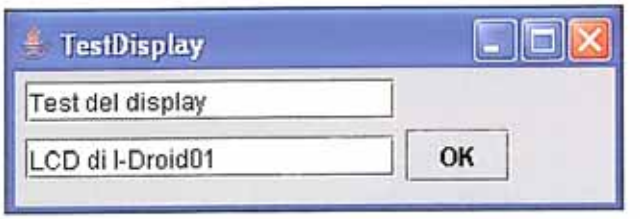
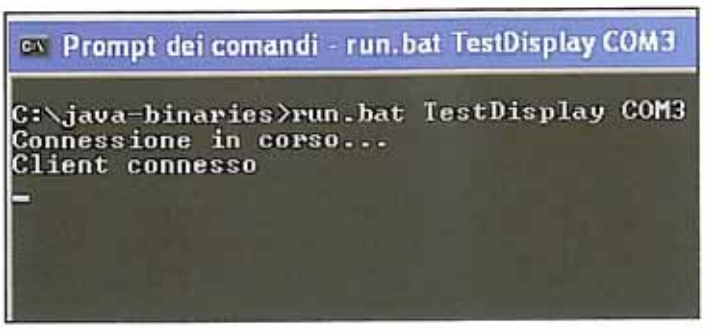
IL CONTROLLO DEL DISPLAY IN JAVA

I-Droid01 è dotato di diversi dispositivi di 'output': LED, hand tool, motori e display. Proprio quest'ultimo è l'oggetto del programma di esempio Java descritto in queste pagine, grazie al quale sarà possibile controllare da PC i messaggi testuali da far visualizzare al display del torso.

Il programma di esempio in Java proposto in queste pagine prosegue il discorso intrapreso nei fascicoli precedenti, relativo alle applicazioni di controllo, via interfaccia grafica, dei vari apparati del robot. Così, dopo i LED della testa e le braccia, è il momento del display montato nel torso di I-Droid01. L'interfaccia grafica proposta presenta due righe, nelle quali sarà possibile inserire fino a 16 caratteri per ognuna, rispettando così le caratteristiche del display. Premendo il pulsante di conferma, il testo sarà inviato al robot, che provvederà a visualizzarlo sul display stesso. Ancora una volta, il codice del programma è strutturato in due classi: **DisplayControl** fornirà il supporto necessario al controllo del display, mentre **TestDisplay** realizzerà il pannello grafico e gestirà l'interfaccia.





Sopra, il pannello di gestione del display, con i due campi per la scrittura dei messaggi e il pulsante di conferma. Sotto a sinistra, il prompt con il lancio dell'esecuzione.

DISPLAYCONTROL

La prima classe del programma è **DisplayControl**. Come detto, essa implementa i metodi di gestione del display stesso. Molti dei metodi inclusi nella classe sono stati scritti riutilizzando (e modificando quando necessario) quelli proposti nell'esempio AnimateDisplay, incluso nel terzo CD-ROM. Dopo le opportune (e solite) dichiarazioni di pacchetto e di importazione (che, in questo caso, per la prima volta riguardano anche alcune classi di supporto per la Motherboard, modulo che gestisce il display), viene presentato il codice della classe vera e propria, compreso quanto riguarda i metodi da essa forniti. Si tratta, essenzialmente, di alcune funzioni per l'accesso alla scrittura (**writeToDisplay**), al controllo (**checkLCD**) e alla cancellazione (**clearLCD**) del display, più un metodo (**createLcdBusData**) necessario per la gestione 'a basso livello' del display stesso, tramite la lettura di appositi registri della Motherboard e l'invio di opportuni dati al bus di comunicazione con la Motherboard stessa.

