TDD da un capo all'altro

Matteo Vaccari
vaccari@pobox.com
matteo.vaccari@xpeppers.com

Introduzione

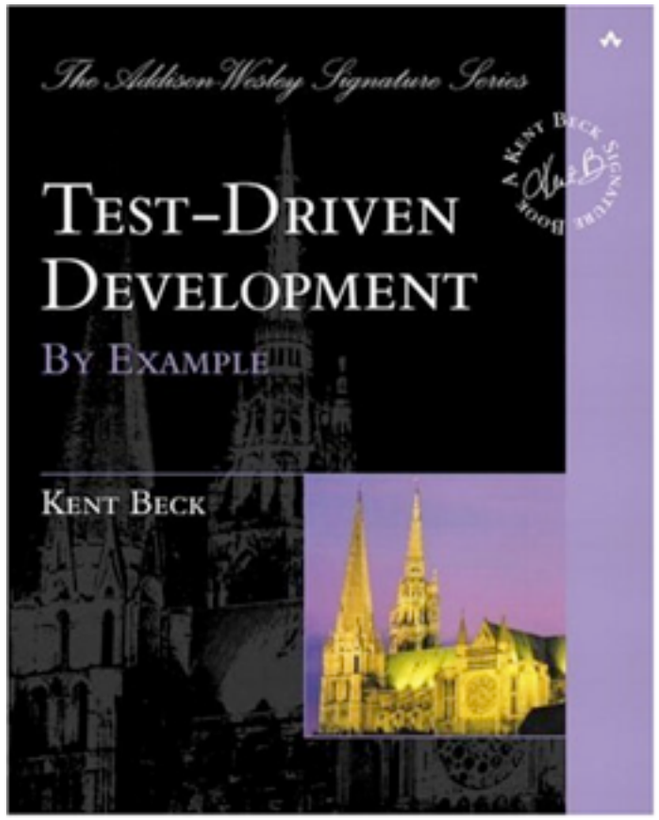
Quando si parla di Test-Driven Development spesso si sente dire “facciamo TDD sul dominio ma testiamo l'interfaccia utente a mano”. Oppure “vorrei fare TDD ma non so come applicarlo al database”. In questa presentazione vorrei dare qualche indicazione su come fare TDD su tutto il sistema, compresa l'interfaccia utente e il database.

L’obiettivo sono i benefici del TDD come strumento di design per tutto il sistema.

I miei maestri - Francesco Cirillo

Kent Beck ha già scritto un libro sul design a oggetti -- è il libro sul TDD

(Workshop Design Emergente 2009)



I miei maestri - Carlo Bottiglieri

Quando ho iniziato il TDD non mi avevano detto che serviva solo per il dominio -- così ho imparato a usarlo su tutta l'infrastruttura :-)

(Italian Agile Day 2010)

Tutto parte da Kent Beck

My Starter Test is often at a higher level, more like an application test. ...

The rest of the tests are "Assuming that we receive a string like this..."

Main points

- Outside In: inizia dagli input e scava
- One Step: ogni test ti insegna *una cosa* (e una sola)

Scriviamo un blog!

a. Mostra una lista di *blog post*
b. Riceve una lista di blog post dal database
c. Fa le *select* sul database
d. Produce HTML
e. Risponde a HTTP

e. Risponde a HTTP -- test

```
Blog blog = new Blog();
HttpUser user = new HttpUser();

@Test
public void answersToHttp() throws Exception {
    blog.start(8080);

    HttpResponse response = user.get("http://localhost:8080/");

    assertEquals(HttpResponse.OK, response.getStatus());
    blog.shutdown();
}
```

e. Risponde a HTTP -- impl

```
public class Blog {

    private Server server;      // org.mortbay.jetty.Server

    public void start(int port) {
        server = new Server(port);
        try {
            server.addHandler(new JettyAdapter());
            server.start();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private class JettyAdapter extends AbstractHandler {
        @Override
        public void handle(String target, HttpServletRequest request,
                           HttpServletResponse response, ... ) {
            response.setStatus(HttpServletResponse.SC_OK);
            response.getWriter().write("hello");
             ((Request)request).setHandled(true);
        }
    }
```

