
Software Construction and User Interface (SE/ComS 319)

Ali Jannesari

Department of Computer Science

Iowa State University, Fall 2018

EVENT-DRIVEN PROGRAMMING

Outline

- Event-Driven Programming (EDP):
 - Concepts
 - Event handling
 - Event-driven architecture
 - Asynchronous programming, etc.
 - Web UI and EDP with JavaScript (Node.js)
 - GUI and EDP with JavaFX

Event-Driven programming (1)

- A programming paradigm in which the flow of the program is determined by **events** such as:
 - User actions (mouse clicks, key presses)
 - Sensor outputs (mostly in embedded systems)
 - Messages from other programs/threads (device drivers)

Event-Driven programming (2)

- Event-driven programming
 - ... is the dominant paradigm used in **graphical user interfaces** and other applications
 - e.g. JavaScript web applications: performing actions in response to user input.
 - ... is used in **Human-computer interaction (HCI)**

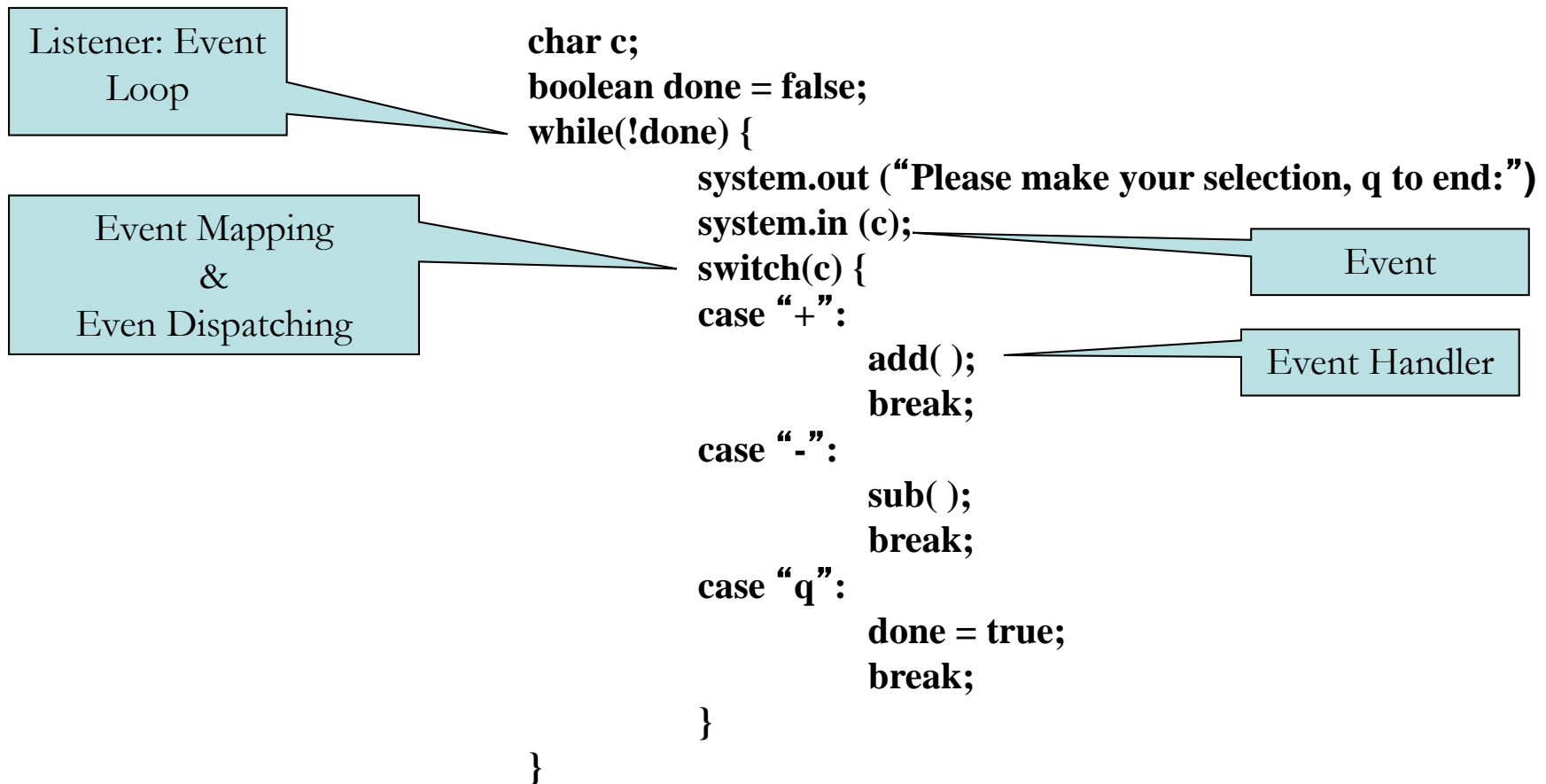
Human-computer interaction (HCI)

- HCI: Interactive computing systems for human use
 - CLI: command line interface (with keyboard)
 - **GUI: graphical user interface (mouse)**
 - NUI: natural user interface with Audio/Video (Kinect)
- A main HCI Component: **Interaction**
 - User interaction
 - Event
 - Event Handling
 - Output
- A **GOOD GUI** allows users to perform interactive tasks easily:
 - What you see is what you get

Event-Driven programming (2)

- Application waits (idles) after initialization until the user generates an event through an input device (keyboard, mouse, ...).
- The OS dispatches the event to the application who owns the active window.
- The corresponding event handler(s) of the application is invoked to process the event.

Event-Driven programming (2)



Event-Driven programming (4)

1. Event generators: GUI components (e.g. buttons, menus, ...)
2. Events/Messages: e.g. `MouseClicked`, ...
3. Event loop (Listener) : an infinite loop constantly waits for events.
4. Event mapping / Event registration: inform event dispatcher which event an event handler is for.
5. Event dispatcher: dispatch events to the corresponding event handlers.
6. Event handlers: methods for processing events. E.g. `OnMouseClicked()`, ...

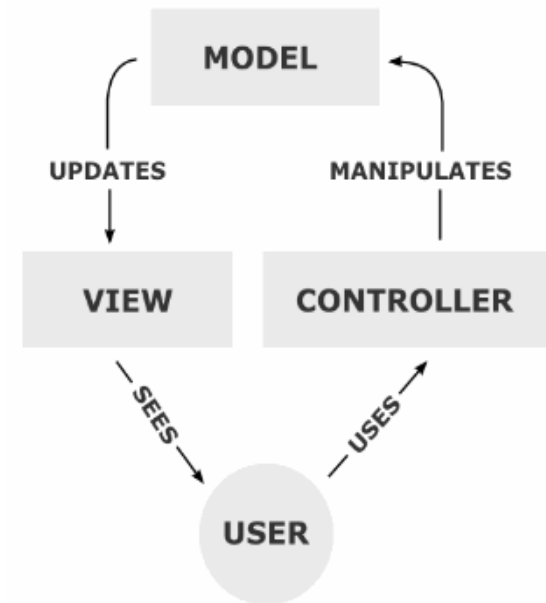
Event-driven programming (5)

- Concepts
- Event-driven programming with
 - JavaScript (Node.js)
 - JavaFX (Java)

EVENT-DRIVEN PROGRAMMING WEB USER INTERFACES

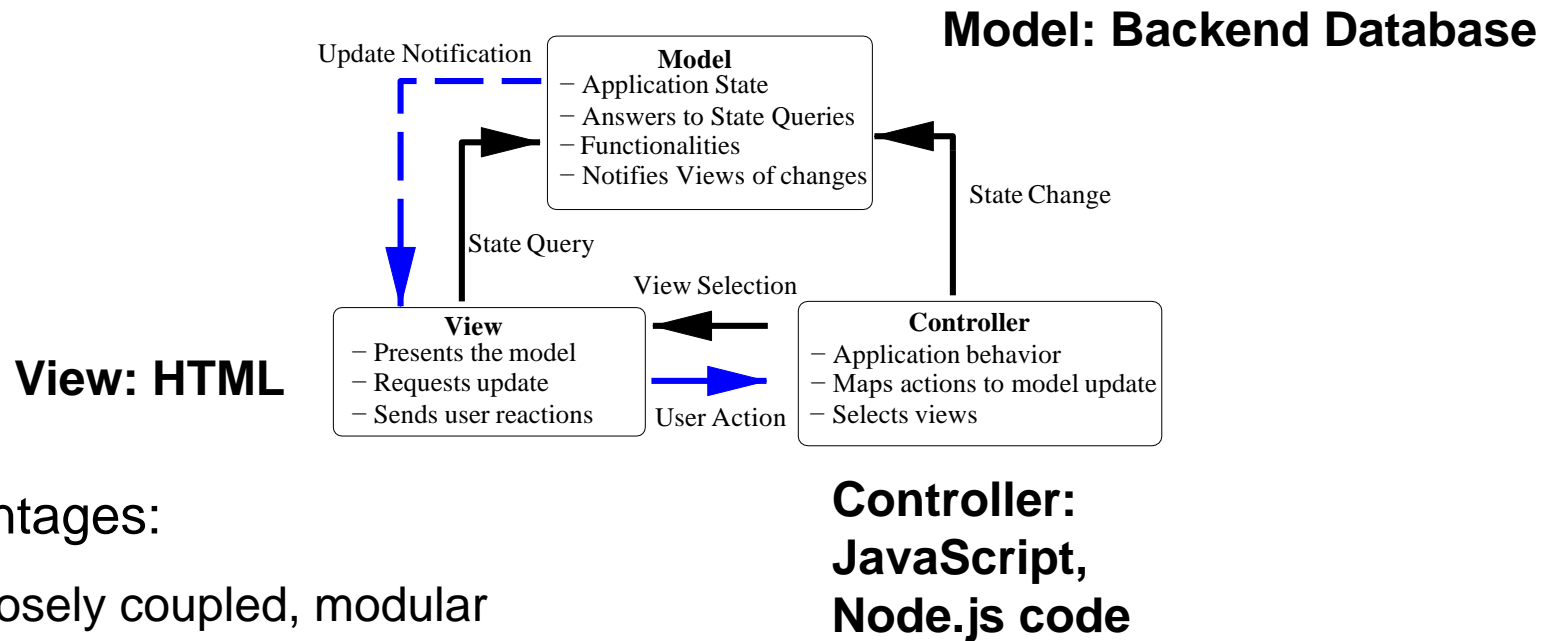
Event-Driven Programming – Web UI

- **MVC (Model – View – Controller) in Web UI:**
 - **View:** Browser presentation (HTML)
 - **Model:** Data (Backend Database or (simple) embedded)
 - **Controller:**
 - Client scripts/programs, e.g. JavaScript
 - Server scripts/programs, e.g. Node.js



