
Artemis 6.4.0

Course Overview

ge64baw

Courses > Praktikum: Grundlagen der Programmierung WS22/23 > Exercises > W05H02 - The Pingu Network

W05H02 - The Pingu Network

Hausaufgabe

Medium

Submission due: 8 months ago

Complaint due: 8 months ago

Points: 6 of 6

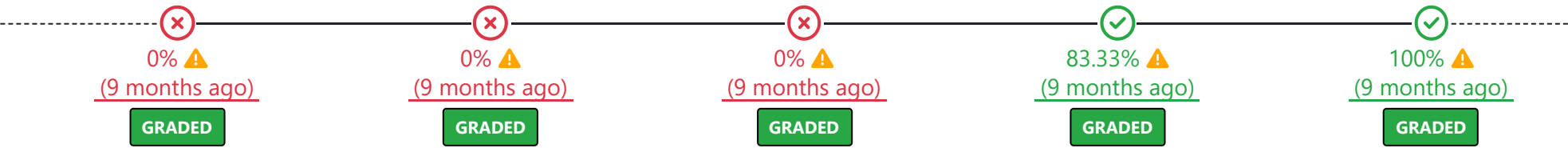
Assessment: automatic ?

Resume practice in exercise

100%  (9 months ago)

GRADED

Recent results:



Show all results

Tasks:

The Pingu Network

Der Pinguin Mark freut sich darüber, dass er so viele neue Freunde an der PUM findet. Leider muss er feststellen, dass die bisherige Methode, mit der er seine Freunde verwaltet, nicht mehr ausreicht. Der Zettel unter seinem Flügel ist schon sehr voll und man kann teilweise die Einträge nicht mehr richtig lesen.

Um sich das Ganze zu vereinfachen, hat Mark ein soziales Netz erdacht. Hier hat er eine Übersicht über seine Freunde und kann sich mit ihnen austauschen.

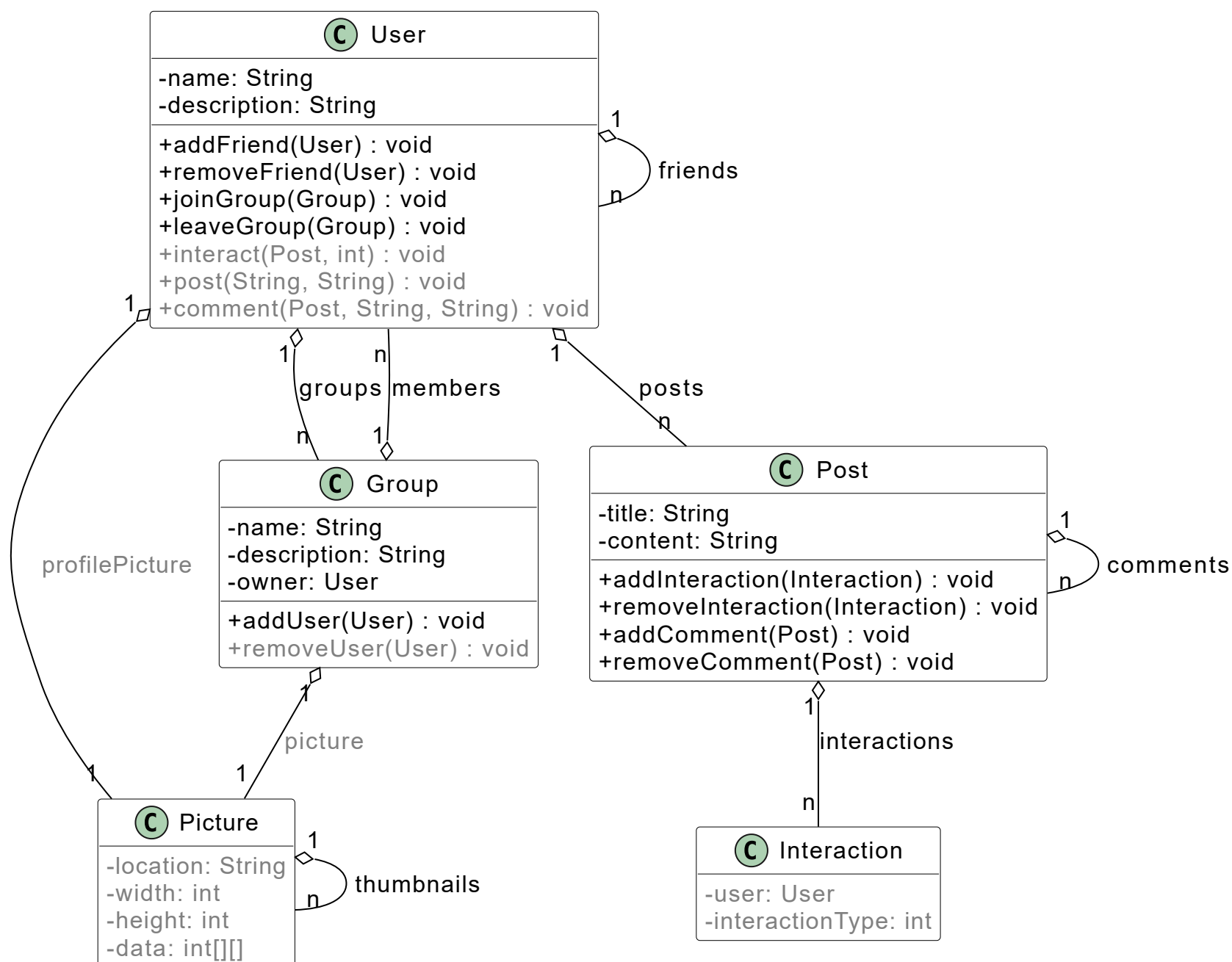
Zur Umsetzung des Projektes hat Mark bereits ein UML-Diagramm entworfen, das das soziale Netz modellieren soll. Zudem hat er eine Spezifikation verfasst, was die jeweiligen Klassen tun sollen. Er hat auch einige Teile davon schon implementiert, hängt jetzt aber an ein Paar noch übrig gebliebenen Teilen des Diagramms fest. Kannst du ihm da weiterhelfen?

