

# **LEGO MINDSTORM EV3**

*STOP DREAMING, START DOING*

ONE DIRECTION  
TEDESCO, TONELLA, ELIA, VULPES, POGGI

# PROJECT

## COSA DOVEVAMO FARE

L'obiettivo principale del progetto è la realizzazione di un sistema di comunicazione tra un computer e un robot LEGO MINDSTORMS EV3, utilizzando il linguaggio Java e il protocollo TCP. Il sistema prevede due progetti distinti:

- uno installato sul robot,
- uno installato su un PC.

Tramite l'applicazione client, l'utente potrà inviare comandi per controllare il movimento del robot e ricevere informazioni sulla sua posizione e velocità. Il server, integrato con le API del robot, gestisce i motori e il sensore, rispondendo ai comandi ricevuti in tempo reale.

Questo progetto ha lo scopo di unire aspetti teorici e pratici, mettendo in campo competenze di programmazione, progettazione software e lavoro di squadra secondo la metodologia agile SCRUM.



### COORDINAZIONE

Il lavoro di gruppo è stato organizzato secondo la metodologia SCRUM, con riunioni regolari e una chiara suddivisione dei ruoli. La comunicazione costante ci ha permesso di collaborare in modo efficace e raggiungere l'obiettivo comune.



### NUOVE TECNOLOGIE

Abbiamo utilizzato il linguaggio Java per sviluppare sia il client che il server. La comunicazione tra i due moduli avviene tramite il protocollo TCP. Sul robot LEGO EV3 sono state impiegate le API Java per controllare motori e sensore.



### TEMPI

Abbiamo rispettato le scadenze previste, organizzando il lavoro in sprint e pianificando con cura ogni fase del progetto. Questo ci ha permesso di completare tutte le attività nei tempi stabiliti.



### ORGANIZZAZIONE

L'organizzazione del lavoro è avvenuta seguendo la metodologia SCRUM. Abbiamo utilizzato il Planning Poker per stimare l'impegno richiesto da ciascun compito e suddividere le attività in modo equo. Ogni membro del gruppo ha ricevuto responsabilità diverse, contribuendo attivamente allo sviluppo del progetto.

# WELCOME TO OUR ORGANIZATION

## CODICE



Il codice è diviso in due moduli: client e server.  
Il server gestisce motori, sensori e comunicazione TCP sul robot.  
Il client, su PC, invia comandi e riceve dati.  
La separazione ha reso il codice chiaro e ben organizzato.



## DIAGRAMMI

Abbiamo scelto di realizzare i diagrammi dopo lo sviluppo del codice, per riflettere con precisione l'architettura effettiva del progetto.  
In questo modo, i diagrammi UML risultano coerenti con le classi, le funzioni e le interazioni realmente implementate.



## DOCUMETAZIONE

Abbiamo pianificato l'intero progetto con una chiara suddivisione dei ruoli e dei compiti, seguendo la metodologia SCRUM. Le attività sono state distribuite in modo equilibrato tra i membri del gruppo e monitorate attraverso incontri regolari.



## GARA

Per prepararci alla gara, abbiamo svolto diverse prove sul tracciato, ottimizzando il comportamento del robot. Ogni membro si è occupato di un aspetto specifico, coordinandosi per garantire il miglior risultato possibile durante la competizione.



# OUR CODE CLIENT

## OBBIETTIVO

- L'obiettivo del client è quello di, dopo essersi connesso con il server, inviare comandi al robot Lego EV3 affinchè esso si possa muovere.
- Una volta inviati i comandi dovrà ricevere indietro ogni determinato tempo la velocità a cui si muove il robot.
- Finito di inviare i comandi per il movimento potrà, tramite un'altro comando, fermarlo.

## RISOLUZIONE

- Questo viene realizzato tramite la creazione di un'interfaccia grafica, che permette di collegarsi al server, e successivamente di guidarlo, tramite comandi WASD, e di fermarlo, comando X, e inviandoli al server, che li riceve ed esegue determinate funzioni.

```
private JTextField ipTextField;
private JTextField portTextField;
private JLabel speedLabel;
private Socket socket;
private PrintWriter outStream;
private BufferedReader inStream;

private JButton btnUp, btnDown, btnLeft, btnRight, btnStop;

public guidaClient() {
    setTitle("Telecomando EV3");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(450, 450);
    setResizable(false);
    setLayout(new BorderLayout(10, 10));
    setLocationRelativeTo(null);

    // Pannello superiore con IP, porta e bottoni
    JPanel topPanel = new JPanel(new GridLayout(2, 1, 5, 5));
    JPanel ipPortPanel = new JPanel();
    ipPortPanel.setLayout(new BoxLayout(ipPortPanel, BoxLayout.X_AXIS));
    ipPortPanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    ipPortPanel.add(new JLabel("IP Server:"));
    ipPortPanel.add(Box.createRigidArea(new Dimension(5, 0)));
    ipPortPanel.add(new JTextField("10.0.1.1", 15));
    ipPortPanel.add(ipTextField);
    ipPortPanel.add(Box.createRigidArea(new Dimension(15, 0)));
    ipPortPanel.add(new JLabel("Porta:"));
    ipPortPanel.add(Box.createRigidArea(new Dimension(5, 0)));
    ipPortPanel.add(new JTextField("1234", 6));
    ipPortPanel.add(portTextField);
    ipPortPanel.add(Box.createRigidArea(new Dimension(5, 0)));
    JButton connectButton = new JButton("Connetti");

    connectButton.addActionListener(e -> connectToServer());

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new GridLayout(2, 2));
    bottomPanel.add(btnUp);
    bottomPanel.add(btnDown);
    bottomPanel.add(btnLeft);
    bottomPanel.add(btnRight);
    bottomPanel.add(btnStop);
    bottomPanel.add(speedLabel);

    Thread t = new Thread(() -> {
        try (BufferedReader inStream = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter outStream = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(), true))) {
            String line;
            while ((line = inStream.readLine()) != null) {
                if (line.startsWith("VEL:")) {
                    String speed = line.split(":")[1].trim();
                    speedLabel.setText(speed);
                }
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Errore nella connessione al server.");
        }
    });

    t.start();

    JPanel centerPanel = new JPanel();
    centerPanel.setLayout(new GridLayout(1, 1));
    centerPanel.add(ipPortPanel);
    centerPanel.add(bottomPanel);
    centerPanel.add(t);
}
```

