SportsClub – Beschreibung der Klassen

- Der Packagename für Ihr Projekt lautet **aMatrikelnummer**.
- Instanzvariablen sind `private` zu deklarieren.
- Die Signaturen der vorgegeben Methoden dürfen nicht verändert werden.
- Java Version am almighty ist Java 11; verwenden Sie diese um Import-Probleme beim Test zu vermeiden!

Zum Test dürfen Sie nur die **Basisimplementierung** mitbringen. Speichern Sie sich also gegebenenfalls einen Zwischenstand bevor Sie die Erweiterung für den Zusatzpunkt implementieren. **Abgabe der Basisimplementierung bis 17.01.21**

1 Basis

Folgendes ist Teil der Basisimplementierung

1.1 Enumeration: Sports

```
public enum Sports {
    ARCHERY,
    BASKETBALL,
    CLIMBING {
        // due to additional insurance fees, getFeeFactor() returns 1.2
    },
    DIVING {
        // due to renting equipment and insurance, getFeeFactor() returns
            1.8
    },
    FOOTBALL,
    GOLF {
        // due to additional fees for the golf course, getFeeFactor()
            returns 2.1
    },
    HANDBALL,
    HOCKEY,
    MOUNTAINBIKING,
    PARKOUR;

    public BigDecimal getFeeFactor() {/* default: 1.0 */}

    public BigDecimal getFee(BigDecimal feePerSports) {
        // calculate fees depending on sport club's feePerSports *
            feeFactor
    }
}
```

Hinweis: Informationen zu BigDecimal siehe JavaDocs.

1.2 Enumeration: Level

```
public enum Level {
    BEGINNER("Anfänger"),
    NORMAL("Normal"),
    ADVANCED("Fortgeschritten"),
    PROFESSIONAL("Profi");

    String mappedName;

    //constructor (check for valid values!) + getter

    public Level next() {
     // returns next level, last level if no further levels left (hint:
         this.ordinal(), Level.values())
    }

    @Override
    public String toString() {/* returns mappedName */}
}
```

1.3 Klasse: Member

```
public class Member implements Comparable<Member> {
    private String name;
    private Map<Sports, Level> sports = new LinkedHashMap<>();

    public Member(String name) {
      // throws IllegalArgumentException if name is null or empty
      // set this.name
    }

    public Member(String name, Map<Sports, Level> sportsLevelMap) {
      // delegate to this constructor (String)
      // throws IllegalArgumentException if sportsLevelMap is null, empty
         or contains null-values.
      // set this.sports to a _copy_ (shallow) of sportsLevelMap
    }

    public String getName() {/*...*/}

    public Map<Sports, Level> getSports() {/*return a _copy_ (shallow) of
        sports */}

    public Set<Sports> getBillableSports() {/* return set of all sports */
        }

    public Level learn(Sports newSports, Level newLevel) {
        // get the member's currentLevel of newSports within this.sports
        // 1) if there is no currentLevel (i.e. null) (meaning the Member
            hasn't practised the sports yet), put (newSports, Level.
            BEGINNER) to this.sports Map and return Level.BEGINNER
        // 2) if the difference between newLevel and currentLevel is
            greater than or equal to 1 (meaning the Member tries a) to
            learn the sports way above his/her skills or b) the correct
            next level) put (newSports, currentLevel.next()) to this.sports
             Map and return the inserted level
        // 3) otherwise return the currentLevel
    }
```

```
    @Override
    public String toString() {
        // format: "name: %s, sports: %s"
    }

    @Override
    public int compareTo(Member member) {/* compare names (case sensitive)
        */}

    @Override
    public boolean equals(Object obj) {/* true if names are equal (case
       sensitive) */}

    @Override
    public int hashCode() {/* return hashCode of name (case sensitive) */}
}
```

1.4 Klasse: Trainer

```
public class Trainer extends Member {
    private Map<Sports, Level> accreditations;

    public Trainer(String name, Map<Sports, Level> accreditations) {
        // delegate to super constructor (String, Map<Sports,Level>)
        // set this.accreditations to a _copy_ (shallow) of accreditations
            argument
    }

    public Map<Sports, Level> getAccreditations() {/* returns a _copy_ (
       shallow) of this.accreditations */}

    @Override
    public Set<Sports> getBillableSports() {
        // get billable sports of super (i.e. shallow copy!) and remove
            all sports that are contained within accreditations,
        // i.e. trainers don't have to pay for the sports they are
            offering
        // return billable sports for this trainer
    }

    @Override
    public String toString() {
        // format: super's toString() + ", accreditations: %s"
    }
}
```

