

Agile Software Development

Produced
by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE



Pacemaker Tests

- Model
- API
- Serializer

pacemaker model

pacemaker model

```
public class User
{
    static Long    counter = 0L;

    public Long    id;
    public String  firstName;
    public String  lastName;
    public String  email;
    public String  password;

    public Map<Long, Activity> activities = new HashMap<>();

    //...
}
```

```
public class Activity
{
    static Long    counter = 0L;

    public Long    id;
    public String  type;
    public String  location;
    public double  distance;

    public List<Location> route = new ArrayList<>();

    //...
}
```

```
public class Location
{
    static Long    counter = 0L;

    public Long    id;
    public float   latitude;
    public float   longitude;

    //...
}
```

```

public class User
{
    //...
    @Override
    public String toString()
    {
        return toStringHelper(this).addValue(id)
                                   .addValue(firstName)
                                   .addValue(lastName)
                                   .addValue(password)
                                   .addValue(email)
                                   .addValue(activities)
                                   .toString();
    }
    @Override
    public boolean equals(final Object obj)
    {
        if (obj instanceof User)
        {
            final User other = (User) obj;
            return Objects.equal(firstName, other.firstName)
                && Objects.equal(lastName, other.lastName)
                && Objects.equal(email, other.email)
                && Objects.equal(password, other.password)
                && Objects.equal(activities, other.activities);
        }
        else
        {
            return false;
        }
    }
    @Override
    public int hashCode()
    {
        return Objects.hashCode(this.id, this.lastName, this.firstName, this.email, this.password);
    }
}

```

pacemaker
model -
equals/toString/hashCode

pacemaker fixtures

```
public class Fixtures
{
    public static User[] users =
    {
        new User ("marge", "simpson", "marge@simpson.com", "secret"),
        new User ("lisa", "simpson", "lisa@simpson.com", "secret"),
        new User ("bart", "simpson", "bart@simpson.com", "secret"),
        new User ("maggie", "simpson", "maggie@simpson.com", "secret")
    };

    public static Activity[] activities =
    {
        new Activity ("walk", "fridge", 0.001),
        new Activity ("walk", "bar", 1.0),
        new Activity ("run", "work", 2.2),
        new Activity ("walk", "shop", 2.5),
        new Activity ("cycle", "school", 4.5)
    };

    public static Location[] locations =
    {
        new Location(23.3f, 33.3f),
        new Location(34.4f, 45.2f),
        new Location(25.3f, 34.3f),
        new Location(44.4f, 23.3f)
    };
}
```

```

public class UserTest
{
    User homer = new User ("homer", "simpson", "homer@simpson.com", "secret");

    @Test
    public void testCreate()
    {
        assertEquals ("homer",          homer.firstName);
        assertEquals ("simpson",        homer.lastName);
        assertEquals ("homer@simpson.com", homer.email);
        assertEquals ("secret",         homer.password);
    }

    @Test
    public void testIds()
    {
        Set<Long> ids = new HashSet<>();
        for (User user : users)
        {
            ids.add(user.id);
        }
        assertEquals (users.length, ids.size());
    }

    @Test
    public void testEquals()
    {
        User homer2 = new User ("homer", "simpson", "homer@simpson.com", "secret");
        User bart   = new User ("bart", "simpson", "bartr@simpson.com", "secret");

        assertEquals(homer, homer);
        assertEquals(homer, homer2);
        assertNotEquals(homer, bart);

        assertSame(homer, homer);
        assertNotSame(homer, homer2);
    }
    //...
}

```

UserTest (1)

UserTest (2)

```
public class UserTest
{
    User homer = new User ("homer", "simpson", "homer@simpson.com", "secret");

    //...

    @Test
    public void testToString()
    {
        assertEquals ("User{" + homer.id + ", homer, simpson, secret, homer@simpson.com, {}]", homer.toString());
    }
}
```


ActivityTest

```
public class ActivityTest
{
    Activity test = new Activity ("walk", "fridge", 0.001);

    @Test
    public void testCreate()
    {
        assertEquals ("walk", test.type);
        assertEquals ("fridge", test.location);
        assertEquals (0.0001, 0.001, test.distance);
    }

    @Test
    public void testToString()
    {
        assertEquals ("Activity{" + test.id + ", walk, fridge, 0.001, []}", test.toString());
    }
}
```

