CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

**Fachhochschul-Studiengang**
**„Wirtschaftsinformatik“**

**GRUNDLAGEN SOFTWAREENTWICKLUNG**

Daniel Kandlhofer, MSc;
DI Stephan Kochauf

Hauptklausur am 16.12.2022
1.Termin

**1. SEMESTER**

**WS 2022/23**

**Jahrgang WI 22**

Name: _____

Beurteilung: _____

Klausurdauer: **120 Minuten**
Erlaubte Unterlagen: **keine**

BITTE AUF JEDEM BLATT DEUTLICH IHREN NAMEN VERMERKEN!
NICHT MIT BLEISTIFT SCHREIBEN!

# Vorbereitungen

1. Sie finden im Moodle die Angabe, sowie ein StarterKit für die Prüfung
2. Laden Sie das StarterKit herunter, entzippen Sie dieses und öffnen Sie den Ordner mit IntelliJ
3. Benennen Sie im IntelliJ das Projekt auf Ihren Nachnamen_Vorname um
4. Die Prüfungsaufsicht behält es sich vor, Ihnen eine unselbstständige Lösung der Aufgabenstellung zu unterstellen, falls Sie das Netzwerkkabel nicht trennen, was die sofortige Abgabe der Arbeit sowie eine negative Bewertung nach sich zieht.

# WICHTIG

Das Verwenden von nicht erlaubten Unterlagen, die Online-Schaltung des Prüfungsrechners sowie der Einsatz von Datenträgern jedweder Art (CDs, USB-Sticks, o.ä.) führt zur sofortigen Abgabe und laut Prüfungsordnung zu einer negativen Beurteilung. Erlaubt sind eine geöffnete Version von IntelliJ oder Moodle. Weitere Unterlagen oder Webseiten sind NICHT erlaubt.

Achten Sie darauf, dass die von Ihnen erzeugten Codestellen **kompilieren**, da nur für funktionierende Methoden **Punkte** vergeben werden.

Achten Sie auch darauf, dass Ihre **Methoden die geforderten Werte** nicht nur berechnen, sondern auch **zurückliefern**, da zur Korrektur die erwarteten Ergebnisse den tatsächlichen Ergebnissen gegenübergestellt werden. Achten Sie auf sinnvolle und sprechende Commit-Messages.

**Anmerkung zur Aufgabenstellung:**
Weiterführende Tipps / Workarounds sind grau hinterlegt. Diesen kann, muss aber nicht gefolgt werden.

