Testen persistenter Objekte

Roman Heilig und Tom Kretschmer

Seminar „Bildanalyse und Simulation mit Java“ im SS 2004
Universität Ulm

Gliederung

- Einführung
- CRM-System
- Persistente Attrappe

Gliederung

- Einführung
- CRM-System
- Persistente Attrappe
- Testen der Persistenzimplementierung
- Testen mit Datenbanken
- Interaktion von Persistenzschicht und Client
- Zusammenfassung

Einführung
Begriffserklärung

- Die meisten objektorientierten Programme haben ein gemeinsames Problem:

  Ein Teil ihres Zustandes soll so abgespeichert werden, daß er das Ende des Programmablauf überlebt

- Diesen Vorgang bezeichnet man als Persistierung

- Einleitendes Beispiel: Highscore-Objekt

Einführung
Persistenzmechanismen

- Abspeichern der Objektattribute in einer Datei
- Abspeichern der Objekte in einer relationalen Datenbank (RDBMS)
- Verwendung einer objektorientierten Datenbank (OODBMS)

Einführung
Persistenzmechanismen

- Abspeichern der Objektattribute in einer Datei
- Abspeichern der Objekte in einer relationalen Datenbank (RDBMS)
- Verwendung einer objektorientierten Datenbank (OODBMS)

Trotz der Verfügbarkeit von objektorientierten Datenbanken wird hier nur die RDBMS betrachtet

Einführung
Probleme beim Testen persistenter Objekte

- Zugriff auf externes Persistenzmedium dauert deutlich länger als auf normale Objekte
- Umfangreiches Vorhandensein von unterstützenden Objekten
- Kein Müll darf hinterlassen werden
- Netzwerk sorgt für zusätzliche Laufzeitverzögerungen

Einführung
Probleme beim Testen persistenter Objekte

- Zugriff auf externes Persistenzmedium dauert deutlich länger als auf normale Objekte
- Umfangreiches Vorhandensein von unterstützenden Objekten
- Kein Müll darf hinterlassen werden
- Netzwerk sorgt für zusätzliche Laufzeitverzögerungen

$\Rightarrow$ Persistenter Testfrust!

Einführung
Ein möglicher Ausweg

- Durch Kombination aus bestimmten Designprinzipien und Techniken kann man Unit Tests für und mit persistenten Objekten den Schrecken nehmen kann

- Hierzu Einführung eines CRM-Systems (Customer Relationship Management)

CRM-System

- CRM-System (*Customer Relationship Management*) ist eine kleine Applikation zur Pflege von Kundenbeziehungen
- Im Zentrum steht der Kunde (`Customer`), der immer *genau einer* Kategorie (`CustomerCategory`) zugeordnet ist und mit dem beliebig viele Kundenkontakte (`CustomerContact`) stattgefunden haben
- Naiver Testansatz
- Statische Datenbankklasse

CRM-System
Programmtext

```
public class CRMDatabase {

   public static void initialize(String dbURL)
     throws CRMException {...}

   public static void shutdown()
     throws CRMException {...}
```

CRM-System
Programmtext

```
public static CustomerCategory
 createCategory(String name)
  throws CRMException {...}

public static void
 deleteCategory(CustomerCategory category)
  throws CRMException {...}

public static Set allCategories()
  throws CRMException {...}
```

CRM-System
Programmtext

```
    public static Customer createCustomer(String name,
     CustomerCategory) throws CRMException {...}

    public static void writeCustomer(Customer customer)
      throws CRMException {...}

    public static void delete Customer(Customer customer)
      throws CRMException {...}

    public static Customer getCustomer(String id)
      throws CRMException {...}

    public Set allCustomers(CustomerCategory category)
      throws CRMException {...}
}
```

CRM-System
Daily Report

- Wichtige Aufgabe von CRM-Systemen ist die Erstellung regelmäßiger Reports
- Ein täglicher Report (`DailyReport`) beispielsweise ermittelt die Anzahl der Kundenkontakte mit bestimmten Kategorien von Kunden
- Ausschnitt aus der zugehörigen Testklasse

CRM-System
Daily Report Testklasse - Programmtext

```
public class DailyReportTest extends TestCase{

  private DailyReport report;
  private Calendar reportDate;
  private CustomerCategory catFortune100,
    catSmallCompany;
  private final String DB_URL = "jdbc:odbc:CRM";
  private List customersToDelete = new ArrayList();
```

