CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

FACHHOCHSCHUL-AKADEMISCHER LEHRGANG „SOFTWARE ENGINEERING“

KLAUSUR AUS DER LEHRVERANSTALTUNG „PROGRAMMIEREN 1“

Silke Jandl, Harris Gerzic

Hauptklausur am 12.04.2023

1. Termin

1. Semester

SS 2023

Name: _____

Beurteilung: _____

Klausurdauer: 150 Minuten
Erlaubte Unterlagen: Alles außer Kommunikation jeglicher Art.

INHALTSVERZEICHNIS

0 EINLEITUNG / INTRODUCTION

The English Version of this document starts on page 12

Sie dürfen alle Ihre Unterlagen, alte Programme und Internetquellen verwenden (bis auf KI). Kommunikation oder Austausch in jedweder Art (analog oder digital) ist nicht erlaubt. Zusätzlich ist das Nutzen jeglicher KI wie zB. Chat-GPT, während der Prüfung strengstens verboten!

Über Moodle haben Sie Zugriff auf ihr Startpaket. Bitte geben Sie am Ende der Zeit Ihr Projektverzeichnis als ZIP-Datei über Moodle rechtzeitig ab.

Nur für kompilierbaren, dokumentierten und funktionsfähigen Code erhalten Sie Punkte! Beschreiben Sie Ihren Code mit kurzen aussagekräftigen Kommentaren (Bitte keine Romane).

Halten Sie sich an die Angaben: Keine Veränderung von vorgegebenen Methodennamen oder der Struktur, wie zB. Textausgabe in die Konsole statt einem return-Wert, usw.

Notenaufteilung

Es gibt bei dieser Prüfung insgesamt 60 Punkte zu erreichen, sie benötigen mindestens 50% der Punkte, um diese Lehrveranstaltung positiv zu absolvieren.

Der Notenschlüssel sieht wie folgt aus:

Note	Erreichte Prozent
1	90 %
2	80 %
3	65 %
4	50 %
5	0 – 49,99 %

1 ANGABE 01: STRINGVERARBEITER [8P]

Schreiben Sie eine Klasse **StringVerarbeiter**, welche gleichzeitig auch die Hauptklasse ist und alle Return-Werte am Ende des Programms in die Konsole ausgibt. Die Klasse soll mit den nachfolgenden Eigenschaften und Funktionen bestückt werden:

1.1 Klassen und Methoden [1,5P pro Methode]

Die Methode **lieferLaenge(String wort)** → eine Methode, die einen String als Übergabeparameter hat und die Länge des übergebenen Strings berechnet. Die Methode soll die Länge von dem, im Methodenparameter übergebenem String **wort** zurückgeben.

Die Methode **liefereZeichhen(String wort, int zahl)** → eine Methode, die einen String und eine ganze Zahl als Übergabeparameter hat und das Zeichen an der Stelle **zahl** im String **wort** berechnet bzw. findet. Die Methode soll das Zeichen an der Stelle **zahl** von **wort** zurückgeben.

Die Methode **ersetzeAlle(String wort, char alt, char neu)** → eine Methode, die einen String und zwei Zeichen (alt und neu) als Übergabeparameter hat und alle Vorkommen von dem Zeichen **alt** in String **wort** durch das Zeichen **neu** ersetzt. Die Methode soll den modifizierten String **wort** zurückgeben. *(**Tipp**: Google hilft hier zu einer schnellen Lösung 😉)*

Die Methode **stringVergleicher(String wort1, String wort2)** → eine Methode, die einfach nur zwei Strings miteinander vergleicht. Beide Strings werden vom Benutzer durch die Konsole eingegeben. Die Methode soll ein **true** zurückgeben, falls die zwei übergebenen Wörter gleich sind und ein **false**, falls die Wörter ungleich sind.

