

# A Simple JTable Example for Display

---

 [codejava.net/java-se/swing/a-simple-jtable-example-for-display](http://codejava.net/java-se/swing/a-simple-jtable-example-for-display)

```
1  package net.codejava.swing;
2  import javax.swing.JFrame;
3  import javax.swing.JScrollPane;
4  import javax.swing.JTable;
5  import javax.swing.SwingUtilities;
6  public class TableExample extends JFrame
7  {
8      public TableExample()
9      {
10         String[] columns = new String[] {
11             "Id" , "Name" , "Hourly Rate" , "Part Time"
12         };
13         Object[][] data = new Object[][] {
14             { 1 , "John" , 40.0 , false },
15             { 2 , "Rambo" , 70.0 , false },
16             { 3 , "Zorro" , 60.0 , true },
17         };
18         JTable table = new JTable(data, columns);
19         this.add( new JScrollPane(table));
20         this.setTitle( "Table Example" );
21         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22         this.pack();
23         this.setVisible( true );
24     }
25     public static void main(String[] args)
26     {
27         SwingUtilities.invokeLater( new Runnable() {
28             @Override
29             public void run() {
30                 new TableExample();
31             }
32         })
33     }
```

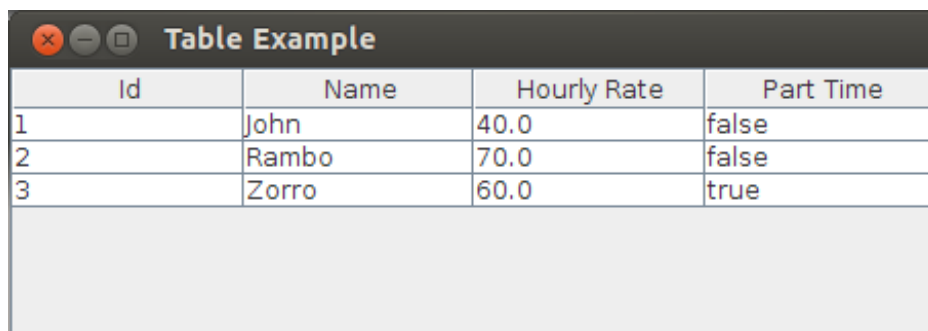
```
32     });  
33 }  
34 }  
35  
36  
37  
38  
39  
40  
41  
42
```

This code attempts to build a table with minimum amount of effort. First of all, the column headers are identified and declared in a String array columns. Next, a 2d Object array named data is declared. Each inner array corresponds to a row of data. Within each array, the columns are separated by commas.

Note that the column values are declared based on their type. For example, the id holds int values, name holds string values, hourly rate holds double values and part-time status holds boolean values. This is important as it allows us to represent the actual values and not just as string values for all.

Coming back to the code, a JTable instance is then created by using the constructor that takes the data and column arrays. Rest of the code is about building the JFrame and adding the JTable instance to the JFrame (the JTable instance is wrapped in a JScrollPane instance to enable scrolling).

When the code is run, we get the following output:



Id	Name	Hourly Rate	Part Time
1	John	40.0	false
2	Rambo	70.0	false
3	Zorro	60.0	true

Initial Display

**Issues with this Program:**

There are a few problems with this program. Firstly, the columns are not aligned properly. All the columns are left-aligned which is not we want. We want the numbers to be right-aligned. Secondly, when you run the program, you can see that the table allows the user to edit the fields. Our idea was to build a table that is for display-only.

Let us see how to correct these issues.

### Customizing the Table Model:

Let us digress a bit to understand how JTable works. All components in the Swing API are built on the MVC pattern. Here, the JTable is the component which provides the view. The model is provided by an interface named TableModel. A default implementation is provided by the Swing API which is named as DefaultTableModel. This is internally used by JTable when we do not provide anything. This is exactly what happened in the above code.

To achieve our objectives, we can subclass this DefaultTableModel and customize it a little bit as the following code snippet shows:

```
1  final Class[] columnClass = new Class[] {
2  Integer.class, String.class, Double.class, Boolean.class
3  };
4  DefaultTableModel model = new DefaultTableModel(data, columns) {
5  @Override
6  public boolean isCellEditable( int row, int column)
7  {
8  return false ;
9  }
10 @Override
11 public Class<?> getColumnClass( int columnIndex)
12 {
13 return columnClass[columnIndex];
14 }
15 };
16 JTable table = new JTable(model);
17
```

We are now using a model to populate the data. This code snippet first extends the DefaultTableModel class to create an anonymous instance. This is needed as we are overriding 2 methods:

One is the isCellEditable() method from which we return false. This simply indicates the table is not editable. This is a simple change.