CRM-System
Daily Report Testklasse - Programmtext

```
protected void setUp() throws Exception {
   reportDate = Calendar.getInstance();
   report = new DailyReport(reportDate);
   CRMDatabase.initialize(DB_URL);

   catFortune100 = CRMDatabase.createCategory
     ("fortune 100");
   catSmallCompany = CRMDatabase.createCategory
     ("small company");
}
```

CRM-System
Daily Report Testklasse - Programmtext

```
protected void tearDown() throws Exception {
   Iterator i = customersToDelete.iterator();
   while (i.hasNext()) {
      Customer each = (Customer) i.next();
      CRMDatabase.deleteCustomer(each);
   }

   CRMDatabase.deleteCategory(catFortune100);
   CRMDatabase.deleteCategory(catSmallCompany);
   CRMDatabase.shutdown();
}
```

CRM-System
Daily Report Testklasse - Programmtext

```
public void testAllContacts()
  throws Exception {

  Customer customer1 = CRMDatabase.createCustomer
     ("Customer 1", catFortune100);
  customerToDelete.add(customer1);
  Customer customer2 = ...
  Customer customer3 = ...
```

CRM-System

Daily Report Testklasse - Programmtext

```
Calendar dayBefore = (Calendar) reportDate.clone();
dayBefore.add(Calendar.DATE, -1);

customer1.addContact(dayBefore, "note 1");
customer1.addContact(reportDate, "note 2");
CRMDatabase.writeCustomer(customer1);

customer2.addContact(reportDate, "note 3");
CRMDatabase.writeCustomer(customer2);

customer3.addContact(dayBefore, "note 4");
CRMDatabase.writeCustomer(customer3);
```

CRM-System
Daily Report Testklasse - Programmtext

```
 List contacts =
    report.allContactsForCategory(catFortune100);
 assertEquals(1, contacts.size());
  //...
  }
}
```

CRM-System
Daily Report Testklasse - Probleme

- Keine Unabhängigkeit
- `tearDown()` erfordert aktive Mitarbeit im Testcode von `customersToDelete.add(...)` innerhalb der Testmethode
- Abhängigkeit von der Verfügbarkeit der Datenbank

Testen persistenter Objekte
Abstraktion der Persistenzschnittstelle

- Wir brechen diese Abhängigkeit, indem wir die Persistenz in einem abstrakten Interface kapseln
- Dadurch Trennung von Persistenzschnittstelle und Datenbankanbindung

Testen persistenter Objekte

Interface - Programmtext

```
public interface CRMPersistence {
  void shutdown() throws CRMException;
  CustomerCategory createCategory(String name)
    throws CRMException;
  void delete Category(CustomerCategory category)
    throws CRMException;
  Set allCategories()
    throws CRMException;
  Customer createCustomer (String name,
   CustomerCategory category)
    throws CRMException;
```

Testen persistenter Objekte

Interface - Programmtext

```
  void writeCustomer(Customer customer)
    throws CRMException;
  void deleteCustomer(Customer customer)
    throws CRMException;
  Customer getCustomer(String id)
    throws CRMException;
  Set allCustomers(CustomerCategory category)
    throws CRMException;
}
```

Persistente Attrappe
Abstraktion der Persistenzschnittstelle

- Alle bislang statischen Methoden der Klasse `CRMDatabase` finden sich nun in `CRMPersistence` wieder
- Ausnahme: `initialize(String url)` ist selbst ein Implementierungsdetail und hat daher nichts im abstrakten Interface verloren
- Mehrere Implementierungen des Interface sind möglich

Persistente Attrappe
Dummy-Implementierung

```
public class DummyCRMPersistence implements CRMPersistence {
  private int id = 0;
  private Set customers = new HashSet();
  private Set categories = new HashSet();
  public CustomerCategory createCategory(String name)
    throws CRMException {
    CustomerCategory category = new CustomerCategory(name);
    categories.add(category);
    return category;
  }
```

Persistente Attrappe
Dummy-Implementierung

```
public Set allCategories()
 throws CRMException {
  return categories;
}

public Customer createCustomer(String name,
 CustomerCategory category)
  throws CRMException {
  Customer customer = new Customer(name, category);
  customers.add(customer);
  return customer;
}
```

Persistente Attrappe
Dummy-Implementierung

```
  public Set allCustomers(CustomerCategory category)
    throws CRMException {
    return customers
  }
  ...
}
```

