AJAX for Java Web Applications

Session: 4

AJAX Client Frameworks-II

Objectives

- Explain jMaki and its widgets
- Explain jQuery and its features
- Explain YUI and its features

jMaki 1-2

- jMaki is an open source, light weight client-server framework.
- jMaki originated in Kumamoto, Japan.
- The letter ‘j’ in jMaki represents JavaScript technology and Maki, which is a Japanese word, means ‘to wrap’.
- In other words, jMaki means JavaScript wrappers.

jMaki is used for creating AJAX applications by integrating JavaScript technology into the applications.

jMaki allows including styles and templates, widget model, and client services, such as event handling, in a client application.

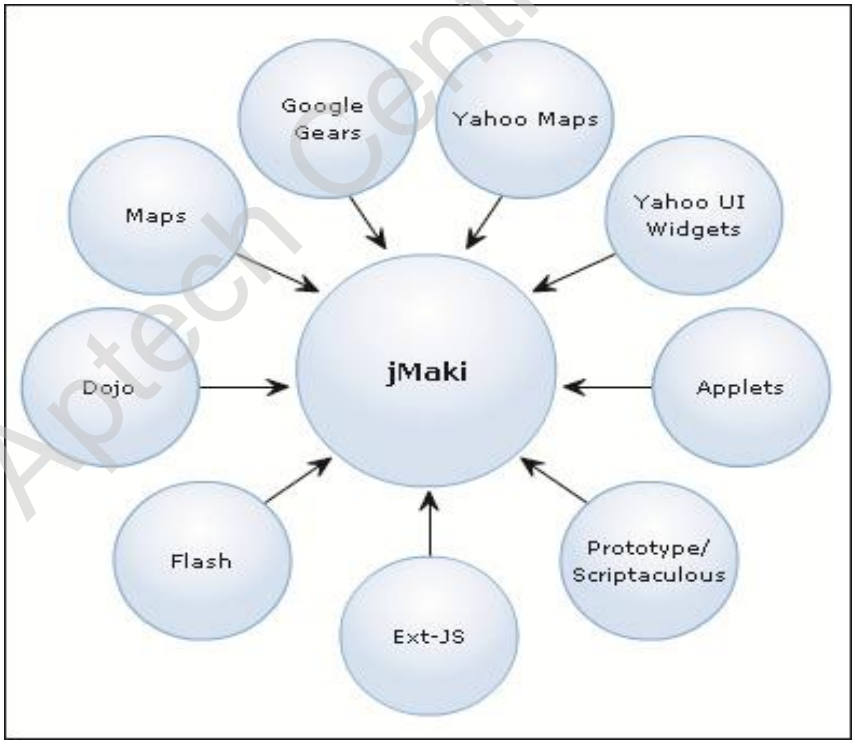
For server applications, jMaki provides server runtime component and a generic proxy, named XMLHttpProxy.

jMaki 2-2

XMLHttpProxy enables the server applications to interact with external Web services outside the application domain.

JMaki provides access to widgets from various toolkits as a JSP taglib or as a JSF component.

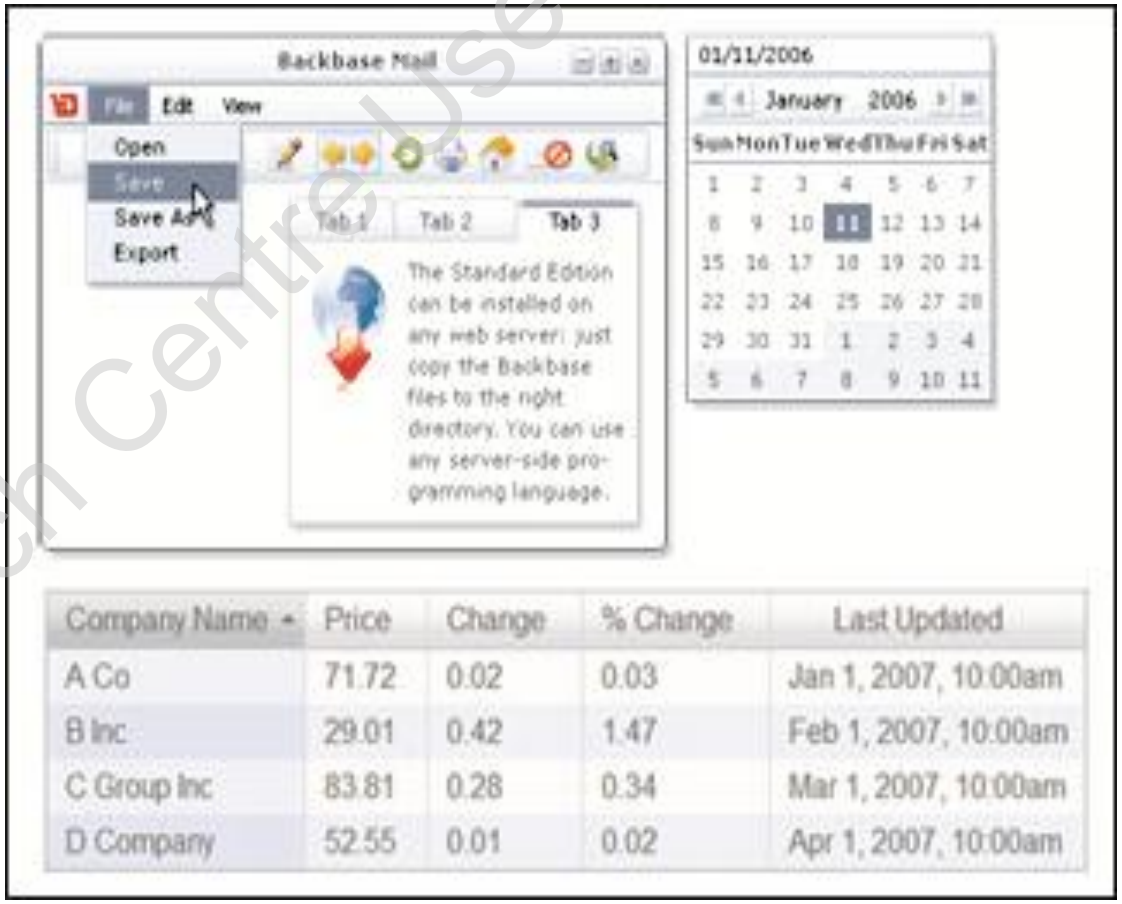
- Following figure shows the origin of jMaki:



Features of jMaki

- Wrapping of AJAX components in Tags
- Standardization of JavaScript Toolkit API
- Support of Multiple Server Technologies
- Preference for Convention over Configuration
- Provision of Standardized Event/Data Model

- Following figure shows the widgets provided by jMaki:



Client–Side Components 1-2

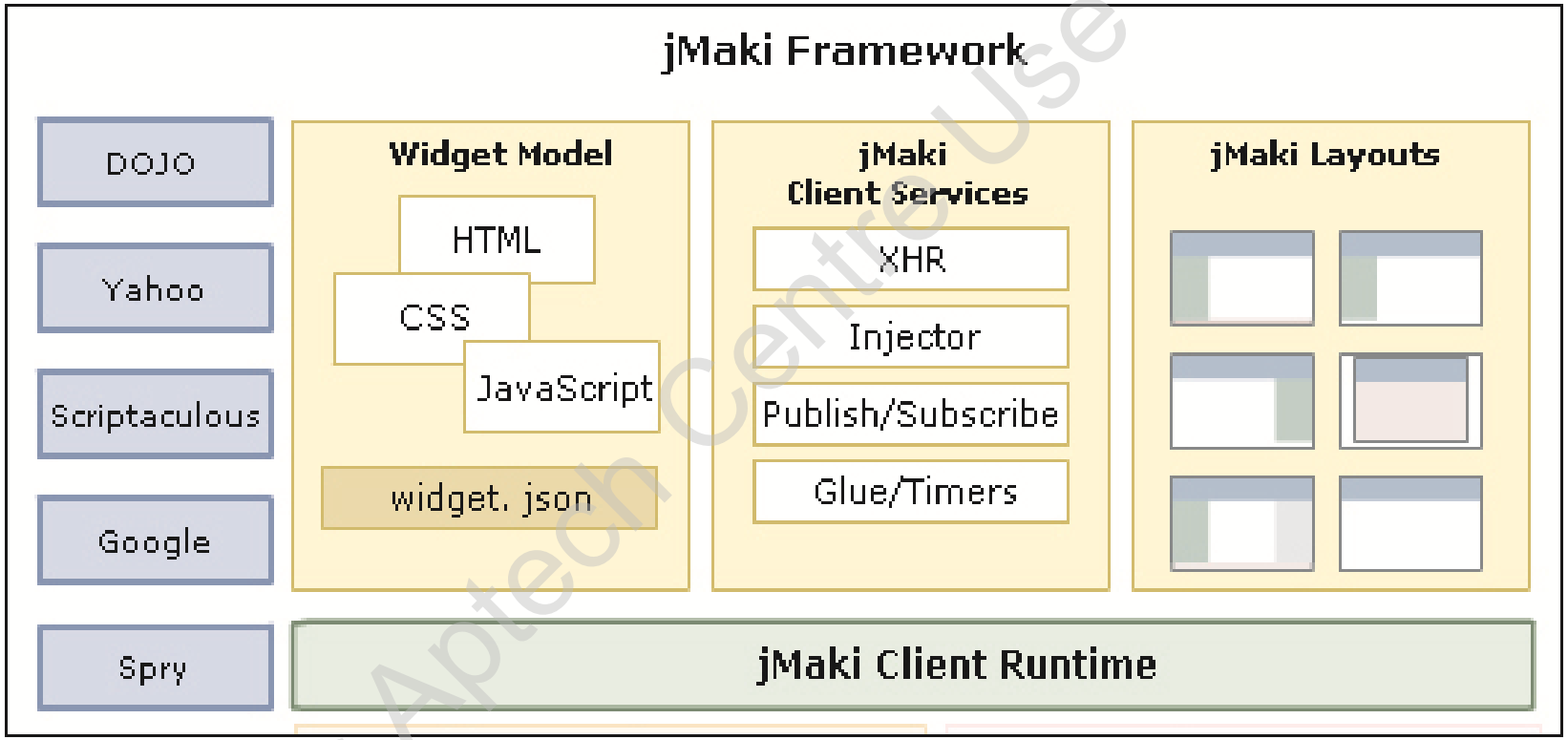
- jMaki framework comprises the client components and server components.

The client components that make up jMaki Architecture are as follows:

jMaki Layouts	• jMaki provides different layouts to help reduce efforts and time required to create/design the layout of a Web page. • jMaki uses HTML and CSS to create these layouts.
jMaki Client Runtime	• jMaki client runtime uses JavaScript. • It is responsible for bootstrapping the widgets and passing unique parameters provided by the server-side runtime to the widgets.
jMaki Client Services	• jMaki client services provide APIs to use XMLHttpRequest object that allows data transfer between the client and the server.
jMaki Widget Model	• jMaki widget model provides a component model for reusable JavaScript components. • The structure of Widgets is based on HTML, CSS, and JavaScript. • jMaki stores widget descriptions in widget.json format.

Client–Side Components 2-2

- Following figure shows the client-side components of jMaki framework:



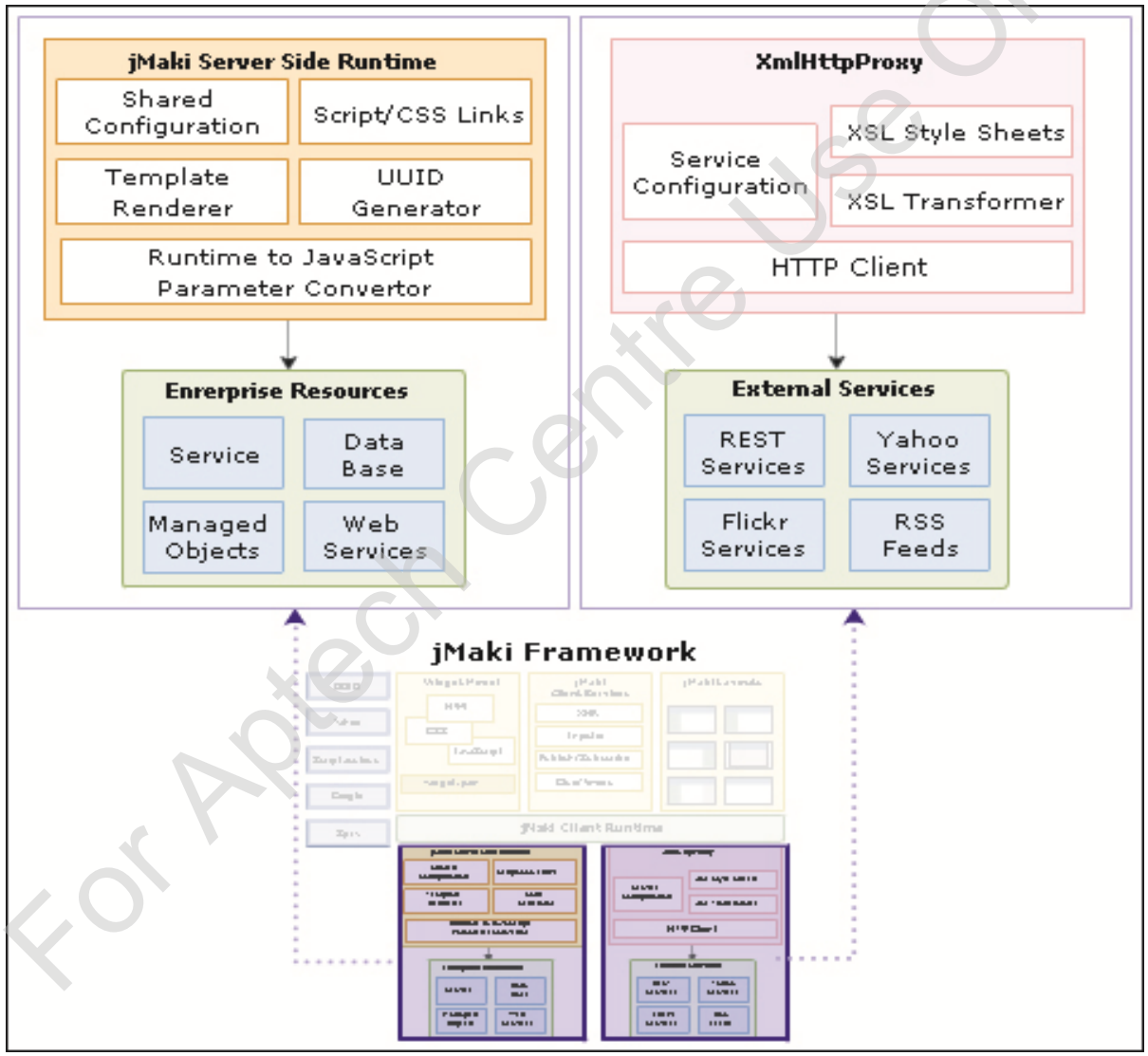
Server–Side Components 1-2

- The two server components that make up jMaki framework are as follows:

jMaki Server Runtime	• jMaki server runtime binds the server-side runtime with the jMaki client runtime. • It is also responsible for tracking and delivering the correct JavaScript, CSS, and HTML references based on the library being used.
XMLHttpProxy	• XMLHttpProxy allows widgets to access JSON or other external services, such as Flickr image searches. • Direct Communication takes place between the widgets and the services.

Server–Side Components 2-2

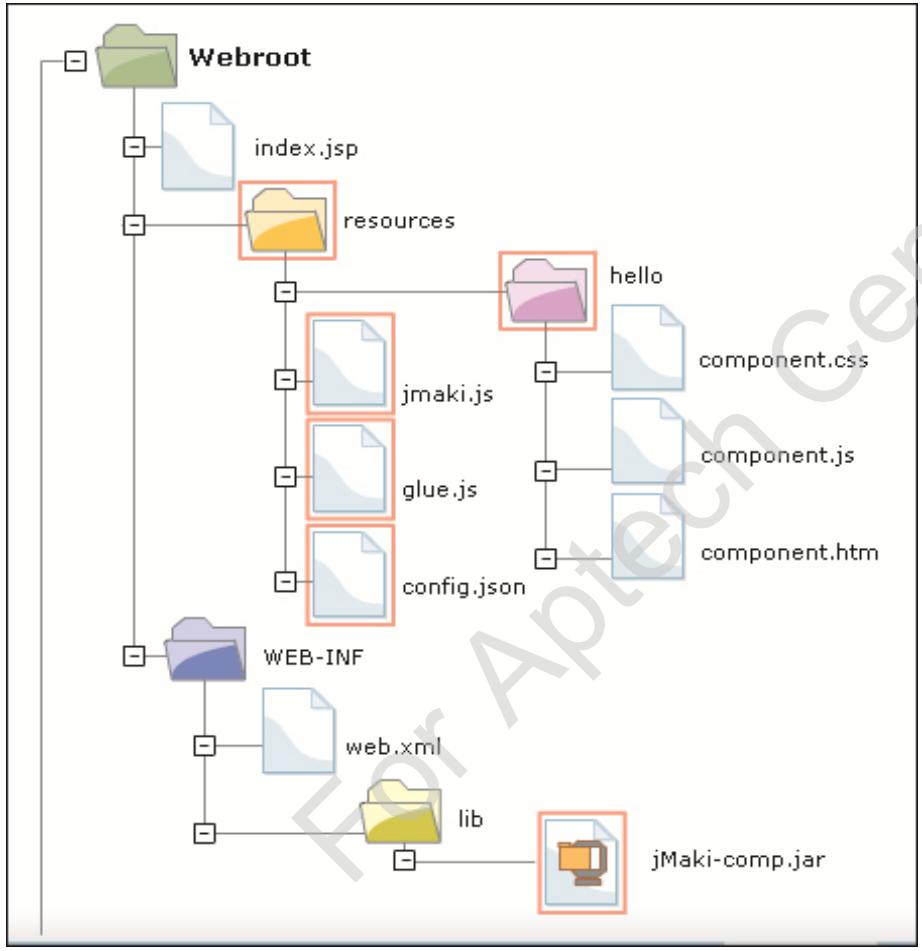
- Following figure shows the server-side components of jMaki framework:



Application Structure

- jMaki applications can range from simple applications containing few widgets to complex applications containing multiple jMaki widgets.

Following figure displays the directory structure of an application:



- **resources**: Contains all the resources used by a jMaki application.
- **jmaki.js**: JavaScript file that contains code for loading jMaki widgets and functions for widget communication.
- **config.json**: Contains theme information, extension mapping information, and glue mapping information for wiring widgets.
- **jmaki-comp.jar**: Contains the server runtime code is present in the /WEB-INF/lib directory.
- **hello**: A widget with its resources present in the /resources/hello directory.
- **glue.js**: Glues widgets together. It is used for registering and defining widget event listener, publishing events to a topic and subscribing to a topic.

Widget Model and Properties

- A jMaki widget is a reusable parameterized component.
- jMaki ensures that proper parameters are passed to a widget code to initialize the widget in a page.
- The name of a widget maps to a directory. In other words, a jMaki widget is a directory or a package where the widget resides.
- The directories are separated using 'dot' notation.

The directory which makes a widget comprises three core resource files as follows:

component.css

- This file defines the CSS styles for a widget when it is displayed.
- It contains the code controlling the appearance of the widget. It is optional.

Following Code Snippet demonstrates the content of a component.css file:

```
.header {
height:150px;
border: 1px solid #000000;
}
.main {
position: relative;
width: 100%;
height:auto;
}
.content {
margin: 0 0 0 250px;
height: auto;
border: 1px solid #000000;
}
...
```

component.js

- This file defines the behavior of the widget.
- It contains code for wrapping of widgets, handling of widget events initiated by the user, and interaction with AJAX.
- It is mandatory to have this file.

Following Code Snippet displays the content of a component.js file:

```
jmaki.namespace("jmaki.widgets.hello");
jmaki.widgets.hello.Widget = function(wargs) {
//widget code
}
```

- The widget is placed in a jmaki.widgets.hello namespace and is called a widget by appending the term Widget to it.
- The term Widget represents the constructor to which the widget argument is passed.

component.htm

- This file defines the default HTML template that will be used by the rendering mechanism to display the widget in the page.
- In other words, it specifies the page layout for the widget.
- jMaki ensures that the HTML template is displayed with unique and instance specific parameters.
- It is mandatory to have this file.

Following Code Snippet demonstrates the content of component.htm file:

```
<div id="${uuid}" >

</div>
```

- The markup that is included in the page is an instance of the widget that is used on the page.
- The code displays a template of a simple element with a unique id.
- The ${uuid} is replaced when jMaki processes the template before the page is displayed.

