

Verification & Validation  
Document

---

**LEARN STUFF SIMPLY ONLINE  
LESSON**

---

Andrea CASATI

Matricola n.  
765490

Daniele DONGHI

Matricola n.  
714338



# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
<b>2</b>	<b>Validazione</b>	<b>5</b>
<b>3</b>	<b>Verifica</b>	<b>9</b>
3.1	Analisi informale . . . . .	9
3.2	Testing . . . . .	10
3.3	Debugging . . . . .	13
<b>4</b>	<b>Conclusioni</b>	<b>16</b>

# Elenco delle figure

2.1	Pagina iniziale per effettuare registrazioni o login . . . . .	6
2.2	Pagina per accettare o rifiutare le richieste di iscrizione al corso . . . . .	7
2.3	Pagina di una lecture di un corso contenente i lecture content . . . . .	7
2.4	Pagina per la effettuare ricerche sui test result . . . . .	8
2.5	Pagina iniziale di un test, il tasto “take test” è presente in quanto la data in cui si tiene il test è la data corrente del sistema . . . . .	8
3.1	Metodo getAnswerQuestion Prima dell’analisi . . . . .	10
3.2	Metodo getAnswerQuestion Dopo l’analisi . . . . .	10
3.3	Screenshot dell’esecuzione del test Junit relativo alla registrazione di uno User . . . . .	11
3.4	Screenshot dell’esecuzione del test Junit relativo alla creazione di un Corso	12
3.5	Screenshot dell’esecuzione del test Junit relativo alla creazione di un Test	13
3.6	Metodo createAnswer Prima del debug . . . . .	14
3.7	Screenshot della fase di debug . . . . .	15
3.8	Metodo createAnswer Dopo il debug . . . . .	15

# Capitolo 1

## Introduzione

LEarn Stuff Simply ONline (LESSON) è un'applicazione software che permette di ottenere una piattaforma di apprendimento utilizzabile da professori, assistenti e studenti. L'obiettivo è quello di fornire la possibilità ai professori di poter gestire dei corsi a cui gli studenti possono iscriversi: questo comprende la gestione dei materiali dei corsi e dei test che gli studenti iscritti ad un corso possono effettuare dalla propria macchina. Attraverso un processo di autorizzazione degli studenti da parte del professore, essi saranno quindi in grado di seguire un intero corso senza spostarsi fisicamente da casa, ma scaricando tutto il necessario online.

Lo scopo di questo documento è la validazione e la verifica del sistema sviluppato e proposto.

# Capitolo 2

## Validazione

In questo capitolo verrà dimostrato come il sistema sviluppato e proposto rispetti tutti i requisiti imposti ed implementi tutte le funzionalità richieste.

L'accesso al sistema avviene attraverso la connessione con un browser alla pagina di home che racchiude la possibilità di registrarsi al sistema e di effettuare il login. Il sistema, per come è stato implementato permette anche la registrazione di uno stesso utente come professore, assistente e allievo, l'unico vincolo è che l'username sia differente. Una volta effettuata la registrazione ed il login sono possibili 3 scenari a seconda del tipo di utente.

Se l'utente è un assistente, la pagina personale permette di visualizzare i corsi in cui si è assistente e di visualizzare tutto ciò che il corso contiene (compresa la possibilità di correggere e valutare i test nel caso l'assistente fosse il checkere del test), oppure un assistente è in grado di effettuare ricerche sui test result (funzionalità non richiesta, ma aggiunta in quanto utile).

Se l'utente accede come insegnante sarà in grado di visualizzare i propri corsi con tutto ciò che ne consegue, creare nuovi corsi ed effettuare ricerche inerenti ai test result. Una volta selezionato un corso da visualizzare il trainer può vedere le richieste di iscrizione al corso e quindi accettarle o rifiutarle, selezionare l'assistente per il corso, editare e visionare i test, le lezioni e i contenuti delle lezioni.

Se l'utente effettua il login come trainee, invece, è in grado di vedere i propri corsi (e di effettuare varie operazioni tra cui vedere i lecture content ed effettuare gli esami) o di richiedere l'iscrizione ad un corso.