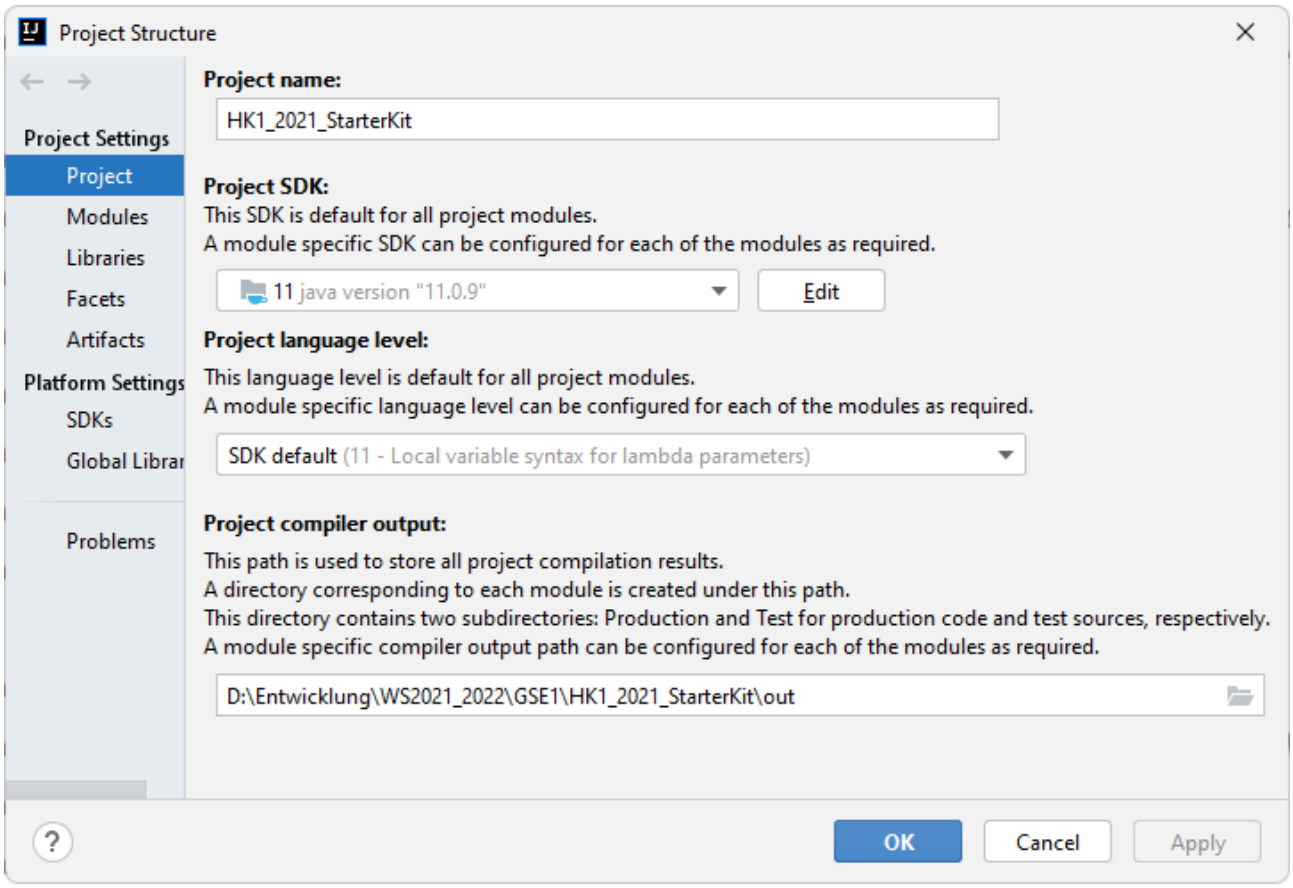
# Abgabe

Die Abgabe erfolgt als .zip-Datei Ihres lokalen Projektes auf Moodle (vorname.nachname.zip). Sie können die .zip-Datei direkt im IntelliJ über File=>Export=>Project to zip-File erstellen.

**Viel Erfolg!**

Das StarterKit ist für Java JDK 11 konfiguriert. Eventuell müssen Sie dieses auf Ihre lokale JDK anpassen:

File=>Project Structure:



**Beispiel 1: Arrays [Gesamtpunkte: 15 Punkte]**
Erzeugen Sie im Package *arrays* eine neue **Klasse** namens **ArrayHelper**. Mit dem Array soll ein Lotto-Tipp abgebildet werden. Der Lotto-Tipp soll dabei universell funktionieren und der Benutzer soll die Anzahl der Stellen (sizeOfArray), sowie die höchstmögliche Zahl festlegen (maxValue).

Legen Sie in dieser Klasse **eine statische Methode** mit folgender Signatur an:

```
public static int[] createIntArray(int sizeOfArray, int maxValue) {
	//TODO: your code here…
}
```

[5 Punkte] Die Methode "createIntArray" soll ein int-Array von der Größe „sizeOfArray" erzeugen und iterativ mit Zufallswerten zwischen 1 (inklusive) und maxValue (inklusive) befüllen. Sie können dazu folgenden Code verwenden: *new Random().nextInt(maxValue + 1)*. Geben Sie anschließend das befüllte Array als Rückgabewert retour.

Beispiel:

| Input | | Output |
|---|---|---|
| 6, 45 | => | [8, 3, 45, 30, 28, 25] |

[3 Punkte] Erstellen Sie innerhalb dieser Klasse die main-Methode und rufen Sie „createIntArray" mit Werten auf. Geben Sie alle Zahlen wie folgt auf der Kommandozeile aus:

*„Die Glückszahl an der Stelle <i> lautet: <nummer>"*

Ersetzen Sie <i> durch den Index (bzw. durch die Stelle, an der die Zahl gezogen wurde) und <nummer> durch die Zahl. Sollten Sie Probleme mit der Erstellung des Arrays haben, verwenden Sie ein selbst definiertes Array und geben Sie dieses wie oben erwähnt aus. Achten Sie hierbei auf die folgende Ausgabe. Es genau die Ausgabe wie folgt (grau markiert) kommen.

Beispiel:

| Array | | Konsolen Output |
|---|---|---|
| [77, 5, 90] | => | Die Nummer an der Stelle 1 lautet: 77 |
| | | Die Nummer an der Stelle 2 lautet: 5 |
| | | Die Nummer an der Stelle 3 lautet: 90 |

[7 Punkte] Die Methode "validateTipp" soll einen Lottotipp auf Korrektheit überprüfen. Ein Lottotipp ist dann korrekt, wenn alle Zahlen eindeutig sind, sprich Zahlen nicht doppelt vorkommen.

```
public static boolean validateTipp(int[] array) {
	//TODO: your code here…
}
```

**Tipp:** Überprüfen Sie für jede Zahl im Array, ob Sie im nachfolgenden / verbleibenden Array die Zahl nochmals finden. Wenn Sie die Zahl nochmals finden können Sie die Methode sofort mit ***false*** beenden.

