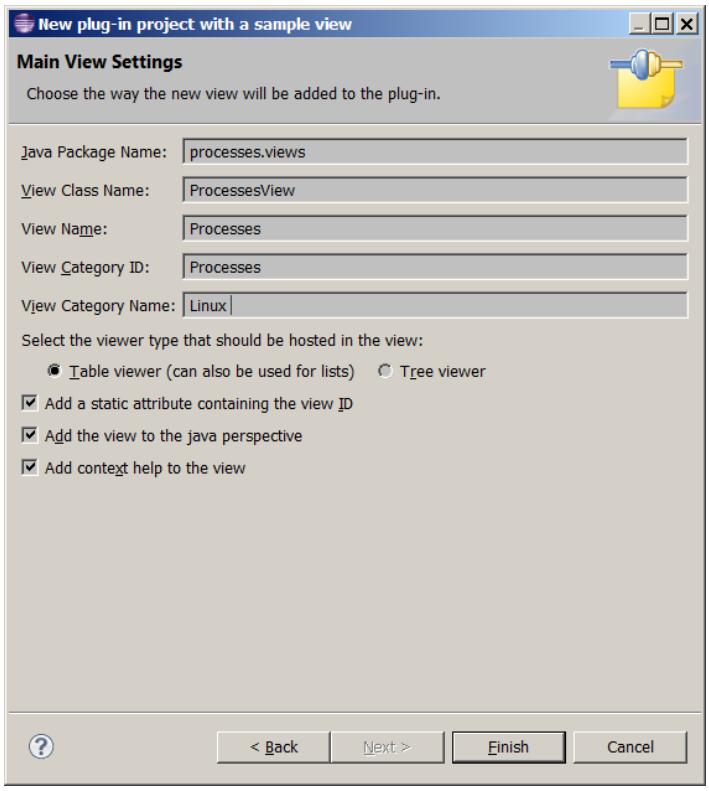
Create the Process view plug-in

1. Select File → New → Project… from the menu
2. Select Plug-In Development -> Plug-In Project
3. Enter Processes as the project name
4. Navigate to 'Templates' page of the wizard (Next button twice)
5. Select 'Plug-in with a view -> Next
6. Fill the name of the view, the class name and the category as below.



7. Finish
   ⇨ Project is created in Package Explorer view
8. Run/Debug the generated plug-in as an Eclipse Application and check the names you have provided above
9. Edit ProcessesView class as follows:
   a. Rename member action1 to refreshAction
   b. Rename member action2 to attachAction
   c. Remove member doubleClickAction and all its references
   d. Add a new member systemCallsAction similar to refresh and attach actions
   e. Add the creation and handling code for systemCallsAction so that we obtain a similar action with attach and refresh ones
   f. Update display text and tooltips of actions to the following values:

i. Refresh, Refresh the List of Processes
ii. Attach, Attach to Running Process
iii. Capture System Calls, Capture System Calls of Running Process

g. Create the below methods for the corresponding actions and call them on run() method of every action

```
private void onAttachAction() { }
private void onRefreshAction() { }
private void onSystemCallsAction() { }
```

10. Run/Debug the generated plug-in as an Eclipse Application and check the updates made above
11. Add ProcessesTableViewer.java file to the project from HelperClasses.zip archive
12. Edit ProcessesView class as follows:
    a. In createPartControl() method create a ProcessesTableViewer object instead of a TableViewer

13. Run/Debug the generated plug-in as an Eclipse Application and check the view now has three columns
14. Edit ProcessesView class as follows:
    a. Create a class that represents the process information:

```
static class ProcessData {
	String pid = "";
	String command = "";
	String pc = "";
}
```

    b. Create an array member that holds the process information:
    List<ProcessData> processes = new ArrayList<ProcessData>();
    c. Add implementation for ViewContentProvider. getElements() method:

```
return processes.toArray();
```

    d. Add implementation for ViewLabelProvider. getColumnText() method:

```
if (obj instanceof ProcessData) {
	ProcessData data = (ProcessData)obj;
	switch(index) {
		case 0:
			return data.pid;
		case 1:
			return data.command;
		case 2:
			return data.pc;
	}
}
```

    e. Add implementation for ViewLabelProvider. getColumnText() method:

15. Add LinuxInteractor.java file to the project from HelperClasses.zip archive
16. Add a constant for refresh command:

```
private static final String REFRESH_COMMAND = "ps ax -o pid= -o comm=";
```

17. Add the following implementation for onRefreshAction method:

```
private void onRefreshAction() {
	int sel = viewer.getTable().getSelectionIndex();
	processes.clear();
	String response = LinuxInteractor.executeCommand(REFRESH_COMMAND, true);
	BufferedReader reader = new BufferedReader(new StringReader(response));
	try {
		String line = reader.readLine();
		while (line != null && !line.isEmpty())  {
			ProcessData data = new ProcessData();
			String[] attrs = line.trim().split(" ");
			if (attrs.length > 1)
				data.command = attrs[1];
			if (attrs.length > 0)
				data.pid = attrs[0];

			if (!data.pid.isEmpty())
				processes.add(data);

			line = reader.readLine();
		}

		reader.close();
	} catch (IOException e) {
	}

	viewer.refresh();
	viewer.getTable().setSelection(sel);
}
```

