

Ausgabe: 27.11.2017
Abgabetermin: Montag, 11.12.2017, 11:00 Uhr

Prof. Norbert Siegmund
Nathalie Dittrich

Bitte lesen Sie die folgenden Informationen zum Übungsablauf **sorgfältig** durch.

Grundsätzlich – wenn nicht anders angegeben – sind die Lösungen zu den Übungen zu Programmierung I jeden **Dienstag bis spätestens 12:00 Uhr** an die jeweiligen Tutoren per E-Mail zu schicken.

Schreiben Sie bitte im Betreff Ihrer E-Mail Ihre **Teamnummer** sowie die Nummer des Übungsblattes. In der E-Mail schreiben Sie bitte zusätzlich Ihren **Namen** und **Matrikelnummer**. Die Lösungen für Sie bitte als Java Dateien als Anlage hinzu. Es werden **keine** kompilierten Dateien, wie *.class oder *.jar angenommen.

Übungen müssen von **minimal ein** und **maximal zwei** Studierenden aus derselben Übungsgruppe in einem festen Team bearbeitet werden (Ausnahmen nur auf Anfrage beim Übungsleiter). Pro Team soll die Lösung nur einmal abgegeben werden. Aufgaben sollen **im Team gelöst** und nicht nur vom Team abgegeben werden. Sie müssen mindestens **50%** dieser Punkte für eine Zulassung zur Prüfung erreichen. Das **Abschreiben** identischer Lösungen wird jeweils mit 0 Punkten bewertet.

Bei Fragen oder Unklarheiten wenden Sie sich bitte **vor der Abgabe** des Übungsblattes an den Übungsleiter (per E-Mail oder persönlich). Es soll nie jemand sagen müssen: „Wir haben die Aufgabe nicht verstanden und konnten sie daher nicht bearbeiten.“

Weitere Informationen, wie aktuelle Ankündigungen, die Angaben finden Sie online (<https://www.uni-weimar.de/de/medien/professuren/intelligente-softwaresysteme/lehre/>) unter Einführung in die Programmierung

Aufgabe 1 Advent Advent (7 Punkte)

Passend zum Dezember-Beginn:

(a) Implementieren Sie eine Klasse Adventskalender.

- Die 24 Türchen (Array) sind mit Objekten vom Typ **Sweetie** gefüllt.
- **Sweetie** ist ein Aufzählungstyp. Es gibt: Lebkuchen, Plätzchen, Praline, Schoko-Nikolaus, Gummibaerchen und Spekulatius.
- Der Konstruktor initialisiert die Türchen per Zufallsgenerator.
- Stellen Sie sicher, dass auf die Türchen nicht direkt von außerhalb der Klasse zugegriffen werden kann.
- Die Methode **public String openDoor(int day)** öffnet das entsprechende Türchen, falls **day** heute ist oder in der Vergangenheit liegt, sonst erfolgt die Rückgabe: Heute ist noch nicht der **day**. Dezember. (Hinweis: `(new java.util.GregorianCalendar()).get(java.util.Calendar.DAY_OF_MONTH)` ist der aktuelle Tag).
- Verbirgt sich hinter dem Türchen ein Schoko-Nikolaus wird zusätzlich ein Sound abgespielt (Verwenden Sie die beigefügte Klasse **MyAudio.java** und die **jingle-bells.wav**. Zum Abspielen des Sounds kann der folgende Aufruf verwendet werden: `MyAudio.play("jingle-bells.wav");`)

(b) Erstellen Sie eine weitere Klasse mit einer **main**-Methode zur Interaktion mit dem Benutzer. Es wird abgefragt, welches Türchen er öffnen will. Der Inhalt wird angezeigt, danach wird der Benutzer gefragt, ob er noch weitere Türchen öffnen will, bis er dies verneint und das Programm so beendet.

Aufgabe 2 Rekursion (3 Punkte)

Implementieren Sie die Funktion **public static boolean isPalindromeRec(String s)**. Diese soll **rekursiv** überprüfen, ob das in **s** übergebene Wort ein Palindrom ist, oder nicht. Ein Wort ist ein Palindrom, wenn es von vorne und hinten gelesen das selbe ergibt, z.B. also reittier oder lagerregal.

Hinsweis: Sie dürfen zusätzliche (Hilfs-)Funktionen implementieren.

