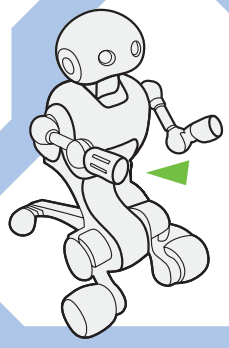
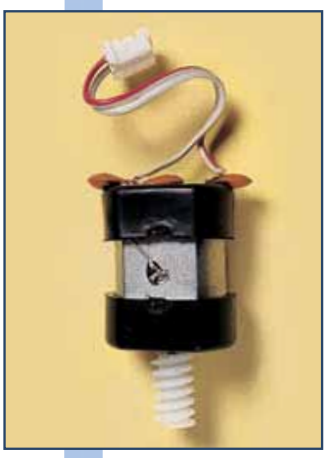
IL MOTORE PER LA PINZA DI I-DO1

Il componente allegato a questo fascicolo è un motore elettrico. Esso servirà a movimentare il dito opponibile della pinza, così da permettere la presa di piccoli oggetti.



Con i fascicoli che hai ricevuto ultimamente (in particolare 76 e 77), hai cominciato a collezionare gli elementi che comporranno la 'mano' di I-Droid01. Anche questo fascicolo è relativo allo stesso dispositivo: dopo le dita e gli elementi dentati, è il momento del motore elettrico. Esso è dotato di una vite senza fine, montata direttamente sull'albero motore: grazie alla vite, il movimento prodotto dal motore può essere direttamente trasferito alla cascata di ingranaggi interni all'avambraccio-pinza, senza dover ricorrere ad alcuna cinghia. Lo stesso motore elettrico troverà posto nella struttura dell'avambraccio, assieme anche a una piccola scheda elettronica (Hand). Quest'ultima avrà il compito di gestire il controllo del motore elettrico (e perciò della posizione del dito mobile), controllo possibile grazie al cavo a due fili che fuoriescono dal motore. Tramite tali fili, il motore viene alimentato e, di conseguenza, posto in movimento in un verso (per chiudere la pinza) o nell'altro (per aprirla). Per il momento, in attesa di ricevere i componenti che realizzeranno la struttura dell'avambraccio, tieni da parte il motore assieme agli altri elementi relativi alla 'mano': prossimamente potrai cominciare l'assemblaggio dell'avambraccio-pinza, iniziando proprio dal posizionamento del motore.

COMPONENTI

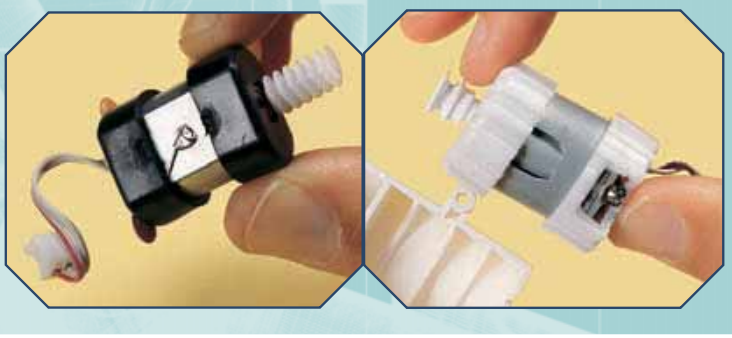


1. Motore elettrico per la pinza

IL MOTORE

DATI

A differenza degli altri sette motori elettrici di cui è dotato I-Droid01 (sotto a destra quello per il movimento destra/sinistra del collo), il motore elettrico della 'mano' (sotto a sinistra) non è dotato di puleggia, ma di vite senza fine. Proprio per questo motivo non verrà utilizzata alcuna cinghia nel sistema di trasmissione del moto.

