Klassentest für die Klasse model.Buchung

# Einleitung

Im Zuge des Klassentests wurde eine Testklasse (test.BuchungTest) mit Hilfe von JUnit 4 erstellt, welche die einzelnen Methoden der Klasse testet. Es folgt ein Protokoll der getesteten Methoden und Konstruktoren.

# Konstruktoren

## Buchung(Gast gast, Zimmer zimmer, Date heute, Date von, Date bis)

JUnit-Testmethode:

testBuchungGastZimmerDateDateDate()

Methodenbeschreibung:

Der Konstruktor erzeugt ein Buchungsobjekt mit einem Gastobjekt, einem Zimmerobjekt, dem Erfassungsdatum und jeweils einem Datum für den Aufenthaltsbeginn und dem Aufenthaltsende und speichert diese, in den dafür vorgesehenen Variablen.

Testbeschreibung:

Zunächst werden die für den Konstruktor erforderlichen Objekte deklariert. Daraufhin wird ein Buchungsobjekt mit Hilfe dieses Konstruktors erzeugt.

Erwartetes Ergebnis:

1. Das Buchungsobjekt ist ungleich null.

Ergebnis:

1. Das Buchungsobjekt ist ungleich null.

## Buchung(int bid)

JUnit-Testmethode:

testBuchungInt()

Methodenbeschreibung:

Der Konstruktor erzeugt ein Buchungsobjekt mit einem Integerwert für die Buchungsnummer und speichert diesen in die dafür vorgesehene Variable.

Testbeschreibung:

Für den Test wird ein Buchungsobjekt mit diesem Konstruktor erzeugt.

Erwartetes Ergebnis:

1. Das Buchungsobjekt ist ungleich null.

Ergebnis:

1. Das Buchungsobjekt ist ungleich null.

## Buchung(int bid, int zbid)

JUnit-Testmethode:

testBuchungIntInt()

Methodenbeschreibung:

Der Konstruktor erzeugt ein Buchungsobjekt mit zwei Zahlenwerten für die Buchungsnummer des Objekts sowie für die zugeordnete Zimmerbuchung und speichert diese in die dafür vorgesehenen Variablen.

Testbeschreibung:

Für den Test wird ein Buchungsobjekt mit diesem Konstruktor erzeugt.

Erwartetes Ergebnis:

1. Das Buchungsobjekt ist ungleich null.

Ergebnis:

1. Das Buchungsobjekt ist ungleich null.

# Buchungsmethoden

## addBuchung(Connection con)

JUnit-Testmethode:

testAddBuchung()

Methodenbeschreibung:

Die Methode schreibt ein Buchungsobjekt in die dafür vorgesehene Datenbanktabelle *„buchung“.* Dafür benötigt wird ein Buchungsobjekt mit Gast- und Zimmerdaten sowie mit Datumsangaben zur Erfassungszeit, Aufenthaltsbeginn und Aufenthaltsende benötigt (siehe in 2.1 getesteter Konstruktor).

Die Methode schreibt in die Datenbank die Werte für die Gastnummer, das Erfassungsdatum, den Aufenthaltsbeginn und den Aufenthaltsende.

Die durch den Insert-Befehl erzeugte Nummer aus dem Feld mit Autoinkrement-Option „BID“ der Datenbanktabelle wird in die dafür vorgesehene Variable im Buchungsobjekt abgespeichert.

Testbeschreibung:

Es werden zunächst die benötigen Objekte für Zimmer und Gast erzeugt und an den Konstruktor für das Buchungsobjekt übergeben. Mit dem Konstruktor werden auch die Datumseingaben hinterlegt.

Mit der Methode *„addBuchung“* wird daraufhin die Buchung in die Datenbank gespeichert.

Erwartetes Ergebnis:

1. Die Identifikationsnummer des Gastobjekts wurde korrekt in die Datenbank geschrieben.
2. Das Erfassungsdatum des Buchungsobjekts wurde korrekt in die Datenbank geschrieben.
3. Der Aufenthaltsbeginn des Buchungsobjekts wurde korrekt in die Datenbank geschrieben.
4. Das Aufenthaltsende des Buchungsobjekts wurde korrekt in die Datenbank geschrieben.

Ergebnis:

1. Die Identifikationsnummer des Gastobjekts wurde korrekt in die Datenbank geschrieben.
2. Das Erfassungsdatum des Buchungsobjekts wurde korrekt in die Datenbank geschrieben.
3. Der Aufenthaltsbeginn des Buchungsobjekts wurde korrekt in die Datenbank geschrieben.
4. Das Aufenthaltsende des Buchungsobjekts wurde korrekt in die Datenbank geschrieben.