# OUR CODE CLIENT

```
11  public class guidaClient extends JFrame {  
12      private JTextField ipTextField;  
13      private JTextField portTextField;  
14      private JLabel speedLabel;  
15      private Socket socket;  
16      private PrintWriter outStream;  
17      private BufferedReader inStream;  
18  
19      private JButton btnUp, btnDown, btnLeft, btnRight, btnStop;  
20  
21  public guidaClient() {  
22      setTitle("Telecomando EV3");  
23      setDefaultCloseOperation(EXIT_ON_CLOSE);  
24      setSize(450, 450);  
25      setResizable(false);  
26     .setLayout(new BorderLayout(10, 10));  
27      setLocationRelativeTo(null);  
28  
29      // Pannello superiore con IP, porta e bottoni  
30      JPanel topPanel = new JPanel(new GridLayout(2, 1, 5, 5));  
31      JPanel ipPortPanel = new JPanel();  
32      ipPortPanel.setLayout(new BoxLayout(ipPortPanel, BoxLayout.X_AXIS));  
33      ipPortPanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));  
34  
35      ipPortPanel.add(new JLabel("IP Server:"));  
36      ipPortPanel.add(Box.createRigidArea(new Dimension(5, 0)));  
37  
38      ipTextField = new JTextField("10.0.1.1", 15);  
39      ipPortPanel.add(ipTextField);  
40      ipPortPanel.add(Box.createRigidArea(new Dimension(15, 0)));  
41      ipPortPanel.add(new JLabel("Porta:"));  
42      ipPortPanel.add(Box.createRigidArea(new Dimension(5, 0)));  
43  
44      portTextField = new JTextField("1234", 6);  
45      ipPortPanel.add(portTextField);  
46      ipPortPanel.add(Box.createRigidArea(new Dimension(15, 0)));  
47  
48      JButton connectButton = new JButton("Connetti");  
49      ipPortPanel.add(connectButton);  
50  
51      topPanel.add(ipPortPanel);  
52  
53      JPanel speedPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
54      speedPanel.add(new JLabel("Velocità: "));  
55      speedLabel = new JLabel("0");  
56      speedPanel.add(speedLabel);  
57      topPanel.add(speedPanel);  
58  
59      add(topPanel, BorderLayout.NORTH);  
60  
61      // Pannello controllo movimento  
62      JPanel controlPanel = new JPanel(new GridLayout(3, 3, 5, 5));  
63      btnUp = new JButton("▲ Avanti");  
64      btnDown = new JButton("▼ Indietro");  
65      btnLeft = new JButton("◀ Sinistra");  
66      btnRight = new JButton("▶ Destra");  
67      btnStop = new JButton("■ Stop");  
68
```

```
11  public class guidaClient extends JFrame {  
12  public guidaClient() {  
controlPanel.add(new JLabel());  
controlPanel.add(btnUp);  
controlPanel.add(new JLabel());  
controlPanel.add(btnLeft);  
controlPanel.add(btnStop);  
controlPanel.add(btnRight);  
controlPanel.add(new JLabel());  
controlPanel.add(btnDown);  
controlPanel.add(new JLabel());  
  
add(controlPanel, BorderLayout.CENTER);  
  
btnUp.addActionListener(e -> sendCommand("W"));  
btnDown.addActionListener(e -> sendCommand("S"));  
btnLeft.addActionListener(e -> sendCommand("A"));  
btnRight.addActionListener(e -> sendCommand("D"));  
btnStop.addActionListener(e -> sendCommand("X"));  
  
setupKeylistener();  
connectButton.addActionListener(e -> connectToServer());  
}  
  
private void connectToServer() {  
String ip = ipTextField.getText().trim();  
int port = Integer.parseInt(portTextField.getText().trim());  
  
try {  
socket = new Socket(ip, port);  
outStream = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(), true));  
inStream = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 JOptionPane.showMessageDialog(this, "Connesso a " + ip + ":" + port);  
System.out.println("Connesso al server!");  
  
// Thread per ricevere la velocità dal server e salvarla su file  
new Thread(() -> {  
try (BufferedWriter logWriter = new BufferedWriter(new FileWriter("C:\\Users\\Esame\\Documents\\GitHub\\OneDirection\\velocità.txt")) {  
String line;  
while ((line = inStream.readLine()) != null) {  
if (line.startsWith("VEL:")) {  
String speed = line.split(":")[1].trim();  
speedLabel.setText(speed);  
  
// Aggiungi timestamp  
String timestamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());  
String logEntry = timestamp + " - Velocità: " + speed;  
  
// Scrive su file  
logWriter.write(logEntry);  
logWriter.newLine();  
logWriter.flush();  
  
System.out.println("LOG: " + logEntry);  
}}  
} catch (IOException e) {  
e.printStackTrace();  
}}).start();  
}
```

```
} catch (IOException e) {  
e.printStackTrace();  
JOptionPane.showMessageDialog(this, "Errore nella connessione al server.");  
}  
}  
  
private void sendCommand(String command) {  
if (outStream != null) {  
outStream.println(command);  
}  
}  
  
private void setupKeyListener() {  
addKeyListener(new KeyAdapter() {  
@Override  
public void keyPressed(KeyEvent e) {  
if (e.getKeyCode() == KeyEvent.VK_W) sendCommand("W");  
if (e.getKeyCode() == KeyEvent.VK_S) sendCommand("S");  
if (e.getKeyCode() == KeyEvent.VK_A) sendCommand("A");  
if (e.getKeyCode() == KeyEvent.VK_D) sendCommand("D");  
if (e.getKeyCode() == KeyEvent.VK_X) sendCommand("X");  
if (e.getKeyCode() == KeyEvent.VK_1) sendCommand("1");  
if (e.getKeyCode() == KeyEvent.VK_2) sendCommand("2");  
if (e.getKeyCode() == KeyEvent.VK_3) sendCommand("3");  
}  
});  
setFocusable(true);  
}  
  
public static void main(String[] args) {  
SwingUtilities.invokeLater(() -> {  
guidaClient client = new guidaClient();  
client.setVisible(true);  
});  
}
```