Diagramm

Das folgende Diagramm stellt die Klassenstruktur des Netzwerks dar. Alle Attribute, welche mehrere Elemente speichern können, sollen als Array implementiert werden. Jedes Array ist genau so groß wie die Anzahl an Elementen, die es enthält. Somit gibt es keine leeren Stellen und bei Veränderungen muss ein neues Array erstellt werden. Der Konstruktor für jede Klasse soll alle Attribute, welche keine Arrays sind, als Parameter übernehmen und zwar in genau der Reihenfolge, in der sie im UML-Diagramm aufgelistet sind. Die Arrays sollen als leere Arrays mit Größe 0 initialisiert werden. Diese beiden Dinge gelten natürlich nur dann, wenn die Spezifikation nicht explizit Anderes verlangt.

Hinweise:

- Das Programm soll absturzsicher geschrieben werden. Insbesondere betrifft das die verwendeten Arrays. Egal, welche `User-`, `Post-` etc. Objekte also den jeweiligen `add` und `remove` Methoden übergeben wird, sollte sichergestellt sein, dass das Array nie an unzulässigen Stellen (`< 0` und `>= array.length`) zugegriffen werden.
- Die im Diagramm farbig hervorgehobenen Member müssen noch implementiert werden. Sie sind grau oder rot, solange sie noch nicht vorhanden sind, grün, sobald sie vorhanden sind. Grüne Farbe bedeutet also nur, dass der korrekte Methodenkopf erkannt wurde und noch nicht, dass die Methode auch korrekt implementiert wurde. Das musst du selbst testen.
- Du solltest die bereits im Template implementierten Methoden (d.h. solche, die nicht leer sind und bei denen nicht explizit ein `TODO` dransteht) nicht verändern. Wenn du das doch tust, tust du es auf eigene Gefahr hin, dass bestimmte Tests dann nicht mehr funktionieren.
- Konstruktoren und Attribute musst du teils anpassen; teils musst du auch neue Methoden hinzufügen.
- Ihr dürft bei allen übergebenen Objekten davon ausgehen, dass diese nicht `null` sind.
- Die Tests sind nicht 100% unabhängig von einander. Es kann also vorkommen, dass ein User Test failed, der Fehler aber in einer anderen Klasse liegt.



Kurzbeschreibung der einzelnen Klassen

Hier findest du eine kurze Beschreibung der verschiedenen im UML-Diagramm dargestellten Klassen und von deren Members. Sowohl der bereits implementierten Teile, als auch der noch zu implementierenden. Wenn beim Einfügen in Arrays überprüft werden soll, ob ein Objekt bereits vorhanden ist, kann das einzufügende Objekt mit den vorhandenen mit `==` verglichen werden. Es gilt also nur, herauszufinden, ob wirklich das exakt gleiche Objekt bereits im Array ist.

Klasse User

1. Zusätzlich zu den im Diagramm beschriebenen Methoden sollen Getter für alle Attribute und Setter für alle Attribute, welche keine Arrays sind, existieren.
2. Die `addFriend/removeFriend` Methoden fügen den spezifizierten Nutzer zur eigenen Freundesliste hinzu bzw. entfernen ihn wieder. Am anderen Nutzer ändert sich nichts. Derselbe Nutzer (Objektgleichheit!) soll nicht doppelt gespeichert werden falls er wiederholt hinzugefügt wird.
3. Ähnlich fügen die `joinGroup/leaveGroup` Methoden die spezifizierte Gruppe zu der Liste an beigetretenen Gruppen hinzu oder entfernt sie. Zudem wird der Nutzer den Mitgliedern der Gruppe hinzugefügt bzw. aus diesen entfernt. Dieselbe Gruppe (simile) soll nicht doppelt gespeichert werden, falls sie wiederholt hinzugefügt wird.
4. Die `interact` Methode erstellt eine neue `Interaction` mit dem übergebenen `int interactionType` und fügt diese dem übergebenen Post hinzu.
5. Mit der `post` Methode wird ein neuer `Post` mit einem übergebenen Titel und Inhalt erstellt und zum Array eigener Posts hinzugefügt.
6. Die `comment` Methode funktioniert wie die `post` Methode, nur dass der neu erstellte `Comment` zudem als Kommentar dem übergebenen `Post` angefügt wird.

