Technology
Arts Sciences
TH Köln

WAWI Testspezifikation 2022

Softwarepraktikum

TH Köln - Informatik Labor

Version 8.0, 2022-03-23 08:40:45 UTC

Inhaltsverzeichnis

1. Vorgaben zur Testspezifikation im SWP

Die Testspezifikation ist ein Dokument, das die zu erstellenden Testfälle in einer strukturierten Weise kurz und prägnant beschreibt.

Die Testspezifikation umfasst alle Testfälle für die Implementierung der Methoden der Schnittstellen einer Komponente (aber **nicht** für die Methoden der Steuerungsklassen).

Die in der Testspezifikation beschriebenen Testfälle sollen in JUnit implementiert werden. Die Gestaltung der Testspezifikation ist deshalb bereits an JUnit angepasst, um eine leichte Umsetzung in JUnit zu ermöglichen. Der Text der Testspezifikation wird als Java-Kommentar des konkreten JUnit-Testfalls verwendet.

1.1. Strukturierung der Testspezifikation und der Testfälle

- In der Testspezifikation sind Testfälle für alle Methoden aller Implementierungs-Klassen der Schnittstellen der Komponente definiert.
- Es werden also die Implementierungsklassen getestet, **nicht** die Schnittstellen-Klassen.
- Für die Testfälle einer Klasse kann eine `@Before`-, `@BeforeClass`-, `@After`- und `@AfterClass` -Methode definiert werden
- Jeder einzelne Testfall besitzt als überschrift `@Test` und den Namen der später zu implementierenden JUnit-Testmethode
- Ein Testfall ist in Form einer Wenn-Dann-Regel definiert.
  - `WENN`, `UND` und `DANN` sind hierbei Schlüsselwörter.
  - Der `WENN`-Teil definiert die Voraussetzung für die Durchführung des Testfalls und den Aufruf der zu testenden Methode:
    - Dieses kann z.B. das Vorhandensein eines bestimmten Objekts sein
    - oder das Vorhandensein eines bestimmten Datenbank-Eintrags
    - oder das Nicht-Vorhandensein eines bestimmten Datenbank-Eintrags
    - oder ein bestimmter Status eines Antrags oder einer Überweisung o.ä.
    - oder ...
  - Aufruf der zu testenden Methode: hier werden die Parameterwerte vorgegeben.
  - Innerhalb des `WENN`-Teils werden die unterschiedlichen Angaben durch das Schlüsselwort `UND` voneinander getrennt. Es wird jeweils eine neue Zeile begonnen.
  - Der `DANN`-Teil definiert die erwartete Rückgabe und alle sonstigen Seiteneffekte der Methodenausführung, wie bspw. erwartete Rückgabe, Änderung des Zustands eines Objekts, Eintrag/Update/Löschung in der DB, Setzen eines Attributwertes.
  - Es gibt keine ODER Verknüpfung. ODER Verknüpfungen können nur durch mehrere Tests durchgeführt werden.
  - Für den überwiegenden Teil der Testfälle existiert ein Positiv- und ein Negativtestfall.

- Beispiel 1 eines Testfalls in der Testspezifikation:
  @Test insertKunde_00()
  `WENN` die Methode `insertKunde` mit einem Testkunden aufgerufen wird,
  `UND` die ID des Testkunden gleich `null` ist,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testkunde sollte in der DB existieren.

  @Test insertKunde_01()
  `WENN` die ID eines Testkunden mit einem Wert `ungleich null` besetzt ist,
  `UND` die Methode `insertKunde` mit dem Testkunden aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` und die DB wurde nicht verändert.

- Beispiel 2 eines Testfalls in der Testspezifikation:
  @Test getKundeById_00()
  `WENN` ein Testkunde bereits in der DB existiert,
  `UND` die Methode `getKundeById` mit der Id des Testkunden aufgerufen wird,
  `DANN` sollte sie den `Testkunden` zurückliefern.

  @Test getKundeById_01()
  `WENN` ein Testkunde nicht in der DB existiert,
  `UND` die Methode `getKundeById` mit der Id des Testkunden aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- Beispiel 3 eines Testfalls in der Testspezifikation:
  @Test getKundenListe_00()
  `WENN` x (x>0) Kunden in der DB existieren,
  `UND` die Methode `getKundenListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Kunden zurückliefern.

  @Test getKundenListe_01()
  `WENN` keine Kunden in der DB existieren,
  `UND` die Methode `getKundenListe` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- Beispiel 4 eines Testfalls in der Testspezifikation:
  @Test deleteKunde_00()
  `WENN` ein Testkunde in der DB existiert,
  `UND` die Methode `deleteKunde` mit der ID des Testkunden aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testkunde sollte nicht mehr in der DB existieren.

  @Test deleteKunde_01()
  `WENN` ein Testkunde nicht in der DB existiert,
  `UND` die Methode `deleteKunde` mit der ID des Testkunden aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern.

2. Testspezifikation

2.1. ProduktVerwaltung

2.1.1. ICRUDProdukt

- @Before: angenommen()
  `ANGENOMMEN` der EntityManager wird korrekt geholt,
  `UND` die Implementierung der ICRUDProdukt Schnittstelle wird als classUnderTest instanziiert,
  `UND` der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  `UND` die Transaktion von em wird gestartet,
  `UND` die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  `AM ENDE` wird die Transaktion zurück gesetzt.
- @Test getProduktById_00()
  `WENN` ein Testprodukt bereits in der DB existiert,
  `UND` die Methode `getProduktById` mit der Id des Testprodukts aufgerufen wird,
  `DANN` sollte sie das `Testprodukt` zurückliefern.
- @Test getProduktById_01()
  `WENN` ein Testprodukt nicht in der DB existiert,
  `UND` die Methode `getProduktById` mit der Id des Testprodukts aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.
- @Test getProduktListe_00()
  `WENN` x (x>0) Produkte in der DB existieren,
  `UND` die Methode `getProduktListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Produkten zurückliefern.
- @Test getProduktListe_01()
  `WENN` keine Produkte in der DB existieren,
  `UND` die Methode `getProduktListe` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.
- @Test getAusverkaufteProdukte_00()
  `WENN` x (x>2) Produkte in der Datenbank existieren,
  `UND` y (y>0 und y<x) Produkte mit einer Stückzahl = 0 existieren,
  `UND` die Methode `getAusverkaufteProdukte` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `y Produkten` zurückliefern.
- @Test getAusverkaufteProdukte_01()
  `WENN` x (x>0) Produkte in der Datenbank existieren,
  `UND` keine Produkte mit einer Stückzahl = 0 existieren,
  `UND` die Methode `getAusverkaufteProdukte` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.
- @Test insertProdukt_00()

WENN die Methode insertProdukt mit einem Testprodukt aufgerufen wird,
UND die ID des Testprodukts gleich null ist,
DANN sollte sie TRUE zurückliefern,
UND das Testprodukt sollte in der DB existieren.