1.2 Beispiel Output [2P]

- Die Methode liefereLaenge("Hallo") sollte den Wert 5 zurückgeben
- Die Methode liefereZeichen("Hallo", 1) sollte den Wert 'a' zurückgeben
- Die Methode ersetzeAlle("Hallo", 'l', 'x') sollte den Wert "Haxxo" zurückgeben
- Die Methode stringVergleicher("Max","Max“) sollte den Wert **true** zurückgeben, bei den Parametern stringVergleicher(„Max“,“Nix“) sollte der Wert **false** zurückgegeben werden.

Alle Return-Werte sollen über die main-Methode auch in der Konsole ausgegeben werden!

```
Methode lieferLaenge: 5
Methode lieferZeichen: l
Methode ersetzeAlle: Haxxo

Bitte erstes Wort eingeben: Max
Bitte zweites Wort eingeben: Max
Methode stringVergleicher: true

Process finished with exit code 0
```

2 ANGABE 02: PERSONENVERWALTUNG [10P]

Schreiben Sie eine Klasse **PersonenVerwaltung,** eine Klasse **Person** und eine Klasse **PersonTest.** Die Klassen sollen mit den nachfolgenden Eigenschaften und Funktionen bestückt werden:

2.1 Klassen und Methoden

Die Klasse **Person** bildet ganz im Sinne der OOP eine reale Person im Programm ab und hat neben einem ***Konstruktor*** und allen ***Getter- & Setter***-Methoden auch noch folgende, nur in dieser Klasse sichtbaren, Attribute: **[2P]**

- ganzzahlige **id** (zur eindeutigen Identifizierung) & String **name** (nur Vorname)

Nachfolgende Variablen müssen weder im Konstruktor sein noch initialisiert werden, hier reicht mir eine Deklaration und ich möchte leidglich alle Getter- und Setter-Methoden:

- int alter, double groesse, int nachname, String geburtsland, String zweitName, String haarfarbe, boolean hatGlatze, float schuhgroesse, boolen hatBeziehung

Die Klasse **PersonenVerwaltung** erzeugt ein Array des Typs Person und hat folgende Funktionen:

- Die Methode **fuegePersonHinzu(Person pers)** → eine Methode, die ein Person-Objekt **pers** als Übergabeparameter hat und dieses Objekt in ein bereits bestehendes Array von Personen hinzufügt. **[2P]**

- Die Methode **liefere**PersonMitId**(int id)** → eine Methode, die eine ganze Zahl **id** als Übergabeparameter hat und die Person mit der angegebenen **id** in dem, bereits bestehendem, Array von Personen zurückgibt. Wenn keine Person mit der angegebenen **id** gefunden wird, soll die Methode null zurückgeben. **[2P]**

- Die Methode **entferne**PersonMitid**(int id)** → eine Methode, die eine ganze Zahl **id** als Übergabeparameter hat und die Person mit der angegebenen **id** aus dem Array von Personen entfernt. Wenn keine Person mit der angegebenen **id** gefunden wird, soll die Methode nichts tun. (Kein Rückgabewert) **[2P]**

Die Klasse **PersonTest** soll ein Objekt des Typs ***PersoneVerwaltung*** erzeugen und alle drei zuvor erstellten Methoden testen.

2.2 Beispiel Output [2P]

Zusätzlich soll ein Konsolen-Menü in der Klasse **PersonTest** aufgebaut werden, sodass der Nutzer zwischen den drei Optionen (Hinzufügen, Anzeigen und Entfernen) wählen kann, siehe dafür den Beispiel Output auf der nächsten Seite.

Dieses Menü soll so lange angezeigt werden, bis der Nutzer das Programm selbständig über eine der Menüoptionen beendet.

```
---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 1
Name der Person eingeben: Maxl
fuegePersonHinzu() -> Neue Person Maxl mit ID: 1 wurde hinzugefuegt!

---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 2
Welche Person soll ausgegeben werden, geben Sie bitte die ID an: 1
Person heisstt: Maxl

---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 3
Welche Person soll entfernt werden, geben Sie bitte die ID an: 1

entfernePersonMitId() -> Person wird nun entfernt.

---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 4

Programm wird Beendet - Ciao

Process finished with exit code 0
```

3 ANGABE 03: TODOLISTE [12P]

Schreiben Sie ein Programm, welches eine simple To-Do-Liste verwaltet. Die Anwendung soll es Benutzern ermöglichen, Aufgaben hinzuzufügen, anzuzeigen, zu markieren und zu löschen. Jede Aufgabe soll eine eindeutige ID, einen Titel und einen Status („**true**“ oder „**false**“) haben.

