

# Programmieren II (Java)

## 2. Praktikum: Grundlagen Objektorientierung

Sommersemester 2024

Christopher Auer



### Abgabetermine

### Lernziele

- ▶ Implementieren nach einer Spezifikation
- ▶ Aufbau von Klassen: Attribute und Methoden
- ▶ Konstruktoren: Verkettung, Kopier-Konstruktor
- ▶ Getter und Setter
- ▶ Java-Standard-Methoden: equals, toString
- ▶ Dokumentation: javadoc
- ▶ Testen mit JUnit
- ▶ **enums**: definieren, erweitern und verwenden

### Hinweise

- ▶ Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- ▶ Sie müssen *4 der 5* Praktika bestehen
- ▶ *Kommentieren* Sie Ihren Code
  - ▶ Jede *Methode* (wenn nicht vorgegeben)
  - ▶ *Wichtige* Anweisungen/Code-Blöcke
  - ▶ *Nicht kommentierter* Code führt zu *Nichtbestehen*
- ▶ Bestehen Sie eine Abgabe *nicht* haben Sie einen *zweiten Versuch*, in dem Sie Ihre Abgabe *verbessern müssen*.
- ▶ *Wichtig*: Sie sind einer *Praktikumsgruppe* zugewiesen, *nur* in dieser werden Ihre Abgaben *akzeptiert*!



## Bevor es losgeht

Bevor wir starten, lesen Sie sich *aufmerksam* folgende Hinweise durch:

- *Importieren* Sie zunächst das beigefügte *Gradle-Projekt* unter SupportMaterial/passwordmanager in Ihre bevorzugte Entwicklungsumgebung. Erstellen Sie Ihre Klassen im Projekt-Verzeichnis unter:

src/main/java

Wenn Sie Ihre Klassen *woanders* erstellen, wird Ihr Projekt *nicht funktionieren*!

- *Dokumentieren* Sie *alle Klassen* und *öffentlichen Methoden* mit *JavaDoc*! Zusätzlich müssen Sie weiterhin Ihren Quellcode *kommentieren*.
- Verwenden Sie zum Prüfen auf *Gleichheit* von  Strings und anderen Objekten die Methode *equals*!
- Prüfen Sie die Parameter jeder Methode auf *Gültigkeit*! Sollte ein Parameter einen ungültigen Wert haben, erzeugen Sie eine  *IllegalArgumentException* wie folgt:

```
throw new IllegalArgumentException("Aussagekräftige (!) Fehlermeldung");
```

Geben Sie eine *aussagekräftige Fehlermeldung* an!

- Achten Sie auf die korrekte *Sichtbarkeit* (*public*, *private*, *protected*) der Klassen, Attribute und Methoden! Es ist im Folgenden *nicht korrekt* einfach *keine Sichtbarkeit* anzugeben — und der Übungsleiter wird Sie darauf ansprechen!
- Testen Sie Ihre Klassen mit den mitgelieferten *JUnit*-Tests in den beigefügten Gradle-Projekten! Führen Sie dazu den Gradle-Task *test* aus. Die *JUnit*-Tests finden Sie im Projektverzeichnis unter:

src/test/java

Machen Sie sich außerdem mit der Unterstützung von *JUnit*-Tests in Ihrer Entwicklungsumgebung vertraut.

- *Tipp*: Haben Sie die Implementierung einer Methode abgeschlossen, kommentieren Sie die Tests mit entsprechenden Namen ein. Diese sind nach dem Schema *testMethodenName...* benannt.
- Solange ein Test scheitert, ist *Ihre Implementierung nicht korrekt*. Betrachten Sie in diesem Fall die Fehlermeldung und den Quellcode des gescheiterten Tests. Für die Abgabe müssen *alle Testfälle* erfolgreich durchlaufen — der Übungsleiter wird Sie dazu auffordern, die Tests laufen zu lassen und die Abgabe *erst akzeptieren* wenn alle Tests *erfolgreich* sind!
- *Verändern Sie nicht die Inhalte der JUnit-Tests* um das Problem zu „lösen“!