## bookZimmer(Connection con)

JUnit-Testmethode:

testBookZimmer()

Methodenbeschreibung:

Die Methode schreibt eine getätigte Zimmerbuchung in die Datenbank. Dafür berechnet die Methode die Dauer des Aufenthalts mit Hilfe der Methode „*getBookedDays*“. Daraufhin schreibt sie zunächst die Buchungs- sowie die Zimmernummer des Buchung-Objekts in die dafür vorgesehene Datenbanktabelle *„zimmer-buchung“.*

Die durch den Insert-Befehl erzeugte Nummer aus dem Feld mit Autoinkrement „ZBID“ der Datenbanktabelle wird in die dafür vorgesehene Variable *„zbid“* im Buchungsobjekt abgespeichert.

Des Weiteren berechnet sie den Preis der durchgeführten Zimmerbuchung und aktualisiert den Gesamtpreis in der Datenbanktabelle *„buchung“*.

Testbeschreibung:

Zunächst wird das Buchung-Objekt aus dem zuvor durchgeführten Test (siehe Testbeschreibung 3.1) übernommen und die Methode *„bookZimmer“* aufgerufen.

Daraufhin wird ein weiteres Zimmer-Objekt mit einer anderen Zimmernummer wie das Zimmer-Objekt des originalen Buchung-Objekts erzeugt und für das Buchungs-Objekt gesetzt. Die Methode *„bookZimmer“* wird erneut aufgerufen.

Durch die beiden Zimmerbuchungen hat die Testbuchung nun zwei Zimmerbuchungen erhalten.

Erwartete Ergebnisse:

1. Die erste Zimmerbuchung enthält die Buchungsnummer des Buchung-Objekts
2. Die zweite Zimmerbuchung enthält die Buchungsnummer des Buchung-Objekts
3. Die Buchungsnummern beider Zimmerbuchungen sind identisch
4. Die Zimmernummer der ersten Zimmerbuchung ist die des ersten Zimmer-Objekts
5. Die Zimmernummer der zweiten Zimmerbuchung ist die des zweiten Zimmer-Objekts
6. Der berechnete Gesamtpreis ist korrekt.

Ergebnis:

1. Die erste Zimmerbuchung enthält die Buchungsnummer des Buchung-Objekts
2. Die zweite Zimmerbuchung enthält die Buchungsnummer des Buchung-Objekts
3. Die Buchungsnummern beider Zimmerbuchungen sind identisch
4. Die Zimmernummer der ersten Zimmerbuchung ist die des ersten Zimmer-Objekts
5. Die Zimmernummer der zweiten Zimmerbuchung ist die des zweiten Zimmer-Objekts
6. Der berechnete Gesamtpreis ist korrekt.

## bookDl(Buchung buchung, Dienstleistung dl, Connection con)

JUnit-Testmethode:

testBookDl()

Methodenbeschreibung:

Zunächst schreibt die Methode die Buchungsnummer des Buchung-Objekts sowie die Dienstleistungsnummer des Dienstleistungsobjekts und das Erfassungsdatum in die dafür vorgesehene Datenbanktabelle *„dl-buchung“.*

Die durch den Insert-Befehl erzeugte Nummer aus dem Feld mit Autoinkrement „DLBID“ der Datenbanktabelle wird in die dafür vorgesehene Variable *„dlbid“* im Buchungsobjekt abgespeichert.

Des Weiteren berechnet sie den Preis der durchgeführten Dienstleistungsbuchung und aktualisiert den entsprechenden Gesamtpreis in der Datenbanktabelle *„buchung“*.

Testbeschreibung:

Zunächst wird eine Buchung mit zwei zugehörigen Zimmern und Gast erzeugt, gebucht und somit in die Datenbank geschrieben.

Daraufhin werden zwei verschiedene Dienstleistung-Objekte erzeugt und mit der Methode *„bookDl“* auf das vorhandene Buchung-Objekt gebucht.

Erwartete Ergebnisse:

1. Die Dienstleistungsnummer der ersten Dienstleistungsbuchung ist korrekt.
2. Die Dienstleistungsnummer der zweiten Dienstleistungsbuchung ist korrekt.
3. Die Buchungsnummern der übergeordneten Buchung ist bei beiden Dienstleistungsbuchungen identisch.
4. Das Erfassungsdatum der ersten Dienstleistungsbuchung ist korrekt.
5. Das Erfassungsdatum der zweiten Dienstleistungsbuchung ist korrekt.
6. Der Gesamtpreis wird korrekt berechnet und aktualisiert.

