

Modellierung und Programmierung 1

Übung 7

Stefan Preußner

14./ 15. Dezember 2020

Organisatorisches

- Einstellung des Lehrbetriebs vom 17.12. bis zum 10.01.
- Der Übungsbetrieb läuft bis zum 16.12. normal weiter
- Der Abgabetermin der Serie 3 (16.12., 22:00 Uhr) bleibt bestehen!
- Die Abgabetermine für die Serien 4 und 5 werden angepasst

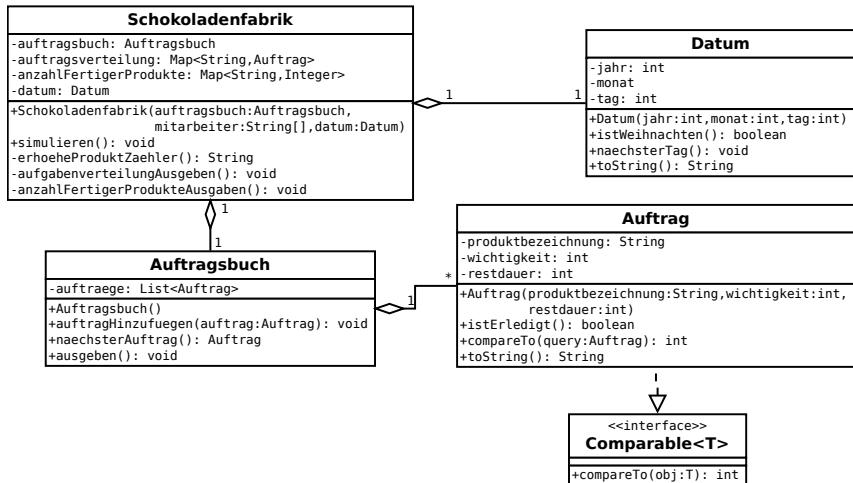
Java-Programmierung - List, Map, Comparable



Programmierübung:

Simulation der Produktion von Süßigkeiten in einer
Schokoladenfabrik in den Tagen vor Weihnachten

Java-Programmierung - List, Map, Comparable



Die Klasse Datum

- `istWeihnachten` gibt `true` zurück, wenn es der 24.12. eines beliebigen Jahres ist
- `naechsterTag` ändert das Datum auf den nächsten Tag und berücksichtigt dabei Monats- bzw. Jahreswechsel (aber keine Schaltjahre)
- `toString` gibt das Datum im Format Tag.Monat.Jahr zurück

Die Klasse Auftrag

- Die Restdauer muss bei der Erzeugung eines neuen Auftrags mindestens 1 betragen
- Die Wichtigkeit muss einen Wert zwischen 0 und 3 haben
- `istErledigt` gibt `true` zurück, wenn die Restdauer 0 ist
- `compareTo` soll so implementiert werden, dass Aufträge nach absteigender Wichtigkeit sortiert werden. Bei gleicher Wichtigkeit soll nach absteigender Restdauer sortiert werden.
- `toString` gibt die Produktbezeichnung, Wichtigkeit und Restdauer als String zurück

Die Klasse Auftragsbuch

- `auftragHinzufuegen` fügt den übergebenen Auftrag zur Liste aller Aufträge hinzu und sortiert die Liste anschließend
- `naechsterAuftrag` entfernt den ersten Auftrag aus der Auftragsliste und gibt ihn zurück. Ist die Auftragsliste leer, dann soll `null` zurückgegeben werden.
- `ausgeben` gibt alle Aufträge aus

Die Klasse Schokoladenfabrik

- simulieren simuliert tagesweise den Zeitraum zwischen datum und Weihnachten
 - Jedem Mitarbeiter wird, solange das Auftragsbuch noch Aufträge enthält, ein Auftrag zugewiesen
 - Bei jedem Auftrag, der einem Mitarbeiter zugewiesen wurde, verringert sich die Restdauer jeden Tag um einen Tag
 - Ist ein Auftrag erledigt, wird dem Mitarbeiter ein neuer Auftrag zugewiesen und der Zähler für das entsprechende Produkt um 1 erhöht
 - Es werden täglich die Auftragsverteilung und Zahl der insgesamt fertiggestellten Produkte ausgegeben