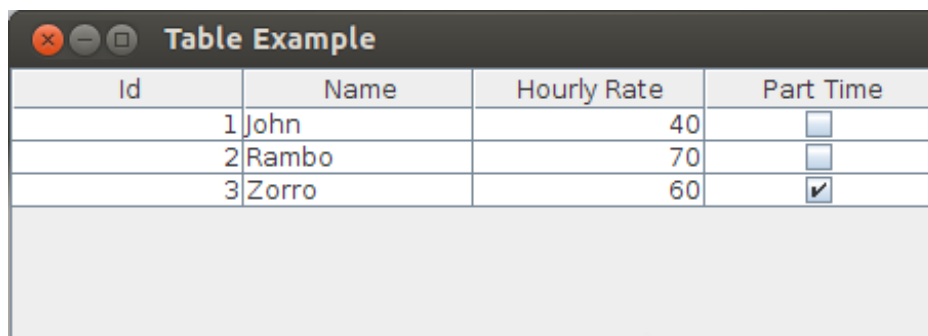
The next method is the `getColumnClass()` method. The `columnIndex` is passed to this method. We need to indicate to the table which class the data is represented by. As we know, the `String` values are represented by the `java.lang.String` class. Therefore, we just indicate the types via the class.

This information is held in an array variable named `columnClass`. The class types are declared corresponding to the data values that they are supposed to display. For example, the first value is `Integer.class`. This is because the first column is 'id' which is supposed to display integer values. The same way, the third column 'hourly rate' would display decimal values and so the type is declared as `Double`.

Finally, the `JTable` is created using the constructor that takes an instance of the `TableModel`. In this case, we pass the model variable which we created as an anonymous instance.

### Output:

When we run the program with this change, we get the following output:



Id	Name	Hourly Rate	Part Time
1	John	40	<input type="checkbox"/>
2	Rambo	70	<input type="checkbox"/>
3	Zorro	60	<input checked="" type="checkbox"/>

### Output

Let us examine the output. The id and hourly rate values are displayed as right-aligned. The name continues to be displayed as left-aligned. This is exactly what we wanted. Also, when you run the program, you can see that we are not able to edit the values in the table.

### Default Rendering:

But wait. There's more. The last column 'Part Time' displays a check-box!

And the box is checked for the third row as the corresponding data for that row is true.

So, how did this happen? Since we declared the last column type to be of `Boolean`, `JTable` automatically displays the boolean values as a check box. This is called rendering. In this case, the `JTable` provides this out-of-the-box which is really handy.

### Source Code:

Let us provide the full source-code here:

```
1 package net.codejava.swing;  
2 import javax.swing.JFrame;  
3 import javax.swing.JScrollPane;
```

```

4  import javax.swing.JTable;
5  import javax.swing.SwingUtilities;
6  import javax.swing.table.DefaultTableModel;
7  public class TableExample extends JFrame
8  {
9      public TableExample()
10     {
11         String[] columns = new String[] {
12             "Id" , "Name" , "Hourly Rate" , "Part Time"
13         };
14         Object[][] data = new Object[][] {
15             { 1 , "John" , 40.0 , false },
16             { 2 , "Rambo" , 70.0 , false },
17             { 3 , "Zorro" , 60.0 , true },
18         };
19         final Class[] columnClass = new Class[] {
20             Integer. class , String. class , Double. class , Boolean. class
21         };
22         DefaultTableModel model = new DefaultTableModel(data, columns) {
23             @Override
24             public boolean isCellEditable( int row, int column)
25             {
26                 return false ;
27             }
28             @Override
29             public Class<?> getColumnClass( int columnIndex)
30             {
31                 return columnClass[columnIndex];
32             }
33         };
34         JTable table = new JTable(model);
35         this .add( new JScrollPane(table));
36         this .setTitle( "Table Example" );
37         this .setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38         this .pack();

```

```
39     this.setVisible( true );
40 }
41 public static void main(String[] args)
42 {
43     SwingUtilities.invokeLater( new Runnable() {
44         @Override
45         public void run() {
46             new TableExample();
47         }
48     });
49 }
50 }
51
52
53
54
55
56
57
58
59
60
```