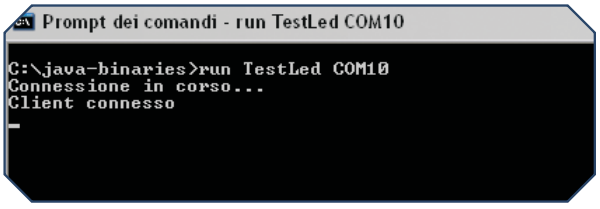


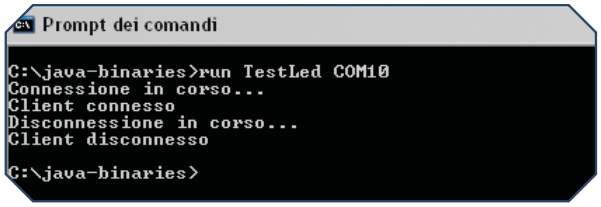
IL CONTROLLO DEI LED IN JAVA

Il linguaggio Java è molto versatile e consente l'implementazione di varie tipologie di programmi. Anche per I-Droid01 vale lo stesso: ad esempio, è possibile realizzare un pannello di controllo per i LED della testa del robot.

Con il codice presentato nelle prossime pagine inizia la carrellata di alcuni programmi di esempio, realizzati in Java, che utilizzano concretamente varie funzioni messe a disposizione dai moduli di I-Droid01. Dopo aver visto un esempio di metodi per la connessione e la disconnessione tra PC e robot (fascicolo 77, pagina 12 e seguenti), infatti, passiamo ora a un programma che implementa un semplice pannello di accensione e spegnimento dei LED presenti negli occhi e nelle orecchie di I-D01 (per queste ultime è opportuno disattivare prima il riconoscimento vocale, che altrimenti le controllerebbe facendole accendere e spegnere periodicamente). Il programma è stato organizzato in due classi Java distinte, una che modella il pannello in sé (**TestLed**), l'altra che fornisce le funzioni per l'accensione, lo spegnimento e il lampeggio dei LED (**LedControl**). Esse possono fornire alcuni metodi riutilizzabili in programmi personalizzati e, in ogni caso, costituiscono un primo interessante esempio riguardante una 'vera' interazione tra PC e I-D01 tramite Java. Per quanto riguarda la compilazione e l'esecuzione del programma di esempio, è possibile attenersi alle stesse indicazioni fornite nei fascicoli precedenti, equiparando, quindi, l'applicazione qui presentata a quelle degli esempi forniti con il terzo CD. Anche per il nome dei file in cui va memorizzato il codice vale quanto detto in passato: le due classi devono essere salvate con file a loro omonimi (in questo caso **LedControl.java** e **TestLed.java**).

Sopra, il pannello mostrato a monitor dall'applicazione Java descritta in queste pagine. Il pannello è costituito da due box di selezione, un menu a tendina e tre pulsanti di scelta.





A sinistra, due schermate che mostrano il prompt dei comandi durante e dopo l'esecuzione del programma TestLed. Sopra, il lancio dell'esecuzione del programma, utilizzando il file batch run.bat; sotto l'aspetto del prompt alla chiusura del pannello e, quindi, dell'intera applicazione.

LE DUE CLASSI

Come detto, il programma di controllo dei LED qui proposto è suddiviso in due classi. Ognuna di esse, in pieno rispetto dello 'spirito' del linguaggio Java, descrive gli attributi e i metodi caratterizzanti una tipologia di 'oggetto'. Nell'esempio, gli oggetti implementati sono il 'controllore LED' (classe **LedControl**) e il 'tester per i LED' (classe **TestLed**). Il primo fornisce un sistema per la gestione via PC dei LED posti nella testa del robot, tramite un controllo 'a basso livello' (ossia intervenendo direttamente su alcuni parametri di funzionamento propri di I-Droid01). La classe può quindi essere riutilizzata per tutti quei programmi che necessitano della gestione dei LED, nella stessa forma con la quale viene qui presentata. La classe **TestLed**, invece, si occupa di definire il pannello che viene mostrato a monitor su PC e che agisce da interfaccia grafica.

