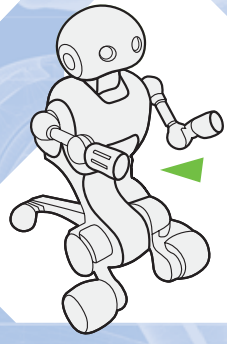
LE RUOTE DENTATE DELLA 'MANO'



La pinza sarà in grado di afferrare oggetti grazie a un apposito motore elettrico. Gli elementi allegati serviranno a trasmettere il moto da quest'ultimo al dito opponibile.

Continua la collezione dei pezzi relativi alla pinza di I-Droid01. Dopo i componenti delle dita, allegati al fascicolo precedente, è ora il momento delle ruote dentate che realizzeranno il meccanismo di movimento delle dita stesse. Più precisamente, il moto riguarderà il dito centrale e opponibile, che verrà posto in rotazione proprio dalla cascata di ingranaggi costituiti dagli elementi allegati. Al momento essi non possono essere posizionati, in quanto manca la struttura portante della 'mano', che ti verrà fornita prossimamente. Per il momento, quindi, è opportuno mantenere da parte le ruote dentate. Puoi sin d'ora, però, cominciare a osservarle, così da saperle distinguere quando verrà il momento di porle nella loro sede. Si tratta, in tutto, di sette elementi, tra cui una 'semiruota' e un albero metallico con ruota e semiruota dentate.

COMPONENTI

1. 2 doppie ruote piccole e medie
2. Semiruota dentata
3. Albero con ruota media e ruota piccola
4. Albero metallico con elementi dentati
5. Doppia ruota media e medio-piccola
6. Doppia ruota media e piccola

A sinistra, uno degli elementi allegati. Si tratta dell'albero in materiale plastico con due ruote dentate, una media e una piccola.

GLI ELEMENTI DENTATI

DATI

In futuro, ci riferiremo agli elementi allegati con i nomi della lista qui a sinistra. In particolare, distingueremo tra 'doppia ruota piccola e media' (a destra), 'doppia ruota media e piccola' (sotto a destra) e 'doppia ruota media e medio-piccola' (sotto).

I METODI JAVA PER LA CONNESSIONE

Una delle prime funzioni da implementare per poter programmare I-Droid01 in Java è, sicuramente, quella di connessione tra il PC e il robot. Nelle prossime pagine vediamo un esempio di codice che realizza proprio la connessione e disconnessione tra computer e I-D01, sfruttando le librerie di supporto.

Qualsiasi applicazione Java che voglia sfruttare I-Droid01, necessariamente, deve innanzitutto prevedere la connessione con il robot. Va ricordato, infatti, che i programmi scritti in Java vengono eseguiti dal PC e utilizzano I-Droid01 come una sorta di 'periferica intelligente', sfruttando il collegamento Bluetooth. Nel seguito vediamo un semplice esempio di come si possono implementare le funzioni che permettono al PC di collegarsi al robot e, in seguito, di disconnettersi da esso. Il codice che mostreremo viene presentato all'interno di una classe Java dal nome **TestConnessione**. Nella programmazione in linguaggio Java, infatti, il codice viene suddiviso in 'classi' che definiscono i cosiddetti 'oggetti'. Dall'interazione tra più oggetti di tipo diverso nasce poi l'esecuzione del programma. In questo semplice caso, è sufficiente un'unica classe, che implementa i metodi basilari di connessione e disconnessione e, allo stesso tempo, fornisce anche un semplice esempio di utilizzo degli stessi. Per qualsiasi riferimento alla programmazione a oggetti e in particolare a quella in linguaggio Java, è possibile consultare uno dei numerosi manuali relativi disponibili sul mercato, oppure uno tra quelli gratuiti presenti in Internet, ad esempio il famoso *Thinking in Java*. Prima di utilizzare le librerie Java di supporto fornite con il CD 3, invece, è opportuno fare riferimento ai file contenuti nella cartella doc (ottenuta scompattando il file **java-documentation.zip**), che forniscono tutta la documentazione relativa all'impiego corretto delle classi pre-elaborate.

