Übungseinheit 27. November - 1. Dezember 2017

Themen:

- Vertiefung der Verwendung bekannter Konzepte wie der Klassen `String` und `ArrayList`
- Neue Methoden von ArrayList: `add(int index, E element)` und `remove(int index)`
- Algorithmisch etwas schwierigere Methoden

Aufgabe 1

Schreiben Sie eine Klasse `Perle` mit einem Konstruktor

```
public Perle(String farbe, double durchmesser),
```

der eine Perle der angegebenen Farbe mit entsprechendem Durchmesser erstellt.

Schreiben Sie weiters eine Klasse `Perlenkette` mit Konstruktor

```
public Perlenkette(String farbe1, double durchmesser1,
                   String farbe2, double durchmesser2),
```

der eine Kette ohne Perlen erstellt. Dabei dürfen der Perlenkette nur zwei mögliche Typen von Perlen hinzugefügt werden, nämlich solche mit `farbe1` und `durchmesser1` oder aber Perlen mit `farbe2` und `durchmesser2`.

```
public boolean addPerle(Perle naechstesGlied)
```

fügt der Kette die Perle `naechstesGlied` hinzu, vorausgesetzt diese ist von erlaubtem Typ (siehe Konstruktor). Weiters soll beachtet werden, dass die Perle `naechstesGlied` nicht vom selben Typ ist wie die zuletzt der Kette hinzugefügte Perle. Falls die Perle `naechstesGlied` hinzugefügt wurde, gibt die Methode `true` zurück, andernfalls `false`.

```
public boolean hatGleichViele()
```

gibt `true` zurück, falls die Kette aus jeweils gleich vielen Perlen der zwei erlaubten Perlentypen besteht.

```
public boolean kannGeschlossenWerden(int mindestlaenge)
```

gibt `true` zurück, falls die Kette aus mindestens so vielen Perlen besteht wie in der Mindestlänge festgelegt wurde und die erste und die letzte Perle unterschiedlichen Typs sind.

Sie können den jeweiligen Klassen weitere Methoden und/oder Attribute hinzufügen, wenn Ihnen das sinnvoll oder notwendig erscheint.

Aufgabe 2

Schreiben Sie eine Klasse `Aktienposition` mit dem Konstruktor

```
public Aktienposition(String bezeichnung)
```

`bezeichnung` enthält den Namen der AG, die diese Aktie ausgibt. Darüber hinaus soll jede Aktienposition über folgende Informationen verfügen: `kurs` gibt den aktuellen Kurs an; `anzahl` gibt an, wieviele Aktien dieser `bezeichnung` vorhanden sind; `kursaenderung` gibt an, um wieviel der Kurs der Aktie in der letzten Woche gefallen (`kursaenderung` < 0) oder gestiegen ist (`kursaenderung` > 0). Wird eine neue `Aktienposition` angelegt, so soll `kurs` eine zufällig gewählte `double`-Zahl aus dem Intervall $[10, 100)$, `anzahl` eine zufällig gewählte `int`-Zahl aus dem Intervall $[100, 1000]$ und `kursaenderung` eine zufällig gewählte `double`-Zahl aus dem Intervall $[-5, 5)$ sein.

Überlegen Sie, nachdem Sie die gesamte Aufgabenstellung verstanden haben, über welche Methoden diese Klasse verfügen sollte!

Schreiben Sie weiters eine Klasse `Aktiendepot` mit dem Konstruktor

```
public Aktiendepot(),
```

der ein Depot ohne Aktienpositionen erzeugt. Implementieren Sie auch die folgenden Methoden:

```
public void einkauf(Aktienposition neueAktien)
```

fügt `neueAktien` dem Depot hinzu, wenn diese Aktie im Depot noch nicht vorhanden ist. (Hinweis: Zwei Aktien gelten als gleich, wenn sie dieselbe `bezeichnung` haben!) Ist eine Aktie im Depot bereits vorhanden, so soll ihre `anzahl` entsprechend erhöht werden.

```
public ArrayList<Aktienposition> getNiedrigerKursUndPotential(double
niedrig)
```

liefert eine ArrayList mit allen im Depot befindlichen Aktien mit einem Kurs $\leq$ `niedrig` zurück, die während der letzten Woche eine positive Entwicklung nahmen.

```
public double gibLetzteWocheDurchschnitt()
```