a. Mostra una lista di *blog post*

b. Riceve una lista di blog post dal database

c. Fa le *select* sul database

d. Produce HTML

e. Risponde a HTTP

a. Mostra una lista di *blog post*

b. Riceve una lista di blog post dal database

c. Fa le *select* sul database

d. Produce HTML

e. ~~Risponde a HTTP~~

f. Produce HTML -- test

```
@Test
public void respondsWithHtml() throws Exception {
   blog.addHandler("/", new Handler() {
      @Override
      public HttpResponse handle(HttpRequest request) {
         return new HttpResponse("<p>Hello</p>");
      }}
   );

   HttpResponse response = user.get("http://localhost:8080/");

   assertEquals(HttpResponse.OK, response.getStatus());
   assertEquals("<p>Hello</p>", response.getBody());
}
```

f. Produce HTML -- impl

```
public class Blog {

    private class JettyAdapter extends AbstractHandler {
        @Override
        public void handle(...) {
            response.setStatus(HttpServletResponse.SC_OK);
            if (null != handler) {
                HttpResponse httpResponse = handler.handle(new HttpRequest());
                response.getWriter().write(httpResponse.getBody());
            }
             ((Request)request).setHandled(true);
        }
    }

    private Server server;
    private Handler handler;
    // ....
```

a. Mostra una lista di *blog post*

b. Riceve una lista di blog post dal database

c. Fa le *select* sul database

d. Produce HTML

e. ~~Risponde a HTTP~~

a. Mostra una lista di *blog post*

b. Riceve una lista di blog post dal database

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

b. Riceve una lista di blog post dal database -- test

```
@Test
public void returnsPostsFromDatabase() throws Exception {
    final List<BlogPost> allPosts = asList(new BlogPost(), new BlogPost());
    BlogRepository repository = new BlogRepository() {
        public List<BlogPost> all() {
            return allPosts;
        }
    };
    BlogHandler handler = new BlogHandler(repository);
    HttpResponse response = handler.handle(null);
    String html = new BlogPage(allPosts).toHtml();
    HttpResponse expected = new HttpResponse(html);
    assertEquals(expected, response);
}
```

```
public class BlogPost {
}

public interface BlogRepository {
    List<BlogPost> all();
}
```

b. Riceve una lista di blog post dal database -- impl

```
public class BlogHandler implements Handler {

   private final BlogRepository repository;

   public BlogHandler(BlogRepository repository) {
      this.repository = repository;
   }

   @Override
   public HttpResponse handle(HttpRequest request) {
      List<BlogPost> posts = repository.all();
      String body = new BlogPage(posts).toHtml();
      HttpResponse response = new HttpResponse(body );
      return response;
   }

}
```

a. Mostra una lista di *blog post*

b. Mostra una lista di blog post dal database

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

a. Mostra una lista di *blog post*

b. ~~Mostra una lista di blog post dal database~~

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

a. Mostra una lista di *blog post* in html

b. ~~Mostra una lista di blog post dal database~~

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

a. Mostra una lista di *blog post* in html -- test

```
BlogPost firstPost = new BlogPost() {{
    put("title", "First p0st!");
    put("body", "first body");
}};

BlogPost secondPost = new BlogPost() {{
    put("title", "Second p0st!");
    put("body", "second body");
}};
```

```
@Test
public void showsAListOfBlogPosts() throws Exception {
    List<BlogPost> list = asList(firstPost, secondPost);
    BlogPage page = new BlogPage(list);
    String expected =
        "<div>" +
            new BlogPostEntry(firstPost).toHtml() +
            new BlogPostEntry(secondPost).toHtml() +
        "</div>";
    assertDomEquals(expected, page.toHtml());
}
```

