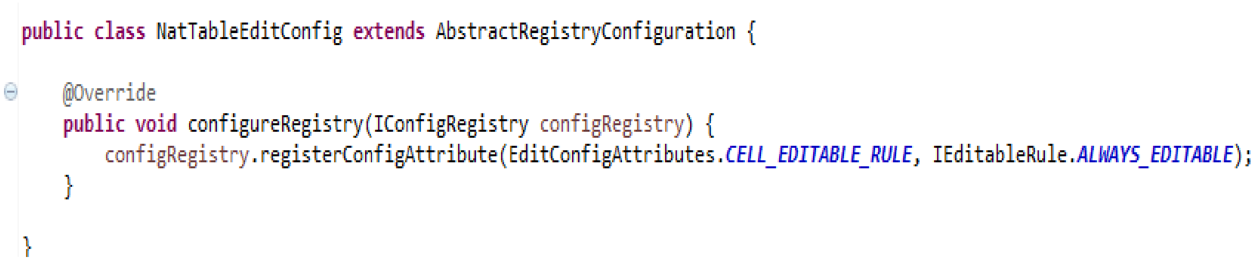
Adding Editor Configuration to edit cell in the NatTable:

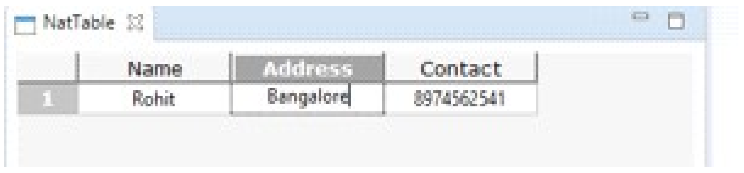
Now for simple text editing purposes only a EditConfigAttributes.CELL_EDITABLE_RULE config attribute has to be registered to the IConfigRegistry.

```
  natTable.addConfiguration(new NatTableEditConfig());

  natTable.configure();
```



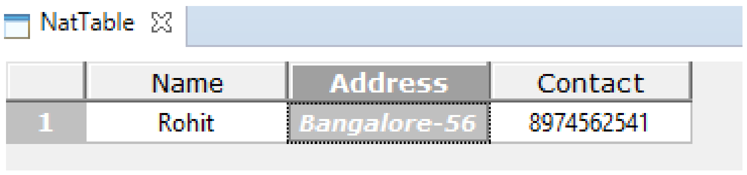
```
public class NatTableEditConfig extends AbstractRegistryConfiguration {

    @Override
    public void configureRegistry(IConfigRegistry configRegistry) {
        configRegistry.registerConfigAttribute(EditConfigAttributes.CELL_EDITABLE_RULE, IEditableRule.ALWAYS_EDITABLE);
    }

}
```

Run the Application :