3.1 Klassen und Methoden

Die Klasse **Task** bildet eine Aufgabe in der ToDo-Liste ab und hat neben einem ***Konstruktor*** und allen ***Getter- & Setter***-Methoden auch noch folgende, nur in dieser Klasse sichtbaren, Attribute: ganzzahlige **id** **–&–** String **title** **–&–** boolean **isComplete** **[1P]**

Die Klasse **TodoListenVerwaltung** erzeugt ein Array des Typs Task und hat folgende Funktionen:

- Die Methode **addTask(String title)** → eine Methode, die einen neuen Task mit einem **title** als Übergabeparameter in ein bereits bestehendes Array namens ***tasks*** hinzufügt. **[2P]**

- Die Methode **getTask(int id)** → eine Methode, die eine ganze Zahl **id** als Übergabeparameter hat und den Task mit der angegebenen **id** in dem, bereits bestehendem, Array ***tasks*** zurückgibt. Wenn kein Task mit der angegebenen **id** gefunden wird, soll die Methode null zurückgeben. **[2P]**

- Die Methode **markTaskComplete(int id)** → eine Methode, die den Status des ausgewählten Tasks auf **true** setzt. **[2P]**

- Die Methode **deleteTask(int id)** → eine Methode, die eine ganze Zahl **id** als Übergabeparameter hat und den Task mit der angegebenen **id** aus dem Array ***tasks*** entfernt. Wenn kein Task mit der angegebenen **id** gefunden wird, soll die Methode nichts tun. (Kein Rückgabewert) **[2P]**

- Die Methode **displayTasks()** → eine Methode, die alle aktuellen Tasks in die Konsole ausgibt. Dabei soll die erste Zeile als eine Art Überschrift immer aus „ID – Titel – Status“ bestehen. Erst dann sollen Zeile für Zeile, Task für Task angezeigt werden (Siehe Beispiel Output, weiter unten). **[2P]**
  - ***Zusatzschritt***: Für die Spalte Status soll bei **true** als Text ein „***Erledigt***“ und bei **false** ein „***Offen***“ in der Konsole ausgegeben werden.

Die Klasse **TestTodo** soll ein Objekt des Typs ***TodoListenVerwaltung*** erzeugen und alle zuvor erstellten Methoden dieser Klasse Schritt für Schritt testen.

Achtung: Für diese Übung ist kein eigenes Menü erforderlich, es reicht, wenn Sie die Methodenaufrufe in der Klasse **TestTodo** hartcodiert hinschreiben und Ihr Konsolenoutput der Ausgabe auf der Folgeseite ähnelt.

3.2 Beispiel Output [1P]

```
addTask() -> Die Tasks Einkaufen, Ueben und Lernen einzeln erstellen.
displayTask() -> Ausgabe aller Tasks.

ID  Title       Status
0   Einkaufen   Offen
1   PR1 Ueben   Offen
2   Lernen      Offen

markTaskComplete() -> Task mit ID 1 als Erledigt markieren
displayTasks() -> Ausgabe aller Tasks.

ID  Title       Status
0   Einkaufen   Offen
1   PR1 Ueben   Erledigt
2   Lernen      Offen

deleteTask() -> Den Task mit der ID 2 loeschen.
displayTasks() -> Ausgabe aller Tasks.

ID  Title       Status
0   Einkaufen   Offen
1   PR1 Ueben   Erledigt

Process finished with exit code 0
```

4 ANGABE 04: SCHIFFE VERSENKEN [25P]

Das Spiel „Schiffe versenken“ ist ein beliebtes Kinderspiel, welches mit Bleistift und Papier durchgeführt werden kann. Es wird dabei ein Spielfeld („map“) aufgebaut.