Persistente Attrappe
Dummy-Implementierung

- Man beachte, dass `allCustomers(...)` alle erzeugten Kunden zurückgibt und keine Filterung nach Kategorie vornimmt
- Prinzip: Dummy-Objekt so einfach wie möglich
- Test darf nur passende `Customer`-Objekte erzeugen

Persistente Attrappe

Test mit Dummy - Programmtext

```
public class DailyReportTest extends TestCase
  {
  private DailyReport report;
  private CRMPersistence persistence;
  private Calendar reportDate;
  private CustomerCategory catFortune100;
  public DailyReportTest(String name) {...}
```

Persistente Attrappe

Test mit Dummy - Programmtext

```
protected void setUp() throws Exception {
  persistence = new DummyCRMPersistence();
  reportDate = Calendar.getInstance();

  report = new DailyReport(persistence, reportDate);

  catFortune100 = persistence.createCategory(
    "fortune100");
}
```

Persistente Attrappe

Test mit Dummy - Programmtext

```
public void testAllContacts()
 throws Exception {
  Customer customer1 = persistence.createCustomer(
    "Customer 1", catFortune100);

  Customer customer2 = persistence.createCustomer(
    "Customer2", catFortune100);

  Calendar dayBefore = (Calendar)
   reportDate.clone();
  dayBefore.add(Calendar.DATE, -1);
```

Persistente Attrappe

Test mit Dummy - Programmtext

```
 customer1.addContact(dayBefore, "note 1");
 customer1.addContact(reportDate, "note 2");
 customer2.addContact(dayBefore, "note 4");

 List contacts =
  report.allContactsForCategory(catFortune100);
 assertEquals(1, contacts.size());
 //...
 }
}
```

Persistente Attrappe

Test mit Dummy

- Einzige bemerkenswerte Ergänzung im geänderten Test ist, dass der Konstruktor von DailyReport nun zusätzlich nach einem `CRMPPersistence`-Objekt verlangt
- Einiges an datenbankspezifischem Code sowie die komplette `tearDown()`-Methode werden nun nicht mehr benötigt
- Dummy-Klasse erlaubt uns jetzt, das korrekte Verhalten der Fehlerbehandlung zu testen
- Alternative: Mock-Objekt

Testen der Persistenzimplementierung
Einführung

- Bisher nur auf Schnittstelle aufgebautem Code getestet
- Implementierung der Persistenzschnittstelle ebenfalls testen

Testen der Persistenzimplementierung
Gestaltung der Datenbankschnittstelle

Grundannahmen:

- Unterscheidung in selbstständige und abhängige Objekte
- Query- und Retrieval-Methoden
- Methoden durch Transaktionen geschützt
- CRMExceptions

Testen der Persistenzimplementierung
Was ist zu testen ?

- Objekte erzeugen/löschen/modifizieren
- Query-Methoden
- Test von Constraint-Verletzungen
- Transaktionsinterface
- Weitere Methoden oder Funktionen

Testen der Persistenzimplementierung
Objekterzeugung(1)

```
public class CRMDatabaseTest extends TestCase   {
    private CRMDatabase basedata;
    protected void setUp() throws Exception   {
        database = new CRMDatabase("jdbc:odbc:CRM");
    }
    protected void tearDown() throws Exeption   {
        if (database.isConnected())   {
            Iterator i = database.allCategories().iterator();
            while (i.hasNext())   {
                CustomerCategory each =
                (CustomerCategory) i.next();
                this.deleteCategoryAndDependentCustomers(each);
            }
            database.shutdown();
    }   }
```

Testen der Persistenzimplementierung
Objekterzeugung(2)

```
private void deleteCategoryAndDependentCustomers{
             CustomerCategory category} throws CRMException {
    Iterator i = database.allCustomers(category).iterator();
    while (i.hasNext())  {
        Customer each = (Customer) i.next();
        database.deleteCustomer(each);
    }
    database.deleteCategory(category);
}
```

Testen der Persistenzimplementierung
Objekterzeugung(3)

```
public void testCustomerCreation() throws Exception  {
    CustomerCategory cat = database.createCategory("cat");
    Customer customer1 =
        database.createCustomer("customer1", cat);
    Customer retrieved1 =
        database.getCustomer(customer1.getId());
    assertEquals(customer1, retrieved1);
    assertEquals(customer1.getName() , retrieved1.getName());
    asertEquals(customer1.getCategory(),
                retrieved1.getCategory());
```