## Aufgabe: Passwort-Manager ★★

In dieser Aufgaben implementieren wir einen einfachen *Passwort-Manager um die Zugangsdaten zu Internetseiten* zu verwalten. Dazu implementieren wir im Folgenden vier Klassen:

- ▶ PasswordComplexity ist eine **enum** und Hilfsklasse um Passworte verschiedener *Stärken* zu erzeugen.
- ▶ PasswordEntry modelliert die *Zugangsdaten* (Login und Passwort) zu einer Website.
- ▶ PasswordStore dient zur *Verwaltung* unserer Zugangsdaten als eine ↗ einfach-verkettete Liste.
- ▶ PasswordManager beinhaltet das *Hauptprogramm* zur Verwaltung der Zugangsdaten. Diese Klasse ist bereits für Sie vorbereitet und muss von Ihnen etwas ergänzt werden.

### Hauptprogramm

Die main-Methode befindet sich in der Klasse PasswordManager, die einen den ganzen Passwort-Manager bereits zum Großteil implementiert. Ist es empfehlenswert, dass Sie eine *weitere main-Klasse* erstellen in dessen main-Methode Sie Ihre bisherigen Implementierung testen.

### Passwort-Komplexität: Das enum PasswordComplexity ★★

Ein guter Passwort-Manager sollte Passwörter *generieren* können. Dazu modellieren wir fünf verschiedene Typen der Passwort-Komplexität mit einer **enum** mit folgenden Eigenschaften:

Wert	String chars	int length
PIN	"0"	4
SIMPLE	"aA"	5
MEDIUM	"aA0"	8
COMPLEX	"aA0!"	10
SUPER_COMPLEX	"aA0!"	16

Das Attribut length gibt die *Länge* des Passworts an. Das Attribut chars gibt an, aus welchen Zeichen das Passwort besteht mit:

Wert	Bedeutung	Zeichen
0	Ziffern	0123456789
a	Kleinbuchstaben	abcdefghijklmnopqrstuvwxyz
A	Großbuchstaben	ABCDEFGHIJKLMNOPQRSTUVWXYZ
!	Sonderzeichen	! ? + - ; , . :

*Implementieren* Sie PasswordComplexity wie angegeben und statuen Sie sie mit zwei *Gettern* für die zwei Attribute aus! Testen Sie Ihre Implementierung mit PasswordComplexityTest.testValues.

### Generieren von Passwörtern ★★

Implementieren Sie die Methode **public String generatePassword()** in PasswordComplexity, die ein Passwort der Länge length mit Zeichen entsprechend chars zufällig generiert und zurückgibt!