MVC

- Model-View-Controller architecture:

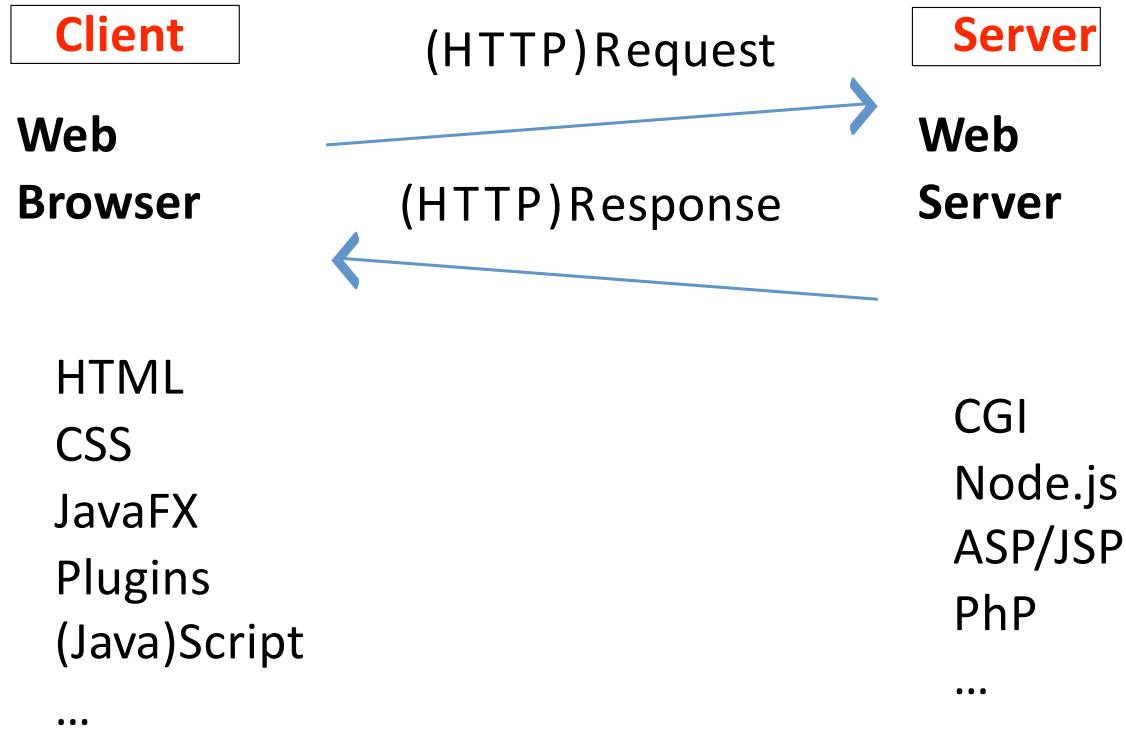


- Advantages:
 - Loosely coupled, modular
 - Model with different views
 - Controller decides when/how to update the model and/or the view
 - Model can change the view

Client/Server programming

- Use **client-side** programming for
 - Validating user input
 - Prompting users for confirmation, presenting quick information
 - Calculations on the client side
 - Preparing user-oriented presentation
 - Any function that does not require server-side information
- Use **server-side** programming for
 - Maintaining data across sessions, clients, applications

Web software: Client/Server (1)

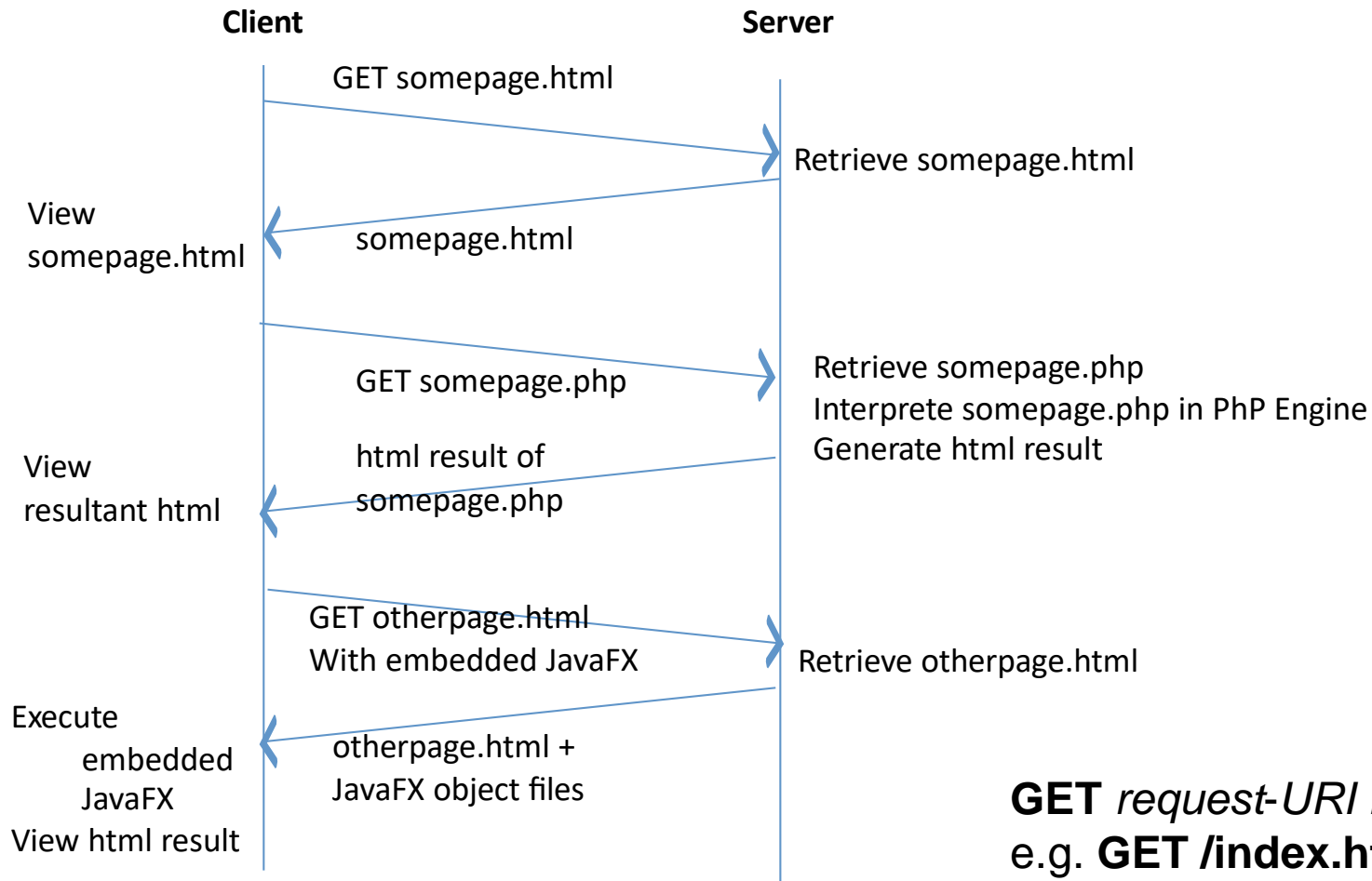


HTTP (Hypertext Transfer Protocol): HTTP is a client-server application-level protocol. It typically runs over a TCP/IP connection.

Web software: Client/Server (2)

- Web-client and Web-server communicates using HTTP protocol
 - Client can send a HTTP request: method “**get**” or “**post**”
 - Server can read a HTTP request and produce HTTP response
- Server side programs should be capable of reading HTTP request and producing HTTP response

Web software: Client/Server (3)



GET *request-URI HTTP-version*
e.g. **GET /index.html HTTP/1.0**

Common Gateway Interface (CGI) – Classic method

- Standard for the server to communicate with external applications
- Server receives a client (Http) request to access a CGI program
- Server creates a new process to execute the program
- Server passes client request data to the program
- Program executes, terminates, produces data (HTML page)
- Server sends back (Http response) the HTML page with result to the client

HTML – Example

```
<html>
<head></head>
<body>
<form action="<some-server side cgi program>" method="post">
First Name: <input type="text" name="fname"/>
Last Name: <input type="text" name="lname"/>
<input type="submit" value="Submit"/>
</form>
</body>
</html>
```

- Once the user clicks the submit button, the data provided in the form fields are “submitted” to the server where it is processed by a CGI program!

HTTP Request/Response Message

- Message Header
 - Who is the requester/responder
 - Time of request/response
 - Protocol used ...
- Message Body
 - Actual message being exchanged

HTTP Request

GET /index.html HTTP/1.1

Host: http://www.se.iastate.edu

Accept-Language: en

User-Agent: Mozilla/8.0

Query-String: ...

