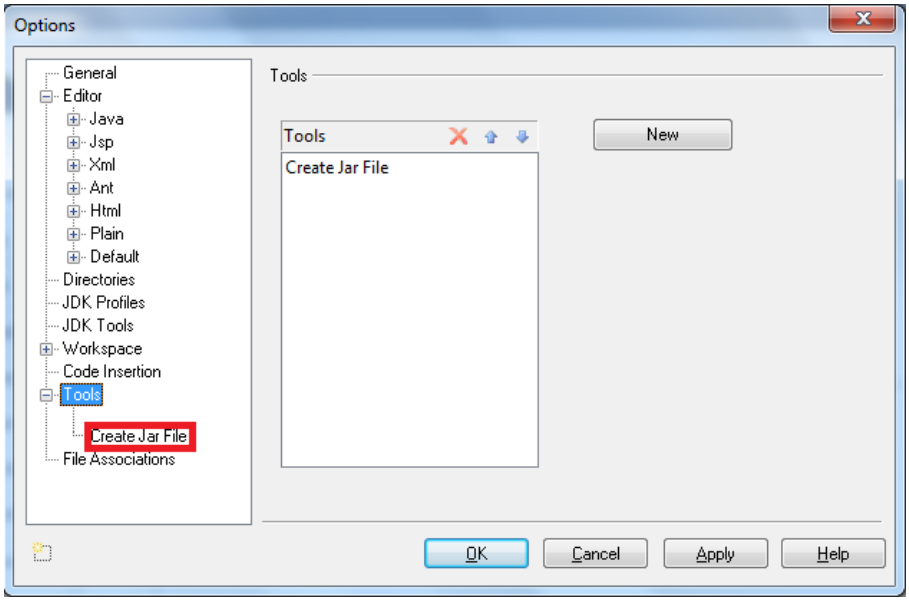
Creare JavaARchive con JCreator

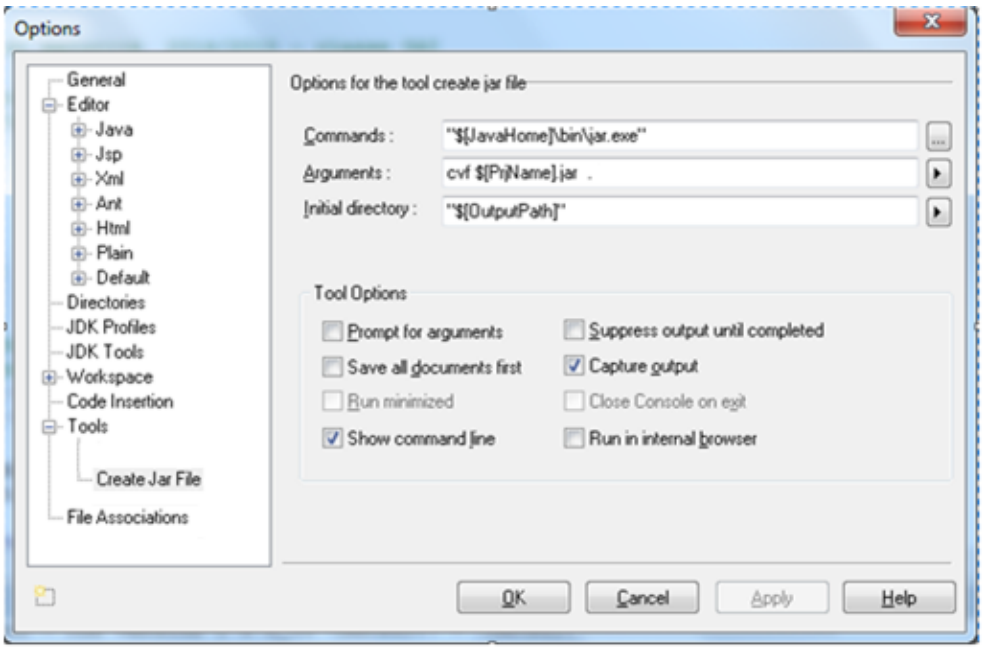
Tra le *Opzioni* di *Configurazione* in JCreator è prevista la creazione di file JAR non eseguibili