Wenn auf einem Feld ein Schiff eines Spielers steht, dann ist dort für Spieler ‚A‘ eine positive Zahl, für Spieler ‚B‘ eine negative Zahl eingetragen. Steht kein Schiff eines Spielers, dann ist der Wert 0 eingetragen. In dieser Variante des Spiels sagt der Wert die Stärke des Schiffs aus. Ein Schiff der Stärke 4 müsste beispielsweise 4-mal getroffen werden, um unterzugehen.

Das Spielfeld sieht wie folgt aus:

```
1   0   0   2   0   0   1   1   0   0
3   0   -1  0   0   0   2   0   -1  0
0   3   0   0   -1  0   0   2   0   0
0   0   0   4   4   0   1   0   0   2
0   -3  2   0   0   0   -1  0   0   0
0   0   -2  0   0   -4  0   0   -1  0
0   0   0   0   -3  0   0   0   0   0
0   0   -2  0   -2  0   0   -2  0   0
```

4.1 Klassen und Methoden

Gegeben ist eine Klasse **BattleShipGame**, welche das oben dargestellte 2-dimensionale Array aufbaut. Dieses wird im Attribut **int[][] map** gespeichert. Gegeben ist auch eine Methode **displayField()**, welche das Array, wie oben dargestellt auf der Konsole ausgibt. Dazu gibt es auch eine Klasse **TestGame**, welche ein Objekt von **BattleShipGame** erzeugt, und die Methode **displayField()** aufruft. Zu implementieren sind folgende Methoden:

0. Die Methode **public int battleShipStrength(char player)** → Diese Methode übernimmt einen Spieler, und soll je Spieler die Gesamtstärke der Flotte berechnen. Bei Spieler ‘A‘ müssen die positiven Zahlen summiert werden, bei Spieler ‘B‘ die negativen Zahlen zusammengezählt. **Achtung**: Bei Spieler ‘B‘ soll aber trotzdem eine positive Zahl zurückgeliefert werden. **[6P]**

1. Die Methode **public int countBattleShips()** → Diese Methode soll die Gesamtanzahl aller Schiffe in der Map-Landkarte berechnen und retour liefern. **[6P]**

2. Die Methode **public int[] countOfShipsPerLine()** → Diese Methode soll je Zeile die Anzahl der Schiffe berechnen und dann ein 1-dimensionales Array retour liefern. Index 0, dieses Arrays, entspricht dabei der Zeile 0, Index 1 der Zeile 2 usw. **[6P]**
    a. Erwartetes Ergebnis: [3, 4, 6, 1, 5, 5, 3, 3]
    b. Also: Zeile 1 hat 3 Schiffe, Zeile 2 hat 4 Schiffe, usw.

3. Die Methode **public double averageShipStrengthPlayerA()** → Diese Methode soll für Spieler ‘A‘ die durchschnittliche Schiffstärke berechnen und als double-Wert an den Aufrufer retour liefern. **[6P]**

4.2 Beispiel Output [1P]

```
1   0   0   2   0   0   1   1   0   0
3   0   -1  0   0   0   2   0   -1  0
0   3   0   0   -1  0   0   2   0   0
0   0   0   4   4   0   1   0   0   2
0   -3  2   0   0   0   -1  0   0   0
0   0   -2  0   0   -4  0   0   -1  0
0   0   0   0   -3  0   0   0   0   0
0   0   -2  0   -2  0   0   -2  0   0

Methode 0:
Player A hat eine staerke von: 28
Player B hat eine staerke von: 23

Methode 1:
Es sind aktuell insgesamt 25 Schiffe auf der Map vorhanden.

Methode 2:
Schiffe pro Zeile (beginnend bei 1): [4, 4, 3, 4, 3, 3, 1, 3]

Methode 3:
A hat eine durchschnittliche Schiffstaerke von 2.0

Process finished with exit code 0
```

5 ANGABE05 BERECHNESUMME [5P]

Erstellen Sie eine neue Klasse mit einem Namen Ihrer Wahl. Schreiben Sie darin eine Methode **berechneSumme(int[] arr)** mit einem Integer-Array als Parameter, welche die Summe aller Zahlen im übergebenen Array berechnet und zurückgibt. Geben Sie den Return-Wert anschließend über die main-Methode in der Konsole aus.

