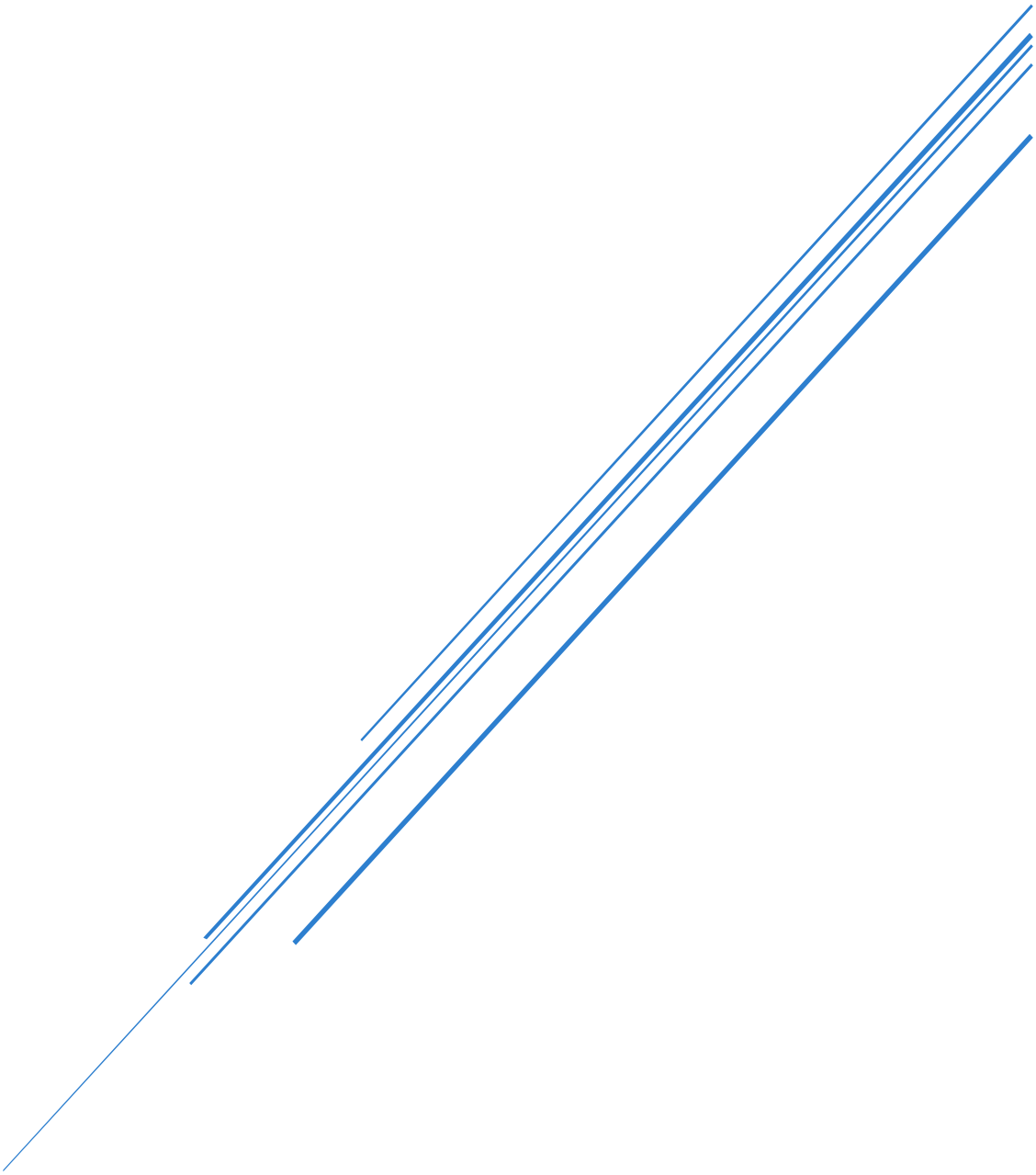
DATA ANALYSIS_UE2

Sophia Spanring



DAN3_UE_T1

12.11.2025

Link zum Git Repository:

sophiaSpanring/Spanring: Java Code for ADS

Aufgabe 1: void append(Object data);

Aufgabenstellung

In der ersten Aufgabe sollte eine Methode append(Object data) entwickelt werden, mit der sich ein neues Element am Ende einer einfach verketteten Liste (SList) anfügen lässt. Zusätzlich war gefordert, die Methode mit realistischen Datensätzen des Typs kapitel_3.tests.Student zu testen.