Ergebnisse:

1. Die Dienstleistungsnummer der ersten Dienstleistungsbuchung ist korrekt.
2. Die Dienstleistungsnummer der zweiten Dienstleistungsbuchung ist korrekt.
3. Die Buchungsnummern der übergeordneten Buchung ist bei beiden Dienstleistungsbuchungen identisch.
4. Das Erfassungsdatum der ersten Dienstleistungsbuchung ist korrekt.
5. Das Erfassungsdatum der zweiten Dienstleistungsbuchung ist korrekt.
6. Der Gesamtpreis wird korrekt berechnet und aktualisiert.

## cancelZimmer (Buchung buchung, Connection con)

JUnit-Testmethode:

testCancelZimmer()

Methodenbeschreibung:

Diese Methode löscht einzelne Zimmerbuchungen aus der Tabelle *„zimmer-buchung“.* Zudem wird der Datensatz der übergeordneten Buchung aus der Tabelle *„buchung“* gelöscht, falls alle zugehörigen Zimmerbuchungen der Buchung storniert wurden. Abschließend aktualisiert die Methode den Gesamtpreis in der Tabelle *„buchung“* neu.

Testbeschreibung:

Zunächst wird eine Buchung mit zwei zugehörigen Zimmern und Gast erzeugt, gebucht und somit in die Datenbank geschrieben.

Daraufhin wird eine der beiden Zimmerbuchungen mit Hilfe der Methode *„cancelZimmer“* storniert.

Erwartete Ergebnisse:

1. Die nicht stornierte Zimmerbuchung ist weiterhin in der Datenbank
2. Die Anzahl der Zimmerbuchungen einer Buchung vor der Stornierung ist nach der Stornierung des zweiten Zimmers um eins weniger geworden.
3. Der Gesamtpreis der Buchung wurde korrekt berechnet und aktualisiert.

Ergebnisse:

1. Die nicht stornierte Zimmerbuchung ist weiterhin in der Datenbank
2. Die Anzahl der Zimmerbuchungen einer Buchung vor der Stornierung ist nach der Stornierung des zweiten Zimmers um eins weniger geworden.
3. Der Gesamtpreis der Buchung wurde korrekt berechnet und aktualisiert.

## cancelDl (Buchung buchung, Dienstleistung dl, Connection con)

JUnit-Testmethode:

testCancelDl()

Methodenbeschreibung:

Diese Methode löscht einzelne Dienstleistungsbuchungen aus der Datenbanktabelle *„dl-buchung“* und aktualisiert den Gesamtpreis in der Tabelle *„buchung“* neu.

Testbeschreibung:

Zunächst wird eine Buchung mit zwei zugehörigen Zimmern und Gast erzeugt, gebucht und somit in die Datenbank geschrieben. Daraufhin werden der Buchung noch zwei Dienstleistungen hinzugefügt. Nachdem die Buchung vollständig ist wird die zuletzt gebuchte Dienstleistung mit Hilfe der Methode *„cancelDl“* wieder storniert.

Erwartete Ergebnisse:

1. Die nicht stornierte Dienstleistungsbuchung ist weiterhin in der Datenbank
2. Die Anzahl der Dienstleistungsbuchung einer Buchung vor der Stornierung ist nach der Stornierung der zweiten Dienstleistung um eins weniger geworden.
3. Der Gesamtpreis der Buchung wurde korrekt berechnet und aktualisiert.

Ergebnisse:

1. Die nicht stornierte Dienstleistungsbuchung ist weiterhin in der Datenbank
2. Die Anzahl der Dienstleistungsbuchung einer Buchung vor der Stornierung ist nach der Stornierung der zweiten Dienstleistung um eins weniger geworden.
3. Der Gesamtpreis der Buchung wurde korrekt berechnet und aktualisiert.

## getBookedDays(Date von, Date bis)

JUnit-Testmethode:

testGetBookedDays()

Methodenbeschreibung:

Die Methode berechnet die Aufenthaltsdauer in Tagen unter der Verwendung der Datumsangaben für den Aufenthaltsbeginn und dem Aufenthaltsende und gibt diesen zurück.

Testbeschreibung:

Es wird ein Datum für den Aufenthaltsbeginn sowie ein Datum für das Aufenthaltsende erzeugt und an die Methode übergeben.