# OUR CODE SERVER

## OBIETTIVO

- L'obiettivo del codice server è quello di gestire il robot LEGO EV3, ricevendo comandi dal modulo client tramite connessione TCP.
- Il server elabora questi comandi per controllare i motori e raccogliere dati dal sensore. Successivamente, invia le informazioni richieste al client.
- Questo permette al robot di muoversi in modo controllato e al sistema di mantenere una comunicazione costante e affidabile tra le due parti.

## RISOLUZIONE

- Abbiamo sviluppato una soluzione basata su socket TCP, che permette una comunicazione diretta tra client e server. Il server interpreta i comandi ricevuti e risponde con i dati del robot, garantendo un funzionamento stabile e reattivo.

```
        socket = new ServerSocket(PORT) {
            "EV3 in ascolto sulla porta " + PORT + "...");

            motori
            Motor motorA = new EV3LargeRegulatedMotor(MotorPort.A);
            Motor motorB = new EV3LargeRegulatedMotor(MotorPort.B);

            ma = 0;
            velocitaMassima;
            (velocitaMassima);
            eration(5000);
            leration(5000);

            {

                <et socket = serverSocket.accept();
                iferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                intWriter out = new PrintWriter(socket.getOutputStream(), true);

                System.out.println("Client connesso: " + socket.getInetAddress());

                // Copie finali per l'uso nel thread
                final EV3LargeRegulatedMotor mA = motorA;
                final EV3LargeRegulatedMotor mB = motorB;

                // Thread per inviare velocità al client ogni secondo
                Thread speedSender = new Thread(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            while (!socket.isClosed()) {
                                int speedA = mA.getSpeed();
                                int speedB = mB.getSpeed();
                                int media = (speedA + speedB) / 2;
                                out.println("VEL:" + media);
                                Thread.sleep(1000);
                            }
                        } catch (InterruptedException e) {
                            System.out.println("Errore invio velocità: " + e.getMessage());
                        }
                    }
                });
                speedSender.start();

                // Thread per ricevere comandi dal client
                Thread commandReceiver = new Thread(new Runnable() {
                    @Override
                    public void run() {
                        while(true) {
                            distanza.fetchSample(sample, 0);
                            float misura = sample[0];
                            if(misura < distanzaMin) {
                                Sound.beep();
                            }
                            try {
                                Thread.sleep(1000);
                            } catch(InterruptedException e) {
                                break;
                            }
                        }
                    }
                });
                commandReceiver.start();
            }
        };
    }
}
```

# OUR CODE SERVER

```
public static void main(String[] args) {
    int PORT = 1234;

    try (ServerSocket serverSocket = new ServerSocket(PORT)) {
        System.out.println("EV3 in ascolto sulla porta " + PORT + "...");

        // Inizializzazione motori
        EV3LargeRegulatedMotor motorA = new EV3LargeRegulatedMotor(MotorPort.A);
        EV3LargeRegulatedMotor motorB = new EV3LargeRegulatedMotor(MotorPort.B);

        int velocitaMassima = 0;
        motorA.setSpeed(velocitaMassima);
        motorB.setSpeed(velocitaMassima);
        motorA.setAcceleration(5000);
        motorB.setAcceleration(5000);

        while (true) {
            try {
                Socket socket = serverSocket.accept();
                BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                PrintWriter out = new PrintWriter(socket.getOutputStream(), true)
            } {
                System.out.println("Client connesso: " + socket.getInetAddress());

                // Copie finali per l'uso nel thread
                final EV3LargeRegulatedMotor mA = motorA;
                final EV3LargeRegulatedMotor mB = motorB;

                // Thread per inviare velocità al client ogni secondo
                Thread speedSender = new Thread(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            while (!socket.isClosed()) {
                                int speedA = mA.getSpeed();
                                int speedB = mB.getSpeed();
                                int media = (speedA + speedB) / 2;
                                out.println("VEL:" + media);
                                Thread.sleep(1000);
                            }
                        } catch (InterruptedException e) {
                            System.out.println("Errore invio velocità: " + e.getMessage());
                        }
                    }
                });
                speedSender.start();
            }
        }
    }
}
```

```
Thread bipThread = new Thread(new Runnable() {
    @Override

    public void run()
    {
        while(true)
        {
            distanza.fetchSample(sample, 0);
            float misura = sample[0];
            if(misura < distanzaMin)
            {
                Sound.beep();
            }
            try
            {
                Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
                break;
            }
        }
    }
});

speedSender.start();
bipThread.start();
```

