Diario di lavoro

Luogo	Canobbio
Data	05.02.2020

Lavori svolti

Oggi come prima cosa ho provato a creare un webserver per le Java Servlet manualmente, cosi da poterci inserire anche un collegamento seriale.
Per questo server ho preso spunto da: https://examples.javacodegeeks.com/enterprise-java/jetty/embedding-jetty-with-servlet/
Questo server comprende 2 servlet per test ed un server:
Codice **MainServlet**:

```
package webTest.servlets;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * Servlet test class.
 *
 * @author giuliobosco (giuliobva@gmail.com)
 * @version 1.0 (2020-02-05 - 2020-02-05)
 */
public class MainServlet extends HttpServlet {
    /**
     * Do get (just test).
     * Writes on CLI: Main servlet - Reached
     * Writes on http: Main Servlet - GET
     *
     * @param req Http request.
     * @param resp Http response.
     * @throws ServletException Servlet exception.
     * @throws IOException I/O Exception.
     */
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        // write on http
        resp.getOutputStream().println("Main Servlet - GET");
        // write on cli
        System.out.println("Main servlet - Reached");
    }
}
```

In questa servlet vi e` solo il metodo GET implementato. Il quale risponde via http con una stringa e stampa sulla CLI un'altra stringa. È anche molto simile la seconda servlet, l'unica differenza sta nel fatto, per cui risponde via http e scrive sulla CLI "Second" al posto di "Main".

Mentre il codice del webserver e` il seguente.

```
package webTest;


import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletContextHandler;
import org.eclipse.jetty.servlet.ServletHolder;
import webTest.servlets.MainServlet;
import webTest.servlets.SecondServlet;

/**
 * Base Test Web Server.
 *
 * @author giuliobosco (giuliobva@gmail.com)
 * @version 1.0 (2020-02-05 - 2020-02-05)
 */
public class App {
    /**
     * Start web server.
     */
    private void start() {
        try {
            // create server
            Server server = new Server(8080);

            // create servlet handler
            ServletContextHandler context = new
ServletContextHandler(ServletContextHandler.SESSIONS);

            // set server requests handler
            server.setHandler(context);

            // attach main servlet handler
            context.addServlet(new ServletHolder(new MainServlet()), "/");
            // attach second servlet handler
            context.addServlet(new ServletHolder(new SecondServlet()), "/second");

            // start server
            server.start();
        } catch (Exception e) {
            System.out.println("ERROR");
        }
    }

    /**
     * Execute class.
     *
     * @param args Command line arguments (not used).
     */
    public static void main(String[] args) {
        // start app
        new App().start();
    }
}
```

Il programma crea un webserver e ci collega le servlet.

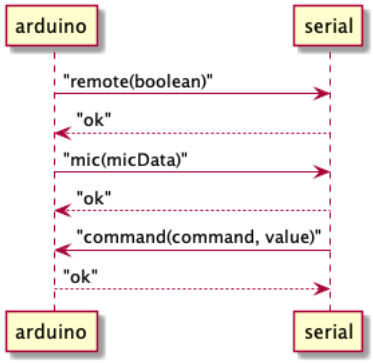
Le librerie le ho gestite con gradle, cosi che nel caso questo codice sia quello definitivo sia facile integrarlo con quello gia presente.

Per testare il codice che ho scritto, bisogna avviare il programma, con il seguente comando:

./gradlew run

Per poi eseguire tramite browser la richiesta **http://localhost:8080**, e poi **http://localhost:8080/second**, il primo link deve ritornare la stringa "**Main Servlet – GET**" mentre il secondo "**Second Servlet – GET**", mentre sulla CLI deve essere scritto "**Main Servlet – reached**" e rispettivamente "**Second Servlet – Reached**".

Dopo di che ho iniziato a pensare su come sviluppare il protocollo per la comunicazione seriale, che verrebbe utilizzata sia per la soluzione basata su Java, sia quella basata su python.
Ho fatto un diagramma di comunicazione fra i due componenti.



Quello mostrato nell'immagine è un ciclo di comunicazione, quando arriva alla fine viene rieseguito.

I comandi che possono essere inviati sono i seguenti:

Comando	Direzione	Tipo di dato	Dimensione	Note
null	←→	-	-	-
frequence	←	int	0-25000	cambio frequenza
genStatus	←	boolean	-	on/off generator
remote	→	-	-	remote changed status
mic	→	int	0-200	valore microfono
ok	→	-	-	-

Quando la seriale non sa che valore inviare invia nulll, che viene comunque con risposta **ok** e valore **null**, cosi che la comunicazione possa ricominciare.
Si potrebbe anche spostare la comparazione del valore del microfono dal lato arduino, per semplificare il codice dal lato seriale, ma poi bisognerebbe modificare il codice qualora si decidesse poi di usare la metodologia basata su python. Mentre si potrebbe fare facilmente nel caso dell'approccio dell'applicativo java.

Problemi riscontrati e soluzioni adottate
Inizialmente ho ripreso il programma dal diario della precedente giornata di lavoro. Ho provato ad implementare le classi ed importando le dipendenze che erano scritte, 2 delle classi usate nell'esempio erano dismesse e la libreria usata non più mantenuta, quindi ho deciso di cercare velocemente un altro esempio che ho trovato ed ho implementato sopra.

Punto della situazione rispetto alla pianificazione
Sono indietro rispetto alla pianificazione siccome sono stato male. Da un vantaggio di circa 6 ore sono passato ad un ritardo di 12 ore.

Programma di massima per la prossima giornata di lavoro
Andare dal docente Thomas Bartesaghi per chiedere se ha tempo per una consulenza.