Prof. Dr. Christoph Bockisch
MSc Steffen Dick
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Übungen zur Vorlesung
Objektorientierte Programmierung: Wintersemester 2021/2022

Nr. 8, Abgabe bis 17.01.2022

Hinweis: Ab diesem Zettel sollen Sie die IntelliJ IDE zum Lösen Ihrer Zettel verwenden. Für den späteren Export zur Abgabe können Sie die eingebaute Funktionalität von IntelliJ verwenden. Gehen Sie dazu auf *File* → *Export* → *Project to Zip File...*

Aufgabe 8.1: Adam Stevenson oder Nicholas Hyde? 3 Punkte

Die Caesar Chiffre oder Caesar Verschlüsselung ist eine Verschlüsselung, in der Man Buchstaben des Alphabets um eine vorgegebene Zahl verschiebt, um sie zu kodieren. Die Caesar Chiffre kann allerdings auch durch eine Gleichsetzung von Buchstaben verstanden werden. Setzt man beispielsweise A = E, bedeutet dies, dass die Buchstaben um 4 verschoben werden müssen. Diese zweite Definition soll nun von Ihnen implementiert werden. Legen Sie hierfür zunächst eine Klasse `Caesar` an.

a) Schreiben Sie eine öffentliche, statische Methode `String decode(String coded, char a, char b)`, die mithilfe der Zuweisung des Buchstaben *a* auf den Buchstaben *b* die übergebene Chiffre entschlüsselt.
b) Testen Sie Ihre Implementierung mit JUnit-Tests und den folgenden Werten:

HGXBO	&	W = Z	→	KJAER
DKKLAJ	&	n = R	→	HOOPEN
TUFSMJOH	&	A = Z	→	STERLING
VJGFKDDWM	&	E = c	→	THEDIBBUK
ZQJCAKJIWOPAN	&	W = A	→	DUNGEONMASTER
XHZRGFLGJSOFRNS	&	G = B	→	SCUMBAGBENJAMIN

Aufgabe 8.2: Zauber und Hexen | 4 Punkte

In dieser Aufgabe sollen Sie Ihre Kartensammlung für das fiktive Kartenspiel *Spells & Witches* sortieren. Auf jeder Karte dieses Spiels ist ein Name, sowie das Erscheinungsjahr abgedruckt. Außerdem können Karten drei verschiedenen Typen angehören: Monsterkarten, Zauberkarte und Fallenkarten. Monsterkarten werden ebenfalls in zwei verschiedene Typen unterteilt: Normale Monster und Effektmonster.

Ihre Aufgabe ist es nun Ihre Kartensammlung zuerst nach Erscheinungsjahr, dann nach Typen (Monster (Normal, dann Effekt), Zauberkarten und dann Fallenkarten) und schlussendlich nach Namen zu sortieren.

a) Erstellen Sie die Klasse `Card`. Diese Klasse soll über ein Feld für den Namen, den Typen und das Erscheinungsjahr verfügen. Legen Sie die Typen der Karten als finale statische Felder in `Card` an, sodass diese zum Beispiel über `Card.EFFECT_MONSTER` aufgerufen werden können. Implementieren Sie außerdem entsprechende Getter-Methoden.

b) Implementieren Sie nun das Interface `Comparable` in Ihrer Klasse `Card`. Achten Sie bei Ihrer Implementierung darauf, dass die folgende Ordnung eingehalten wird.
   - Die Karten werden zunächst aufsteigend nach ihrem Erscheinungsjahr sortiert.
   - Bei gleichem Erscheinungsjahr sollen die Karten nach ihrem Typen sortiert werden. Dabei gilt:

     Normales Monster $\rightarrow$ Effekt Monster $\rightarrow$ Zauberkarte $\rightarrow$ Fallenkarte
   - Zu guter Letzt soll bei gleichem Jahr und gleichem Typen der Name der Karte für die Sortierung entscheidend sein.

c) Testen Sie Ihre Implementierung mithilfe eines JUnit-Tests, in welchem Sie ein Array mit mindestens 10 Karten sortieren. Dabei sollen wenigstens 5 einzigartige Karten in Ihrem Array vorhanden sein, sowie mindestens zwei Karten, bei denen sich lediglich das Erscheinungsjahr unterscheidet.

   Beachten Sie, dass Ihre Sammlung auch Duplikate enthalten kann.

Aufgabe 8.3: Jonah Hex | 5 Punkte

In dieser Aufgabe sollen Sie ein Interface entwickeln, mit dessen Hilfe natürliche Zahlen addiert und multipliziert werden können. Erstellen Sie das Interface *Number* und fügen Sie diesem die folgenden Methoden hinzu:

a) Definieren Sie die folgenden Interface-Methoden:
   - `int toIntValue()`, welche den Wert der Objektinstanz als `int` zurückgibt.
   - `void fromIntValue(int value)`, welche der Objektinstanz den Wert `value` zuweist.

b) Implementieren Sie die folgenden *Default*-Methoden mithilfe von *toIntValue* und *fromIntValue*:
   - `void add(Number number)`, addiert number zum Wert der Objektinstanz.
   - `void subtract(Number number)`, subtrahiert number vom Wert der Objektinstanz.
   - `void multiply(Number number)`, multipliziert number mit dem Wert der Objektinstanz und setzt das Ergebnis als neuen Wert.
   - `void divide(Number number)`, teilt den Wert der Objektinstanz durch number und setzt das Ergebnis als neuen Wert.

Erstellen Sie nun eine Klasse *HexaDecimal*, die das Interface *Number* implementiert. Die Klasse soll über ein Feld vom Typen *String* verfügen, in welchem der Wert gespeichert wird. Der initiale Wert dieses Feldes soll innerhalb des Konstruktors gesetzt werden. Betrachten Sie lediglich positive natürliche Zahlen.

c) Implementieren Sie die Interface-Methode `toIntValue`, die die Hexadezimalzahl in das Dezimalsystem umrechnet.

d) Implementieren Sie die Interface-Methode `fromIntValue`, die eine Dezimalzahl in das Hexadezimalsystem umrechnet und das Ergebnis als Wert der Objektinstanz setzt.

e) Testen Sie Ihre Implementierung mithilfe von JUnit Tests. Erstellen Sie pro Methode mindestens zwei Tests mit unterschiedlichen *HexaDecimal*-Objekten.