Bitte auf jedem Blatt Ihren Vor- und Nachnamen eintragen. | Please fill in your full name on every page.

.....

Software Engineering I (DiBSE-B-3-SE1-SE1-ILV)

WS 2022

Dipl.-Ing. (FH) Kristian Hasenjäger

DiBSE 2021.

Klausur

26. Januar 2023
Bearbeitungszeit: 13.15 – 15:45

Erlaubte Unterlagen bzw. Hilfsmittel | Aids permitted: -

- Persönlicher Laptop
- Open Books

Individuelle, eigenständige Bearbeitung.
Datenaustausch jeglicher Art (Text, Sprache, Dateien) direkt oder indirekt mit Dritten Personen oder deren Ressourcen ist *nicht* zulässig.
Kopfhörer sind nicht zulässig

1 Zur Beachtung

1. Die Gesamtaufgabe muss mit **`gradlew run`** (auch) von der Kommandozeile ausführbar sein, nachdem diese zuvor mit **`gradle clean`** gesäubert wurde (Achtung vor FileExplorer/Console lock!).
2. Testklassen, wo gefordert, müssen mit **`gradlew test`** ausführbar sein.
3. Die Abgabe ist unbedingt mit **`gradle mcisrczip`** zu packen:
   a. Das entsprechende Paket wird in **`build/distributions/Assignment.zip`** erzeugt.
   b. Dieses bitte umbenennen in **`Nachname_Vorname.zip`** ...
   c. ... und in Sakai Assignments laden.
4. Beurteilt kann nur Code werden, welcher fehlerfrei kompiliert.
5. Auskommentierter Code kann nicht gewertet werden.
6. Es gibt 3 Aufgaben mit insgesamt 100 Punkten.
7. Die Aufgaben der Klausur basieren auf einem Klassengerüst (inkl. Gradle build file), das von **`Sakai->SE1->Resources->Klausur->WH-Klausur_Grundgeruest_SE1_2023_01_26.ZIP`** heruntergeladen und in Ihrer IDE installiert werden sollte.

In der Klasse App finden Sie die (einzige) `main()` Methode:

- *Arbeiten Sie Aufgabe für Aufgabe ab:*
  In der **`main()-`**Methode der Klasse **`App`** demonstrieren Sie die Funktionalitäten ihrer erstellten Klassen. Hier bitte mit wenigen Codezeilen lediglich wesentliche Klassen instanziieren, konfigurieren und verwenden. Die eigentliche Logik sollte in den jeweils verantwortlichen Klassen implementiert sein. Tw. ist der Code in main bereits vorgegeben.

- *Erklären Sie, was Ihr Programm macht:*
  Relevante Programmschritte sind schlüssig mit **`System.out.println(...)`** zu visualisieren, um Ihren Programmablauf erklärend nachvollziehbar zu machen.

- *Kommen Sie mit einer Aufgabe nicht weiter?*
  Schreiten Sie weiter zur nächsten und behelfen Sie sich ggf. mit kleinen Patches falls dies nötig sein sollte.

- *Bitte keine vollständigen Ruinen*
  Geben Sie nur kompilierfähigen Code ab, der auch einen Sinn hat.

Aufgabe 1 (35Punkte)

Zu implementierendes Pattern:	**Factory + Singleton**
Paketname:	**`se1.klausur.schneekofel`**
Main-Klassenname:	**`Skipasskassa`**

Hinweis: Zum implementierende Klassen sind unterstrichen dargestellt

Der `Skipass`-Verkauf im Skigebiet Schneefokel soll durch ein IT-System beschleunigt werden.

- Es gibt die Skipassarten `Kind, Erwachsen, Senior`
- Der Kaufpreis richtet sich nach der Art des Skipasses, und der Anzahl Gültigkeitstage
- Es wird Rabatt gewährt: >=3 Tage:→10%, >=7 Tage.-->20%, >=21 Tage→30%.
- Die maximal mögliche Gültigkeitsdauer beträgt 365 Tage.