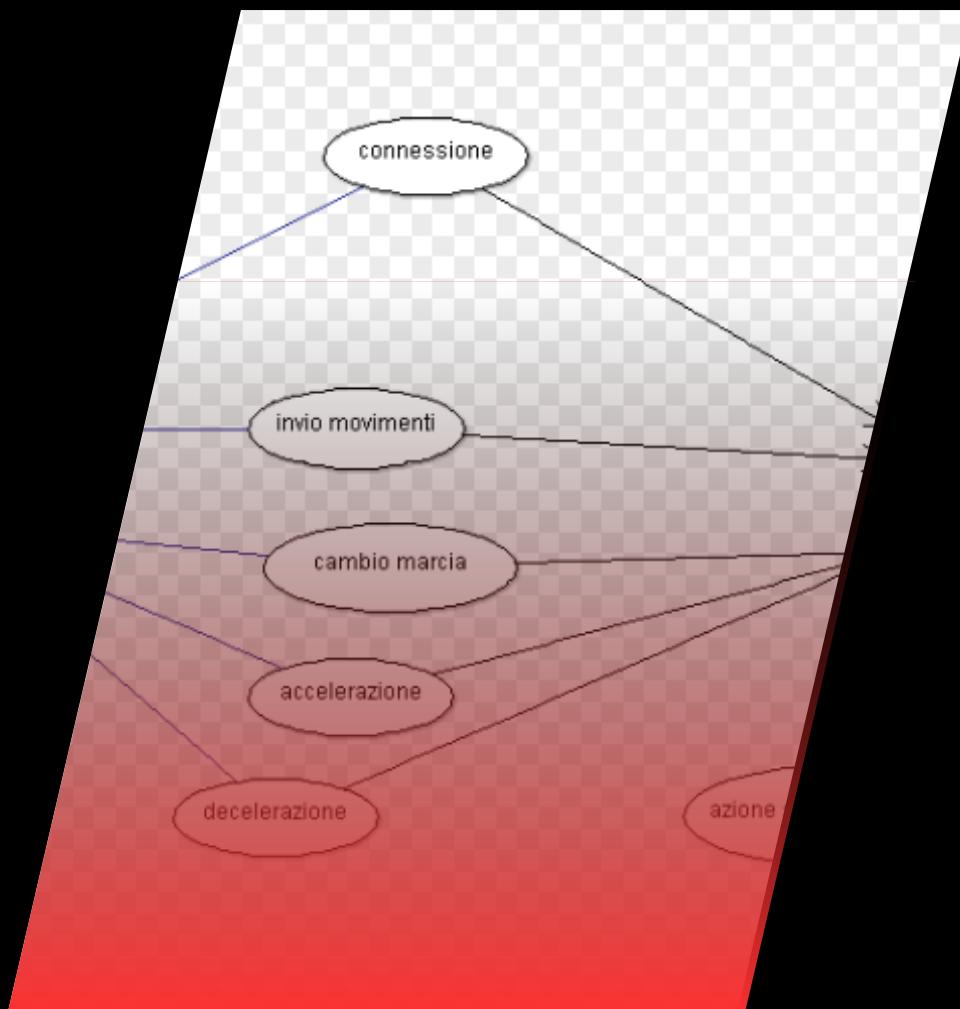
```
while ((command = in.readLine()) != null) {
    System.out.println("Ricevuto comando: " + command);

    switch (command) {
        case "W":
            resetMotorsSpeed(motorA, motorB, velocitaMassima);
            motorA.forward();
            motorB.forward();
            break;
        case "S":
            motorA.backward();
            motorB.backward();
            break;
        case "A":
            motorA.setSpeed((int) (velocitaMassima / 2));
            motorB.setSpeed(velocitaMassima);
            motorA.forward();
            motorB.forward();
            break;
        case "D":
            motorA.setSpeed(velocitaMassima);
            motorB.setSpeed((int) (velocitaMassima / 2));
            motorA.forward();
            motorB.forward();

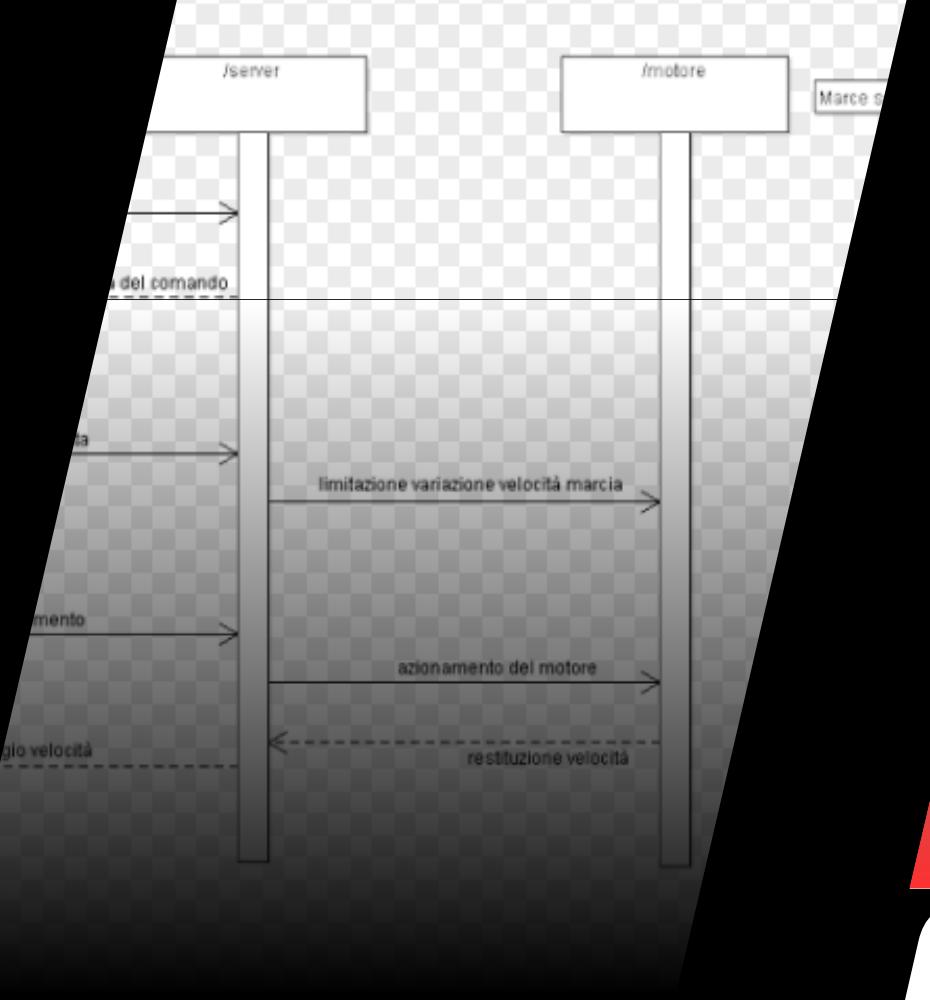
            break;
        case "I":
            velocitaMassima = 400;
            System.out.println("Marcia 1 impostata: velocità massima = " + velocitaMassima);
            break;
        case "2":
            velocitaMassima = 800;
            System.out.println("Marcia 2 impostata: velocità massima = " + velocitaMassima);
            break;
        case "3":
            velocitaMassima = 1100;
            System.out.println("Marcia 3 impostata: velocità massima = " + velocitaMassima);
            break;
        case "TURBO":
            velocitaMassima = (int) motorA.getMaxSpeed();
            System.out.println("Modalità TURBO attivata! Velocità massima = " + velocitaMassima);
            break;
        case "X":
            motorA.stop();
            motorB.stop();
            System.out.println("Motori arrestati.");
            break;
    }
}
```