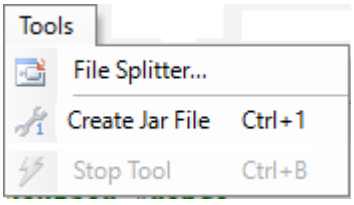
- clicca su *Configure/Options*
- Entra in *Tools*
- Fai *New* e scegli *Create Jar file*
- Ora entra sulla nuova opzione creata
- In *Arguments* trovi la stringa: cvf $[PrjName] .jar .

c per creare archivio
v per monitorare passi di compressione
f per specificare il nome del file JAR
. per aggiungere tutti i file e le sottocartelle

- Clicca OK



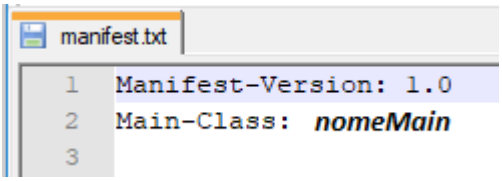
Uso del tool creato:



Se il progetto (l'applicazione) ha nome Menu, crea Menu. jar in **classes**
aggiungendo (comprimendolo) il contenuto delle sottocartelle .
crea automaticamente file Manifest.mf senza MainClass **da aggiungere**

Manifest-Version: 1.0
Created-By: 1.8.0_151 (Oracle Corporation)
Main-Class: Menu
<linea vuota>

Creare JAR eseguibile con JCreator



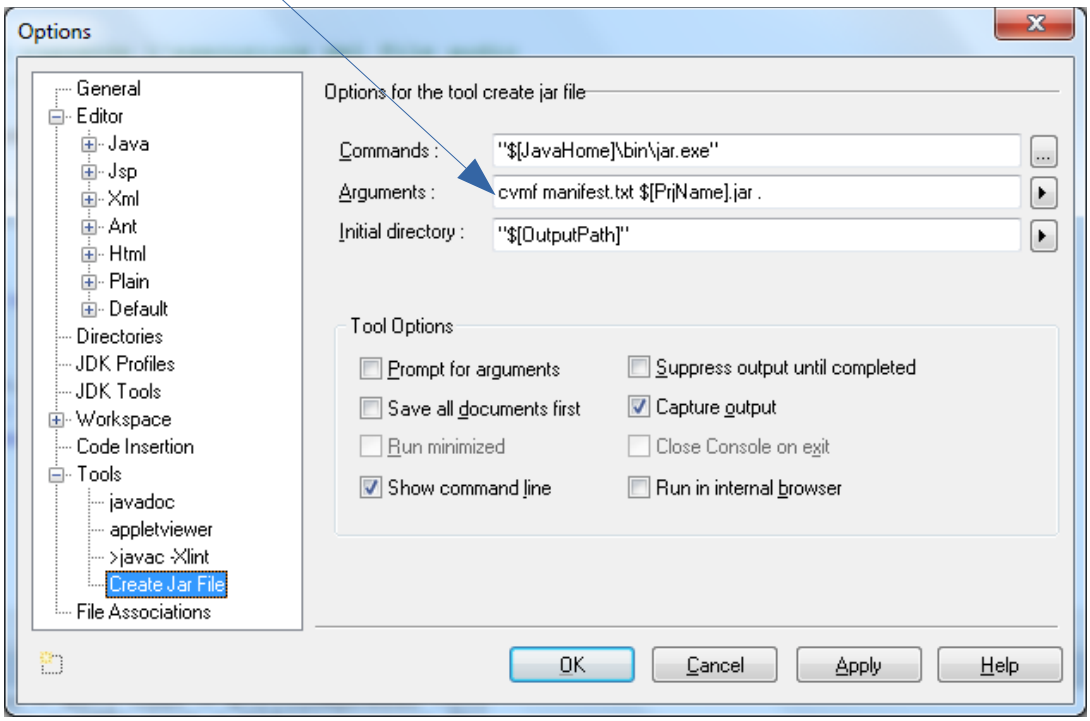
Per **creare eseguibile** è necessario:

1. un file *manifesto* che indica quale **file contiene il main**
2. nel codice, recuperare in *modo dinamico* le *risorse (immagini, audio etc…)*

Primo punto) si crea il file *manifesto* con nome diverso da quello creato ad es: **manifest.txt** salvandolo nella stessa cartella dei file bytecode

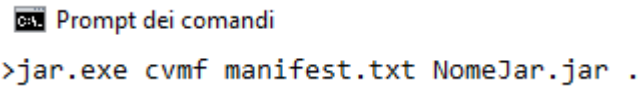
In *Arguments* si inserisce la stringa:

cvmf manifest.txt $[PrjName].jar .



. per aggiungere **directory**

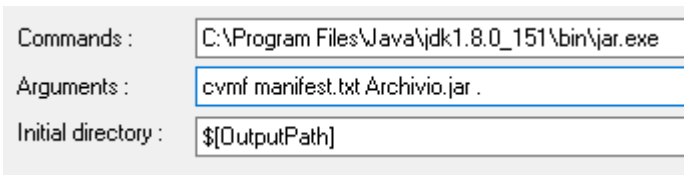
equivalente a: Command : "C:\Program Files\Java\jdk1.8.0_151\bin\jar.exe" **cvmf** manifest.txt *NomeJar*.jar **.**
Directory : "C:*path**NomeProgetto* **\classes**"

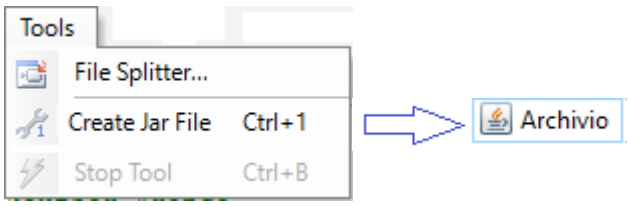


Attenzione*:* cvmf richiede prima il nome del file manifesto e dopo il nome del file jar
cvfm richiede prima il nome del file jar e dopo il nome del file manifesto

Nb: non comprime le sottocartelle (solo il contenuto delle stesse)

Se non si crea un progetto e si vuole *distribuire* l'applicazione o un insieme di classi come file JAR, il nome può essere qualsiasi: ad esempio **Archivio.jar**





Significato delle opzioni del comando per creare file JAR:

c	Crea un nuovo archivio o un archivio vuoto.
t	Elenca l'indice dei contenuti.
x	Estrae tutti i file.
x *file*	Estrae il file specificato.
f	Consente di specificare il nome del file JAR. Senza questa opzione, **jar** suppone che l'input proverrà da standard input oppure, in fase di creazione di un archivio, che l'output sarà destinato a standard output.
m	Indica che il primo argomento sarà il nome del file di manifesto creato dall'utente.
v	Produce una descrizione prolissa (*verbose*) dell'attività che **jar** sta eseguendo.
0 (zero)	Registra soltanto i file, senza comprimerli. Questa opzione è utilizzata per creare un archivio JAR da includere nel CLASSPATH.
M	Non crea automaticamente un file di manifesto.

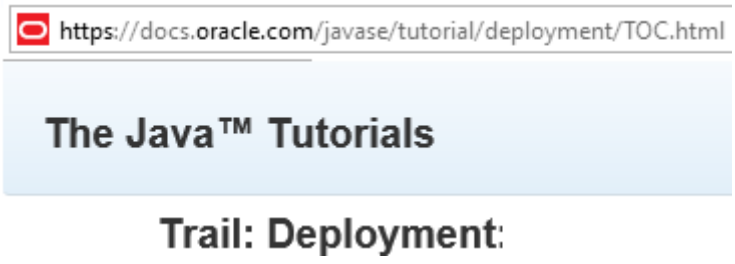
Although you don't have to understand what this means, here's a brief run-through anyways:

- the **c** means "create new archive"
- the **v** means "create verbose output"...can leave this out if you want without hurting anything
- the **m** means "I'm supplying the manifest file" (here, manifest.txt, which you saved in the current directory of the file, or "$[FileDir]\manifest.txt").
- the **f** means "I want to tell you what I'm naming the JAR file". Here, $[CurClass].jar. Whatever class is running in JCreator when you select the "Create Jar" option will be the name of your Jar. However, you can always rename it like you would any other file later on if you want it to be something else.
- The *.* means "I want to include every file in this directory into my JAR file. The first star is a wildcard meaning "any file name", and the .* means with any file extension (.java, .class, .jpg, or whatever other file types you might have in that folder).
- Putting "$[FileDir]" in place of "$[OutputPath]" makes it start working in the same class as your current file.

http://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html

➔ **jar command options** http://docs.oracle.com/javase/tutorial/deployment/jar/build.html

- **basic command format** / options and arguments used in this command http://docs.oracle.com/javase/tutorial/deployment/jar/modman.html



Secondo punto)
Se si vuole caricare una immagine nel file JAR (o altra risorsa: suoni etc..), e recuperarla correttamente in esecuzione, si deve *recuperare dinamicamente* l'id di tale immagine utilizzando l'URL e non un percorso relativo: esistono diversi metodi per farlo.

Una possibilità è usare il metodo con **getResourse()**
http://docs.oracle.com/javase/tutorial/uiswing/components/icon.html#getresource

Ad esempio, se all'interno del JAR ho un'immagine contenuta nel file '/img/logo.gif', compressa a partire dalla sottodirectory che contiene i *bytecode* (in JCreator **classes)**, per recuperarla non devo fare altro che invocare:

```
String name =    "/img/logo.gif"; // la ricerca è del file /img/logo.gif
oppure
String name =    "img/logo.gif";  // Il package della classe è usato come base
URL logoUrl = getClass().getResource(name);          // importando java.net.*;
Image img = new ImageIcon (logoUrl).getImage();
```

java.awt Class Image

Il metodo **getResource()** di java.lang.Class ritorna un oggetto di tipo java.net.URL

Tale metodo fa sì che il programma di caricamento classi (*class loader*) controlli le directory e i file JAR nel *classpath* del programma, restituendo un URL non appena trova il file desiderato. Se un altro file o directory JAR nel percorso della classe contiene il file il programma di caricamento classi restituisce la prima istanza che contiene il file.

```
URL getResource(String name)
    Finds a resource with a given name
```

Nb: per un confronto tra i metodi per ***recuperare dinamicamente risorse*** tramite URL

```
URL url = ClassLoader.getSystemResource(name);  // utilizza il classloader di sistema
URL url = getClass().getResource(name);  //  preferibile se le risorse sono memorizzate
                                         // in una gerarchia di directory
                                         // rispecchiando le classi del progetto
URL url = nomeClasse.class.getResource(name); // come il precedente delega alla classe
                                              // alla fine cercherà la risorsa fino
                                              // al classloader del sistema
```

Esempio:


immagini

nella sottocartella **immagini**
sono archiviate N immagini di nome 0.png, 1.png etc..

```
BufferedImage [] imgs = new BufferedImage[N];
String s = "";
for(int i=0;i<imgs.length;i++){
        try{
                s = "immagini/"+i+".png";
                imgs[i]= ImageIO.read(getURL(s));
        }catch(IOException ex){System.out.println(ex);}
  }
}

public URL getURL(String name) {          //metodo che ritorna
                                          // dinamicamente l'id dell'immagine
    URL url = ClassLoader.getSystemResource(name);
   //oppure URL url = getClass().getResource(name);
   return url;
}
```

scaricabile JAR

nb: per istanziare immagini (cap.26 – Inserire immagini o disegnare con ImageIcon)

javax.swing Interface Icon **Icon** icona = new **ImageIcon**(*url*); javax.swing Class ImageIcon

Un altro metodo utile per recuperare il contenuto di una risorsa (se non si ha il problema di creare JAR eseguibili) è **getResourceAtStream(String name)** che ritorna un oggetto di tipo java.io.InputStream (utile ad esempio per la lettura di file di testo).

