# Übungsblatt 3

## 1 - Klasse `Box`

Schreibe eine Klasse `Box`, die eine Kiste in Form eines Quaders beschreibt. Ein Quader ist ein geometrischer Körper mit sechs rechteckigen Flächen, bei dem alle Winkel 90 Grad haben. Ein Quader lässt sich durch drei Angaben spezifizieren:
Breite, Höhe und Tiefe. Nutzen Sie dazu reelle Zahlen des Typs `double`.

Schreibe für die Klasse `Box` verschiedene Methoden:

- Der Konstruktor setzt die Breite, Höhe und Tiefe der Kiste.
- Drei Methoden, die die Breite, Höhe und Tiefe zurückgeben:`getWidth, getHeight, getDepth`),
- Drei Methoden, die das Volumen (`getVolumeSize`), die Oberfläche (`getAreaSize`) und die Summe der Kantenlängen (`getEdgesLength`) berechnen und zurückgeben,
- Die Methode `boolean isCube(), die` den Wert `true` zurückgibt, falls es sich um einen *Würfel* handelt, also alle Kanten die gleiche Länge besitzen.
- Die Methode `int compareTo( Box f )`, die zwei Kisten miteinander vergleicht, das ausführende Objekt und das als Parameter übergebene Objekt. Als Ergebnis wird zurückgegeben:
  – eine Wert größer als `0`, falls das ausführende `Box`-Objekt ein größeres Volumen als die als Argument übergebene `Box` besitzt,
  – der Wert `0`, falls beide Kisten das gleiche Volumen besitzen,
  – eine Wert kleiner als `0`, falls die ausführende `Box` ein kleineres Volumen als die als Argument übergebene `Box` besitzt.
- Die Methode `encloses` ermittelt, ob eine als Argument übergebene Kiste bei *parallel liegenden* Seitenflächen vollständig in das ausführende Objekt *echt* passt. Entsprechend wird der Wert `true` oder `false` zurückgegeben.
  Beachte bei der Implementierung, dass Kisten gedreht werden können: Eine Kiste mit den Maßen *30x20x10* passt echt in eine Kiste mit den Maßen *11x31x21*.

## 2 - Klassen `Period`, `PointInTime` und `Date`

*Klasse* `Period`

Definiere eine Klasse `Period`, die als einziges Attribut eine Zeitangabe in Minuten im Attribut `minutes` verwaltet. Eine negative Dauer soll nicht möglich sein.

Implementiere in `Period`::

- Zwei Konstruktoren, die eine Parameter (nur Minuten) und zwei Parameter (Minuten und Stunden) besitzen. Werden negative Argumente angegeben, wird ersatzweise `0` zur Initialisierung verwendet.
- Die Methode `getMinutes`, die adie Anzahl der abgelegten Minuten zurückgibt.
- Die Methode `getHours`, die die Zahl der in den abgelegten Minuten enthaltenen ganzen Stunden zurückgibt.
- Die Methode `getMinorMinutes`, die den Wert der nicht in ganzen Stunden enthaltenen Minuten zurückgibt, so dass gilt: `getHours()*60+getMinorMinutes()==minutes`.
- Eine Methode `toString`, die die Dauer als Text der Form *hh:mm* zurückgibt.
- Eine Methode `clone`, die eine *Kopie* des `Period`-Objektes erstellt und zurückgibt.
- Eine Methode `change`, die eine als Argument übergebene Anzahl von Minuten zu `minutes` hinzuaddiert. Wird der Methode `change` ein negativer Wert übergeben, soll nichts geschehen.

*Klasse* `PointInTime`

Ein Zeitpunkt wird durch drei Attributen beschrieben:

- einer Jahreszahl,
- der Nummer eines Tages im Jahr und
- einer Stundenangabe.

Die Jahreszahl soll immer positiv und vierstellig (also größer als *999* und kleiner als *10000*) sein. Schaltjahre werden nicht berücksichtigt, so dass der Tag zwischen *1* und *365* und die Stundenangabe zwischen *0* und *23* liegen dürfen. Die Klasse `PointInTime` besitzt einen geeigneten Konstruktor und drei Methoden zum Zurückgeben der drei Attribute bereitstellen.

Zusätzlich bietet soll die Klasse `PointInTime` folgende Methoden anbieten:

- Die Methode `toString`, die den Zeitpunkt als Text der Form *jjjj/ttt/hh* zurückgibt.
- Die Methode `clone`, die eine Kopie des `PointInTime`-Objektes erstellt und zurückgibt.
- Eine Methode `change`, die eine als Argument übergebene Anzahl von Stunden zu dem Zeitpunkt hinzuaddiert. Wird dieser Methode ein negativer Wert übergeben, soll nichts geschehen.

*Klasse* `Date`

Ein Termin besteht aus einer Bezeichnung, einem Startzeitpunkt und einer Dauer. Implementieren Sie die Klasse `Date` unter Zuhilfenahme der Klassen `PointInTime` und `Period`.