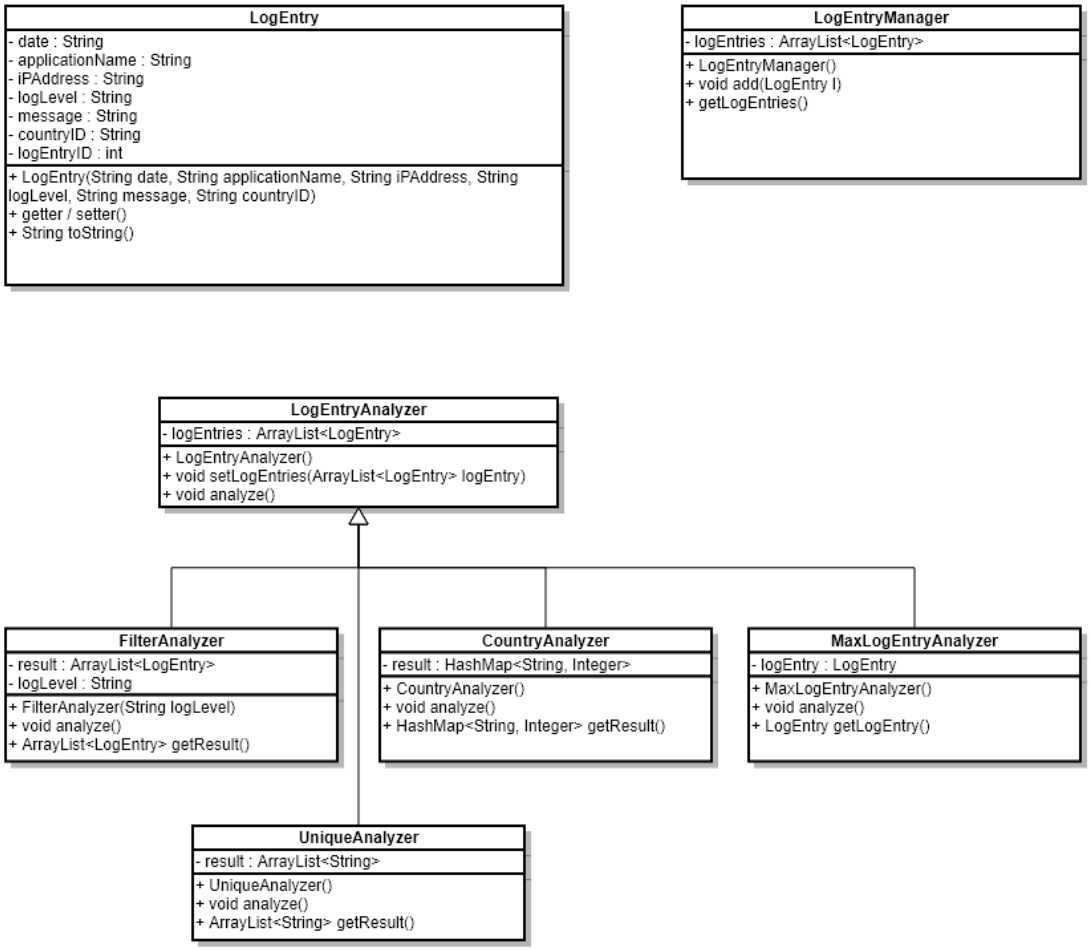
**Beispiel 2: Rekursion [Gesamtpunkte: 5 Punkte]**
Erzeugen Sie im Package *recursion* eine neue **Klasse** namens **RecursiveMultiplication**. Eine mathematische Multiplizierung kann auch über Additionen abgebildet werden, welche in diesem Fall rekursiv abgebildet werden soll.
Beispiel:
a * b = a + a + a + a = a*b ("a wird b mal zu a addiert")
5 * 4 = 5 + 5 + 5 + 5 = 20.

Erzeugen Sie dazu in dieser Klasse eine **statische Methode** mit folgender Signatur:

```
public static int multiplicate(int a, int b) {
      //TODO: your code here…
}
```

**Beispiel 3: OOP [Gesamtpunkte: 50 Punkte]**

Betrachten Sie das folgende Klassendiagramm.




[3 Punkte] Die Klasse ***LogEntry*** mit den Attributen ist bereits im StarterKit gegeben. Diese implementiert alle Attribute des Klassendiagramms mit dem Access-Modifier *private*. Finalisieren Sie den Konstruktor entsprechend der gegebenen Signatur.

[1 Punkt] Erstellen Sie getter- und setter-Methoden für alle Attribute. Erstellen Sie die Methode .toString(), .hashCode() und .equals().

[2 Punkte] Stellen Sie im Konstruktor sicher, dass nur LogLevel (Attribut *logLevel*) mit den Werten „debug“, „info“, „warning“, „error“ und „fatal“ übergeben werden. Wird ein anderes LogLevel übergeben, so soll der Wert „undefiniert“ gesetzt werden.
[3 Punkte] Verwenden Sie für die Sicherstellung des *logLevel* eine switch-case-Anweisung anstatt von *if*-Abfragen.