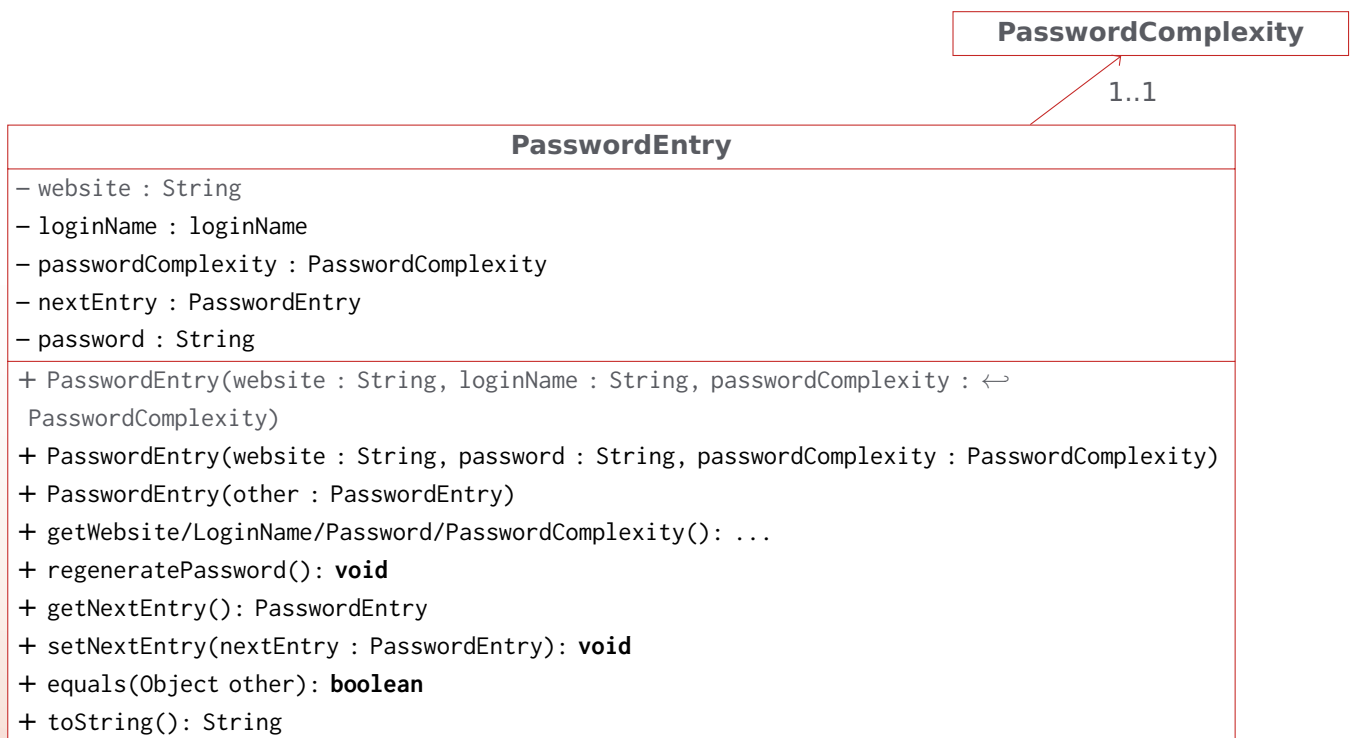
Ist z.B. `chars="aA0` und `length=8`, so könnte ein mögliches Passwort `K9mn10Pi` sein. Testen Sie `generatePassword` mit `PasswordComplexityTest.testGeneratePassword`.

*Hinweise* falls Sie nicht weiterwissen:

- ▶ Sie können mit `String.charAt(index)` auf das Zeichen am Index `index` (beginnend bei 0) in einem String zugreifen. Die Länge eines Strings ermitteln Sie mit `String.length()`.
- ▶ Die Methode `Random.nextInt(bound)` erzeugt `int`-Zufallszahlen von 0 bis `bound-1`. Sie müssen dazu ein Objekt vom Typ `Random` erzeugen.
- ▶ Erzeugen Sie sich zunächst einen String `allowedChars`, der *alle* erlaubten Zeichen beinhaltet. Iterieren Sie über die einzelnen Zeichen des Attributs `chars` und hängen an `allowedChars` alle erlaubten Zeichen gemäß der obigen Tabelle an. Erzeugen Sie dann Zeichen für Zeichen des Passworts, indem Sie zufällig Zeichen aus `allowedChars` auswählen und an das Passwort anhängen.

### Die Klasse `PasswordEntry` ★★

Die Klasse `PasswordEntry` modelliert einen *Eintrag in unserem Passwort-Manager*:<sup>1</sup>



Die Attribute von `PasswordEntry` modellieren:

- ▶ `String website` — Webseite für die die Zugangsdaten hinterlegt werden (*unveränderlich*, darf *nicht null* oder „blank“ sein, d.h. nur aus „Whitespaces“ bestehen)
- ▶ `String loginName` — der Login-Name (*unveränderlich*, darf `null` sein)
- ▶ `PasswordComplexity passwordComplexity` — die Komplexität des Passworts (*unveränderlich*, darf nicht `null` sein)
- ▶ `String password` — das Passwort (*veränderlich*, wird durch `PasswordComplexity.generatePassword()` erzeugt)

<sup>1</sup>Aus *Platzgründen* sind manche Methoden, durch *Schrägstriche* getrennt, in einer Zeile aufgelistet.