- @Test insertProdukt_01()
  WENN die Methode insertProdukt mit einem Testprodukt aufgerufen wird,
  UND die ID des Testprodukts ungleich null ist,
  DANN sollte sie FALSE zurückliefern,
  UND die DB wurde nicht verändert.

- @Test updateProdukt_00()
  WENN ein Testprodukt in der DB existiert,
  UND die Methode updateProdukt mit einem verändertem Testprodukt (aber gleicher ID) aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND das Testprodukt sollte in der DB verändert sein.

- @Test updateProdukt_01()
  WENN ein Testprodukt nicht in der DB existiert,
  UND die Methode updateProdukt mit dem Testprodukt aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND das Testprodukt sollte nicht in der DB existieren.

- @Test deleteProdukt_00()
  WENN ein Testprodukt in der DB existiert,
  UND die Methode deleteProdukt mit der ID des Testprodukts aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND das Testprodukt sollte nicht mehr in der DB existieren.

- @Test deleteProdukt_01()
  WENN ein Testprodukt nicht in der DB existiert,
  UND die Methode deleteProdukt mit der ID des Testprodukts aufgerufen wird,
  DANN sollte sie FALSE zurückliefern.

2.1.2. ICRUDKategorie

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der ICRUDKategorie Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.

- @Test getKategorieById_00()
  WENN eine Testkategorie bereits in der DB existiert,
  UND die Methode getKategorieById mit der Id der Testkategorie aufgerufen wird,

`DANN` sollte sie die `Testkategorie` zurückliefern.

- @Test getKategorieById_01()
  `WENN` eine Testkategorie nicht in der DB existiert,
  `UND` die Methode `getKategorieById` mit der Id der Testkategorie aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getKategorieListe_00()
  `WENN` x (x>0) Kategorien in der DB existieren,
  `UND` die Methode `getKategorieListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Kategorien zurückliefern.

- @Test getKategorieListe_01()
  `WENN` keine Kategorien in der DB existieren,
  `UND` die Methode `getKategorieListe` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test insertKategorie_00()
  `WENN` die Methode `insertKategorie` mit einer Testkategorie aufgerufen wird,
  `UND` die ID der Testkategorie gleich `null` ist,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Testkategorie sollte in der DB existieren.

- @Test insertKategorie_01()
  `WENN` die Methode `insertKategorie` mit einer Testkategorie aufgerufen wird,
  `UND` die ID der Testkategorie `ungleich null` ist,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die DB wurde nicht verändert.

- @Test updateKategorie_00()
  `WENN` eine Testkategorie in der DB existiert,
  `UND` die Methode `updateKategorie` mit einer veränderten Testkategorie (aber gleicher ID) aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Testkategorie sollte in der DB verändert sein.

- @Test updateKategorie_01()
  `WENN` eine Testkategorie nicht in der DB existiert,
  `UND` die Methode `updateKategorie` mit der Testkategorie aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die Testkategorie sollte nicht in der DB existieren.

- @Test deleteKategorie_00()
  `WENN` eine Testkategorie in der DB existiert,
  `UND` die Methode `deleteKategorie` mit der ID der Testkategorie aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Testkategorie sollte nicht mehr in der DB existieren.

- @Test deleteKategorie_01()
  `WENN` eine Testkategorie nicht in der DB existiert,

`UND` die Methode `deleteKategorie` mit ID der Testkategorie aufgerufen wird,
`DANN` sollte sie `FALSE` zurückliefern.

2.1.3. ILieferProdukt

- @Before: angenommen()
  `ANGENOMMEN` der EntityManager wird korrekt geholt,
  `UND` die Implementierung der ILieferProdukt Schnittstelle wird als classUnderTest instanziiert,
  `UND` der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  `UND` die Transaktion von em wird gestartet,
  `UND` die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  `AM ENDE` wird die Transaktion zurück gesetzt.
- @Test produktStueckzahlErhoehen_00()
  `WENN` ein Testprodukt bereits in der DB existiert,
  `UND` die Methode `ProduktStueckzahlErhoehen` mit der Id des Testprodukts und einer Stückzahl x > 0 aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Stückzahl des Testprodukts sollte in der DB um x erhöht sein.
- @Test produktStueckzahlErhoehen_01()
  `WENN` ein Testprodukt bereits in der DB existiert,
  `UND` die Methode `produktStueckzahlErhoehen` mit der Id des Testprodukts und einer Stückzahl x <= 0 aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die Stückzahl des Testprodukts sollte in der DB unverändert sein.
- @Test produktStueckzahlErhoehen_02()
  `WENN` ein Testprodukt `nicht` in der DB existiert,
  `UND` die Methode `produktStueckzahlErhoehen` mit der Id des Testprodukts und einer Stückzahl x > 0 aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die Stückzahl des Testprodukts weiterhin nicht in der DB existieren.
- @Test produktStueckzahlVerringern_00()
  `WENN` ein Testprodukt bereits in der DB existiert,
  `UND` die Methode `produktStueckzahlVerringern` mit der Id des Testprodukts und einer Stückzahl x > 0 aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Stückzahl des Testprodukts sollte in der DB um x verringert sein.
- @Test produktStueckzahlVerringern_01()
  `WENN` ein Testprodukt bereits in der DB existiert,
  `UND` die Methode `produktStueckzahlVerringern` mit der Id des Testprodukts und einer Stückzahl x <= 0 aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die Stückzahl des Testprodukts sollte in der DB unverändert sein.

- @Test produktStueckzahlVerringern_02()
  `WENN` ein Testprodukt `nicht` in der DB existiert,
  `UND` die Methode `produktStueckzahlVerringern` mit der Id des Testprodukts und einer Stückzahl x > 0 aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die Stückzahl des Testprodukts weiterhin nicht in der DB existieren.

- @Test getProduktById_00()
  `WENN` ein Testprodukt bereits in der DB existiert,
  `UND` die Methode `getProduktById` mit der Id des Testprodukts aufgerufen wird,
  `DANN` sollte sie das `Testprodukt` zurückliefern.

- @Test getProduktById_01()
  `WENN` ein Testprodukt nicht in der DB existiert,
  `UND` die Methode `getProduktById` mit der Id des Testprodukts aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getProduktListe_00()
  `WENN` x (x>0) Produkte in der DB existieren,
  `UND` die Methode `getProduktListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Produkten zurückliefern.

- @Test getProduktListe_01()
  `WENN` keine Produkte in der DB existieren,
  `UND` die Methode `getProduktListe` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

2.2. LagerDaten

2.2.1. ICRUDLieferant

- @Before: angenommen()
  `ANGENOMMEN` der EntityManager wird korrekt geholt,
  `UND` die Implementierung der ICRUDLieferant Schnittstelle wird als classUnderTest instanziiert,
  `UND` der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  `UND` die Transaktion von em wird gestartet,
  `UND` die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  `AM ENDE` wird die Transaktion zurück gesetzt.

- @Test getLieferantById_00()
  `WENN` ein Testlieferant bereits in der DB existiert,
  `UND` die Methode `getLieferantById` mit der Id des Testlieferanten aufgerufen wird,
  `DANN` sollte sie den `Testlieferanten` zurückliefern.

