|  |  |  |  |
| --- | --- | --- | --- |
| Methode | #Tests | #Fehler | Voll.Abd. |
| CoupleManagerTest() | 8 | 0 | Ja |
| GroupManagerTest() | 4 | 0 | Ja |

CoupleManagerTest()

testCalcCouples() Vorbedingung: Es gibt eine Liste von unverheirateten Personen (singles), für die Paare gebildet werden sollen. Eine geographische Lage (location) ist spezifiziert. Ablauf: Die Methode givePeopleWithoutPartner() wird aufgerufen, um Paare aus den vordefinierten Singles zu bilden, ohne eine Mindest- oder Höchstanzahl von Paaren festzulegen. Erwartetes Verhalten: Die Methode bildet gültige Paare entsprechend der gegebenen Vorbedingungen. Die resultierende Liste von Paaren (couples) ist nicht leer, und jedes Paar enthält gültige Personen (person1 und person2).

testAddPerson()

Vorbedingung: Die Methode addPerson() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist.

Ablauf: Die Methode addPerson(person1) wird aufgerufen, um person1 zur Liste der einzelnen Teilnehmer (singleList) von coupleManager hinzuzufügen.Erwartetes Verhalten: Nach dem Hinzufügen von person1 zur singleList sollte person1 in der singleList von coupleManager enthalten sein. Dies wird durch die Assertion assertTrue(coupleManager.getSingleList().contains(person1)) überprüft.

testRemovePerson()

Vorbedingung: Die Methode removeSinglePerson() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist. person1 ist eine Instanz von Person, die erstellt wurde.

Ablauf: Die Methode addPerson(person1) wird aufgerufen, um person1 zur Liste der einzelnen Teilnehmer (singleList) von coupleManager hinzuzufügen. Dann wird removeSinglePerson(person1) aufgerufen, um person1 aus der singleList zu entfernen.

Erwartetes Verhalten: Nach dem Entfernen von person1 aus der singleList sollte die Aussage assertFalse(coupleManager.getSingleList().contains(person1)) wahr sein. Das bedeutet, dass person1 nicht mehr in der singleList von coupleManager enthalten ist.

testGetPerson()

Vorbedingung: Die Methode getSinglePerson() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist. person1 ist eine Instanz von Person, die erstellt wurde und wird zu coupleManager hinzugefügt.

Ablauf: Die Methode addPerson(person1) wird aufgerufen, um person1 zur Liste der einzelnen Teilnehmer (singleList) von coupleManager hinzuzufügen. Dann wird getSinglePerson(person1.getID()) aufgerufen, um die Person mit der ID von person1 aus coupleManager abzurufen und in retrievedPerson zu speichern.

Erwartetes Verhalten: Der Wert von retrievedPerson sollte gleich person1 sein, da die zurückgegebene Person mit der hinzugefügten Person person1 übereinstimmen soll. Die Assertion assertEquals(person1, retrievedPerson) sollte daher erfolgreich sein.

TestGetCouple()

Vorbedingung: Die Methode getCouple() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist. person1, person2, person3 und person4 sind Instanzen von Person, die erstellt wurden und zu coupleManager hinzugefügt werden.

Ablauf: Die Personen person1, person2, person3 und person4 werden der Liste der einzelnen Teilnehmer (singleList) von coupleManager hinzugefügt. Dann wird die Methode givePeopleWithoutPartner(singles, 0, 0, new Location(0.2, 0.4)) aufgerufen, um Paare zu bilden. Anschließend wird getCouple() aufgerufen, um die Liste der Paare (couples) von coupleManager abzurufen und in couples zu speichern.

Erwartetes Verhalten: Die Liste couples sollte nicht null sein, da nach dem Aufruf von givePeopleWithoutPartner Paare gebildet wurden und getCouple() die Liste der Paare zurückgeben soll. Die Assertion assertNotNull(couples) sollte daher erfolgreich sein.

testCalculateCost()

Vorbedingung: Die Methode calculateCost() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist. person1, person2, person3, person4, person5 und person6 sind Instanzen von Person, die entsprechend ihren Eigenschaften erstellt wurden (z. B. Alter, Küche, Ernährungspräferenz).

Ablauf: Die Methode calculateCost() wird für verschiedene Paarkombinationen aufgerufen:

Erwartetes Verhalten: Für jede Kombination sollte der berechnete Kostenwert entsprechend den implementierten Regeln in der Methode calculateCost() zurückgegeben werden. Die spezifischen Assertionen (assertEquals, assertTrue) prüfen, ob die berechneten Kostenwerte den erwarteten Bedingungen entsprechen (z. B. negativer Wert für dieselbe Person oder gleiche Küche, positiver Wert für unterschiedliche Ernährungspräferenzen, -1 für Personen ohne Küche).