LocationTest

```
public class LocationTest
{
    @Test
    public void testCreate()
    {
        assertEquals (0.01, 23.3f, locations[0].latitude);
        assertEquals (0.01, 33.3f, locations[0].longitude);
    }

    @Test
    public void testIds()
    {
        assertNotEquals(locations[0].id, locations[1].id);
    }

    @Test
    public void testToString()
    {
        assertEquals ("Location{" + locations[0].id + ", 23.3, 33.3}", locations[0].toString());
    }
}
```

pacemaker api

PacemakerAPI (1)

- Implement the core features of the pacemaker service
- Not concerned with UI at this stage

```
public class PacemakerAPI
{
    private Map<Long, User>    userIndex      = new HashMap<>();
    private Map<String, User>  emailIndex    = new HashMap<>();
    private Map<Long, Activity> activitiesIndex = new HashMap<>();

    //...

    public Collection<User> getUsers ()
    {
        return userIndex.values();
    }

    public void deleteUser()
    {
        userIndex.clear();
        emailIndex.clear();
    }

    public void deleteUser(Long id)
    {
        User user = userIndex.remove(id);
        emailIndex.remove(user.email);
    }

    public Activity createActivity(Long id,          String type,
                                   String location, double distance)
    {
        Activity activity = null;
        Optional<User> user = Optional.fromNullable(userIndex.get(id));
        if (user.isPresent())
        {
            activity = new Activity (type, location, distance);
            user.get().activities.put(activity.id, activity);
            activitiesIndex.put(activity.id, activity);
        }
        return activity;
    }
}
```

PacemakerAPI (2)

```
public class PacemakerAPI
{
    private Map<Long, User>    userIndex      = new HashMap<>();
    private Map<String, User>  emailIndex    = new HashMap<>();
    private Map<Long, Activity> activitiesIndex = new HashMap<>();

    //...

    public Activity getActivity (Long id)
    {
        return activitiesIndex.get(id);
    }

    public void addLocation (Long id, float latitude, float longitude)
    {
        Optional<Activity> activity = Optional.fromNullable(activitiesIndex.get(id));
        if (activity.isPresent())
        {
            activity.get().route.add(new Location(latitude, longitude));
        }
    }
}
```

"Null sucks." - [Doug Lea](#)

"I call it my billion-dollar mistake." - [Sir C. A. R. Hoare](#), on his invention of the null reference

- Careless use of null can cause a staggering variety of bugs.
- In Google code base 95% of collections weren't supposed to have any null values in them
- This could should fail fast rather than silently accept null.
- Null is highly ambiguous, e.g., `Map.get(key)` can return null because
 - the value in the map is null,
 - or the value is not in the map.
- I.e. Null can mean failure, can mean success, can mean almost anything. Using something other than null makes your meaning clear.

Optionals in Guava

- `Optional<T>` is a way of replacing a nullable T reference with a non-null value.
- An `Optional` may either contain a non-null T reference (in which case we say the reference is "present"), or it may contain nothing (in which case we say the reference is "absent"). It is never said to "contain null."

```
Optional<Activity> activity = Optional.fromNullable(activitiesIndex.get(id));  
if (activity.isPresent())  
{  
    activity.get().route.add(new Location(latitude, longitude));  
}
```

- `activitiesindex.get(id)` will return null if id not present
- Wrap this in a 'Optional' wrapper object - noting that the it object it wraps may be null
- Use 'isPresent' to determine wrapped object is null or not

```
public class PacemakerAPITest
{
    private PacemakerAPI pacemaker;

    @Before
    public void setup()
    {
        pacemaker = new PacemakerAPI(null);
        for (User user : users)
        {
            pacemaker.createUser(user.firstName, user.lastName, user.email, user.password);
        }
    }

    @After
    public void tearDown()
    {
        pacemaker = null;
    }

    @Test
    public void testUser()
    {
        assertEquals (users.length, pacemaker.getUsers().size());
        pacemaker.createUser("homer", "simpson", "homer@simpson.com", "secret");
        assertEquals (users.length+1, pacemaker.getUsers().size());
        assertEquals (users[0], pacemaker.getUserByEmail(users[0].email));
    }

    @Test
    public void testUsers()
    {
        assertEquals (users.length, pacemaker.getUsers().size());
        for (User user: users)
        {
            User eachUser = pacemaker.getUserByEmail(user.email);
            assertEquals (user, eachUser);
            assertNotSame(user, eachUser);
        }
    }
}
```