Vorgehensweise

Ich habe dafür eine eigene Klasse MySList erstellt, die von der vorgegebenen SList erbt. Die Methode append war in der Oberklasse nicht vorhanden, deshalb musste ich sie komplett selbst implementieren.
Der grundlegende Ablauf ist folgender:

- Wenn die Liste noch leer ist (head == null), wird das neue Element einfach zum Kopf der Liste.
- Wenn es schon Elemente gibt, laufe ich mit einem Zeiger (current) bis zum letzten Knoten.
- Anschließend wird ein neuer Node erzeugt und am Ende eingehängt.

Damit wird das Verhalten einer normalen einfach verketteten Liste umgesetzt.

Testdaten

Zum Testen habe ich drei Objekte der Klasse Student erstellt und nacheinander mit append in die Liste eingefügt. Danach habe ich mit dem Iterator die komplette Liste ausgegeben, um zu prüfen, ob die Einfüge-Reihenfolge stimmt.

Java Implementierung:

```java
public void append(Object data) {
    Node newNode = new Node(data, next: null);

    if (head == null) {
        head = newNode;
        return;
    }

    Node current = head;
    while (current.next != null) {
        current = current.next;
    }

    current.next = newNode;
}
```

Testdatei:

```
// Aufgabe 1: append testen

// Beispiel-Studenten anlegen
Student s1 = new Student(name: "M001", sureName: "Anna",  matrNr: "Muster");
Student s2 = new Student(name: "M002", sureName: "Ben",   matrNr: "Beispiel");
Student s3 = new Student(name: "M003", sureName: "Carla", matrNr: "Demo");

// append testen
list.append(s1);
list.append(s2);
list.append(s3);

// Ausgabe zur Kontrolle
System.out.println(x: "Inhalt der Liste nach append:");
IFIterator it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

Ausgabe:

```
Inhalt der Liste nach append:
Name: M001
Surname: Anna
MatrNr: Muster
Name: M002
Surname: Ben
MatrNr: Beispiel
Name: M003
Surname: Carla
MatrNr: Demo
```

Beobachtung / Ergebnis

Die Ausgabe zeigt die drei Studenten genau in der Reihenfolge, in der ich sie hinzugefügt habe. Damit funktioniert die Methode wie erwartet und erfüllt die Anforderungen der Aufgabenstellung.

Aufgabe 2: boolean insert(Objet prev, Object data);

Aufgabenstellung

In dieser Aufgabe sollte eine Methode boolean insert(Object prev, Object data) implementiert werden.
Die Methode soll ein neues Element direkt hinter dem Element prev in die einfach verkettete Liste einfügen. Falls prev gar nicht in der Liste vorkommt, darf nichts eingefügt werden. Zusätzlich muss die Methode true zurückgeben, wenn das Einfügen erfolgreich war, und false, wenn kein Einfügen stattgefunden hat. Wie bei Aufgabe 1 sollten für den Test wieder Student-Objekte verwendet werden.

Vorgehensweise

Die Methode wurde in meiner MySList implementiert. Der Ablauf ist folgender:

- Zuerst prüfe ich, ob die Liste überhaupt ein Element enthält (head == null).
  Ist sie leer, kann prev nicht existieren → Rückgabe false.
- Danach gehe ich mit einer Schleife alle Knoten durch, bis ich denjenigen finde, dessen data-Feld dem übergebenen Objekt prev entspricht.

- Wenn dieser Knoten nicht gefunden wird, breche ich ab und gebe ebenfalls false zurück.
- Wird der Knoten gefunden, erstelle ich einen neuen Node und hänge ihn direkt hinter dem gefundenen Knoten ein.
- In diesem Fall gebe ich true zurück.

Damit erfüllt die Methode genau das geforderte Verhalten für eine einfach verkettete Liste.

Testdaten

Ich habe wieder mit mehreren Student-Objekten gearbeitet. Zuerst habe ich drei Studenten in die Liste eingefügt.
Anschließend habe ich ein viertes Student-Objekt hinter s2 eingefügt, um einen erfolgreichen Insert-Fall zu testen.
Danach habe ich noch einen Negativfall getestet, indem ich ein Objekt verwendet habe, das nicht in der Liste vorkommt.
Dieser Test muss false zurückliefern und die Liste unverändert lassen.

Java Implementierung