ClassLoader: public InputStream getResourceAsStream(String name)

> Returns an input stream for reading the specified resource.
> The search order is described in the documentation for . `getResource(String)`
> **Parameters:** `name` - The resource name

Vedi esempio con uso **InputStreamReader** *utile per creare jar eseguibili*

Se si ha la necessità di recuperare il percorso con il quale il file JAR è stato lanciato

```
File f = new File (System.getProperty ("java.class.path"));
File dir = f.getAbsoluteFile ().getParentFile ();
```

Se si ha la necessità di recuperare percorsi di File

Gli url relativi a risorse contenute in un archivio JAR hanno un indicatore non compatibile con la definizione di un file quindi tentando di eseguire il seguente segmento di codice:

```
url=ThisClass.getResource("/resources/listaNomi.txt");
File myFile = new File(url.getPath());
```

si otterrebbe il seguente errore:

file:/C:/*mioProgramma*.jar!/resources/listaNomi.txt
java.io.FileNotFoundException: file:\C:\mioProgramma.jar!\resources\listaNomi.txt (La sintassi del nome del file, della directory o del volume è incorretta)

(notare quel punto esclamativo nel percorso restituito da getPath()).

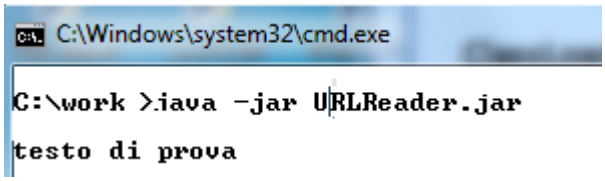
Soluzione: Noto l'URL, si può ottenere direttamente lo stream dall'url:

```
URL url=getClass().getResource("/resources/nomefile");
InputStream inStream=url.openStream();
```

Leggere file da url con Java con uso **InputStreamReader** in = new InputStreamReader(u.openStream());

```
import java.io.*;
import java.net.URL;

public class URLReader {

private String url;

public URLReader(String url) {
      this.url = url;
}

public String read() throws Exception {
      StringBuilder sb = new StringBuilder();    // performances :  catene di testo mutabili
      URL u=getClass().getResource(url);
      InputStreamReader in = new InputStreamReader(u.openStream());
      BufferedReader buf = new BufferedReader(in);
      String line;
      while ((line = buf.readLine()) != null) {
            sb.append(line);
       }
       return sb.toString();
}

public static void main(String[] args) throws Exception {
       URLReader rd = new URLReader("/prova.txt");
       System.out.println(rd.read());
}
}
```

Effetto dell'esecuzione da linea di commando



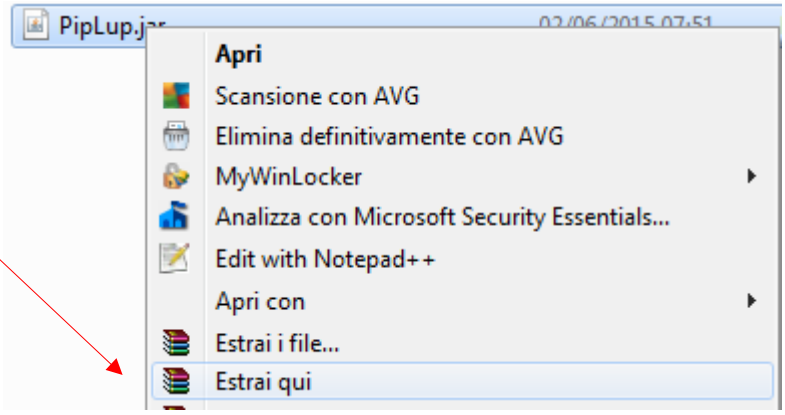
Passi successivi:

Common JAR file operations	
Operation	**Command**
To view the contents of a JAR file	`jar tf jar-file`
To extract the contents of a JAR file	`jar xf jar-file`
To extract specific files from a JAR file	`jar xf jar-file archived-file(s)`

per **vedere il contenuto** del .jar
(.jar è analogo a file .zip)

o da prompt

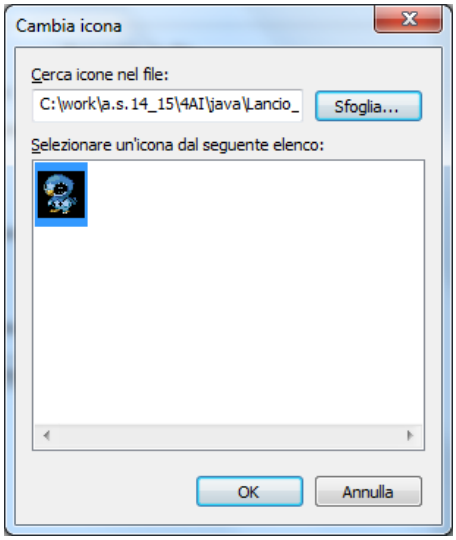
```
>jar xf nomeJAR.jar
```

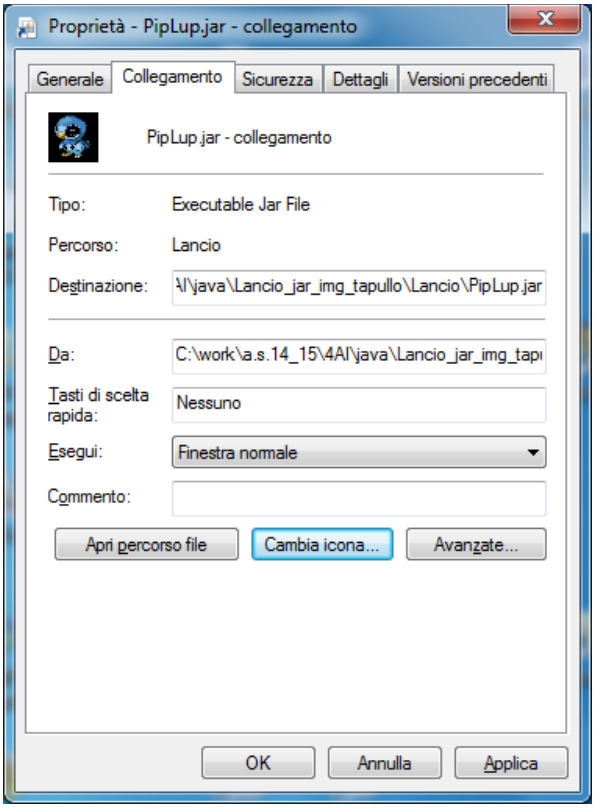


Se non si è previsto, nel codice, il *caricamento dinamico* delle risorse, per *distribuire* il programma si deve salvare in una sottocartella sia il **file compresso** sia le **risorse** (sottocartelle con immagini e audio)

Se è un programma grafico basta un **doppio click per eseguire**. (*scaricabile PipLup.jar*)

Si può creare collegamento e personalizzare l'icona





Creare icona

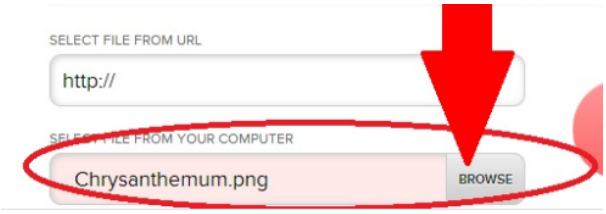
Cerca o crea l'immagine che userai come icona del programma. Considera che l'icona è una parte importante della grafica del programma. Viene usata ad ogni avvio dell'eseguibile, e, probabilmente, è sempre presente sul desktop! Cerca quindi di sceglierla accuratamente, in modo che sia indicativa del suo contenuto. La dimensione dell'icona deve essere **256x256 pixel**, per funzionare correttamente.