Aufgabe 3 Bundesliga (14 Punkte)

Die Deklarationen in Klammern in den folgenden Punkten sind die Klassen Member Variablen (Attribute) der jeweiligen Klassen. Implementieren Sie folgende Objekte:

- (a) Ein Fußballspieler (**SoccerPlayer**) hat einen Namen, eine Nummer, gehört zu einem Team, kann gelbe und rote Karten haben und hat eine bestimmte Anzahl an Toren geschossen.
Fügen Sie Ihrer Klasse einen geeigneten Konstruktor hinzu, um Teamzugehörigkeit und Spielernamen zu initialisieren.
- (b) Folgende Methoden werden von der **Soccerplayer** Klasse implementiert:
- `public void redCard()` setzt `redCard` (Der Spieler wird vom Platz gestellt.)
 - `public void yellowCard()` setzt `yellowCard` (Der Spieler hat eine gelbe Karte bekommen.)
 - `public int getNumber()` gibt die Spielernummer zurück.
 - `public String getName()` gibt den Spielernamen zurück.
 - `public String getTeam()` gibt das zugehörige Team zurück.
 - `public void increaseGoalCount()` Der Spieler hat ein Tor geschossen, erhöhe `numberOfGoals`.
 - `public void setTeam(String newTeam)` Der Spieler hat das Team gewechselt und spielt jetzt für `newTeam`.
 - `public String toString()` Diese Methode wird beim Aufruf durch z.B. `System.out.println(soccerPlayer)` aufgerufen und gibt eine textuelle Repräsentation des **soccerPlayer** Objektes zurück. Der Rückgabe String soll den Spielernamen, dessen Nummer, das Team und die Anzahl der vom Spieler geschossenen Tore enthalten.
- (c) Ein Team **SoccerTeam** hat einen Teamnamen, einen Trainer (komplexer Datentyp), eine Tabellenposition (als Nummer). Zusätzlich wird die Anzahl der geschossenen Tore und die Anzahl der Gegentore, die das Team in einem Spiel erhalten hat, gespeichert. Die Spieler werden in einem Array gespeichert, das 11 Spieler aufnimmt. Der Konstruktor des **SoccerTeam** Objektes soll den Teamnamen, den Trainer und die Tabellenposition initialisieren.
- (d) Folgende Methoden sollen durch die **SoccerTeam** Klasse implementiert werden:
- `public void addPlayer(String name)` fügt einen Spieler mit dem Namen `name` zum Team hinzu. Die Methode erzeugt ein neues Spieler Objekt und fügt es in das Spieler array `players` an der ersten freien Position(*i*) ein. Der Spieler erhält die Spielernummer *i* + 1. Achten Sie darauf, evtl schon eingefügt Spieler nicht zu überschreiben. Ist das Array voll, so soll eine Fehlermeldung ausgegeben werden.
 - `public void goal(int playerNumber)` Der Spieler mit Nummer `playerNumber` hat ein Tor geschossen. Erhöhe sowohl im Spieler die Anzahl an Toren als auch im Team Objekt (`goals`). Im Spieler Array steht der Spieler an der Stelle `playerNumber - 1`.
 - `public void goalAgainst()` Der Gegner hat ein Tor geschossen, erhöhe `goalAgainst`.
 - `public String ratioString()` gibt den Spielstand als String `goals:goalsAgainst` (`goals` die Anzahl der geschossenen Tore, `goalsAgainst` Anzahl der Gegentore) zurück (z.B. 3:2).
 - `public String toString()` String Repräsentation mit Teamname, Trainernamen, Anzahl der geschossenen und erhalten Tore, den Ratio String, und zeilenweise die Spieler.

Aufgabe 4 Zementlager (4 Punkte)

Implementieren Sie eine Gewichts- und Mengenkontrolle für ein Zementlager. Das Lager kann max. 40.000 kg in max. 1000 Säcken Zement fassen. Die Säcke haben unterschiedliches Gewicht. Die einzelne Speicherung von Säcken ist nicht notwendig.

Schreiben Sie eine Java-Klasse **CementStorage** die folgende 3 Methoden realisiert. Achten Sie dabei auf angemessene Sichtbarkeit der enthaltenen Elemente.

- `int freeAmount()`: Die Funktion liefert die maximale Anzahl an freien Abstellplätzen für je einen Sack im Lager.
- `int freeWeight()`: Die Funktion liefert das maximal noch zu lagernde Gewicht.
- `boolean stockCement(int[] bags)`: Das Array `bags` enthält die Gewichte der Säcke, die eingelagert werden sollen. Die Größe des Arrays entspricht der Anzahl der gelieferten Säcke. Die Funktion liefert `true`, wenn die Lagerung erfolgreich war, sonst `false`.