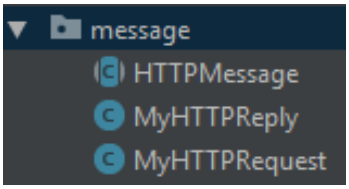
Simone Cipriani, **5951907**
Progetto per "**Reti di Calcolatori**"
Livelli implementati: **1**, **2**, **3**, **4**, **5**, **6**

Repository **git** disponibile su: *https://github.com/gesucca/networks-class-homework*

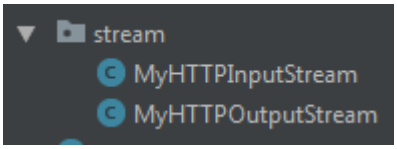
Descrizione del codice:

- Livello 0:



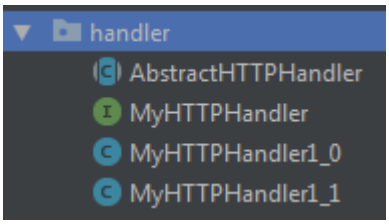
Le due classi *MyHTTPReply* e *MyHTTPRequest* implementano rispettivamente le interfacce *HTTPReply* e *HTTPRequest*, come da specifica. È stata creata una classe astratta *HTTPMessage* per fattorizzare parti comuni di codice e rendere consistente l'implementazione di servizi simili fra le due classi.

- Livello 1:



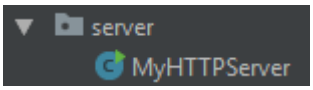
Come per il livello precedente, le due classi *MyHTTPInputStream* e *MyHTTPOutputStream* implementano le interfacce fornite come da specifica.

- Livelli 2, 3, 4, 5:



L'interfaccia data *HTTPHandler* è stata estesa nell'interfaccia *MyHTTPHandler*, in modo da aggiungere il metodo *setNextHandler(HTTPHandler h)* per implementare il design pattern Chain-Of-Responibility. Inoltre, è stata realizzata una classe astratta *AbstractHTTPHandler* per fattorizzare operazioni comuni a tutti gli handler concreti.
Si noti che le funzioni richieste nei livelli 2 e 3 (così come il 4 e il 5) sono state accorpate in una sola classe concreta tramite overload del costruttore e un'opportuna scrittura dei metodi chiave.

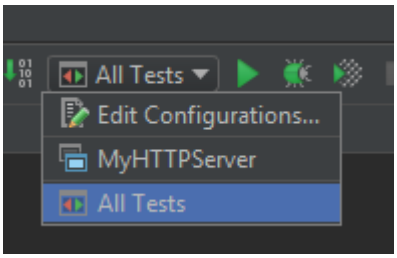
- Livello 6:

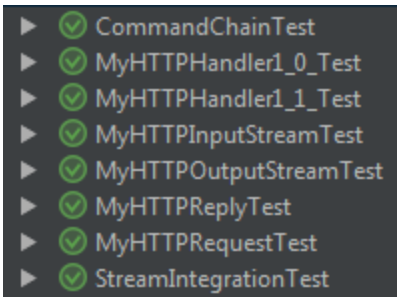


Per questo livello è stata semplicemente realizzata una classe concreta che implementa l'interfaccia data.

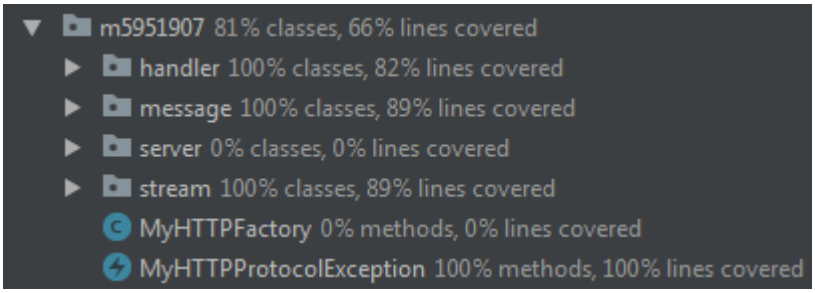
Per una descrizione dettagliata della struttura del codice si veda ./javadoc/index.html

Unit test:



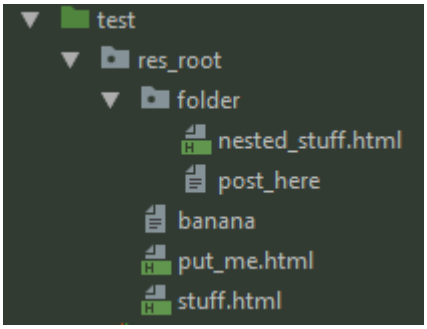


Sono stati realizzati 42 metodi di test unitari utilizzando il framework JUnit 4, al fine di garantire il funzionamento dei metodi chiave di ciascuna classe.



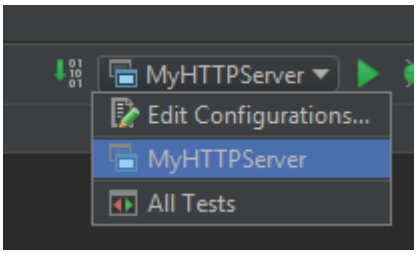
Si noti come, data la natura non critica dell'applicazione, non è stata ricercata una test coverage del 100%.

Per la classe MyHTTPServer non sono stati realizzati Unit Tests, ma solo Integration Test manuali tramite scripts Python.



Per testare l'implementazione dei metodi HTTP nelle classi handler, è stata realizzata una struttura di cartelle nello stesso percorso delle classi contenenti i metodi di test. Questo consente di rendere estremamente robusta la portabilità dei test fra varie macchine, al costo di sporcare il package dei sorgenti con file non correlati alla compilazione del bytecode Java.

Test manuali: server Java



Per lanciare il server Java realizzato è stato scritto un metodo main direttamente nella classe. Esso non fa altro che costruire un oggetto della classe *MyHTTPServer* e invocare il suo metodo *start*().

Ovviamente, per ragioni di semplicità, l'oggetto server istanziato in questo modo è impostato per ascoltare richieste dirette all'host locale, verso una porta libera.

```
server = new MyHTTPServer( port: 8080,  backlog: 10, InetAddress.getLocalHost());
```

Test manuali: client Python

```
host = socket.gethostname()
port = 8080
```

Sono stati realizzate delle funzioni Python che creano delle richeiste HTTP e che le inoltrano al server Java, contattandolo all'indirizzo dell'host locale e sulla stessa porta su cui esso ascolta.

Esempio 1: invio di una richiesta *GET* impossibile da soddisfare

```
get('/stuff_I_do_not_have.html')
```

```
--- RECEIVED:
 HTTP/1.0 404 Not Found
Date: Wed, 3 Jan 2018 10:17:45 +0100
Host: DE-SIMOCIP01
```

Esempio 2: invio di una richiesta *GET* lecita

```
get('/stuff.html')
```

```
--- RECEIVED:
 HTTP/1.0 200 OK
Date: Wed, 3 Jan 2018 10:42:59 +0100
Host: DE-SIMOCIP01
Last-Modified: Wed, 03 Jan 2018 09:00:08 CET
Handler-ID: DEFAULT ID

too lazy to type actual html
so here's bananas:
banana banana banana banana banana banana
banana banana banana banana banana banana
```