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

Philipps Universität Marburg

**Companion exercises**
**Objektorientierte Programmierung: Wintersemester 2021/2022**

No. 8, due until 17.01.2022

---

**Attention**: Please use IntelliJ IDE to solve tasks from this sheet onwards. To export your code for handing it in later, please use the built-in functionality of IntelliJ. To use that functionality, follow *File* → *Export* → *Project to Zip File...*

**Task 8.1:** Adam Stevenson or Nicholas Hyde? 3 Points

Caesar's cipher is a cipher where letters of the alphabet are moved by a number of letters to encode a message. The cipher can also be interpreted as an equation of letters. You can, for ecample, set A to E which means that every letter gets moved by 4 letters down the alphabet. This second interpretation should now be implemented. For that purpose, write a class `Caesar` first.

a) Implement a public, static method `String decode(String coded, char a, char b)` that decodes the give cipher by setting *a* to *b*.
b) Test your implementation with JUnit-Tests and the following values:

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

**Task 8.2:** Spells and Witches | 4 Points

As a serious card collector you would want to organize your card collection of the super awesome game *Spells & Witches*. Every card in this game has a name as well as a date of printing. Furthermore, every card has one of three different types: Monster, Spell or Trap. Monsters can also be classified as a Normal Monster or as an Effect Monster.

Your task is to organize your collection by sorting it. Cards are sorted by printing date first, then by type of the card and lastly by their name.

a) Write a class `Card`. `Card` should have fields for Name, Type and the printing date. Implement static final fields for the types of cards within the `Card`-class of type String in such a way that you may call them by `Card.EFFECT_MONSTER` for instance. Also implement Get-methods as needed.

b) Implement the interface `Comparable` into your `Card`-class. Make sure that the discussed order is represented correctly.
   - Cards are sorted by their printing date.
   - Should two cards have the same printing date, they are sorted by type. Types are sorted as follows:

     Normal Monster $\rightarrow$ Effect Monster $\rightarrow$ Spellcard $\rightarrow$ Trapcard
   - If two cards have the same type and the same prin ting date, they should be sorted lexicographically by name.

c) Test your implementation by writing a JUnit-test. Use an array with at least 10 cards, whereas at least 5 cards should be unique. Furthermore, there should be two cards that only have a different printing date.

   Consider that your collection can contain duplicates.

**Task 8.3:** Jonah Hex | 5 Points

Within this task, you are supposed to develop an interface that adds and multiplies natural numbers. Create the interface *Number* and add the following methods:

a) Define the following interface-methods::
   - `int toIntValue()` that returns the value of the instance as `int`.
   - `void fromIntValue(int value)` that sets the value of the instance to value.

b) Implement the following *default*-methods using *toIntValue* and *fromIntValue*:
   - `void add(Number number)` that adds a number to the value of the instance.
   - `void subtract(Number number)` that subtracts a number from the value of the instance.
   - `void multiply(Number number)` that multiplies a number with the value of the instance.
   - `void divide(Number number)` that divides the value of the instance by *number*.

Create a class *HexaDecimal* that implements the *Number*-interface. The class should use a field of type *String* in which the value is saved. The initial value of the field should be set within the constructor. Only positive natural numbers need to be accounted for.

c) Implement the interface-method `toIntValue` that translates a number in hexadecimal into decimal.

d) Implement the interface-method `fromIntValue` that translates a decimal number into hexadecimal and sets it as the new value.

e) Test your implementation using JUnit-tests. Create at least two tests per method using different *HexaDecimal*-objects.