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 2020

Dipl.-Ing. (FH) Kristian Hasenjäger

DiBSE 2020

Klausur

27.November 2020
Bearbeitungszeit: 15.15 – 17:45

Erlaubte Unterlagen bzw. Hilfsmittel | Aids permitted: -

- Persönlichen PC/Laptop aus der Ferne
- Internet Verbindung
- Open Books

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

1 Zur Beachtung

1. Jede Aufgabe muss mit `gradlew mcirun -Pex=name` (auch) von der Kommandozeile ausführbar sein, nachdem diese zuvor mit `gradle clean` gesäubert wurde.
2. Testklassen müssen mit `gradlew test` ausführbar sein.
3. Die Abgabe ist mit `gradle clean` zu säubern und mit `gradle mciSrcZip` zu packen:
   a. Das entsprechende Paket wird in `build/distributions/Assignment.zip` erzeugt.
   b. Dieses bitte umbenennen in `Nachnamen_Vorname.zip` ...
   c. ... und in Sakai Assignments laden.

4. Beurteilt kann nur Code werden, welcher fehlerfrei kompiliert
5. Der Programmablauf ist durch geeignete Ausgaben nachvollziehbar zu gestalten.

Beachten Sie bei der Implementierung insbesondere

- ... gelernte Design Prinzipien zu beachten
- **...**eine robuste/fehlerbewusste API-Implementierung aus Sicht des Clients

Bei jeder Aufgabe:

- In der **`main()-`**Methode der jeweiligen Klasse demonstrieren Sie die lt. Aufgabenstellung erstellte Funktionalität.
- Alle relevanten Schritte sind bitte mit **`System.out.println(...)`** zu visualisieren, um Ihren Programmablauf nachvollziehbar zu mache.

2 Hinweise:

3 Aufgabe 1 (30Punkte)

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

In der ersten Aufgabe werden wir für ein fiktives Betriebssystem **`MiniOs`** API und Funktionalität für den Zugriff auf Ausgabe-Geräte bereitstellen: Bevor ein Device („Gerät“) üblicherweise verwendet werden kann, muss hierzu ein passender Device Driver „Gerätetreiber“ geladen und initialisiert werden.

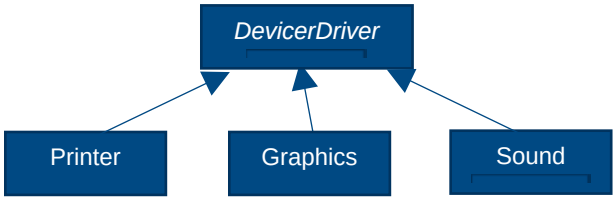
`MiniOs` ermöglicht es somit mittels **`DeviceDriver`** („Gerätetreiber“) auf Device-Typen wie Printer, Graphics Card oder Sound System zuzugreifen:
Auf Anfrage eines Clienten wird hierzu eine Instanz von **`DevicerDriver`** für ein bestimmtes Device („Ausgabegerät“) vom Betriebssystem einmalig *erzeugt* und *initialisiert*, womit dieser als installiert gilt. Diese gleiche Instanz wird dann für alle weiteren Anfragen für den Gerätetyp ausgeliefert.

NB: Beachten Sie bei der Implementierung von MiniOS das **Open-Closed-Prinzip (OCP).**

Ein Client verwendet das **Device API** zB. wie folgt:

```
Printer p= miniOS.getDriver("Printer");
p.print(„Hello Printer").
```

- Es ist zunächst eine geeignete Klassenhierarchie in Java zu entwickeln:



- Jeder **DeviceDriver...**
  - ... muss vor erstmaliger Verwendung mit **init()** installiert werden, und gilt somit als Installiert. Wir einfach das so:
    System.out.println("**Treiber**" **+getName()+** "**installiert**");
  - ... hat einen Namen, der mit **getName()** erfragt werden kann.

- Jeder *spezialisierte* DeviceDriver für die Devices...
  - **Printer → ...kann print(String text)**
  - **Graphics → ...kann draw(String svg)**
  - **Sound → ...kann playSong(String title)**I

→ Implementieren Sie **MiniOS**, welches mit **getDriver(name)** auf Anfrage eines Client (=main Methode von MiniOs) den geeigneten Treiber liefert!