TESTDISPLAY

La classe **TestDisplay**, invece, gestisce l'interfaccia. Dopo le normali dichiarazioni di pacchetto e di importazione, il codice relativo descrive gli attributi (soprattutto grafici e di comunicazione con il robot) dell'oggetto **TestDisplay**, oltre ai suoi metodi. Essi comprendono il **main** di esecuzione del programma, il costruttore **TestDisplay** che lancia la connessione e invoca la realizzazione del pannello grafico, **createDisplayPanel** che costruisce nel concreto il pannello e i metodi **connetti** e **disconnetti**. Inoltre, all'interno di **TestDisplay** sono incluse due sotto-classi. La prima, **ButtonHandler**, ha il solito compito di monitorare il pannello in attesa di comandi da parte dell'utente. La seconda, di nome **MaxLengthDoc**, contiene un attributo e due metodi. L'attributo **maxChars**

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di una classe di controllo per il display di I-Droid01 e di una classe per la realizzazione di un pannello di gestione del display stesso. Per quanto riguarda la prima classe (DisplayControl), vengono mostrate le dichiarazioni iniziali di pacchetto (necessaria per l'esecuzione del programma finale sfruttando il file batch run.bat incluso nel terzo CD) e di importazione. Tra queste ultime figurano anche quelle relative alla comunicazione con la Motherboard, responsabile della gestione del display. A seguire, sono indicati gli attributi della classe e le dichiarazioni dei vari metodi. Riguardo a questi ultimi, viene qui mostrato solo il codice del metodo costruttore DisplayControl; gli altri metodi vanno invece completati inserendo il codice descritto nella pagina seguente. Per quanto riguarda la classe di implementazione del pannello grafico di gestione del display (TestDisplay), sono mostrate le dichiarazioni iniziali (sia di pacchetto sia di importazione), quelle degli attributi della classe e dei metodi. Oltre a tali dichiarazioni, è indicato il codice delle due sotto-classi ButtonHandler e MaxLengthDoc. Riguardo ai metodi, va segnalato che anche in questo caso essi vanno completati con il codice relativo, mostrato a pagina 16.

CLASSE DISPLAYCONTROL

```
package communication.examples;

import communication.handler.bus.BusData;
import communication.handler.bus.BusHandler;
import communication.handler.internal.InternalHandler;
import communication.handler.internal.InternalModule;
import communication.handler.internal.MotherboardData;
import communication.handler.internal.MotherboardRegister;
import java.io.IOException;

public class DisplayControl {

  private BusHandler bus;
  private InternalHandler internal;

  public DisplayControl(BusHandler bus, InternalHandler internal) {
      this.bus = bus;
      this.internal = internal;
  }

  public void writeToDisplay(String line1, String line2) { [...] }

  private boolean checkLCD() throws IOException, InterruptedException
{ [...] }

  public void clearLCD() { [...] }

  private BusData createLcdBusData(String line1, String line2) { [...] }

} // class DisplayControl
```

CLASSE TESTDISPLAY

```
package communication.examples;
import communication.handler.ProtocolHandler;
import communication.handler.internal.InternalHandler;
import communication.handler.bus.BusHandler;
import communication.transport.ConnectionProvider;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;

public class TestDisplay extends JFrame {
  private JPanel displayPanel;
  private JButton ok;
  private JTextField line1;
  private JTextField line2;
  private static String portName;
  private ProtocolHandler protocol;
  private InternalHandler internal;
  private BusHandler bus;
  private DisplayControl display;

  public static void main(String[] args) { [...] }

  public TestDisplay() { [...] }

  JPanel createDisplayPanel() { [...] }

  private class ButtonHandler implements ActionListener {
      public void actionPerformed(ActionEvent e) {
          if(e.getSource()==ok) {
              display.writeToDisplay(line1.getText(),line2.getText());
          }
      }
  } // class ButtonHandler

  private class MaxLengthDoc extends PlainDocument {
      private int maxChars;
      public MaxLengthDoc(int length) {
          maxChars = length;
      }
      public void insertString(int offs, String str, AttributeSet a)
throws BadLocationException {
          int length = getLength();
          if(length + str.length() <= maxChars) {
              super.insertString(offs, str, a);
          } else {
              int allowed = maxChars - length;
              super.insertString(offs, str.substring(0, allowed), a);
          }
      }
  } // class MaxLengthDoc

  private boolean connetti() { [...] }

  private void disconnetti() { [...] }

} // class DisplayTest
```

