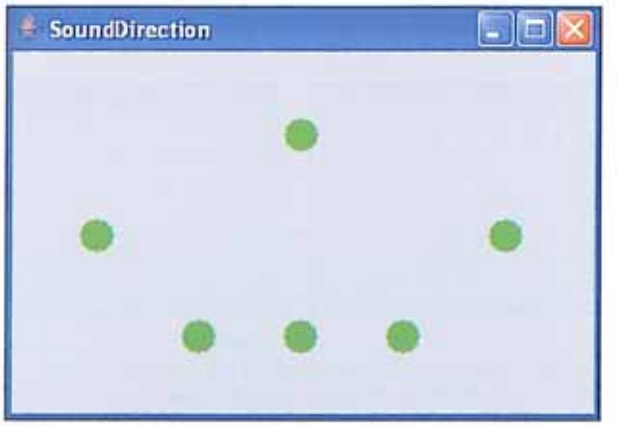
UN DISPLAY SOUND DIRECTION IN JAVA

In queste pagine viene mostrato un nuovo esempio di codice Java per I-D01. Stavolta, si tratta di un display grafico che permette di visualizzare la direzione di provenienza dei suoni secchi rilevata dal robot grazie al Sound Follower.

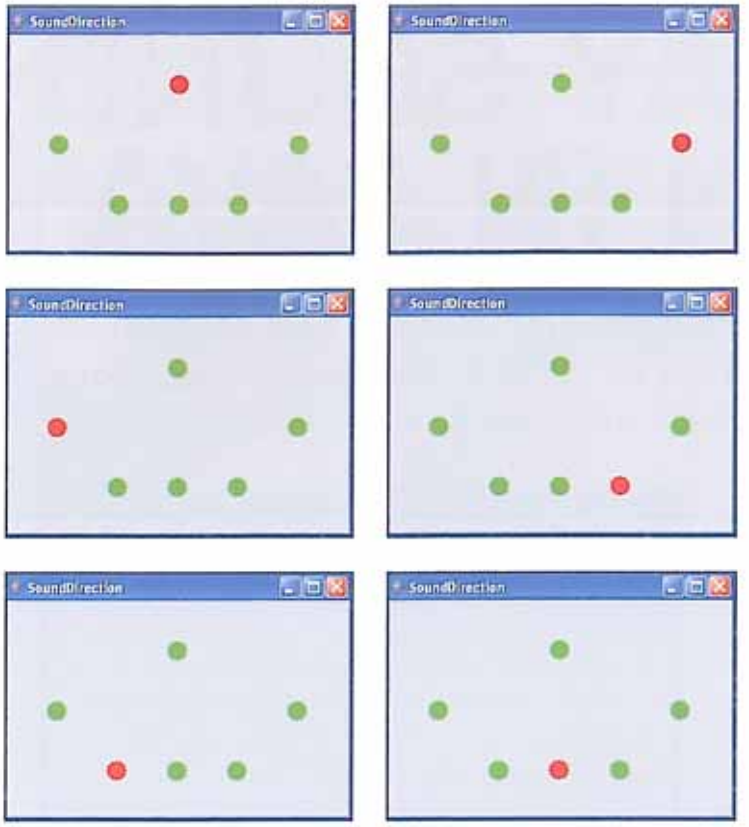
Abbiamo già avuto modo di presentare, nel corso dei fascicoli precedenti, svariati esempi di codice Java relativi a programmi di interazione con I-Droid01. In particolare, fino a questo punto sono stati mostrati diversi controllori con interfaccia grafica, che permettono di gestire il funzionamento di alcuni apparati del robot agendo su oggetti mostrati a video dal PC. Anche con l'esempio descritto in questo fascicolo si realizza un'interfaccia grafica che, però, stavolta non riguarda un controllore, bensì un monitor: essa, infatti, servirà a 'monitorare' la direzione dei suoni secchi percepiti dal robot, percezione che il robot ha sempre, anche quando il comportamento 'segui suoni' (che invece si riferisce all'azione del robot di voltarsi in direzione del suono percepito) non è attivo. Il display che verrà visualizzato sullo schermo del PC mostrerà il risultato delle elaborazioni effettuate continuamente dalla scheda Sound Follower quando il robot 'sente' suoni come battiti di mani o schiocchi di dita. Come negli altri esempi fin qui descritti, anche in questo caso il programma è composto da due classi, **DisplayPanel** e **SoundDirection**, da memorizzare rispettivamente nei file **DisplayPanel.java** e **SoundDirection.java**. Per la compilazione e l'esecuzione valgono le solite considerazioni più volte riportate in passato.

