# LE TABELLE

Le liste permettono di avere una sequenza di righe

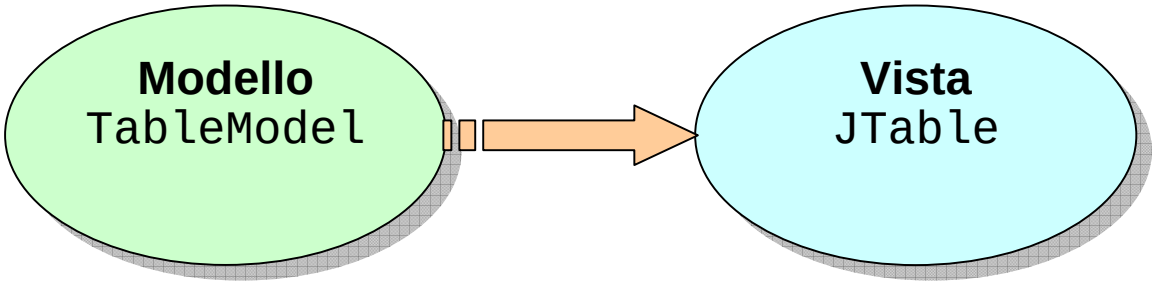
In alcune applicazioni è però necessario visualizzare le informazioni anche in colonne

| **Nome** | **Cognome** | **Indirizzo** | **Telefono** |
|---|---|---|---|
| Mario | Bianchi | Via Roma, 12 | 059/1111111 |
| Franco | Rossi | Via Milano, 33 | 059/2222222 |

La libreria swing mette a disposizione un componente per la rappresentazione di informazioni in forma tabellare, implementato dalla classe `JTable`

Ogni tabella implementata con `JTable` recupera i dati da rappresentare tramite un modello, istanza di una classe che implementa l'interfaccia `TableModel`

Questa interfaccia mette a disposizione metodi per sapere quante righe/colonne servono, qual è il valore di ogni singola cella, se le celle sono editabili, ecc.

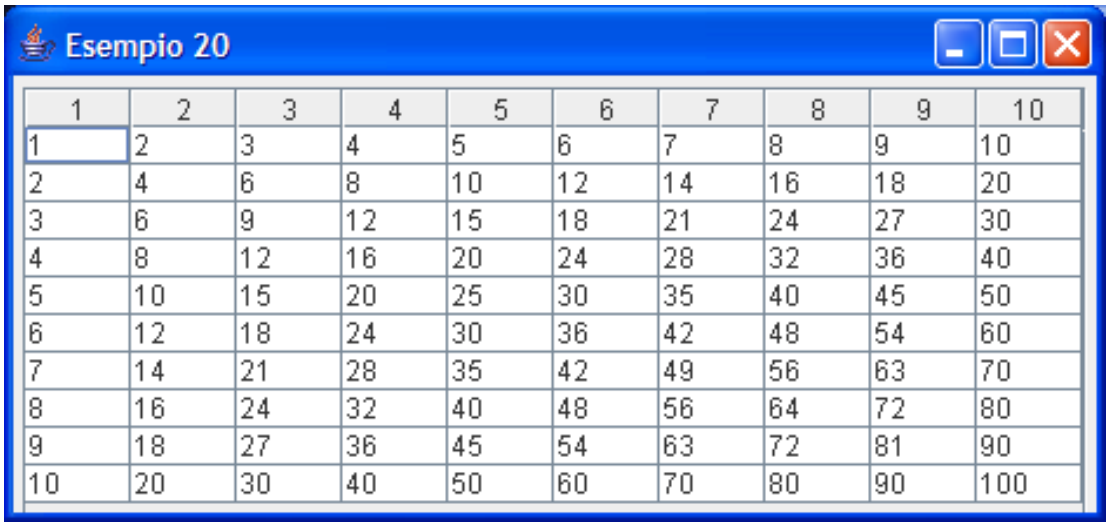



È disponibile anche una classe astratta, `AbstractTableModel`, che implementa la maggior parte dei metodi di `TableModel`, e lascia al programmatore da implementare solo i tre metodi:

```
public int getRowCount();
public int getColumnCount();
public Object getValueAt(int row, int column);
```

# ESEMPIO 20: LE TABELLINE

Si vuole implementare la visualizzazione delle classiche tabelline che vengono imparate a memoria alle scuole elementari




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

È quindi necessario:

1) Definire il modello dei dati
2) Definire la vista dei dati

**Modello dei dati**

Il modello è dato da una classe che implementa `TableModel` (o meglio, estende `AbstractTableModel`)

Il contenuto di ogni cella è il **prodotto** dell'indice di riga per l'indice di colonna. Siccome gli indici partono da 0, dovremo incrementarli entrambi.

Il modello definisce anche l'**intestazione** delle colonne

**Vista dei dati**

La tabella che mostra i dati è una istanza di `JTable`, che viene inizializzata con una istanza della classe precedente.