- `PasswordEntry nextEntry` — der nächste Passwort-Eintrag in der *Liste* (*veränderlich*, darf `null` sein und wird mit `null` initialisiert)

*Deklarieren* Sie die Klasse `PasswordEntry` und die *Attribute* wie angegeben! Achten Sie auf *korrekte Sichtbarkeit* und *Modifizierer*. Implementieren Sie den Konstruktor `PasswordEntry(String website, String loginName, PasswordComplexity passwordComplexity)`, der die drei Attribute *initialisiert*. Rufen Sie `PasswordComplexity.generate` auf um das Passwort zu erzeugen.

*JUnit-Test*: `PasswordEntryTest.testConstructor`.

### Konstruktor-Verkettung ★★

Implementieren Sie den zweiten Konstruktor, der alle Argument des ersten Konstruktors aufnimmt, bis auf `loginName`! Rufen Sie dazu den ersten Konstruktor auf und übergeben Sie für `loginName` den Wert `null`.

*JUnit-Test*: `PasswordEntryTest.testConstructor2`.

### Kopier-Konstruktor ★★

Statten Sie `PasswordEntry` mit einem *Kopier-Konstruktor* `PasswordEntry>PasswordEntry other` aus!

*JUnit-Test*: `PasswordEntryTest.testCopyConstructor`.

### Getter und Setter ★★

*Implementieren* Sie die Getter für alle Attribute und einen Setter für `nextEntry`!

*JUnit-Tests*: keine

### Password neu generieren ★★

*Implementieren* Sie die Methode `regeneratePassword`, die das Passwort mit Hilfe der Methode `PasswordComplexity.generatePassword()` neu generiert.

*JUnit-Tests*: `PasswordEntryTest.testRegeneratePassword`

### String-Repräsentation mit `toString` ★★

*Implementieren* Sie die Methode `String toString()`, die eine String-Repräsentation des Eintrags nach folgendem Schema zurückgibt:

`moodle.haw-landshut.de L:s-cauer P:XVurk05o (MEDIUM)`

D.h. zuerst `website`, dann `loginName`, dann `password` und schließlich `passwordComplexity` in Klammern. Sie können für die Erstellung der String-Repräsentation *Konkatenation* oder die Methode `String.format` verwenden.