# OUR DIAGRAM

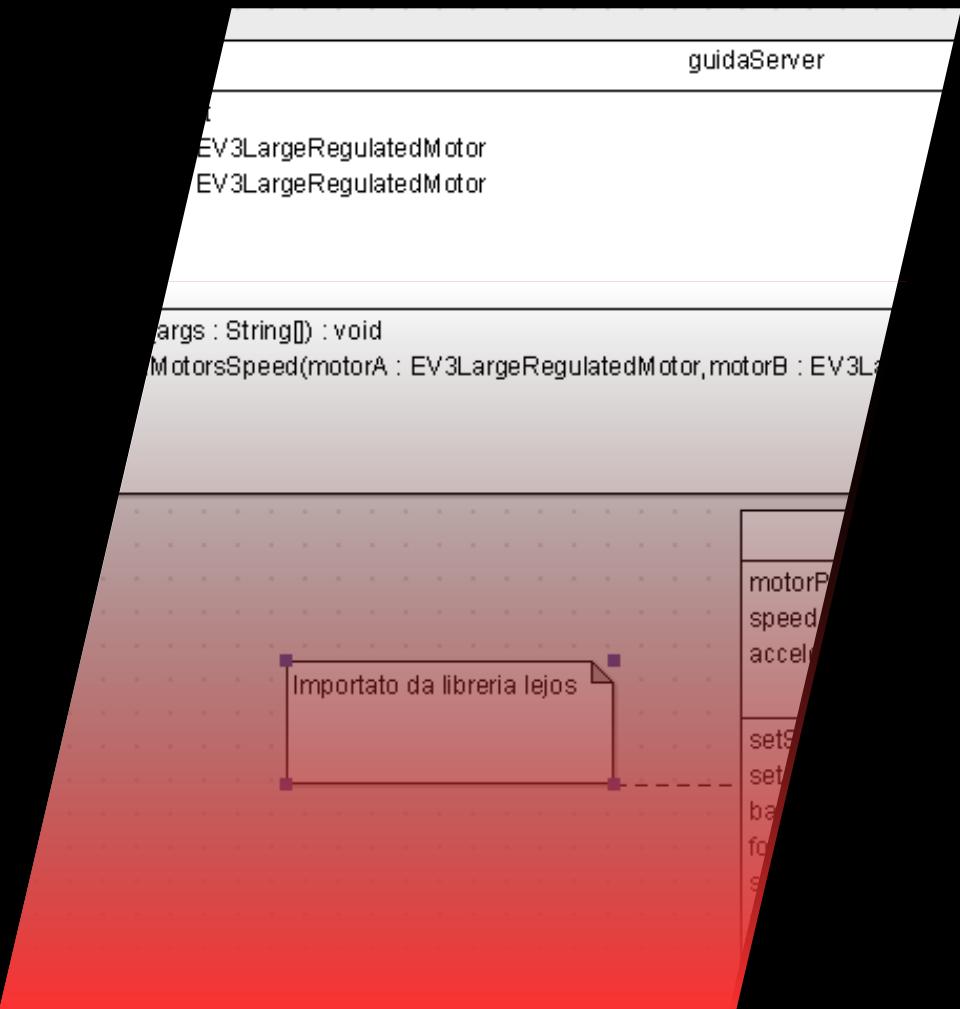
**01** USECASE



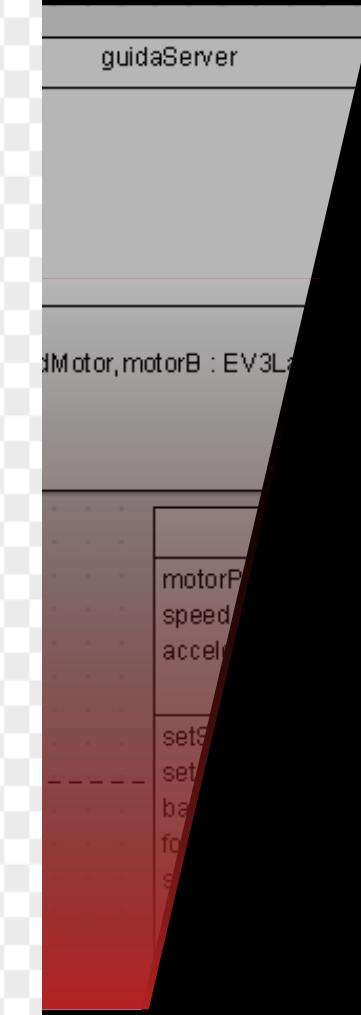
