JOHANNES KEPLER
UNIVERSITÄT LINZ

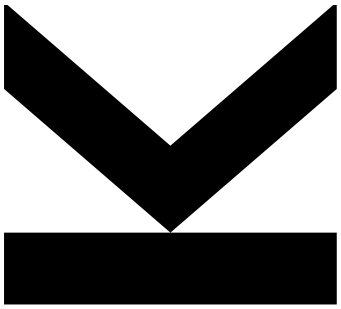
Eingereicht von
Hoedt, Mayer, Pfoser (Team 12)

Angefertigt am
Institute of Software Engineering

LVA-Leiter
Dr. Reinhold Plösch, Mag. Rudolf Ramler

4. April 2016

1st Assignment



Unit Testing

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Inhaltsverzeichnis

1 Assignment Part A: Unit testing the RingBuffer

The following JUnit-Tests were written in order to test the functionallity of the Java Class RingBuffer. There are some comments available for each test case, so the expected outcome of all test cases should be clear. See the written unit tests below.

```
/**
  * Capacity for the buffer that is used in the tests.
  * In order to avoid trivial buffers, it should not be less than 2.
  */
private static final int SIZE = 10;
 /**
  * Default element to add to the buffer that can be used in the tests.
  * The tests were designed assuming that this element is not {@code null}.
  */
 private static final String ELEMENT = "item";

 private RingBuffer<String> emptyBuffer;

 /** Provides a new empty buffer for each test. */
 @Before
 public void setUp() {
     emptyBuffer = new RingBuffer<>(SIZE);
 }

 /** Test normal initialisation of RingBuffer. */
@Test
public void testRingBufferInitialization() {
   RingBuffer<String> buffer = new RingBuffer<>(SIZE);

   assertEquals(0, buffer.size());
   assertTrue(buffer.isEmpty());
}

 /**
  * Tests initialisation of empty RingBuffer.
  * The assumption is that a (useless) buffer is expected upon return.
  */
 @Test
 public void testRingBufferInitializationZeroCapacity() {
     RingBuffer<String> buffer = new RingBuffer<>(0);

     assertEquals(0, buffer.size());
     assertTrue(buffer.isEmpty());
 }

 /**
  * Tests initialisation of empty RingBuffer.
  * The assumption is that a (possibly useless) buffer is expected upon return.
  */
@Test
public void testRingBufferInitializationNegativeCapacity() {
     RingBuffer<String> buffer = new RingBuffer<>(-SIZE);

     assertEquals(0, buffer.size());
     assertTrue(buffer.isEmpty());
 }

 /** Tests enqueue operation of the buffer. */
 @Test
 public void testEnqueue() throws RingBufferException {
     for(int i = 1; i <= SIZE; i++) {
         emptyBuffer.enqueue(ELEMENT);
         assertEquals(i, emptyBuffer.size());
     }

     assertFalse(emptyBuffer.isEmpty());
 }
```

```java
/**
 * Tests enqueue operation of {@code null} element.
 * The assumption is that {@code null} elements should not cause any trouble.
 */
@Test
public void testEnqueueNull() throws RingBufferException {
    emptyBuffer.enqueue(null);

    assertEquals(1, emptyBuffer.size());
    assertFalse(emptyBuffer.isEmpty());
}

/** Tests dequeue operation of the buffer. */
@Test
public void testDequeue() throws RingBufferException {
    // fill empty buffer
    for(int i = 0; i < SIZE; i++) {
        emptyBuffer.enqueue(ELEMENT + i);
    }

    // make sure the enqueues did their job
    assertEquals(SIZE, emptyBuffer.size());
    assertFalse(emptyBuffer.isEmpty());

    for(int i = 0; i < SIZE / 2; i++) {
        assertEquals(ELEMENT + i, emptyBuffer.dequeue());
    }
}

/** Tests if underflow is correctly detected. */
@Test(expected = RingBufferException.class)
public void testRingBufferUnderflow() throws RingBufferException {
   emptyBuffer.dequeue(); // should throw RingBufferException
}

/** Tests if overflow is correctly detected. */
@Test(expected = RingBufferException.class)
public void testRingBufferOverflow() throws RingBufferException {
    // fill buffer
    for (int i = 0; i < SIZE; ++i) {
        try {
            emptyBuffer.enqueue(ELEMENT);
        } catch (RingBufferException e) {
            // exception only expected when emptyBuffer is already full
            fail("Unexpected Buffer overflow");
        }
    }

    emptyBuffer.enqueue(ELEMENT);
}

/** Tests if the iterator over the buffer works as expected. */
@Test
public void testRingBufferIterator() throws RingBufferException {
    // fill buffer
    for(int i = 0; i < SIZE; i++) {
        emptyBuffer.enqueue(ELEMENT + i);
    }

    // explicit use of iterator
    Iterator<String> it = emptyBuffer.iterator();
    for(int i = 0; i < SIZE; i++) {
        assertTrue(it.hasNext());
        assertEquals(ELEMENT + i, it.next());
    }

    // implicit use of iterator
    int i = 0;
    for(String test : emptyBuffer) {
        assertEquals(ELEMENT + i++, test);
    }
}
```

```
/** Tests if improper use of the iterator behaves as expected */
@Test(expected = NoSuchElementException.class)
public void testRingBufferIteratorImproperUse() throws RingBufferException {
    emptyBuffer.iterator().next();
}
```

