Projektblatt P 7 (24 + 2* P)

Abgabe: Donnerstag 31. Oktober 2024, 12:00h

Entpacken Sie zunächst die Archiv-Datei `vorgaben-p6.zip`, in der sich die Rahmendateien für die Aufgaben befinden. Ergänzen Sie die Rahmendateien zunächst durch Ihren Namen. Ergänzen Sie die Dateien dann durch Ihre Lösungen gemäß der Aufgabenstellung unten. Der hinzuzufügende Java-Sourcecode sollte **syntaktisch richtig** und **vollständig formatiert** sein. Alle Java-Dateien sollten am Ende fehlerfrei übersetzt werden können.
Verpacken Sie die von Ihnen bearbeiteten `.java` Dateien (und bitte nur diese, also keine `.class` Dateien) für Ihre Abgabe in einem ZIP-Archiv mit dem Namen `IhrNachname.IhrVorname.P6.zip`, welches Sie auf Ilias hochladen. Führen Sie dazu in dem Verzeichnis, in dem Sie die Dateien bearbeitet haben, folgenden Befehl auf der Kommandozeile aus:

```
zip  IhrNachname.IhrVorname.P6.zip  *.java
```

Nutzen Sie beim Bearbeiten der Aufgaben nur die Klassen und Methoden aus der Java API, welche explizit durch den Aufgabentext erlaubt sind.

Aufgabe 1: String Verarbeitung 5 (= 1+1+2.5+0.5) P

In dieser Aufgabe sollen Sie ein Programm schreiben, dem beim Aufruf zwei Worte als Argumente von der Kommandozeile übergeben werden und das alle Buchstaben bestimmt sowie in der Konsole ausgibt, die entweder in beiden Worten vorkommen oder in genau einem der Worte. Die Buchstaben sollen dabei jeweils aufsteigend sortiert sein und jeder Buchstabe soll nur einmal genannt werden. Sie dürfen davon ausgehen, dass die Worte nur Buchstaben enthalten.
Die Ausgabe sollte dann wie in den folgenden drei Beispielen aussehen:

```
Word 1: sharp
Word 1: soap
Shared letters: aps
Unique letters of word 1: hr
Unique letters of word 2: o

Word 1: board
Word 1: bored
Shared letters: bdor
Unique letters of word 1: a
Unique letters of word 2: e

Word 1: Happiness
Word 1: Envelope
Shared letters: enp
Unique letters of word 1: ahis
Unique letters of word 2: lo
```

Ergänzen Sie hierzu die Klasse `Letters` wie im Folgenden beschrieben. Sie dürfen hierbei alle Methoden aus den Klassen `String` und `StringBuffer` verwenden.

(a) Schreiben Sie eine öffentliche Klassenmethode mit dem Namen `contains`, die zwei Parameter vom Typ `String` und `char` besitzt und die einen logischen Wert zurückgibt. Die Methode soll prüfen, ob der String, der den ersten Parameter der Methode darstellt, das Zeichen, das den zweiten Parameter der Methode bildet, (mindestens einmal) enthält. Ist dies der Fall, soll die Methode den Wert `true` zurückgeben und ansonsten den Wert `false`.

Überladen Sie die Methode mit einer Version, die zwei Parameter vom Typ `StringBuffer` und `char` besitzt und das Gleiche wie die erste Methode leistet.

(b) Ergänzen Sie die Methode `addLetter` so, dass diese das Zeichen `c`, welches der Methode als Parameter übergeben wird, (lexikalisch) sortiert in den als Parameter übergebenen `StringBuffer` einfügt. Sie dürfen davon ausgehen, dass das Zeichen `c` sowie alls Zeichen in dem `StringBuffer` Objekt Kleinbuchstaben darstellen und dass die aktuell in dem `StringBuffer` Objekt enthaltene Buchstabensequenz sortiert ist.

Hinweis: Suchen Sie durch Vergleich von `c` mit den Zeichen in dem `StringBuffer` Objekt die korrekte Position zum Einfügen. Verwenden Sie die Methode `insert` um ein Zeichen an einer bestimmten Position im `StringBuffer` einzufügen.