Die Klasse Schokoladenfabrik

- `auftragsverteilung` bildet jeden Mitarbeiter (repräsentiert durch seinen Namen) auf einen Auftrag ab
- `anzahlFertigerProdukte` gibt an, wie oft ein Produkt (repräsentiert durch seine Bezeichnung) bereits fertiggestellt wurde
- `erhoeheProduktZaehler` erhöht den Zähler für ein fertiggestelltes Produkt um 1
- `aufgabenverteilungAusgeben` gibt die aktuelle Aufgabenverteilung aus
- `anzahlFertigerProdukteAusgaben` gibt den Produktzähler aus

char

- char ist ein primitiver Datentyp mit einer Größe von 2 Bytes
- Ein char kann einen Wert zwischen 0 und 65535 ($2^{16} - 1$, da 2 Bytes = 16 Bits) annehmen und verhält sich wie eine vorzeichenlose Zahl
 - → chars können wie Zahlen addiert, subtrahiert etc. werden
 - Das Ergebnis der Addition zweier chars ist ein int (!)
- Die Zuordnung eines chars zu einem bestimmten Buchstaben/Zeichen/Symbol hängt vom verwendeten Zeichensatz (z.B. UTF-8) ab

char[]

- Ein Array von chars, `char []`, verhält sich wie ein Array von Zahlen
- Zwei Arrays können mit der Funktion `Arrays.equals(char [], char [])` aus dem Paket `java.util` verglichen werden
 - → Als primitiver Datentyp hat ein `char []` selbst keine `equals`-Funktion
- Arrays sind **veränderlich** (*mutable*) - einzelne chars im Array können beliebig durch andere ersetzt werden

CharSequence


- CharSequence ist ein **Interface**
- Die Schnittstelle wird u.a. von `String` und `StringBuilder` implementiert
 - Einige Methoden in diesen Klassen akzeptieren als Parameter Objekte von allen Klassen, die dieses Interface implementiert haben

CharSequence



- Die in CharSequence deklarierten Methoden sind
 - `charAt(int)` - gibt den char an der angegebenen Stelle zurück
 - `length()` - gibt die Länge der Sequenz zurück
 - `subSequence(int start, int ende)` - gibt eine neue CharSequence zurück, die alle Zeichen zwischen der (mit eingeschlossenen) Position start und der (nicht mit eingeschlossenen) Position ende der ursprünglichen CharSequence enthält
 - `toString()` - gibt die in der CharSequence gespeicherte Zeichenfolge als String zurück

String

- String hat in Java eine Sonderstellung: es ist sowohl eine Klasse als auch ein eingebauter Datentyp
 - String-Objekte müssen (im Gegensatz zu allen anderen Objekten) nicht mit `new` erzeugt werden
 - Strings können mit dem `+`-Operator aneinandergefügt (konkateniert) werden, dieser Operator ist sonst primitiven Datentypen vorbehalten
- Die in String-Objekten gespeicherte Zeichenkette ist **unveränderlich** (immutable) 
 - Funktionen, die Zeichen in Strings verändern, erzeugen immer neue String-Objekte

Einige wichtige Funktionen der Klasse String:

- `char charAt(int index)` und `int length()` aus dem `CharSequence`-Interface

```
String s = "MuP ist toll!";  
char c = s.charAt(2);    // c == 'P'  
int l = s.length();     // l == 13
```

- `String substring(int start, int ende)` - wie `subSequence`, nur gibt `substring` einen `String` zurück. Wichtig: Groß-/Kleinschreibung bei `subSequence`/`substring` beachten!

```
String s = "MuP ist toll!";  
String subs = s.substring(5,10);    // subs ist "st to"
```

Einige wichtige Funktionen der Klasse String:

- `boolean equals(Object obj)` - testet zwei Strings auf Gleichheit unter Berücksichtigung von Groß-/Kleinschreibung

```
String s = "MuP ist toll!";  
String t = "mup ist toll!";  
String u = "MuP ist toll!";  
boolean s_gleich_t = s.equals(t);    // false  
boolean s_gleich_u = s.equals(u);    // true
```



- `equals` übernimmt zwar beliebige Objekte, kann aber nur `true` zurückgeben, wenn `obj` auch ein `String` ist
- Wichtig: der `==`-Operator testet zwei `String`-Objekte auf Identität, nicht auf die Gleichheit der gespeicherten Zeichenketten. `s == t` gilt für zwei Strings also nur dann, wenn `s` und `t` ein und das selbe Objekt sind.