2 Assignment Part B: Unit tests for code of your own choice

For the second part of the assignment, we used some code from last semester's course "Service Engineering PR". There we had to develope a Java backend for a Webservice. The backend uses a (In Memory-) Neo4J graph database for the data storage.

The test class `at.archkb.server.test.neo4j.ArchProfileServiceTest.java` tests the CRUD functionallity of the Java Class `ArchProfileService.java`. See the written unit tests below.

```
/**
 * Class under test
 */
@Autowired
private ArchProfileService archProfileService;

/**
 * Provides CRUD-Functionality for Users
 */
@Autowired
private UserService userService;

/**
 * Provides possibilities to communicate with the graph database
 */
@Autowired
private Neo4jOperations template;

/**
 * Clears the DB and inits two user to test authentication
 */
@Before
public void clearDB() {
    final String clearall = "Match (n) Optional Match (n)-[r]-(n2) Delete n,r,n2";
    template.query(clearall, new HashMap<String, String>());
    log.info("DB cleared");
    initUsers();
}

/**
 * Creates one Architectural Profile with all possible properties in the
 * database and loads this object afterwards from the database
 * @throws Exception
 */
@Test
public void creatAndGetArchProfile() throws Exception {
    log.info("Create ArchProfile");

    SecurityContextHolder.getContext()
            .setAuthentication(new UsernamePasswordAuthenticationToken("admin@archkb.at", "password"));

    ArchProfile created = initArchProfiles();

    /*
     * Saving the Object in the db forces an update of the object To compare
     * the object with the one loaded from the db the object is cloned
     */
    ArchProfile copy = (ArchProfile) ObjectCloner.deepCopy(created);
```

```
        archProfileService.createArchProfile(created);

        log.info("ArchProfile created");

        // load all Profiles
        ArchProfile loaded = archProfileService.getArchProfiles().iterator().next();
        Assert.assertEquals(loaded, copy);

        // load one Profile
        loaded = archProfileService.getArchProfile(loaded.getId());
        Assert.assertEquals(loaded, copy);
    }

    /**
     * Tries to retrieve an Architecture Profile with a non existing id
     */
    @Test
    public void getProfileWithNonExistingID() {
        log.info("Get ArchProfile with non existing ID");

        SecurityContextHolder.getContext()
                .setAuthentication(new UsernamePasswordAuthenticationToken("admin@archkb.at", "password"));

        ArchProfile loaded = archProfileService.getArchProfile(new Long(-1));
        Assert.assertEquals(loaded, new ArchProfile());
    }

    /**
     * Tries to create an Architecture Profile with null Value
     */
    @Test(expected = IllegalArgumentException.class)
    public void createArchProfileWithNullValue() {
        log.info("Create ArchProfile with null Value");

        SecurityContextHolder.getContext()
                .setAuthentication(new UsernamePasswordAuthenticationToken("admin@archkb.at", "password"));

        archProfileService.createArchProfile(null);
    }

    /**
     * Tries to create an Architectural Profile without authentication
     */
    @Test(expected = AuthenticationCredentialsNotFoundException.class)
    public void createArchProfileUnauthorized() {
        log.info("Create ArchProfile without Authrization");

        ArchProfile create = new ArchProfile();
        create.setTitle("Test Title");
        archProfileService.createArchProfile(create);
    }

    /**
     * Tries to create an Architectural Profile with wrong username
     */
    @Test(expected = BadCredentialsException.class)
    public void createArchProfilewithWrongUser() {
        log.info("Create ArchProfile with wrong User");

        SecurityContextHolder.getContext()
                .setAuthentication(new UsernamePasswordAuthenticationToken("admin@wronguser.at", "password"));

        ArchProfile create = new ArchProfile();
        create.setTitle("Test Title");
        archProfileService.createArchProfile(create);
    }

    /**
     * Tries to create an Architectural Profile with wrong password
     */
    @Test(expected = BadCredentialsException.class)
```

```