LEDCONTROL

Il codice della classe **LedControl** inizia con le dichiarazioni di **import**, che nel caso specifico riguardano essenzialmente alcune classi incluse nella directory **communication.handler.internal**. Tali classi forniscono l'accesso ai parametri interni di funzionamento citati prima. In realtà, infatti, l'accensione, lo spegnimento e il lampeggio dei LED degli occhi e delle orecchie vengono gestiti scrivendo opportuni valori in alcuni registri di I-Droid01. A parte il costruttore della classe (metodo **LedControl**), che si limita a settare una variabile di funzionamento del programma (**internal**), gli altri metodi di **LedControl** necessitano, come parametro, di un numero, che definisce un colore. Le dichiarazioni interne alla classe servono a definire la corrispondenza tra numeri e colori. Il codice dei metodi, poi, utilizzerà il parametro colore per definire le particolari stringhe di caratteri (ossia sequenze di lettere e cifre numeriche) che devono essere scritte nei registri del robot. Così, ad esempio, l'accensione con il colore verde dell'occhio destro avviene indicando (funzione **internal.writeRegister**) nel registro della testa del robot (**InternalModule.HEAD**) che si è intenzionati ad accendere (**HeadRegister.LED_ON**) i LED verdi dell'occhio destro (parametro **buf** nel quale è stata memorizzata la sequenza di caratteri 0x80, che ingloba l'indicazione di 'colore verde' e di 'occhio destro'). A colori e occhi (o orecchie) diversi corrispondono valori della variabile **buf** diversi, tutti però rappresentanti particolari numeri scritti in sistema esadecimale. Tale sistema, molto usato in informatica, rappresenta i numeri usando le cifre da 0 a 9 e le prime sei lettere dell'alfabeto (a, b, c, d, e, f). In Java, i numeri scritti in esadecimale vengono sempre preceduti dai caratteri '0x'. I vari metodi della classe **LedControl** mostrano un codice analogo: in sostanza, cambiano solo i parametri e i codici esadecimali da scrivere nei registri del robot.

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di una classe di controllo per i LED della testa di I-Droid01 (occhi e orecchie). Vengono mostrate solo le dichiarazioni di importazione e di pacchetto iniziali, quelle delle costanti (che indicano i numeri associati ai colori) e quella dell'attributo internal di tipo InternalHandler. I metodi della classe (il costruttore 'LedControl', 'ledOff', 'ledOffSx', 'ledOffDx', 'ledOn', 'ledOnSx', 'ledOnDx', 'ledBlink', 'ledBlinkSx' e 'ledBlinkDx') vengono solo citati. Il codice che ne implementa il funzionamento viene mostrato nelle prossime pagine.

CLASSE LEDCONTROL

```
package communication.examples;

import communication.handler.internal.InternalHandler;
import communication.handler.internal.HeadData;
import communication.handler.internal.HeadRegister;
import communication.handler.internal.InternalModule;

public class LedControl {

    public static final int YELLOW = 1;
    public static final int RED = 2;
    public static final int GREEN = 3;
    public static final int BLUE = 4;
    public static final int ALL = 5;

    private InternalHandler internal;

    public LedControl(InternalHandler internal) {
        this.internal = internal;
    } // LedControl

    public boolean ledOff(int color) { [...] }

    public boolean ledOffSx(int color) { [...] }

    public boolean ledOffDx(int color) { [...] }

    public boolean ledOn(int color) { [...] }

    public boolean ledOnSx(int color) { [...] }

    public boolean ledOnDx(int color) { [...] }

    public boolean ledBlink(int color) { [...] }

    public boolean ledBlinkSx(int color) { [...] }

    public boolean ledBlinkDx(int color) { [...] }

} // class LedControl
```

TESTLED

Se la classe **LedControl** implementa l'accesso a basso livello dei LED degli occhi e delle orecchie, risultando, pertanto, piuttosto ripetitiva e a schema fisso, quella **TestLed** si presta di più a personalizzazioni e modifiche. Come detto, la sua funzione è di definire l'interfaccia grazie alla quale è possibile da PC far accendere, spegnere o lampeggiare i LED di I-Droid01. Per far ciò è necessaria la visualizzazione sullo schermo di un pannello grafico. Proprio alla gestione di tale pannello servono le importazioni di determinate classi, facenti parte di alcuni pacchetti Java, in particolare **swing**, **awt**, e **awt.event**. Essi fanno parte di quell'insieme di classi 'pre-fabbricate'

