# PLF 4BKIF 9. April 2025

## Themen Collections, Aggregatsfunktionen, Lambda, Stream, Exception

Der Code der Testklasse darf nicht verändert werden.

Öffnen Sie das Projekt auf Laufwerk ”Z:” und lösen Sie dort die Aufgaben.

Ziel der Arbeit ist es, alle Testfälle auf ”passed” (grün, bestanden) zu bekommen, indem Code in ”main/java” gemäß der Angabe editiert wird.

## Erklärungen zu den einzelnen Testfällen

`aufstellen` **(leicht, 3P)**

BITTE dies zuerst machen, da die meisten anderen Tests hiervon abhängen!

`entfernen` **(leicht, 4P)**

Auch dies bitte eher bald machen, andere Tests hängen hiervon ab!

`entfernenAlle` **(mittel, 5P)**

Tipp: `List.removeIf(Predicate)`, welches mit Lambda ausgedrückt werden kann

`getMaxAnzahl` **(superleicht, 2P)**

`maxWartungsIntervall` **(leicht-mittel, 5P)**

`sortierenNachName` **(recht leicht, 5P)**

Wird erst klappen, wenn das aufstellen funktioniert.

`berechneAvgPreisMaschinen` **(mittel, 5P)**

Wird erst klappen, wenn das aufstellen funktioniert.

`getMaxPreisEur` **(leicht, 4P)**

Wird erst klappen, wenn das aufstellen funktioniert.

`getGroupedMachines` **(anspruchsvoll, 7P)**

Ziel ist es hier, in einer Map eine Liste mit allen Beinpressen und eine zweite Liste mit allen Ergometern zu erhalten. Siehe den Test-Code! Tipp: Collectors.groupingby

## Punkteschlüssel

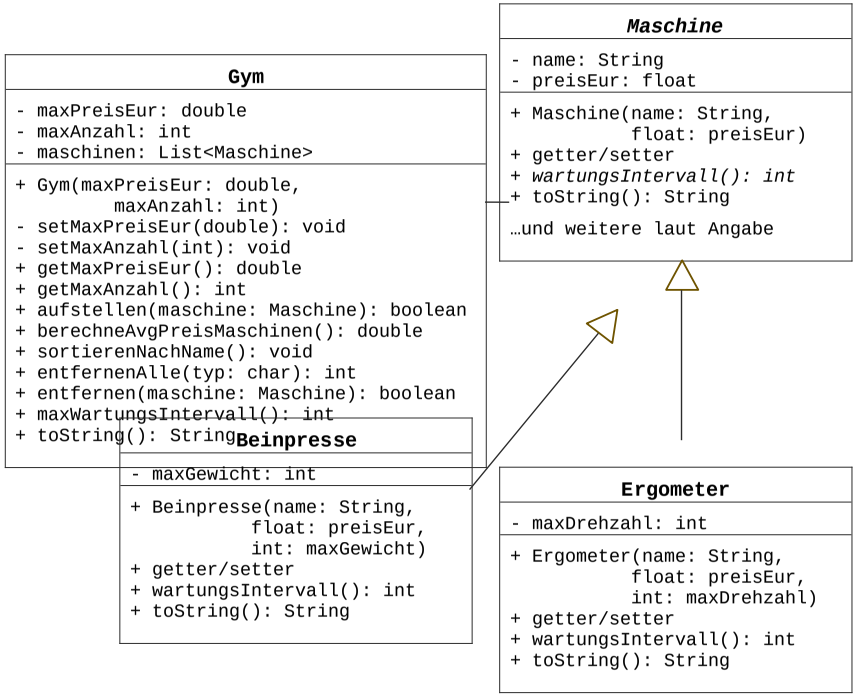
0-20 / 21-25 / 26-30 / 31-35 / 36-40

`Gutes Gelingen!`

# ÜBUNG GYM

Klassen implementieren mit der IDE IntelliJ Community Edition mit Java laut UML-Klassendiagramm

Für ein Gym (in Deutsch als Fitnesscenter bekannt) sollen dessen Trainingsgeräte (von den Auftraggebern als Maschinen bezeichnet) in einer Java-Applikation verwaltet werden. Halten Sie sich bei der Implementierung an die Vorgaben im folgenden UML-Klassendiagramm.




