1 Beispiel

1.1 Aufgabe

Lies die Datei „weapons.csv“ welche sich im Moodle befindet ein. Programmiere dazu eine passende Klasse mit geeigneten Objektvariablen, Methoden und Enums.

1.2 Aufgabe

Programmiere einen Comparator mit Hilfe von Lambdas. Sortiere die eingelesenen Objekte absteigend nach damage. Schreibe einen Unit-Test.

1.3 Aufgabe

Programmiere einen Comparator mit Hilfe von Lambdas. Sortiere die eingelesenen Objekte alphabetisch zuerst nach combatType, dann nach damageType und zuletzt nach name. Schreibe einen Unit-Test.

1.4 Aufgabe

Programmiere das funktionale Interface Printable mit der Methode:

```
void print(List<Weapon> weapons)
```

1.5 Aufgabe

Erstelle mit Hilfe von Lambdas ein neues Objekt vom Typ Printable, welches die übergebene Liste wie gewohnt auf der Konsole ausgibt. Rufe die Methode für die sortierte Liste auf.

1.6 Aufgabe

Erstelle mit Hilfe von Lambdas ein neues Objekt vom Typ Printable, welches die übergebene Liste als Tabelle ausgibt. Bestimme dabei zuerst, wie breit jede Spalte sein muss und nutze Leerzeichen, um Einträge in der Tabelle bei der Darstellung aufzufüllen. Verwende die Zeichen -, | und +, um Zeilen und Spalten voneinander zu trennen. Rufe die Methode für die sortierte Liste auf.

1.7 Aufgabe

Nutze für das Einlesen der Datei einen einzigen Stream mit Lambdas.

2 Beispiel

Im Moodle finden Sie ein Stundent_Stub Projekt welches die Methoden-Stubs enthält, welche Sie implementieren müssen.
Wichtig: Es gibt nur alle Punkte auf die Hausübung wenn alle Unit-Tests welche in dem Student Stub enthalten sind funktionieren!
Sie müssen Ihre Methoden in einer eigenen Main-Methode testen und mit den mitgelieferten Unit-Tests!

2.1 Aufgabe

Erstelle ein int-Array, welches 10000 Zufallszahlen zwischen 0 und 100 enthält.

2.2 Aufgabe

Programmiere die Methode double average(int[] numbers), welche den Mittelwert der übergebenen Zahlen berechnet. Verwende dafür einen einzigen Stream. Schreibe einen Unit-Test.

2.3 Aufgabe

Erstelle ein String-Array, welches 10 Zufalls-Strings mit einer Länge von 10 enthält.

2.4 Aufgabe

Programmiere die Methode List<String> upperCase(String[] strings), welche alle Buchstaben der übergebenen Strings in Großbuchstaben umwandelt und diese zurückgibt. Verwende dafür einen einzigen Stream. Überlege, wie man Code noch kompakter schreiben kann (Stichwort: Methodenreferenz). Schreibe einen Unit-Test.

2.5 Aufgabe

Lies die Datei „Weapons.csv“ aus dem Moodle ein. Programmiere dazu eine passende Klasse Weapon mit geeigneten Objektvariablen, Methoden und Enums.

2.6 Aufgabe

Programmiere eine Methode mit einem Stream, welche

- in den eingelesenen Objekten das findet, das den geringsten damage hat.
- in den eingelesenen Objekten das findet, das die höchste strength hat.
- in den eingelesenen Objekten alle findet, die den DamageType.MISSILE haben.
- in den eingelesenen Objekten das findet, das den längsten name hat.
- die eingelesenen Objekten in eine List<String> der names umwandelt.

- die eingelesenen Objekten in eine int[] der speeds umwandelt.
- die Summe aller values zurückgibt.
- die HashCodes aller Objekte in einen einzigen zusammenrechnet und diesen zurückgibt.
- die Liste ohne Duplikate zurückgibt.
- den value jedes Objektes um 10% erhöht und die Liste zurückgibt.

3 Beispiel

Formuliere die Bedingungen „Zahl ist gerade“, „Zahl ist positiv“ „Zahl ist Null“ und „Wert ist null“ als `Predicate<Integer>` und/oder als `IntPredicate` und prüfe diese mit verschiedenen Eingaben.

```
final Predicate<Integer> isEven = // ... TODO ...
final IntPredicate isPositive = // ... TODO ...
```

Formuliere die Bedingung „Wort kürzer als 4 Buchstaben"“ und prüfe das damit realisierte `Predicate<String>` `isShortWord`.
Kombiniere die Prädikate wie folgt und prüfe wieder:

- Zahl ist positiv und Zahl ist gerade (Nutze die Methode `and()` von Predicate)
- Zahl ist positiv und ungerade (Nutze einen Aufruf von `negate()` von Predicate¸)

4 Beispiel

Quadriere alle ungeraden Zahlen von 1 bis 10 und ermittle deren Summe:

```
final int result = IntStream.of(1,2,3,4,5,6,7,8,9,10).filter(...)
                                                      .map(...).
                                                      .reduce(0, ...);
```

5 Beispiel (Nicht notwendig für 4 Punkte - gibt Extrapunkte! ZUSATZAUFGABE)

Schreibe Functional Interfaces für die folgenden Abbildungen / Aufgaben

1. Bilde Objekte von Typ String auf int ab: StringToIntConverter
2. Bilde Objekte von Typ String auf String: StringToStringMapper

Gegeben sei folgendes rudimentäres Fragment eines Functional Interfaces als Ergänzung der obigen String-Mapper:

```
public interface Mapper<S, T>
{
// TODO: map S -> T
// TODO: mapAll List<S> -> List<T>
}
```

Vervollständige die Implementierung, sodass nachfolgendes Programm kompiliert:

```
public static void main(final String[] args)
{
    final List<String> names = Arrays.asList("Tim", "Andi", "Michael");
    final Mapper<String, Integer> intMapper = String::length;
    System.out.println(intMapper.mapAll(names));
    final Mapper<String, String> stringMapper = str -> ">> " +
                                                 str.toUpperCase() + " << ";
    final List<String> uppercaseNames = stringMapper.mapAll(names);
    System.out.println(uppercaseNames);
}
```

Tipp: Stelle die mapAll()-Funktionalität mithilfe einer Defaultmethode (`http://www.gridtec.at/java-8-default-methoden/`) bereit.
Recherchiere zur Defaultmethode. Was ist von Defaultmethoden in Applikationsklassen zu halten? Welche Vorteile bringen diese, welche Fallstricke sollte man bedenken? Erkläre dies bei der Hausübungskontrolle!