incluse nel JDK (Java Development Kit), appositamente realizzate per fornire il supporto necessario (tra l'altro) alla creazione e alla gestione di oggetti come finestre, menu, ecc. Oltre a queste classi di supporto, la classe **TestLed** ne necessita di altre, da utilizzare stavolta per la connessione e disconnessione da I-D01: **communication.handler.ProtocolHandler**, **communication.handler.internal.InternalHandler**, **communication.transport.ConnectionProvider**.
Gli attributi della classe **TestLed**, come al solito dichiarati all'inizio della classe stessa, comprendono diversi oggetti per l'interfaccia grafica (ad esempio il pannello **ledPanel**, il pulsante **ledOnB**, il menu a tendina **ledColorCB**, i 'check box' **ledSxCB** e **ledDxCB**), realizzati in concreto grazie al metodo **createLedPanel**. A essi si aggiungono gli attributi per la gestione della connessione (**portName**, **protocol** e **internal**), una stringa per la definizione dei colori (**ledColor**), un oggetto 'controllore dei LED' (**LedControl**) e, infine, un oggetto per la gestione degli oggetti grafici (**ledHandler**).
La definizione di quest'ultimo è effettuata da una 'mini-classe' Java, inclusa nel codice della classe **TestLed**. Si tratta di **LedButtonHandler**, che non fa altro se non 'implementare' il 'modello di classe' (o 'interfaccia', in gergo Java) di nome **ActionListener**: esso fornisce le funzioni necessarie al pannello per 'accorgersi' che l'utente ha utilizzato gli elementi grafici per effettuare certe scelte. Osservando il codice che implementa il metodo **actionPerformed**, infatti, si può notare come venga controllato se l'utente ha selezionato 'destra' e 'sinistra' nei due check box, per poi 'recuperare' il colore scelto tramite il menu a tendina (**ledColorCB.getSelectedIndex()+1**) e, in seguito, a seconda delle diverse combinazioni, richiamare gli opportuni metodi della classe **LedControl**.
Il funzionamento del programma, quindi, si basa molto sul metodo **actionPerformed**. Il costruttore della classe **TestLed**, invece, si limita a lanciare la connessione (**connetti**), a realizzare un'istanza dell'oggetto **LedControl** (**new LedControl** con parametro **internal**), a invocare la costruzione del pannello grafico (**createLedPanel**), e a gestire la chiusura del pannello grafico, che termina l'applicazione (**windowClosing**, all'inizio del codice del metodo **TestLed**). Il metodo **main**, infine, incluso anch'esso nella classe **TestLed**, non fa altro che controllare che sia stata inserita la porta di comunicazione tra PC e I-D01, per poi invocare l'esecuzione del costruttore **TestLed**.
I metodi **connetti** e **disconnetti**, per conto loro, sono del tutto identici a quelli presentati nell'esempio del fascicolo precedente.

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di un pannello grafico di gestione dei LED. I LED controllati sono quelli di occhi e orecchie, che possono essere accesi, spenti o fatti lampeggiare. Sono mostrate le dichiarazioni di pacchetto, di importazione e relative agli attributi della classe. Inoltre viene mostrata la dichiarazione della 'sotto-classe' LedButtonHandler. Essa, come anche i metodi della classe TestLed ('main', il costruttore 'TestLed', 'createLedPanel', 'connetti' e 'disconnetti'), viene solo citata. I metodi 'connetti' e 'disconnetti' hanno codice identico a quello mostrato nel fascicolo 77, pagine 13 e 14.

CLASSE TESTLED

```
package communication.examples;

import communication.handler.ProtocolHandler;
import communication.handler.internal.InternalHandler;
import communication.transport.ConnectionProvider;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TestLed extends JFrame {

    private JPanel ledPanel;
    private JButton ledBlinkB;
    private JButton ledOnB;
    private JButton ledOffB;
    private JComboBox ledColorCB;
    private JCheckBox ledSxCB;
    private JCheckBox ledDxCB;
    private String ledColor[] = {"Yellow","Red","Green","Blue","All"};
    private LedButtonHandler ledHandler;
    private static String portName;
    private ProtocolHandler protocol;
    private InternalHandler internal;
    private LedControl led;

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

    public TestLed() { [...] }

    JPanel createLedPanel() { [...] }

    private class LedButtonHandler implements ActionListener {
        public void actionPerformed(ActionEvent e) { [...] }
    } // class LedButtonHandler

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

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

} // TestLed
```

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe LedControl per l'accensione e lo spegnimento dei LED di occhi e orecchie. Viene mostrato il codice relativo ai metodi di spegnimento ('ledOff' per entrambi gli occhi/orecchie, 'ledOffSx' per occhio/orecchio sinistro, 'ledOffDx' per occhio/orecchio destro). Il codice corrispondente per i metodi di accensione ('ledOn', 'ledOnSx' e 'ledOnDx') si ottengono sostituendo le formule in grassetto alle corrispettive dei metodi di spegnimento. La scelta del colore indicato con BLUE implica la gestione dei LED delle orecchie.

