HSLU Hochschule Luzern

Algorithmen und Datenstrukturen (AD)

Übungen: Entwicklung Intro (E0)

Themen: Installation bzw. Aktualisierung der Entwicklungsumgebung und aller Werkzeuge, Wiedereinstieg in die Objektorientierte Programmierung und Nutzung von Datenstrukturen.
Zeitbedarf: ca. 120 Minuten

Roland Gisler, Version 2.0.1 (HS24)

1 Aktualisierung der Entwicklungswerkzeuge und Umgebung (ca. 20')

1.1 Lernziele

- Installation bzw. Aktualisierung der benötigten Software.
- Einrichten eines neuen Java-Projektes für die Übungen.

1.2 Grundlagen

Der Start eines neuen Semesters ist ein guter Moment um die Entwicklungswerkzeuge auf den aktuellen Stand zu bringen. Das beinhaltet die Installation aktueller Bugfixes sowie das Einrichten eines neuen Java-Projektes für die Übungen. Hinweis: Eine Auswahl der am häufigsten benutzten Software wird ihnen wie aus OOP/PLAB schon bekannt über SWITCHdrive (http://bit.ly/2OH3Uhh) zur Verfügung gestellt.

1.3 Aufgaben

a.) Auf ILIAS steht ihnen in den Modulunterlagen im Unterverzeichnis «`Java Template und Anleitung`» das Dokument `OOP_JavaDevelopmentManual_jdk21.pdf` und das Projekttemplate `oop_maven_template_jdk21-5.2.1.zip`[1] zur Verfügung, welche Sie beide (aber in älteren Versionen) schon aus dem Modul OOP oder PLAB kennen. Laden Sie diese beiden Dateien herunter.

b.) Im **Kapitel 8** des Dokumentes `OOP_JavaDevelopmentManual_jdk21.pdf` finden Sie Tipps und Hinweise zur Aktualisierung der vorhandenen Software.

c.) Wie im Modul OOP empfehlen wir ihnen, mit dem zur Verfügung gestellten Projekttemplate zu arbeiten. Da wir auch in AD wöchentliche Übungen lösen, macht es Sinn ein Projekt (Namensvorschlag: `ad_exercises`) einzurichten, und darin fortlaufend pro Woche/Thema getrennte Packages anzulegen (z.B. `ch.hslu.ad.sw01`). Eine Anleitung dazu finden Sie ebenfalls im oben referenzierten Dokument.

[1] Kann auch in einer neueren Version vorliegen.

2 Wiedereinstieg in die Programmierung mit Java (ca. 100')

2.1 Lernziele

- (Wieder-)Einstiegsübung in die Programmierung mit Java.
- Implizite Verifikation der Entwicklungsumgebung und -tools.
- Vorbereitung für zukünftige Übungen zu Datenstrukturen.

2.2 Grundlagen

Um das Verständnis der Funktionsweise verschiedener Datenstrukturen zu erhöhen, führen wir einige Übungen im gleichen Themenkontext aus: Wir wollen eine sehr einfache, direkte Speicherverwaltung exemplarisch entwerfen und auch implementieren.

Dabei gehen wir davon aus, dass wir in einem Gesamtspeicher von maximal 1GB beliebig grosse Blöcke reservieren und wieder freigeben können.[2] Was sehr einfach klingt, birgt ein paar sehr interessante Herausforderungen, sowohl in Bezug auf die verwendeten Datenstrukturen als auch auf Algorithmen!

Im Rahmen dieser Aufgabe wollen wir eine minimale Infrastruktur aufbauen, mit welcher wir dann experimentieren können. Das hilft uns gleichzeitig auch einiges was Sie im Modul OOP schon gelernt haben (und das war eine Menge!) wieder etwas präsent zu machen.

2.3 Aufgaben

a.) Hier sehen Sie ein kleines Beispielprogramm, welches die Verwendung dieser Speicherverwaltung zeigt:

```
public final class MemoryDemo {
    ...
    public static void main(final String[] args) {
        final Memory memory = new MemorySimple(1024);
        LOG.info(memory);
        final Allocation block1 = memory.malloc(16);
        LOG.info(block1);
        LOG.info(memory);
        final Allocation block2 = memory.malloc(8);
        LOG.info(block2);
        LOG.info(memory);
        memory.free(block1);
    }
}
```

Hier die Ausgabe des obigen Programmes (Beispielhaft):

```
2024-09-11 17:14:22,968 INFO - MemorySimple[Belegt: 0; Frei: 1024]
2024-09-11 17:14:22,970 INFO - Allocation[Address:0; Size:16]
2024-09-11 17:14:22,971 INFO - MemorySimple[Belegt: 16; Frei: 1008]
2024-09-11 17:14:22,971 INFO - Allocation[Address:16; Size:8]
2024-09-11 17:14:22,971 INFO - MemorySimple[Belegt: 24; Frei: 1000]
```

Identifizieren Sie die verwendeten Klassen, Interfaces, Methoden, Attribute und Datentypen. Skizzieren Sie sich zur Übersicht ein UML-Klassendiagramm aller Klassen, Interfaces und Beziehungen, es lohnt sich! Tipp: Auf ILIAS finden Sie eine praktische UML-Notationsübersicht. Hinweise zur Implementation finden Sie in den folgenden Teilaufgaben.

[2] Wenn Ihnen das zu Abstrakt ist, hilft Ihnen sicher die folgende Analogie: Stellen Sie sich vor, Sie haben einen Kinosaal mit 2^{30} Plätzen in einer einzigen Reihe (Hui! ☺). Diese Plätze können in beliebiger Anzahl (also auch für Gruppen, deren Mitglieder jeweils nebeneinandersitzen wollen) reserviert, aber auch wieder storniert werden.

b.) Wir benötigen auf jeden Fall eine Klasse **`Allokation`**, welche einen Speicherbereich (egal ob frei oder belegt) eindeutig identifiziert. Das geschieht am Einfachsten über eine Startadresse und die Grösse des Speicherblockes (Anzahl Bytes). Welche Datentypen verwenden Sie?

c.) Die Klasse **`Allokation`** soll immutable implementiert sein. Das bedeutet, dass Objekte dieser Klasse nach der Instanziierung nicht mehr verändert werden können.

d.) Identische (gleiche) Allokationen sollen erkannt werden. Implementieren Sie dafür die beiden Methoden **`equals()`** und **`hashcode()`**. Sie sind bekanntlich eine wichtige Basis, dass Datenstrukturen überhaupt korrekt und effizient funktionieren können.

e.) Die Klasse **`Allokation`** soll das Interface **`Comparable`** implementieren, so dass Sie eine natürliche Ordnung (Sortierung) auf Basis der Startadresse aufweist.

f.) Es versteht sich von selbst, dass Sie schon längst entsprechende Unittests implementiert haben, oder? Ansonsten sollten Sie das spätestens jetzt **unbedingt** nachholen!

Hinweis: Denken Sie auch an AssertJ (siehe http://joel-costigliola.github.io/assertj/) als eine Erweiterung von JUnit, welche ihnen erlaubt einfachere und deutlich aussagekräftigere Asserts zu schreiben. Probieren Sie es aus, es ist im Projekttemplate bereits integriert!

g.) Die korrekte Implementation von **`equals()`** und **`hashcode()`** können Sie mit dem Tool EqualsVerifier (http://jqno.nl/equalsverifier/) verifizieren. Auch das kennen Sie bereits aus OOP, und es ist extrem wertvoll.