gibt die durchschnittliche Kursentwicklung der Aktien in der letzten Woche zurück. Bei der Berechnung des Durchschnittswertes ist die Anzahl der Aktien in einer Aktienposition zu berücksichtigen! Beispiel: Von der Aktie 'Ramsch' seien in der entsprechenden Aktienposition 23 Stück vorhanden; `kursaenderung` sei mit -3.17 gegeben. Der Beitrag dieser Aktienposition zum Durchschnittswert beträgt dann nicht -3.17 sondern $-3.17 * 23$. Natürlich ist dann schlussendlich auch durch die Summe aller vorhandenen Aktien und nicht durch die Anzahl der Aktienpositionen zu dividieren!

```
public int zaehleSchlechte()
```

gibt die Anzahl all jener Aktienpositionen zurück, die in der letzten Woche eine negative Entwicklung nahmen.

Hinweis: Diese Aufgabe kann mit dem Testsystem nicht überprüft werden.

Aufgabe 3

Schreiben Sie eine Klasse `TagesTemp` mit dem Konstruktor

```
public TagesTemp(String datum, double temperatur)
```

und den folgenden `setze-` und `gib-`Methoden:

```
public String gibDatum(),
public double gibTemp() und
public void setzeTemp(double neueTemp).
```

Verfassen Sie auch eine Klasse `Wetterdaten` mit dem Konstruktor

```
public Wetterdaten(),
```

der ein `Wetterdaten`-Objekt mit einer leeren ArrayList `temperaturListe<TagesTemp>` erzeugt. Implementieren Sie auch die folgenden Methoden:

```
public void addTag(TagesTemp einTag)
```

fügt `einTag` der Temperaturliste hinzu, wobei nicht zu überprüfen ist, ob dieser Tag in der Liste schon enthalten ist. Die Methode

```
public double getMittlereTemp()
```

soll das arithmetische Mittel aller in `temperaturListe` gespeicherten Temperaturen zurückgeben. Die Methode

```
public boolean zuWarm(TagesTemp einTag)
```

soll `true` zurückgeben, falls die für `einTag` gemessene Temperatur um mindestens 3 Grad über der mittleren Temperatur liegt, sonst `false`.

Aufgabe 4

Schreiben Sie eine Klasse `TaxiBot` mit dem Konstruktor

```
public TaxiBot(RobotSE robi)
```

der eine zu steuernden Roboter in ein Attribut speichert.

Ein `TaxiBot` soll auch über die Methoden

```
public int getZeile(),
public int getSpalte(),
public int getDistanz(int zeile, int spalte),
public void fahreZu(int zeile, int spalte) und
public void einsteigen()
```

verfügen. Die Methoden `getZeile` und `getSpalte` liefern die aktuelle Position des Taxis in der Stadt zurück. Der Aufruf von `getDistanz` gibt die Anzahl der Schritte (ohne Drehungen) zurück, die der `TaxiBot` machen muss, um von seiner derzeitigen Position die angegebene Kreuzung zu erreichen. Die Methode `fahreZu` lässt den `TaxiBot` zur angegebenen Kreuzung fahren. Die letzte Methode `einsteigen` lässt einen Fahrgast (ein `Thing`) in das Taxi einsteigen (`pickThing()`).

Schreiben Sie weiters eine Klasse `Taxiflotte` mit dem Konstruktor

```
public Taxiflotte(City city)
```

der für die angegebene Stadt eine Taxiflotte erzeugt, die allerdings noch keine Taxis enthält (leere `ArrayList`). `Taxiflotte` soll auch eine statische Methode `public static void erzeugeFlotte()` enthalten, die mehrere Instanzen von `TaxiBot` erzeugt und in einer `City` an zufällig gewählte Kreuzungen (i, j) platziert, wobei $0 \leq i \leq 10$, $0 \leq j \leq 10$.

Mit der Methode

```
public boolean add(TaxiBot taxi)
```

kann der Taxiflotte das angegebene `taxi` hinzugefügt werden, allerdings nur wenn es an den Koordinaten von `taxi` noch kein anderes Taxi gibt. Bei erfolgreichem Hinzufügen liefert die Methode `true`, ansonsten `false`. Die Methode

```
public void holeAb(int zeile, int spalte)
```

soll die Taxiflotte veranlassen, an der angegebenen Kreuzung eine Person als `Thing` zu erzeugen, das der Kreuzung am nächsten liegende Taxi der Flotte zu der Kreuzung zu schicken, und die Person aufzunehmen.