METODI DI SPEGNIMENTO E ACCENSIONE LED

```
public boolean ledOff(int color) {
public boolean ledOn(int color) {

    int buf[] = new int[1];
    switch (color) {
        case BLUE:
            buf[0] = 0x11;
            break;

        case YELLOW:
            buf[0] = 0x22;
            break;

        case RED:
            buf[0] = 0x44;
            break;

        case GREEN:
            buf[0] = 0x88;
            break;

        case ALL:
            buf[0] = 0xFF;
            break;

        default:
            return false;
    }

    try {
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_OFF, buf);
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_ON, buf);
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

} // ledOff


public boolean ledOffSx(int color) {
public boolean ledOnSx(int color) {

    int buf[] = new int[1];
    switch (color) {
        case BLUE:
            buf[0] = 0x1;
            break;

        case YELLOW:
            buf[0] = 0x2;
            break;

        case RED:
            buf[0] = 0x4;
            break;

        case GREEN:
            buf[0] = 0x8;
            break;

        case ALL:
            buf[0] = 0xF;
            break;

        default:
            return false;
    }

    try {
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_OFF, buf);
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_ON, buf);
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

} // ledOffSx


public boolean ledOffDx(int color) {
public boolean ledOnDx(int color) {

    int buf[] = new int[1];
    switch (color) {
        case BLUE:
            buf[0] = 0x10;
            break;

        case YELLOW:
            buf[0] = 0x20;
            break;

        case RED:
            buf[0] = 0x40;
            break;

        case GREEN:
            buf[0] = 0x80;
            break;

        case ALL:
            buf[0] = 0xF0;
            break;

        default:
            return false;
    }

    try {
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_OFF, buf);
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_ON, buf);
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

} // ledOffDx
```

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe LedControl per il controllo del lampeggio dei LED di occhi e orecchie ('ledBlink', 'ledBlinkSx', 'ledBlinkDx'). Il codice è del tutto analogo a quello dei metodi per l'accensione e lo spegnimento dei LED. Le uniche differenze riguardano i parametri della funzione di scrittura nel registro di I-Droid01 (writeRegister). Anche i numeri in base esadecimale utilizzati per la scelta dei vari colori e per la selezione degli occhi/orecchie (destro, sinistro, entrambi) sono identici a quelli impiegati nell'accensione e nello spegnimento. Inoltre, anche in questo caso la scelta del colore BLUE permette di gestire i LED delle orecchie. A tale proposito va ricordato che i metodi di accensione, spegnimento e lampeggio funzionano correttamente con le orecchie solo nel caso sia stato disattivato il riconoscimento di comandi vocali da parte di I-D01: se ciò non avviene, il controllo dei LED delle orecchie è gestito autonomamente dal modulo Voice.

METODI DI LAMPEGGIO LED

```
public boolean ledBlink(int color) {

    int buf[] = new int[1];
    switch (color) {
        case BLUE:
            buf[0] = 0x11;
            break;

        case YELLOW:
            buf[0] = 0x22;
            break;

        case RED:
            buf[0] = 0x44;
            break;

        case GREEN:
            buf[0] = 0x88;
            break;

        case ALL:
            buf[0] = 0xFF;
            break;

        default:
            return false;
    }

    try {
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_BLINK, buf);
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

} // ledBlink


public boolean ledBlinkSx(int color) {

    int buf[] = new int[1];
    switch (color) {
        case BLUE:
            buf[0] = 0x1;
            break;

        case YELLOW:
            buf[0] = 0x2;
            break;

        case RED:
            buf[0] = 0x4;
            break;

        case GREEN:
            buf[0] = 0x8;
            break;

        case ALL:
            buf[0] = 0xF;
            break;

        default:
            return false;
    }

    try {
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_BLINK, buf);
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

} // ledBlinkSx


public boolean ledBlinkDx(int color) {

    int buf[] = new int[1];
    switch (color) {
        case BLUE:
            buf[0] = 0x10;
            break;

        case YELLOW:
            buf[0] = 0x20;
            break;

        case RED:
            buf[0] = 0x40;
            break;

        case GREEN:
            buf[0] = 0x80;
            break;

        case ALL:
            buf[0] = 0xF0;
            break;

        default:
            return false;
    }

    try {
        return internal.writeRegister(InternalModule.HEAD,
HeadRegister.LED_BLINK, buf);
    }
    catch (Exception e) {
        e.printStackTrace();
        return false;
    }

} // ledBlinkDx
```