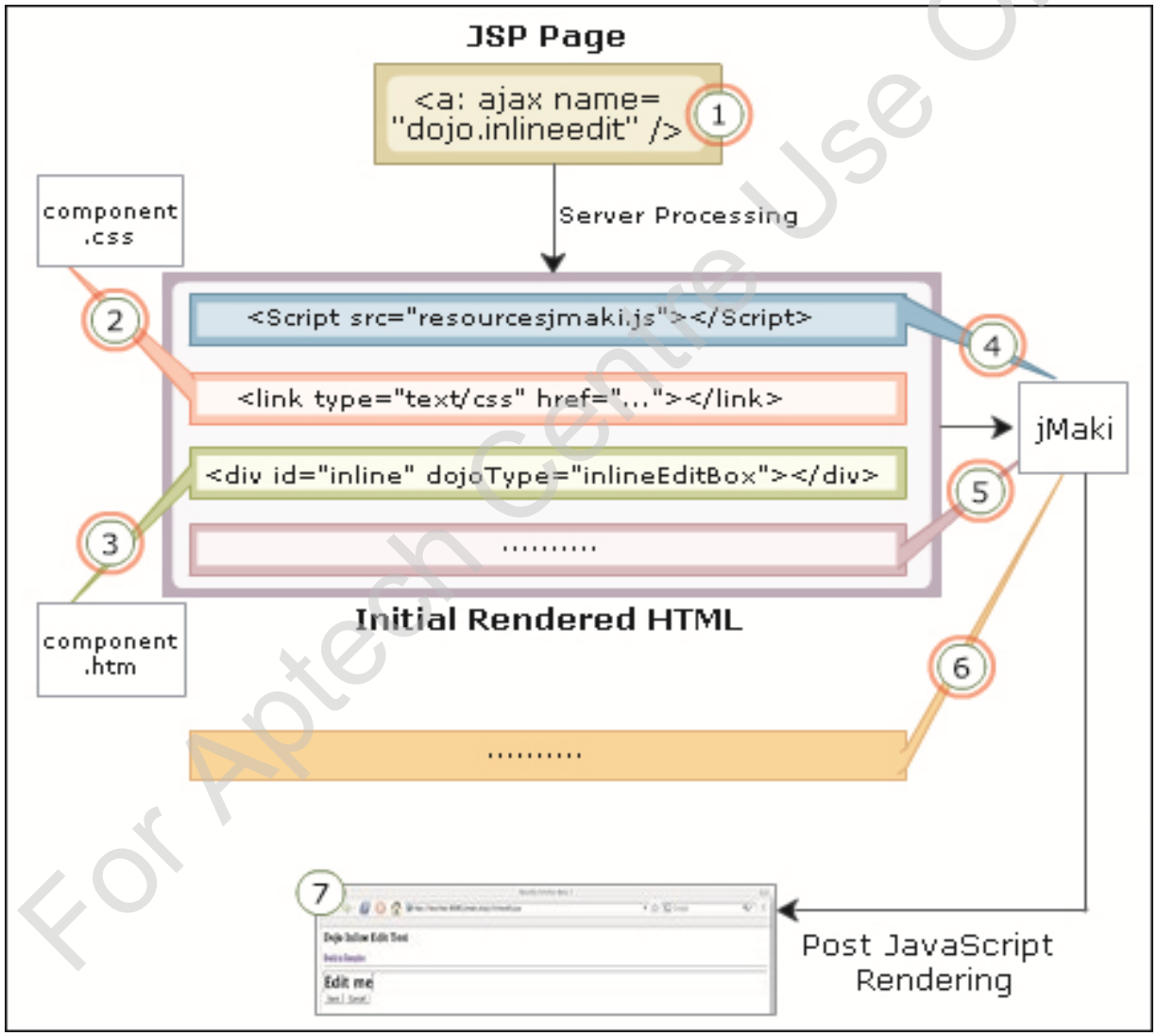
Life-Cycle 1-2

- Both client and server interactions are needed for displaying jMaki widgets. The sequence for the interactions is as follows:

1. The jMaki widget defined in a JSP page along with the taglibs is interpreted.
2. The jMaki server-side components provide the correct HTML content along with their links to component.css file that is rendered to the page.
3. The jMaki server-side runtime component provides the content from the template file (component. htm) containing unique identifier of the widget to the page.
4. The jMaki bootstrapper script present in the jmaki.js file is rendered first to create a global object named jMaki. The object contains the properties and functions for registering, loading, and supporting jMaki widgets.
5. Once the widget's template has been rendered, addWidget() function of jMaki object creates and registers the widget with the jMaki bootstrapper.
6. When the onload event of the page is fired, the registered widgets are initialized by the jmaki bootstrapper.
7. The rendered page is available for event processing.

Life-Cycle 2-2

- Following figure shows the life-cycle of a jMaki widget:



Adding Widgets

- A jMaki widget is added to a page only after the page with the required template has been created in the application. On adding a widget to a module, the three events that takes place are:
  - Widget resources such as component.js and component.htm files are added to the application under the resources directory.
  - Definition of the jMaki tag library is added to the page.
  - Custom jMaki widget tag is added to the page that refers the widgets and sets the widget attributes to default value. The tag represents a JSP handler. It also adds the tag library declaration.
- Following Code Snippet demonstrates adding of the tag library declaration and AJAX tag to the page:

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
<a:widget name="dojo.table"
args="{columns: { 'title' : 'Title', 'author': 'Author', 'bookId':
'BookID', 'price': 'Price'}}"
value="{rows:[
['JavaScript by Dummies', 'Alex John','A101', '450'],
['Ajax with Java', 'Jean Thomas','A102', '650']
]}" />
```

Loading Data

- jMaki widgets can be populated with data.
- There are three ways by which data can be loaded onto a widget. They are as follows:

Referring to a static file that contains JSON data.

Referring to the data in a bean by using an Expression Language (EL) expression in the tag's value attribute.

Referring to the data provided by a JSP page or servlet using the widget's service attribute 'All', the data needs to be passed to jMaki widget's in JSON format.

- The three steps to be followed for adding data into a widget using EL expression are as follows:
  - Creation of a bean class that represent a single object
  - Conversion of the data into JSON format
  - Loading of the data from the bean into a widget

Creation of a Bean Class

- Following Code Snippet demonstrates the creation of a Bean class:

```
public class Book {
private int ID;
    private String bookName;
    private String author;
    private String price;

    public Book(int num, String bname, String auth, String price){
        ID = num;
        bookName = bname;
        author = auth;
        this.price = price;
    }

    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }
...
}
```

Data Conversion into JSON Format

- Following Code Snippet demonstrates the code that converts data into JSON format:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.json.simple.JSONArray;

public class BookApplicationBean {

    public List addBooks() throws
Exception {
        ArrayList booklist = new
ArrayList();
        Book bookObj = new Book(201,
"Who Moved My Cheese", "Alfred John",
"450");
        booklist.add(bookObj);

        return booklist;
    }
…
```

```
public JSONArray displayBookData()
throws Exception {
        JSONArray booksArray = new
JSONArray();
        JSONArray book = new
JSONArray();
        ArrayList bookList =
(ArrayList) addBooks();
        Iterator itr =
bookList.iterator();
        while (itr.hasNext()) {
            Book bookData = (Book)
itr.next();

book.add(bookData.getID());

book.add(bookData.getBookName());

book.add(bookData.getAuthor());
            booksArray.add(book);
            book = new
JSONArray();
        }
        return booksArray;
    }
}
```

Populating the Widget

- Following Code Snippet demonstrates how to populate the widget:

```
<jsp:useBean id="bookBean" scope="session"
class="src.BookApplicationBean" />

    <a:widget name="dojo.table"
              value="{columns:[ {label : 'ID', id : 'id'},
              {label : 'Name', id : 'name'},
              {label : 'Author', id : 'author'},
              {label : 'Price', id : 'price'} ],
              rows:${bookBean.displayBookData()}}"
              />
```

- The useBean tag is used to access the property of the bean, BookApplicationBean.
- The widget, Dojo table is added to the Web page. The widget name is specified using the name attribute of the widget tag.