Testen der Persistenzimplementierung
Objekterzeugung(4)

```
        Customer customer2 =
            database.createCustomer("customer2", cat);
        Customer retrieved2 =
            database.getCustomer(customer2.getId());
        assertEquals(customer2, retrieved2);
        Set allCustomers = database.allCustomers(cat);
        assertEquals(2, allCustomers.size());
        assertTrue(allCustomers.contains(customer1));
        assertTrue(allCustomers.contains(customer2));
        }
}
```

Testen der Persistenzimplementierung
Query-Methoden(1)

```
public void testAllCustomers() throws Exception  {
    CustomerCategory cat1 = database.createCategory("cat1");
    CustomerCategory cat2 = database.createCategory("cat2");
    Customer customer1 =
        database.createCustomer("customer1", cat1);
    Customer customer2 =
        database.createCustomer("customer2", cat2);
    Customer customer3 =
        database.createCustomer("customer3", cat1);
    Set cat1Customers = database.allCustomers(cat1);
```

Testen der Persistenzimplementierung
Query-Methoden(2)

```
        assertEquals(2, cat1Customers.size());
        assertTrue(cat1Customers.contains(customer1));
        assertTrue(cat1Customers.contains(customer3));
        Set cat2Customers = database.allCustomers(cat2);
        assertEquals(1, cat2Customers.size());
        assertTrue(cat2Customers.contains(customer2));
    }
```

Testen der Persistenzimplementierung
Constraint-Verletzungen

```
public void testCategoryDeletionCustomerFailure()
    throws Exception   {
    CustomerCategory cat =
            database.createCategory("Category 1");
    Customer cust = database.createCustomer("customer1", cat);
    try  {
        database.deleteCategory(cat);
        fail("CRMException expected");
    } catch (CRMException expected) {}
    database.deleteCustomer(cust);
    database.deleteCustomer(cat);
}
```

Testen der Persistenzimplementierung
Transaktionsinterface(1)

```
public interface CRMTransaction  {
    Object run() throws Exception;
}


public interface CRMPersistence  {
    ...
    Object executeTransaction(CRMTransaction transaction)
        throws CRMException;
}
```

Testen der Persistenzimplementierung
Transaktionsinterface(2)

```
public void testExecuteTransactionRollback() throws Exception {
    CRMTransaction t = new CRMTransaction()   {
        public Object run() throws CRMException  {
            database.createCategory("cat1");
            //should fail and rollback:
            database.createCategory("cat1");
            return database.allCategories();
    }   };
    try  {
        database.executeTransaction(t);
        fail("CRMException should have been thrown");
    } catch (CRMException expected)  {}
    assertTrue(database.allCategories().isEmpty());
}
```

Testen mit Datenbanken
Ansätze für Testdatenkonsistenz

- Bisheriger Ansatz: exklusive Testdatenbank
- Multiuser-Datenbank

Testen mit Datenbanken
4-Datenbank-Ansatz

- Produktionsdatenbank
- Lokale Entwicklerdatenbank
- Entwicklerdatenbank mit realistischem Datenbestand
- Deployment-Datenbank

Testen mit Datenbanken
Beschleunigung der Testsuite

- Mit einer Transaktion
- Mit einem SQL-Skript
- JDBC-Mocks

Testen mit Datenbanken
Evolution der Persistenztechnologie

Welcher Persistenzmechanismus wird für ein Projekt verwendet ?

- Java Properties Klasse
- Serialisierung in Java
- kostenlose SQL-Datenbank
- komerzielle Datenbanksysteme

Interaktion von Persistenzschicht und Client

- Einzelner Test von Client und Interface genügt aus theoretischer Sicht meistens nicht
- Auch Zusammenspiel benachbarter Objekte muss untersucht werden

Wie viele Interaktionstests sind notwendig ?

Zusammenfassung

- Unit Tests für Persistenzmechanismen wird häufig von Schwierigkeiten begleitet
- Abstraktes Interface vereinfacht Testfälle stark
- Nur wenige Interaktionstests nötig

Testen persistenter Objekte
Quellen

J. Link: *Unit Tests mit Java*, erschienen im dpunkt.verlag, 2002

http://dbunit.sourceforge.net

http://www.dallaway.com/acad/dbunit.html