Achtung: Verwenden Sie Rekursion, um diese Aufgabe zu lösen!

5.1 Beispiel Output

Zum Beispiel sollte die Methode **berechneSumme({1, 2, 3, 4})** den Wert 10 zurückgeben, da die Summe aller Zahlen im Array, also 1 + 2 + 3 + 4 = 10 ist.

```
Folgendes Array wird an die Methode berechneSumme() uebergeben: [1, 2, 3, 4]
Die Summe des uebergebenen Arrays lautet: 10

Process finished with exit code 0
```

English Version of this document

0 INTRODUCTION

You are allowed to use all your documents, old programs, and internet sources (except AI). Communication or exchange of any kind (analog or digital) is not allowed. In addition, the use of any AI, such as chat GPT, is strictly prohibited during the exam!

You will have access to your start package via Moodle. At the end of the time, please hand in your project directory as a ZIP file via Moodle in time.

You will only receive points for compilable, documented, and working code! Describe your code with short meaningful comments (no long text please).

Stick to the specifications: Do not change predefined method names or structure, e.g. text output to the console instead of a return value, etc.

Important: Please keep the variable and method names in German as given, because my automatic tests are aligned from it.

Grade distribution

There are a total of 60 points to be achieved on this exam, you need at least 50% of the points to pass this course.

The grade key is as follows:

Grade	Percent reached
1	90 %
2	80 %
3	65 %
4	50 %
5	0 – 49,99 %

1 EXAMPLE 01: STRINGPROCESSOR [8P]

Write a class **StringVerarbeiter**, which is also the main class and outputs all return values to the console at the end of the program. The class is to be equipped with the following properties and functions:

1.1.1 Classes and methods [1,5P per method]

- The method **lieferLaenge(String wort)** → is a method that has a string as a passing parameter and calculates the length of the passed string. The method should return the length of the string **wort** passed in the method parameter.

- The method **lieferZeichen(String word, int number)** → a method that has a string and an integer as passing parameters and calculates or finds the character at the position **number** in the string **word**. The method is to return the character at the position **number** of **wort**.

- Method **ersetzeAlle(String word, char old, char new)** → a method that has a string and two characters (old and new) as passing parameters and replaces all occurrences of the character **old** in string **word** with the character new. The method should return the modified string **word**. *(**Tip**: Google helps here to a quick solution 😉)*

- The **stringVergleicher(String wort1, String wort2)** → method is a method that simply compares two strings. Both strings are entered by the user through the console. The method should return a **true** if the two words passed are equal and a **false** if the words are not equal.

1.1.2 Sample Output [2P]

- The method lieferLaenge("Hello") should return the value 5
- The method lieferZeichen("Hello", 1) should return the value 'a'
- The method ersetzeAlle("Hello", 'l', 'x') should return the value "Haxxo
- The method stringVergleicher("Max", "Max") should return the value **true**, for the parameters stringVergleicher("Max", "Nix") should return the value **false**.

All return values should also be output to the console via the main method!

```
Methode lieferLaenge: 5
Methode lieferZeichen: l
Methode ersetzeAlle: Haxxo

Bitte erstes Wort eingeben: Max
Bitte zweites Wort eingeben: Max
Methode stringVergleicher: true

Process finished with exit code 0
```

2 EXAMPLE 02: PEOPLEMANAGEMENT [10P]

Write a **PersonenVerwaltung** class, a **Person** class, and a **PersonTest** class. The classes are to be populated with the following properties and functions:

2.1.1 Classes and methods

The class **Person** represents a real person in the program in the sense of the OOP and has beside a ***constructor*** and all ***Getter & Setter*** methods also the following, only in this class visible, attributes: **[2P]**

- integer **id** (for unique identification) & string **name** (first name only).