indica la lunghezza massima delle stringhe che possono essere visualizzate dal display. Tale valore viene assegnato dal primo dei metodi della classe, il costruttore omonimo. Nel caso del programma presentato, la lunghezza massima è settata a 16. Oltre al costruttore, la sotto-classe **MaxLengthDoc** presenta un altro metodo, **insertString**: esso, in sostanza, gestisce le stringhe troppo lunghe, troncandole se necessario. Va segnalato, infine, che il metodo **connetti**, rispetto a quello usualmente proposto, è 'arricchito' da alcune istruzioni. Infatti, esso inizializza l'oggetto **display** (**new DisplayControl**) e, una volta connesso il PC al robot, 'ripulisce' il display stesso (**display.clearLCD**).

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe DisplayControl. Essa fornisce i metodi di controllo del display, il cui codice potrà essere riutilizzato in altri programmi, dove fosse necessaria la gestione del display stesso. Viene mostrato il codice di implementazione dei metodi writeToDisplay, checkLCD, clearLCD e createLcdBusData, mentre il codice del costruttore DisplayControl è stato inserito nella descrizione della classe (pagina precedente). Il metodo writeToDisplay inizializza un canale di comunicazione (o bus) con il display, sfruttando il metodo createLcdBusData, poi, se il display è pronto (checkLCD), ne cancella il contenuto preesistente (clearLCD), per finire inviando il nuovo testo (bus.busTransfer). Essenzialmente, gli altri metodi della classe forniscono il supporto necessario a writeToDisplay. I metodi checkLCD e clearLCD non fanno altro che (rispettivamente) verificare lo stato del display interrogando l'apposito registro del robot e scrivere sul display stesso due righe da 16 spazi ciascuna. Il metodo createLcdBusData, infine, si occupa della gestione del bus di comunicazione tra PC e Motherboard e dell'invio delle stringhe da visualizzare.

METODI DELLA CLASSE DISPLAYCONTROL

```
public void writeToDisplay(String line1, String line2) {

    if ((line1.length()==0) && (line2.length()==0)) return;

    BusData lcd;
    lcd = createLcdBusData(line1, line2);

    try {

        if (!checkLCD()) return;

        clearLCD();
        bus.busPause(100);
        bus.busTransfer(lcd);
        lcd.waitForResult();

    } catch (Exception e) {
        e.printStackTrace();
    }

} // writeToDisplay


private boolean checkLCD() throws IOException, InterruptedException {

    int[] buf1 = new int[1];
    internal.readRegister(InternalModule.MOTHERBOARD,
MotherboardRegister.LCD_CONTROL, buf1);
    return 0 == (buf1[0] & MotherboardData.LCD_ERROR);

} // checkLCD


public void clearLCD() {

    BusData lcd;
    lcd = createLcdBusData("                ", "                ");
// i parametri sono due stringhe composte da 16 spazi ciascuna

    try {

        if (!checkLCD()) return;

        bus.busPause(100);
        bus.busTransfer(lcd);
        lcd.waitForResult();

    } catch (Exception e) {
        e.printStackTrace();
    }

} // clearLCD


private BusData createLcdBusData(String line1, String line2) {

    int num = 1;
    int len1 = -1;

    if (line1.length()>0) {
        len1 = Math.min(line1.length(), 16);
        ++num;
    }

    int len2 = -1;

    if (line2.length()>0) {
        len2 = Math.min(line2.length(), 16);
        ++num;
    }

    BusData lcd = new BusData(num);

    if (len1 > 0) {
        byte[] buf = new byte[len1 + 1];
        buf[0] = (byte) (MotherboardRegister.LCD_LINE1);
        System.arraycopy(line1.getBytes(), 0, buf, 1, len1);
        lcd.addWriteMessage(BusData
.internalAddress(InternalModule.MOTHERBOARD), buf);
    }

    if (len2 > 0) {
        byte[] buf = new byte[len2 + 1];
        buf[0] = (byte) (MotherboardRegister.LCD_LINE2);
        System.arraycopy(line2.getBytes(), 0, buf, 1, len2);
        lcd.addWriteMessage(BusData
.internalAddress(InternalModule.MOTHERBOARD), buf);
    }

    byte[] buf1 = new byte[2];
    buf1[0] = MotherboardRegister.LCD_CONTROL;
    buf1[1] = MotherboardData.LCD_WRITE |
MotherboardData.LCD_EXTERNAL;

    lcd.addWriteMessage(
BusData.internalAddress(InternalModule.MOTHERBOARD), buf1);

    return lcd;

} // createLcdBusData
```