HTTP Response

HTTP/1.1 200 OK

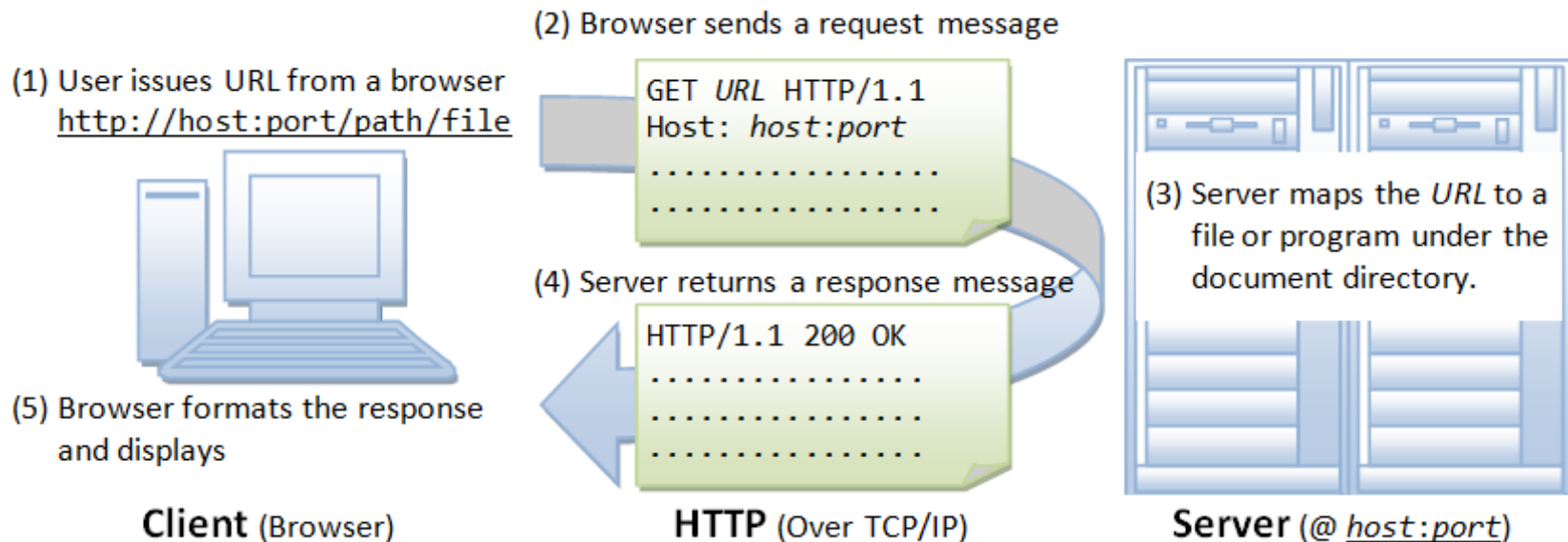
Date: Sat, 27 Oct 2007 16:00:00 GMT

Server: Apache

Content-Type: text/html

- Response Codes:
 - 200s: good request/response
 - 300s: redirection as the requested resource is not available
 - 400s: bad request leading to failure to respond
 - 500s: server failure

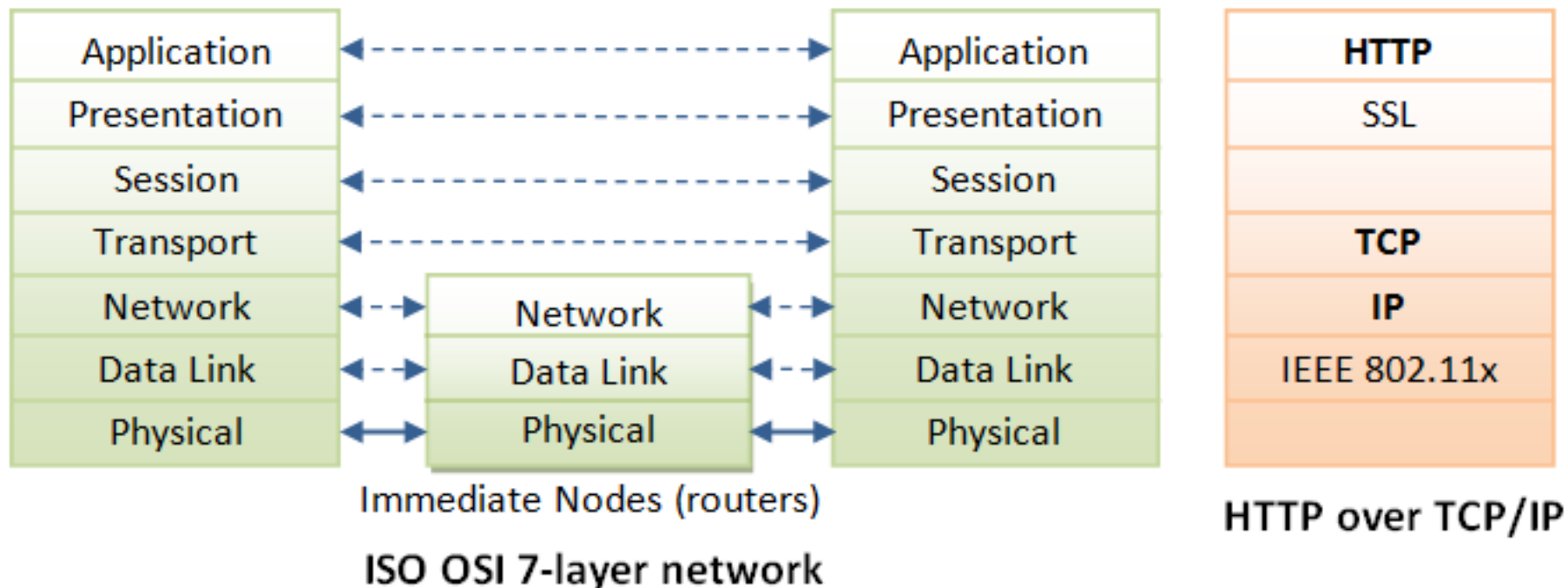
Web software: Client/Server



Source: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

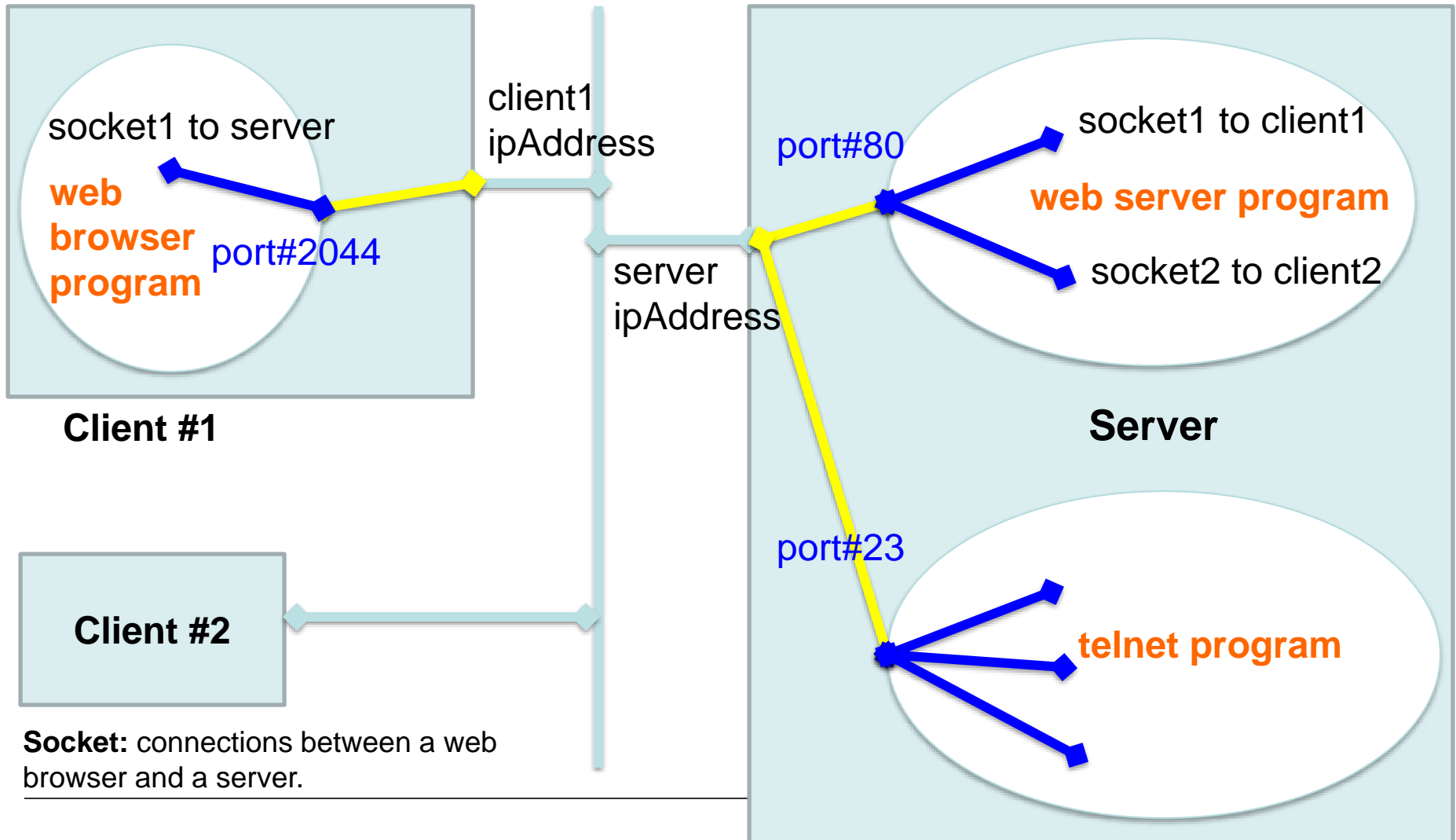
- **GET:** The GET method is used to retrieve information from the given server using a given URI.
 - Requests using GET should only retrieve data and should have no other effect on the data.

Client/Server: HTTP over TCP/IP



Source: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Web software: Client/Server – Connections



Client-Side Dynamics (1)

- HTML + Javascript
- Html elements: **forms**
- Html style elements: fonts, headings, breaks
- CSS: uniformly manipulate styles
- JavaScript:
 - manipulate styles (CSS)
 - manipulate html elements
 - validate user data
 - communicate with the server-side programs
- In HTML: `<input id="clkb" type="button" value="Click" onclick="clkF()"/>`
- In Javascript file: `function clkF() { alert("Hello"); }`

Client-Side Dynamics (2)

- Html elements: **View**
- CSS: **Model**
- Javascript: **Controller**
- CSS: A simple mechanism for adding style to Web documents.
 - Look & feel of Webpages
 - Layouts, fonts, text, image size, location
 - Objective: Uniform update
- Javascript as a client side event-driven programming
 - Client-side computations
 - Form validation + warnings
 - Dynamic views

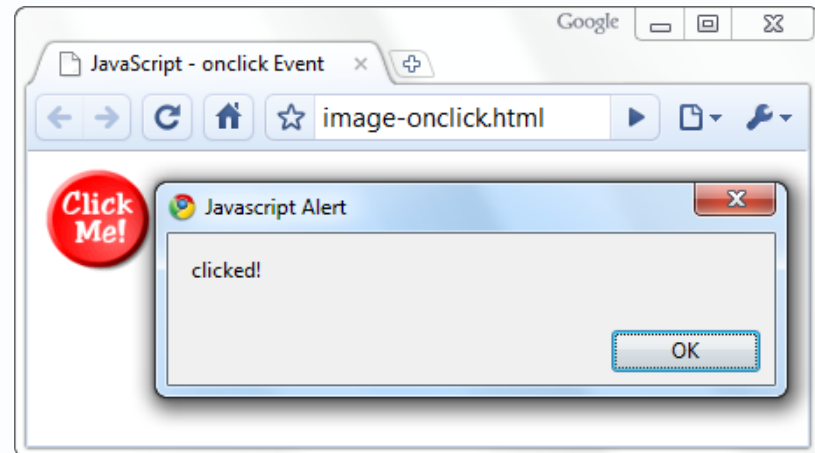
How to add JavaScript to html file?