Figura 2.1: Pagina iniziale per effettuare registrazioni o login

The screenshot shows a web browser window with the title 'LESSON: Home'. The page has a dark blue header with the text 'LESSON - LEarn Stuff Simply ONLINE' and a logo on the left. Below the header, there are two main sections: 'REGISTER' and 'LOGIN', both in green text. The 'REGISTER' section has a form with radio buttons for 'Trainer', 'Trainee', and 'Assistant' (all unselected). Below these are input fields for 'Username:', 'Password:', 'Name:', 'Surname:', and 'Email:', followed by a 'Register' button. The 'LOGIN' section has radio buttons for 'Trainer', 'Trainee', and 'Assistant' (with 'Assistant' selected). Below these are input fields for 'Username:' (containing 'caso') and 'Password:' (containing four dots), followed by a 'Login' button.

Alcuni esempi sono mostrati nelle screenshot.


Tutte le altre funzionalità richieste sono state implementate, come ad esempio la possibilità di rendere disponibili nelle lezioni dei corsi materiali di diverso genere: sia contenuti differenti, sia tipologie differenti che metodologie differenti. Infatti, è possibile salvare un materiale rendendo disponibili l'url o salvando il file sul server affinché sia disponibile per il download.

Infine è possibile creare test con (da 1 a N) domande di vario genere. Per ogni test è necessario scegliere uno ed un solo checker che può essere il trainer del corso, l'assistente del corso o un software che corregge le risposte in modo automatico.

Il checker software è un modulo Junit che corregge sempre e solo risposte che sono necessariamente implementazioni Java. Ogni test con il checker software deve avere solo una domanda e tale domanda deve implicare una risposta contenente un'implementazione compilabile ed eseguibile.

Altre sono le funzionalità ma non è opportuno continuare ad elencarne altre. Se necessario è possibile provare il sistema per avere un'idea di come sono state implementate altre funzionalità: ad esempio, si può provare le varie versioni del retrieve test result.

Figura 2.2: Pagina per accettare o rifiutare le richieste di iscrizione al corso



# LESSON

**Enrolling requests of course data base**

andrea casati ☒

daniele donghi ☐

Accept ☒ Refuse ☐

[Confirm](#)

[Back to Course](#)

Figura 2.3: Pagina di una lecture di un corso contenente i lecture content



# LESSON

**Page of Lecture: lezione2**

**List of lecture content:**

Creation Date	Name	Type	Resource
14/1/2011	<a href="#">introduzioneSQL</a>	web site	<a href="http://en.wikipedia.org/wiki/SQL">http://en.wikipedia.org/wiki/SQL</a>
14/1/2011	<a href="#">SQL Slides.ppt</a>	slide SQL	<a href="#">SQL Slides.ppt</a>

[Back to course](#)



Figura 2.4: Pagina per la effettuare ricerche sui test result

The screenshot shows a web application titled "Test Results". It features a search interface with six distinct sections, each for a different role: "Retrieve tests by trainee", "Retrieve tests by course", "Retrieve tests by test mark", "Retrieve tests by trainer checker", "Retrieve tests by assistant checker", and "Retrieve tests by software checker". Each section contains a dropdown menu labeled "Choose one option" and a "Search" button. At the bottom of the page, there is a "Back to Personal Page" button.

Figura 2.5: Pagina iniziale di un test, il tasto “take test” è presente in quanto la data in cui si tiene il test è la data corrente del sistema

The screenshot shows a web application titled "Tests". It displays details for "Test 1 of course: data base". The details are as follows:

- Test details:
- Creation date: 14/1/2011
- Test day: 14/1/2011
- Verbalization date: 14/2/2011
- Checker type: Trainer
- Checker ID: 2
- Checker name: stefano
- Checker surname: ceri
- Checker Email: ceri@mail.com

At the bottom, there is a section titled "Take the test:" with a "Take test" button.

# Capitolo 3

## Verifica

In questo capitolo verrà descritta la fase di verifica del sistema. La prima sezione presenta l'analisi informale svolta, le seconda contiene esempi di debug svolti, mentre la terza sezione è dedicata alla fase di testing svolta principalmente attraverso l'utilizzo di Junit.

### 3.1 Analisi informale

L'analisi è lo studio analitico del codice sorgente che ha l'obiettivo di dimostrare che il prodotto non contiene errori. Due sono le tipologie di analisi: quella formale e quella informale. In questo contesto ci si è accontentati di svolgere quest'ultima.