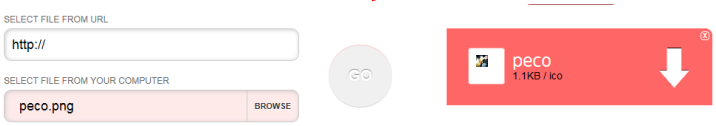
Salva come immagine **bitmap** e rinomala **.ico**

Alternativa: Apri il browser e vai sul sito convertico.com. Questo sito gratuito converte le immagini (.png, .jpg) in file per icone (.ico).

PNG to ICO/ICO to PNG Conversion

Puoi selezionare l'immagine indicando il suo URL, oppure indicando il percorso del file. Clicca su "Go" per confermare, e avviare la conversione



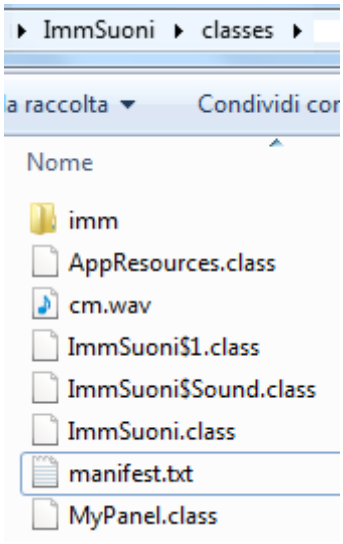


Se di dimensione **256x256** …. tempo troppo lungo (interrotto)