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe TestDisplay. Essa gestisce il pannello grafico, mostrato sul monitor del PC, che costituisce l'interfaccia di controllo del display. Tale pannello mostra due campi editabili (in cui è possibile scrivere i messaggi da far visualizzare al display) e un pulsante di conferma. Viene mostrato il codice dei metodi main (che cura l'esecuzione del programma complessivo), del costruttore TestDisplay (che invoca la realizzazione del pannello e gestisce la connessione e la disconnessione, lanciando quando necessario l'esecuzione dei metodi relativi), di createDisplayPanel (che realizza nel concreto il pannello grafico), di connetti (che, al solito, costruisce il canale di comunicazione via Bluetooth tra PC e I-Droid01 e, in più, in questo caso crea un oggetto DisplayControl e ripulisce il display una prima volta) e disconnetti (il cui codice è uguale a quello mostrato negli esempi dei fascicoli precedenti).

METODI DELLA CLASSE TESTDISPLAY

```
public static void main(String[] args) {

    if (args.length < 1) {
        System.err.println("Specifica la porta da usare per la
connessione (es. COM5)");
        return;
    }
    portName = args[0];
    TestDisplay test = new TestDisplay();

} // main


public TestDisplay() {

    super("TestDisplay");
    addWindowListener(
        new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                disconnetti();
                System.exit(0);
            }
    });
    setBounds(0,0,300,100);
    displayPanel = createDisplayPanel();
    getContentPane().add(displayPanel);
    if (!connetti()) System.exit(1);
    setVisible(true);

} // TestDisplay


JPanel createDisplayPanel() {

    JPanel panel = new JPanel();
    FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
    panel.setLayout(layout);
    ok = new JButton("OK");
    line1 = new JTextField(16);
    line2 = new JTextField(16);
    line1.setDocument(new MaxLengthDoc(16));
    line2.setDocument(new MaxLengthDoc(16));
    ButtonHandler buttonHandler= new ButtonHandler();
    ok.addActionListener(buttonHandler);
    panel.add(line1);
    panel.add(line2);
    panel.add(ok);
    return panel;

} // createDisplayPanel


private boolean connetti() {

    ConnectionProvider provider =
new SerialJavaxCommProvider(portName, 9600);
    protocol = new ProtocolHandler(provider, 9600);
    internal = new InternalHandler();
    bus = new BusHandler();
    display = new DisplayControl(bus, internal);
    protocol.addCapabilityHandler(internal);
    protocol.addCapabilityHandler(bus);
    System.out.println("Connessione in corso...");

    try {
        if (!protocol.connect()) {
            System.err.println("Impossibile connettersi!");
            return false;
        }
        if (!protocol.setup(5000)) {
            System.err.println("Settaggio della connessione fallito!");
            protocol.disconnect(true);
            return false;
        }
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

    System.out.println("Client connesso");
    display.clearLCD();

    return true;

} // connetti


private void disconnetti() {

    System.out.println("Disconnessione in corso...");

    try {
        protocol.disconnect(false);
    }
    catch (Exception e) {
        e.printStackTrace();
        return;
    }

    System.out.println("Client disconnesso");

} // disconnetti
```