Per eseguire l'analisi informale del codice prodotto ci si è scambiati i metodi implementati tra i diversi componenti del gruppo di progetto al fine di simulare manualmente l'esecuzione dei moduli sviluppati dagli altri per capire se il codice contenesse errori oppure no. E' stato importante l'essersi scambiati i moduli, in quanto un programmatore difficilmente riesce a trovare errori poco visibili e tal volta nascosti nel proprio codice.

Un esempio di utilizzo di questa tecnica è l'analisi del metodo: “getAnswerQuestion” che preso in ingresso l'id di una risposta, produce in uscita l'id della domanda corrispondente.

Il metodo presentato nella prima immagine contiene un errore: nella query in EJBQL invece di “Answer” è stato scritto “Anwer”. Questo errore è particolarmente difficile da vedere, soprattutto allo sviluppatore del metodo, in quanto si tratta di un errore di

Figura 3.1: Metodo getAnswerQuestion Prima dell'analisi

```
@Override
public int getAnswerQuestion(int answerId) throws IllegalArgumentException {
    if (answerId <= 0) {
        throw new IllegalArgumentException("Answer Id not valid");
    }

    try{

        Query q = this.em
            .createQuery("SELECT a.question.qid FROM Answer AS a WHERE a.aid = :n");
        q.setParameter("n", answerId);

        return (int) (Integer)q.getSingleResult();

    } catch (IllegalArgumentException e){
        throw new IllegalArgumentException("Error in retrieving the question " +
            "of the answer: " +e.getMessage());
    }
}
```

Figura 3.2: Metodo getAnswerQuestion Dopo l'analisi

```
@Override
public int getAnswerQuestion(int answerId) throws IllegalArgumentException {
    if (answerId <= 0) {
        throw new IllegalArgumentException("Answer Id not valid");
    }

    try{

        Query q = this.em
            .createQuery("SELECT a.question.qid FROM Answer AS a WHERE a.aid = :n");
        q.setParameter("n", answerId);

        return (int) (Integer)q.getSingleResult();

    } catch (IllegalArgumentException e){
        throw new IllegalArgumentException("Error in retrieving the question " +
            "of the answer: " +e.getMessage());
    }
}
```

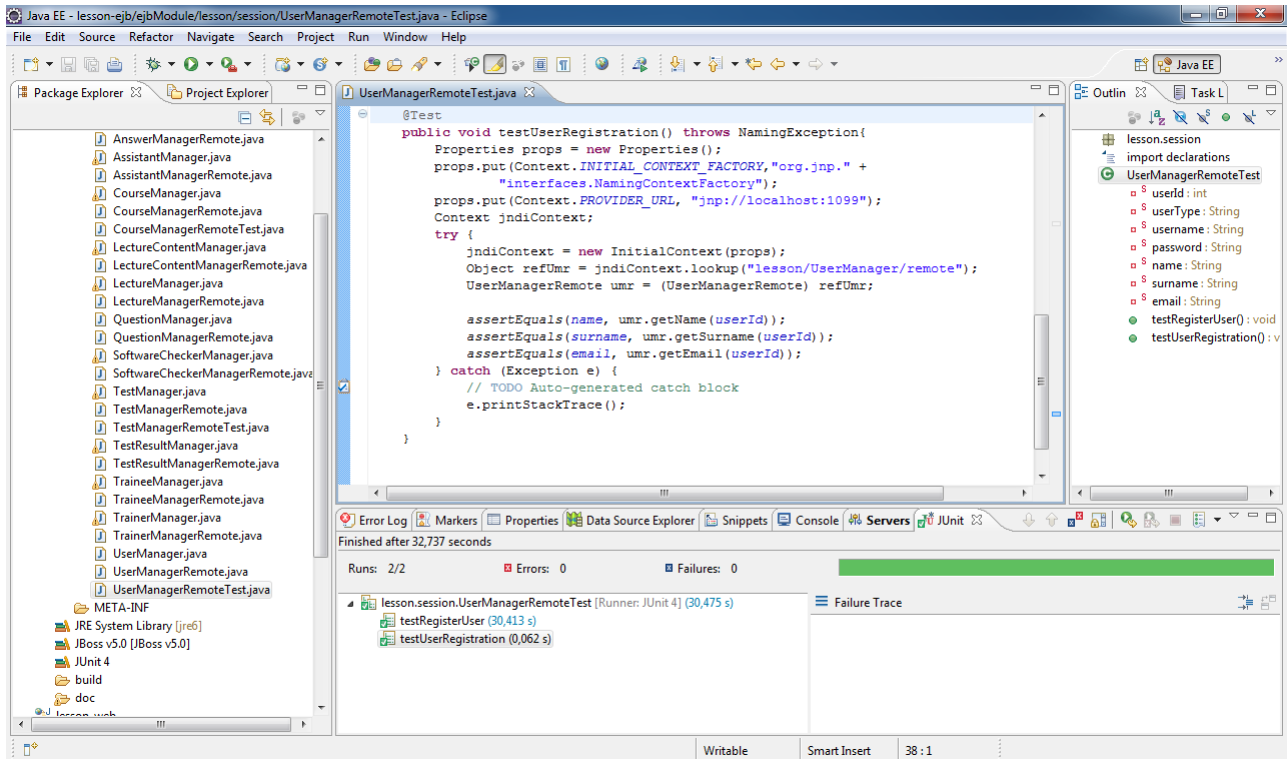
battitura in una query che non viene controllata dal compilatore java, ma viene trattata direttamente come stringa. In casi simili è molto utile un'azione di analisi informale.