Einige wichtige Funktionen der Klasse String:

- `boolean equalsIgnoreCase(String str)` - testet zwei Strings auf Gleichheit und ignoriert dabei Groß-/Kleinschreibung

```
String s = "MuP ist toll!";  
String t = "mup ist toll!";  
String u = "MuP ist toll!";  
boolean s_gleich_t = s.equalsIgnoreCase(t);    // true  
boolean s_gleich_u = s.equalsIgnoreCase(u);    // true
```

Einige wichtige Funktionen der Klasse String:

- `int compareTo(String str)` - vergleicht zwei Strings lexikographisch unter Berücksichtigung der Groß-/Kleinschreibung
 - `s.compareTo(t)` gibt eine Zahl ≤ -1 zurück, wenn `s` lexikographisch kleiner als `t` ist
 - `s.compareTo(t)` gibt 0 zurück, wenn `s` gleich `t` ist
 - `s.compareTo(t)` gibt eine Zahl ≥ 1 zurück, wenn `s` lexikographisch größer als `t` ist

```
String s = "MuP ist toll!";  
String t = "mup ist toll!";  
String u = "MuP ist toll!";  
String v = "A&D ist toll!";  
int s_compared_t = s.compareTo(t);    // -32  
int s_compared_u = s.compareTo(u);    // 0, da Strings gleich  
int s_compared_v = s.compareTo(v);    // 12
```

Einige wichtige Funktionen der Klasse String:

- `int compareTo(String str)` - durch die bereits vorhandene Implementierung von `compareTo` können alle Collections (List, ArrayList, HashSet, ...) von String mit `Collection.sort()` sortiert werden. Die von `s.compareTo(t)` zurückgegebene Zahl ergibt sich wie folgt:
 - Unterscheiden sich s und t an der Stelle k, wird `s.charAt(k) - t.charAt(k)` zurückgegeben
 - Unterscheiden sich s und t an keiner Stelle, wird `s.length() - t.length()` zurückgegeben
 - Hierdurch hat der kürzere String bei der Sortierung immer Vorrang
 - Sind beide Strings gleich lang, wird 0 zurückgegeben

Einige wichtige Funktionen der Klasse String:

- `int indexOf(int ch)` - gibt die Position des ersten Auftretens des Zeichens `ch` zurück, oder `-1`, falls das Zeichen nicht auftritt
- `int indexOf(int ch, int start)` - wie `indexOf(int ch)`, beginnt mit der Suche an der Position `start`
- `int lastIndexOf(int ch)`, `int lastIndexOf(int ch, int start)` - wie `indexOf`, sucht von rechts nach links

```
String s = "MuP ist toll!";  
int t = s.indexOf('i');      // 4  
int u = s.indexOf('t', 7);   // 8  
int v = s.indexOf('x');      // -1  
int w = s.lastIndexOf('l');  // 11  
System.out.println(t);  
System.out.println(u);  
System.out.println(v);  
System.out.println(w);
```

Einige wichtige Funktionen der Klasse String:

- `indexOf(String str)` - wie oben, übernimmt `String` statt `char`
- `contains(CharSequence s)` - gibt `true` zurück, wenn die gesuchte `CharSequence s` enthalten ist

```
String s = "MuP ist toll!";  
String t = "P ist t";  
String u = "mup";  
String v = "xyz";  
boolean s_contains_t = s.contains(t); // true  
boolean s_contains_u = s.contains(u); // false  
boolean s_contains_v = s.contains(v); // false  
System.out.println(s_contains_t);  
System.out.println(s_contains_u);  
System.out.println(s_contains_v);
```

Einige wichtige Funktionen der Klasse String:

- `String toLowerCase()` - gibt den String in Kleinbuchstaben zurück
- `String toUpperCase()` - gibt den String in Großbuchstaben zurück

```
String s = "MuP ist toll!";  
String t = s.toLowerCase(); // mup ist toll!  
String u = s.toUpperCase(); // MUP IST TOLL!  
System.out.println(t);  
System.out.println(u);
```

Einige wichtige Funktionen der Klasse String:

- `String replace(char orig, char repl)` - gibt einen neuen String zurück, in dem jedes Auftreten von `orig` durch `repl` ersetzt wurde. Der ursprüngliche String wird von links nach rechts prozessiert (wichtig, wenn mehrere ersetzbare Zeichen direkt aufeinander folgen).
- `String replace(CharSequence orig, CharSequence repl)` - methodisch identisch zur obigen Funktion, nur anderer Datentyp des Parameters

```
String s = "MuP ist toooll!";  
String t = s.replace('o', 'z'); // MuP ist tzzzll!  
String u = s.replace("oo", "z"); // MuP ist tzoll!  
System.out.println(t);  
System.out.println(u);
```

Einige wichtige Funktionen der Klasse String:

- static String valueOf(*) - erzeugt die Stringrepräsentation für beliebige Objekte und alle primitiven Datentypen
 - Bei Objekten wird die Methode toString() aufgerufen
 - Da die Methode statisch ist, kann sie direkt als Funktion der Klasse String aufgerufen werden: String.valueOf()

```
String s = String.valueOf(3.1);  
char tmp[] = {'H', 'a', 'l', 'l', 'o'};  
String t = String.valueOf(tmp);  
String u = String.valueOf(new Testobjekt());  
String v = String.valueOf(true);
```


Einige wichtige Funktionen der Klasse String:

- `static String format(String format, Object obj1, Object obj2, ...)`
 - Erzeugt einen formatierten String, bei dem bestimmte Platzhalter durch konkrete Werte ersetzt werden
 - `String.format("Der Wert von x beträgt %f", x)`
→ der Platzhalter `%f` wird durch den Wert von `x` ersetzt und der sich dadurch ergebende String zurückgegeben
 - Die Methode ist statisch, d.h. es muss kein String-Objekt erzeugt werden; stattdessen kann direkt `String.format()` aufgerufen werden

Einige wichtige Funktionen der Klasse String:

- static String format(String format, Object obj1, Object obj2, ...)
 - Einige der möglichen Platzhalter sind:
 - %d - Ersetzung durch Ganzzahl wie byte, int, long
 - %7d - Ersetzung durch Ganzzahl. Der Platzhalter wird durch mindestens sieben Zeichen ersetzt, bei Zahlen mit weniger als 7 Ziffern wird links mit Leerzeichen aufgefüllt.
 - %f - Ersetzung durch Fließkommazahl wie float, double
 - %7.2f - Ersetzung durch Fließkommazahl. Mindestens 7 Zeichen, davon genau zwei Zeichen für Nachkommastellen, ein Zeichen für das Komma und mindestens 4 (7-2-1) Zeichen für Vorkommastellen.
 - %.2f - Fließkommazahl mit genau zwei Nachkommastellen und beliebig vielen Vorkommastellen

Einige wichtige Funktionen der Klasse String:

- static String format(String format, Object obj1, Object obj2, ...)
 - Einige der möglichen Platzhalter sind:
 - %s - Ersetzung durch String (ein Nullzeiger wird durch "null" ersetzt, bei Objekten wird die toString()-Funktion aufgerufen)
 - %7s - Ein String mit einer Länge von mindestens 7 Zeichen. Fehlende Zeichen werden von links mit Leerzeichen aufgefüllt.
 - %c - Ein einzelner Buchstabe. Eine positive Ganzzahl wird dabei (entsprechend der *Locale*) in den Buchstaben umgewandelt, den sie repräsentiert.

Einige wichtige Funktionen der Klasse String:

- static String format(String format, Object obj1, Object obj2, ...)
 - Nebenbei:
 - Weitere Formatierungsoptionen ermöglichen bspw. Linksbündigkeit, führende Nullen oder erzwungene Vorzeichen
 - Zur Datums- und Uhrzeitformatierung gibt es eigene Platzhalter
 - Weitere Informationen liefert die Dokumentation der Klasse `Formatter`

Einige wichtige Funktionen der Klasse String:

- `boolean matches(String regex)` - gibt an, ob der String dem regulären Ausdruck `regex`
- `String replaceAll(String regex, String replacement)` - ersetzt alle Treffer des regulären Ausdrucks `regex` durch den String `replacement`
- `String[] split(String regex)` - teilt den String bei jedem Auftreten von `regex` und gibt ein Array aller so entstandenen Teilstrings zurück