PacemakerAPITest (1)

PacemakerAPITest (2)

```
@Test
public void testDeleteUsers()
{
    assertEquals (users.length, pacemaker.getUsers().size());
    User marge = pacemaker.getUserByEmail("marge@simpson.com");
    pacemaker.deleteUser(marge.id);
    assertEquals (users.length-1, pacemaker.getUsers().size());
}

@Test
public void testAddActivity()
{
    User marge = pacemaker.getUserByEmail("marge@simpson.com");
    Activity activity = pacemaker.createActivity(marge.id, activities[0].type, activities[0].location, activities[0].distance);
    Activity returnedActivity = pacemaker.getActivity(activity.id);
    assertEquals(activities[0], returnedActivity);
    assertNotSame(activities[0], returnedActivity);
}

@Test
public void testAddActivityWithSingleLocation()
{
    User marge = pacemaker.getUserByEmail("marge@simpson.com");
    Long activityId = pacemaker.createActivity(marge.id, activities[0].type, activities[0].location, activities[0].distance).id;

    pacemaker.addLocation(activityId, locations[0].latitude, locations[0].longitude);

    Activity activity = pacemaker.getActivity(activityId);
    assertEquals (1, activity.route.size());
    assertEquals(0.0001, locations[0].latitude, activity.route.get(0).latitude);
    assertEquals(0.0001, locations[0].longitude, activity.route.get(0).longitude);
}
```

PacemakerAPITest (3)

```
@Test
public void testAddActivityWithMultipleLocation()
{
    User marge = pacemaker.getUserByEmail("marge@simpson.com");
    Long activityId = pacemaker.createActivity(marge.id, activities[0].type, activities[0].location, activities[0].distance).id;

    for (Location location : locations)
    {
        pacemaker.addLocation(activityId, location.latitude, location.longitude);
    }

    Activity activity = pacemaker.getActivity(activityId);
    assertEquals(locations.length, activity.route.size());
    int i = 0;
    for (Location location : activity.route)
    {
        assertEquals(location, locations[i]);
        i++;
    }
}
}
```

pacemaker serializer

pacemaker persistence

```
public interface Serializer
{
    void push(Object o);
    Object pop();
    void write() throws Exception;
    void read() throws Exception;
}
```

```
public class PacemakerAPI
{
    private Map<Long, User>    userIndex      = new HashMap<>();
    private Map<String, User>  emailIndex    = new HashMap<>();
    private Map<Long, Activity> activitiesIndex = new HashMap<>();

    private Serializer serializer;

    public PacemakerAPI(Serializer serializer)
    {
        this.serializer = serializer;
    }

    @SuppressWarnings("unchecked")
    public void load() throws Exception
    {
        serializer.read();
        activitiesIndex = (Map<Long, Activity>) serializer.pop();
        emailIndex      = (Map<String, User>)   serializer.pop();
        userIndex       = (Map<Long, User>)     serializer.pop();
    }

    public void store() throws Exception
    {
        serializer.push(userIndex);
        serializer.push(emailIndex);
        serializer.push(activitiesIndex);
        serializer.write();
    }
}
```

serializer

```
public class XMLSerializer implements Serializer
{
    private Stack stack = new Stack();
    private File file;

    public XMLSerializer(File file)
    {
        this.file = file;
    }

    public void push(Object o)
    {
        stack.push(o);
    }

    public Object pop()
    {
        return stack.pop();
    }

    @SuppressWarnings("unchecked")
    public void read() throws Exception
    {
        ObjectInputStream is = null;

        try
        {
            XStream xstream = new XStream(new DomDriver());
            is = xstream.createObjectInputStream(new FileReader(file));
            stack = (Stack) is.readObject();
        }
        finally
        {
            if (is != null)
            {
                is.close();
            }
        }
    }
}
```

```
public void write() throws Exception
{
    ObjectOutputStream os = null;

    try
    {
        XStream xstream = new XStream(new DomDriver());
        os = xstream.createObjectOutputStream(new FileWriter(file));
        os.writeObject(stack);
    }
    finally
    {
        if (os != null)
        {
            os.close();
        }
    }
}
```