Following variables don't have to be in the constructor or initialized, here a declaration is enough, and I want all getter and setter methods: *(please keep the german wording here)*

- int alter, double groesse, int nachname, String geburtsland, String zweitName, String haarfarbe, boolean hatGlatze, float schuhgroesse, boolen hatBeziehung

The **PersonManagement** class creates an array of type Person and has the following functions:

- The **method fuegePersonHinzu(Person pers)** → a method that has a Person object **pers** as a passing parameter and adds this object to an already existing array of persons. **[2P]**

- The method **deliverPersonWithId(int id)** → a method that has an integer **id** as a passing parameter and returns the person with the given **id** in the, already existing, array of persons. If no person with the given **id** is found, the method shall return null. **[2P]**

- The method **removePersonMitid(int id)** → a method that has an integer **id** as a passing parameter and removes the person with the given **id** from the array of persons. If no person with the specified **id** is found, the method shall do nothing. (No return value) **[2P]**

The **PersonTest** class shall create an object of type ***PersonenVerwaltung*** and test all three methods created before.

2.1.2 Sample Output [2P]

In addition, a console menu shall be built in the **PersonTest** class so that the user can choose between the three options (Add, Show and Remove), see the sample output on the next page for this.

This menu is to be displayed until the user exits the program independently via one of the menu options.

```
---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 1
Name der Person eingeben: Maxl
fuegePersonHinzu() -> Neue Person Maxl mit ID: 1 wurde hinzugefuegt!

---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 2
Welche Person soll ausgegeben werden, geben Sie bitte die ID an: 1
Person heisstt: Maxl

---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 3
Welche Person soll entfernt werden, geben Sie bitte die ID an: 1

entfernePersonMitId() -> Person wird nun entfernt.

---Willkommen zu Aufgabe01---
1: neue Person hinzufuegen
2: Personendaten einer gewissen Person ausgeben
3: Eine gewisse Person entfernen
4: Exit -> Programm beenden

Bitte jetzt Zahl eingaben: 4

Programm wird Beendet - Ciao

Process finished with exit code 0
```

3 EXAMPLE 03: TODOLIST [12P]

Write a program that manages a simple to-do list. The application should allow users to add, view, mark and delete tasks. Each task should have a unique ID, a title and a status ("**true**" or "**false**").

3.1.1 Classes and methods

The class **Task** represents a task in the ToDo list and has besides a ***constructor*** and all ***getter & setter*** methods also the following attributes, visible only in this class: integer **id** -&- string **title** -&- boolean **isComplete** **[1P]**

The **TodoListenVerwaltung** class creates an array of type Task and has the following functions:

- Method **addTask(String title)** → a method that adds a new task with a title as a passing parameter into an already existing array named tasks. **[2P]**

- The method **getTask(int id)** → a method that has an integer **id** as a passing parameter and returns the task with the specified **id** in the, already existing, array called tasks. If no task with the specified **id** is found, the method shall return null. **[2P]**

- The method **markTaskComplete(int id)** → a method that sets the status of the selected task to **true**. **[2P]**

- The **deleteTask(int id)** → method a method that has an integer **id** as a passing parameter and removes the task with the specified **id** from the tasks array. If no task with the specified **id** is found, the method shall do nothing. (No return value) **[2P]**

- The method **displayTasks()** → a method that prints all current tasks to the console. The first line should always consist of "ID - Title - Status" as a kind of heading. Only then, line by line, task by task should be displayed (see example output, below). **[2P]**
    - ***Additional step***: For the Status column, the text "Completed" should be displayed in the console if true and "Open" if false.

The class **TestTodo** should create an object of the type **TodoListenVerwaltung** and test all previously created methods of this class step by step.

Attention: For this exercise no own menu is necessary, it is sufficient if you write down the method calls in the class **TestTodo** hardcoded and your console output is similar to the output on the following page.