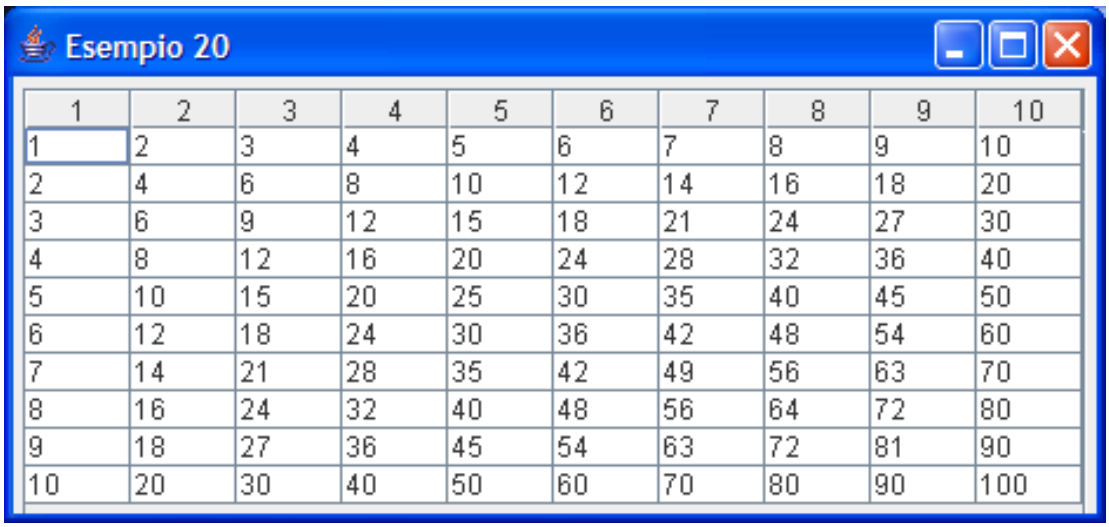
# ESEMPIO 20: IL MODELLO DI DATI

```
import javax.swing.table.AbstractTableModel;

public class MyTableModel extends
AbstractTableModel {

    // ritorna il numero di colonne
    public int getColumnCount() { return 10; }

    // ritorna il numero di righe
    public int getRowCount() { return 10;}

    // ritorna il contenuto di una cella
    public Object getValueAt(int row, int col)
    {
        // ritorna il prodotto
        return new Integer((row+1)*(col+1));
    }

    // ritorna il nome della colonna
    public String getColumnName(int col) {
        // e' il numero di colonna
        return Integer.toString(col+1);
    }

    // specifica se le celle sono editabili
    public boolean isCellEditable(int row, int col)
    {
        // nessuna cella editabile
        return false;
    }
}
```

# ESEMPIO 20: LA TABELLA

```
import javax.swing.*;

public class Es20Panel extends JPanel {
  public Es20Panel() {
    // crea il modello di dati
    TableModel dataModel = new MyTableModel();
    // crea la tabella
    JTable t = new JTable(dataModel);
    // aggiunge la tabella ad uno ScollPane
    JScrollPane scrollpane = new JScrollPane(t);
    // aggiunge lo ScrollPane al pannello
    add(scrollpane);
  }
}
```

Il pannello viene poi inserito in un JFrame come al solito




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

# EVENTI DI TABELLA

Naturalmente, ogni azione dell'utente sulla tabella genera un evento che può essere ascoltato da un ascoltatore
La classe `JTable` stessa implementa diversi ascoltatori

- `CellEditorListener`
- `TableModelListener`
- `ListSelectionListener`
- `TableColumnModelListener`

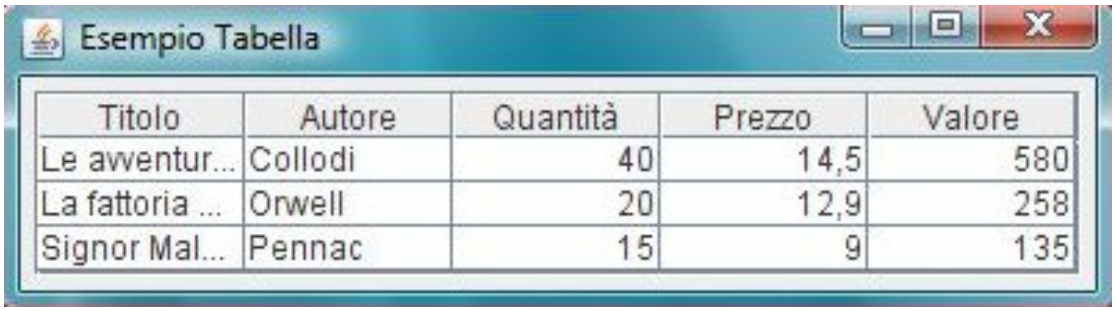
Non è sempre necessario implementare i metodi degli ascoltatori, perché la tabella rimane una vista dei dati contenuti nel modello: il programma fa quindi riferimento a quest'ultimo