- @Test getLieferantById_01()
  `WENN` ein Testlieferant nicht in der DB existiert,
  `UND` die Methode `getLieferantById` mit der Id des Testlieferanten aufgerufen wird,

`DANN` sollte sie `NULL` zurückliefern.

- @Test getLieferantenListe_00()
  `WENN` x (x>0) Lieferanten in der DB existieren,
  `UND` die Methode `getLieferantenListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Lieferanten zurückliefern.

- @Test getLieferantenListe_01()
  `WENN` keine Lieferanten in der DB existieren,
  `UND` die Methode `getLieferantenListe` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test insertLieferant_00()
  `WENN` die Methode `insertLieferant` mit einem Testlieferanten aufgerufen wird,
  `UND` die ID des Testlieferanten gleich `null` ist,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlieferant sollte in der DB existieren.

- @Test insertLieferant_01()
  `WENN` die Methode `insertLieferant` mit einem Testlieferanten aufgerufen wird,
  `UND` die ID des Testlieferanten `ungleich null` ist,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die DB wurde nicht verändert.

- @Test updateLieferant_00()
  `WENN` ein Testlieferant in der DB existiert,
  `UND` die Methode `updateLieferant` mit einem veränderten Testlieferanten (aber gleicher ID) aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlieferant sollte in der DB verändert sein.

- @Test updateLieferant_01()
  `WENN` ein Testlieferant nicht in der DB existiert,
  `UND` die Methode `updateLieferant` mit dem Testlieferanten aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` der Testlieferant sollte nicht in der DB existieren.

- @Test deleteLieferant_00()
  `WENN` ein Testlieferant in der DB existiert,
  `UND` die Methode `deleteLieferant` mit der ID des Testlieferanten aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlieferant sollte nicht mehr in der DB existieren.

- @Test deleteLieferant_01()
  `WENN` ein Testlieferant nicht in der DB existiert,
  `UND` die Methode `deleteLieferant` mit der ID des Testlieferanten aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern.

2.2.2. ICRUDEinlieferung

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der ICRUDEinlieferung Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.
- @Test getEinlieferungById_00()
  WENN eine Testeinlieferung bereits in der DB existiert,
  UND die Methode getEinlieferungById mit der Id der Testleinlieferung aufgerufen wird,
  DANN sollte sie die Testeinlieferung zurückliefern.
- @Test getEinlieferungById_01()
  WENN eine Testeinlieferung nicht in der DB existiert,
  UND die Methode getEinlieferungById mit der Id der Testeinlieferung aufgerufen wird,
  DANN sollte sie NULL zurückliefern.
- @Test getEinlieferungListe_00()
  WENN x (x>0) Einlieferungen in der DB existieren,
  UND die Methode getEinlieferungListe aufgerufen wird,
  DANN sollte sie eine Liste mit x Einlieferungen zurückliefern.
- @Test getEinlieferungListe_01()
  WENN keine Einlieferungen in der DB existieren,
  UND die Methode getEinlieferungListe aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.
- @Test insertEinlieferung_00()
  WENN die Methode insertEinlieferung mit einer Testeinlieferung aufgerufen wird,
  UND die ID der Testeinlieferung gleich null ist,
  DANN sollte sie TRUE zurückliefern,
  UND die Testeinlieferung sollte in der DB existieren.
- @Test insertEinlieferung_01()
  WENN die Methode insertEinlieferung mit einer Testeinlieferung aufgerufen wird,
  UND die ID der Testeinlieferung ungleich null ist,
  DANN sollte sie FALSE zurückliefern,
  UND die DB wurde nicht verändert.
- @Test updateEinlieferung_00()
  WENN eine Testeinlieferung in der DB existiert,
  UND die Methode updateEinlieferung mit einer veränderten TestEinlieferung (aber gleicher ID) aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,

`UND` die Testeinlieferung sollte in der DB verändert sein.

- @Test updateEinlieferung_01()
  `WENN` eine Testeinlieferung nicht in der DB existiert,
  `UND` die Methode `updateEinlieferung` mit der Testeinlieferung aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die Testeinlieferung sollte nicht in der DB existieren.

- @Test deleteEinlieferung_00()
  `WENN` eine Testeinlieferung in der DB existiert,
  `UND` die Methode `deleteEinlieferung` mit der ID der Testeinlieferung aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Testeinlieferung sollte nicht mehr in der DB existieren.

- @Test deleteEinlieferung_01()
  `WENN` eine Testeinlieferung nicht in der DB existiert,
  `UND` die Methode `deleteEinlieferung` mit der ID der Testeinlieferung aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern.

- @Test insertLagerverkehr_00()
  `WENN` die Methode `insertLagerverkehr` mit einem Testlagerverkehr aufgerufen wird,
  `UND` die ID des Testlagerverkehrs gleich `null` ist,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlagerverkehr sollte in der DB existieren.

- @Test insertLagerverkehr_01()
  `WENN` die Methode `insertLagerverkehr` mit einem Testlagerverkehr aufgerufen wird,
  `UND` die ID der Testlagerverkehr `ungleich null` ist,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die DB wurde nicht verändert.

- @Test updateLagerverkehr_00()
  `WENN` eine Testlagerverkehr in der DB existiert,
  `UND` die Methode `updateLagerverkehr` mit einem veränderten TestLagerverkehr (aber gleicher ID) aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlagerverkehr sollte in der DB verändert sein.

- @Test updateLagerverkehr_01()
  `WENN` ein Testlagerverkehr nicht in der DB existiert,
  `UND` die Methode `updateLagerverkehr` mit dem Testlagerverkehr aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` der Testlagervekehr sollte nicht in der DB existieren.

- @Test deleteLagerverkehr_00()
  `WENN` ein Testlagerverkehr in der DB existiert,
  `UND` die Methode `deleteLagerverkehr` mit der ID des Testlagerverkehrs aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlagerverkehr sollte nicht mehr in der DB existieren.

- @Test deleteLagerverkehr_01()
  `WENN` ein Testlagerverkehr nicht in der DB existiert,
  `UND` die Methode `deleteLagerverkehr` mit der ID des Testlagerverkehrs aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern.

- @Test insertLieferposition_00()
  `WENN` die Methode `insertLieferposition` mit einer Testlieferposition aufgerufen wird,
  `UND` die ID der Testlieferposition gleich `null` ist,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Testlieferposition sollte in der DB existieren.

- @Test insertLieferposition_01()
  `WENN` die Methode `insertLieferposition` mit einer Testlieferposition aufgerufen wird,
  `UND` die ID der Testlieferposition `ungleich null` ist,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die DB wurde nicht verändert.

- @Test updateLieferposition_00()
  `WENN` eine Testlieferposition in der DB existiert,
  `UND` die Methode `updateLieferposition` mit einer veränderten TestLieferposition (aber gleicher ID) aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die TestLieferposition sollte in der DB verändert sein.