```java
public boolean insert(Object prev, Object data) {

    if (head == null) {
        return false;
    }

    Node current = head;
    while (current != null && current.data != prev) {
        current = current.next;
    }

    if (current == null) {
        return false;
    }

    Node newNode = new Node(data, current.next);
    current.next = newNode;

    return true;
}
```

Testdatei:

```java
// Aufgabe 2: insert testen

// neues Student-Objekt, das wir hinter s2 einfügen wollen
Student s4 = new Student(name: "M010",sureName: "Lena",matrNr: "Neu");

System.out.println();
System.out.println(x: "Aufgabe 2: insert");
System.out.println(x: "Liste vor insert(s2, s4):");
it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}

// erfolgreicher Insert: s4 soll hinter s2 eingefügt werden
boolean result1 = list.insert(s2, s4);
System.out.println();
System.out.println("Rückgabewert von insert(s2, s4): " + result1);

System.out.println(x: "Liste nach insert(s2, s4):");
it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}

// Negativfall: prev ist NICHT in der Liste - insert muss false liefern
Student notInList = new Student(name: "X999",sureName: "Fake",matrNr: "KeinEintrag");
boolean result2 = list.insert(notInList,
        new Student(name: "M011",sureName: "Extra",matrNr: "SollNichtEingefügtWerden"));

System.out.println();
System.out.println("Rückgabewert von insert(notInList, ...): " + result2);
System.out.println(x: "(hier soll false herauskommen, Liste bleibt unverändert)");
```

Ausgabe:

```
Aufgabe 2: insert
Liste vor insert(s2, s4):
Name: M001
Surname: Anna
MatrNr: Muster
Name: M002
Surname: Ben
MatrNr: Beispiel
Name: M003
Surname: Carla
MatrNr: Demo

Rückgabewert von insert(s2, s4): true
Liste nach insert(s2, s4):
Name: M001
Surname: Anna
MatrNr: Muster
Name: M002
Surname: Ben
MatrNr: Beispiel
Name: M010
Surname: Lena
MatrNr: Neu
Name: M003
Surname: Carla
Surname: Ben
MatrNr: Beispiel
Name: M010
Surname: Lena
MatrNr: Neu
```

```
Surname: Carla
MatrNr: Beispiel
Name: M010
Surname: Lena
MatrNr: Neu
Name: M003
Surname: Carla
MatrNr: Neu
Name: M003
Surname: Carla
Name: M003
Surname: Carla
Surname: Carla
MatrNr: Demo
MatrNr: Demo

Rückgabewert von insert(notInList, ...): false
(hier soll false herauskommen, Liste bleibt unverändert)
```

Beobachtung / Ergebnis

Der erfolgreiche Insert-Fall hat wie erwartet funktioniert: Der neue Student wurde direkt hinter s2 eingefügt.
Beim Negativfall wurde false zurückgegeben, und die Liste wurde überhaupt nicht verändert.
Damit arbeitet die Methode korrekt und erfüllt die Anforderungen der Aufgabe.

Aufgabe 3: SList searchAll(IKey key);

Aufgabenstellung

In der dritten Aufgabe sollte eine Methode SList searchAll(IKey key) implementiert werden. Damit sollen alle Elemente in der Liste gefunden werden, die zu einem bestimmten Schlüssel passen. Alle passenden Datensätze müssen in einer neuen SList gespeichert und anschließend zurückgegeben werden. Auch hier sollten wieder Student-Objekte als Testdaten verwendet werden.

Vorgehensweise

Ich habe in meiner Klasse MySList eine neue searchAll-Methode geschrieben.
Der Ablauf ist:

- Ich durchlaufe die gesamte Liste mit einem current-Zeiger.
- Für jedes Element wird mit key.matches(current.data) geprüft, ob es zum Suchkriterium passt.

- Wenn es ein Treffer ist, füge ich das Element mit prepend in eine neue Ergebnisliste ein.
- Am Ende wird diese Liste zurückgegeben.

Testdaten

Zum Testen habe ich wieder mehrere Student-Objekte in meine Liste eingefügt.
Zusätzlich habe ich einen weiteren Studenten hinzugefügt, der denselben Matrikelnummer-Text enthält, damit es mehrere Treffer gibt.
Der Schlüssel (IKey) wurde so implementiert, dass er alle Studenten findet, deren toString() das Wort „Muster“ enthält.

Java Implementierung:

```
public SList searchAll(IKey key) {
    SList result = new SList();

    Node current = head;
    while (current != null) {
        if (key.matches(current.data)) {
            result.prepend(current.data);
        }
        current = current.next;
    }

    return result;
}
```

Testdatei:

```
// Aufgabe 3: searchAll testen

// Noch einen Student ergänzen, der auch "Muster" in der Ausgabe hat
Student s5 = new Student(name: "M004",sureName: "Lena",matrNr: "Muster");
list.append(s5);

System.out.println();
System.out.println(x: "Aufgabe 3: searchAll");

// IKey, der alle Studenten findet, deren toString() "Muster" enthält
IKey key = new IKey() {
    @Override
    public boolean matches(Object object) {
        return object != null && object.toString().contains(s: "Muster");
    }
};

// Suche alle passenden Studenten
SList resultList = list.searchAll(key);

System.out.println(x: "Alle Studenten, die 'Muster' enthalten:");
IFIterator it3 = resultList.iterator();
while (it3.hasNext()) {
    System.out.println(it3.next());
}
```

Ausgabe:

```
Aufgabe 3: searchAll
Alle Studenten, die 'Muster' enthalten:
Name: M004
Surname: Lena
MatrNr: Muster
Name: M001
Surname: Anna
MatrNr: Muster
```

Beobachtung / Ergebnis

Die Methode findet beide passenden Studenten und gibt sie in einer neuen SList zurück. Durch die Verwendung von prepend erscheinen die Treffer in umgekehrter Reihenfolge (zuerst der zuletzt gefundene Student).
Das Verhalten stimmt mit meiner Implementierung überein und erfüllt die Anforderungen der Aufgabe vollständig.

Aufgabe 4: Telefonbuch

Aufgabenstellung

In der letzten Aufgabe sollte eine kleine Telefonbuchverwaltung umgesetzt werden, die auf der bereits vorhandenen SList basiert.
Dazu mussten zwei eigene Klassen entwickelt werden:

1. TelephoneBookEntry – ein Datensatz, der Vorname, Nachname, mehrere Telefonnummern und die Adresse einer Person speichern kann.
2. TelephoneBookEntryKey – ein Suchschlüssel, der anhand von Vorname und Nachname passende Einträge in der Liste findet.

Abschließend sollte die Implementierung mit mehreren realistischen Beispielen getestet werden, sowohl mit search() als auch mit searchAll().

Vorgehensweise

Zuerst habe ich die Klasse TelephoneBookEntry erstellt.
Der Datensatz enthält folgende Informationen:

- firstName
- lastName
- eine Liste für mehrere Telefonnummern (List<String>)
- address

Außerdem gibt es eine Methode addPhoneNumber(), um eine neue Nummer zum Eintrag hinzuzufügen.
Das war sinnvoll, weil eine Person ja oft mehrere Telefonnummern besitzen kann.
Als nächstes habe ich die Klasse TelephoneBookEntryKey implementiert.
Diese Klasse verwendet die Schnittstelle IKey und überprüft im matches-Methodenaufruf, ob sowohl der Vorname als auch der Nachname eines TelephoneBookEntry zu den vorgegebenen Suchwerten passen.
Damit kann man gezielt nach genau einer Person suchen, auch wenn es mehrere Personen mit demselben Nachnamen gibt.

Testdaten

Für den Test habe ich eine eigene MySList für die Telefonbucheinträge angelegt und drei Einträge eingefügt.
Bei einem Eintrag habe ich zwei Telefonnummern angelegt, um die Mehrfachnummernfunktion zu testen.
Der Schlüssel für die Suche war ein TelephoneBookEntryKey, der auf die Werte ("Anna", "Huber") gesetzt wurde.
Damit habe ich sowohl search() als auch searchAll() ausprobiert.

Java Implementierung:

```java
package ue_2_s2410238052;

import java.util.ArrayList;
import java.util.List;

public class TelephoneBookEntry {

    private String firstName;
    private String lastName;
    private List<String> phoneNumbers;
    private String address;

    public TelephoneBookEntry(String firstName, String lastName, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.phoneNumbers = new ArrayList<>();
    }

    public void addPhoneNumber(String number) {
        phoneNumbers.add(number);
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public List<String> getPhoneNumbers() {
        return phoneNumbers;
    }

    public String getAddress() {
        return address;
    }

    @Override
    public String toString() {
        return "Name: " + firstName + " " + lastName +
                "\nAdresse: " + address +
                "\nTelefonnummern: " + phoneNumbers + "\n";
    }
}
```

```java
package ue_2_s2410238052;

import lecture.chapter03.IKey;

public class TelephoneBookEntryKey implements IKey {

    private String firstName;
    private String lastName;

    public TelephoneBookEntryKey(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public boolean matches(Object object) {
        if (!(object instanceof TelephoneBookEntry)) {
            return false;
        }

        TelephoneBookEntry entry = (TelephoneBookEntry) object;

        return entry.getFirstName().equalsIgnoreCase(firstName)
            && entry.getLastName().equalsIgnoreCase(lastName);
    }
}
```

Testdatei:

```java
// Aufgabe 4: Telefonbuch-Einträge testen

System.out.println();
System.out.println(x: "Aufgabe 4: Telefonbuch Testen");

MySList phoneList = new MySList();

TelephoneBookEntry e1 = new TelephoneBookEntry(firstName: "Anna", lastName: "Huber", address: "Linz, Landstrasse 1");
e1.addPhoneNumber(number: "0660 1234567");
e1.addPhoneNumber(number: "0732 555555");

TelephoneBookEntry e2 = new TelephoneBookEntry(firstName: "Ben", lastName: "Huber", address: "Wels, Kaiser Josef Platz 3");
e2.addPhoneNumber(number: "0650 9876543");

TelephoneBookEntry e3 = new TelephoneBookEntry(firstName: "Anna", lastName: "Müller", address: "Steyr, Ennskai 10");
e3.addPhoneNumber(number: "0676 1122334");

phoneList.append(e1);
phoneList.append(e2);
phoneList.append(e3);

// Key zum Suchen nach Vorname + Nachname
TelephoneBookEntryKey keyHuberAnna = new TelephoneBookEntryKey(firstName: "Anna", lastName: "Huber");

System.out.println(x: "\nSuche nach Anna Huber (search):");
Object found = phoneList.search(keyHuberAnna);
System.out.println(found);

System.out.println(x: "Suche nach allen Hubers mit Anna (searchAll):");
SList allFound = phoneList.searchAll(keyHuberAnna);

var it4 = allFound.iterator();
while (it4.hasNext()) {
    System.out.println(it4.next());
}
```

Ausgabe:

```
Aufgabe 4: Telefonbuch Testen

Suche nach Anna Huber (search):
Name: Anna Huber
Adresse: Linz, Landstrasse 1
Telefonnummern: [0660 1234567, 0732 555555]

Suche nach allen Hubers mit Anna (searchAll):
Name: Anna Huber
Adresse: Linz, Landstrasse 1
Telefonnummern: [0660 1234567, 0732 555555]
```

Beobachtung / Ergebnis

Beide Suchmethoden haben den richtigen Eintrag gefunden.
Da es nur genau eine Person gab, die zu diesem Namen passt, war das Ergebnis bei searchAll() identisch zu search().
Die gespeicherten Telefonnummern wurden korrekt in der Ausgabe angezeigt.
Damit funktioniert die gesamte Telefonbuchverwaltung wie erwartet und erfüllt vollständig die Anforderungen aus der Aufgabenstellung.