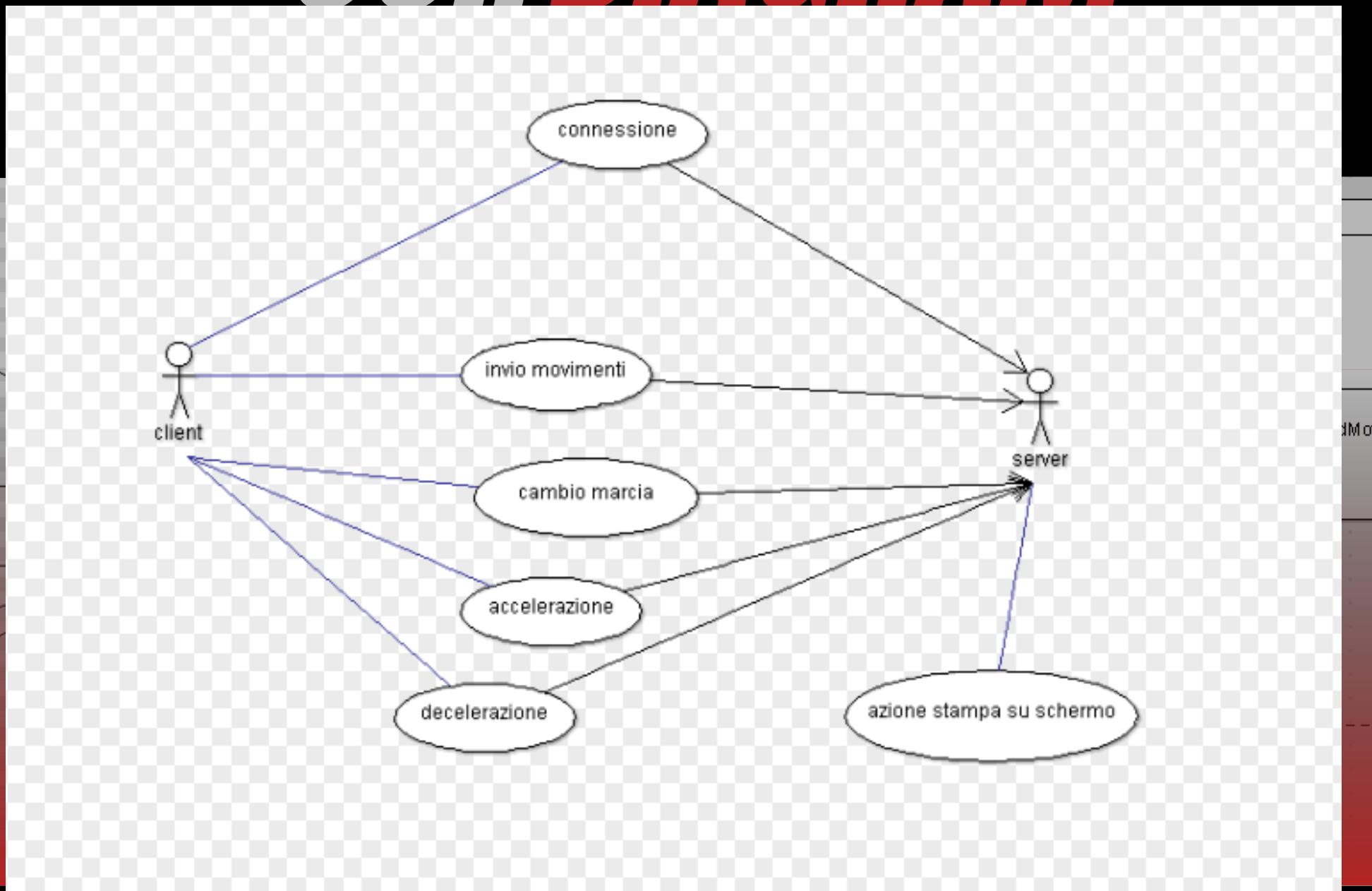
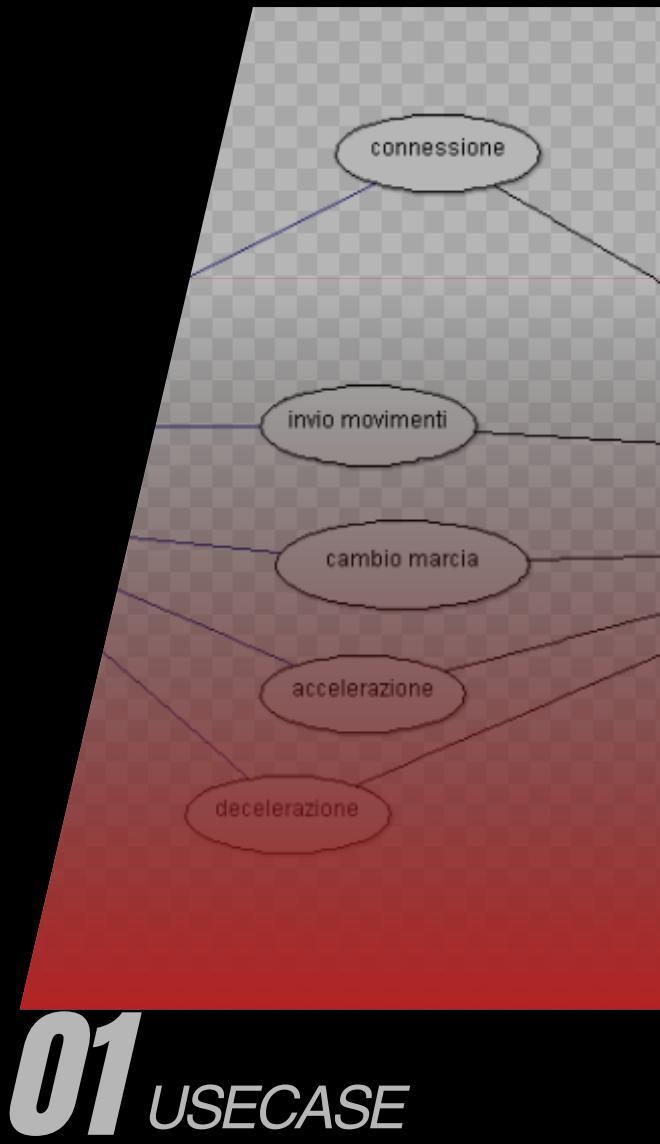
**02** SEQUENCE