1.5 Klasse: SportsClub

```
public class SportsClub {
    private String name;
    private BigDecimal feePerSports;
    private Set<Member> members = new LinkedHashSet<>();
    private Map<Sports, Set<Trainer>> offeredSports = new LinkedHashMap
        <>();

    public SportsClub(String name, BigDecimal feePerSports) {
        // throws IllegalArgumentException if name is null or empty
        // throws IllegalArgumentException if feePerSports is null
        // set member variables
    }

    public String getName() {/* return name */}

    public BigDecimal getFeePerSports() {/* return feePerSports */}

    public Set<Member> getMembers() {/* return _copy_ (shallow) of members
         */}

    public Set<Sports> getSports() {/* return set of offered sports */}

    public BigDecimal calculateMembershipFee(Member member) {
        // throw IllegalArgumentException if member is no member of this
            sports club
        // get member's billable sports and filter the member's billable
            sports according to this.offeredSports
        // i.e. the sports club is only allowed to bill sports that are
            offered
        // based on the filtered sports list, return accumulated fees
            using Sports.getFee(BigDecimal) method using this.feePerSports
    }

    public boolean registerSports(Member member, Sports sports, Level
        level) {
        // throw IllegalArgumentException if member is no member of this
            sports club
        // register a member for a sports course for a specific level:
        // 1. check whether there is a trainer available for the given
            sports that has an accreditation greater or equal to level
        // 2. return false if there is no trainer available
        // 3. let the member learn the sports using Member.learn(...) and
            return true if he/she was successful to learn at the specified
            level, false otherwise
    }

    public boolean addMember(Member member) {
        // if the member is a Trainer (check using ' instanceof ' keyword)
        // add the trainer's accreditations to the sport club's
            offeredSports map, adding the trainer to the Set (value) of the
             map.
        // return whether the member was added to the this.members set or
            not
    }
```

```
    public boolean removeMember(Member member) {
        // if the member is a Trainer (check using ' instanceof ' keyword)
        // - remove the trainer from the sport club's offeredSports-Map (
            Values)
        // - if the trainer was the only Trainer for a sports, remove the
            key from the offeredSports
        // return true if the member was successfully removed from this.
            members
    }

    @Override
    public String toString() {
        // format: SportsClub[name: %s, feePerSports: %s, offeredSports: %
            s]
    }
}
```

1.6 Klasse: Main

Ein `Main.java`-File wird über Moodle zum Download bereitgestellt. Dieses garantiert keine Korrektheit Ihres Programmes, kann aber einen groben Überblick geben, ob die grundlegenden Funktionalitäten vorhanden sind.

2 Zusatzaufgabe

- Sofern Sie den Zusatzpunkt erhalten möchten, beachten Sie bitte, dass in diesem Fall sowohl für die Basisimplementierung, als auch für die Zusatzimplementierung eine automatische **Plagiatsüberprüfung** durchgeführt wird.
- Projektpunkte tragen **nicht** zu den, für einen positiven Abschluss erforderlichen Testpunkten bei, führen aber im Fall einer positiven Bewertung gegebenenfalls zu einer Verbesserung der Note.
- **Abgabe des Zusatzes bis 31.01.21**
- Denken Sie daran sich Ihre Basisimplementierung für den Test zwischen zu speichern bevor Sie mit dem Zusatz beginnen.

Laden Sie sich die vorgegebenen Dateien für den Zusatz von Moodle herunter.

example.dot Beispielhafte Ausgabe-Datei des zu erstellenden Programms.

example.png Gerenderter Graph der auf Basis der *example.dot* Datei erstellt wurde.

SimpleCsvParser.java Vorgegebener und zu ergänzender CSV-Parser zum Einlesen der CSV Dateien sowie zum Erzeugen der SportsClub-Objekte.

csv/ Ordner, mit CSV Eingabe-Dateien. Eine Datei beschreibt einen `SportsClub`; eine Zeile beschreibt jeweils einen `Member`/`Trainer`.