Data Models 1-2

- Data models specify the type of data expected by various widgets.
- Following are some of the features of data models:

Data models are standard for widgets, such as combo boxes, menus, trees, tables, and so on, across toolkits.

Data model of a widget can be used in any toolkit without changing the format of the data.

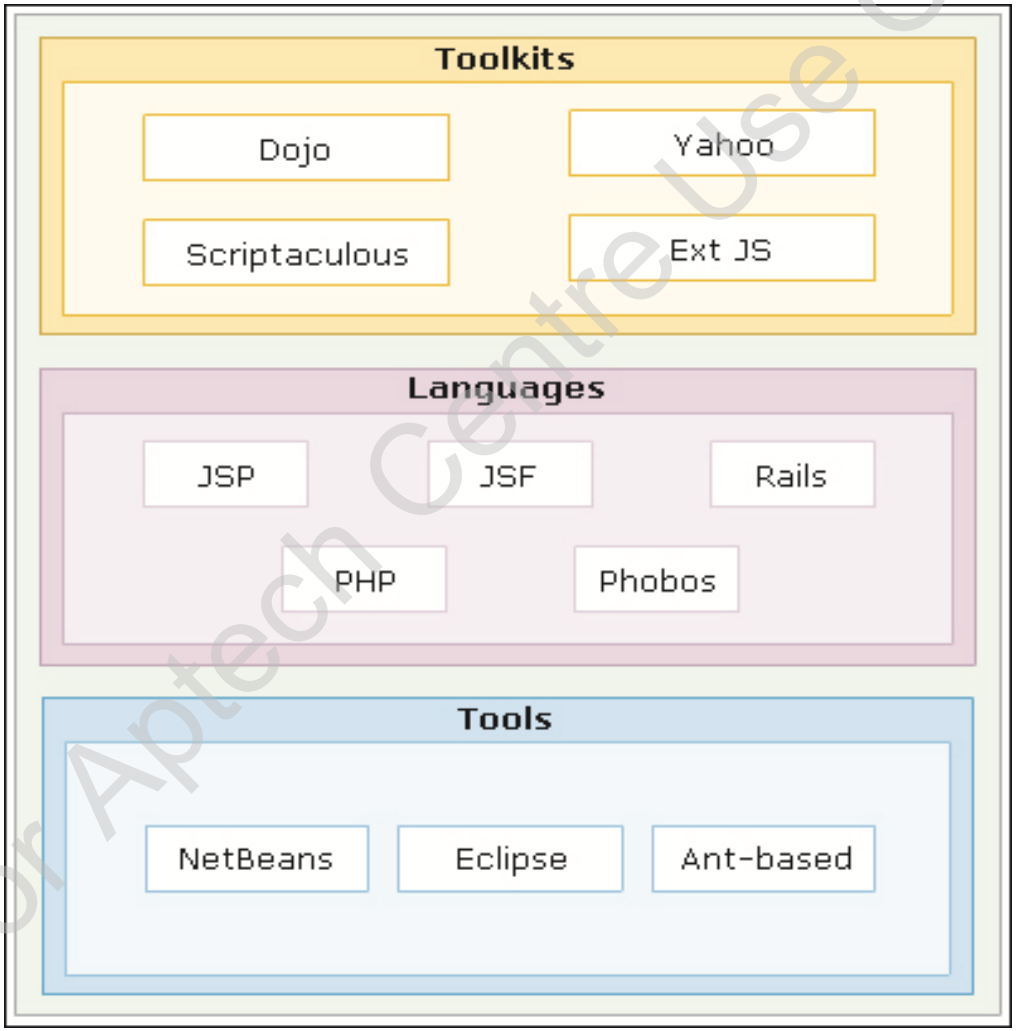
When you want to use a widget in another toolkit, the widget wrappers convert the jMaki data model as per the data requirement of the new toolkit.

Data model of a widget includes publishers and subscribers. These allow and simplify dynamic update and communication between widgets.

Data models across various widgets have similar properties and events. If you learn one data model, most of the information of the data model will be applicable for other data models too.

Data Models 2-2

- Following figure shows the data model of jMaki:



Types of Data Models

- The data models supported by various widgets are listed in the following table:

Data Model	Widget Supporting the Data Model
jMaki Menu	Yahoo Menu, jMaki Menu, jMaki Tab Menu, jMaki Accordion Menu
jMaki Table	Yahoo Datatable, Dojo Table
jMaki Tree	Yahoo Tree, Dojo Tree
jMaki Combobox	Dojo Combobox
jMaki Multiview	jMaki Dynamic Container, Dojo Accordion, Dojo Tabbedview, Yahoo Tabbedview
jMaki Fisheye	Dojo Fisheye
jMaki Drawer	Dojo Drawer
jMaki Map	Yahoo Map, Google Map