- @Test updateLieferposition_01()
  `WENN` eine TestLieferposition nicht in der DB existiert,
  `UND` die Methode `updateLieferposition` mit der TestLieferposition aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die TestLieferposition sollte nicht in der DB existieren.

- @Test deleteLieferposition_00()
  `WENN` eine TestLieferposition in der DB existiert,
  `UND` die Methode `deleteLieferposition` mit der ID der TestLieferposition aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die TestLieferposition sollte nicht mehr in der DB existieren.

- @Test deleteLieferposition_01()
  `WENN` eine TestLieferposition nicht in der DB existiert,
  `UND` die Methode `deleteLieferposition` mit der ID der TestLieferposition aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern.

2.2.3. ILagerService

- @Before: angenommen()
  `ANGENOMMEN` der EntityManager wird korrekt geholt,
  `UND` die Implementierung der ILagerService Schnittstelle wird als classUnderTest instanziiert,
  `UND` der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  `UND` die Transaktion von em wird gestartet,
  `UND` die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  `AM ENDE` wird die Transaktion zurück gesetzt.

- @Test geAlleEinlieferungen_00()
  `WENN` x (x>0) Einlieferungen in der DB existieren,
  `UND` die Methode `getAlleEinlieferungen` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Einlieferungen zurückliefern.

- @Test getAlleEinlieferungen_01()
  `WENN` keine Einlieferungen in der DB existieren,
  `UND` die Methode `getAlleEinlieferungen` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test getEinlieferungenByDatum_00()
  `WENN` x (x>0) Einlieferungen in der DB existieren,
  `UND` y (y<x) Einlieferungen in der DB existieren im Zeitraum von a bis b,
  `UND` die Methode `getEinlieferungenByDatum` mit Zeitraum von a bis b aufgerufen wird,
  `DANN` sollte sie eine Liste mit `y` Einlieferungen zurückliefern.

- @Test getEinlieferungenByDatum_01()
  `WENN` x (x>0) Einlieferungen in der DB existieren,
  `UND` keine Einlieferungen in der DB existieren im Zeitraum von a bis b,
  `UND` die Methode `getAlleEinlieferungen` mit Zeitraum von a bis b aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test getLagerverkehrByDatum_00()
  `WENN` x (x>0) Lagerverkehre in der DB existieren,
  `UND` y (y<x) Lagerverkehre in der DB existieren im Zeitraum von a bis b,
  `UND` die Methode `getLagerverkehrByDatum` mit Zeitraum von a bis b aufgerufen wird,
  `DANN` sollte sie eine Liste mit `y` Lagerverkehren zurückliefern.

- @Test getLagerverkehrByDatum_01()
  `WENN` x (x>0) Lagerverkehre in der DB existieren,
  `UND` keine Lagerverkehre in der DB existieren im Zeitraum von a bis b,
  `UND` die Methode `getLagerverkehrByDatum` mit Zeitraum von a bis b aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test getGesamtLagerverkehr_00()
  `WENN` x (x>0) Lagerverkehre in der DB existieren,
  `UND` die Methode `getGesamtLagerverkehr` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Lagerverkehren zurückliefern.

- @Test getGesamtLagerverkehr_01()
  `WENN` keine Lagerverkehre in der DB existieren,
  `UND` die Methode `getGesamtLagerverkehr` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test getEinlieferungById_00()
  `WENN` eine Testeinlieferung bereits in der DB existiert,
  `UND` die Methode `getEinlieferungById` mit der Id der Testleinlieferung aufgerufen wird,

`DANN` sollte sie die `Testeinlieferung` zurückliefern.

- @Test getEinlieferungById_01()
  `WENN` eine Testeinlieferung nicht in der DB existiert,
  `UND` die Methode `getEinlieferungById` mit der Id der Testeinlieferung aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getLagerortById_00()
  `WENN` ein Testlagerort bereits in der DB existiert,
  `UND` die Methode `getLagerortById` mit der Id des Testlagerorts aufgerufen wird,
  `DANN` sollte sie den `Testlagerort` zurückliefern.

- @Test getLagerortById_01()
  `WENN` ein Testlagerort nicht in der DB existiert,
  `UND` die Methode `getLagerortById` mit der Id des Testlagerorts aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getLagerortListe_00()
  `WENN` x (x>0) Lagerorte in der DB existieren,
  `UND` die Methode `getLagerortListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Lagerorten zurückliefern.

- @Test getLagerortListe_01()
  `WENN` keine Lagerorte in der DB existieren,
  `UND` die Methode `getLagerortListe` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

2.2.4. ICRUDLager

- @Before: angenommen()
  `ANGENOMMEN` der EntityManager wird korrekt geholt,
  `UND` die Implementierung der ICRUDLager Schnittstelle wird als classUnderTest instanziiert,
  `UND` der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  `UND` die Transaktion von em wird gestartet,
  `UND` die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  `AM ENDE` wird die Transaktion zurück gesetzt.

- @Test insertLager_00()
  `WENN` die Methode `insertLager` mit einem Testlager aufgerufen wird,
  `UND` die ID des Testlagers gleich `null` ist,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` der Testlager sollte in der DB existieren.

- @Test insertLager_01()
  `WENN` die Methode `insertLager` mit einem Testlager aufgerufen wird,
  `UND` die ID des Testlagers `ungleich null` ist,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` die DB wurde nicht verändert.

- @Test updateLager_00()
  `WENN` ein Testlager in der DB existiert,
  `UND` die Methode `updateLager` mit einem veränderten TestLager (aber gleicher ID) aufgerufen wird,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` das Testlager sollte in der DB verändert sein.

- @Test updateLager_01()
  `WENN` ein Testlager nicht in der DB existiert,
  `UND` die Methode `updateLager` mit dem Testlager aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` das Testlager sollte nicht in der DB existieren.

- @Test getLagerById_00()
  `WENN` ein Testlager bereits in der DB existiert,
  `UND` die Methode `getLagerById` mit der Id des Testlagers aufgerufen wird,
  `DANN` sollte sie das `Testlager` zurückliefern.

- @Test getLagerById_01()
  `WENN` ein Testlager nicht in der DB existiert,
  `UND` die Methode `getLagerById` mit der Id des Testlagers aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getAlleLager_00()
  `WENN` x (x>0) Lager in der DB existieren,
  `UND` die Methode `getAlleLager` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Lagern zurückliefern.

- @Test getAlleLager_01()
  `WENN` keine Lager in der DB existieren,
  `UND` die Methode `getAlleLager` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test getLagerortById_00()
  `WENN` ein Testlagerort bereits in der DB existiert,
  `UND` die Methode `getLagerortById` mit der Id des Testlagerorts aufgerufen wird,
  `DANN` sollte sie den `Testlagerort` zurückliefern.

- @Test getLagerortById_01()
  `WENN` ein Testlagerort nicht in der DB existiert,
  `UND` die Methode `getLagerortById` mit der Id des Testlagerorts aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getLagerortListe_00()
  `WENN` x (x>0) Lagerorte in der DB existieren,
  `UND` die Methode `getLagerortListe` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `x` Lagerorten zurückliefern.