public void createArchProfilewithWrongPassword() {
    log.info("Create ArchProfile with wrong Password");

    SecurityContextHolder.getContext()
            .setAuthentication(new UsernamePasswordAuthenticationToken("admin@archkb.at", "wrongpassword"));

    ArchProfile create = new ArchProfile();
    create.setTitle("Test Title");
    archProfileService.createArchProfile(create);
}

/**
 * Tries to update an existing profile and checks if the changes took effect
 * in the database Deletion of properties of the Architecture Profile will
 * not take effect, since this behavior will be treated in a sepperate class
 * @throws Exception
 */
@Test
public void updateArchProfile() throws Exception {
    log.info("Update ArchProfile");

    String user = "admin@archkb.at";
    SecurityContextHolder.getContext().setAuthentication(new UsernamePasswordAuthenticationToken(user, "password"));

    ArchProfile profile = createProfileforUpdate(user);

    profile = archProfileService.getArchProfile(profile.getId());

    // change root property
    profile.setTitle("updated Title");

    // change properties of related nodes
    profile.getArchProfileQualityattributes().forEach(qa -> {
        qa.setDescription("new Description");
        qa.getQualityAttribute().setName("new Name");
    });

    // change properties of related nodes
    profile.getDiagrams().forEach(dia -> {
        dia.setPath("new path");
    });

    // add new node
    Diagram diagram = new Diagram();
    diagram.setPath("new Dia");
    profile.getDiagrams().add(diagram);

    ArchProfile updated = archProfileService.updateArchProfile(profile);

    int newDia = 0;
    for(Diagram dia:updated.getDiagrams()){
        if(dia.getPath().equals("new Dia")){
            newDia++;
        }else{
            Assert.assertEquals(dia.getPath(), "new path");
        }
    }
    Assert.assertEquals(newDia, 1);

    for(ArchProfileQualityattribute qa:updated.getArchProfileQualityattributes()){
        Assert.assertEquals(qa.getDescription(), "new Description");
        Assert.assertEquals(qa.getQualityAttribute().getName(), "new Name");
    }
}

/**
 * Tries to update an existing profile with an user who is not the owner of
 * the profile
 *
 * @throws IOException
 * @throws OptimisticLockingException
 */
```

```
@Test(expected = AccessDeniedException.class)
public void updateArchProfileWithNotOwner() throws IOException, OptimisticLockingException {
    log.info("Update ArchProfile with wrong user");
    String user = "admin@archkb.at";
    SecurityContextHolder.getContext().setAuthentication(new UsernamePasswordAuthenticationToken(user, "password"));

    ArchProfile profile = createProfileforUpdate(user);

    // Reauthenticate with wrong user
    SecurityContextHolder.getContext()
            .setAuthentication(new UsernamePasswordAuthenticationToken("user@archkb.at", "password"));

    // change root property
    profile.setTitle("updated Title");

    archProfileService.updateArchProfile(profile);
}

/**
 * Tries to delete an existing profile and checks if the changes took effect
 * in the database
 */
@Test
public void deletArchProfile() {
    log.info("Delete ArchProfile");

    String user = "admin@archkb.at";
    SecurityContextHolder.getContext().setAuthentication(new UsernamePasswordAuthenticationToken(user, "password"));
    ArchProfile profile = createProfileforUpdate(user);

    archProfileService.deleteArchProfile(profile.getId());

    ArchProfile loaded = archProfileService.getArchProfile(profile.getId());

    Assert.assertEquals(loaded, new ArchProfile());
}

/**
 * Tries to delete an existing profile with an user who is not the owner of
 * the profile
 */
@Test(expected = AccessDeniedException.class)
public void deletArchProfileWithNotOwner() {
    log.info("Delete ArchProfile with wrong User");

    String user = "admin@archkb.at";
    SecurityContextHolder.getContext().setAuthentication(new UsernamePasswordAuthenticationToken(user, "password"));

    ArchProfile profile = createProfileforUpdate(user);

    // Reauthenticate with wrong user
    SecurityContextHolder.getContext()
            .setAuthentication(new UsernamePasswordAuthenticationToken("user@archkb.at", "password"));

    archProfileService.deleteArchProfile(profile.getId());
}
```