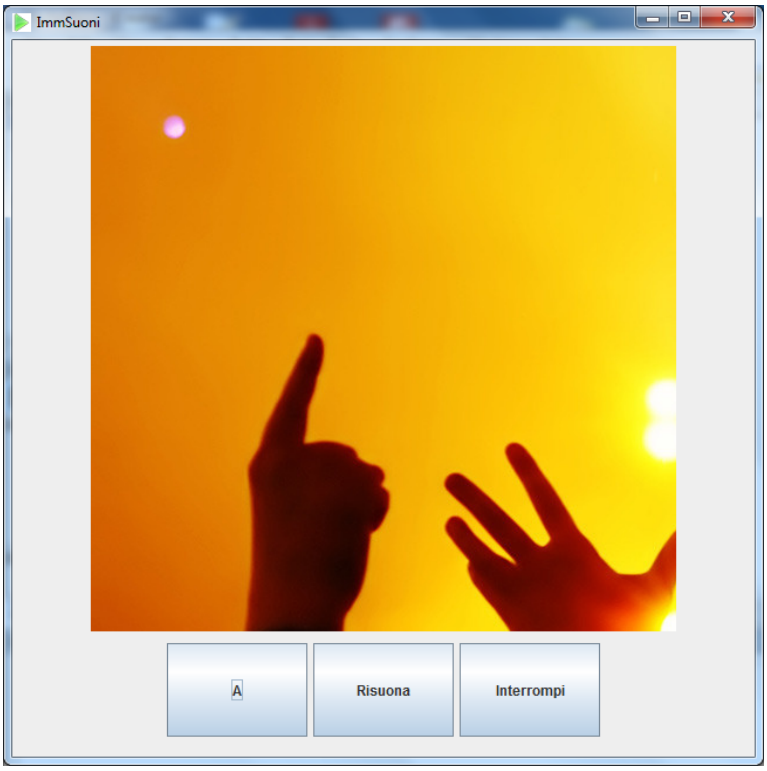
Listing esempio: JAR scaricabile

```
/**
 * ImmSuoni.java
 *
 * ImmSuoni application
 *
 * @author 4AI
 * @version 1.00 2015/6/4
 */
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.image.*;                    // per BufferedImage
import java.awt.event.*;
import javax.swing.*;
import javax.sound.sampled.*;
import javax.sound.midi.*;
import java.net.URL;                        // per URL
import javax.imageio.ImageIO;
import sun.audio.*;

public class ImmSuoni implements ActionListener {
  private JFrame f;
  private Container c;
  private JPanel p2;
  private MyPanel mio;
  long micPos;
  Sound sound;
  private JButton b, b1,b2;

  /** costruttore di default   */
  public ImmSuoni() {
            f = new JFrame("ImmSuoni");
            c = f.getContentPane();
            f.setSize(650,650);
            String s = "/imm/play1.png";
             Image ico = Toolkit.getDefaultToolkit().getImage(AppResources.getURL(s));    // anche  con   url =AppResources.getURL(s)
                                                                                          // Image ico = new ImageIcon(url).getImage();
             f.setIconImage(ico);                                                         // se non trova immagine, genera eccezione
            mio = new MyPanel();
            mio.setPreferredSize(new Dimension(500,500));
            p2 = new JPanel();
            b=new JButton("A");
            b.setPreferredSize(new Dimension(120,80));
            b.addActionListener(this);
            b1=new JButton("Risuona");
            b1.setPreferredSize(new Dimension(120,80));
            b1.addActionListener(this);
            b2=new JButton("Interrompi");
            b2.setPreferredSize(new Dimension(120,80));
            b2.addActionListener(this);
            sound=new Sound("cm.wav");
            p2.add(b);
            p2.add(b1);
            p2.add(b2);
            c.setLayout(new FlowLayout());
            c.add(mio);
            c.add(p2);
            f.setVisible(true);
             f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
            }
```

```
//class che gestisce i suoni
class Sound {
	String clipS;
	Clip clip;
	AudioInputStream audioIn;
	 /**
	  *  costruttore
	  *  @param clipS - stringa descrittiva del clip audio
	  */
	  public Sound(String clipS){
				this.clipS=clipS;
				set();
	  }
	   /** metodo che gestisce gli audio  */
	   public void set() {
				try{
					URL url = getClass().getClassLoader().getResource(clipS);
					audioIn = AudioSystem.getAudioInputStream(url);
					// Get a sound clip resource.
					clip = AudioSystem.getClip();
					// Open audio clip and load samples from the audio input stream.
					clip.open(audioIn);
					}catch(Exception e){e.printStackTrace();}
				}
	   }
	   /** metodo che fa suonare i file audio     */
	   public void playSound(){
				sound.clip.start();
				sound.clip.setMicrosecondPosition(0);
	   }
	   /**
	    *   metodo eseguito in risposta all'evento
	    *   @param e - messaggio evento
	    */
	    public void actionPerformed(ActionEvent e) {
		JButton o = (JButton)e.getSource();
		if(o==b)
			playSound();
		if(o==b1){    //re-play
			resumeTrack();
			}
		if(o==b2){   //stop
			stopTrack();
			}
	    }
	    /**   metodo che interrompe il file audio     */
	    public void stopTrack()  {
		try{
			micPos=sound.clip.getMicrosecondPosition();
			sound.clip.stop();
		    }catch(Exception e){}
	    }
	    /**   metodo che riprende l'esecuzione del file audio */
	    public void resumeTrack() {
				sound.clip.setMicrosecondPosition(micPos);
				sound.clip.start();
	    }
	    public static void main(String args[])  {
		SwingUtilities.invokeLater(new Runnable(){ public void run(){
					new ImmSuoni();
				}
		});
	    }
}
```




```
/// classe per recuperare URL delle immagini
class AppResources {
   public AppResources() {}

   public static URL getURL(String name) {
          URL url = AppResources.class.getResource(name);
      return url;
   }
}

class MyPanel extends JPanel {  //classe pannello personalizzato

   private BufferedImage img;

   /** costruttore di default   */
   public MyPanel() {
          try{
                 String s ="/imm/"+0+".png";
                 img= ImageIO.read(AppResources.getURL(s)); // caricamento dinamico per creare jar eseguibile
          }catch (IOException ex) {  }
   }
   /**
    *  metodo di disegno
    *   @param g - il contesto grafico
    */
   public void paintComponent(Graphics g){
                 super.paintComponent(g);
                 g.drawImage(img, 0, 0, this);          /* Immagine, posizione nel pannello (x,y)
                                                         * e un oggetto
                                                         * a cui notificare l'avvenuto caricamento
                                                         */
   }
}
```



Doppio click sul file jar *distribuito*