Qui sopra, una schermata del prompt dei comandi che mostra il lancio dell'esecuzione del programma SoundDirection, con indicazione della porta seriale virtuale di collegamento tra PC e I-Droid01, in questo caso COM3. Più in alto a destra, invece, il pannello che viene visualizzato a monitor: i cerchi (a 'riposo' di colore verde) indicano le sei possibili direzioni che il robot identifica come provenienza dei suoni secchi percepiti.



LA CLASSE DISPLAYPANEL

La prima classe, **DisplayPanel**, si occupa di gestire i cambiamenti che intercorrono nel display grafico. In pratica, cioè, tale classe descrive le azioni da compiere quando viene rilevata una particolare direzione di provenienza del suono. Rispetto alle classi di controllo viste nei fascicoli precedenti, il codice della classe **DisplayPanel** è piuttosto semplice. Come consueto, si comincia con le dichiarazioni **package** e **import**. Queste ultime riguardano alcune classi di supporto Java per elementi grafici. La stessa classe **DisplayPanel**, in effetti, non fa altro che 'estendere' una classe pre-elaborata Java, **JPanel**, che modellizza appunto i pannelli grafici standard. Ciò significa che un oggetto **DisplayPanel** è dotato di tutti i metodi e attributi della classe standard **JPanel** più quelli definiti all'interno del codice della stessa **DisplayPanel**. L'unico attributo 'specifico' di quest'ultima è un numero intero, chiamato **sensor**: esso assume un valore compreso tra 0 e 6, a seconda della direzione di provenienza del suono (l'assegnamento viene effettuato da un apposito metodo della classe **SoundDirection**). Sulla base del valore di **sensor**, il primo metodo di **DisplayPanel**, **paintComponent**, si preoccupa di modificare i cerchi del display mostrato a video, colorandone uno di



Qui sopra, le varie possibili configurazioni del pannello, sulla base della direzione dei suoni percepiti rispetto alla testa del robot. In senso orario, partendo dall'alto a sinistra: suono percepito posteriormente, suono percepito lateralmente a sinistra, suono percepito frontalmente a sinistra, suono percepito frontalmente, suono percepito frontalmente a destra, suono percepito lateralmente a destra.

rosso: proprio tale cerchio indicherà la direzione rilevata. In concreto, il metodo **paintComponent** risulta costituito essenzialmente da un blocco **switch**, che discrimina il valore assunto da **sensor** e richiama di conseguenza le azioni più corrette, mostrando i cerchi di colore opportuno. Il secondo, e ultimo, metodo inserito nel codice della classe **DisplayPanel** (**draw**) non fa altro che fornire il supporto necessario per il corretto funzionamento del programma, 'ricopiando' il valore di **sensor** e lanciando il metodo **repaint** pre-definito nella classe **JPanel**.