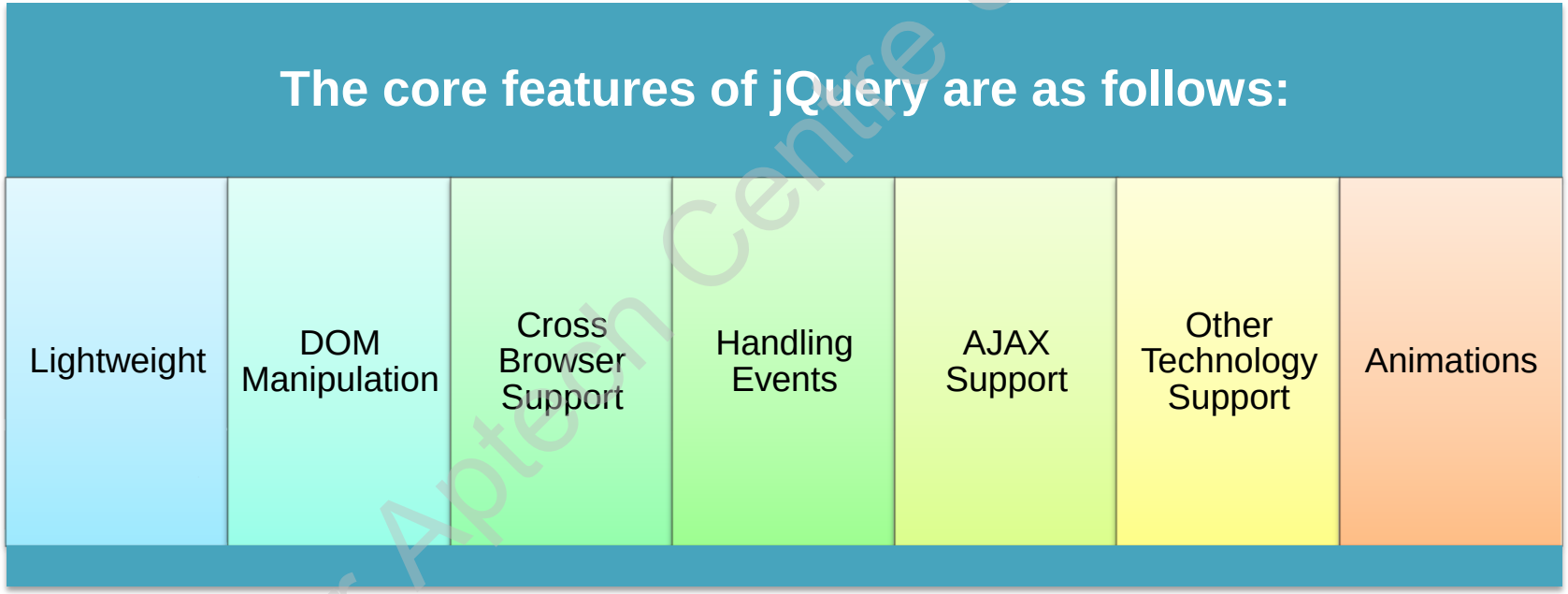
Common Properties

- There are a set of properties which are common among the different data models. They are as follows:

id	• Indicates the identifier of items such as table row, tree node, tab, and so on.
label	• Indicates the title of items such as table column, tree node, tab, and so on.
href	• Indicates that the string will act as a hyperlink. Clicking the link will navigate to the specified url.
include	• Indicates that the content from the specified url will be included in the page.
action	• Indicates that the object communicates an action to be performed by a widget.
targetId	• Indicates the id of an element on which a specific action is to be performed. • The id of the target element and targetId of the data model should be the same.

JQuery

- JQuery is a concise and fast JavaScript library. It was developed by John Resig in 2006 to do more and write less.
- It simplifies event handling, animating, document traversing, and AJAX interactions implementation.



JQuery Selectors

- The Cascading Style Sheets (CSS) selectors are harnessed even in JQuery library for easy and quick access to the elements or groups of elements in the Document Object Model (DOM).
- Following table lists some of the commonly used selectors:

Selectors	Description
$(#id)	It selects the element with given id. For example, `$("#container")`
$(.class)	It selects the element with given class. For example, `$(".name")`
$('*')	This selector selects all elements in the document.
$(element)	It selects all the elements of the type mentioned on the Webpage respectively. For example, `("<p>")`
$("element:first") and $("element:last")	It will select the first and last element on the Webpage of the specified element. For example, `$("p:first")`
$("element:odd") and $("element:even")	It will select the odd and even positioned elements on the Web page of the specified element. For example, `$("tr:odd")`

Using jQuery 1-3

- Following is the process for using the JQuery library:

1. Download the latest version of jQuery from http://jquery.com/.
2. To use the JQuery library, directly place the file in the directory where the Web page resides.
3. For using it in the particular Web page, use the script tag.

- Consider a JSP page where number of tags are to be counted on the click of a button.
- Following Code Snippet demonstrates the use of selectors.
- The index.jsp page uses the jQuery library.

```
...
<script type="text/javascript" src="jquery-1.11.1.js"></script>
        <script type="text/javascript" language="javascript">
            $(document).ready(function() {
                $("button").click(function() {
                    var count = $("li").length;
                $("#div1").text("There are " + count + " li tags");
                 });
            });
        </script>
```

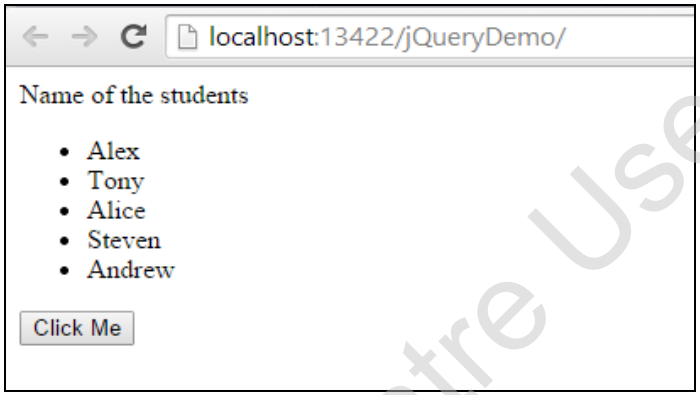
Using jQuery 2-3

```
</head>
    <body>
        <div id="container"> Name of the students
            <ul>
                <li>Alex</li>
                <li>Tony</li>
                <li>Alice</li>
                <li>Steven</li>
                <li>Andrew</li>
            </ul>
        </div>
        <button> Click Me </button> <br/>
        <div id="div1"> </div>
   </body>
...
```

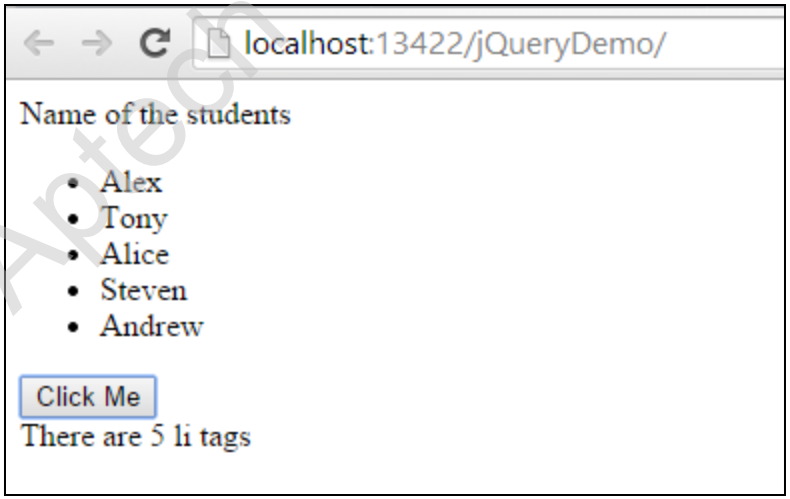
- The <li> tags on the index.jsp page are counted using the $("li") selector.
- The data is placed inside a div tag which is accessed using the $("#div") selector.
- The events such as load and click are bounded to the script using selectors $(document).ready and $("button").click.

Using jQuery 3-3

- The output for the index.jsp is shown in the following figure.