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di una classe di connessione tra PC e I-Droid01. Vengono mostrate solo le dichiarazioni di 'pacchetto' e di importazione (introdotte da 'package' e 'import') e quella degli attributi (portName, protocol, internal). I metodi della classe (main, TestConnessione, connetti, disconnetti) vengono solo citati: per essere utilizzato, il codice va completato inserendo all'interno dei vari metodi i codici relativi, mostrati nelle pagine seguenti. Il codice della classe va memorizzato in un file dal nome TestConnessione.java (dove '.java' è l'estensione che il file deve avere).

CLASSE TESTCONNESSIONE

```
package communication.examples;

import communication.handler.ProtocolHandler;
import communication.handler.internal.InternalHandler;
import communication.transport.ConnectionProvider;

public class TestConnessione {

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

    public static void main(String[] args) {
        [...]  // qui va inserito il codice del main
    }

    public TestConnessione () {
        [...]  //  qui va inserito il codice del metodo TestConnessione
    }

    private boolean connetti() {
        [...]  // qui va inserito il codice del metodo connetti
    }

    private void disconnetti() {
        [...]  // qui va inserito il codice del metodo disconnetti
    }

} // class TestConnessione
```

LE LIBRERIE DI SUPPORTO E LE DICHIARAZIONI

La scrittura della classe **TestConnessione** inizia con alcune 'dichiarazioni' preliminari. Innanzitutto, vengono indicate le classi di supporto delle quali i metodi di **TestConnessione** (descritti nel seguito) avranno bisogno. Come già detto nel fascicolo 76, pagina 16, come prima cosa si può indicare che la classe che si sta scrivendo appartiene al pacchetto **communication.examples**: in questo modo sarà possibile compilare ed eseguire, in modo semplice, il programma realizzato, affiancandolo agli esempi inclusi nel terzo CD e contenuti nella directory **bin/communication/examples**.
In seguito, si devono inserire i riferimenti alle librerie di supporto: **communication.handler.ProtocolHandler**, **communication.handler.internal.InternalHandler** e **communication.transport.ConnectionProvider** (presenti nella cartella **src** ottenuta dal file **java-library.zip**). Esse fanno parte di quelle realizzate ad hoc per permettere al PC di comunicare con I-D01 tramite una connessione di tipo Bluetooth. Nel caso di **TestConnessione**, quelle che servono sono esattamente queste tre. Esse, come tutte le altre classi per I-Droid01 incluse nel CD, sono descritte nei relativi file di help presenti nella cartella **doc**. Completate queste dichiarazioni preliminari, è possibile cominciare a descrivere i dettagli della classe **TestConnessione**. Essa necessita di tre attributi, che tutti i metodi della classe potranno utilizzare. In particolare essi sono il nome della porta di comunicazione tra PC e I-D01 (**portName**) e altri due oggetti (ossia **protocol** e **internal**) che sfruttano i metodi di supporto alla comunicazione stessa.
È poi il turno dei metodi della classe: quello principale (**main**; è necessario per far partire l'esecuzione del programma finale), il 'costruttore' (**TestConnessione**, omonimo della classe; è un altro dei metodi basilari, che permette di 'realizzare' un oggetto della classe **TestConnessione**), il metodo per effettuare la connessione (**connetti**) e, infine, quello per avviare la disconnessione (**disconnetti**).

IL METODO CONNETTI

Partiamo dal metodo **connetti**. Esso realizza il 'cuore' del funzionamento della classe, fornendo la procedura necessaria al programma per collegarsi al robot. Il metodo mostrato va visto come una tra le possibili implementazioni, qui riportata per la sua 'essenzialità'; in realtà la stessa funzione può essere realizzata anche in modo diverso.
In ogni caso, ti potrà essere utile tenere da parte questo codice per utilizzarlo quando vorrai implementare un programma Java che preveda di collegarsi a I-Droid01. Passando al codice proposto, come prima cosa il metodo **connetti** crea un oggetto dal nome **provider**, di tipo definito dalla classe **SerialJavaxCommProvider** (inclusa nelle librerie di supporto), passando come parametri il nome della porta con cui effettuare la connessione (**portName**) e 9600 (la velocità, in bit al secondo, alla quale viene

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione del metodo 'connetti'. Dopo aver creato un canale di comunicazione e aver inizializzato il protocollo, il metodo prova a connettere il PC con I-Droid01 (invocando le funzioni protocol.connect e protocol.setup, fornite dalla classe di supporto ProtocolHandler).