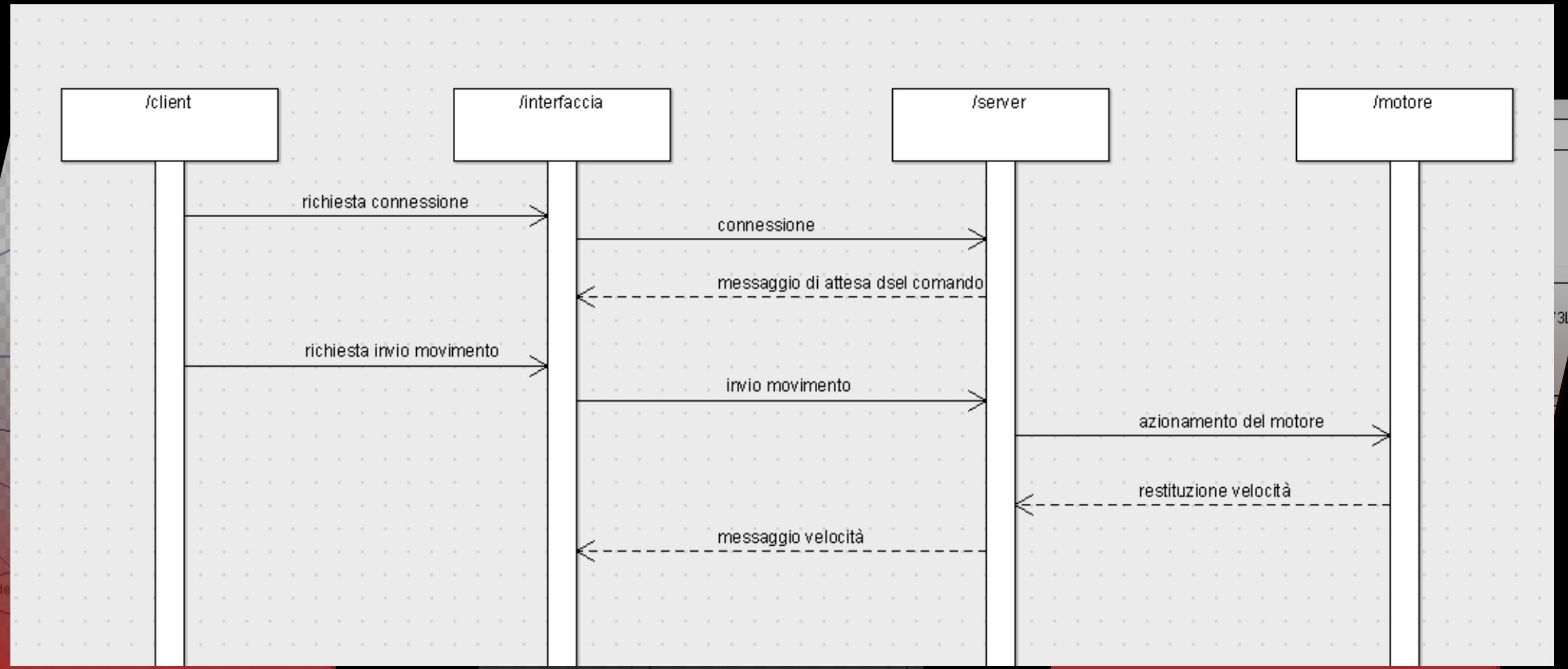
**03** CLASS



# OUR DIAGRAM



# OUR DIAGRAM

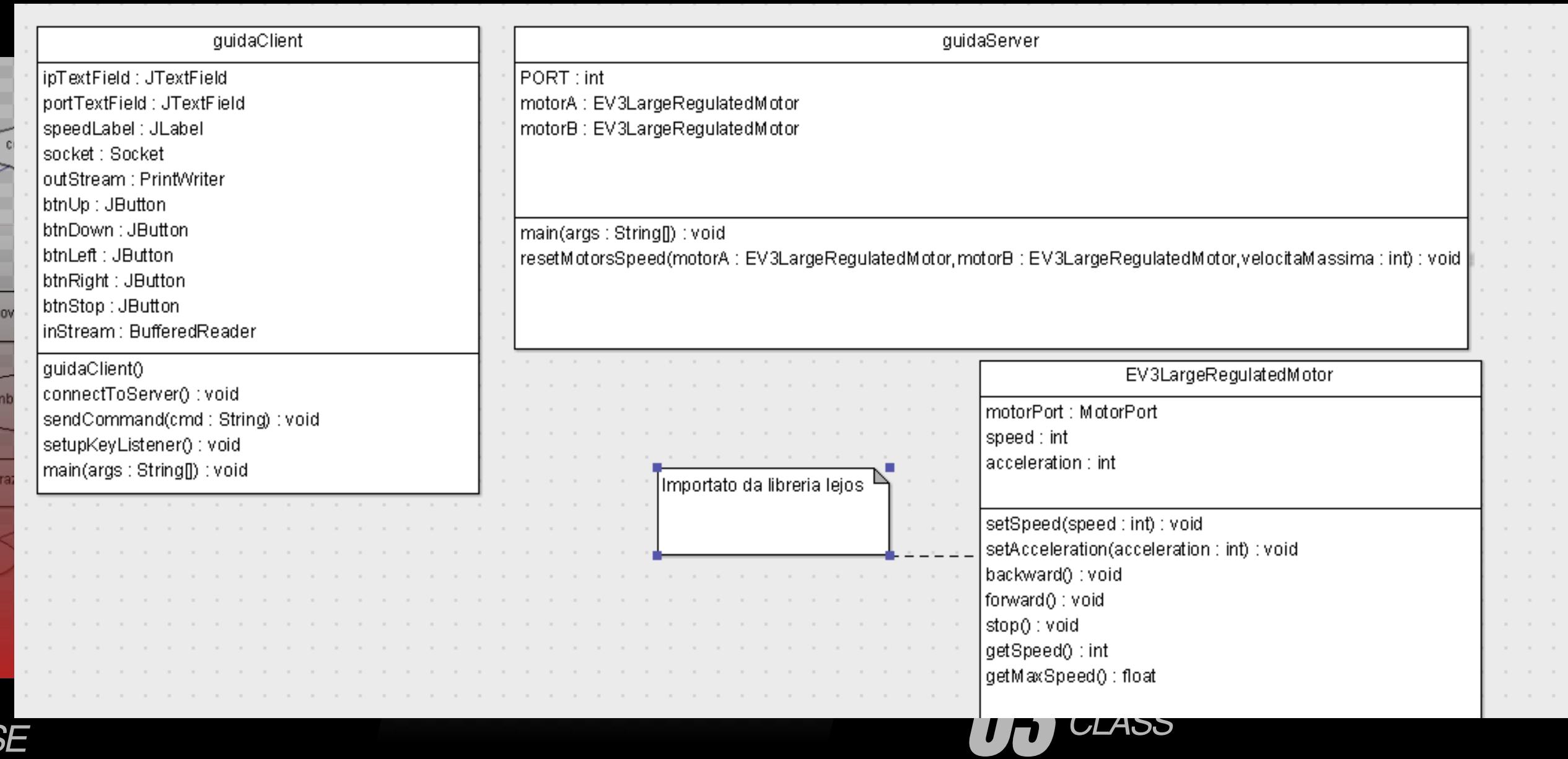


01 USECASE

03 CLASS

# OUR DIAGRAM

01 USECASE



02 CLASS

# **DOCUMENTATION TEST AND DOCUMENTS**

**01**

## **DOCUMENTAZIONE**

La documentazione è stata costruita in modo progressivo, annotando giorno per giorno le attività svolte e i punti emersi durante i daily scrum. Questo ci ha permesso di tenere traccia dell'avanzamento del progetto in modo chiaro e organizzato.

Al suo interno abbiamo descritto il funzionamento del Planning Poker, l'assegnazione dei compiti e la struttura della presentazione finale.

Inoltre, sono presenti collegamenti diretti alle sezioni secondarie, come la parte relativa ai test, garantendo una visione completa e ordinata di tutto il lavoro svolto.

**02**

## **TEST**

I test sono stati suddivisi in due fasi.

La prima ha verificato il funzionamento del codice e del robot, assicurandosi che la comunicazione TCP e i comandi fossero corretti.

La seconda fase ha riguardato il test su tracciato, per osservare il comportamento del robot in movimento e valutarne precisione e risposta ai comandi.



# THE RACE



È arrivato il momento della gara contro i nostri compagni di classe, un confronto diretto tra i veicoli LEGO realizzati da ciascun team.

Per prepararci al meglio, abbiamo effettuato una serie di test interni focalizzati sulla scelta del pilota. Ogni membro del gruppo ha provato il veicolo in condizioni simulate, con rilevazioni sul tempo, la precisione di guida e la gestione delle curve.

Alla fine, Tonella ha dimostrato la maggiore costanza e velocità, risultando la scelta più adatta per affrontare la competizione.

La gara si svolgerà in due fasi: una standard e una "truccata", con la possibilità di introdurre alcune modifiche al mezzo. Tuttavia, la nostra preparazione si è concentrata soprattutto sull'affinamento della guida e sull'affidabilità complessiva in pista.



# **OUR PROFESSIONAL TEAM**



**TOMMASO  
POGGI**

*SCRUM MASTER*



**DAVIDE  
VULPES**

*PROGRAMATORE*



**GABRIELE  
TONELLA**

*DESIGNER/TESTER*



**MATTEO  
ELIA**

*DOCUMENTATORE*



**FRANCESCO  
TEDESCO**

*DESIGNER*



**THANK YOU**