Questo tipo di attività è stata utilizzata per l'analisi fino a quando non si è stati in grado di aver un minimo di client per poter testare e debuggare i moduli di basso livello. Ovviamente un altro tipo di tecnica utilizzata è stata quella di testing, attraverso l'utilizzo di Junit.

## 3.2 Testing

Il testing è una metodologia di verifica differente dall'analisi, ma complementare. L'obiettivo di questa fase è quello di palesare più errori possibili contenuti nel codice sorgente cercando di eseguire il minor numero di casi di test. Per questo è importante,

Figura 3.3: Screenshot dell'esecuzione del test Junit relativo alla registrazione di uno User



nella fase di testing, andare a determinare i casi più significativi, suddividendo i possibili parametri di input in situazioni rappresentative: infatti, facendo ciò è possibile eseguire un caso di test per ogni situazione. Ovviamente particolare attenzione è da porre nei casi di confine tra le varie situazioni.

Per la tecnica di testing è stato utilizzato Junit. Attraverso questo strumento è stato possibile creare casi di test per alcuni metodi di basso livello (principalmente metodi contenuti nei session bean) problematici.

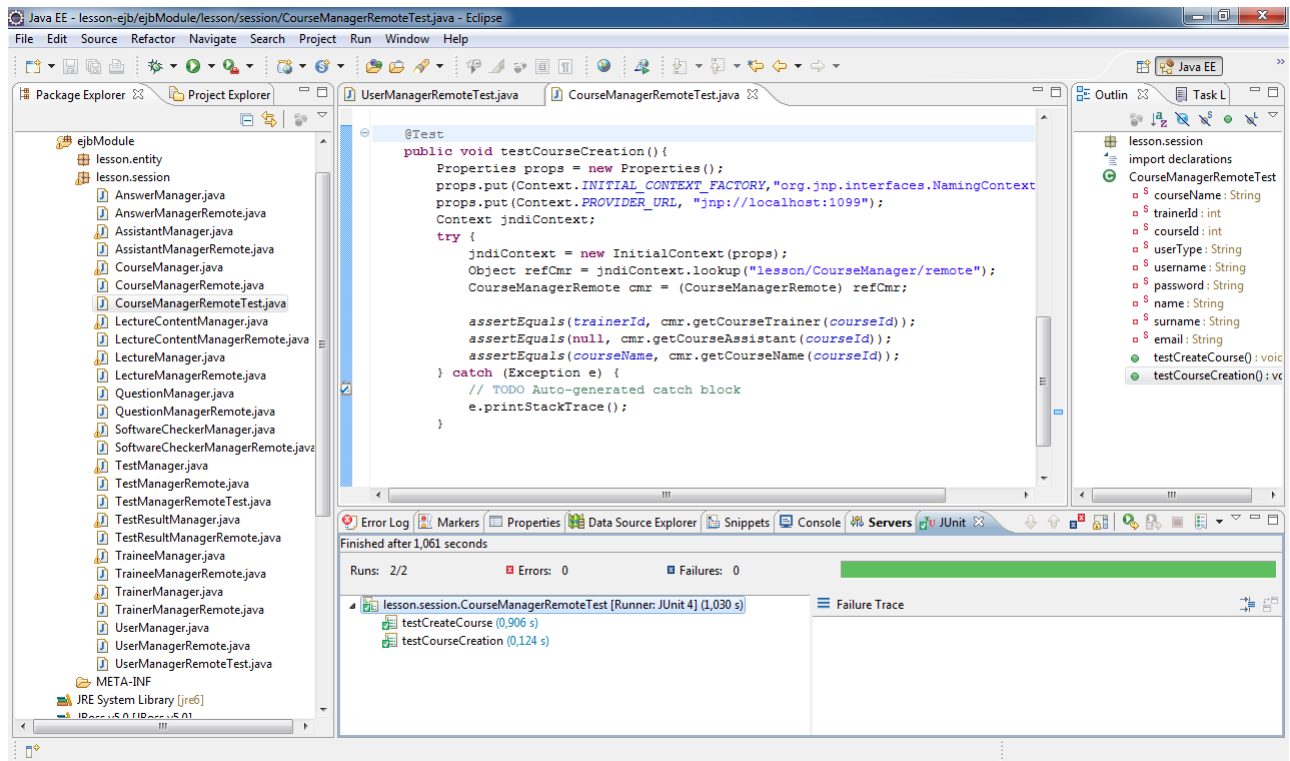
Di seguito ne vengono proposti esempi.