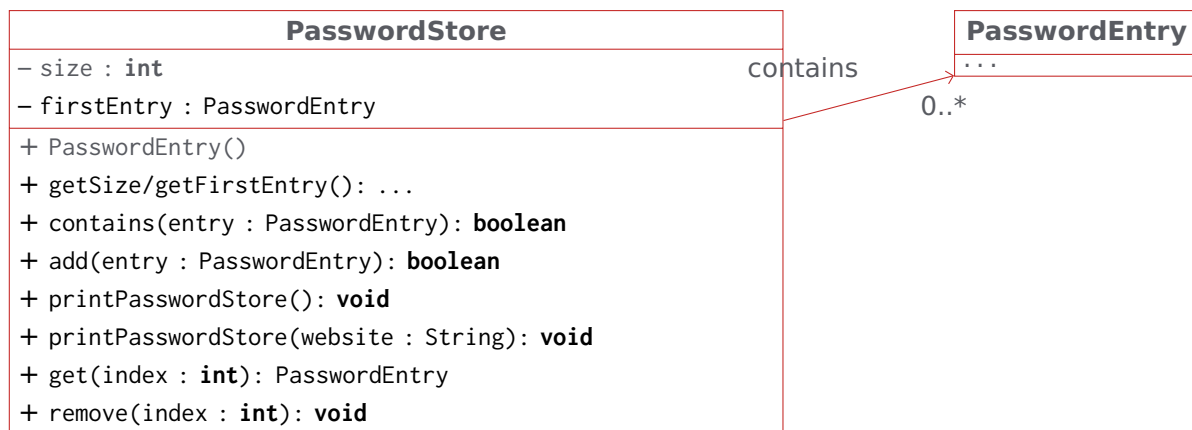
*JUnit-Test*: keiner

**Gleichheit mit equals** ★★

**Implementieren** Sie die equals-Methode in PasswordEntry! Zwei PasswordEntry-Instanzen sind *gleich* wenn die Attribute website, loginName und passwordComplexity *gleich* sind. Gehen Sie dabei nach dem in der Vorlesung gezeigten „Kochrezept“ vor. Vergessen Sie nicht @Override zu verwenden und beachten Sie, dass das Argument der Methode equals vom *Typ Object* und *nicht vom Typ PasswordEntry* ist.

**JUnit-Test:** PasswordEntryTest.testEquals

Die Klasse PasswordStore dient zur Verwaltung von mehreren PasswordEntryS.



Sie besitzt zwei Attributen:

- ▶ **int** size — aktuelle Anzahl der Einträge
- ▶ PasswordEntry firstEntry — Referenz auf den ersten Eintrag

Die Einträge sind in einer einfach-verketteten Liste abgelegt, wobei firstEntry auf das *erste Element* der Liste verweist und das Attribut nextEntry in PasswordEntry auf das *nächste Element*:<sup>2</sup>

**Attribute und Konstruktor von PasswordStore** ★★

**Deklarieren** Sie die Klasse PasswordStore mit den *zwei Attributen* wie oben definiert! Achten Sie wieder auf die *Sichtbarkeit*. Implementieren Sie den Default-Konstruktor, der size mit 0 und firstEntry mit null initialisiert. Fügen Sie zudem die Getter für die beiden Attribute hinzu und einen Setter für firstEntry! Warum macht es keinen Sinn, dass size einen Setter besitzt?

**JUnit-Test:** PasswordStore.testConstructor

**Die Methode contains** ★★

Die Methode **boolean** contains>PasswordEntry entry) prüft ob sich eine PasswordEntry-Instanz in der Liste befindet, die *gleich* (!) zu entry ist. In dem Fall, liefert sie **true** zurück, sonst **false**. **null** ist für entry *nicht erlaubt*. Implementieren Sie contains!

<sup>2</sup>Später lernen wir die Java-Collections kennen, die diese Art von Datenstruktur und noch viele mehr bereits implementieren.



*JUnit-Test:* PasswordStore.testContains

### Die Methode add ★★

Die Methode `boolean add(PasswordEntry entry)` fügt einen Eintrag in die Liste ein, aber *nur wenn dieser noch nicht vorhanden ist*. Ist er schon vorhanden, gibt die Methode `false` zurück. Sonst wird entry *am Anfang* der Liste eingefügt, size um 1 erhöht und `true` zurückgegeben. `null` ist als Parameter wie immer *nicht erlaubt*. Implementieren Sie `add`!

*JUnit-Test:* PasswordStore.testAdd

*Hinweise* wenn Sie nicht weiterwissen:

- ▶ Verwenden Sie `contains` um herauszufinden, ob das Element bereits *vorhanden* ist.
- ▶ entry wird das *neue erste Element* im PasswordStore.
- ▶ Das *neue erste Element* muss auf das *alte erste Element* als *nächstes Element* verweisen.

### Ausgeben des PasswordStores

Implementieren Sie die Methode `void printPasswordStore`, die alle Einträge wie folgt ausgibt:

```
0. reddit.com L:extremelurker P:gBn0v
1. github.com L:java-programmer P:Fo,AqWte9C
2. moodle.haw-landshut.de L:s-cauer P:ENZ5hAyt
```

Für die Ausgaben verwenden Sie `PasswordEntry.toString`. Diese Methoden testen Sie mit Hilfe des `main`-Programms in der nächsten Aufgabe.

