Softwarepraktikum - Testdokumentation

Finn Brecher Daniel Hinkelmann Ole Krenzer Davide Piacenza

13.06.2024

Test Vorgehen:

Wie man der Tabelle auf der nächsten Seite entnehmen kann, werden einige aber nicht alle unserer Klassen getestet. Wir verwenden zum Testen JUnit-Tests, die einzelne Methoden vollständig oder gewisse Szenarien testen.

Triviale oder sehr einfache Methode wie beispielsweise: Getter, Setter, equals, hashCode, toString und compareTo Methoden werden im Allgemeinen nicht getestet.

Aus diesem Grund müssen die meisten `enum`s und `record`s (außer AgeRange, Location und Kitchen) nicht getestet werden.

Die `add` und `remove` Methoden in Pair und Group wurde noch nicht implementiert, da diese für Meilenstein 2 noch nicht benötigt werden und auch noch nicht entschieden wurde, wie und ob diese bei der Erstellung der GUI relevant sein werden.

Da ParticipantCollectionList java.util.ArrayList erweitert und nichts wesentliches ändert (außer Participant-Methoden, Null-Checks und Duplikt-Checks hinzuzufügen), muss diese Klasse nicht vollständig getestet werden. Folgende Methoden werden getestet:

- add
- remove
- contains
- containsParticipant
- containsAnyParticipant
- containsAllParticipants
- addAll
- removeAll
- retainAll
- containsAll

Bei den Klassen PairList, GroupList, InputData und OutputData werden keine einzelnen Methoden sondern die gesamt-Funktionalität der Klassen getestet.

Typ	Name	Zu Testen	Zu testende Methoden
`class`	**Main**	true	Ausgabe der CSV-Dateien
`class`	**Participant**	false	-
`record`	Name	false	-
`enum`	AgeRange	true	• getAgeRange • getAgeDifference
`enum`	Course	false	-
`enum`	FoodType	false	-
`enum`	Gender	false	-
`record`	**Location**	true	• getDistance
`record`	**Kitchen**	true	• toString • equals • compareTo
`enum`	KitchenAvailability	false	-
`interface`	**ParticipantCollection**	false	• add • remove
`class`	Pair (erweitert ParticipantCollection)	false	wie Oberklasse
`class`	Group (erweitert ParticipantCollection)	false	wie Oberklasse
`abstract class`	**IdentNumber**	false	• calcNumElems • calcNumSuccessors • calcGenderDiversity • calcAgeDifference • calcPreferenceDeviation
`class`	PairIdentNumber (erweitert IdentNumber)	true	wie Oberklasse
`class`	GroupIdentNumber (erweitert IdentNumber)	true	wie Oberklasse + • getAveragePathLength • getTotalPathLength • getPathLengthStdDev
`abstract class`	**Weights**	false	-
`class`	PairingWeights (erweitert Weights)	false	-
`class`	GroupWeights (erweitert Weights)	false	-
`abstract class`	**ParticipantCollectionList** (erweitert java.util.ArrayList)	true	Testen wichtiger Listen Methoden
`class`	PairList	true	Testen auf Funktionalität
`class`	GroupList	true	Testen auf Funktionalität
`class`	**InputData**	true	Testen auf Funktionalität
`class`	**OutputData**	true	Ausgabe der CSV-Dateien

AgeRange Test:

Methode	# Tests	# Fehler	Voll. Abd.
getAgeRange(int)	18	0	ja
getAgeDifference(AgeRange)	5	0	ja

Location Test:

Methode	# Tests	# Fehler	Voll. Abd.
getDistance(Location)	2	0	ja

Kitchen Test:

Methode	# Tests	# Fehler	Voll. Abd.
Constructor(Location, int)	1	0	ja
toString()	1	0	ja
equals(Object)	4	0	ja
compareTo(Kitchen)	3	0	ja

IdentNumbers Tests:

PairIdentNumber Test

Methode	# Tests	# Fehler	Voll. Abd.
calcNumElems(ParticipantCollectionList)	1	0	ja
calcNumSuccessors(ParticipantCollectionList)	1	0	ja
calcGenderDiversity(ParticipantCollectionList)	2	0	ja
calcAgeDifference(ParticipantCollectionList)	2	0	ja
calcPreferenceDeviation(ParticipantCollectionList)	2	0	ja

GroupIdentNumber Test

Methode	# Tests	# Fehler	Voll. Abd.
calcNumElems(ParticipantCollectionList)	1	0	ja
calcNumSuccessors(ParticipantCollectionList)	1	0	ja
calcGenderDiversity(ParticipantCollectionList)	1	0	ja
calcAgeDifference(ParticipantCollectionList)	1	0	ja
calcPreferenceDeviation(ParticipantCollectionList)	1	0	ja
getAveragePathLength()	1	0	ja
getTotalPathLength()	1	0	ja
getPathLengthStdDev()	1	0	ja

Listen Tests:

ParticipantCollectionList Test

Methode	# Tests	# Fehler	Voll. Abd.
add	1	0	ja
remove	2	0	ja
contains	1	0	ja
containsParticipant	1	0	ja
containsAnyParticipant	1	0	ja
containsAllParticipants	1	0	ja
addAll	1	0	ja
removeAll	1	0	ja
retainAll	1	0	ja
containsAll	1	0	ja

PairList Test

Testname	Voll. Abd.
pairListTest	ja
pairListTestVariedPairingWeights	ja

Testname: pairListTest
Vorbedingung: Eine Instanz von InputData und PairingWeights existiert.
Ablauf: PairList wird mit den gegebenen Instanzen instanziiert und anschließend überprüft, ob alle gebildeten Paare erlaubt sind.
Erwartetes Verhalten: Alle Paare sind erlaubt und es wird **true** zurückgegeben.
Tatsächliches Verhalten: -

Testname: pairListTestVariedPairingWeights
Vorbedingung: Eine Instanz von InputData und zwei Instanzen von PairingWeights existieren.
Ablauf: Es werden zwei PairLists mit der gegebenen Instanz von InputData und je unterschiedlichen Instanzen von PairingWeights instanziiert.
Erwartetes Verhalten: Alle Paare sind erlaubt und es wird **true** zurückgegeben. Außerdem unterscheiden sich die gebildeten Paare der PairLists.
Tatsächliches Verhalten: -

GroupList Test

Testname	Voll. Abd.
groupListTest	ja
groupListTestVariedGroupWeights	ja

Testname:
groupListTest
Vorbedingung:
Eine Instanz von InputData und GroupWeights existiert.
Ablauf:
GroupList wird mit den gegebenen Instanzen instanziiert und anschließend überprüft, ob alle gebildeten Gruppen erlaubt sind.
Erwartetes Verhalten:
Alle Gruppen sind erlaubt und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
-

Testname:
groupListTestVariedGroupWeights
Vorbedingung:
Eine Instanz von InputData und zwei Instanzen von GroupWeights existieren.
Ablauf:
Es werden zwei GroupLists mit der gegebenen Instanz von InputData und je unterschiedlichen Instanzen von GroupWeights instanziiert.
Erwartetes Verhalten:
Alle Gruppen sind erlaubt und es wird **true** zurückgegeben. Außerdem unterscheiden sich die gebildeten Gruppen der GroupLists.
Tatsächliches Verhalten:
-

InputData Test:

Testname	Voll. Abd.
DataValueValidation	ja
GetMethods	ja
EventLocationLoading	ja
ParticipantsLoading	ja
PairsLoading	ja
ParticipantSuccessorLoading	ja
PairSuccessorLoading	ja
CorrectPairAssignment	ja
CorrectSuccessorAssignment	ja

Testname: DataValueValidation
Vorbedingung: Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf: Es werden alle Participants überprüft und getestet, ob deren Felder (Gender, FoodType, Age) valide Daten enthalten.
Erwartetes Verhalten: Alle Participants enthalten valide Daten und es wird **true** zurückgegeben.
Tatsächliches Verhalten: -