(c) In der Methode `checkSharedLetters` sind zunächst drei leere `StringBuffer` Instanzen vorgegeben, in denen Sie die Buchstaben speichern sollen, die in beiden bzw. in genau einem der beiden Worte vorkommen. Dabei soll in jedem der `StringBuffer` Objekte jeder Buchstabe maximal einmal vorkommen und die Buchstaben sollen darüberhinaus lexikalisch aufsteigend sortiert sein.

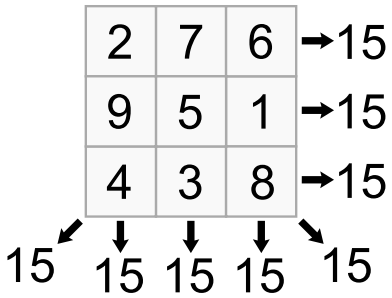
Prüfen Sie nun die beiden als Parameter übergebenen Worte zeichenweise, entscheiden Sie mit Hilfe der in den Aufgabenteilen (a) und (b) implementierten Methoden, ob und in welche `StringBuffer` ein einzelnes Zeichen eingefügt werden muss und führen Sie die Einfügungen ggf. durch.

Ergänzen Sie dann die Ausgabeanweisungen am Ende der Methode so, dass die Inhalte der `StringBuffer` ausgegeben werden.

(d) Ergänzen Sie die `main`-Methode so, dass für den Fall, dass das Programm mit zwei Kommandozeilenargumenten aufgerufen wird, die Methode `checkSharedLetters` mit den beiden Argumenten aufgerufen wird. Hierbei sollen zuvor alle Großbuchstaben in den Strings in Kleinbuchstaben umgewandelt werden. Verwenden Sie hierzu eine Methode aus der Klasse `String`.

Aufgabe 2: Hüllklassen, Strings 7 (=2+1+1+3+2*) P

In der Mathematik versteht man unter einem magischen Quadrat der Ordnung n ein Quadrat der Seitenlänge n, auf dessen Feldern n^2 paarweise verschiedene natürliche Zahlen so platziert werden, dass jede Zeile und jede Spalte sowie die beiden Diagonalen die gleiche Summe ergeben.[1]. Das folgende Bild zeigt ein magisches Quadrat der Ordnung 3.



Sie sollen in dieser Aufgabe nun aus einem String, welcher eine quadratische Anzahl von Zahlen enthält, die durch Kommas voneinander getrennt sind, ein zweidimensionales Feld der Größe $n \times n$ vom Typ `Integer` generieren und dann prüfen, ob dieses ein magisches Quadrat darstellt. Hierbei sollen Sie bei allen Operationen kein Autoboxing bzw. Autounboxing verwenden, sondern nur Operationen aus der Klasse `Integer`.
In der Klasse `MagicSquare` sind bereits zwei Instanzvariablen für die Größe `n` und das Feld vorgegeben, welches das magische Quadrat repräsentiert. Dazu enthält die Klasse mehrere, noch leere und von Ihnen zu ergänzende Methoden und eine `main`-Methode, mit deren Hilfe Sie Ihre Implementierung testen können.
Ergänzen Sie die Klasse `MagicSquare` nun wie im Folgenden beschrieben:

(a) Implementieren Sie zunächst den Konstruktor der Klasse, welcher alle Zahlen aus dem als Parameter übergebenen String (siehe die Beispiele aus der `main`-Methode) extrahiert und diese dazu verwendet, das Feld `mSq` zu initialisieren. Verwenden Sie die Methode `split` aus der Klasse `String`, um die Werte aus dem String-Parameter zu separieren. Verwenden Sie das Feld, dass die Methode `split` liefert, um zunächst den

[1] https://de.wikipedia.org/wiki/Magisches_Quadrat

Wert von `n` zu bestimmen und dann das Feld `mSq` anzulegen und zu füllen. Sie dürfen davon ausgehen, dass die Anzahl der Zahlen in dem String-Parameter eine Quadratzahl darstellt und die Werte dort Zeile für Zeile eingetragen sind.