2.1 Eingabeformat – CSV

Machen Sie sich mit dem vorgegebenen CSV[1] Format vertraut: Jedes CSV-File beginnt mit einer "Header"-Zeile welche das Format nochmals erklärt und beim Einlesen ignoriert werden soll.

```
#type(M|T);name;sports,level,sports,level,...
T;Feike;FOOTBALL,BEGINNER,PARKOUR,ADVANCED,CLIMBING,NORMAL
M;Iyanna;PARKOUR,NORMAL,GOLF,ADVANCED,HANDBALL,ADVANCED
...
```

Wie im obigen Beispiel ersichtlich besteht jede Zeile aus drei Spalten, die durch Semikolons (;) voneinander getrennt sind:

1. type: die Spalte kann die Werte M (Member) bzw. T (Trainer) annehmen.
2. name: die Spalte enthält den Namen des Members bzw des Trainers.
3. sports,level - map: durch Komma (,) getrennte, beliebig lange Liste bestehend aus (Sports, Level) - Einträgen

2.2 Klasse: SimpleCsvParser

Machen Sie sich mit der vorgegebenen Klasse `SimpleCsvParser` vertraut. Diese muss nur noch um das Erstellen der Objekte aus einer Zeile ergänzt werden (`parseLine(...)`). Falls notwendig ergänzen Sie die Klasse um weitere Helper-Methoden. Beispielhafte-Verwendung der Klasse[2]:

```
public static void main() {
    final SportsClub sportsClub = new SportsClub("sport_club_a",
        BigDecimal.TEN);
    final List<String> allLines = SimpleCsvParser.readAllLinesFrom("src/
        a123456/sport_club_a.csv");
    //... beg iterate over each line
    //..... SimpleCsvParser.parseLine(...) -> returns Member, add Member
        to sportsClub
    //... end
}
```

2.3 Klasse: UmbrellaOrganisation

Erstellen Sie eine neue Klasse `UmbrellaOrganisation`

```
public class UmbrellaOrganisation {
    private final Sport sports;
    private Set<SportsClub> clubs;
    //... more members / methods you need
}
```

2.4 Klasse: ExtensionMain

Erstellen Sie eine neue Java Datei `ExtensionMain.java` und entsprechenden Code um alle CSV Dateien aus dem Ordner einzulesen. Erstellen Sie für jede CSV Datei genau einen `SportsClub` und fügen Sie die eingelesenen `Member`/`Trainer` dem jeweiligen `SportsClub` hinzu. Fügen Sie anhand der angebotenen Sportarten (`offeredSports`) aller `SportsClub`s diese den jeweiligen `UmbrellaOrganisation`s hinzu bzw. erstellen Sie für Sportarten neue (sofern noch keine entsprechenden existieren).

[1] https://de.wikipedia.org/wiki/CSV_(Dateiformat)

[2] normalerweise werden Artefakte **nicht** im src-Folder abgelegt

```
public class ExtensionMain {
    //... more members / methods you need

    public static void main(String[] args) throws Exception {
        // read csv files and generate SportsClubs containing members and
            trainers
        // (SportsClub's name is the filename of the CSV, assume a
            feePerSports of BigDecimal.TEN)
        // for each sports that occurs, generate an UmbrellaOrganisation
            and add all SportsClubs that offer that specific sports
        // for each UmbrellaOrganisation write a DOT-file:
        // - name of the digraph is the name of the sports
        // - the root of the tree is the sports
        // - the 2nd level contains all SportsClubs offering that specific
             sports
        // - the 3rd level contains all Members and Trainers that practise
             that specific sports
        // - if a member is Trainers for that specific sports, the trainer
            's node and the edge is red
    }

    // for an example see example.dot and example.png respectively
}
```

2.5 Ausgabeformat – DOT

Das Ausgabeformat DOT[3] ist eine einfache Sprache um Graphen zu beschreiben. Für die Abbildung der Hierarchie soll ein gerichteter Graph (digraph) verwendet werden; als Name für den Graph wird die Sportart der `UmbrellaOrganisation` gewählt. Die einzelnen Kanten werden am Einfachsten paarweise beschrieben. Aufgrund von potenziellen Leerzeichen in den Namen müssen Namen der Knoten von Hochkommata umgeben sein.

```
digraph CLIMBING {
  "CLIMBING" -> "Sportverein 1";
  "Sportverein 1" -> "Member 1";
  "Sportverein 1" -> "Member 2";
  ...
  "CLIMBING" -> "Sportverein n";
  "Sportverein n" -> "Trainer m"[color="red"];
  "Trainer m"[color="red"];
}
```

Um aus der Beschreibung des Graphen ein Bild zu generieren, können Sie z.B. GraphvizOnline oder webgraphviz.com verwenden oder Sie installieren sich `dot`. Folgender Befehl generiert ein PNG-File:

```
dot climbing.dot -Tpng -o climbing.png
```

[3] https://de.wikipedia.org/wiki/DOT_(GraphViz)