- Include in html file:
 - `<script> your javascript code goes in here </script>`
- Can also include from a separate file:
 - `<script src="./01_example.js"></script>`
- Can include from a remote web site:
 - `<script src="http://.../a.js"></script>`

JavaScript Event Handler – Example

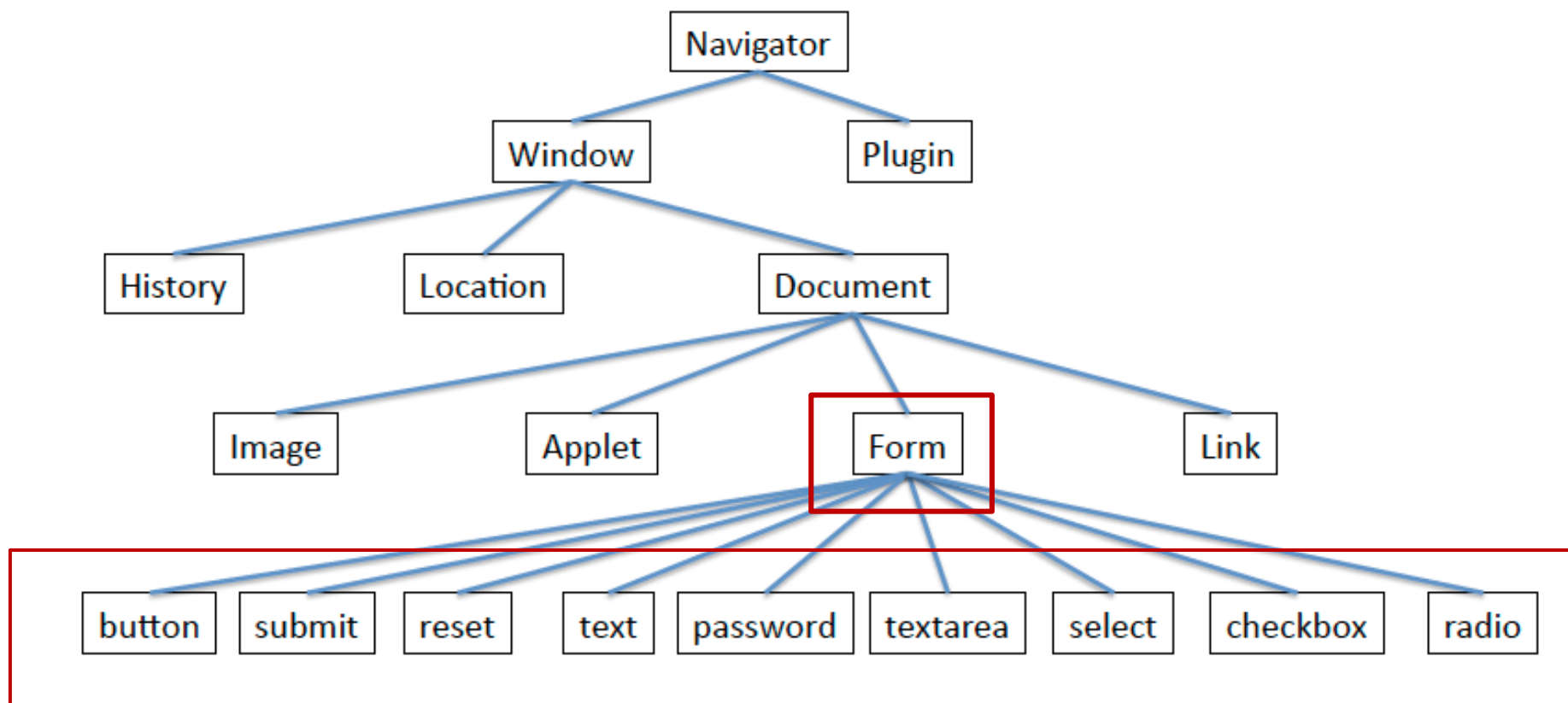
```
<html>
<head>
<script type="text/javascript">
  function test (message) {
    alert(message);
  }
</script>
</head>

<body>
  
</body>
</html>
```



Using **onclick**, we attach **event handlers**.

JavaScript accessibility hierarchy





NODE.JS

Event-driven programming – Node.js

- Open-source, cross-platform **JavaScript** run-time environment that executes JavaScript code **server-side**
 - Historically JavaScript used for client-side programming
- "JavaScript everywhere" paradigm (popular)
 - Unifying web application development
 - Same language for server side and client side scripts.



A JavaScript runtime
environment running Google
Chrome's V8 engine

Goal is to provide an easy way to
build scalable network programs

Why Node.js?

- Non Blocking I/O
- V8 Javascript Engine
 - V8 is Google's open source high-performance JavaScript engine, written in C++ and used in Node.js
- Single Thread with Event Loop
- 40,025 modules: JavaScript libraries you can include in your project
- Different platforms: Windows, Linux, Mac,...
- 1 Language for Frontend and Backend
 - Core in C++ on top of V8
 - Rest of it in javascript
- Active community

Event-driven programming – Node.js

- Event-driven architecture
 - Asynchronous I/O
 - Scalability with many input/output operations
 - Real-time Web applications
 - e.g., real-time communication programs, browser games and data streaming, etc.
- Node.js functions are non-blocking
 - Commands execute concurrently or even in parallel (unlike PHP that commands execute only after previous commands finish)
 - Node.js uses callbacks to signal completion or failure

Asynchronous programming – Node.js

- Node.js uses asynchronous programming (runs single-threaded, **non-blocking**) → very memory efficient
- Handling a file request:
 - In PHP/ASP.net:
 1. Sends the task to the computer's file system.
 2. Waits while the file system opens and reads the file.
 3. Returns the content to the client.
 4. Ready to handle the next request.
 - In Node.js:
 1. Sends the task to the computer's file system.
 2. Ready to handle the next request.
 3. When the file system has opened and read the file, the server returns the content to the client.

Blocking vs. non-blocking: PHP vs. Node.js

- PHP:

```
<?php
$result = mysql_query('SELECT * FROM ...');
while($r = mysql_fetch_array($result)){
    // Do something
}
```

```
// Wait for query processing to finish...
?>
```

- Node.js:

```
<script type="text/javascript">
mysql.query('SELECT * FROM ...', function (err, result, fields){
    // Do something
});
```

```
// Don't wait, just continue executing
</script>
```

To select data from a table in MySQL, use the "SELECT" statement

Error handler

Callback!

The third parameter of the callback function is an array containing information about each field in the result object

Blocking vs. non-blocking

- **Blocking:**

- Read data from file `var data = fs.readFileSync("test.txt");`
- Show data `console.log(data);`
- Do other tasks `console.log("Do other tasks");`

- **Non-blocking:**

- Read data from file
 - When read data completed, show data!



Callback!

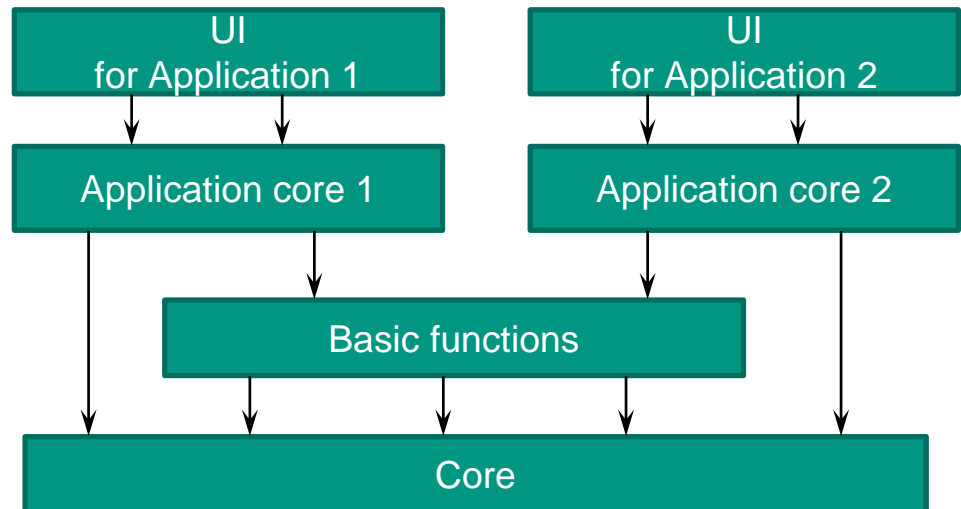
- Do other tasks `fs.readFile("test.txt", function(err, data) {
 console.log(data);
});`

Event-driven programming – When to use Node.js?