[7 Punkte] Die Klasse LogEntry hat auch ein Attribut *logEntryID*. Dieses wird nicht als Parameter im Konstruktor übernommen. Erstellen Sie stattdessen eine entsprechende Implementierung innerhalb des Konstruktors, sodass jedes Objekt von LogEntry eine eindeutige, aufsteigende Nummer bekommt. Gerne können Sie hierzu auch weitere Hilfsvariablen oder Attribute anlegen.
Im Starterkit finden Sie bereits eine Fallback-Implementierung. Diese können Sie bestehen lassen, wenn Sie mit der oben beschriebenen Implementierung Probleme haben.

[1 Punkt] Betrachten Sie die Klasse ***LogEntryAnalyzer*** und deren Subklassen genauer. Implementieren Sie vorerst nur die Klasse *LogEntryAnalyzer*. Implementieren Sie eine setter-Methode für das private Attribut *logEntries (ArrayList von LogEntry)*.

[0,5 Punkte] Schreiben Sie die Methode *public void analyze()*.
[1,5 Punkte] Stellen Sie sicher, dass die Methode in allen Sub-Klassen implementiert werden muss.

[1 Punkt] Schreiben Sie die Klasse ***LogEntryManager*** mit dem privaten Attribut und Konstruktor entsprechend dem Klassendiagramm und erstellen Sie die getter-Methode für das private Attribut.

[1 Punkt] Implementieren Sie die Methode *public void add(LogEntry p)*, welche eine neue LogEntry in die Liste aufnimmt.

[2 Punkte] Betrachten Sie die Demo-Klasse *DemoApp()*. In dieser finden Sie eine Methode *createTestLogEntry()*, die Ihnen zufällige Test-LogEntries erstellt. Erstellen Sie in der Main-Methode ein Objekt von LogEntryManager und rufen Sie die die Methode add(…) einhundert Mal auf, um LogEntryManager mit den Testdaten zu befüllen.

[2 Punkte] Leiten Sie die Klasse ***FilterAnalyzer*** von Basisklasse LogEntryAnalyzer ab und erstellen Sie die privaten Attribute (result und logLevel) entsprechend dem Klassendiagramm. Erstellen Sie eine getter-Methode für diese privaten Attribute. Im Konstruktor wird ein Attribut mit dem Namen logLevel übernommen.

[3 Punkte] Implementieren Sie die Methode *public void analyze()* in der Klasse FilterAnalyzer. Im Attribut result sollen hierbei nur LogEntries abgelegt werden, die ein logLevel entsprechend dem Attribut haben.

[1 Punkt] Leiten Sie die Klasse ***CountryAnalyzer*** von der Basisklasse LogEntryAnalyzer ab und erstellen Sie ein privates Attribut (result) entsprechend dem Klassendiagramm. Erstellen Sie eine getter-Methode für dieses private Attribut.

[7 Punkte] Implementieren Sie die Methode *public void analyze()* in der Klasse CountryAnalyzer. Speichern Sie in der HashMap *result* die Anzahl der LogEntries je Land.

[1 Punkt] Schreiben Sie die Klasse ***MaxLogEntryAnalyzer*** und leiten Sie diese von der Basisklasse LogEntryAnalyzer ab. Implementieren Sie das private Attribut *logEntry* mit dem im Klassendiagramm angegebenen Datentyp.

[5 Punkte] Implementieren Sie die Methode *public void analyze()* in der Klasse MaxLogEntryAnalyzer. Stellen Sie das LogEntry mit der höchsten ID in der Liste der LogEntries fest und speichern Sie diesen im Attribut logEntry ab.

[1 Punkt] Schreiben Sie eine Klasse ***UniqueAnalyzer*** und leiten Sie diese von der Basisklasse LogEntryAnalyzer ab. Erstellen Sie ein privates Attribut mit dem Namen *result* vom Typ ArrayList<String> und eine getter-Methode hierzu.

[5 Punkte] Implementieren Sie die Methode *public void analyze()* in der Klasse UniqueAnalyzer. In der Methode sollen alle eindeutigen Messages aller LogEntries in der Liste result gespeichert werden.

[2 Punkte] Erweitern Sie die Demo-Anwendung, sodass Sie unterschiedliche Objekte der LogEntryAnalyzer erstellen und jeweils die analyze-Methode aufrufen. Kontrollieren Sie jeweils das Ergebnis. Schreiben Sie für die vier konkreten Klassen von LogEntryAnalyzer je einen Anwendungsfall. Sofern vorhanden, zeigen Sie auch, dass die getResult()-Methoden funktionieren.