- @Test getLagerortListe_01()
  `WENN` keine Lagerorte in der DB existieren,

UND die Methode getLagerortListe aufgerufen wird,
DANN sollte sie eine leere Liste zurückliefern.

- @Test insertLagerort_00()
  WENN die Methode insertLagerort mit einem Testlagerort aufgerufen wird,
  UND die ID des Testlagerorts gleich null ist,
  DANN sollte sie TRUE zurückliefern,
  UND der Testlagerort sollte in der DB existieren.

- @Test insertLagerort_01()
  WENN die Methode insertLagerort mit einem Testlagerort aufgerufen wird,
  UND die ID des Testlagerorts ungleich null ist,
  DANN sollte sie FALSE zurückliefern,
  UND die DB wurde nicht verändert.

- @Test updateLagerort_00()
  WENN ein Testlagerort in der DB existiert,
  UND die Methode updateLagerort mit einem veränderten Testlagerort (aber gleicher ID) aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND der Testlagerort sollte in der DB verändert sein.

- @Test updateLagerort_01()
  WENN ein Testlagerort nicht in der DB existiert,
  UND die Methode updateLagerort mit dem Testlagerort aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND der Testlagerort sollte nicht in der DB existieren.

- @Test deleteLagerort_00()
  WENN ein Testlagerort in der DB existiert,
  UND die Methode deleteLagerort mit der ID des Testlagerorts aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND der Testlagerort sollte nicht mehr in der DB existieren.

- @Test deleteLagerort_01()
  WENN ein Testlagerort nicht in der DB existiert,
  UND die Methode deleteLagerort mit der ID des Testlagerorts aufgerufen wird,
  DANN sollte sie FALSE zurückliefern.

2.2.5. ILieferBestellung

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der ILieferBestellung Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()

`AM ENDE` wird die Transaktion zurück gesetzt.

- @Test getBestellungById_00()
  `WENN` eine Testbestellung bereits in der DB existiert,
  `UND` die Methode `getBestellungById` mit der Id der Testbestellung aufgerufen wird,
  `DANN` sollte sie die `Testbestellung` zurückliefern.

- @Test getBestellungById_01()
  `WENN` eine Testbestellung nicht in der DB existiert,
  `UND` die Methode `getBestellungById` mit der Id der Testbestellung aufgerufen wird,
  `DANN` sollte sie `NULL` zurückliefern.

- @Test getAlleBezahltenBestellungen_00()
  `WENN` x (x>0) Bestellungen in der DB existieren,
  `UND` y (y<x) bezahlte Bestellungen in der DB existieren,
  `UND` die Methode `getAlleBezahltenBestellungen` aufgerufen wird,
  `DANN` sollte sie eine Liste mit `y` Bestellungen zurückliefern.

- @Test getAlleBezahltenBestellungen_01()
  `WENN` x (x>0) Bestellungen in der DB existieren,
  `UND keine bezahlten` Bestellungen in der DB existieren,
  `UND` die Methode `getAlleBezahltenBestellungen` aufgerufen wird,
  `DANN` sollte sie eine `leere Liste` zurückliefern.

- @Test setBestellungVerschickt_00()
  `WENN` Bestellungen in der DB existieren,
  `UND` die Methode `setBestellungVerschickt` aufgerufen wird,
  `UND` die ID einer existierenden Bestellung übergeben wird,
  `UND` die Bestellung den `Status bezahlt` hat,
  `DANN` sollte sie `TRUE` zurückliefern,
  `UND` die Bestellung in der DB den `Status verschickt` haben.

- @Test setBestellungVerschickt_01()
  `WENN` Bestellungen in der DB existieren,
  `UND` die Methode `setBestellungVerschickt` aufgerufen wird,
  `UND` die ID einer nicht existierenden Bestellung übergeben wird,
  `DANN` sollte sie `FALSE` zurückliefern.

- @Test setBestellungVerschickt_02()
  `WENN` Bestellungen in der DB existieren,
  `UND` die Methode `setBestellungVerschickt` aufgerufen wird,
  `UND` die ID einer existierenden Bestellung übergeben wird,
  `UND` die Bestellung einen `anderen Status als bezahlt` hat,
  `DANN` sollte sie `FALSE` zurückliefern,
  `UND` der Status der Bestellung in der DB nicht verändert sein.

2.3. BestellungVerwaltungS

2.3.1. IBestellungSach

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der IBestellungSach Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.

- @Test getAlleNeuenBestellungen_00()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND y (y<x) neue Bestellungen in der DB existieren,
  UND die Methode getAlleNeuenBestellungen aufgerufen wird,
  DANN sollte sie eine Liste mit y Bestellungen zurückliefern.

- @Test getAlleNeuenBestellungen_01()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND keine neuen Bestellungen in der DB existieren,
  UND die Methode getAlleNeuenBestellungen aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.

- @Test getAlleAbgerechnetenBestellungen_00()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND y (y<x) abgerechnete Bestellungen in der DB existieren,
  UND die Methode getAlleAbgerechnetenBestellungen aufgerufen wird,
  DANN sollte sie eine Liste mit y Bestellungen zurückliefern.

- @Test getAlleAbgerechnetenBestellungen_01()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND keine abgerechneten Bestellungen in der DB existieren,
  UND die Methode getAlleAbgerechnetenBestellungen aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.

- @Test getProduktById_00()
  WENN ein TestProdukt bereits in der DB existiert,
  UND die Methode getProduktById mit der Id des TestProdukts aufgerufen wird,
  DANN sollte sie das TestProdukt zurückliefern.

- @Test getProduktById_01()
  WENN ein TestProdukt nicht in der DB existiert,
  UND die Methode getProduktById mit der Id des TestProdukts aufgerufen wird,
  DANN sollte sie NULL zurückliefern.

- @Test setBestellungRechnungGesendet_00()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungRechnungGesendet aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,

UND die Bestellung den Status neu hat,
DANN sollte sie TRUE zurückliefern,
UND die Bestellung in der DB den Status abgerechnet haben.

- @Test setBestellungRechnungGesendet_01()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungRechnungGesendet aufgerufen wird,
  UND die ID einer nicht existierenden Bestellung übergeben wird,
  DANN sollte sie FALSE zurückliefern.

- @Test setBestellungRechnungGesendet_02()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungRechnungGesendet aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung einen anderen Status als neu hat,
  DANN sollte sie FALSE zurückliefern,
  UND der Status der Bestellung in der DB nicht verändert sein.

- @Test setBestellungBezahlt_00()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungBezahlt aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung den Status abgerechnet hat,
  DANN sollte sie TRUE zurückliefern,
  UND die Bestellung in der DB den Status bezahlt haben.

- @Test setBestellungBezahlt_01()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungBezahlt aufgerufen wird,
  UND die ID einer nicht existierenden Bestellung übergeben wird,
  DANN sollte sie FALSE zurückliefern.