- Creation of Web servers and networking tools
 - Ideal for applications that serve a lot of requests but don't use/need lots of computational power per request
- Using JavaScript and a collection of **modules** that handle various core functionality such as:
 - File system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams, etc.
 - Modules use an API (**interfaces**) designed to **reduce the complexity** of writing server applications

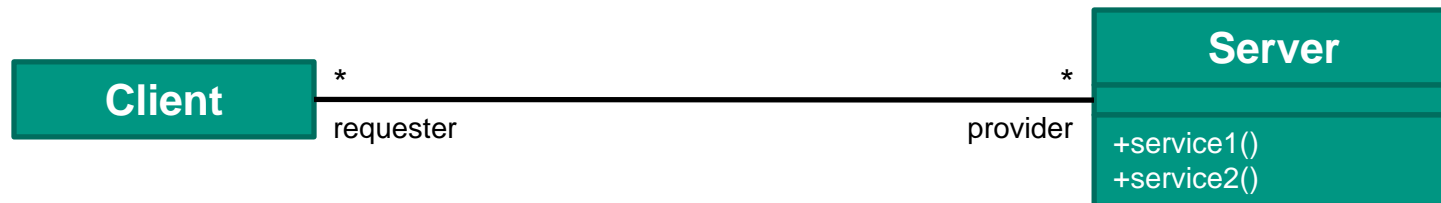
Event-driven architecture (1)

- Architectural patterns (n-tier, client/server, ...) may be applied by the design and implementation of applications and systems
 - Transmit events among loosely coupled software components and services
- n-tier architecture (layered architecture, see section Architectural styles)
 - 4-tier:



2-tier architecture – Client/Server

- One or more servers provide services for other subsystems called clients.
- Each client invokes a function of the server which performs the desired service and returns the result.
 - The client must know the interface of the server.
 - Conversely, the server does not need to know the client's interface.
- An example of a 2-tier, distributed architecture:



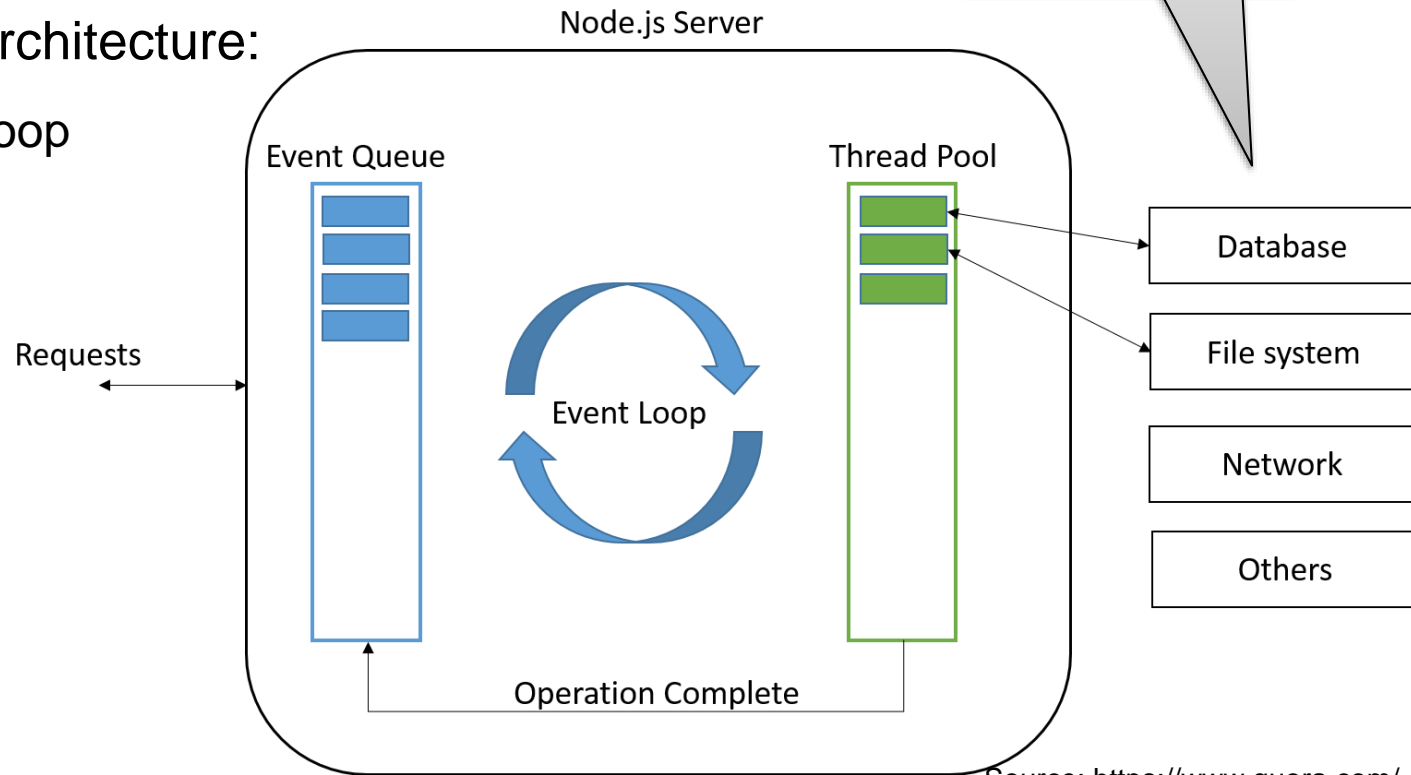
- Event-driven architecture: A **single thread** (server), of the event loop processes all the requests from clients (event queue)

Event-driven architecture (2)

- Event-driven architecture:
 - **Processing loop**
 - **Event queue**
 - **Call-back**

Event-driven architecture – Node.js

- Event-driven architecture:
 - Processing loop
 - Event queue
 - Call-back



Source: <https://www.quora.com/>

- Node.js Architecture: The event loop simply iterates over the event queue (a list of events) and callbacks of completed operations.

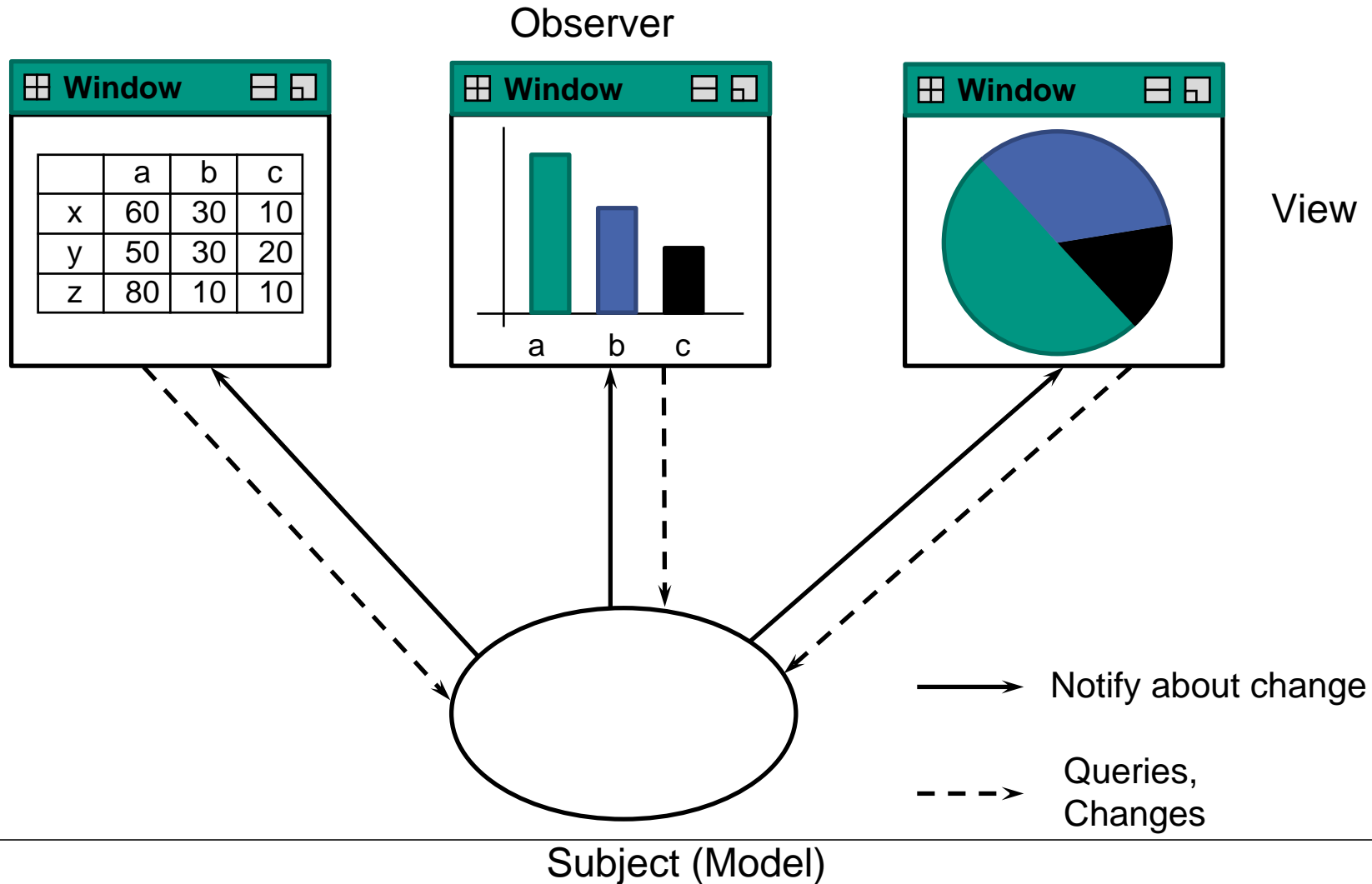
Event-driven architecture (3)

- Event-driven architecture can complement **service-oriented architecture (SOA)**
- Services can be activated by triggers fired on incoming events.
- SOA is an architecture style that assembles applications from (independent) services (see section Architectural styles)
- Services are considered as central elements of a company (keyword: services)
- Provide encapsulated functionality to other services and applications

Observer design pattern – Node.js

- A **single thread**, using non-blocking I/O calls
- ➔ **Observer** design pattern: sharing a single thread among all the requests
- Defines a **1-to-n** dependency between objects so that changing a state of an object causes all dependent objects to be **notified** and **updated automatically**.
- One to many relationship
- The many need to know changes in “one” immediately
- Synonyms (aka)
 - Dependence
 - Publisher-subscriber
 - Subject-observer

Observer example – MVC



Threading – Node.js

- A **single thread**, using non-blocking I/O calls
 - Support tens of thousands of concurrent connections **without** the **cost of thread context switching**
 - Building highly concurrent applications
 - A thread pool handles execution of parallel tasks
- Good for horizontal scaling (lots of request)!
- **Drawback** of the single-threaded approach: **No vertical** scaling by increasing the number of cores (not good for massive parallel computing)
 - Needs additional module: such as cluster, StrongLoop Process Manager, etc.
 - **Mitigation:** Developers can increase the default number of threads in the thread pool → OS distributes the threads across multiple cores

Thread-based vs. Event-based (Node.js)

Threads	Asynchronous Event-driven
monitor (difficult to program)	event handler (using queue and then processes it)
scheduling (ready, running, waiting, ...)	event loop (only one thread, which repeatedly fetches an event)
exported functions (thread-safe)	event types accepted by event handler
returning from a procedure (using context switching)	dispatching a reply (no contention and no context switches)
executing a blocking procedure call	dispatching a message, awaiting a reply
waiting on condition variables	awaiting messages

Conclusion:

- Use threads for performance critical applications (kernels)
- Use events for GUI and distributed systems

What can you do with Node.js?