Klasse Post

1. Zusätzlich zu den im Diagramm beschriebenen Methoden sollen Getter für alle Attribute und Setter für alle Attribute, welche keine Array sind, existieren.
2. Mit `addInteraction/removeInteraction` wird die übergebene Interaktion in das Array der Interaktionen des Posts hinzugefügt oder aus dieser entfernt. Dieselbe Interaktion soll nicht doppelt gespeichert werden, falls sie wiederholt hinzugefügt wird.
3. Gleiches gilt für `addComment/removeComment`, nur eben mit übergebenen Posts.

Klasse Group

Hier gibt es auch Getter für alle Attribute und Setter für alles außer die Mitglieder. `addUser/removeUser` sind dazu da, um neue Nutzer der Gruppe und dem Array an Mitgliedern hinzuzufügen oder aus dieser zu entfernen.

Derselbe Nutzer soll nicht doppelt beitreten können.

Falls bei `removeUser` der `owner` entfernt wird, soll der neue `owner` der erste Eintrag im Array der Mitglieder werden. Gibt es keine verbleibende Mitglieder in der Gruppe, ist der `owner` `null`.

Wird der Owner manuell gesetzt, muss dieser natürlich auch vorher in der Gruppe sein.

Falls der zu setzende `owner` kein Mitglied ist, passiert nichts. Der Konstruktor dieser Klasse übernimmt, ganz nach der oben beschriebenen Regel, den `owner` dieser als Parameter. Dieser soll, wie gehabt, in das entsprechende Attribut gesetzt werden. Allerdings muss er dann natürlich auch als Mitglied der Gruppe zu `members` hinzugefügt werden.

Klasse **Interaction**

Die Klasse **Interaction** hat lediglich Getter für alle ihre Attribute. Sonst sind keine Methoden vorhanden.

Klasse **Picture**

Picture soll als Methoden alle Getter (einschließlich für **data**) und zusätzlich noch einen Setter für Thumbnails beinhalten. Zudem sollen im Konstruktor die Attribute **location** und **data** übergeben werden. Höhe und Breite müssen aus dem **data** Feld erschlossen werden. In der **Picture** Klasse werden zudem Höhe und Breite anhand der Dimensionen des übergebenen Arrays berechnet. Ein Array `int[2][3]` hat zum Beispiel Höhe 2 und Breite 3. Das **data** Array ist immer ein valides Array, wessen Teilarrays alle die gleiche Länge haben.

Aufgaben

Nutzer aus Gruppe Entfernen

In der Klasse **Group** fehlt noch die Implementierung der Methode `void removeUser(User)`. Diese soll das übergebene **User**-Objekt aus dieser Gruppe entfernen. Falls dabei der **owner** der Gruppe entfernt wird, soll der erste im **members**-Array der neue **owner** werden.

Nutzer-Operationen mit Kommentaren

In der Klasse **User** sind die drei Methoden `void interact(Post, int)`, `void post(String, String)` und `void comment(Post, String, String)` noch leer. Implementiere diese, sodass sie das in der Spezifikation geforderte Verhalten aufweisen!

Inhalt der **Interaction**-Klasse

Die Klasse **Interaction** ist noch komplett leer. Fülle diese, wie in der Spezifikation verlangt.

Bilder

Zur Implementierung der Klasse **Picture** ist Mark leider gar nicht mehr gekommen. Füge diese Klasse sowie all ihre Verwendungen in **User** und **Group**, wie in der Spezifikation (und dem UML-Diagramm) verlangt, hinzu. (Du wirst in **User** und **Group** also evtl. neue Attribute, Getter und Setter erstellen müssen.)

 **Grading** No results

Hier werden erst nach der Deadline Ergebnisse angezeigt. Während der Abgabe steht hier also immer 0 von 6 Tests.

[Musterlösung](#)

[Tests](#)

FAQ

Q: Wofür ist das FAQ?

A: Um häufig gestellte Fragen zu sammeln. Wer trotzdem noch fragt macht Pinguine traurig.

~~Q: Warum schlagen meine **Group** **Public** oder **Turn Green** Tests fehl?~~

Exercise details

Release date:	Nov 17, 2022 18:30
Submission due:	Dec 4, 2022 18:00
Complaint due:	Dec 11, 2022 18:00

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have **998** complaints left. 