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 v.1.

Klausur

25.November 2022
Bearbeitungszeit: 15.15 – 17:45

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 5 Aufgaben mit pro Aufgabe 20 Punkten.**

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.

- *Grenzen Sie Aufgaben voneinander ab:*
  Bei Beginn jeder neuen Aufgabe ist mit der Methode **`aufgabenabgrenzer(int beginAufgabe)`** die Aufgabennummer am Bildschirm auszugeben

- *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.

Die Aufgaben der Klausur basieren auf einem Klassengerüst (inkl. Gradle build file), welches von Sakai→SE1→Resources→_Klausur→ KlausurGrundgeruest_SE1_2022_11_25.ZIP heruntergeladen und in Ihrer IDE installiert werden sollte.

Folgende Klassenhierarchie für ein Dateisystem ist nebst der Hauptklasse `App` enthalten:

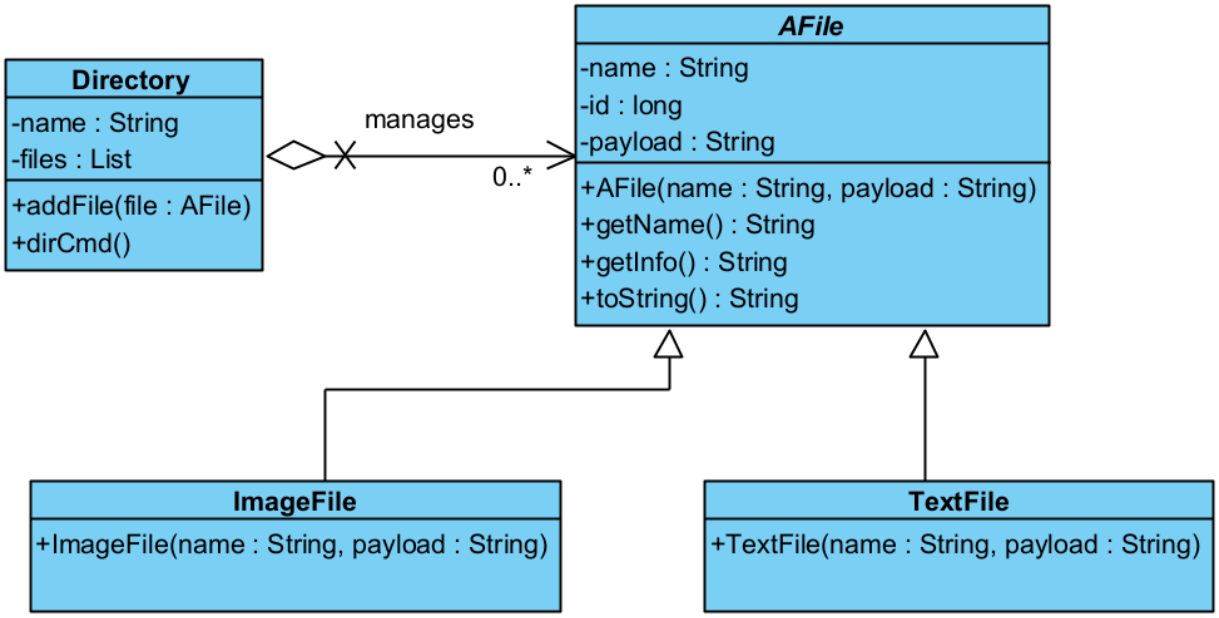

Abbildung: Zwei konkreten Dateitypen erben von der *abstrakten* Superklasse `AFile`.
Eine Klasse `Directory` kann beliebig viele Dateien aufnehmen

Jedes `AFile` besteht aus:

1. dem Dateinamen `name`,
2. der eindeutigen `id`
3. und dem eigentlichen Dateiinhalt `payload`

Die Klasse `Directory` kann:

1. mit `addFile(AFile af)` Dateien aufnehmen,
2. und mit `dirCmd()` alle Dateien am Bildschirm auflisten

1: Singleton (20 Punkte)

Bei der Instantierung eine jeden Datei vom Typ `AFile` soll der Eigenschaft `id` vom Typ `long` eine eindeutige Identifikationsnummer zugewiesen werden.
Zuständig hierfür soll eine neue Klasse `IdGenerator` sein, die nach dem Singleton Pattern eindeutige IDs liefert.

In `main()` werden im Bereich zu Aufgabe 1 bereits zwei unterschiedliche Dateien vom Typ `AFile` instanziiert, dem Verzeichnis `myDir` vom hinzugefügt und mit `myDir.dirCmd()` am Bildschirm aufgelistet. Kontrollieren Sie sodann, ob die `id`**'s** auch wirklich **unique** sind:

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
# Aufgabe: 1:

'duck.txt' dem Verzeichnis 'myDir' hinzugefuegt.
'flower.jpg' dem Verzeichnis 'myDir' hinzugefuegt.

Listing directory 'myDir':
duck.txt        TextFile        52 bytes (id=1)
flower.jpg      ImageFile       26 bytes (id=2)
```

Hinweis: Bitte ändern Sie in main() den bereits vorhandenen Code zu Aufgabe 1 *nicht*.

Aufgabe 2: Factory (20 Punkte)

Eine Klasse `FileFactory` soll auf Anforderung unterschiedliche Dateiarten erzeugen. Hierbei kommt das Factory Pattern zur Anwendung.

Spezifikation:

1. Die Methode `createFile(String type)` liefert folgende Instanzen:
   a. type="`image`" → `ImageFile`
   b. type="`text`" → `TextFile`
2. Bei Fehlerhafter Parametrisierung wird eine `IllegalArgumentExpeption` geworfen, welche den Fehler genauer beschreibt.
3. Der Dateiname muss mindestens ein Zeichen lang sein (`name.length()>=1`)
4. Keiner der Parameter darf null sein.

In `main()` im Bereich zu Aufgabe 2 beauftragen Sie die Fabrik, zwei Dateien mit *gültigen* Parameter `type`, sowie eine Datei mit *ungültigem* Parameter `type`, herzustellen. Hierbei müssen eventuelle `Exception`s aufgefangen werden, begleitet von sinnvollen Fehlermeldungen als Bildschirmausgabe.

Die Bildschirmausgabe sollte im Wesentlichen wie folgt erfolgen:

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
# Aufgabe: 2:

cat.jpg         ImageFile       29 bytes (id=3) wurde erzeugt.
dog.txt         TextFile        13 bytes (id=4) wurde erzeugt.
stick.txt       TextFile        14 bytes (id=5) wurde erzeugt.
Fabrik kann kein 'wrong' herstellen!
```

