## Inleiding.

Met de onderstaande opgaven ga je de API van de peristenceManager goed leren kennen. Bestudeer en gebruik hoofdstuk 6 van het boek [[Beginning Java EE 7](http://antoniogoncalves.org/2013/05/29/beginning-java-ee-7-book-arriving-soon/)] om de volgende opgaven te maken. Dit boek zal ook gebruiken in de afstudeerfase voorgeschreven worden in semester 6.

***Voorbereiding***.

### De database

Download en installeer een [MySql](http://www.mysql.com/) database[[1]](#footnote-1) en je favoriete database IDE. Mocht je nog geen favoriet hebben is [Wampserver](http://www.wampserver.com/en/) wellicht een kandidaat: database en IDE via één installatie. Ook NetBeans zelf biedt een database IDE. Zie bij voorbeeld [deze tutorial](http://netbeans.org/kb/docs/ide/mysql.html).

### Libraries

### Op sharepoint is in het bestand ’bank.zip’ een uitgewerkte JPA applicatie te vinden. Bestudeer de bank applicatie goed.Wellicht valt je op dat het bestand geen standaard Netbeans Java project is maar een Maven java project. Maven faciliteert een uniform buildproces. Het grote voordeel van het gebruik van Maven komt met name naar voren in het projectwerk. Doordat alle gebruikte libraries op een centrale plaats bijgehouden worden hoef je dit niet meer op elke PC van de projectleden apart te doen

In SE42 gaan we EclipseLink als JPA implementatie gebruiken. Het toevoegen van de benodigde libraries aan het project gaat als volgt:

Klik hiervoor met de rechtermuisknop op het punt *Dependencies* en kies A*dd Dependency*.

Via het query veld gaan we de juiste library selecteren. Vul *org.eclipse.persistence* in. Klap vervolgens de regel met eclipselink open en kies de nieuwste stabiele jar.

Het invoegen van de MySql connector gaat op een analoge manier: Vul *mysql* in en het queryveld, klap vervolgens de regel *mysql:mysql-connector-java* open en selecteer de laatste stabiele jar.

**Opmerking**: Clean en rebuild regelmatig

Bij het aanpassen van persisitence.xml, entities wellicht ook andere klassen wordt door NetBeans geen adequate automatische clean en rebuild gedaan. Doe dit daarom regelmatig zelf.

## Opdracht 1

De code uit het bestand bank.zip vormt de basis van de volgende opdrachten. Elke subopdracht bestaat uit een (gedeelte van) testscipt. Integreer de testscipts in de bank applicatie. Zorg ervoor dat voor elk testscript de database leeg gemaakt wordt. Je kunt hiervoor het script DatabaseCleaner gebruiken.

Voor elke test moet je in ieder geval de volgende vragen beantwoorden:

1. Wat is de waarde van asserties en printstatements? Corrigeer verkeerde asserties zodat de test ‘groen’ wordt.
2. Welke SQL statements worden gegenereerd?
3. Wat is het eindresultaat in de database?
4. Verklaring van bovenstaande drie observaties.

De antwoorden op de vragen kun je als commentaar bij de testen vastleggen.