All'inizio della fase d'implementazione dei Session Bean, il problema è capire se i metodi sviluppati sono corretti, in quanto il client non è ancora stato creato e per cui non è possibile eseguire debug su tali metodi. In queste circostanze risulta molto utile l'utilizzo di uno strumento di testing come Junit.

Il primo esempio proposto è relativo alla registrazione di un utente.

Come si può vedere è stato creato un metodo di test junit che registra in automatico un utente e che poi esegue i metodi getter per verificare che i campi inseriti nella base di dati siano corretti.

Figura 3.4: Screenshot dell'esecuzione del test Junit relativo alla creazione di un Corso

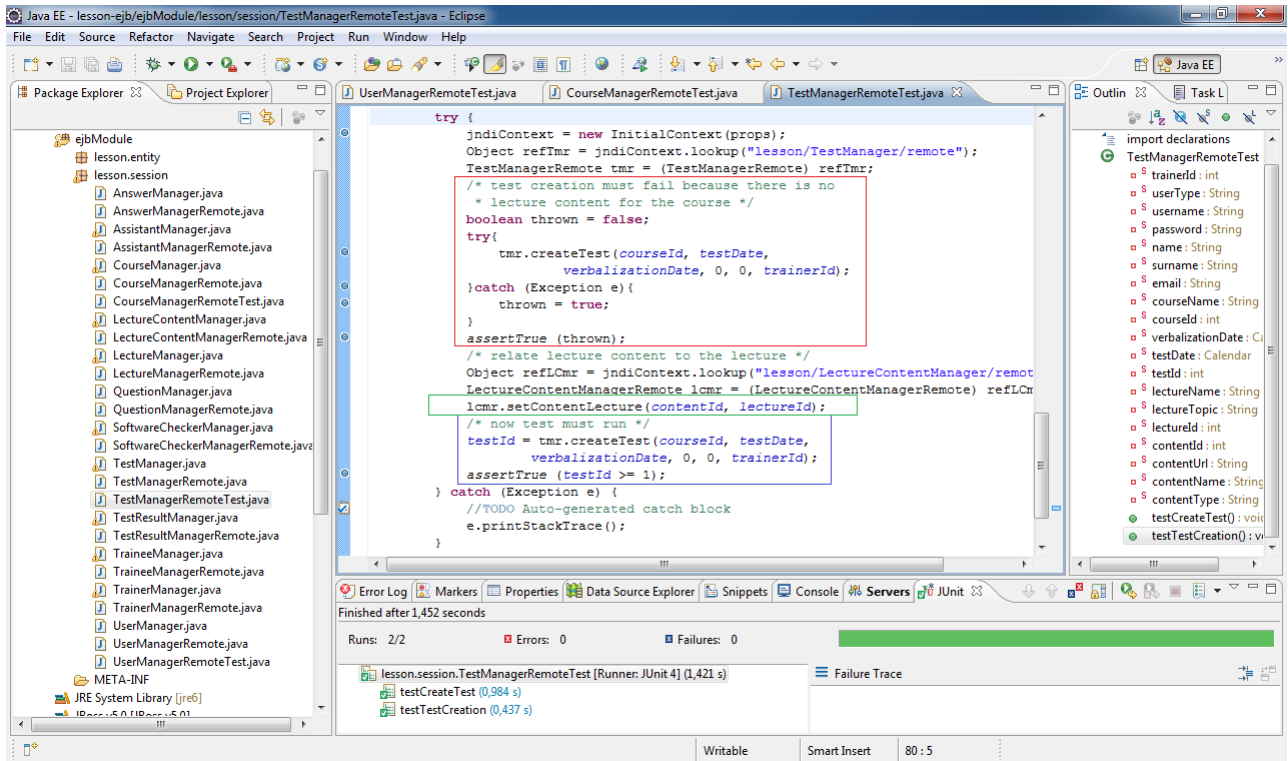


Nel secondo esempio proposto, il test viene eseguito sulla creazione di un corso. In questo caso viene prima registrato un Trainer, poi viene creato il corso ed infine vengono eseguiti i metodi getter per verificare la corretta creazione del corso stesso.

Nel terzo ed ultimo esempio si è testata la creazione di un Test. Tale testing è stato svolto per verificare che la creazione non venga permessa nel caso in cui il corso non abbia nemmeno un Lecture Content e che, invece, venga permessa se ne è almeno uno. Infatti, come mostrato dalle selezioni colorate nell'immagine, la prima assert verifica che venga sollevata un'eccezione (in quanto precedentemente sono stati creati il Trainer, il Corso, una Lecture, un LectureContent, ma il lecture content non è ancora stato associato alla lecture del corso), la seconda selezione evidenzia la creazione di una relazione tra il LectureContent e la Lecture del corso, mentre la terza selezione mostra che quando il corso ha almeno un Lecture Content allora il Test viene creato con successo.

Una volta implementato anche il web client, si è preferito continuare la fase di verifica e testing attraverso il metodo di debugging che viene presentato nella prossima sezione.

Figura 3.5: Screenshot dell'esecuzione del test JUnit relativo alla creazione di un Test