La classe *BlogPost*

```
public class BlogPost extends HashMap<String, Object> {

}
```

assertDomEquals

```
public static void assertDomEquals(String message, String expected, String actual) {
    try {
        XMLUnit.setControlEntityResolver(ignoreDoctypesResolver());
        XMLUnit.setTestEntityResolver(ignoreDoctypesResolver());
        XMLUnit.setIgnoreWhitespace(true);
        XMLAssert.assertXMLEqual(message,
            ignoreEntities(expected), ignoreEntities(actual));
    } catch (Exception e) {
        fail(format("Malformed input: '%s'", actual));
    }
}
```

http://xmlunit.sourceforge.net/

a. Mostra una lista di *blog post* in html -- impl

```
public class BlogPage implements HtmlGenerator {
    private final List<BlogPost> posts;

    public BlogPage(List<BlogPost> posts) {
        this.posts = posts;
    }

    public String toHtml() {
        String result = "<div>";
        for (BlogPost post : posts) {
            result += new BlogPostEntry(post).toHtml();
        }
        result += "</div>";
        return result;
    }
}
```

```
public class BlogPostEntry implements HtmlGenerator {
    public String toHtml() {
        return "x";
    }
}
```

BlogPage > BlogPostEntry

```
@Test
public void showsABlogPostEntry() throws Exception {
  BlogPostEntry entry = new BlogPostEntry(firstPost);
  String expected =
    "<div>" +
      new BlogPostTitle(firstPost).toHtml() +
      new BlogPostBody(firstPost).toHtml() +
    "</div>";
  assertDomEquals(expected, entry.toHtml());
}
```

BlogPostEntry > BlogPostTitle, Body

```
public class BlogPostEntry implements HtmlGenerator {
   private final BlogPost post;

   public BlogPostEntry(BlogPost post) {
      this.post = post;
   }

   public String toHtml() {
      String result = "<div>"
         + new BlogPostTitle(post).toHtml()
         + new BlogPostBody(post).toHtml()
         + "</div>";
      return result;
   }
}
```

BlogPostEntry > BlogPostTitle, Body

```
@Test
public void showsPostTitleInHtml() throws Exception {
    BlogPostTitle title = new BlogPostTitle(firstPost);
    String expected = "<h2>First p0st!</h2>";
    assertDomEquals(expected, title.toHtml());
}

@Test
public void showsPostBodyInHtml() throws Exception {
    BlogPostBody body = new BlogPostBody(firstPost);
    assertDomEquals("<div>first body</div>", body.toHtml());
}
```

a. Mostra una lista di *blog post* in html

b. ~~Riceve una lista di blog post dal database~~

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

a. ~~Mostra una lista di *blog post* in html~~

b. ~~Riceve una lista di blog post dal database~~

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

c. Fa le *select* sul database -- test

```
Database database = new Database(
    "localhost", "blog_test", "blog_user", "password");

@Test
public void selectsOneRow() throws Exception {
    List<DatabaseRow> rows = database.select("select 2+2");
    assertEquals(1, rows.size());
    assertEquals(new Long(4), rows.get(0).getLong(0));
}

@Test
public void selectsMoreRows() throws Exception {
    List<DatabaseRow> rows = database.select("(select 2) union (select 3)");
    assertEquals(2, rows.size());
    assertEquals("2", rows.get(0).getString(0));
    assertEquals("3", rows.get(1).getString(0));
}
```

c. Fa le *select* sul database -- impl (che pizza)

```
public List<DatabaseRow> select(String sql) {
    List<DatabaseRow> result = new ArrayList<DatabaseRow>();

    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        statement = prepareStatement(sql);
        resultSet = statement.executeQuery();
        ResultSetMetaData metaData = resultSet.getMetaData();
        while (resultSet.next()) {
            DatabaseRow row = new DatabaseRow();
            for (int i = 0; i < metaData.getColumnCount(); i++) {
                row.put(metaData.getColumnName(i+1), resultSet.getObject(i+1));
            }
            result.add(row);
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally {
        close(resultSet);
        close(statement);
    }
    return result;
}
```