- Node.js file contains tasks and executes them upon set of events
 - Generate dynamic content (even desktop programs in js)
 - Create, open and read, or delete files on the server
 - Gather and modify data in the database
 - Collect form data, etc.
- Availability of rich frameworks
 - Angular, Node, Backbone, Ember, etc.
- Ability to keep data in native JSON (JavaScript Object Notation, similar to XML) format in your database
- Very good supportive community
 - Linux Foundation, Google, PayPal, Microsoft, ...

Node.js – Libraries

standard lib

process.argv // command line arguments
console.log
setInterval(callback, time)
require(library)

fs

Readdir // reads the contents of a directory
readFile //read entire file
readFileSync
createReadStream //read in chunks

path

Extname // get the extension from a file path

Node.js – Example 'Hello World!'

```
var http = require('http');
```

include a module (library), use the require() function with the name of the module

```
//create a server object:
```

```
http.createServer(function (req, res) {  
  res.write('Hello World!'); //write a response to the client  
  res.end(); //end the response  
}).listen(8080); //the server object listens on port 8080
```

Use the createServer() method to create an HTTP server

Represents the request/response from/to the client

Writes "Hello World!" if a web browser tries to access your computer on port 8080

Node.js – Example 'Hello World!' (2)

- Create a file named "app.js"

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Source:
<https://nodejs.org/en/docs/guides/getting-started-guide/>

Using Existing modules

```
var fs = require('fs');    // include File System module
var path = require('path');
// typically an object or a function is returned.
```

```
var buf = fs.readdir(process.argv[2], // command line arguments
function(err, data) {
  for (i = 0; i < data.length; i++) {
    var s = path.extname(data[i]);
    if (s === "." + process.argv[3]) {
      console.log(data[i]);
    }
  }
} // end of for
} // end of callback function for readdir
```

Create your own modules – Example (1)

```
exports.myDateTime = function () {  
    return Date();  
};
```

Save the code above in a file called "myfirstmodule.js"

```
var http = require('http');  
var dt = require('./myfirstmodule');
```

Include and use the module in any of your Node.js files.

```
http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write("The date and time are currently: " + dt.myDateTime());  
    res.end();  
}).listen(8080);
```

Create your own modules – Example (2)

// FILE myModule.js

```
module.exports = function (dir, ext, callback) {  
  var fs = require('fs');  
  var path = require('path');  
  var retValue = [];  
  fs.readdir(dir, function(err, data) {  
    if (err) return callback(err);  
    retValue = data.filter(function(filename) {  
      return path.extname(filename) === "." + ext;  
    });  
    callback(null, retValue);  
  }); // end of callback to readdir  
}; // end of function
```



USERS of this module will need to provide **dir**, **extension**, and **callback**

Create your own modules – Example (2)

Using the created module

```
var x = require('./mymodule');
```

```
// users need to provide dir, extension, callback
```

```
//
```

```
x(process.argv[2], process.argv[3], function(err, data)
```

```
{
```

```
  if (err) return console.error ("error:", err);
```

```
  data.forEach(function(file) { // for each array element
```

```
    console.log (file);
```

```
  });
```

```
} // end of callback function
```

```
); // end of call to x
```

Asynchronous I/O – Example

- **NO WAIT!** until read is complete:

```
var fs = require('fs');  
var buf = fs.readFile(process.argv[2],  
  function(err, data) { //CALLBACK  
    if (err) { return console.log(err); }  
    var sArray = data.toString().split("\n");  
    console.log(sArray.length-1);  
  } );
```

Include the File System module:
fs = require('fs');
fs.readFile(file, [encoding], [callback]);

fs.readFile() method is used to read files .

// NO WAIT! – DO THE NEXT INSTRUCTION RIGHT AWAY

Synchronous I/O – Example

Waits until i/o is done

EXAMPLE:

```
var fs = require('fs'); // node's modular code
```

```
var buf = fs.readFileSync(process.argv[2]);
```

```
//WAIT!
```

```
var sArray = buf.toString().split("\n");
```

```
console.log(sArray.length-1); // print number of lines
```


Standard callback pattern

Callback function will look like:

```
function (err, data) {  
    if (err) { // handle error }  
    else {  
        // do something with data  
    }  
});
```

This callback is **called once** when event happens (for example, i/o is complete)

Event handling – Event emitter pattern

// Instead of only completed event, many events may be fired.

// Handlers can be registered for each event.

```
var fs = require('fs');
```

```
var file = fs.createReadStream('./' + process.argv[2]);
```

readStream object fires events when opening and closing a file

```
file.on('error', function(err) {  
  console.log("Error:" + err);  
  throw err;  
});
```

createReadStream fires **error**, **data**, and **end** events

```
file.on('data', function(data) {  
  console.log("Data: " + data);  
});
```

Using **on** function, we attach **event handlers**.

```
file.on('end', function() {  
  console.log("finished reading all of data");  
});
```

Event emitter API

- **Event types (determined by emitter)**
 - error (special type)
 - data
 - end
- **API**
 - `.on` or `.addListener`
 - `.once` (will be called at most once)
 - `.removeEventListener`
 - `.removeAllEventListeners`

Creating an event emitter – Example

// file named myEmitter.js

var util = require('util'); // step 1

Util module provides access to some utility functions.

var EventEmitter = require('events').EventEmitter; // step 2

var Ticker = function() {

var self = this;

setInterval (function() {

self.emit('tick'); // step 3

}, 1000) ;

};

With "events" you can create-, fire-, and listen for- your own events.

util.inherits (Ticker, EventEmitter); // step 4

Inherits methods from one function into another

module.exports = Ticker;

Creating an event emitter – Example (using Ticker)

```
// testingTicker
```

```
var Ticker = require("./myEmitter");
```

```
var ticker = new Ticker();
```

```
ticker.on ('tick', function() { // handler for 'tick' event  
    console.log("Tick");  
});
```

Servers

Simple servers

```
require('net')
```

```
createServer()
```

```
listen(port#)
```

```
'error'
```

```
'connection'
```

```
'data'
```

```
'close'
```

HTTP servers

```
require('http')
```

```
createServer()
```

```
listen(port#)
```

```
'request'
```

```
req.on 'data'
```

- net module provides an asynchronous network API for creating stream-based TCP servers

Use Node.js for ...

- Chat/Messaging
 - Real-time Applications
 - Intelligent Proxies
 - High Concurrency Applications
 - Communication Hubs
 - Coordinators
 -
- Web application
 - Websocket server
 - Ad server
 - Proxy server
 - Streaming server
 - Fast file upload client
 - Any Real-time data apps
 - Anything with high I/O
 -

Literature – Node.js

- <https://nodejs.org/en/>
- <https://www.w3schools.com/nodejs/default.asp>
- <https://www.tutorialspoint.com/nodejs/index.htm>
- <https://npmjs.org/>

FRAMEWORKS AND API

Libraries and frameworks for Node.js & JavaScript (1)

- Chrome DevTools
 - Debugging JavaScript
 - Performance analysis
 - Chromium project: open-source projects behind the Google Chrome browser and Google Chrome OS
- **NPM** is a package manager for Node.js packages, or modules

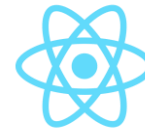


- npm is the package manager for JavaScript
- world's largest software registry:

<https://www.npmjs.com/>

Libraries and frameworks for Node.js & JavaScript (2)

- Libraries:
 - **React.js:** JavaScript library for building user interfaces by Facebook
 - **Angular/Angular.js:** TypeScript-based Javascript framework by Google
 - **Vue.js:** rapidly growing JS frameworks
- Develop across all platforms
 - ➔ Progressive Web App (PWA)
- Turn websites into native phone and desktop applications



Next Step: Progressive Web App – PWA (1)

- **PWAs are web applications** that can appear to the user like **traditional applications** or **native mobile applications!**
 - Combines features offered by browsers with the benefits of a mobile experience
 - Let users upgrade web apps to progressive web applications in their native OS
- Native Apps: coded in a programming language like Java
- Traditional Web Apps: coded in standard HTML, CSS, and JavaScript

Next Step: Progressive Web App – PWA (2)

- **PWAs:** visit in a browser tab, no install required!
 - Visit the site, add to home screen,
 - Go to home screen and open site, use the app!
 - Supported by Google Chrome and Mozilla Firefox

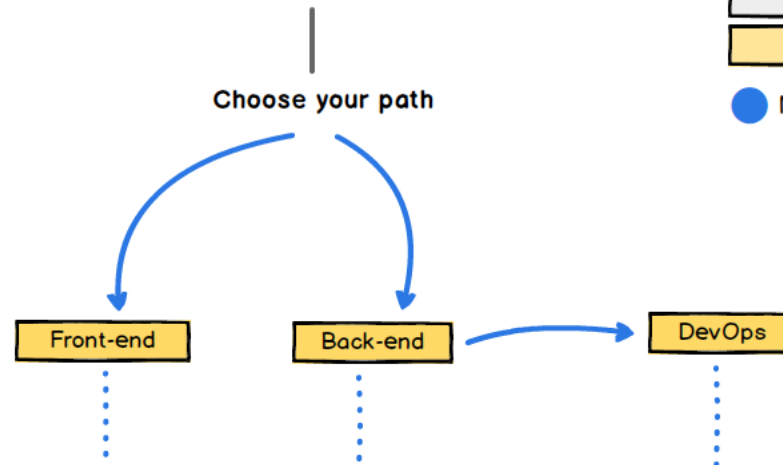


The 2018 Web Developer Roadmap (1)

Required for any path

Git - Version Control
SSH
HTTP/HTTPS and APIs
Basic Terminal Usage
Learn to Research
Data Structures & Algorithms
Character Encodings
Design Patterns
GitHub
Create a profile. Explore relevant open source projects. Make a habit of looking under the hood of projects you like. Create and contribute to open source projects.