- @Test setBestellungBezahlt_02()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungBezahlt aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung einen anderen Status als abgerechnet hat,
  DANN sollte sie FALSE zurückliefern,
  UND der Status der Bestellung in der DB nicht verändert sein.

- @Test setBestellungStorniert_00()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungStorniert aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung den Status abgerechnet hat,
  DANN sollte sie TRUE zurückliefern,
  UND die Bestellung in der DB den Status storniert haben.

- @Test setBestellungStorniert_01()

WENN Bestellungen in der DB existieren,
UND die Methode `setBestellungStorniert` aufgerufen wird,
UND die ID einer existierenden Bestellung übergeben wird,
UND die Bestellung den `Status neu` hat,
DANN sollte sie `TRUE` zurückliefern,
UND die Bestellung in der DB den `Status storniert` haben.

- @Test setBestellungStorniert_02()
  WENN Bestellungen in der DB existieren,
  UND die Methode `setBestellungStorniert` aufgerufen wird,
  UND die ID einer nicht existierenden Bestellung übergeben wird,
  DANN sollte sie `FALSE` zurückliefern.

- @Test setBestellungStorniert_03()
  WENN Bestellungen in der DB existieren,
  UND die Methode `setBestellungStorniert` aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung nicht den `Status abgerechnet` oder nicht den `Status neu` hat,
  DANN sollte sie `FALSE` zurückliefern,
  UND der Status der Bestellung in der DB nicht verändert sein.

- @Test updateBestellung_00()
  WENN eine TestBestellung in der DB existiert,
  UND die Methode `updateBestellung` mit einer veränderten TestBestellung (aber gleicher ID) aufgerufen wird,
  DANN sollte sie `TRUE` zurückliefern,
  UND die TestBestellung sollte in der DB verändert sein.

- @Test updateBestellung_01()
  WENN eine TestBestellung nicht in der DB existiert,
  UND die Methode `updateBestellung` mit der TestBestellung aufgerufen wird,
  DANN sollte sie `FALSE` zurückliefern,
  UND die TestBestellung sollte nicht in der DB existieren.

- @Test deleteBestellung_00()
  WENN eine TestBestellung in der DB existiert,
  UND die Methode `deleteBestellung` mit der ID der TestBestellung aufgerufen wird,
  DANN sollte sie `TRUE` zurückliefern,
  UND die TestBestellung sollte nicht mehr in der DB existieren.

- @Test deleteBestellung_01()
  WENN eine TestBestellung nicht in der DB existiert,
  UND die Methode `deleteBestellung` mit der ID der TestBestellung aufgerufen wird,
  DANN sollte sie `FALSE` zurückliefern.

- @Test getBestellungById_00()
  WENN eine TestBestellung bereits in der DB existiert,
  UND die Methode `getBestellungById` mit der Id der TestBestellung aufgerufen wird,
  DANN sollte sie die `TestBestellung` zurückliefern.

- @Test getBestellungById_01()
  WENN eine TestBestellung nicht in der DB existiert,
  UND die Methode `getBestellungById` mit der Id der TestBestellung aufgerufen wird,
  DANN sollte sie `NULL` zurückliefern.

- @Test insertBestellposition_00()
  WENN die Methode `insertBestellposition` mit einer TestBestellposition aufgerufen wird,
  UND die ID der TestBestellposition gleich `null` ist,
  DANN sollte sie `TRUE` zurückliefern,
  UND die TestBestellposition sollte in der DB existieren.

- @Test insertBestellposition_01()
  WENN die Methode `insertBestellposition` mit einer TestBestellposition aufgerufen wird,
  UND die ID der TestlBestellposition `ungleich null` ist,
  DANN sollte sie `FALSE` zurückliefern,
  UND die DB wurde nicht verändert.

- @Test updateBestellposition_00()
  WENN eine TestBestellposition in der DB existiert,
  UND die Methode `updateBestellposition` mit einer veränderten TestBestellposition (aber gleicher ID) aufgerufen wird,
  DANN sollte sie `TRUE` zurückliefern,
  UND die TestBestellposition sollte in der DB verändert sein.

- @Test updateBestellposition_01()
  WENN eine TestBestellposition nicht in der DB existiert,
  UND die Methode `updateBestellposition` mit der TestBestellposition aufgerufen wird,
  DANN sollte sie `FALSE` zurückliefern,
  UND die TestBestellposition sollte nicht in der DB existieren.

- @Test deleteBestellposition_00()
  WENN eine TestBestellposition in der DB existiert,
  UND die Methode `deleteBestellposition` mit der ID der TestBestellposition aufgerufen wird,
  DANN sollte sie `TRUE` zurückliefern,
  UND die TestBestellposition sollte nicht mehr in der DB existieren.

- @Test deleteBestellposition_01()
  WENN eine TestBestellposition nicht in der DB existiert,
  UND die Methode `deleteBestellposition` mit der ID der TestBestellposition aufgerufen wird,
  DANN sollte sie `FALSE` zurückliefern.

- @Test getBestellpositionById_00()
  WENN eine TestBestellposition bereits in der DB existiert,
  UND die Methode `getBestellpositionById` mit der Id der TestBestellposition aufgerufen wird,
  DANN sollte sie die `TestBestellposition` zurückliefern.

- @Test getBestellpositionById_01()
  WENN eine TestBestellposition nicht in der DB existiert,
  UND die Methode `getBestellpositionById` mit der Id der TestBestellposition aufgerufen wird,

DANN sollte sie NULL zurückliefern.

2.3.2. INachrichtSach

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der INachrichtSach Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.
- @Test sendNachrichtAnKunde_00()
  WENN die zu sendende Nachricht vom Typ ankunde ist,
  UND der Status der Nachricht ungelesen ist,
  UND die übergebene Kunden ID in der DB existiert,
  UND die Methode sendNachrichtAnKunde mit der Nachricht aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND die Nachricht in der DB vorhanden sein.
- @Test sendNachrichtAnKunde_01()
  WENN die zu sendende Nachricht vom Typ anwawi ist,
  UND der Status der Nachricht ungelesen ist,
  UND die übergebene Kunden ID in der DB existiert,
  UND die Methode sendNachrichtAnKunde mit der Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.
- @Test sendNachrichtAnKunde_02()
  WENN die zu sendende Nachricht vom Typ ankunde ist,
  UND der Status der Nachricht gelesen ist,
  UND die übergebene Kunden ID in der DB existiert,
  UND die Methode sendNachrichtAnKunde mit der Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.
- @Test sendNachrichtAnKunde_03()
  WENN die zu sendende Nachricht vom Typ ankunde ist,
  UND der Status der Nachricht ungelesen ist,
  UND die übergebene Kunden ID nicht in der DB existiert,
  UND die Methode sendNachrichtAnKunde mit der Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.
- @Test getAlleNachrichten_00()
  WENN x (x>0) Nachrichten in der DB existieren,
  UND y (y<x) Nachrichten mit Status anwawi in der DB existieren,
  UND die Methode getAlleNachrichten aufgerufen wird,

DANN sollte sie eine Liste mit y Nachrichten zurückliefern.