a. ~~Mostra una lista di *blog post*~~

b. ~~Riceve una lista di blog post dal database~~

c. Fa le *select* sul database

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

a. ~~Mostra una lista di *blog post*~~

b. ~~Riceve una lista di blog post dal database~~

c. ~~Fa le *select* sul database~~

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

a. ~~Mostra una lista di *blog post*~~

b. ~~Riceve una lista di blog post dal database~~

c. ~~Fa le *select* sul database~~

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

f. Traduce le righe in BlogPost

f. Traduce le righe in BlogPost -- test

```
Database database = new Database(
     "localhost", "blog_test", "blog_user", "password");

@Test
public void getsPostsFromDatabase() throws Exception {
    MysqlBlogRepository repository = new MysqlBlogRepository(database);

    List<BlogPost> all = repository.all();
    assertEquals(1, all.size());

    BlogPost actual = all.get(0);
    assertEquals("a title"), actual.get("title"));
    assertEquals("a body"), actual.get("body"));
}
```

f. Traduce le righe in BlogPost -- test setup

```
@Before
public void setUp() throws Exception {
   database.execute("delete from blog_posts");
   insertBlogPost("a title", "a body");
}

private void insertBlogPost(String title, String body) {
   String sql =
      "insert into blog_posts " +
      "(title, body) values (?, ?)";
   database.execute(sql, title, body);
}
```

f. Traduce le righe in BlogPost -- impl

```
public class MysqlBlogRepository implements BlogRepository {
   private final Database database;

   public MysqlBlogRepository(Database database) {
      this.database = database;
   }

   public List<BlogPost> all() {
      List<DatabaseRow> rows = database.select("select * from blog_posts");
      List<BlogPost> result = new ArrayList<BlogPost>();
      for (DatabaseRow row : rows) {
         BlogPost post = new BlogPost();
         post.putAll(row.getMap());
         result.add(post);
      }
      return result;
   }
}
```