METODO CONNETTI

```
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 connesione fallito!");
            protocol.disconnect(true);
            return false;
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return false;
    }

    System.out.println("Client connesso");
    return true;

} // connetti
```

ESEMPI DI PROGRAMMAZIONE

Codice del metodo 'disconnetti'. Molto semplicemente, la connessione tra PC e I-Droid01 viene conclusa invocando la funzione protocol.disconnect fornita dalla classe ProtocolHandler.

METODO DISCONNETTI

```
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
```

effettuata la comunicazione via porta seriale, ininfluente con le porte seriali virtuali Bluetooth utilizzate nella normale connessione tra I-D01 e PC). Poi il metodo passa a realizzare nel concreto gli oggetti **protocol** e **internal**, dichiarati all'inizio della classe **TestConnessione**. A seguire, il metodo **connetti** mostra a monitor la scritta "Connessione in corso..." (funzione **System.out.println**), poi utilizza un blocco **try-catch** (costrutti tipici del linguaggio Java, utili per la gestione degli errori che possono avvenire in corso di esecuzione di un programma). All'interno della 'zona' try viene concretizzata la procedura di connessione, grazie all'invocazione del metodo **protocol.connect**, anch'esso fornito dalle librerie di supporto. A seguire viene invocato il metodo **protocol.setup**, al quale viene passato un parametro, ossia il tempo in millisecondi che il PC dovrà al più attendere per una risposta da parte del robot. In questo esempio è stato utilizzato un valore indicativo, pari a 5000, equivalente a 5 secondi: se entro questo lasso di tempo il robot non risponde, il tentativo di connessione viene interrotto. Nel caso opposto (cioè se il robot risponde entro il tempo di attesa), invece, la connessione viene avviata a tutti gli effetti. A questo punto, se non avvengono errori, eventualmente segnalati dal relativo ramo catch del codice, il metodo mostra a video il messaggio "Client connesso", sempre utilizzando la funzione **System.out.println**. Con tale istruzione il metodo **connetti** termina.

IL METODO DISCONNETTI

Le operazioni necessarie per disconnettere il programma Java da I-Droid01 sono piuttosto semplici. Il metodo proposto inizia con la visualizzazione sullo schermo del messaggio "Disconnessione in corso...", che annuncia all'utente l'avvio della procedura. In seguito viene invocato, all'interno di un blocco **try-catch**, il metodo **protocol.disconnect**, al quale viene passato il parametro **false**. Tale metodo, pre-implementato nella classe **ProtocolHandler**, effettua la disconnessione vera e propria; il parametro **false** indica che l'interruzione della connessione con il robot deve avvenire in modo 'amichevole', ossia tramite una procedura secondo la quale I-D01 viene preventivamente 'avvisato' della prossima chiusura del collegamento. L'alternativa (ossia parametro **true** passato al metodo **protocol.disconnect**), invece, realizza una disconnessione immediata. Dopo l'invocazione del **protocol.disconnect**, il codice di **disconnetti** mostra il ramo catch, per la gestione di eventuali errori in fase di esecuzione; se non ci sono problemi, l'esecuzione del metodo prosegue con la visualizzazione del messaggio "Client disconnesso", per poi terminare.

IL COSTRUTTORE E IL MAIN

Come detto, all'interno della classe **TestConnessione** è presente anche un metodo omonimo della classe stessa, denominato, in gergo Java, 'costruttore'. In Java, in generale, ogni classe rappresenta la 'descrizione' di una tipologia di oggetti, e i metodi costruttori hanno la funzione di permettere la creazione di 'istanze' dei vari oggetti descritti dalle classi. In questo caso molto particolare, la classe presentata descrive quindi 'l'oggetto TestConnessione'. Il suo costruttore, di conseguenza, indica le operazioni da compiere per 'realizzare' una connessione di test. Il metodo **TestConnessione** inizia richiamando **connetti**. Se l'esecuzione di quest'ultimo fallisce, il costruttore termina immediatamente (funzione **System.exit** con parametro pari a 1, a indicare che l'uscita è stata forzata da un problema). Se, invece, la connessione ha buon fine, il costruttore semplicemente attende 5000 millisecondi (**Thread.sleep** con parametro 5000), per poi invocare il metodo **disconnetti**. Il tutto viene incluso nel solito blocco **try-catch**.