- @Test getAlleNachrichten_01()
  WENN x (x>0) Nachrichten in der DB existieren,
  UND keine Nachrichten mit Status anwawi in der DB existieren,
  UND die Methode getAlleNachrichten aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.

- @Test setNachrichtGelesen_00()
  WENN Nachrichten in der DB existieren,
  UND die Methode setNachrichtGelesen aufgerufen wird,
  UND die ID einer existierenden Nachricht übergeben wird,
  DANN sollte sie TRUE zurückliefern,
  UND die Nachricht in der DB den Status gelesen haben.

- @Test setNachrichtGelesen_01()
  WENN Nachrichten in der DB existieren,
  UND die Methode setNachrichtGelesen aufgerufen wird,
  UND die ID einer nicht existierenden Nachricht übergeben wird,
  DANN sollte sie FALSE zurückliefern,
  UND der Status der Nachricht in der DB nicht verändert sein.

- @Test rechnungAnKundeSenden_00()
  WENN die zu sendende Nachricht (Rechnung) vom Typ ankunde ist,
  UND der Status der Nachricht ungelesen ist,
  UND der übergebene Kunde existiert in der DB ,
  UND die Methode rechnungAnKundeSenden mit Kunden und Nachricht aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND die Nachricht in der DB vorhanden sein.

- @Test rechnungAnKundeSenden_01()
  WENN die zu sendende Nachricht vom Typ anwawi ist,
  UND der Status der Nachricht ungelesen ist,
  UND der übergebene Kunde existiert in der DB,
  UND die Methode rechnungAnKundeSenden mit Kunde und Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.

- @Test rechnungAnKundeSenden_02()
  WENN die zu sendende Nachricht vom Typ ankunde ist,
  UND der Status der Nachricht gelesen ist,
  UND der übergebene Kunde existiert in der DB,
  UND die Methode rechnungAnKundeSenden mit Kunde und Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.

- @Test rechnungAnKundeSenden_03()
  WENN die zu sendende Nachricht vom Typ ankunde ist,
  UND der Status der Nachricht ungelesen ist,

UND der übergebene Kunden nicht in der DB existiert,
UND die Methode rechnungAnKundeSenden mit Kunden und Nachricht aufgerufen wird,
DANN sollte sie FALSE zurückliefern,
UND die Nachricht nicht in der DB vorhanden sein.

2.3.3. IBestellungAdmin

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der IBestellungAdmin Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.
- @Test getAlleBestellungen_00()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND die Methode getAlleBestellungen aufgerufen wird,
  DANN sollte sie eine Liste mit x Bestellungen zurückliefern.
- @Test getAlleBestellungen_01()
  WENN keine Bestellungen in der DB existieren,
  UND die Methode getAlleBestellungen aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.
- @Test getBestellungenByDatum_00()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND y (y<x) Bestellungen in der DB existieren im Zeitraum von a bis b,
  UND die Methode getBestellungenByDatum mit Zeitraum von a bis b aufgerufen wird,
  DANN sollte sie eine Liste mit y Bestellungen zurückliefern.
- @Test getBestellungenByDatum_01()
  WENN x (x>0) Bestellunegn in der DB existieren,
  UND keine Bestellungen in der DB existieren im Zeitraum von a bis b,
  UND die Methode getBestellungenByDatum mit Zeitraum von a bis b aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.
- @Test getBestellungById_00()
  WENN eine Testbestellung bereits in der DB existiert,
  UND die Methode getBestellungById mit der Id der Testbestellung aufgerufen wird,
  DANN sollte sie die Testbestellung zurückliefern.
- @Test getBestellungById_01()
  WENN eine Testbestellung nicht in der DB existiert,
  UND die Methode getBestellungById mit der Id der Testbestellung aufgerufen wird,
  DANN sollte sie NULL zurückliefern.

2.3.4. IKundeService

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der IKundeService Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.
- @Test getAktiveProdukte_00()
  WENN x (x>2) Produkte in der Datenbank existieren,
  UND y (y>0 und y<x) aktive Produkte existieren,
  UND die Methode `getAktiveProdukte` aufgerufen wird,
  DANN sollte sie eine Liste mit `y Produkten` zurückliefern.
- @Test getAktiveProdukte_01()
  WENN x (x>0) Produkte in der Datenbank existieren,
  UND keine aktiven Produkte existieren,
  UND die Methode `getAktiveProdukte` aufgerufen wird,
  DANN sollte sie eine `leere Liste` zurückliefern.

2.4. KundeDaten

2.4.1. ICRUDKunde

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der ICRUDKunde Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.
- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.
- @Test getKundeById_00()
  WENN ein Testkunde bereits in der DB existiert,
  UND die Methode `getKundeById` mit der Id des Testkunden aufgerufen wird,
  DANN sollte sie den `Testkunden` zurückliefern.
- @Test getKundeById_01()
  WENN ein Testkunde nicht in der DB existiert,
  UND die Methode `getKundeById` mit der Id des Testkunden aufgerufen wird,
  DANN sollte sie `NULL` zurückliefern.
- @Test getKundenListe_00()
  WENN x (x>0) Kunden in der DB existieren,

UND die Methode getKundenListe aufgerufen wird,
DANN sollte sie eine Liste mit x Kunden zurückliefern.

- @Test getKundenListe_01()
  WENN keine Kunden in der DB existieren,
  UND die Methode getKundenListe aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.

- @Test insertKunde_00()
  WENN die Methode insertKunde mit einem Testkunden aufgerufen wird,
  UND die ID des Testkunden gleich null ist,
  DANN sollte sie TRUE zurückliefern,
  UND der Testkunde sollte in der DB existieren.

- @Test insertKunde_01()
  WENN die ID eines Testkunden mit einem Wert ungleich null besetzt ist,
  UND die Methode insertKunde mit dem Testkunden aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND und die DB wurde nicht verändert.

- @Test updateKunde_00()
  WENN ein Testkunde in der DB existiert,
  UND die Methode updateKunde mit einem verändertem Testkunden (aber gleicher ID) aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND der Testkunde sollte in der DB verändert sein.

- @Test updateKunde_01()
  WENN ein Testkunde nicht in der DB existiert,
  UND die Methode updateKunde mit dem Testkunden aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND der Testkunde sollte nicht in der DB existieren.

- @Test deleteKunde_00()
  WENN ein Testkunde in der DB existiert,
  UND die Methode deleteKunde mit der ID des Testkunden aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND der Testkunde sollte nicht mehr in der DB existieren.

- @Test deleteKunde_01()
  WENN ein Testkunde nicht in der DB existiert,
  UND die Methode deleteKunde mit der ID des Testkunden aufgerufen wird,
  DANN sollte sie FALSE zurückliefern.

2.5. BestellungVerwaltungK

2.5.1. IBestellungKunde

- @Before: angenommen()

ANGENOMMEN der EntityManager wird korrekt geholt,
UND die Implementierung der IBestellungKunde Schnittstelle wird als classUnderTest instanziiert,
UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
UND die Transaktion von em wird gestartet,
UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.