LA CLASSE SOUNDDIRECTION

Se **DisplayPanel** gestisce le modifiche nella colorazione dei cerchi all'interno del pannello grafico, **SoundDirection** si occupa della gestione generale del programma e della lettura degli appositi registri di I-D01 relativi ai suoni percepiti. Di seguito alla dichiarazione di pacchetto, il codice di **SoundDirection** mostra le importazioni necessarie. Si tratta delle classi di supporto per I-Droid01 relative alla comunicazione con il PC e alla lettura dei registri della scheda Sound Follower, oltre a specifiche classi di supporto Java, comprendenti alcune tra quelle per la gestione delle interfacce grafiche e degli eventi che si generano al loro interno. Come nel caso di **DisplayPanel**, anche **SoundDirection** estende una classe standard, **JFrame**, modellizzante un frame generico. In gergo Java, per frame si intende una finestra mostrata a video: nel nostro caso si tratta della finestra nella quale viene mostrato il display di monitoraggio della direzione dei suoni. Gli attributi specifici della classe **SoundDirection** comprendono i 'soliti' **portName**, **protocol** e **internal**, relativi alla comunicazione robot-PC. A questi si aggiungono **displayPanel** (oggetto della classe omonima), **t** (di tipo **Thread**) e un attributo vero/falso, di nome **active**. Questi ultimi due attributi rivestono una particolare importanza nel funzionamento del programma, in quanto consentono al display di monitorare con costanza lo stato dei registri di direzione del suono. Il primo dei metodi mostrati nel codice della classe **SoundDirection** è il metodo principale. Come al solito, esso verifica la presenza di un parametro che indichi la porta di connessione da usare, e poi lancia l'esecuzione del costruttore **SoundDirection**. Quest'ultimo si occupa di gestire la chiusura della finestra del programma (con conseguente disconnessione dal robot e terminazione del programma stesso), poi crea un'istanza di **DisplayPanel**, mostrando infine il display a video per la prima volta con tutti i cerchi colorati di verde (**displayPanel.draw**, con parametro pari a 0). Una volta lanciata la connessione, il metodo costruttore **SoundDirection** 'crea' l'attributo **t** di tipo **Thread**, associandolo a un nuovo oggetto appartenente alla mini classe **RefreshDisplay**. Quest'ultima viene definita all'interno di **SoundDirection** e implementa l'interfaccia Java standard di nome **Runnable**. Questa classe permette di gestire una particolare tipologia di oggetti Java, di cui fanno parte i thread, ossia 'processi' che continuano a essere eseguiti in modo 'parallelo' agli altri programmi. In pratica, una volta avviato un thread, questo continua a effettuare la propria esecuzione fino a quando non viene fermato. Tornando alla mini classe **RefreshDisplay**, essa descrive il metodo **run**, caratteristico di ogni classe che implementi **Runnable** e che specifica le operazioni da far eseguire al 'processo parallelo'. Nel nostro caso, quando attivo (ciclo **while** con variabile di controllo **active**), il thread deve leggere i registri del Sound Follower, memorizzare in **sensor** la direzione rilevata, richiamare il metodo **draw**

di **DisplayPanel** e poi aspettare dieci millesecondi prima di ripetere il tutto (**Thread.sleep** con parametro pari a 10). In questo modo, quindi, mentre il programma 'principale' controlla il display mostrato a video (e nel caso l'utente chiuda la finestra relativa, interrompe tutto), il thread continua a controllare i registri del Sound Follower, verificando se un suono viene rilevato e da quale direzione. Il codice della classe, infine, si chiude con i soliti metodi **connetti** e **disconnetti**.

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di una classe di gestione per un pannello grafico per il monitoraggio della direzione dei suoni secchi percepiti da I-Droid01 tramite la scheda Sound Follower. La classe mostrata, che estende JPanel, in particolare si occupa di mostrare a video alcuni cerchi colorati in modo opportuno a seconda del valore dell'attributo numerico sensor.