|  |  |
| --- | --- |
|  | **Vraag** |
|  |  |
| **1** | **Hoe werken persist en commit in samenhang met de database.** |
|  | Account account = new Account(111L);  em.getTransaction().begin();  em.persist(account);  //TODO: verklaar en pas eventueel aan  assertNull(account.getId());  em.getTransaction().commit();  System.out.println("AccountId: " + account.getId());  //TODO: verklaar en pas eventueel aan  assertTrue(account.getId() > 0L); |
|  |  |
| **2** | **Rollback** |
|  | Account account = new Account(111L);  em.getTransaction().begin();  em.persist(account);  assertNull(account.getId());  em.getTransaction().rollback();  // TODO code om te testen dat table account geen records bevat. Hint: bestudeer/gebruik AccountDAOJPAImpl |
|  |  |
| **3** | **Flushen maar** |
|  | Long expected = -100L;  Account account = new Account(111L);  account.setId(expected);  em.getTransaction().begin();  em.persist(account);  //TODO: verklaar en pas eventueel aan  //assertNotEquals(expected, account.getId();  em.flush();  //TODO: verklaar en pas eventueel aan  //assertEquals(expected, account.getId();  em.getTransaction().commit();  //TODO: verklaar en pas eventueel aan |
|  |  |
| **4** | **Veranderingen na de persist** |
|  | Long expectedBalance = 400L;  Account account = new Account(114L);  em.getTransaction().begin();  em.persist(account);  account.setBalance(expectedBalance);  em.getTransaction().commit();  assertEquals(expectedBalance, account.getBalance());  //TODO: verklaar de waarde van account.getBalance  Long acId = account.getId();  account = null;  EntityManager em2 = emf.createEntityManager();  em2.getTransaction().begin();  Account found = em2.find(Account.class, acId);  //TODO: verklaar de waarde van found.getBalance  assertEquals(expectedBalance, found.getBalance()); |
|  |  |
| **5** | **Refresh** |
|  | In de vorige opdracht verwijzen de objecten account en found naar dezelfde rij in de database. Pas een van de objecten aan, persisteer naar de database. Refresh vervolgens het andere object om de veranderde state uit de database te halen. Test met asserties dat dit gelukt is. |
|  |  |
| **6** | **Merge** |
|  | Merge is een van de lastigere methoden uit JPA api. Het is belangrijk dat je deze opgave daarom zorgvuldig uitvoert.  Account acc = new Account(1L);  Account acc2 = new Account(2L);  Account acc9 = new Account(9L);  // scenario 1  Long balance1 = 100L;  em.getTransaction().begin();  em.persist(acc);  acc.setBalance(balance1);  em.getTransaction().commit();  //TODO: voeg asserties toe om je verwachte waarde van de attributen te verifieren.  //TODO: doe dit zowel voor de bovenstaande java objecten als voor opnieuw bij de entitymanager opgevraagde objecten met overeenkomstig Id.  // scenario 2  Long balance2a = 211L;  acc = new Account(2L);  em.getTransaction().begin();  acc9 = em.merge(acc);  acc.setBalance(balance2a);  acc9.setBalance(balance2a+balance2a);  em.getTransaction().commit();  //TODO: voeg asserties toe om je verwachte waarde van de attributen te verifiëren.  //TODO: doe dit zowel voor de bovenstaande java objecten als voor opnieuw bij de entitymanager opgevraagde objecten met overeenkomstig Id.  // HINT: gebruik acccountDAO.findByAccountNr  // scenario 3  Long balance3b = 322L;  Long balance3c = 333L;  acc = new Account(3L);  em.getTransaction().begin();  Account acc2 = em.merge(acc);  assertTrue(em.contains(acc)); // verklaar  assertTrue(em.contains(acc2)); // verklaar  assertEqual(acc,acc2); //verklaar  acc2.setBalance(balance3b);  acc.setBalance(balance3c);  em.getTransaction().commit();  //TODO: voeg asserties toe om je verwachte waarde van de attributen te verifiëren.  //TODO: doe dit zowel voor de bovenstaande java objecten als voor opnieuw bij de entitymanager opgevraagde objecten met overeenkomstig Id.  // scenario 4  Account account = new Account(114L);  account.setBalance(450L);  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  em.persist(account);  em.getTransaction().commit();  Account account2 = new Account(114L);  Account tweedeAccountObject = account2;  tweedeAccountObject.setBalance(650l);  assertEquals((Long)650L,account2.getBalance()); //verklaar  account2.setId(account.getId());  em.getTransaction().begin();  account2 = em.merge(account2);  assertSame(account,account2); //verklaar  assertTrue(em.contains(account2)); //verklaar  assertFalse(em.contains(tweedeAccountObject)); //verklaar  tweedeAccountObject.setBalance(850l);  assertEquals((Long)650L,account.getBalance()); //verklaar  assertEquals((Long)650L,account2.getBalance()); //verklaar  em.getTransaction().commit();  em.close(); |
|  |  |
| **7** | **Find en clear** |
|  | Account acc1 = new Account(77L);  em.getTransaction().begin();  em.persist(acc1);  em.getTransaction().commit();  //Database bevat nu een account.  // scenario 1  Account accF1;  Account accF2;  accF1 = em.find(Account.class, acc1.getId());  accF2 = em.find(Account.class, acc1.getId());  assertSame(accF1, accF2);  // scenario 2  accF1 = em.find(Account.class, acc1.getId());  em.clear();  accF2 = em.find(Account.class, acc1.getId());  assertSame(accF1, accF2);  //TODO verklaar verschil tussen beide scenario's |
|  |  |
| **8** | **Remove** |
|  | Account acc1 = new Account(88L);  em.getTransaction().begin();  em.persist(acc1);  em.getTransaction().commit();  Long id = acc1.getId();  //Database bevat nu een account.  em.remove(acc1);  assertEquals(id, acc1.getId());  Account accFound = em.find(Account.class, id);  assertNull(accFound);  //TODO: verklaar bovenstaande asserts |
| **9** | **Generation type** |
|  | Opgave 1 heb je uitgevoerd met @GeneratedValue(strategy = GenerationType.IDENTITY)  Voer dezelfde opdracht nu uit met GenerationType SEQUENCE en TABLE.  Verklaar zowel de verschillen in testresultaat als verschillen van de database structuur.  Sequence zorgt ervoor dat de id van een sequence word opgenomen ipv een auto incremental id. En table kijkt naar de tabel zoals die nu is. |
|  |  |
|  |  |