### 3.3 Debugging

Una volta cominciata la produzione del web client è stato possibile utilizzare la tecnica di debug per individuare e correggere gli errori che di volta in volta si presentavano durante l'esecuzione del client stesso. Per questa fase è stato utilizzato come strumento il debugger fornito in Eclipse.

La tecnica di debugging è come la tecnica di testing una metodologia di tipo sperimentale, tuttavia differisce da essa per l'obiettivo. Infatti, l'obiettivo del debug è di ridurre al minimo la distanza tra l'errore, il malfunzionamento e il fallimento, in modo da poter localizzare più facilmente l'errore e correggerlo. Infatti, non sempre è facile capire in quale parte del codice sia l'errore in quanto il momento in cui si presenta il fallimento potrebbe essere molto distante da quando l'errore è avvenuto.

Molti sono stati i casi in cui il debug è stato utile. Di seguito ne proporremo uno. In questo frangente, durante l'esecuzione, la creazione delle risposte falliva continuamente, ma non si riusciva a capire dove fosse l'errore. Attraverso il debug è stato possibile seguire passo passo l'esecuzione e i valori assunti dalle variabili, individuare l'errore nella riga della "getSingleResult" inerente a "q2" e correggerlo come mostrato nella

Figura 3.6: Metodo createAnswer Prima del debug

```
@Override
public int createAnswer(int questionId, int traineeId,
    byte[] answerFile, Calendar startTime) throws IllegalArgumentException {

    Calendar endTime = Calendar.getInstance();

    if (questionId <= 0) {
        throw new IllegalArgumentException("Question id not valid");
    }
    if (traineeId <= 0) {
        throw new IllegalArgumentException("Trainee Id not valid");
    }
    if (startTime.after(endTime)){
        throw new IllegalArgumentException("startTime can't be after endTime");
    }

    try{

        Query q = this.em
            .createQuery("FROM Question AS q WHERE q.qid = :n");
        q.setParameter("n", questionId);
        Question question = (Question) q.getSingleResult();

        Query q2 = this.em
            .createQuery("FROM Trainee AS t WHERE t.uid = :n");
        q2.setParameter("n", traineeId);
        Trainee trainee = (Trainee) q.getSingleResult();
```

terza immagine.