(b) Implementieren Sie nun die Methode `printSquare`, welche den Inhalt des Felds `mSq` zeilen- und spaltenweise ausgeben soll. Verwenden Sie die Methode `format` aus der Klasse `String`, um jede Zahl rechtsbündig mit drei Ziffern zu formatieren.

(c) Implementieren Sie die Methode `sumArr`, die die Summe der ganzzahligen Werte berechnen und zurückgeben soll, die durch die Objekte repräsentiert werden, welche in dem als Parameter übergebenen Feld gespeichert sind.

(d) Damit das zweidimensionale Feld `mSq` ein magisches Quadrat darstellt, müssen die Summen der Zeilen, Spalten und Diagonalen in dem Feld jeweils den gleichen Wert darstellen. In der Methode `checkSquare` soll dies überprüft werden und die Methode soll das Ergebnis in Form eines logischen Werts zurückgeben (`true`, wenn dies der Fall ist und ansonsten `false`).

Implementieren Sie die Methode nun wie folgt: Speichern Sie nacheinander die Diagonalen, Zeilen und Spalten von `mSq` in einem Feld, berechnen Sie mit Hilfe der Methode aus Aufgabenteil (c) die jeweiligen Summen und überprüfen Sie, ob diese jeweils den gleichen Wert haben. Verwenden Sie hierbei die beiden vorgegebenen Felder `arr1` und `arr2`.

Hinweis: Die Zeilen müssen eigentlich nicht kopiert werden, dam man die Referenzen auf die Zeilen in `mSq` verwenden kann.

(***e***) In einem magischen Quadrat der Größe $n \times n$ müssen alle Werte der Menge $\{1, 2, \ldots, n^2\}$ vorkommen. Implementieren Sie hiezu die Methode `checkSquare2`, die den Wert `true` zurückgeben soll, wenn dies der Fall ist und anderenfalls den Wert `false`. Generieren Sie hierzu für alle ganzahligen Werte in dem Intervall $[1, n^2]$ ein `Integer`-Objekt und prüfen Sie jeweils, ob `mSq` ein Objekt enthält, welches diesen Wert repräsentiert. Verwenden Sie die Methode `equals` um Objekte der Klasse

`Integer` zu vergleichen. Sorgen Sie dafür, dass das Ergebnis zurückgegeben wird, sobald dieses feststeht (entweder wenn die erste Zahl aus dem Intervall nicht gefunden wird oder ansonsten am Ende).

Beim Aufruf der Klasse sollte für nun folgende Ausgabe generiert werden. Falls Sie die Bonusaufgabe (d) nicht implementiert haben, wird für das letzte Beispiel allerdings ein falsches Ergebnis ausgegeben.

```
  2  7  6
  9  5  1
  4  3  8

... is a magic square


 15 10  3  6
  4  5 16  9
 14 11  2  7
  1  8 13 12

... is a magic square


  1  2
  3  4

... is NO magic square


  1  1  1
  1  1  1
  1  1  1

... is NO magic square
```

Aufgabe 3: Date and Time API 12 (=1.5+2+2.5+2+2+2) P

In der Datei `Termin.java` finden Sie eine Klasse `Termin`, welche drei Attributte und einen privaten Default-Konstruktor besitzt und die Sie im Folgenden durch einen zweiten Konstruktor und mehrere Methoden ergänzen sollen, um so die Planung von Terminen zu ermöglichen.
Die Dauer eines Termins wird dabei durch die Klasse `java.time.Duration` repräsentiert. Schauen Sie sich diese Klasse in der API Dokumentation an und lesen Sie sich insbesondere die Beschreibung der Methoden `ofXXX` und `toXXX` durch, mit denen Objekte der Klasse `Duration` generiert bzw. in andere Datentypen umgewandelt werden können.
Hinweis: Die Methode `plus (long amountToAdd, TemporalUnit unit)` aus der Klasse `LocalDateTime` ermöglicht die Verschiebung eines Zeitpunkts um eine bestimmte Anzahl (1. Parameter) von Einheiten (2.Parameter). Um ein `TemporalUnit` Objekt für den 2. Parameter festzulegen, sollten Sie die Konstanten des Aufzählungstyps `java.time.temporal.ChronoUnit` verwenden. Schauen Sie diese ebenfalls in der API Dokumentation nach.