Aufgabe 3: Unit Tests (20 Punkte)

Die Klasse `FileFactory` soll nach Spezifikation (vgl. Aufgabe 2) eingehend überprüft werden. Erstellen Sie die zugehörige Testklasse mit sinnvoller Testabdeckung. Kontrollieren Sie das generierte HTML-Testprotokoll (üblicherweise in `/reports/tests/test/index.html`).

Aufgabe 4: Observer (20 Punkte)

Jedes Directory soll fortab sicherstellen, dass es nur virenfreie Dateien aufnimmt. Dies soll mithilfe des Observer Patterns umgesetzt werden.
Es gilt:

1. An jedem `Directory` können sich beliebig viele `IVirusscanner` unterschiedlicher Art *registrieren*. `IVirusscanner` nehmen die Rolle von Observern ein.
2. Wird dem `Directory` eine Datei mit der bestehenden Methode `addFile(..)` hinzugefügt, so wird diese Datei an *alle* observierenden `Virenscanner` zum antivirenscan übermittelt.
3. Findet auch nur einer der registrierten `IVirusscanner` einen Virus, darf die Datei *nicht* im Directory aufgenommen werden.
4. Ein Virus liegt vor, falls im „payload“ der Datei der Textstring „`virus`“ oder „`scam`“ vorkommt.
   Hinweis: `myString.indexOf("virus")` liefert Werte >=0, falls `myString` die Zeichenkette „`virus`“ enthält
5. Es sind zwei Klassen unterschiedlicher `IVirusscanner` zu implementieren:
   a. Der Virenscanner `AntivirusOne` findet nur den Textstring „`virus`“
   b. Der Virenscanner `AntivirusTwo` findet nur den Textstring „`scam`“

Im main() Bereich zu Aufgabe 4 demonstrieren Sie ihre Implementierung, indem Sie für die bereits vorhandene Directory-Instanz `myDir` je eine Instanz der zwei unterschiedlichen Virenscanner herstellen. Diese verhindern sodann, dass die zwei im Quellcode bereits vorhandenen „gefährlichen“ Dateien `virus1` und `virus2` dem Directory `myDir` mit `myDir.addFile()` hinzugefügt werden können. Stattdessen werden entsprechende Virenalarme am Bildschirm ausgegeben.

Ausgabe mit beiden aktiven Virenscannern nach dem Versuch, die gefährlichen Dateien `virus1` und `virus2` zu `myDir` hinzuzufügen, gefolgt von `myDir.dirCmd ():`

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
# Aufgabe: 3:

VIRUSALARM: 'danger.exe' enthält Virus!
VIRUSALARM: 'boom.exe' enthält Virus!
VIRUSALARM: 'boom.exe' enthält Virus!

Listing directory 'myDir':
duck.txt        TextFile        52 bytes (id=1)
flower.jpg      ImageFile       26 bytes (id=2)
```

Aufgabe 5: Composite (20 Punkte)

Die Klasse „Directory“ kann durch *Änderung weniger Programmzeilen* dahingehend refaktorisiert werden, dass sie *„vom Typ AFile“* wird. Dann lassen sich baumartig verschachtelte Verzeichnisstrukturen anlegen, wie in jedem Dateisystem üblich.

Achtung: Damit der bisher erstellte Code unbescholten bleibt, arbeiten Sie fortan mit einer neuen Kopie der Klasse Directory: In der Projektbaumansicht in IntelliJ links *markieren* Sie die Klasse Directory, dann → Rechtsklick → „copy“ → „paste“ mit neuem Namen „DirectoryNew“.

Im main() Bereich zu Aufgabe 5 bauen Sie, ausgehend von einer Instanz root vom Typ DirectryNew, eine wie folgt strukturierte Datei- bzw. Objektstruktur auf:

Instanz	Typ
root	DirectoryNew
datei1.txt	TextFile
image1.jpg	ImageFile
Verzeichnis2	DirectoryNew
robots.txt	TextFile
songs	DirectoryNew
justinbiber.txt	TextFile

Beim Zusammenbau der Objektstruktur wird Schrittweise folgendes (von addFile() in DirectoryNew):ausgegeben:

```
'datei1.txt' dem Verzeichnis 'root' hinzugefuegt.
'image1.jpg' dem Verzeichnis 'root' hinzugefuegt.
'Verzeichnis2' dem Verzeichnis 'root' hinzugefuegt.
'robots.txt' dem Verzeichnis 'Verzeichnis2' hinzugefuegt.
'songs' dem Verzeichnis 'Verzeichnis2' hinzugefuegt.
'justinbiber.txt' dem Verzeichnis 'songs' hinzugefuegt.
```

1. Hinweise:
   Es muss lediglich DirectoryNew refaktorisiert werden wo nötig.
2. Änderungen an weiteren oder neue Klassen bzw. Interfaces sind nicht nötig.

Ende der Klausur ☺