3.1.2 Sample Output [1P]

```
addTask() -> Die Tasks Einkaufen, Ueben und Lernen einzeln erstellen.
displayTask() -> Ausgabe aller Tasks.

ID  Title       Status
0   Einkaufen   Offen
1   PR1 Ueben   Offen
2   Lernen      Offen

markTaskComplete() -> Task mit ID 1 als Erledigt markieren
displayTasks() -> Ausgabe aller Tasks.

ID  Title       Status
0   Einkaufen   Offen
1   PR1 Ueben   Erledigt
2   Lernen      Offen

deleteTask() -> Den Task mit der ID 2 loeschen.
displayTasks() -> Ausgabe aller Tasks.

ID  Title       Status
0   Einkaufen   Offen
1   PR1 Ueben   Erledigt

Process finished with exit code 0
```

4 EXAMPLE 04: SINK SHIPS [25P]

The game "Sink ships" is a popular children's game, which can be done with pencil and paper. A playing field ("map") is built up.

If there is a ship of a player on a field, then a positive number is entered there for player 'A', and a negative number for player 'B'. If there is no ship of a player, then the value 0 is entered. In this variant of the game, the value indicates the strength of the ship. For example, a ship of strength 4 would have to be hit 4 times to sink.

The game board looks like this:

```
1   0   0   2   0   0   1   1   0   0
3   0   -1  0   0   0   2   0   -1  0
0   3   0   0   -1  0   0   2   0   0
0   0   0   4   4   0   1   0   0   2
0   -3  2   0   0   0   -1  0   0   0
0   0   -2  0   0   -4  0   0   -1  0
0   0   0   0   -3  0   0   0   0   0
0   0   -2  0   -2  0   0   -2  0   0
```

4.1.1 Classes and methods

Given is a class **BattleShipGame**, which builds the 2-dimensional array shown above. This is stored in the **int[][] map** attribute. Given is also a method **displayField()**, which displays the array as shown above on the console. There is also a class **TestGame**, which creates an object of **BattleShipGame**, and calls the method **displayField()**. The following methods are to be implemented:

0. The method **public int battleShipStrength(char player)** → This method takes a player, and should calculate per player the total strength of the fleet. For player 'A' the positive numbers must be summed up, for player 'B' the negative numbers must be summed up. **Attention**: For player 'B' a positive number should be returned anyway. **[6P]**

1. the method **public int countBattleShips()** → This method shall calculate the total number of all ships in the map and return that number. **[6P]**

2. the method **public int[] countOfShipsPerLine()** → This method shall calculate per line the number of ships and then return a 1-dimensional array. Index 0, of this array, corresponds to line 0, index 1 to line 2, etc. **[6P]**
    a. Expected result: [3, 4, 6, 1, 5, 5, 3, 3]
    b. So: row 1 has 3 ships, row 2 has 4 ships, etc.

3. the method **public double averageShipStrengthPlayerA()** → This method should calculate the average ship strength for player 'A' and return it as a double value to the caller. **[6P]**

4.1.2 Sample Output [1P]

```
1   0   0   2   0   0   1   1   0   0
3   0   -1  0   0   0   2   0   -1  0
0   3   0   0   -1  0   0   2   0   0
0   0   0   4   4   0   1   0   0   2
0   -3  2   0   0   0   -1  0   0   0
0   0   -2  0   0   -4  0   0   -1  0
0   0   0   0   -3  0   0   0   0   0
0   0   -2  0   -2  0   0   -2  0   0

Methode 0:
Player A hat eine staerke von: 28
Player B hat eine staerke von: 23

Methode 1:
Es sind aktuell insgesamt 25 Schiffe auf der Map vorhanden.

Methode 2:
Schiffe pro Zeile (beginnend bei 1): [4, 4, 3, 4, 3, 3, 1, 3]

Methode 3:
A hat eine durchschnittliche Schiffstaerke von 2.0

Process finished with exit code 0
```

5 EXAMPLE 05: CALCULATIONSUM [5P]

Create a new class with a name of your choice. In it, write a method **berechneSumme(int[] arr)** with an integer array as a parameter, which calculates and returns the sum of all numbers in the array passed. Then output the return value to the console using the main method.

***Attention**: Use recursion to solve this task!*

5.1.1 Sample Output

```
Folgendes Array wird an die Methode berechneSumme() uebergeben: [1, 2, 3, 4]
Die Summe des uebergebenen Arrays lautet: 10

Process finished with exit code 0
```