(a) Schreiben Sie zunächst einen Konstruktor, der Parameter für alle zur Initialisierung der Attribute benötigten Informationen besitzt und diese dann entsprechend nutzt. Überlegen Sie sich, welche Informationen Sie sinnvollerweise benötigen und definieren Sie dementsprechend die Parameter. Als Randbedingung soll gelten, dass dem Konstruktor nur Werte übergeben werden, die entweder der Klasse `String` oder einem elementaren Datentyp (wie z.B. `int` angehören).

(b) Implementieren Sie die (leer bzw. nur mit einem Dummy Return ausgestattete) Methode `clone`, um ein Objekt der Klasse `Termin` kopieren zu können. Um Kopien der Attribute `zeit` und `dauer` zu erstellen, dürfen Sie hier nicht die `clone` Methode der Klassen `LocalDateTime` und `Duration` verwenden. Verwenden Sie stattdessen den vorgegebenen privaten Konstruktor der Klasse sowie eine geeignete Kombination der `ofXXX` und `toXXX` Methoden der Klassen `Duration`, `LocalDateTime`, `LocalDate` und `LocalTime`.

Anmerkung: Da Objekte der genannten Klassen und auch die der Klasse `String` immutable (unveränderbar) sind, würde es hier eigentlich reichen, die Referenzen zu kopieren. Zu Übungszwecken sollen Sie hier jedoch, wie oben beschrieben, Kopien anlegen.

(c) Überschreiben Sie die von der Klasse `Object` geerbte Methode `toString`, um eine `String`-Repräsentation eines `Termin`-Objekts zu generieren und als Ergebnis zurückzugeben. Verwenden Sie hierzu eine Instanz der Klasse `StringBuffer` sowie eine Instanz der Klasse `DateTimeFormatter` um die Datums- und Zeitangaben zu formatieren. Der Ergebnisstring sollte wie in folgendem Beispiel aussehen:

```
Termin: Zahnarzt
Beginn: Mittwoch 06.11.2024 15:00h
Ende:   Mittwoch 06.11.2024 16:00h
```

Die Dauer des Termins muss hierbei zur Berechnung eines expliziten Datums für das Ende eines Termins verwendet werden.

(d) Schreiben sie eine Klassenmethode `serieGenerieren`, die eine Zeitserie generiert, d.h. eine Folge von n Terminen, die in einem festen Abstand von jeweils k Tagen voneinander liegen. Die resultierenden Termine sollen in einem Feld gespeichert werden, welches am Ende als Ergebnis zurückgegeben werden soll. Als Parameter sollen der erste Termin sowie die Gesamtzahl der Termine und der Abstand der einzelnen Termine, in Tagen gemessen, definiert sein. Der als Parameter übergebene Termin soll dabei nur als Muster dienen, aber selbst nicht in das Feld übertragen werden. Hinweis: Generieren Sie mit Hilfe des als Muster übergebenen Termins (der den ersten Termin der Serie darstellt, n Kopien dieses Termins, die sie jeweils $i * k$ (mit $i = 0, 1, \ldots, n-1$) Tage verschieben. Verwenden Sie dabei zum Kopieren eines Termins die `clone` Methode aus Aufgabenteil (b).

Ein Beispiel für eine solche Serie könnten die zu einem Semesterkurs gehörigen Termine einer Vorlesung sein.

(e) Schreiben Sie eine Methode `verschieben`, mit der ein Termin verschoben werden kann, d.h. mit der der Zeitpunkt eines `Termin`-Objekts verändert werden kann. Wählen sie passende Parameter und implementieren Sie die Methode so, dass ein Termin um eine vorgegebene Anzahl von Wochen, Tagen, Stunden oder Minuten verschoben werden kann. Die Anzahl und die Einheit sollen dabei als Parameter übergeben werden. Übergeben Sie die Einheit als `String` ("Woche", "Tag", "Stunde" oder "Minute") und bestimmen Sie hierzu in der Methode zunächst jeweils den zugehörigen Wert aus der Klasse `ChronoUnit`.