- @Test bestellungAnlegen_00()
  WENN die Methode bestellungAnlegen mit einer Testbestellung aufgerufen wird,
  UND die ID der Testbestellung gleich null ist,
  UND der Status der Testbestellung neu ist,
  DANN sollte sie TRUE zurückliefern,
  UND die Testbestellung sollte in der DB existieren.

- @Test bestellungAnlegen_01()
  WENN die ID einer Testbestellung mit einem Wert ungleich null besetzt ist,
  UND die Methode bestellungAnlegen mit der Testbestellung aufgerufen wird,
  UND der Status der Testbestellung neu ist,
  DANN sollte sie FALSE zurückliefern,
  UND und die DB wurde nicht verändert.

- @Test bestellungAnlegen_02()
  WENN die ID einer Testbestellung mit dem Wert null besetzt ist,
  UND die Methode bestellungAnlegen mit der Testbestellung aufgerufen wird,
  UND der Status der Testbestellung nicht neu ist,
  DANN sollte sie FALSE zurückliefern,
  UND und die DB wurde nicht verändert.

- @Test getBestellungenDesKunden_00()
  WENN x (x>0) Bestellungen in der DB existieren,
  UND y (y>0 und y<x) Bestellung für einen Testunden existieren,
  UND die Methode getBestellungenDesKunden mit der ID des Testkunden aufgerufen wird,
  DANN sollte sie eine Liste mit y Bestellungen zurückliefern.

- @Test getBestellungenDesKunden_01()
  WENN x (x>0) Bestellungen in der Datenbank existieren,
  UND keine Bestellungen für einen Testkunden existieren,
  UND die Methode getBestellungenDesKunden mit der Id des Testkunden aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.

- @Test setBestellungStorniert_00()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungStorniert aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung den Status neu hat,
  DANN sollte sie TRUE zurückliefern,
  UND die Bestellung in der DB den Status storniert haben.

- @Test setBestellungStorniert_01()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungStorniert aufgerufen wird,
  UND die ID einer nicht existierenden Bestellung übergeben wird,
  DANN sollte sie FALSE zurückliefern.

- @Test setBestellungStorniert_02()
  WENN Bestellungen in der DB existieren,
  UND die Methode setBestellungStorniert aufgerufen wird,
  UND die ID einer existierenden Bestellung übergeben wird,
  UND die Bestellung nicht den Status neu hat,
  DANN sollte sie FALSE zurückliefern,
  UND der Status der Bestellung in der DB nicht verändert sein.

- @Test getProduktById_00()
  WENN ein Testprodukt bereits in der DB existiert,
  UND die Methode getProduktById mit der Id des Testprodukts aufgerufen wird,
  DANN sollte sie das Testprodukt zurückliefern.

- @Test getProduktById_01()
  WENN ein Testprodukt nicht in der DB existiert,
  UND die Methode getProduktById mit der Id des Testprodukts aufgerufen wird,
  DANN sollte sie NULL zurückliefern.

2.5.2. INachrichtKunde

- @Before: angenommen()
  ANGENOMMEN der EntityManager wird korrekt geholt,
  UND die Implementierung der INachrichtKunde Schnittstelle wird als classUnderTest instanziiert,
  UND der EntityManager wird per setEntityManager Methode der classUnderTest gesetzt,
  UND die Transaktion von em wird gestartet,
  UND die Daten der betreffenden Entitäten wurden in der DB gelöscht.

- @After: amEnde()
  AM ENDE wird die Transaktion zurück gesetzt.

- @Test sendeNachrichtAnSach_00()
  WENN die zu sendende Nachricht vom Typ anwawi ist,
  UND der Status der Nachricht ungelesen ist,
  UND die übergebene Kunden in der Nachricht in der DB existiert,
  UND die Methode sendNachrichtAnSach mit der Nachricht aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,
  UND die Nachricht in der DB vorhanden sein.

- @Test sendeNachrichtAnSach_01()
  WENN die zu sendende Nachricht vom Typ ankunde ist,
  UND der Status der Nachricht ungelesen ist,
  UND der übergebene Kunde in der Nachricht in der DB existiert,
  UND die Methode sendeNachrichtAnSach mit der Nachricht aufgerufen wird,

DANN sollte sie FALSE zurückliefern,
UND die Nachricht nicht in der DB vorhanden sein.

- @Test sendeNachrichtAnSach_02()
  WENN die zu sendende Nachricht vom Typ anwawi ist,
  UND der Status der Nachricht gelesen ist,
  UND der übergebene Kunde in der Nachricht in der DB existiert,
  UND die Methode sendeNachrichtAnSach mit der Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.

- @Test sendeNachrichtAnSach_03()
  WENN die zu sendende Nachricht vom Typ anwawi ist,
  UND der Status der Nachricht ungelesen ist,
  UND der übergebene Kunde in der Nachricht nicht in der DB existiert,
  UND die Methode sendeNachrichtAnSach mit der Nachricht aufgerufen wird,
  DANN sollte sie FALSE zurückliefern,
  UND die Nachricht nicht in der DB vorhanden sein.

- @Test getEigeneNachrichten_00()
  WENN x (x>0) Nachrichten in der DB existieren,
  UND y (y<x) Nachrichten eines Kunden in der DB existieren,
  UND die Methode getEigeneNachrichten mit der ID des Kunden aufgerufen wird,
  DANN sollte sie eine Liste mit y Nachrichten zurückliefern.

- @Test getEigeneNachrichten_01()
  WENN x (x>0) Nachrichten in der DB existieren,
  UND keine Nachrichten eines bestimmten Kunden in der DB existieren,
  UND die Methode getEigeneNachrichten mit der ID des Kunden aufgerufen wird,
  DANN sollte sie eine leere Liste zurückliefern.

- @Test setNachrichtGelesen_00()
  WENN Nachrichten in der DB existieren,
  UND die Methode setNachrichtGelesen aufgerufen wird,
  UND die ID einer existierenden Nachricht übergeben wird,
  DANN sollte sie TRUE zurückliefern,
  UND die Nachricht in der DB den Status gelesen haben.

- @Test setNachrichtGelesen_01()
  WENN Nachrichten in der DB existieren,
  UND die Methode setNachrichtGelesen aufgerufen wird,
  UND die ID einer nicht existierenden Nachricht übergeben wird,
  DANN sollte sie FALSE zurückliefern,
  UND der Status der Nachricht in der DB nicht verändert sein.

- @Test deleteNachricht_00()
  WENN eine Testnachricht in der DB existiert,
  UND die Methode deleteNachricht mit der ID der Testnachricht aufgerufen wird,
  DANN sollte sie TRUE zurückliefern,

`UND` die Testnachricht sollte nicht mehr in der DB existieren.

- @Test deleteNachricht_01()
  `WENN` eine Testnachricht nicht in der DB existiert,
  `UND` die Methode `deleteNachricht` mit der ID der Testnachricht aufgerufen wird,
  `DANN` sollte sie `FALSE` zurückliefern.