Infine, va descritto il metodo principale, ossia il **main**. Esso, in pratica, definisce l'esecuzione generale del programma Java. Nel caso di programmi costituiti da più classi, il metodo **main** è presente solo in una di esse e, a partire da tale classe 'principale', invoca i vari metodi necessari all'esecuzione. Nel nostro esempio, il **main** comincia col verificare che l'utente abbia indicato una porta seriale virtuale con la quale tentare la connessione. Proprio a questo controllo serve il test **if**, che usa come 'variabile di verità' l'espressione **args.length** < 1: se i parametri dati dall'utente sono minori strettamente di 1 (ossia, in linguaggio più comprensibile, se l'utente non ha inserito alcun parametro), allora certamente non è stata fornita la porta di connessione. In questo caso non sarà possibile tentare la connessione, e il **main** terminerà senza effettuare altre operazioni.

Se, invece, esiste almeno un parametro indicato dall'utente, il test **if** viene 'passato' e viene memorizzato il primo dei parametri (che dovrebbe anche essere l'unico) come nome della porta (**portName** = **args[0]**), per poi lanciare il metodo costruttore, il quale si occuperà del resto. Come detto, infatti, esso provvederà all'invocazione del metodo **connetti**, attenderà 5 secondi e poi effettuerà la disconnessione. I metodi costruttore e **main** costituiscono la 'struttura portante' dell'esempio proposto, caratterizzandone il comportamento complessivo. Probabilmente, quindi, non li riutilizzerai per i tuoi programmi Java, se non dopo averli modificati secondo le tue esigenze. Il codice dei metodi **connetti** e **disconnetti**, invece, può essere facilmente riutilizzato in qualsiasi programma per implementare le funzioni di connessione/disconnessione dal robot. Ciò può essere fatto, ovviamente, anche in altri modi, ad esempio invocando direttamente i metodi **connect** e **disconnect** pre-implementati dalla classe **ProtocolHandler** oppure, in alternativa, scrivendo 'da zero' i metodi **connetti** e **disconnetti**, personalizzandoli in modo opportuno. Puoi comunque tenere quelli qui proposti come base da cui iniziare.

ESEMPI DI PROGRAMMAZIONE

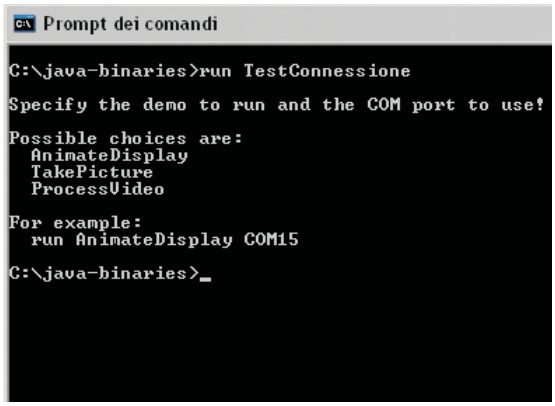
Codice dei metodi 'main' e 'TestConnessione'. Il primo è il metodo principale, che viene invocato automaticamente all'esecuzione del programma Java dopo la sua compilazione. In questo esempio, il metodo semplicemente verifica che all'esecuzione del programma sia stata indicata la porta seriale virtuale sulla quale è in ascolto I-Droid01, poi invoca l'esecuzione del metodo TestConnessione. Quest'ultimo è il metodo costruttore della classe, quello cioè che realizza l'oggetto descritto dalla classe stessa. In questo caso, il costruttore a sua volta lancia il metodo connetti, poi aspetta 5000 millisecondi (ossia 5 secondi) e in seguito invoca il metodo disconnetti, chiudendo di fatto la connessione con il robot.

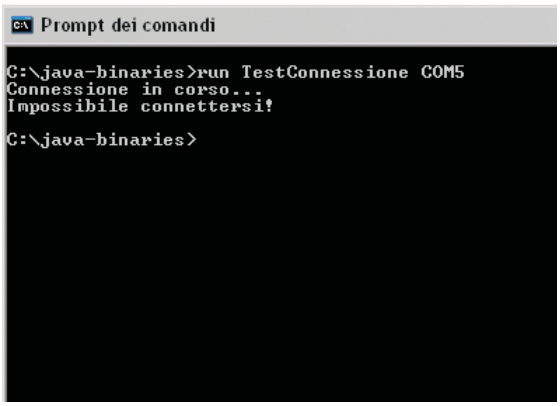
CLASSE TESTCONNESSIONE

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

    if (args.length < 1)
    {
        return;
    }

    portName = args[0];
    TestConnessione test = new TestConnessione();