Hierfür kann an der `Skipasskassa` (**realisiert als Factory-Pattern**) über eine Methode `kaufen( alterInhaberIn, vonTag, anzahlTage)` jeweils eine der folgenden Skipass-Arten als Instanz erzeugt werden.

Skipass-Art	Altersbereich	Preis	Max. Gültigkeit (Tage)
`Erwachsen`	18 Jahre oder alter	40 € / Tag	365
`Kind`	jünger als 18 Jahre	25 € / Tag	365
`Senior`	65 Jahre oder älter	30 € / Tag	365

Jeder Skipass beinhaltet nach dem Kauf folgende gültigen Daten:

- `serienNummer` Typ: `long`
- Anfang Gültigkeitsbereich `vonTag` Typ: `int` (1. Januar = 1, durchlaufend)
- `anzahlTage` Typ: `int`
- `preis` Typ: `double`

Jeder Skipass bietet folgende Methoden:

- `get`ter zu obigen Daten
- `toString(): String` // Alle Daten als String liefern
- `istGueltig(tag): boolean` // Ist Skipass an tag gültig ?

❖ Die `serienNummer` im Skipass wird durch ein **Singleton-Pattern** `SeriennummernGenerator` ermöglicht.

a.) → (5P.) Implementieren Sie zunächst für die Skipässe eine geeignete **Klassenhierarchie** welche alle nötigen Klassen, Eigenschaften und zumindest die noch leeren Methodenrümpfe beinhaltet. Jede Skipassart ist durch eine eigene Klasse definiert.

b.) → (25P.) Implementieren sie in einem zweiten Schritt die **zugehörigen Methoden** bzw. weiteren Klassen.
(Factory Klasse `Skipasskassa` 15P, Singleton Klasse `SeriennummernGenerator` 10P.).

c.) → (5P.) In der main() Methode von Skipasskassa **kaufen** Sie bitte mindestens einmal jede Skipassart, wobei dies sowie einige **Gültigkeits-abrfragen** durch geeignete Ausgaben am Bildschirm protokolliert werden sollen.

Aufgabe 2 (35Punkte)

Zu implementierendes Wirkprinzip: **JUnit5 Tests**

Diese Aufgabe soll die korrekte Funktionalität der Klassen aus Aufgabe 1 validieren.

→ **(35P.)** Implementieren Sie für die Klassen `Skipasskassa` und `Skipass` geeignete **Testklassen**, die insbesondere folgende Operationen auf Korrektheit prüfen:

- Klasse `Skipasskassa` den Skipasskauf:
  Test der Erzeugung der korrekten Skipassart (*), mit valider Seriennummer und korrekter Preisberechnung inkl. Rabattierung.
- Klasse `Skipass`: Test der Gültigkeitsabfrage

Hinweise:

- In IntelliJ eine leere Testklasse erzeugen:
  → Cursor auf Klassennamen der zu testenden Klasse platzieren
  → <ALT> + <ENTER>
  → "Create Test"
  → Neue Testklasse in: `/src/test/java/se1.klausur/schneekofel/...`

- Mit `gradle test` soll wie üblich ein Test-Report generiert werden.

- Die Aufgabe kann auch dann bearbeitet werden, wenn Aufgabe 1 nicht vollständig abgeschlossen wurde. Dann ergeben sich u.U. fehlschlagende Tests, was nicht zum Punkteabzug führt sofern der Test korrekt erstellt wurde. Aufgabe 1 muss jedoch mindestens kompilieren und die nötigen Klassen/Methoden bereitstellen, ungeachtet einer korrekten Implementierung.

- (*) Feststellen ob ein Objekt vom Typ einer gewissen Klasse ist:
  `if ((myobject instanceof MyClass) == true)`
  dann ist `myobject` eine **Instanz** der Klasse `MyClass`.