PersistenceTest - fixtures

```
public class PersistenceTest
{
    PacemakerAPI pacemaker;

    void populate (PacemakerAPI pacemaker)
    {
        for (User user : users)
        {
            pacemaker.createUser(user.firstName, user.lastName, user.email, user.password);
        }

        User user1 = pacemaker.getUserByEmail(users[0].email);
        Activity activity = pacemaker.createActivity(user1.id, activities[0].type, activities[0].location, activities[0].distance);
        pacemaker.createActivity(user1.id, activities[1].type, activities[1].location, activities[1].distance);
        User user2 = pacemaker.getUserByEmail(users[1].email);
        pacemaker.createActivity(user2.id, activities[2].type, activities[2].location, activities[2].distance);
        pacemaker.createActivity(user2.id, activities[3].type, activities[3].location, activities[3].distance);

        for (Location location : locations)
        {
            pacemaker.addLocation(activity.id, location.latitude, location.longitude);
        }
    }

    void deleteFile(String fileName)
    {
        File datastore = new File ("testdatastore.xml");
        if (datastore.exists())
        {
            datastore.delete();
        }
    }
}
```

```
public class Fixtures
{
    public static User[] users =
    {
        new User ("marge", "simpson", "marge@simpson.com", "secret"),
        new User ("lisa", "simpson", "lisa@simpson.com", "secret"),
        new User ("bart", "simpson", "bart@simpson.com", "secret"),
        new User ("maggie", "simpson", "maggie@simpson.com", "secret")
    };

    public static Activity[] activities =
    {
        new Activity ("walk", "fridge", 0.001),
        new Activity ("walk", "bar", 1.0),
        new Activity ("run", "work", 2.2),
        new Activity ("walk", "shop", 2.5),
        new Activity ("cycle", "school", 4.5)
    };

    public static Location[] locations =
    {
        new Location(23.3f, 33.3f),
        new Location(34.4f, 45.2f),
        new Location(25.3f, 34.3f),
        new Location(44.4f, 23.3f)
    };
}
```

Verify Fixture

```
@Test
public void testPopulate()
{
    pacemaker = new PacemakerAPI(null);
    assertEquals(0, pacemaker.getUsers().size());
    populate (pacemaker);

    assertEquals(users.length, pacemaker.getUsers().size());
    assertEquals(2, pacemaker.getUserByEmail(users[0].email).activities.size());
    assertEquals(2, pacemaker.getUserByEmail(users[1].email).activities.size());
    Long activityID = pacemaker.getUserByEmail(users[0].email).activities.keySet().iterator().next();
    assertEquals(locations.length, pacemaker.getActivity(activityID).route.size());
}
```

Serializer Test (XML)

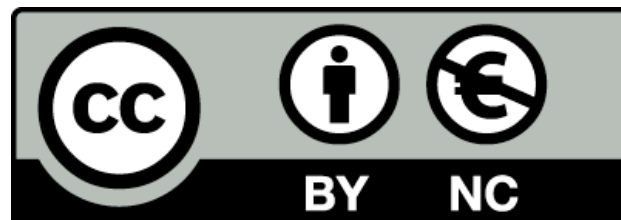
```
@Test
public void testXMLSerializer() throws Exception
{
    String datastoreFile = "testdatastore.xml";
    deleteFile (datastoreFile);

    Serializer serializer = new XMLSerializer(new File (datastoreFile));

    pacemaker = new PacemakerAPI(serializer);
    populate(pacemaker);
    pacemaker.store();

    PacemakerAPI pacemaker2 = new PacemakerAPI(serializer);
    pacemaker2.load();

    assertEquals (pacemaker.getUsers().size(), pacemaker2.getUsers().size());
    for (User user : pacemaker.getUsers())
    {
        assertTrue (pacemaker2.getUsers().contains(user));
    }
    deleteFile ("testdatastore.xml");
}
```

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