- Mehr Informationen zu regulären Ausdrücken liefert die Dokumentation zur Klasse `Pattern`

StringBuilder

- Ein `StringBuilder`-Objekt repräsentiert wie ein `String` eine Folge von Zeichen
- Während `Strings` unveränderlich sind, sind `StringBuilder` **veränderliche** Zeichenketten
- Die Klasse `StringBuilder` stellt einige Funktionen bereit, um die Zeichenkette zu manipulieren
 - Bspw. können Zeichen hinzugefügt, gelöscht oder geändert werden
 - Da die im Objekt gespeicherte Zeichenkette manipuliert wird, müssen nicht ständig neue `StringBuilder`-Objekte oder andere Zwischenvariablen erzeugt werden

Einige wichtige Funktionen der Klasse StringBuilder:

- `char charAt(int index)`, `int length()` und `String substring(int start, int ende)`, da `StringBuilder` das Interface `CharSequence` implementiert
- `int indexOf(String str)`, `int indexOf(String str, int start)`, `int lastIndexOf(String str)` - wie `indexOf` bzw. `lastIndexOf` der Klasse `String`

Einige wichtige Funktionen der Klasse StringBuilder:

■ StringBuilder append(*)

- Die append()-Funktion ist für alle primitiven Datentypen sowie für Objekte der Klasse Object überladen, sie akzeptiert somit beliebige Argumente
- Wird append(Object obj) aufgerufen, so wird automatisch obj.toString() aufgerufen und der zurückgegebene String angehängen

```
Testobjekt obj = new Testobjekt();  
StringBuilder s = new StringBuilder();  
s.append("Hallo!");  
s.append(123);  
s.append(obj);  
System.out.println(s); // Hallo!123Testobjekt@4aa298b7
```


Einige wichtige Funktionen der Klasse StringBuilder:

- `StringBuilder insert(int offset, *)`
 - Die `insert()`-Funktion akzeptiert wie `append` alle Datentypen als Argumente
 - Während `append` einen neuen String immer am Ende des bisherigen Strings **anfügt**, fügt `insert` den neuen String an der gegebenen Position `offset` **ein**

```
Testobjekt obj = new Testobjekt();  
StringBuilder s = new StringBuilder();  
s.insert(0, "Hallo!");  
s.insert(0, 123);  
s.insert(3, obj);  
System.out.println(s); // 123Testobjekt@4aa298b7Hallo!
```

Einige wichtige Funktionen der Klasse StringBuilder:

- `StringBuilder delete(int start, int ende)` - löscht den String, der an der Position `start` beginnt und an der Position `ende - 1` endet, aus dem Gesamtstring heraus
 - Gilt `start == ende` wird nichts gelöscht
- `StringBuilder deleteCharAt(int index)` - löscht einen einzelnen Buchstaben an der angegebenen Position
- Einige Eigenschaften des Strings (Länge u.ä.) werden automatisch angepasst

```
StringBuilder s = new StringBuilder("MuP ist toll!");  
s.delete(4, 8);  
System.out.println(s);    // MuP toll!  
s.deleteCharAt(8);  
System.out.println(s);    // MuP toll
```

Einige wichtige Funktionen der Klasse StringBuilder:

- `StringBuilder replace(int start, int ende, String str)`
 - Ersetzt den String, der an der Position `start` beginnt und an der Position `ende - 1` endet, durch einen neuen String `str`
 - Entspricht der Kombination von `delete(start, ende)` und `insert(start, str)`
 - Ist `str` der leere String `""`, entspricht `replace` der Funktion `delete`

```
StringBuilder s = new StringBuilder("MuP ist toll!");  
s.replace(8, 12, "super");  
System.out.println(s);    // MuP ist super!  
s.replace(4, 8, "");  
System.out.println(s);    // MuP super!!
```

Einige wichtige Funktionen der Klasse StringBuilder:

- reverse()
 - Kehrt den kompletten String um
- setCharAt(int index, char ch)
 - Ersetzt einen einzelnen Buchstaben an der gegebenen Stelle index durch den Buchstaben ch

```
StringBuilder s = new StringBuilder("MuP ist toll!");  
s.reverse();  
System.out.println(s);    // !llot tsi PuM  
s.setCharAt(5, ' ');  
s.setCharAt(9, '_');  
System.out.println(s);    // !llot.tsi_PuM
```