Erwartete Ergebnisse:

1. Die errechnete Dauer des Aufenthalts ist korrekt.

Ergebnisse:

1. Die errechnete Dauer des Aufenthalts ist korrekt.

## getPreis(int bid, Connection con)

JUnit-Testmethode:

testGetPreis()

Methodenbeschreibung:

Die Methode gibt den Gesamtpreis der Tabelle *„buchung“* unter Berücksichtigung der mitgegebenen Buchungsnummer zurück.

Testbeschreibung:

Es wird eine Buchung mit zwei zugehörigen Zimmern und Gast erzeugt, gebucht und somit in die Datenbank geschrieben. Anschließend wird mit der Methode *„getPreis“* der Gesamtpreis der Buchung ausgelesen.

Erwartete Ergebnisse:

1. Der Zurückgegebene Gesamtpreis ist korrekt.

Ergebnis:

1. Der Zurückgegebene Gesamtpreis ist korrekt.

# Getter- und Setter-Methoden

Mit Hilfe des JUnit-Tests wurden auch sämtliche Getter- und Setter-Methoden der Klasse getestet. Diese Tests werden innerhalb dieses Dokuments allerdings nicht näher beschrieben. Die Tests können im beigelegten Ausdruck der Test-Klasse begutachtet werden.

# Grenzwertanalyse

Um zusätzlich einen Klassentest unter Betrachtung der Grenzwerte zu erzeugen, testen wir zusätzlich die Methode *„checkBookingDate“* der Klasse control.BHHelp.

## checkBookingDate(Date vonDate, Date bisDate)

JUnit-Testmethode:

testCheckBookingDate()

Methodenbeschreibung:

Die Methode überprüft den gewählten Zeitraum auf dessen Gültigkeit. Ein Ungültigkeit wird festgestellt wenn: 1. Start- und Endtermin des Aufenthalts auf dasselbe Datum fallen, 2. Start- und oder Endtermin des Aufenthalt bereits in der Vergangenheit liegen oder 3. Ein Endtermin gewählt wird, der zeitlich vor dem Starttermin liegt. In all diesen drei Fällen wird „false“ in Verbindung mit einer GUI-Exception zurückgegeben, die den Nutzer auf eine ungültige Eingabe aufmerksam macht.

Testbeschreibung:

Es werden drei Paare, bestehend aus einem Datum für den Aufenthaltsbeginn und einem Datum für das Aufenthaltsende, erstellt.

Das erste paar repräsentiert eine korrekte Datumseingabe. Das zweite Paar bezieht sich auf den Fall, dass das Aufenthaltsende vor dem Aufenthaltsbeginn liegt und das dritte auf den Fall das der Aufenthaltsbeginn und das Aufenthaltsende identisch sind.

Für jedes der drei Paare wird die Methode *„checkBookingDate()“* aufgerufen.

Erwartetes Ergebnis:

1. Das erste Paar wird als korrekt identifiziert.
2. Das zweite Paar wird als ungültig identifiziert.
3. Das dritte Paar wird als ungültig identifiziert.

Ergebnis:

1. Das erste Paar wird als korrekt identifiziert.
2. Das zweite Paar wird als ungültig identifiziert.
3. Das dritte Paar wird als ungültig identifiziert.

Grenzwertanalyse:

Typisch für eine Grenzwertanalyse wird der Test mit den letzten noch gültigen bzw. ersten ungültigen Daten ausgeführt. Die Tests beziehen sich auf das aktuelle Datum (24.07.2014).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Voraussetzung:  Aktuelles Datum (24.07.2014) |  | | | | |
|
| Testfälle (Kontrolle des Datums) | 1 | 2 | 3 | 4 | 5 |
| Testeingabedaten |  |  |  |  |  |
|  |  |  |  |  |  |
| gewählter Starttermin | 22.08.2014 | 22.08.2014 | 20.07.2014 | 24.07.2014 | 22.08.2014 |
| gewählter Endtermin | 22.08.2014 | 23.08.2014 | 23.07.2014 | 28.07.2014 | 21.08.2014 |
|  |  |  |  |  |  |
| Wirkung: |  |  |  |  |  |
|  |  |  |  |  |  |
| Datum gültig | 0 | 1 | 0 | 1 | 0 |
| Datum nicht gültig | 1 | 0 | 1 | 0 | 1 |
| Starttermin = Endtermin | 1 | 0 | 0 | 0 | 0 |
| Starttermin > Endtermin | 0 | 0 | 0 | 0 | 1 |
| Start- u. od. Endtermin < aktuelles Datum | 0 | 0 | 1 | 0 | 0 |