Web Developer in 2018

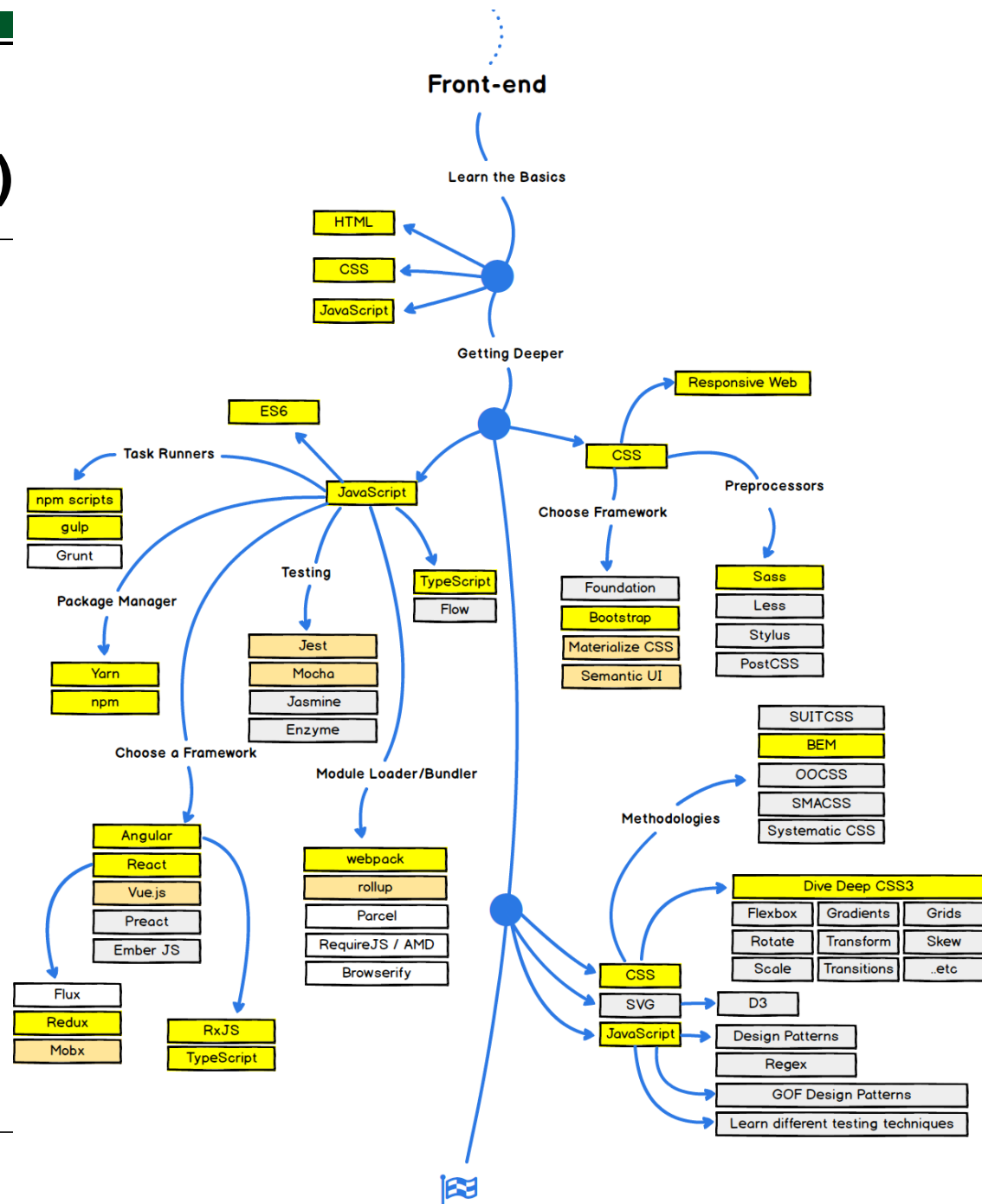


Legends

Personal Recommendation!
Possibilities
Pick any!
Now build something

Source: <https://codeburst.io/the-2018-web-developer-roadmap-826b1b806e8d>

The 2018 Web Developer Roadmap (2)





Laravel – The PHP Framework

- What is it?
 - A framework
- Why learn it?
 - Organize code into appropriate folders (modular development)
 - Use conventions for quick development + maintenance
 - Access libraries and utilities, authentication, etc.

Laravel – MVC Framework

1. Submit User Request

2. Route to appropriate Laravel Controller

Routing

Controller

3. Interact with Data Model

4. Controller invokes results View

View

Model

5. Render view in users browser



Database



Laravel – Main files and locations

./.env

./app/Http/routes.php

1 - routes

./app/Http/Controllers/Auth/AuthController.php

./app/Http/Controllers/Auth/PasswordController.php

./app/Http/Controllers/Controller.php

./app/Http/Controllers/MyController.php

2 -
controllers

./app/TestTable.php

./app/User.php

3 - model

./config/app.php

./config/database.php

./config/view.php

4 - config

./public/.htaccess

./public/index.php

./public/web.config

5 - website

./resources/views/errors/503.blade.php

./resources/views/welcome.blade.php

6 - views

./tests/ExampleTest.php

./tests/TestCase.php

7 - tests

Routes

Way to redirect the request (or URL) from the client.

In the below example, requests to home will return a welcome view.

```
<?php
Route::get('/', function () {
    return view('welcome');
});
```

1

```
...
Route::get('/myStuff', 'MyController@getTableData');
```

2

Controllers

1. This is where the main work takes place
2. It may make calls to the database
3. It may make calls to some processing modules
4. It may make calls to create some views
5. and then returns the results back to the client.

Controllers – Example

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Foundation\Bus\DispatchesJobs;  
use Illuminate\Routing\Controller as BaseController;  
use Illuminate\Foundation\Validation\ValidatesRequests;  
use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
```

```
class MyController extends BaseController  
{  
    public function getTableData(){  
        echo json_encode(\App\testTable::all());  
    }  
}
```

Eloquent ORM – Object Relational Mapper

- Provides a simple ActiveRecord implementation for working with a database.
- Each database table has a corresponding "Model" which is used to interact with that table
 - Models data
 - Simple access to data

Model – Example

```
<?php

namespace App;

use Illuminate\Foundation\Auth\User as Authenticatable;

class TestTable extends Authenticatable
{
    protected $table = 'testTable';
    public $timestamps = false;
}
```


Configuration file (.env file) – Example

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=2uUDtMdi5ldsgHNQjdHQAyUwRHgWfjmW
APP_URL=http://localhost
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=testSchema
DB_USERNAME=testuser
DB_PASSWORD=testuser
```

```
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync
```

```
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

```
MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
```

in config/database.php

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Additional Tools

- Blade: UI templating tool
- Artisan: Utility tool to help developer
 - Tinker: Tool to help developer run and try methods on server side
 - Laravel artisan's tinker: a **repl (read-eval-print loop)**
 - An interactive language shell
 - It takes in a single user input, evaluates it, and returns the result to the user
 - Quick and easy way to see the data in your database:

<https://scotch.io/tutorials/tinker-with-the-data-in-your-laravel-apps-with-php-artisan-tinker>

```
// find a specific user and see their attributes  
App\User::where('username', 'samuel')->first();
```

```
// see the count of all users  
App\User::count();
```

- Composer: Dependency manager for php (similar to make tool)

Literature – LARAVEL

- Documentation
 - See <https://laravel.com/docs/5.3/> (incl. installation)
- BEST RESOURCE FOR LEARNING LARAVEL
 - <https://laracasts.com/series/laravel-5-from-scratch> (there are 18 screencasts walking you thru the materials)

EVENT-DRIVEN PROGRAMMING WITH JAVAFX

Graphical User Interfaces with JavaFX

- Design and build graphical user interfaces (GUI)
- Event-driven software development
- Can use large libraries
- Good practices in user-interface development!
- Possibility to deploy in the Browser
 - There is a JavaScript API (Java Plugin) to simplify web deployment of JavaFX applications
 - Use JavaFX for rich client applications with a lot of interactive elements

Graphical user interfaces in Java

- There are several interfaces in Java for creating GUIs
- In Java integrated (supplied) interfaces:
 - **A**bstract **W**indow **T**oolkit (AWT)
 - Swing
 - JavaFX
- Alternative interfaces (not integrated in Java):
 - **S**tandard **W**idget **T**oolkit (SWT)
 - Uses GUI components from the underlying platform and tries to combine the best of AWT and Swing.
 - Efficiency issues on non-Windows platforms due to missing features (SWT library must be included with the application).
 - JFace (extension of SWT)

Light vs. heavyweight components

- **Lightweight components**

- Are not bound to platform-specific component.
- Must be drawn ultimately on a heavyweight component.
- Look the same on all platforms.
- Emulating the look of the target platform is expensive.

- **Heavyweight components**

- Are bound to a platform-specific component.
- Only the intersection of the components available on all target platforms is usable
- Components not offered on the platform need to be recreated "by hand".

Abstract Window Toolkit (heavyweight)

- **Uses** the GUI components offered by the **underlying platform**.
- There are only GUI components that exist on all platforms supported by AWT.
- Creation of complex GUI very expensive: GUI components, such as progress bar, must be created by hand.
- But: (almost) just as **fast** GUI as in native applications.

Swing and JavaFX (lightweight)

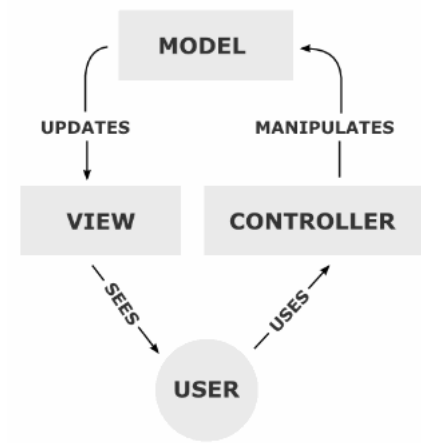
- Uses only **windows** and **drawing** operations of the underlying platform.
- Swing applications are **resource-hungry** and often **slower** than to use AWT or other platform-specific applications.
- This problem is often intensified by awkward programming:
 - E.g. **JFileChooser** (Swing): Rebuild each time instead of reconfiguring only
 - **JFileChooser** provides a simple mechanism for the user to choose a file.

JavaFX

- Should replace Swing in the medium-term
- Better support for devices with very different graphical display options (smartphone vs. desktop)
- Supports better the **Model/View/Controller** architecture pattern than Swing by separating the program logic (written in Java) from the GUI (written in FXML).
 - FXML is an XML-based language that provides the structure for building a user interface separate from the application logic of your code.
- Supports and uses DirectX and OpenGL

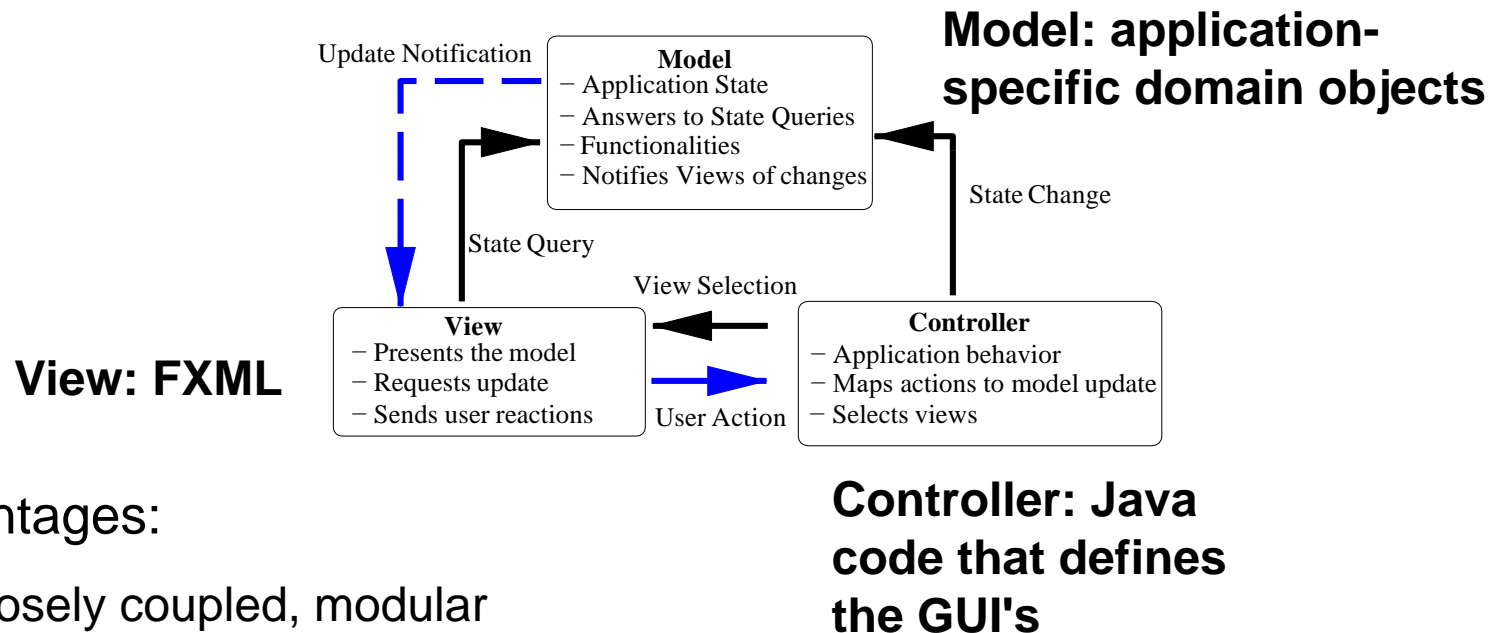
Recap: Model/View/Controller (MVC)

- MVC is a class combination (components) for constructing user interfaces (first created in Smalltalk programming language).
 - Model: Application object (data and the state of the object)
 - View: Display of the model on the screen (possibly several times)
 - Controller: Defines user interface response to inputs (event reaction)
- More in the section "Architectural Styles"



MVC in JavaFX

- The central theme of JavaFX: Model-View-Controller architecture:



- Advantages:
 - Loosely coupled, modular
 - Model with different views
 - Controller decides when/how to update the model and/or the view
 - Model can change the view

MVC in JavaFX

- **Enforce Model-View-Controller (MVC) with FXML**
 - The "Model" consists of application-specific domain objects
 - The "View" consists of FXML
 - The "Controller" is Java code that defines the GUI's behavior for interacting with the user.

Sample application in Swing