testCancelPerson() Vorbedingung: Die Methode cancelPerson() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist. person1, person2, person3 und person4 sind Instanzen von Person, die entsprechend ihren Eigenschaften erstellt wurden (z. B. Alter, Küche, Ernährungspräferenz). Es gibt keine vorhandenen Paare. Ablauf: Es werden Personen (person1, person2, person3 und person4) hinzugefügt. Die Methode givePeopleWithoutPartner() wird aufgerufen, um Paare zu bilden, indem sie aus der Liste der Singles (singles) Personen auswählt und Paare bildet. Die aktuelle Anzahl der initialen Paare (initialCouplesSize) wird gespeichert. Die Methode cancelPerson() wird aufgerufen, um person1 zu stornieren und damit das Paar, in dem sie enthalten ist, zu entfernen. Erwartetes Verhalten: Nach der Stornierung von person1 wird erwartet, dass die Anzahl der Paare in updatedCouples um eins kleiner ist als die Anzahl in initialCouples. Die spezifische Assertion (assertEquals) prüft dies, um sicherzustellen, dass das Entfernen einer Person ein Paar korrekt beeinflusst hat.

testCancelCouple() Vorbedingung: Die Methode cancelCouple() der Klasse CoupleManager ist implementiert. Es gibt eine Instanz von CoupleManager namens coupleManager, die initialisiert ist. person1, person2, person3 und person4 sind Instanzen von Person, die entsprechend ihren Eigenschaften erstellt wurden (z. B. Alter, Küche, Ernährungspräferenz). Es gibt keine vorhandenen Paare. Ablauf: Personen (person1, person2, person3 und person4) werden hinzugefügt. Die Methode givePeopleWithoutPartner() wird aufgerufen, um Paare zu bilden, indem sie aus der Liste der Singles (singles) Personen auswählt und Paare bildet. Die aktuelle Anzahl der initialen Paare (initialCouplesSize) wird gespeichert. Das erste Paar (couple1) aus der Liste der initialen Paare wird storniert, indem die Methode cancelCouple() mit couple1 als Parameter aufgerufen wird. Erwartetes Verhalten: Nach der Stornierung von couple1 wird erwartet, dass die Anzahl der Paare in updatedCouples um eins kleiner ist als die Anzahl in initialCouples. Die spezifische Assertion (assertEquals) prüft dies, um sicherzustellen, dass das Entfernen eines Paares korrekt funktioniert hat und die Anzahl der Paare entsprechend aktualisiert wurde.

GroupManagerTest()

testCalcGroups()

Vorbedingung: Die Methode calcGroups() der Klasse GroupManager ist implementiert. Eine Instanz von GroupManager namens groupManager ist initialisiert. Es gibt eine Liste von couples, die Couple-Objekte enthält, die für die Gruppierung verwendet werden können. Ablauf: Die Methode calcGroups() wird aufgerufen, um Gruppen von Couple-Objekten zu erstellen, die in Group-Objekten organisiert sind. Ein Map-Objekt courseGroupMap wird erstellt, das Kurse auf Gruppen abbildet, die von groupManager.getLedger() abgerufen werden. Ein Map-Objekt coupleCounter wird initialisiert, um die Anzahl der Vorkommen jedes Paares zu zählen. Es wird überprüft, dass jede Gruppe in courseGroupMap die gleiche Anzahl von Elementen hat (mapEntrySize). Für jedes Couple in jeder Group werden mehrere Assertions durchgeführt: - Jedes Couple sollte genau einmal Gastgeber gewesen sein (h.wasHost()). - Jedes Couple sollte genau 6 andere Paare und sich selbst getroffen haben (h.getMetCouples().size()). - Jedes Couple sollte in der WithWhomAmIEating-Map eingetragen sein, ohne dass der Wert -1 ist. Es werden zusätzliche Tests für die Gruppen durchgeführt. Es wird überprüft, dass jedes Couple genau in 3 Gruppen enthalten ist. Es wird überprüft, dass die Anzahl der verarbeiteten Paare (groupManager.processedCouples.size()) gleich der Anzahl der Gruppen in groupManager.getLedger() ist. Randfälle werden getestet: - Wenn die Eingabeparameter für calcGroups() leer sind, sollte groupManager.getLedger() eine leere Liste zurückgeben. - -- Wenn weniger als 9 Paare in der Eingabeparameterliste sind, sollten keine Gruppen erstellt werden. Erwartetes Verhalten: Der Test prüft, dass die Methode calcGroups() korrekt funktioniert und Gruppen von Paaren erstellt, die bestimmte Bedingungen erfüllen, einschließlich der Anzahl der Treffen, der Gastgeberrolle und der Zuordnung zu Gruppen. Randfälle wie leere Eingaben oder weniger als 9 Paare werden ebenfalls korrekt behandelt und geprüft