Hinweis: achten Sie bei den beiden set-Methoden der Klasse Gym auf den Access-Modifier private

Verwenden Sie **durchgängiges Exception Handling** mit der eigenen **Exception-Klasse GymException**.

## 1) Implementieren Sie die abstrakte Klasse Maschine

### 1.1) Attribute

Implementieren Sie die Attribute gemäß den Angaben im UML-Klassendiagramm.

### 1.2) Konstruktor

o) der Konstruktor übernimmt die Werte für Name und Preis in EUR der Maschine

### 1.3) get-Methoden

Implementieren Sie alle get-Methoden

### 1.4) set-Methoden

Implementieren Sie alle set-Methoden und berücksichtigen Sie die folgenden Anforderungen:
o) Der Name darf nicht leer sein (und natürlich auch nicht null).
o) Der Preis in EUR (Attribut preisEur) muss positiv sein. Ein Preis von 0 EUR ist erlaubt.

### 1.5) *+wartungsintervall(): int*

Diese Methode ist abstrakt und muss zwingend in den Subklassen implementiert werden.

### 1.6) +toString(): String

Soll mit einem erklärenden Text alle Informationen über das Objekt zurückgeben, Vorgabe-Beispiele:
„Smartrider, 2500.0 EUR“ oder
„Arnies, 8000.0 EUR“

## 2) Implementieren Sie die Subklassen Ergometer und Beinpresse

Die Superklasse beider Klassen (Ergometer und Beinpresse) ist die Klasse Maschine.

### 2.1) Attribute

Implementieren Sie die Attribute beider Klassen gemäß den Angaben im UML-Klassendiagramm.

### 2.2) get-Methoden

Implementieren Sie alle get-Methoden

### 2.3) set-Methoden

Implementieren Sie alle set-Methoden und berücksichtigen Sie die folgenden Anforderungen:
o) Der Wert von maxDrehzahl muss zwischen 5 und 150 liegen (jeweils inklusive).
o) Der Wert von maxGewicht muss zwischen 100 und 500 liegen (jeweils inklusive).

### 2.4) +wartungsintervall(): int

Das Wartungsintervall beträgt bei jeder Ergometer-Maschine 12 Monate, bei jeder Beinpresse-Maschine beträgt das Wartungsintervall 6 Monate.

### 2.5) +toString(): String

Soll mit einem erklärenden Text alle Informationen über das Objekt zurückgeben, Vorgabe-Beispiele:
„Ergometer: Smartrider, 2500.0 EUR, Wartungsintervall: 12 Monate, max. Drehzahl 150“ oder
„Beinpresse: Arnies, 8000.0 EUR, Wartungsintervall: 6 Monate, max. Gewicht: 500“

## 3) Implementieren Sie die Klasse Gym

### 3.1) Attribute

Implementieren Sie die Attribute gemäß den Angaben im UML-Klassendiagramm.

### 3.2) Konstruktor

o) übernimmt den Maximalpreis in EUR für eine Maschine (Attribut maxPreisEur) und die maximale Anzahl an Maschinen (Attribut maxAnzahl), die im Gym aufgestellt werden dürfen
o) erzeugt eine von Ihnen für diese Anforderung gewählte List vom Typ Maschine
o) ein neu erstelltes Gym-Objekt hat keine Maschinen

### 3.3) set-Methoden

`-setMaxPreisEur(double): void`
o) setzt den Maximalpreis in EUR für eine aufzustellende Maschine
o) der Maximalpreis für eine Maschine darf höchstens 9000.0 EUR betragen (und darf nicht negativ sein)

`-setMaxAnzahl(int): void`
o) setzt die maximal erlaubte Anzahl an Maschinen, die im Gym aufgestellt werden dürfen
o) gültige Werte von maxAnzahl liegen zwischen 5 und 100 (jeweils inklusive)

### 3.4) get-Methoden

Implementieren Sie die get-Methoden für die Attribute maxPreisEur und maxAnzahl.