Figura 3.7: Screenshot della fase di debug

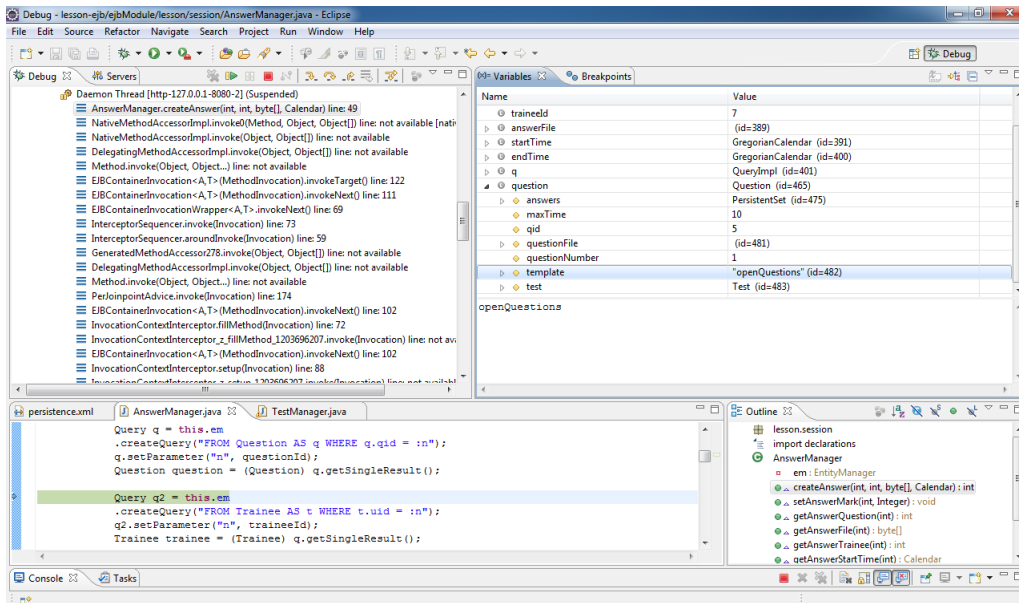


Figura 3.8: Metodo createAnswer Dopo il debug

```
@Override
public int createAnswer(int questionId, int traineeId,
    byte[] answerFile, Calendar startTime) throws IllegalArgumentException {

    Calendar endTime = Calendar.getInstance();

    if (questionId <= 0) {
        throw new IllegalArgumentException("Question id not valid");
    }
    if (traineeId <= 0) {
        throw new IllegalArgumentException("Trainee Id not valid");
    }
    if (startTime.after(endTime)) {
        throw new IllegalArgumentException("startTime can't be after endTime");
    }

    try{

        Query q = this.em
            .createQuery("FROM Question AS q WHERE q.qid = :n");
        q.setParameter("n", questionId);
        Question question = (Question) q.getSingleResult();

        Query q2 = this.em
            .createQuery("FROM Trainee AS t WHERE t.uid = :n");
        q2.setParameter("n", traineeId);
        Trainee trainee = (Trainee) q2.getSingleResult();
```



# Capitolo 4

## Conclusioni

Il sistema LEarn Stuff Simply ONline (LESSON) è stato sviluppato interamente da:

- Donghi Daniele (*d.donghi@gmail.com*)
- Andrea Casati (*casokaks@gmail.com*)

per il progetto di Ingegneria Del Software 2 della Prof.ssa Di Nitto, relativo al corso della laurea specialistica in Ingegneria Informatica, presso il Politecnico di Milano.

Il documento di verifica e validazione del progetto LEarn Stuff Simply ONline (LESSON) è stato scritto per il progetto di Ingegneria Del Software 2 della Prof.ssa Di Nitto, relativo al corso della laurea specialistica in Ingegneria Informatica, presso il Politecnico di Milano.

Gli autori sono:

- Donghi Daniele (*d.donghi@gmail.com*)
- Andrea Casati (*casokaks@gmail.com*)

Per la realizzazione del documento è stato utilizzato:

- L<sup>A</sup>T<sub>E</sub>X