NatTable

	Name	Address	Contact
1	Rohit	Bangalore	8974562541



NatTable

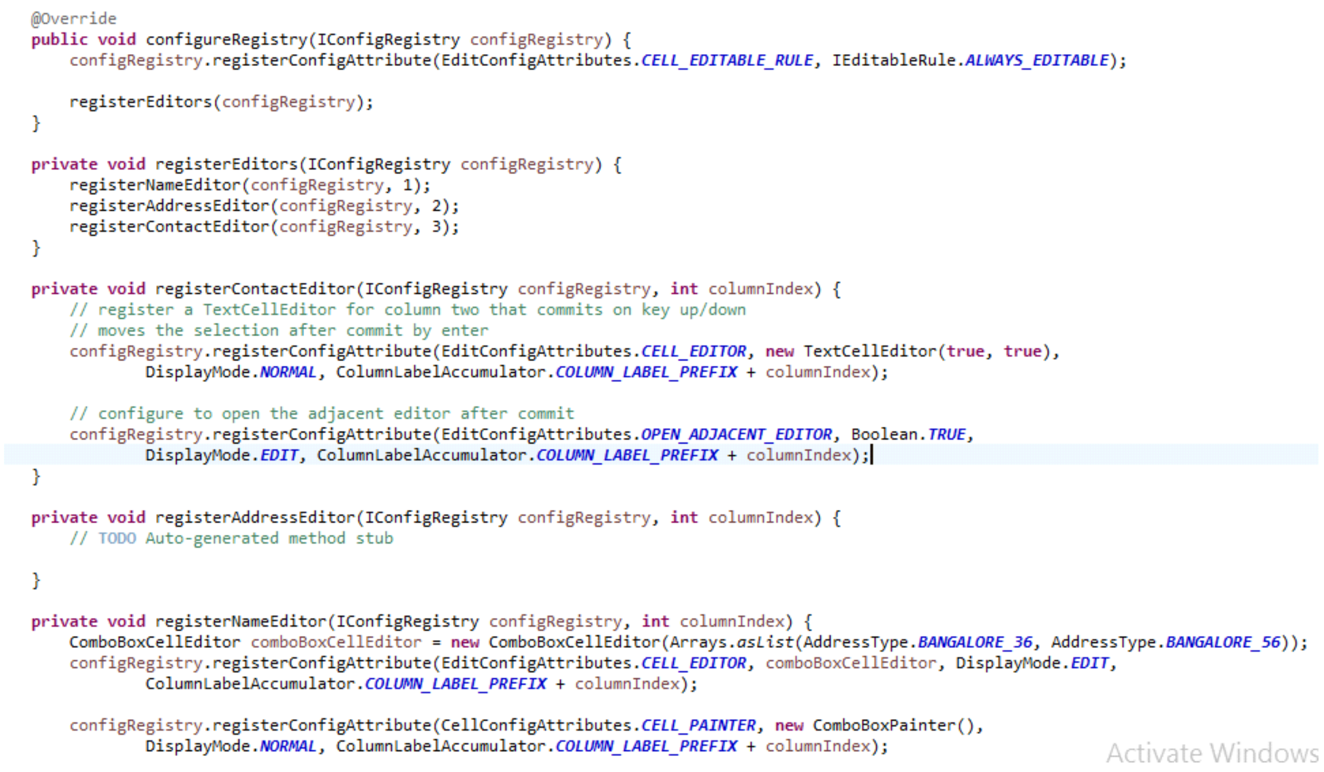
	Name	Address	Contact
1	Rohit	Bangalore-56	8974562541

Add more advanced editing support to a HYPERLINK

In this exercise editing support will be provided for a table, which contains different data than only strings. So different cell editors have to be applied for different columns.

For the `SimpleEditor` an anonymous inner class of an `AbstractRegistryConfiguration` was used to apply basic editing capabilities.

Now that the configuration becomes more complex it is better to encapsulate it in another class:



```java
@Override
public void configureRegistry(IConfigRegistry configRegistry) {
    configRegistry.registerConfigAttribute(EditConfigAttributes.CELL_EDITABLE_RULE, IEditableRule.ALWAYS_EDITABLE);

    registerEditors(configRegistry);
}

private void registerEditors(IConfigRegistry configRegistry) {
    registerNameEditor(configRegistry, 1);
    registerAddressEditor(configRegistry, 2);
    registerContactEditor(configRegistry, 3);
}

private void registerContactEditor(IConfigRegistry configRegistry, int columnIndex) {
    // register a TextCellEditor for column two that commits on key up/down
    // moves the selection after commit by enter
    configRegistry.registerConfigAttribute(EditConfigAttributes.CELL_EDITOR, new TextCellEditor(true, true),
            DisplayMode.NORMAL, ColumnLabelAccumulator.COLUMN_LABEL_PREFIX + columnIndex);

    // configure to open the adjacent editor after commit
    configRegistry.registerConfigAttribute(EditConfigAttributes.OPEN_ADJACENT_EDITOR, Boolean.TRUE,
            DisplayMode.EDIT, ColumnLabelAccumulator.COLUMN_LABEL_PREFIX + columnIndex);
}

private void registerAddressEditor(IConfigRegistry configRegistry, int columnIndex) {
    // TODO Auto-generated method stub

}

private void registerNameEditor(IConfigRegistry configRegistry, int columnIndex) {
    ComboBoxCellEditor comboBoxCellEditor = new ComboBoxCellEditor(Arrays.asList(AddressType.BANGALORE_36, AddressType.BANGALORE_56));
    configRegistry.registerConfigAttribute(EditConfigAttributes.CELL_EDITOR, comboBoxCellEditor, DisplayMode.EDIT,
            ColumnLabelAccumulator.COLUMN_LABEL_PREFIX + columnIndex);

    configRegistry.registerConfigAttribute(CellConfigAttributes.CELL_PAINTER, new ComboBoxPainter(),
            DisplayMode.NORMAL, ColumnLabelAccumulator.COLUMN_LABEL_PREFIX + columnIndex);
```

ABOUT ANCIT:

ANCIT Consulting is an Eclipse Consulting Firm located in the "Silicon Valley of Outsourcing", Bangalore. Offers professional Eclipse Support and Training for various Eclipse based Frameworks including RCP, EMF, GEF, GMF. Contact us on annamalai@ancitconsulting.com to learn more about our services.