CLASSE DISPLAYPANEL

```
package communication.examples;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DisplayPanel extends JPanel {

    private int sensor = 0;


    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        switch(sensor) {

            case 0: {
                g.setColor(Color.GREEN);
                g.fillOval(160,40,20,20);
                g.fillOval(280,100,20,20);
                g.fillOval(220,160,20,20);
                g.fillOval(160,160,20,20);
                g.fillOval(100,160,20,20);
                g.fillOval(40,100,20,20);
            }
            break;

            case 1: {
                g.setColor(Color.RED);
                g.fillOval(160,40,20,20);
                g.setColor(Color.GREEN);
                g.fillOval(280,100,20,20);
                g.fillOval(220,160,20,20);
                g.fillOval(160,160,20,20);
                g.fillOval(100,160,20,20);
                g.fillOval(40,100,20,20);
            }
            break;

            case 2: {
                g.setColor(Color.RED);
                g.fillOval(280,100,20,20);
                g.setColor(Color.GREEN);
                g.fillOval(160,40,20,20);
                g.fillOval(220,160,20,20);
                g.fillOval(160,160,20,20);
                g.fillOval(100,160,20,20);
                g.fillOval(40,100,20,20);
            }
            break;

            case 3: {
                g.setColor(Color.RED);
                g.fillOval(220,160,20,20);
                g.setColor(Color.GREEN);
                g.fillOval(160,40,20,20);
                g.fillOval(280,100,20,20);
                g.fillOval(160,160,20,20);
                g.fillOval(100,160,20,20);
                g.fillOval(40,100,20,20);
            }
            break;

            case 4: {
                g.setColor(Color.RED);
                g.fillOval(160,160,20,20);
                g.setColor(Color.GREEN);
                g.fillOval(160,40,20,20);
                g.fillOval(280,100,20,20);
                g.fillOval(220,160,20,20);
                g.fillOval(100,160,20,20);
                g.fillOval(40,100,20,20);
            }
            break;

            case 5: {
                g.setColor(Color.RED);
                g.fillOval(100,160,20,20);
                g.setColor(Color.GREEN);
                g.fillOval(160,40,20,20);
                g.fillOval(280,100,20,20);
                g.fillOval(220,160,20,20);
                g.fillOval(160,160,20,20);
                g.fillOval(40,100,20,20);;
            }
            break;

            case 6: {
                g.setColor(Color.RED);
                g.fillOval(40,100,20,20);
                g.setColor(Color.GREEN);
                g.fillOval(160,40,20,20);
                g.fillOval(280,100,20,20);
                g.fillOval(220,160,20,20);
                g.fillOval(160,160,20,20);
                g.fillOval(100,160,20,20);
            }
            break;

        } // switch

    } // paintComponent


    public void draw(int sensor) {

            this.sensor = sensor;

            repaint();

    } // draw


} // class DisplayPanel
```

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di una classe di controllo dei registri relativi alla direzione di provenienza dei suoni secchi percepiti da I-Droid01. La classe, che estende JFrame, contiene il metodo principale del programma (main). All'interno della classe viene definita anche una mini classe (RefreshDisplay) che implementa Runnable. Tale mini classe definisce l'esecuzione dell'attributo t di tipo Thread. Esso si occupa di interrogare i registri relativi alla direzione di provenienza dei suoni e di invocare di conseguenza il metodo draw della classe DisplayPanel.

CLASSE SOUNDDIRECTION

```
package communication.examples;

import communication.handler.ProtocolHandler;
import communication.handler.internal.InternalHandler;
import communication.transport.ConnectionProvider;
import communication.handler.internal.InternalModule;
import communication.handler.internal.HeadData;
import communication.handler.internal.HeadRegister;

import java.io.IOException;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SoundDirection extends JFrame {

  private static String portName;
  private ProtocolHandler protocol;
  private InternalHandler internal;

  DisplayPanel displayPanel;
  Thread t;
  boolean active = false;


  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];
     SoundDirection gui = new SoundDirection();
  }

  public SoundDirection () {
     super("SoundDirection");
     addWindowListener(
           new WindowAdapter() {
              public void windowClosing(WindowEvent e) {
                 active = false;
                 disconnetti();
                 System.exit(0);
              }
           });
     setBounds(0,0,350,250);
     displayPanel =  new DisplayPanel();
     displayPanel.draw(0);
     getContentPane().add(displayPanel);
     if (!connetti()) System.exit(0);
     t = new Thread(new RefreshDisplay());
     active = true;
     t.start();
     setVisible(true);
  } // class constructor

  private class RefreshDisplay implements Runnable {

     int[] buf = new int[1];
     int sensor = 0;

     public void run() {
        while(active) {
           try {
              internal.readRegister(InternalModule.HEAD,
HeadRegister.SOUND_AND_TOUCH, buf);
              sensor = buf[0] >> 1;
              displayPanel.draw(sensor);
              Thread.sleep(10);
           } catch (Exception e) {
              e.printStackTrace();
           }
        }
     }

  } // class RefreshDisplay();


  private boolean connetti() {

     ConnectionProvider provider = new
SerialJavaxCommProvider(portName,9600);

     protocol = new ProtocolHandler(provider, 9600);
     internal = new InternalHandler();
     protocol.addCapabilityHandler(internal);

     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");
     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

} // class SoundDirection
```