### 3.5) +aufstellen(maschine: Maschine): boolean

o) fügt die übergebene gültige Objektreferenz in der List maschinen ein und liefert true zurück
o) Wenn der Preis der Maschine größer ist als der Maximalpreis in EUR, dann darf die Maschine nicht im Gym aufgestellt werden. Die Methode liefert in diesem Fall false zurück.
o) Es dürfen nur so viele Maschinen aufgestellt werden, wie im Attribut maxAnzahl vorgesehen sind. Wenn die maximale Anzahl an Maschinen durch die übergebene Maschine überschritten werden würde, dann liefert die Methode false zurück.
o) Eine Maschine-Referenz darf nur einmal hinzugefügt werden (keine doppelten Maschinen erlaubt). Wenn eine Maschine doppelt aufgestellt werden sollte, dann wird KEINE Maschine aufgestellt und die Methode liefert false zurück.
Tipp: implementieren Sie in der Klasse Maschine die Methoden equals() und hashCode() .



### 3.6) +berechneAvgPreisMaschinen(): double

Liefert den Durchschnittspreis aller im Gym aufgestellten Maschinen zurück. Ist das Gym leer, dann wird 0.0 zurückgeliefert. Tipp: erstellen Sie selbstständig eine Hilfsmethode -berechnePreisMaschinen(): float

### 3.7) +sortierenNachName(): void

Sortiert die Maschine-Referenzen in der List maschinen nach dem Attribut name alphabetisch. Erweitern Sie die Implementierung entsprechend. Verwenden Sie einen Comparator mit der Methode compare(...)

### 3.8) +entfernenAlle(typ: char): int

Entfernt alle Maschine-Referenzen mit dem übergebenen Typ aus der List maschinen und liefert die Anzahl der entfernten Maschine-Referenzen zurück. Wenn der Typ ‚b' ist, dann werden alle Beinpressen entfernt. Wenn der typ ‚e' ist, dann werden alle Ergometer entfernt. Die Groß- und Kleinschreibung beim übergebenen Parameter ist egal. Ist der übergebene Typ ungültig oder wenn das Gym leer ist, dann wird -99 zurückgeliefert.

### 3.9) +entfernen(maschine: Maschine): boolean

Die übergebene Maschine-Referenz wird aus der List maschinen entfernt und es wird true zurückgeliefert. Ist das Gym leer oder wird die übergebene Maschine-Referenz nicht gefunden, dann wird false zurückgeliefert. Achten Sie auf einen inhaltlichen Vergleich bei der Suche (nach name und preisEur). Tipps: contains() und equals()

### 3.10) +maxWartungsintervall(): int

Ermittelt unter allen Maschinen im Gym den größten Wert beim Wartungsintervall und liefert diesen zurück. Ist das Gym leer, dann wird -99 zurückgeliefert.

### 3.11) +toString(): String

Gibt mit einem kurzen erklärenden Text alle Informationen über das Gym-Objekt mit den einzelnen Maschine-Objekten zurück.

## 4) Implementieren Sie die JUnit Testklasse GymTest

Implementieren Sie in der Testklasse die folgenden Testmethoden:

`+testAufstellen_sollFunktionieren_neuesErgometer_returnsTrue()`

`+testAufstellen_sollNichtFunktionieren_null_returnsFalse()`

`+testEntfernenmaschine_sollFunktionieren_beinpresseNichtImGym_returnsFalse()`

`+testEntfernenMaschine_sollFunktionieren_beinpresseImGym_returnsTrue()`

GUTES GELINGEN UND VIEL ERFOLG!