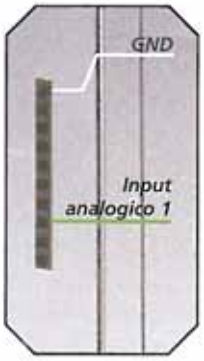
UN 'TESTER' PER PILE IN C-LIKE

Grazie alla sua dotazione e ai diversi strumenti di programmazione, I-Droid01 si presta a realizzare svariate funzioni. Ad esempio, basta un apposito codice in C-like e due cavi per breadboard per realizzare un voltmetro per pile.

Abbiamo visto nelle pagine precedenti come controllare il display da PC. Adesso, invece, vediamo come sfruttare lo stesso display per misurare il voltaggio fornito dalle pile stilo, mediante un programma in C-like. Il tutto sarà possibile collegando, tramite i fili per breadboard allegati al fascicolo 67, i due poli della pila da testare ai pin del connettore del marsupio indicati come GND (per il polo negativo della pila) e Input analogico 1 (per il polo positivo). Va sottolineato che le pile da testare non devono fornire un voltaggio superiore a 3 volt, massimo valore gestibile dagli ingressi del connettore.

IL PROGRAMMA TESTER

Come prima cosa, il codice del programma **Tester** inizia con le dichiarazioni **#include**, che permettono l'importazione di **c-like.h** e **robot.h**. Poi vengono dichiarati il comportamento **TesterPile** e la funzione **aggiornaDisplay**. A seguire, viene il codice del comportamento **Main**, che inizia lanciando la funzione preimpostata **lcd_clear**, per 'ripulire' il display. Il **Main** prosegue avviando **TesterPile**, che legge, tramite protocollo I2C, il valore fornito dal pin 'Input analogico 1' del marsupio. A tale scopo, in alternativa si può usare la funzione della libreria C-like **new_value = get(analog_in).pin_1;** relativa alla lettura dei valori di input analogici. In ogni caso, se il valore rilevato è diverso da quello misurato in precedenza, esso viene mostrato sul display. Di ciò si occupa **aggiornaDisplay**, che converte il valore dell'input analogico e lo 'scrive', come numero decimale, sul display.



A sinistra, i pin del connettore di interfaccia con la breadboard posto sul marsupio. Perché il programma Tester funzioni correttamente, bisogna collegare (usando ad esempio i fili per breadboard) il pin GND (il primo in alto) al polo negativo della pila da testare e il pin Input analogico 1 (il terzo dal basso) al polo positivo.

ESEMPI DI PROGRAMMAZIONE

Codice C-like del programma Tester (da memorizzare in un file dal nome tester.clike). Esso permette di visualizzare sul display di I-Droid01 il voltaggio rilevato collegando i poli di una pila ai pin del connettore del marsupio GND e Input analogico 1. Il programma effettua il calcolo del voltaggio usando i numeri di tipo float. Tale tipologia di calcoli, in generale, può comportare un oneroso carico di lavoro per i processori del robot: in applicazioni C-like più complesse, quindi, può essere opportuno evitarla. Va ricordato infine di non testare pile con voltaggio superiore a 3 volt.

CODICE DEL PROGRAMMA TESTER

```
#include "c-like.h"
#include "robot.h"

declare( behavior(TesterPile) );

void aggiornaDisplay();

define( behavior(Main))
{
    lcd_clear();
    start(TesterPile);
}

unsigned char old_value = 0;

define( behavior(TesterPile) )
{
    unsigned char new_value;
    old_value = new_value;
    i2c_read(44, 8, &new_value, 1);
    if (new_value != old_value)
        aggiornaDisplay(new_value);
    msleep(100);
}

void aggiornaDisplay(unsigned char val)
{
    char buffer[17];
    float x = (float) ( (3.000f * val) / 255.0f );
    snprintf(buffer, 17, "%0.3f Volt", x);
    lcd_write_string(1, 1, buffer);
}
```