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe TestLed. Essa realizza il pannello grafico di controllo dei LED e, in più, gestisce la connessione e la disconnessione tra PC e I-Droid01. A tale proposito, i metodi 'connetti' e 'disconnetti' non sono mostrati, in quanto del tutto identici a quelli già descritti nel fascicolo 77, pagine 13 e 14. Il metodo 'main' si limita a verificare che l'utente abbia inserito la porta di connessione da utilizzare per la comunicazione tra computer e robot. In caso positivo, viene invocato il costruttore della classe TestLed. Esso gestisce la chiusura del pannello grafico e la conseguente disconnessione dal robot, tenta la connessione con I-D01, lancia l'esecuzione del costruttore LedControl della classe omonima (che si preoccupa della gestione 'a basso livello' dei LED) e finisce con l'invocare la creazione del pannello. A sua volta, il metodo 'createLedPanel' inizializza i vari elementi grafici del pannello e rimane in attesa di eventi generati da questi ultimi, utilizzando a tal proposito il metodo 'actionPerformed', della 'mini-classe' LedButtonHandler. Il metodo 'actionPerformed' non fa altro che controllare lo stato degli elementi grafici, in attesa che l'utente li utilizzi per effettuare eventuali scelte a proposito dei LED. In corrispondenza di tali scelte, vengono invocati gli opportuni metodi dell'oggetto led, istanza della classe LedControl.

METODI DELLA CLASSE TESTLED

```
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];
    TestLed testLed = new TestLed();

} // main


public TestLed() {

    super("TestLed");

    addWindowListener(
        new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                disconnetti();
                System.exit(0);
            }
    });

    if (!connetti()) System.exit(1);

    led = new LedControl(internal);

    setBounds(0,0,400,70);
    ledPanel =  createLedPanel();
    add(ledPanel);
    setVisible(true);

} // TestLed
```

```
JPanel createLedPanel() {

    JPanel panel = new JPanel();
    ledBlinkB = new JButton("Blink");
    ledOffB = new JButton("Off");
    ledOnB = new JButton("On");
    ledSxCB = new JCheckBox("SX");
    ledDxCB = new JCheckBox("DX");
    ledColorCB = new JComboBox(ledColor);
    panel.add(ledSxCB);
    panel.add(ledDxCB);
    panel.add(ledColorCB);
    panel.add(ledOnB);
    panel.add(ledBlinkB);
    panel.add(ledOffB);
    LedButtonHandler lbh = new LedButtonHandler();
    ledOffB.addActionListener(lbh);
    ledOnB.addActionListener(lbh);
    ledBlinkB.addActionListener(lbh);
    return panel;

} // createLedPanel


private class LedButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        boolean sx,dx;

        sx = ledSxCB.isSelected();
        dx = ledDxCB.isSelected();

        int color = ledColorCB.getSelectedIndex()+1;

        // Entrambi i led
        if (sx&&dx) {
            if (e.getSource() == ledOnB) {
                led.ledOn(color);
            } else if (e.getSource() == ledOffB) {
                    led.ledOff(color);
                } else if (e.getSource() == ledBlinkB) {
                        led.ledBlink(color);
                    }
            return;
        }

        // Solo led sinistro
        if (sx) {
            if (e.getSource() == ledOnB) {
                led.ledOnSx(color);
            } else if (e.getSource() == ledOffB) {
                    led.ledOffSx(color);
                } else if (e.getSource() == ledBlinkB) {
                        led.ledBlinkSx(color);
                    }
            return;
        }

        // Solo led destro
        if (dx) {
            if (e.getSource() == ledOnB) {
                led.ledOnDx(color);
            } else if (e.getSource() == ledOffB) {
                    led.ledOffDx(color);
                } else if (e.getSource() == ledBlinkB) {
                        led.ledBlinkDx(color);
                    }
            return;
        }
    }

} // class LedButtonHandler
```