Aufgabe 3 (30Punkte)

Zu implementierendes Pattern:	**Composite**
Paketname:	**`se1.klausur.gui`**
Main-Klassenname:	**`Minigui`**

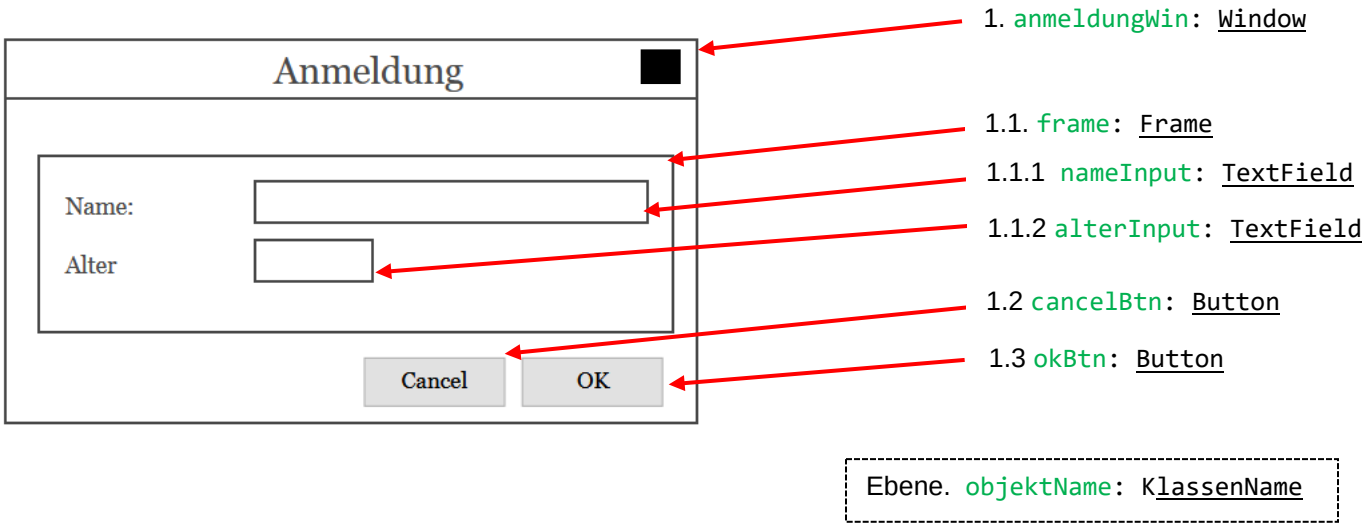
In einem GUI (Graphical User Interface) werden üblicherweise `Component`en mittels Composite Pattern zusammengefügt. Ein GUI besteht somit aus einer Anzahl an `Component`en die wiederum weitere `Component`en enthalten können usw. In dieser Aufgabe bauen wir ein einfaches GUI.

In einem GUI unterscheiden sich zwei Arten von `Component`en:

1.) Komponenten die *keine* weiteren Komponenten enthalten können:
   a. `Button`
   b. `TextField`
2.) Komponenten, die weitere Komponenten enthalten können:
   a. `Window`
   b. `Frame`

Jede `Component` besitzt:

1.) Einen `name` vom Typ `String` (u.a. für Fenstertitel TextField-Label, Button-Label)
2.) Koordinaten `x,y, width, height` vom Typ `int`
3.) Die Methode `draw()` gibt am Bildschirm Klassennamen (*), Namen, Koordinaten und sodann *rekursiv* eventuelle Kinderkomponenten aus.



a. → (15P.) Implementieren Sie alle nach dem **Composite Pattern** nötigen Klassen und/oder Interfaces mitsamt Methoden und Eigenschaften.

b. → (15P.) Bauen Sie in der `main`()-Methoder einer Klasse **`Minigui`** obiges **Beispielgui** zusammen, indem Sie in a.) definierte Klassen instantieren und nach dem Composite Pattern bzw. der Abbildung folgend ineinander schachteln.
Beachten Sie bitte die gegebenen Objektnamen.

Rufen Sie sodann die Methode `draw`() im `Window`-objekt `anmeldungWin` auf. Damit soll rekursiv der GUI Aufbau aller enthaltenen Komponenten nachvollziehbar **am Bildschirm ausgegeben**.

(*) Der Klassenname eines Objekts `myObject` kann mittels `myObject.getClass().getSimpleName()` erfragt werden.

Ende der Klausur ☺