### main-Methode

Implementieren Sie die Klasse `PasswordManager.addExampleEntries` wie folgt: Erstellen Sie drei beispielhafte `PasswordEntry`s Ihrer Wahl und fügen Sie sie in den übergebenen `PasswordStore` ein. Kommentieren Sie danach die Implementierungen der Methoden `printPasswordStore` und `addPasswordEntry` ein und führen Sie das Programm aus. Sie können über das Drücken der Taste 1 mit Enter testen, ob ihre erstellten Einträge richtig ausgegeben werden:

```
(N)ew entry, (L)ist entries, re(G)enerate password, (F)ind entry, (R)emove entry, (Q)uit
1
0. reddit.com L:extremelurker P:CPRkc (SIMPLE)
1. github.com L:java-programmer P:Bc7G4Bx8po (COMPLEX)
2. moodle.haw-landshut.de L:s-cauer P:xQjWwjqo (MEDIUM)
(N)ew entry, (L)ist entries, re(G)enerate password, (F)ind entry, (R)emove entry, (Q)uit
```

Über `n` können Sie einen neuen Eintrag erzeugen. Warum werden die Einträge in der zum Einfügen *umgekehrten Reihenfolge* ausgegeben?

```
(N)ew entry, (L)ist entries, re(G)enerate password, (F)ind entry, (R)emove entry, (Q)uit
n
Website: gitlab.com
Login name: supercoder0815
Choose a password complexity:
0: PIN - 0 Length: 4
```

```
1: SIMPLE - aA Length: 5
2: MEDIUM - aA0 Length: 8
3: COMPLEX - aA0! Length: 10
4: SUPER_COMPLEX - aA0! Length: 16
Selection: 3
New entry added: gitlab.com L:supercoder0815 P:v2GLeV:.SH (COMPLEX)
```

### Ausgabe mit Suche ★★

Implementieren Sie die *überladene* Methode `printPasswordStore(String website)`, die *die gleiche Ausgabe* wie `printPasswordStore` macht, aber nur für die Website, die im Parameter angegeben ist (darf nicht `null` sein)! Kommentieren Sie danach die Methode `printPasswordStoreForWebsite` ein und testen Sie Ihre Implementierung mit dem Hauptprogramm über „find entry“:

```
(N)ew entry, (L)ist entries, re(G)enerate password, (F)ind entry, (R)emove entry, (Q)uit
f
Website: github.com
0. github.com L:supercoder0815 P:DRcfY (SIMPLE)
2. github.com L:java-programmer P:G5C,K-lcd4 (COMPLEX)
```

#### Achtung:

- ▶ Beachten Sie, dass der ausgegebene Index der Stelle in der verketteten Liste entspricht!
- ▶ Überlegen Sie sich, wie sie die Methode `printPasswordStore` implementieren können *ohne Quellcode zu duplizieren*.

### Die Methode `get` ★★

Sie haben es gleich geschafft! Was noch fehlt ist die Methode `PasswordEntry get(int index)`, die für den übergebenen Index den entsprechenden `PasswordEntry` in der verketteten Liste zurückgibt. Der index muss dabei gültig sein. Testen Sie `get` mit `PasswordStoreTest.testGet` und indem Sie die Methode `PasswordManager.regeneratePassword` einkommentieren! Jetzt können Sie mit der Eingabe `g` und dem Index ein Passwort *neu generieren*.

### Optional: Die Methode `remove` ★★

Implementieren Sie die Methode `remove(int index)`, die den Eintrag an der Stelle `index` entfernt. Testen Sie `remove` mit `PasswordStore.testRemove` und indem Sie die Methode `PasswordManager.removeEntry` einkommentieren. Jetzt können Sie mit der Eingabe `g` und dem Index einen Eintrag wieder entfernen.