Testname: GetMethods
Vorbedingung: Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf: Es wird überprüft, ob die Getter für die Dateipfade und die Getter für die Eingabedaten (Participant und Pair) valide Ergebnisse liefern.
Erwartetes Verhalten: Alle spezifizierten Getter liefern valide Ergebnisse und es wird **true** zurückgegeben.
Tatsächliches Verhalten: -

Testname:
EventLocationLoading
Vorbedingung:
Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData und die Koordinaten der EventLocation sind bekannt (zum Testen).
Ablauf:
Es wird überprüft, ob InputData die EventLocation richtig eingelesen hat.
Erwartetes Verhalten:
InputData hat die EventLocation richtig eingelesen und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
-

Testname:
ParticipantsLoading
Vorbedingung:
Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf:
Es wird überprüft, ob die Liste der Participants nicht-leer ist und ob der erste Participant Person1 ist.
Erwartetes Verhalten:
Die Participants wurden richtig eingelesen und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
-

Testname:
PairsLoading
Vorbedingung:
Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf:
Es wird überprüft, ob die Liste der Pairs nicht-leer ist und ob sich das erste Paar zusammen angemeldet hat.
Erwartetes Verhalten:
Die Paare wurden richtig eingelesen und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
-

Testname:
ParticipantSuccessorLoading
Vorbedingung:
Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf:
Es wird überprüft, ob die Liste der Participant-Nachrücker richtig gebildet wurde und nur Participants enthält, die eine Küche haben (Da jede Küche nur genau 3 mal benutzt werden kann, werden Participants mit überschüssigen Küchen direkt aussortiert).
Erwartetes Verhalten:
Die Liste der Participant-Nachrücker wird richtig erstellt und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
Die gegebenen Daten enthalten Küchen, die mehr als 3 mal verwendet werden, jedoch werden in diesem Fall die Paare in die Nachrückerliste eingeordnet.

Testname:
PairSuccessorLoading
Vorbedingung:
Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf:
Es wird überprüft, ob die Liste der Pair-Nachrücker richtig gebildet wurde und nur Pairs enthält, die sich zusammen angemeldet haben (Da jede Küche nur genau 3 mal benutzt werden kann, werden Pairs mit überschüssigen Küchen direkt aussortiert).
Erwartetes Verhalten:
Die Liste der Pair-Nachrücker wird richtig erstellt und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
-

Testname:
CorrectPairAssignment
Vorbedingung:
Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf:
Es wird überprüft, ob die Liste der Paare richtig gebildet wird und nur Paare enthält, die sich zusammen angemeldet haben und aus unterschiedlichen Participants bestehen.
Erwartetes Verhalten:
Die Liste der Pairs wurde richtig gebildet und es wird **true** zurückgegeben.
Tatsächliches Verhalten:
-

Testname: CorrectSuccessorAssignment
Vorbedingung: Es existieren Dateien mit den Eingabedaten, deren Pfade bekannt sind. Außerdem existiert eine Instanz von InputData.
Ablauf: Es wird überprüft, ob die Küchen der Teilnehmer in den Participant- und Pair-Nachrückerlisten mehr als 3 mal verwendet werden.
Erwartetes Verhalten: Die Küchen der Teilnehmer in den Nachrückerlisten werden mehr als 3 verwendet und es wird **true** zurückgegeben.
Tatsächliches Verhalten: Die Participant-Nachrückerliste ist in diesem Fall immer leer (Siehe Test ParticipantSuccessorLoadin).

OutputData Test:

Testname	Voll. Abd.
manualTestingOfOutputCSV	nein

Testname: manualTestingOfOutputCSV
Vorbedingung: Es existieren Instanzen von allen Klassen, die benötigt werden, um die GroupList zu erstellen (insbesondere InputData und PairList müssen existieren)
Ablauf: Die Main Methode wird ausgeführt und OutputData erstellt zwei CSV Dateien (pairs und groups). Es wird manuell überprüft, ob die entstandenen Dateien den Anforderungen entsprechen.
Erwartetes Verhalten: Die entstandenen Dateien sind lesbare nicht-korrumpierte CSV-Dateien und entsprechen vollständig den Anforderungen.
Tatsächliches Verhalten: -