- On click of the button, the function bounded with the click event is called and the count of the <li> tag is returned in the tag as shown in the following figure:



jQuery Callback Functions

- The statements in JavaScript are executed line by line.
- However, with effects, the next line of code can be executed even if the effect is not completed. This can lead to errors due to incomplete effects.
- A callback function can be created to prevent this. The callback function is executed once the effect is fully completed.
- The example of the callback function is as follows:
  `$(selector).hide(speed,callback);`
- The callback function is executed after the hide() method execution is completed.

```
$("button").click(function(){
  $("p").hide("slow",function(){
    alert("The paragraph is now hidden");
  });
});
```

- Here, the callback parameter, that is a function, will be executed after the hide effect is completed.

JQuery Properties

- The JQuery object is associated with properties.
- Following table shows the commonly used properties:

Property	Description
jquery	Contains the version number of jQuery.
jQuery.fx.interval	It changes the rate of firing animation in milliseconds.
jQuery.fx.off	It enables/disables animation globally.
jQuery.support	It contains properties of different browser features and bugs which can be used internally by jQuery.
length	It is the count for elements in a jQuery object.

JQuery AJAX 1-3

jQuery provides several functions for implementing AJAX functionality.

The text, XML, HTML, or JSON content can be requested with JQuery AJAX methods from a remote server. The data can also be loaded in the selected HTML element directly.

A powerful and simple method of JQuery used for AJAX is the load() method that loads data from a server.

The returned data is displayed/added in the selected element.

- The syntax of the load method is as follows:
  - **Syntax**: `$(selector).load(URL,data,callbackfunction);`

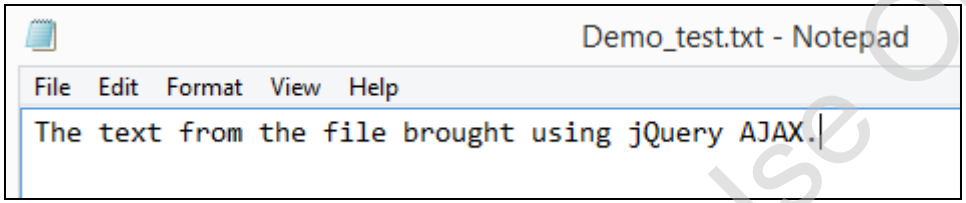
JQuery AJAX 2-3

- Following Code Snippet demonstrates the use of jQuery library and the AJAX technology within the index.jsp Web page:

```
...
<script type="text/javascript" src="jquery-1.11.1.js"> </script>
        <script>
        $(document).ready(function(){
          $("button").click(function(){
            $("#container").load("Demo_test.txt");
          });
        });
        </script>

    </head>
    <body>
        <div id="container">
            <h2>Let jQuery AJAX change this text.</h2>
        </div>
        <br/>
        <button>Get File Content</button>

    </body>
...
```

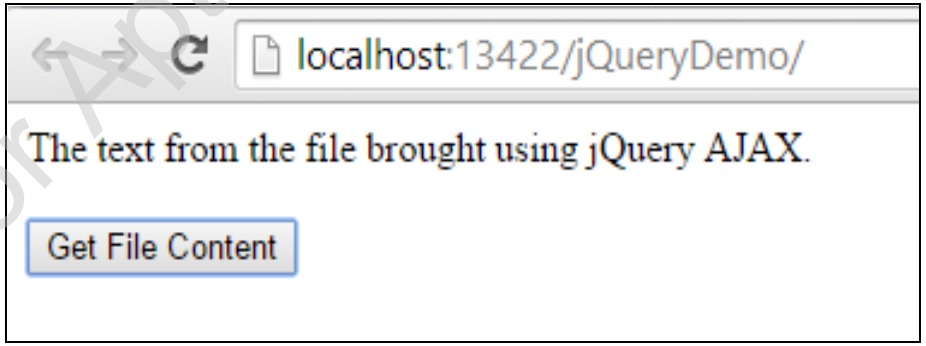
JQuery AJAX 3-3

- The contents of the Demo_test.txt file are shown in the following figure:



- The output for the index.jsp page is shown in the following figure:

- On clicking the button, the data is brought asynchronousy from the file and the content on the div is modified as shown in the following figure:



YUI

- YUI is a robust, fast, and scalable framework.
- It is a JavaScript library which runs on mobile, server, or desktop browsers.
- There are currently two supported versions of YUI namely, YUI 2 launched in 2006 and YUI 3 which was released in 2009.
- Following are the YUI features that have made it popular:

A huge library of widgets, including one of the most feature-complete datatables.

Stellar documentation whereby each component and widget has detailed instructions, examples, and api documentation.

YUI provides a number of rich development tools including a profiler, in-browser console, and testing framework.

Flexible event handling with built-in support for touch and gesture events.

Sandboxing

- This is a new feature in YUI 3. It states each instance in YUI is protected, limited, and self-contained.
- Thus, the other YUI instances are segregated and specific need for functionality is tailored properly.
- The process of creating isolated iframe sandboxes is simplified by sandbox where tasks such as unit testing and profiling are done.

Following Code Snippet demonstrates the use of sandbox:

```
<script>
// Create a YUI sandbox.
YUI().use('node', 'event', function (Y) {
    // The Node and Event modules are loaded and can be used.

});
</script>
```

Selectors

- The Selector class can be used to provide support for using the CSS selector for querying the DOM.
- It returns an HTML Element.
- The methods of the Selector class are shown in the following table:

Method	Description
query()	It is used for attaching an event to number of elements.
test()	It returns true or false depending on the node, if matched with selector provided.
filter()	It returns a set of nodes based on a simple selector.

Custom Events

The DOM events are also handled in YUI using the Event Utility API's.

The YUI Custom Event System allows to define and use events which are not included in the DOM — events that are specific to an application.

Custom Events work much like DOM events. They can bubble, pass event facades, have their default behaviors and propagation suppressed, and so on.

The EventTarget class provides the API for the custom events.

EventTarget is extended by all other classes, but if the custom event APIs are required, user can augment or extend classes with EventTarget directly.

Nodes and NodeList 1-4

- The two methods provided by YUI for accessing DOM nodes through its Node module are as follows:
  - Y.one('selector') - returns a YUI Node representing a DOM Node.
  - Y.all('selector') - returns a YUI NodeList of all node matches.
- The steps required for using YUI framework are as follows:

1. • Download the seed file for YUI from http://yuilibrary.com/.
2. • Place it in the Web Pages folder for using it.

Nodes and NodeList 2-4

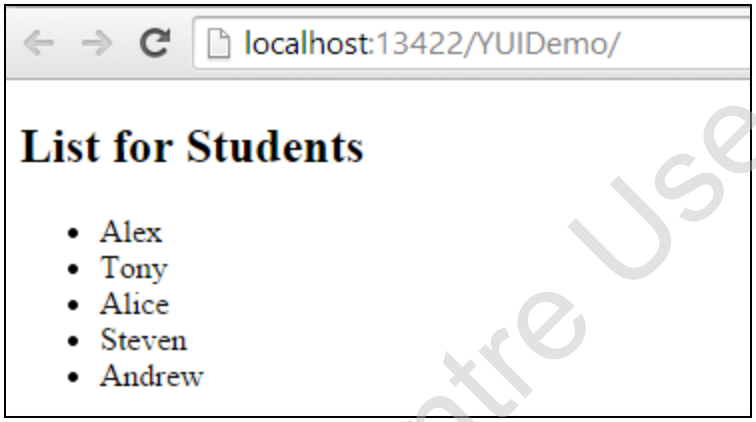
- Consider the scenario where all features of the YUI listed in the session have been used.

