                int id = Integer.parseInt(reader.readLine());
                String firstName = reader.readLine();
                String lastName = reader.readLine();
                Customer customer = new Customer(id, firstName, lastName);
                container.addCustomer(customer);
                line = reader.readLine();
            }
            if (reader.readLine() != null) {
                throw new PersistenceException("Illegal File Format");
            }
        } catch (IOException | NumberFormatException | CustomerException e) {
            throw new PersistenceException(e);
        }
    }

    @Override
    public void save(CustomerContainer container) throws PersistenceException {
        try (PrintWriter writer = new
        ↪  PrintWriter(Files.newBufferedWriter(Paths.get(FILE_NAME)))) {
            container.forEach(customer -> {
                    writer.println("NEW");
                    writer.println(customer.getId());
                    writer.println(customer.getFirstName());
                    writer.println(customer.getLastName());
            });
            writer.println("END");
        } catch (IOException e) {
            throw new PersistenceException(e);
        }
    }

    @Override
    public void add(Customer customer) {
        throw new UnsupportedOperationException("Incremental storage not avaiable");
    }

    @Override
    public void remove(Customer customer) {
        throw new UnsupportedOperationException("Incremental storage not avaiable");
    }

    @Override
    public void modify(Customer customer) {
        throw new UnsupportedOperationException("Incremental storage not avaiable");
    }
}
```

Aufgabe 44 ** *(Objektserialisierung, 20 Minuten)*

a) *(**, 10 Minuten)*
Machen Sie sich zuerst mit den im Digicampus gegebenen Klassen `BinaryDialog` und `BinaryDialogFactory` vertraut:

- Führen Sie die Klasse `BinaryDialogFactory` aus (Die Funktionalität der Buttons `Save` und `Load` werden Sie implementieren).
- Arbeiten Sie sich in die Klasse `BinaryDialogFactory` ein.
- Schauen Sie sich die Methode `copyFromBinaryDialog` der Klasse `BinaryDialog` an (letzte Methode im Quellcode).

Implementieren Sie jetzt die Methoden `onSave` und `onLoad` der Klasse `BinaryDialogFactory`:

- Beim Ausführen von `onSave` soll der `BinaryDialog` in eine Binärdatei mit dem Namen `FILENAME` serialisiert werden (die Methode `actionPerformed` stellt sicher, dass dieser auch tatsächlich existiert, bevor onSave aufgerufen wird).
- beim Ausführen von `onLoad` soll die Binärdatei `FILENAME` in ein `BinaryDialog`-Objekt deserialisiert werden.
- Verwenden Sie die Methode `copyFromBinaryDialog`, um den Inhalt des deserialisierten Objekts in das aktuelle `BinaryDialog`-Objekt zu kopieren.
- Stellen Sie dabei außerdem sicher, dass auch ein `BinaryDialog`-Objekt existiert, in das die geladenen Werte kopiert werden können.
- Beim Auftreten eines Fehlers soll eine Fehlermeldung über den Standard-Fehlerstrom auf der Kommandozeile ausgegeben werden.
- Die verwendeten Ressourcen sollen in beiden Methoden in allen Fällen wieder freigegeben werden.

Lösung:

```
private void onSave() {
    // TODO: Implement serialization of the window
    try (ObjectOutputStream writer = new ObjectOutputStream(new
    ↪  FileOutputStream(FILENAME))) {
        writer.writeObject(binaryDialog);
        JOptionPane.showMessageDialog(this, "Saving successful");
    } catch (IOException e) {
        System.err.println("Serialization failed, window was not saved!");
    }
}

private void onLoad() {
    // TODO: Implement deserialization of the window
    try (ObjectInputStream reader = new ObjectInputStream(new FileInputStream(FILENAME)))
    ↪  {
        BinaryDialog temp = (BinaryDialog) reader.readObject();
        if (binaryDialog == null)
            onCreate();
        binaryDialog.copyFromBinaryDialog(temp);
        JOptionPane.showMessageDialog(this, "Loading successful");
    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Deserialization failed, maybe no file " + FILENAME + " was
        ↪  found?");
    }
}
```

b) *(**, 15 Minuten)*
Implementieren Sie eine Klasse `Person` mit folgenden Eigenschaften:

- Die Klasse hat eine Zeichenkette und ein ganzzahliges Alter als Attribute.
- Es gelten keine Constraints für die beiden Attribute.
- Die Klasse verfügt über Setter und Getter sowie einen Konstruktor für beide Attribute.
- Die Zeichenkettendarstellung eines `Person`-Objekts beinhaltet die Werte beider Attribute.
- Die Klasse ist serialisierbar.
- Das Alter wird beim Serialisieren nicht mitgespeichert, recherchieren Sie dazu das Keyword `transient`.

Implementieren Sie außerdem eine `main()`-Methode in der Klasse, die

- ein `Person`-Objekt erzeugt und auf der Kommandozeile ausgibt,
- versucht, dieses Objekt zu serialisieren,
- versucht, dieses Objekt wieder zu deserialisieren und
- das deserialisierte Objekt auf der Kommandozeile ausgibt.
- Im Fehlerfall soll auf das Programm beendet und eine geeignete Meldung auf der Kommandozeile ausgegeben werden.

Lösung:

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class Person implements Serializable {
    private String name;
    private transient int age;

    public Person(String name, int age) {
        setName(name);
        setAge(age);
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                '}';
    }
```

```
    public static void main(String[] args) {
        Person person = new Person("John Doe", 16);
        System.out.println("Original object: " + person);
        try (ObjectOutputStream writer = new ObjectOutputStream(new
        ↪  FileOutputStream("person"))) {
            writer.writeObject(person);
        } catch (IOException ex) {
            System.err.println("Serialization failed");
            System.exit(1);
        }
        try (ObjectInputStream reader = new ObjectInputStream(new FileInputStream("person")))
        ↪  {
            Person deserializedObject = (Person) reader.readObject();
            System.out.println("Deserialized object: " + deserializedObject);
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Deserialization failed");
            System.exit(1);
        }
    }
}
```