} // main


public TestConnessione () {

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

    try {
        Thread.sleep(5000);
    }

    catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    }

    disconnetti();

} // TestConnessione
```

COMPILAZIONE ED ESECUZIONE

Una volta visto nel dettaglio il codice della classe **TestConnessione**, passiamo a vedere come effettuarne la scrittura, per poi procedere con la sua compilazione ed esecuzione. Il codice può essere scritto su un semplice file di testo (a tal proposito sono disponibili diversi semplici editor di scrittura, sufficienti allo scopo, tra i quali, per i sistemi Windows, Notepad o Blocco Note), oppure usando un ambiente di programmazione specifico (come Eclipse). In ogni caso, va ricordato di salvare il file nominandolo **TestConnessione.java**, dove '.java' indica l'estensione del file: essa, infatti, identifica i file contenenti il codice dei programmi scritti in questo linguaggio. Per quanto riguarda la posizione nella quale memorizzare il file, come già detto nel fascicolo 76, pagina 16, una soluzione 'di comodo' consiste nel porre il codice nella stessa





In senso orario a partire da qui sopra, le schermate del prompt relative all'esecuzione di TestConnessione in situazioni diverse: senza aver indicato la porta di connessione, avendo indicato una porta non corretta e, infine, con porta corretta.

directory dove sono memorizzati gli eseguibili dei file di esempio (memorizzati nella sottocartella **bin**) inclusi nel terzo CD-ROM. Dopo si può lanciare la compilazione della classe. In generale, la compilazione di un programma Java necessita che sul PC sia installato il JDK (Java Development Kit, disponibile sul sito www.java.sun.com, qualora non fosse già presente sul tuo PC). Tale kit di sviluppo, infatti, contiene le librerie di supporto basilari e, soprattutto, il compilatore Java. Si tratta di un particolare programma che, dati i file di codice sorgente .java, costruisce i file oggetto (cioè quelli eseguibili), identificati dall'estensione '.class'. Tornando al nostro esempio, si può quindi lanciare la compilazione dando al prompt il comando **javac TestConnessione.java**. Se la compilazione va a buon fine (ossia il compilatore non rileva alcun errore sintattico nel codice sorgente), verrà creato nella stessa cartella un file che porta il nome di **TestConnessione.class**. A questo punto non resta che lanciare l'esecuzione del programma. Per farlo, si può procedere utilizzando il file batch **run.bat** (vedi anche fascicolo 76, pagina 16). Sarà sufficiente spostarsi (nel prompt dei comandi) nella cartella dove tale file è memorizzato sul computer, per poi lanciare il comando:

run.bat TestConnessione 'porta-COM'

dove '**porta-COM**' è la porta seriale alla quale è associato I-Droid01. In alternativa, per l'esecuzione si può utilizzare il comando **java**, che va però utilizzato indicando alcuni parametri (necessari, in sostanza, per indicare dove si trovano le librerie di supporto per I-Droid01). Per avere un'idea di quali siano tali parametri, è possibile vedere (ed eventualmente modificare e salvare con altro nome) il contenuto del file **run.bat** fornito con il CD: è sufficiente aprire il file con un editor di testi. In ogni caso, è sempre opportuno mantenere una copia di salvataggio del file originale, così da non rischiare di effettuare modifiche che lo rendano inservibile.

```
File  Modifica  Formato  Visualizza  ?
@echo off
if "%1"=="" goto syntax
if "%2"=="" goto syntax
java -Djava.library.path=.\bin;.\lib -cp .\bin;.\lib\comm.jar communication.examples.%1 %2
goto end

:syntax
echo.
echo Specify the demo to run and the COM port to use!
echo.
echo Possible choices are:
echo   AnimateDisplay
echo   TakePicture
echo   ProcessVideo
echo.
echo For example:
echo   run AnimateDisplay COM15
echo.
:end
```

A sinistra, una schermata che mostra il contenuto del file run.bat fornito con il terzo CD. Aprendo tale file con un editor di testi, esso può essere modificato secondo le proprie esigenze.