testKitchen() Vorbedingung: Die Methode testKitchen() ist implementiert. Es gibt eine Instanz von groupManager, die initialisiert ist und über Daten verfügt, um Gruppen von Paaren (Group) zu erstellen. Diese Gruppen sind in GroupManager über groupManager.getLedger() zugänglich. Ablauf: Ein Map-Objekt courseGroupMap wird erstellt, das Kurse auf Gruppen von Paaren (Group) abbildet, die von groupManager.getLedger() abgerufen werden. Ein Map-Objekt kitchenCounter wird initialisiert, um die Anzahl der Verwendungen jeder Küche zu zählen. Es wird durch jede Liste von Group-Objekten (i) in courseGroupMap.values() iteriert. Für jede Group (j) wird die Küche des Gastgebers (Host) abgerufen. Es wird überprüft, ob die verwendete Küche bereits in kitchenCounter enthalten ist: - Wenn nicht, wird sie hinzugefügt und auf 1 gesetzt. - Wenn sie bereits vorhanden ist, wird der Zähler erhöht. Es wird sichergestellt, dass keine Küche mehr als 3 Mal verwendet wird (assertTrue(3 >= i, "a kitchen cant be used more than 3 times")), indem über die Werte in kitchenCounter iteriert wird.Erwartetes Verhalten: Der Test prüft, dass die Küchen innerhalb der erstellten Gruppen korrekt verwendet werden. Jede Küche sollte höchstens 3 Mal als Gastgeberküche auftreten. Dies hilft sicherzustellen, dass die Verteilung der Paare und ihrer Gastgeberküchen innerhalb der Gruppen korrekt ist und keine übermäßige Nutzung einer einzelnen Küche stattfindet.

testFillCourse() Vorbedingung: Die Methode testFillCourse() ist implementiert. GroupManager wird zurückgesetzt (GroupManager.clear()), um sicherzustellen, dass der Test in einer sauberen Umgebung läuft. Es gibt drei Paare (hostCouple1, hostCouple2, hostCouple3), die als Gastgeber für verschiedene Kurse dienen. Jedes Paar hat spezifische Eigenschaften wie Alter, Geschlecht und Ernährungspräferenz, sowie eine gemeinsame Küche (kitchen). Diese Paare sind in hostCouples gespeichert.

Ablauf: Jedes Paar in hostCouples wird der gleichen Küche (kitchen) zugeordnet, um die Testbedingungen zu vereinfachen. Die Methode fillCourse() von GroupManager wird aufgerufen und mit hostCouples und dem Kurs Course.STARTER als Parameter aufgerufen, um Gruppen zu bilden. Es wird überprüft, dass die zurückgegebene Liste groups nicht null ist. Es wird überprüft, dass genau eine Gruppe erstellt wurde (assertEquals(1, groups.size())). Die erste Gruppe (group) aus der Liste wird abgerufen und überprüft, dass sie nicht null ist. Es wird überprüft, dass die Gruppe genau drei Mitglieder hat (assertEquals(3, group.getAll().size())). Erwartetes Verhalten: Der Test prüft, ob die Methode fillCourse() korrekt funktioniert, indem sie die übergebenen Paare in eine Gruppe für den angegebenen Kurs (Course.STARTER) einfügt. Jede Gruppe sollte genau drei Mitglieder haben, was den Anforderungen entspricht, dass jedes Paar in einer Gruppe platziert wird.

testRemoveCouple() Vorbedingung: Der Test testRemoveCouple() ist implementiert. Es gibt eine Instanz von GroupManager, die vermutlich vorbereitete Gruppen und Paare enthält (processedCouples, succeedingCouples, overBookedCouples, ledger). Es gibt ein Paar (person3 und personx1), das als unknownCouple deklariert ist, um ein nicht existierendes Paar zu repräsentieren. Ablauf: Eine Liste toDelete wird erstellt, die eine Kopie der aktuellen verarbeiteten Paare (processedCouples) des GroupManagers enthält. Eine Liste l wird erstellt, die eine Kopie der aktuellen Buchungen (ledger) des GroupManagers enthält. Es wird versucht, ein nicht existierendes Paar (unknownCouple) zu entfernen, was keine Änderung bewirken sollte. Die Anzahl der Paare in der Liste succeedingCouples vor und nach dem Entfernen eines Paares wird überprüft, um sicherzustellen, dass die Anzahl um eins verringert wurde. Die Anzahl der Paare in der Liste overBookedCouples vor und nach dem Entfernen eines Paares wird überprüft, um sicherzustellen, dass die Anzahl um eins verringert wurde. Es wird überprüft, dass die Buchungen (ledger) des GroupManagers unverändert geblieben sind. Jedes Paar in toDelete wird aus dem GroupManager entfernt, und es wird überprüft, dass diese Paare nicht mehr in der Liste der verarbeiteten Paare (processedCouples) enthalten sind. Erwartetes Verhalten: Der Test überprüft die Funktionalität der Methode remove() des GroupManagers, um sicherzustellen, dass:

* Ein nicht existierendes Paar nicht entfernt wird.
* Paare erfolgreich aus den succeedingCouples und overBookedCouples entfernt werden können.
* Die Buchungen (ledger) des GroupManagers bei den Operationen unverändert bleiben.
* Die entfernten Paare nicht mehr in der Liste der verarbeiteten Paare (processedCouples) enthalten sind.