18. Inspect the code above focusing on the following:
    a. Selection index is saved and restored after the view refresh
    b. Output of the command is read line by line using a StringReader
    c. Every line returned is split in space delimited words: PID and Command
    d. New ProcessData object is created and pushed in the array
    e. Viewer is refreshed at the end
19. Call onRefreshAction() at the end of createPartControl() method in order to populate the view at startup
20. Run/Debug the generated plug-in as an Eclipse Application and check the list of processes
21. Add a compare() method for a sort by PID of the processes

```
// add sorter
viewer.setComparator(new ViewerSorter() {
	public int compare(Viewer viewer, Object o1, Object o2) {
		ProcessData pd1 = (ProcessData)o1;
```

```
                ProcessData pd2 = (ProcessData)o2;
                if (pd1.pid.isEmpty())
                        return -1;

                if (pd2.pid.isEmpty())
                        return -1;

                //TODO: add your code here

                return res;
        }
});
```

22. Run/Debug the generated plug-in as an Eclipse Application and check the list of processes
23. Add a constant for attach command:

```
private static final String ATTACH_COMMAND = "sudo ~/linux_tools/ptrace_attach %s";
```

24. Add the following implementation for onAttachAction method:

```
private void onAttachAction() {
        int sel = viewer.getTable().getSelectionIndex();

        TableItem[] selection = viewer.getTable().getSelection();
        if (selection.length > 0) {
                ProcessData pd = (ProcessData)selection[0].getData();
                String command = String.format(ATTACH_COMMAND, pd.pid);
                String response = LinuxInteractor.executeCommand(command, true);
                if (Platform.getOS().equals(Platform.WS_WIN32))
                        response = "4004b7"; //hardcoded

                String[] resp = response.split("\n");
                if (resp.length > 0)
                        pd.pc= resp[0].trim();
                else
                        pd.pc= response;
        }

        viewer.refresh();
        viewer.getTable().setSelection(sel);
}
```

25. Inspect the code above focusing on the following:
    a. Selection index is saved and restored after the view refresh
    b. First line of the response is the PC where the application is stopped
    c. ProcessData object from the selected element is updated
    d. View is refreshed
26. Run/Debug the generated plug-in as an Eclipse Application and check the attach to processes feature
27. Add a constant for system call command:

```
private static final String SYSTEM_CALL_COMMAND = "sudo strace -p %s 2> /tmp/%s";
```

28. Add the following implementation for onSystemCallsAction method:

```
private void onSystemCallsAction() {

        TableItem[] selection = viewer.getTable().getSelection();
        if (selection.length > 0) {
                ProcessData pd = (ProcessData)selection[0].getData();
                new ConsoleJob(pd).schedule();
        }
}
```

29. Inspect the code above focusing on the following:
    a. Selected object is determined and passed to a new processing job
30. Add the implementation for the processing job:

```
private class ConsoleJob extends Job {

        ProcessData pd;

        ConsoleJob (ProcessData pd) {
                super("Output job for process" + pd.command);
                this.pd = pd;
        }

        protected IStatus run(IProgressMonitor monitor) {
                try {
                        monitor.beginTask("Output job for process" + pd.command,
IProgressMonitor.UNKNOWN);

                        MessageConsole console = new MessageConsole(pd.command , null);
                        ConsolePlugin plugin = ConsolePlugin.getDefault();
                        IConsoleManager conMgr = plugin.getConsoleManager();
                        conMgr.addConsoles(new IConsole[] { console});
                        conMgr.showConsoleView(console);
                        MessageConsoleStream outStream = console.newMessageStream();

                        String command = String.format(SYSTEM_CALL_COMMAND, pd.pid,
pd.command);
                        ProcessBuilder pb = LinuxInteractor.postCommand(command);
                        Process shell = pb.start();

                        Thread.sleep(1000);
                        BufferedReader reader = new BufferedReader( new FileReader ("/tmp/"
+ pd.command));

                        while (true) {
                                String resp = reader.readLine();
                                if (resp != null) {
```

```
                                        outStream.println(resp);
                                        outStream.flush();
                                }
                                if (monitor.isCanceled()) {
                                        //Kill the process
                                        shell.destroy();
                                        reader.close();
                                        break;
                                } else {
                                        Thread.sleep(100);
                                }
                        }

                } catch (IOException e) {
                        e.printStackTrace();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                } finally {
                        monitor.done();
                }
                return Status.OK_STATUS;
        }
}
```

31. Solve compile errors by adding "org.eclipse.ui.console" as a required bundle
    a. Edit plugin.xml file
    b. Go to Dependencies tab and add "org.eclipse.ui.console" plugin
32. Inspect the code of the ConsoleJob focusing on the following:
    a. Implementation provides a run() method – will run on a separate thread
    b. Run method has a "IProgressMonitor monitor" parameter that is used to check if the user has cancelled the job
    c. Output of the system call command is read from a file
    d. IConsoleManager is used for accessing the console manager and creating a new console
    e. The console is written using MessageConsoleStream outStream
    f. The console job runs infinitely until the user cancels it
    g.
33. Run/Debug the generated plug-in as an Eclipse Application and check the system calls output
34. Exercise: add icons for actions and for processes
    a. Use the view 'Plug-in Image Browser' for finding icons
35. Exercise: export the plugin as JAR
    a. File → Export → Plug-in Development → Deployable plug-ins and fragments.
36. Exercise: export the plugin as update site