→ **(30P.)** Demonstrieren Sie eingehend ihre in MiniOS implementierten Funktionalitäten im **main()** Bereich, wobei alle „relevanten" Schritte mit **System.out.println(...)** nachvollziehbar gemacht werden müssen.

4 Aufgabe 2 (30Punkte)

Zu implementierendes Pattern:	**Composite**
Paketname:	**`se1.klausur.composite`**
Main-Klassenname:	**`Dateisystem`**

Wir möchten ein Dateisystem mit einem Composite Muster abbilden:

- In einem Dateisystem gibt es **`Datei`**(en).
- Und **`Ordner`**, die wiederum Dateien enthalten.

Alle haben einen **`fileName`** als ***`String`***. Dateien haben auch ein **`dataRef`** des Dateiinhalts als ***`String`*** (=Simulation) und sowie die zugehörige **`fileSize`** als ***`int`*** in Bytes.

→ **(15P.)** Implementieren Sie die zugehörigen Java-Klassen/Interfaces/.... nach dem Composite Muster, zunächst nur die nötigen Definitionen noch ohne Methoden.

→ **(15P.)** Implementieren Sie Funktionalität in obigen Java-Klassen:

```
mkdir(name)                              // Verzeichnis name erzeugen
createFile(String name, String data)     // Datei name mit Inhalt data erzeugen
ls()                                     // Dateiinhalt mit in der Konsole ausgeben
cd(name)                                 // In Verzeichnis name wechseln
```

Demonstrieren Sie eingehend ihre in **`Dateisystem`** implementierten Funktionalitäten im **`main()`** Bereich, wobei alle „relevanten“ Schritte mit **`System.out.println(...)`** nachvollziehbar gemacht werden müssen in den einzelnen Methoden. In Main könnte sich etwa folgendes finden:

```
Dateisystem ds = new Dateisystem()
Verzeichnis vz= ds.mkdir(“/“);   // Wurzelverzeichnis
vz.createFile(“cheatsheet.txt“,“keine Rekursion nötig in Aufgabe 2“);
vz.createFile(“bluesky.jpg“,“[Tiroler Himmel]“);
vz.mkdir(“Unterverzeichnis“);

usw…

vs.ls();  // Bildschirmausgabe der Verzeichnissr und Dateien+Grösse
```

5 Aufgabe 3 (40Punkte)

Zu implementierendes Wirkprinzip: **JUnit5 Tests**
Paketname: **`se1.klausur.junit`**
Main-Klassenname: **`Micrologger`**

- Ein Micrologger soll zum Loggen („protokollieren“) von Integer Zahlen eingesetzt werden.
- Aufgrund der „Ultra Low Power“ Eigenschaft des Microcontrollers und seiner geringen Kosten sind max. 1000 Protokolleinträge möglich,
  - danach muss das Protokoll mit clear() gelöscht werden,
  - ansonsten wird eine **`IllegalStateException(„protocoll full“)`** geworfen..

- Folgende Methoden stellt die Klasse **`Micrologger`** somit zur Verfügung:

```
log(int i) throws IllegalStateException;    //wert i in Protokoll aufnehmen
                                            // throws IllegalStateException wenn voll.
cheat():int;   // letzten Protokolleintrag zurückliefern und aus Protokoll entfernen
getMax():int; // größten protokollierten Wert zurückliefern
clear();      // Protokoll löschen und Maximalwert zurücksetzen
getSize():int; // Anzahl der Protokolleintrage liefern
```

→ **(10P.)** Implementieren Sie die Klasse **`Micrologger`**.

→ **(30P.)** Implementieren Sie für die oben spezifizierte Funktionalität von **`Micrologger`** eine geeignete Testklasse.

Achten Sie dabei auf geeignete Auswahl der Testfälle:
- Möglichst umfassende Abdeckung ...
- ... bei möglichst geringe Anzahl.

Ende der Klausur ☺