Si veda la documentazione Java per un approfondimento

È invece importante riportare nel modello le modifiche fatte dall'utente ai valori nella tabella, come spiegato nell'esempio seguente

# ESEMPIO 21: UNA LISTA DI LIBRI

Vogliamo visualizzare una lista di libri, ognuno con titolo, autore, quantità, prezzo e valore totale, dato dal prezzo moltiplicato per la quantità
Diamo la possibilità all'utente di modificare quantità e prezzo




Ogni libro è rappresentato da una istanza della classe Book

```
public class Book {
    public String title; // titolo
    public String author; // autore
    public int quantity; // quantita'
    public float price; // prezzo

    // costruttore
    public Book(String title, String author, int
quantity, float price) {
        this.title = title;
        this.author = author;
        this.quantity = quantity;
        this.price = price;
    }
}
```

# ESEMPIO 21: IL MODELLO DI DATI

```
import java.util.Vector;
import javax.swing.table.AbstractTableModel;

public class VectorTableModel extends
 AbstractTableModel {
 Vector v = null;
 // intestazioni delle colonne
 String[] ColName = {"Titolo", "Autore",
 "Quantità", "Prezzo", "Valore" };

 public VectorTableModel(Vector v) {
   this.v = v; // inizializzato con il vettore
 }
 /** il numero di colonne */
  public int getColumnCount()
 { return ColName.length; }

  /** numero righe = dimensione del vettore */
  public int getRowCount() { return v.size(); }

  /** ritorna il contenuto di una cella */
  public Object getValueAt(int row, int col) {
        // seleziona il libro
        Book b = (Book)v.elementAt(row);
        // la stringa corrispondente alla colonna
        switch (col){
            case 0: return b.title;
            case 1: return b.author;
            case 2: return b.quantity;
            case 3: return b.price;
            case 4: return b.price * b.quantity;
            default: return "";
        }
  }
```

# ESEMPIO 21: IL MODELLO DI DATI - segue

```
  /**  ritorna il nome della colonna */
  public String getColumnName(int col) {
    return ColName[col];
  }

  /** ritorna il tipo dei valori
    * serve per allineare correttamente i numeri */
  public Class getColumnClass(int col) {
       return getValueAt(0, col).getClass();
  }
```

A differenza dell'esempio precedente, dobbiamo dire alla tabella che le colonne di quantità e prezzo sono editabili: il metodo `isCellEditable()` ritornerà quindi un valore diverso a seconda del numero di colonna.

```
  /** specifica se le celle sono editabili */
    public boolean isCellEditable(int row, int col)
    {
        if ((col == 2) || (col == 3))
            // solo la quantita' e il prezzo
            // sono modificabili
            return true;
        else
            // nessuna altra cella editabile
            return false;
    }
```

# ESEMPIO21: IL MODELLO DI DATI - segue

Per modificare i valori del modello a fronte di una modifica dell'utente sulla vista, è necessario implementare il metodo `setValueAt()`, che viene invocato dalla tabella ogni volta che l'utente modifica un valore in una cella

Conoscendo la riga della cella modificata, possiamo recuperare il libro interessato; conoscendo la colonna, possiamo sapere quale attributo è stato modificato
Dopodiché informiamo la tabella delle modifiche inviando un evento tramite il metodo `fireTableDataChanged()`

```
/** metodo per gestire le modifiche dell'utente */
    public void setValueAt(Object value, int row,
int col) {
        Book b = (Book)v.elementAt(row);
        if (col == 2)
           // modifica la quantita'
           b.quantity =
((Integer)value).intValue();
        if (col == 3)
           // modifica il prezzo
           b.price = ((Float)value).floatValue();
        // notifica il cambiamento
        fireTableDataChanged();
    }
}
```

si noti che va aggiornata la vista non solo della cella modificata dall'utente, ma anche del campo "Valore" che viene calcolato in base a quantità e prezzo

# ESEMPIO 21: LA TABELLA

```
import java.util.Vector;
import javax.swing.*;
import javax.swing.table.TableModel;

public class PannelloTabellaVettore extends JPanel
{
    public PannelloTabellaVettore () {
        // predispone il vettore
        Vector v = new Vector(3);
        Book b1 = new Book("Le avventure di
Pinocchio", "Collodi", 40, 14.50F);
        Book b2 = new Book("La fattoria degli
animali", "Orwell", 20, 12.90F);
        Book b3 = new Book("Signor Malaussene",
"Pennac", 15, 9.00F);
        v.add(b1);
        v.add(b2);
        v.add(b3);

        // crea il modello di dati dal vettore
        TableModel dataModel = new
VectorTableModel(v);
        // crea la tabella
        JTable t = new JTable(dataModel);
        // imposta la dimensione di visualizzazione
        t.setPreferredScrollableViewportSize(
                            t.getPreferredSize());
        // aggiunge la tabella ad uno ScollPane
      JScrollPane scrollpane = new JScrollPane(t);
    // aggiunge lo ScrollPane al pannello
        add(scrollpane);
    }
}
```

Il pannello viene poi inserito in un JFrame come al solito