(f) Ergänzen sie die `main`-Methode der Klasse `Terminplaner`, indem Sie zunächst zwei verschiedene Einzeltermine anlegen:

- für den ersten Termin der PP1 Vorlesung im HWS 24 am 5. September um 12h (90 Min. Dauer) und
- für einen Zahnarztbesuch am 3.12.2024 um 15h (60 Min. Dauer)

Nutzen Sie den ersten Termin zur Erzeugung einer Terminserie mit 14 Veranstaltungen. Verschieben Sie den 5. Termin der Serie um einen Tag zurück (vom 3.10. auf den 2.10.) und geben Sie die Terminserie in der Konsole aus.

Verschieben Sie den Zahnarzttermin dann um sechs Wochen nach hinten und um drei Stunden und dreißig Minuten zurück (z.B. von 3.12. 15h auf den 14.1. um 11:30h). Geben Sie diesen Termin vor und nach der Verschiebung in der Konsole aus.

.
Die Ausgabe sollte dann wie folgt aussehen:

```
1. Termin: Vorlesung PP1
Beginn: Donnerstag, 05.09.2024 12:00 h
Ende:   Donnerstag, 05.09.2024 13:30 h

2. Termin: Vorlesung PP1
Beginn: Donnerstag, 12.09.2024 12:00 h
Ende:   Donnerstag, 12.09.2024 13:30 h

3. Termin: Vorlesung PP1
Beginn: Donnerstag, 19.09.2024 12:00 h
Ende:   Donnerstag, 19.09.2024 13:30 h

4. Termin: Vorlesung PP1
Beginn: Donnerstag, 26.09.2024 12:00 h
Ende:   Donnerstag, 26.09.2024 13:30 h

5. Termin: Vorlesung PP1
Beginn: Mittwoch, 02.10.2024 12:00 h
Ende:   Mittwoch, 02.10.2024 13:30 h
```

```
6. Termin: Vorlesung PP1
Beginn: Donnerstag, 10.10.2024 12:00 h
Ende:   Donnerstag, 10.10.2024 13:30 h

7. Termin: Vorlesung PP1
Beginn: Donnerstag, 17.10.2024 12:00 h
Ende:   Donnerstag, 17.10.2024 13:30 h

8. Termin: Vorlesung PP1
Beginn: Donnerstag, 24.10.2024 12:00 h
Ende:   Donnerstag, 24.10.2024 13:30 h

9. Termin: Vorlesung PP1
Beginn: Donnerstag, 31.10.2024 12:00 h
Ende:   Donnerstag, 31.10.2024 13:30 h

10. Termin: Vorlesung PP1
Beginn: Donnerstag, 07.11.2024 12:00 h
Ende:   Donnerstag, 07.11.2024 13:30 h

11. Termin: Vorlesung PP1
Beginn: Donnerstag, 14.11.2024 12:00 h
Ende:   Donnerstag, 14.11.2024 13:30 h

12. Termin: Vorlesung PP1
Beginn: Donnerstag, 21.11.2024 12:00 h
Ende:   Donnerstag, 21.11.2024 13:30 h

13. Termin: Vorlesung PP1
Beginn: Donnerstag, 28.11.2024 12:00 h
Ende:   Donnerstag, 28.11.2024 13:30 h

14. Termin: Vorlesung PP1
Beginn: Donnerstag, 05.12.2024 12:00 h
Ende:   Donnerstag, 05.12.2024 13:30 h

Termin: Zahnarzt
```

```
Beginn: Dienstag, 03.12.2024 15:00 h
Ende:   Dienstag, 03.12.2024 16:00 h

Termin: Zahnarzt
Beginn: Dienstag, 14.01.2025 11:30 h
Ende:   Dienstag, 14.01.2025 12:30 h
```