## Beoordeling

|  |  |  |
| --- | --- | --- |
| **4** | **7** | **10** |
| Opgave 1, 2, 4 en 6 vrijwel helemaal goed gemaakt. | Enkele kleine fouten in de realisatie van de opgaven of verantwoording. | Correcte realisatie van de opdrachten en verantwoording van de gemaakte keuzes. |

# Bijlage 1: Benodige kennis over het Java Collections Framework.

De interface **java.util.Map<K,V>** specificeert een verzameling objectparen. Elk paar bestaat uit een 'key' van type K en een 'value' van type V. Een Map kan geen gelijke keys bevatten.

('Gelijk' betekent binnen het Java Collections Framework altijd dat de methode *equals* van de betreffende klasse **true** teruggeeft.) De belangrijks methoden van Map zijn:

**V put(K key, V value)** Deze voegt een objectpaar toe aan de Map. Als de key nog niet in de Map aanwezig was, wordt null teruggegeven. Als de key wel aanwezig is wordt de oude value vervangen door de tweede parameter en teruggegeven als functiewaarde.

**V get(Object key)** Deze geeft de value terug die hoort bij de gegeven key of null als de key niet aanwezig is in de map.

In klasse UserRepository wordt verder gebruikt maakt van

**Collection<V> values( )** Deze geeft een verzameling terug van alle values die zich in de map bevinden. De afzonderlijke values kunnen verkregen worden d.m.v. een Iterator, die geleverd wordt door methode iterator( ) van de Collection.

Een **java.util.Iterator** levert de waarden van de onderliggende verzamelingen een voor een. De belangrijkste methoden zijn:

**next( )** Deze levert het volgende element.

**hasNext( )** Geeft **true** terug als de iterator nog elementen bevat.

De interface Map wordt o.a. geïmplementeerd door de klassen **HashMap** en **TreeMap.** In een HashMap kan de value die bij een key hoort zeer snel gevonden worden. In een TreeMap kan het zoeken iets langzamer gaan, maar worden de keys gesorteerd opgeslagen, zodat het gemakkelijk is om voorgangers en opvolgers te vinden. Objecten van type HashMap en TreeMap zijn niet persistent Ze zijn wel Serializable en kunnen dus in een file opgeslagen worden, maar zijn daar niet doorzoekbaar.

1. Je mag ook een andere database gebruiken. Expertise over configuratie en drivers dien je zelf te leveren. [↑](#footnote-ref-1)