Aufgabe 5

Schreiben Sie eine Klasse `Mannschaft` mit Konstruktor

```
public Mannschaft(String name),
```

der eine Mannschaft mit angegebenem Namen und 0 Punkten und Tordifferenz 0 erstellt.

Schreiben Sie weiters eine Methode

```
public void addPunkte(int punkte),
```

die der Mannschaft `punkte` zusätzliche Punkte zuweist, sowie eine Methode

```
public void changeTordifferenz(int tordiff),
```

die die Tordifferenz der Mannschaft um `tordiff` ändert. Natürlich benötigen wir in `Mannschaft` auch die Methoden

```
public String getName(),
```

```
public int getPunkte() und

public int getTordifferenz(),
```

um auf die Attribute dieser Klasse zugreifen zu können.

Schreiben Sie weiters eine Klasse `Tabelle` mit dem Konstruktor

```
public Tabelle(ArrayList<String> dieMannschaften),
```

der eine Tabelle von Mannschaften mit den in `dieMannschaften` angegebenen Namen erstellt, wobei jede Mannschaft 0 Punkte und Tordifferenz 0 aufweist.

Implementieren Sie in der Klasse `Tabelle` auch die folgenden Methoden:

```
public void werteSpielAus(String m1Name, String m2Name,
int tore1, int tore2)
```

wertet ein Spiel von Mannschaft `m1Name` gegen Mannschaft `m2Name` mit `tore1` Toren für Mannschaft `m1Name` und `tore2` Toren für Mannschaft `m2Name` in der Tabelle aus. D.h. Torverhältnis und Punkte der beiden Mannschaften ändern sich entsprechend, wobei es für einen Sieg drei Punkte und für ein Unentschieden einen Punkt gibt. Weiters sollen die Positionen der beiden Mannschaften in der Tabelle gegebenenfalls so abgeändert werden, dass auch nach dem Spiel alle Mannschaften absteigend nach Punkten und im Falle eines Punktegleichstands absteigend nach Torverhältnis sortiert sind.

Hinweis: Eine Möglichkeit, besteht darin, die Mannschaften zunächst aus der Tabelle zu löschen und sie anschließend an der richtigen Stelle wieder einzufügen. Dazu können Sie eine Hilfsmethode

```
private void addMannschaft(Mannschaft m)
```

schreiben, die die Mannschaft `m` ihrem Punktestand und ihrer Tordifferenz entsprechend in die Tabelle einfügt.

Eine weitere Methode

```
public ArrayList<Mannschaft> getTabelle()
```

soll die Tabelle wie oben angegeben sortiert zurückgeben.

Aufgabe 6

Schreiben Sie eine Klasse `Handymast` mit dem Konstruktor

> public Handymast(String bezeichnung, double xKoordinate, double yKoordinate, int kapazitaet).

Dabei ist eine Handymast durch seine Bezeichnung eindeutig gegeben, und `xKoordinate` und `yKoordinate` geben die Position des Masts in einem (kartesischen) Koordinatensystem an. Die Kapazität bezeichnet die maximale Anzahl von Gesprächen, die gleichzeitig über diesen Mast geführt werden können.

Ergänzen Sie die Klasse `Handymast` um geeignete weitere Methoden.

Schreiben Sie weiters eine Klasse `Handynetz` mit einem parameterlosen Konstruktor und folgenden Methoden:

> public Handynetz()

erzeugt ein Handynetz ohne Handymasten.

> public boolean add(Handymast mast)

fügt dem Handynetz den Handymast `mast` hinzu, aber nur dann, wenn das Netz noch keinen Mast mit gleicher Bezeichnung enthält. In diesem Fall liefert die Methode `true`, ansonsten `false`.

> public boolean addKapazitaet(String bezeichnung, int zusatzkapazitaet)

erhöht die Kapazität des Handymasts mit der angegebenen Bezeichnung um `zusatzkapazitaet`, sofern das Handynetz einen Mast mit dieser Bezeichnung enthält. In diesem Fall liefert die Methode `true`, ansonsten `false`.

> public int entferne(int mindestKapazitaet)

entfernt aus dem Handynetz alle Handymasten, deren Kapazität unter der angegebenen Mindestkapazität liegt. Die Anzahl der entfernten Handymasten soll zurückgegeben werden.

> public int getAnzahlGespraeche(double xKoordinate, double yKoordinate, double radius)

liefert die Summe der Gespräche zurück, die über jene Handymasten des Handynetzes geführt werden können, die sich innerhalb des angegebenen Radius um die durch x- und y-Koordinate gegebene Position befinden.