a. ~~Mostra una lista di *blog post* in html~~

b. ~~Riceve una lista di blog post dal database~~

c. ~~Fa le *select* sul database~~

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

f. Traduce le righe in BlogPost

a. ~~Mostra una lista di *blog post* in html~~

b. ~~Riceve una lista di blog post dal database~~

c. ~~Fa le *select* sul database~~

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

f. ~~Traduce le righe in BlogPost~~

a. ~~Mostra una lista di *blog post* in html~~

b. ~~Riceve una lista di blog post dal database~~

c. ~~Fa le *select* sul database~~

d. ~~Produce HTML~~

e. ~~Risponde a HTTP~~

f. ~~Traduce le righe in BlogPost~~

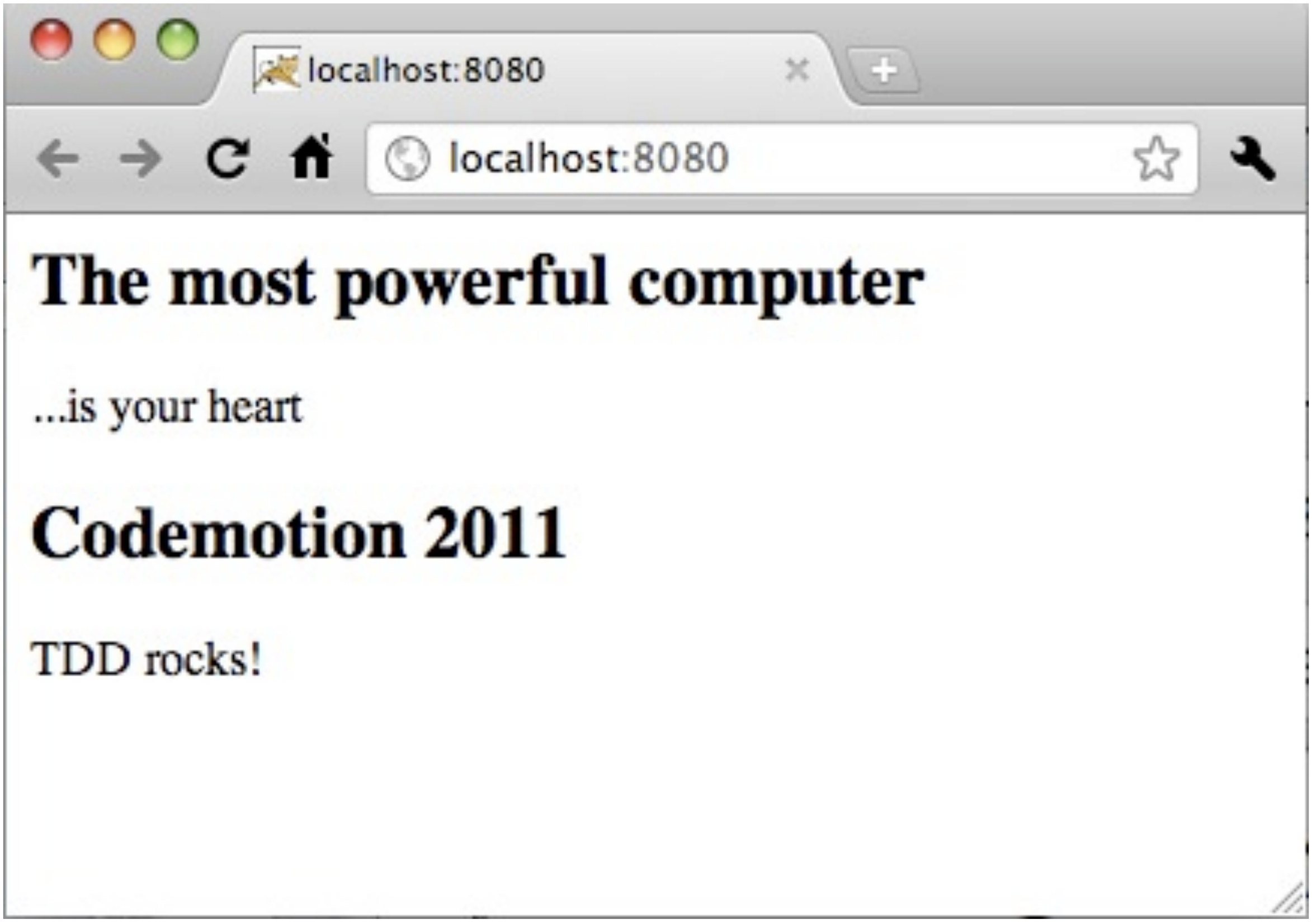
g. Test Drive

Insert test data

```
$ mysql -uroot blog_development

mysql> insert into blog_posts (title, body)
values ('The most powerful computer', '...is your heart');
Query OK, 1 row affected (0.26 sec)

mysql> insert into blog_posts (title, body)
values ('Codemotion 2011', 'TDD rocks!');
Query OK, 1 row affected (0.00 sec)

mysql> quit
Bye
$
```

Implement *main*

```
public static void main(String[] args) {
   Database database = new Database(
      "localhost", "blog_development", "blog_user", "password");
   BlogRepository repository = new MysqlBlogRepository(database);
   BlogHandler handler = new BlogHandler(repository);

   Blog blog = new Blog();
   blog.addHandler("/", handler);
   blog.start(8080);
}
```


localhost:8080
localhost:8080
The most powerful computer
...is your heart
Codemotion 2011
TDD rocks!

Riferimenti

- Kent Beck, *Test-Driven Development*, 2003
- Carlo Bottiglieri, *Web Apps in TDD*, Agile Day 2010
- Matteo Vaccari, *TDD per le viste*, Agile Day 2010
- Matteo Vaccari, *Programmazione web libera dai framework*, Webtech Italia 2010
- Miško Hevery, *Clean Code Talks*

Contattateci per stage
o assunzione

Peppers

Extreme Programming:
sviluppo e mentoring