Die Klasse `Date` besitzt einen Konstruktor mit je einem Parameter für die Bezeichnung, den Zeitpunkt und die Dauer. Es soll möglich sein, für den Zeitpunkt oder die Dauer kein Objekt - sondern den Wert `null` – zu übergeben. Dann soll der Termin einen unbestimmten Beginn oder eine unbestimmte Dauer besitzen.

Weiterhin bietet die Klasse `Date` die folgenden Methoden an:

- Die Methode `clone` zum Kopieren eines Termins.
- Drei Methoden zum Zurückgeben der drei Attribute.
- Die Methode `set`, die einen Termin ändert. Diese Methode soll zwei Parameter für den Zeitpunkt und die Dauer besitzen, die die entsprechenden Werte des Objekts überschreiben.
- Die Methode `change`, die einen Termin verschiebt. Diese Methode soll einen Parameter des Typs `int` besitzen, der die Anzahl der Stunden angibt, um die der Startzeitpunkt des Termins verschoben wird. Beachten Sie, dass Termine einen unbestimmten Beginn besitzen können. Dann ist keine Verschiebung möglich.
- Die Methode `toString`, die eine geeignete Darstellung der Attribute des Termins als `String` zurückgibt.

## Aufgabe 3 - Game of Life

Implementiere eine Klasse `GameOfLife`, welche als Attribut ein zweidimensionales Feld `population` mit dem Grundtyp `boolean` besitzt. Die Elemente dieses Feldes werden als *Zellen* interpretiert werden, die *leben* – dann besitzt das Element den Wert `true` – oder *tot* sind – dann besitzt das Element den Wert `false`. Vorgegebene Regeln bestimmen, wann eine *Zelle* lebendig wird, lebendig bleibt oder stirbt. Weitere Informationen finden sich auch unter:

`de.wikipedia.org/wiki/Conways_Spiel_des_Lebens`

Es sollen folgende Regeln implementiert werden:

- Eine tote *Zelle* wird lebendig, wenn sie genau drei lebende *Nachbarzellen* hat.
- Eine lebende *Zelle* bleibt lebendig, wenn sie zwei oder drei lebende *Nachbarzellen* hat.
- Eine lebende *Zelle* stirbt an Vereinsamung, wenn sie weniger als zwei lebende *Nachbarzellen hat.*
- Eine lebende *Zelle* stirbt an Überbevölkerung, wenn sie mehr als drei lebende *Nachbarzellen* hat.
- Alle Regeln werden immer zeitgleich auf alle *Zellen* angewandt.

Implementiere vier Methoden, die einander benutzen sollen:

- Die Methode `boolean nextState( int i, int j )` bestimmt den Folgezustand für die *Zelle* `population[i][j]` aufgrund der Belegungen ihrer *Nachbarzellen* in dem Feld `population`.
- Die Methode `void nextGeneration()` trägt für das Attribut `population` die nächste Generation von *Zellen* ein, nachdem die Methode `nextState` auf alle *Zellen* des Feldes `population` angewandt worden ist. Beachte, dass die Zusammenstellung der neuen Generation von Zellen vollständig aus der vorangehenden Generation bestimmt werden muss.
- Die Methode `void futureGeneration( int n )` erzeugt ein neues Feld, das den Zustand der *Zellen* enthält, nachdem die Methode `nextGeneration` n-mal auf das Feld `population` angewandt worden ist.
- Die Methode `void show()` gibt das Feld `population` in einer übersichtlichen rechteckigen Darstellung aus.

## Aufgabe 4 - Klasse `MultipleStrings`

In dieser Aufgabe sollst Du das Benutzen einer bereits vorliegenden Klasse üben. Die Klasse `String` ist in Teilen schon in der Vorlesung vorgestellt worden. Eine vollständige Beschreibung aller in `String` verfügbaren Methoden findest Du unter:
`https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/String.html`

Konzipiere die Klasse `MultipleStrings`, die ein Feld mit Texten `String[] texts` verwaltet. Implementiere für die Klasse `MultipleStrings` die folgenden Methoden auf der Basis der in der Klasse `String` bereits verfügbaren Methoden:

- Den Konstruktor `MultipleStrings( int n )`, der für `texts` ein Feld der Länge `Math.abs(n)` erzeugt.
- Die Methode `boolean addAtPosition( String s, int p )`, die den Text `s` am Index `p` in das Feld `texts` einfügt. Bereits vorhandene Inhalte sollen dabei überschrieben werden. Ist `p` kein gültiger Index, soll nichts geschehen und `false` zurückgegeben werden; sonst soll `true` zurückgegeben werden.
- Die Methode `int size()`, die die Anzahl der über `texts` erreichbaren Texte zurückgibt.
- Die Methode `int atEnd( String end )`, die zurückgibt, wie häufig der Text `end` als Teiltext am Ende der über `texts` erreichbaren Texte auftritt.
- Eine Methode `boolean inAll( char ch )`, die `true` zurückgibt, falls das Zeichen `ch` in jedem der über `texts` erreichbaren Texte vorkommt. Sonst soll `false` zurückgegeben werden.
- Eine Methode `void replace( char ch, char rep )`, die diejenigen über `texts` erreichbaren Texte, die das Zeichen `ch` enthalten, durch einen Text ersetzt, in dem `ch` durch `rep` ersetzt ist.