Following Code Snippet demonstrates use of events, selectors, and sandboxing:

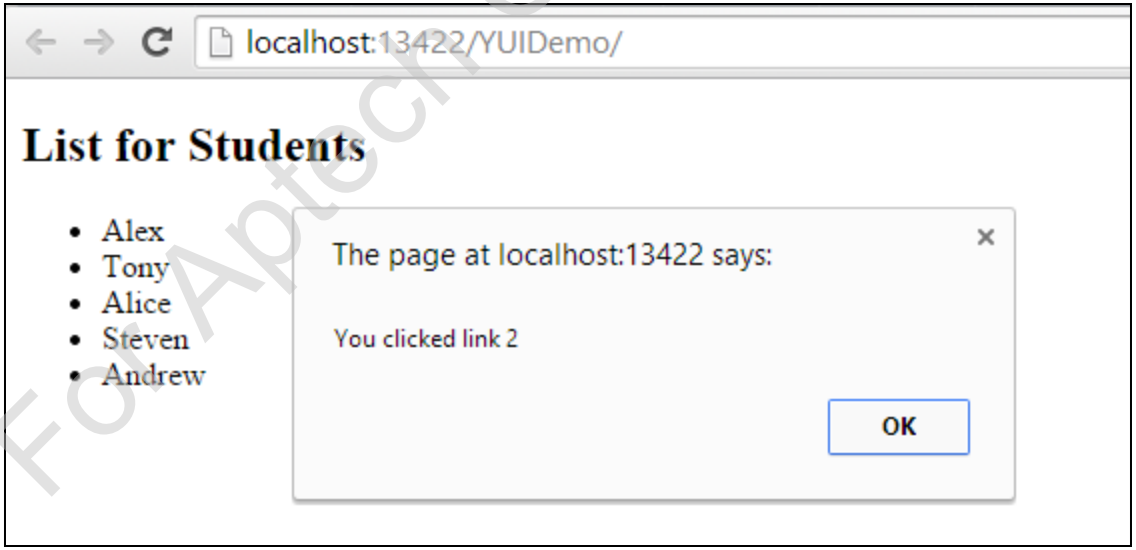
```
...
<script src="yui-min.js"></script>
        <script>
            YUI().use("node",
function(Y) {

Y.one('ul').delegate("click",
function(em) {
                    var itemList =
em.container.all('li');
                    var node =
em.currentTarget;
                    alert("You
clicked link " +
itemList.indexOf(node));

node.addClass("Clicked");
alert(node.getAttribute("class"));
                }, 'li');
            });
        </script>
</head>
    <body>
        <h2>List for Students</h2>
        <ul>
            <li>Alex</li>
            <li>Tony</li>
            <li>Alice</li>
            <li>Steven</li>
            <li>Andrew</li>
        </ul>
    </body>
...
```

Nodes and NodeList 3-4

Following figure shows the output of the index.jsp page:



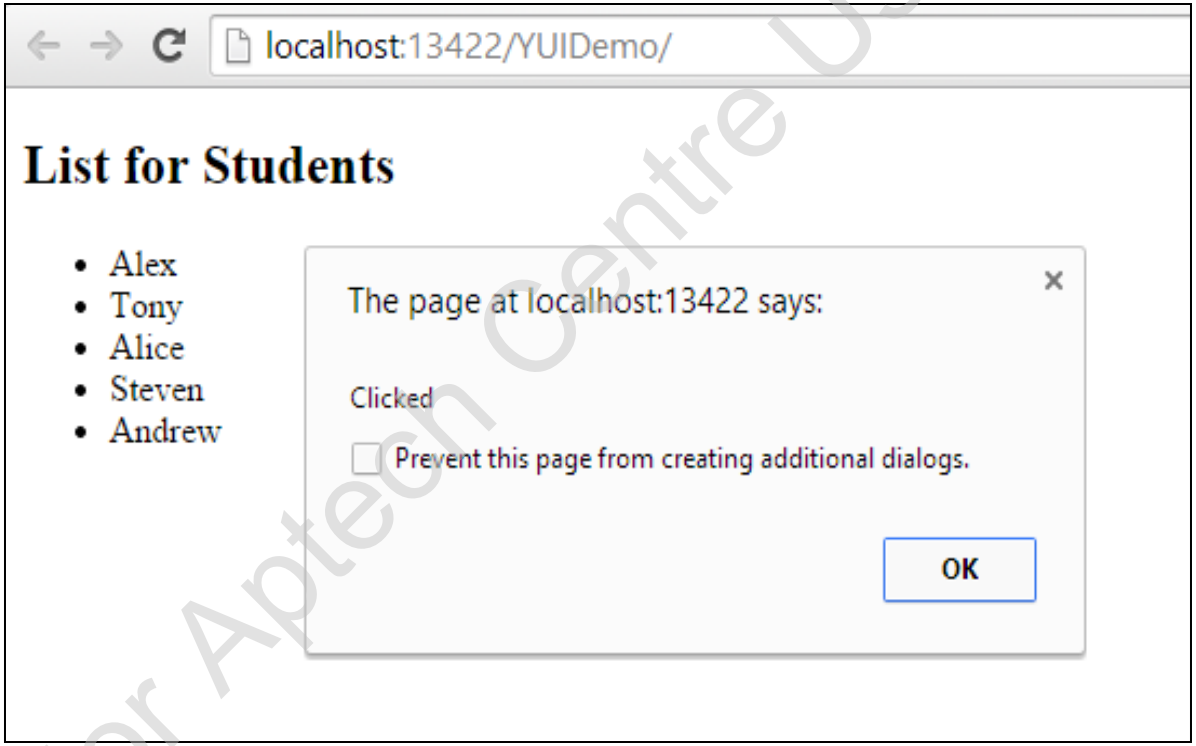
- On clicking any name, the index of the node will be displayed in the message box as shown in the following figure:



Nodes and NodeList 4-4

- On clicking Alice, this message box is shown and the event delegation is seen when the next message box shows the class name of the li tag.

Following figure shows event delegation output:



Summary

- jMaki is a lightweight client-server framework used for creating AJAX applications. It wraps the functionality of JavaScript technology.
- jMaki Architecture comprises client and server components.
- The three core resource files for a widget are component.css, component.js, and component.htm.
- jQuery is a concise and fast JavaScript library developed by John Resig in 2006 to do more and write less.
- The callback function in jQuery is executed once the effect is fully completed.
- YUI is a robust, fast, and scalable framework. It is JavaScript library which runs on mobile, server, or desktop browsers.
- The process of creating isolated iframe sandboxes is simplified by sandbox where tasks such as unit testing and profiling are done.