```
import java.awt.event.*;  
import javax.swing.*;
```

Import the required packages

```
public class HelloWorld extends JFrame {
```

Inherits from the main window class

```
    public HelloWorld() {
```

Call constructor of JFrame with the title of the window.

```
        super("Hello world!");
```

```
        JButton button = new JButton("01 + 01 = ?");
```

Create a button with the specified caption.

```
        button.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                JOptionPane.showMessageDialog(null, "01 + 01 = 10",  
                    "Answer:", JOptionPane.QUESTION_MESSAGE);  
            }  
        });
```

Specify what should happen on an action on the button.

```
    });
```

Add a button to the window.

```
    this.add(button);
```

```
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Specify the action to be triggered when the window is closed.

```
    this.pack();
```

Adjust the window size, adapt the arrangement of the components to the window size.

```
}
```

```
public static void main(String[] args) {
```

```
    HelloWorld window = new HelloWorld();
```

```
    window.setVisible(true);
```

Displays the created window.

```
}
```

```
}
```

Sample application in JavaFX

```
public class HelloWorld extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Hello world!");  
        Button btn = new Button();  
        btn.setText("01 + 01 = ?");  
        btn.setOnAction(new EventHandler<ActionEvent>() {  
            @Override  
            public void handle(ActionEvent event) {  
                Stage dialogStage = new Stage();  
                dialogStage.initModality(Modality.WINDOW_MODAL);  
                VBox vbox = new VBox(); vbox.setAlignment(Pos.CENTER);  
                vbox.getChildren().add(new Text("Answer:"));  
                vbox.getChildren().add(new Button("01 + 01 = 10"));  
                dialogStage.setScene(new Scene(vbox));  
            }  
        });  
        StackPane root = new StackPane();  
        root.getChildren().add(btn);  
        primaryStage.setScene(new Scene(root, 300, 250));  
        primaryStage.show();  
    }  
}
```

Inherits from the main window class

Start the application
Note: Call launch(), JavaFX will call start().

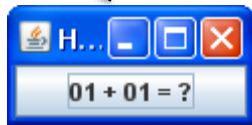
Create a button with the specified caption.

Specify what should happen on an action on the button.

Add a button to the window.

Sample application: Appearance

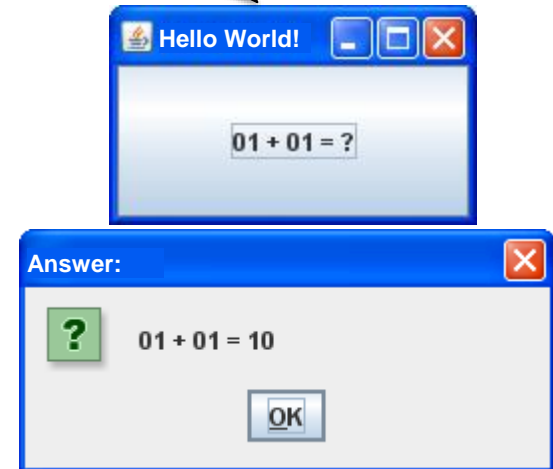
After the start.



After manual enlargement.



After clicking on the button.



Creating windows: Stage

- Windows are usually created in JavaFX with the class **Stage**.
- Then you can set additional properties of the window and place GUI components.

Creating windows: Stage – Example (1)

Stage holds a **Scene**, the Scene holds a root or main container:

```
@Override
public void start(Stage stage)
{
    // Set the title of the Stage
    stage.setTitle("Stage with a Button in the Scene");
    // Create the Button
    Button button = new Button("Hello");
    // Create the VBox
    VBox root = new VBox(button);
    // Create the Scene
    Scene scene = new Scene(root, 200, 100);
    // Add the Scene to the Stage
    stage.setScene(scene);
    // Set the width and height of the Stage
    stage.setWidth(400);
    stage.setHeight(100);
    // Display the Stage
    stage.show();
}
```

VBox component is a layout component (container) which positions all its child nodes

Source: <https://examples.javacodegeeks.com/desktop-java/javafx/javafx-stage-example/>

Creating windows: Stage – Example (2)

- Resulted GUI:



Source: <https://examples.javacodegeeks.com/desktop-java/javafx/javafx-stage-example/>

Structure of a dialog window

- **Stage** holds a **Scene**, the Scene holds a root or main container, like a Flow, Grid, Border, Stack or Anchor **pane**.



The root container holds everything else, which could include other nested containers.

Source: <https://examples.javacodegeeks.com/desktop-java/javafx/dialog-javafx/javafx-dialog-example/>

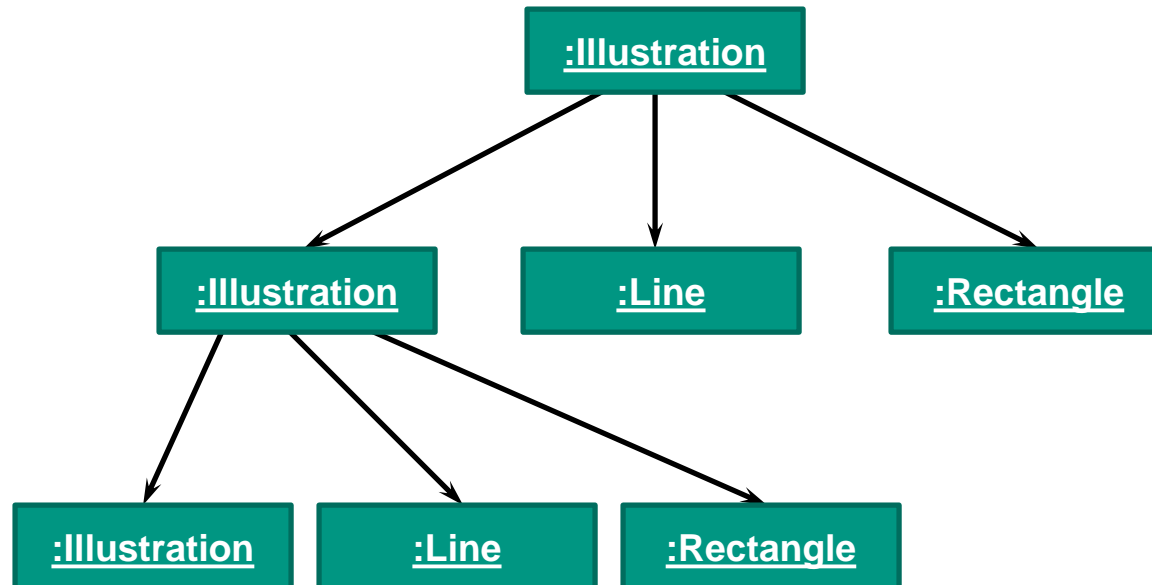
- Stage → Scene → Pane (Panels can be nested), button, labeling, text box , ...

Design pattern "Composite"

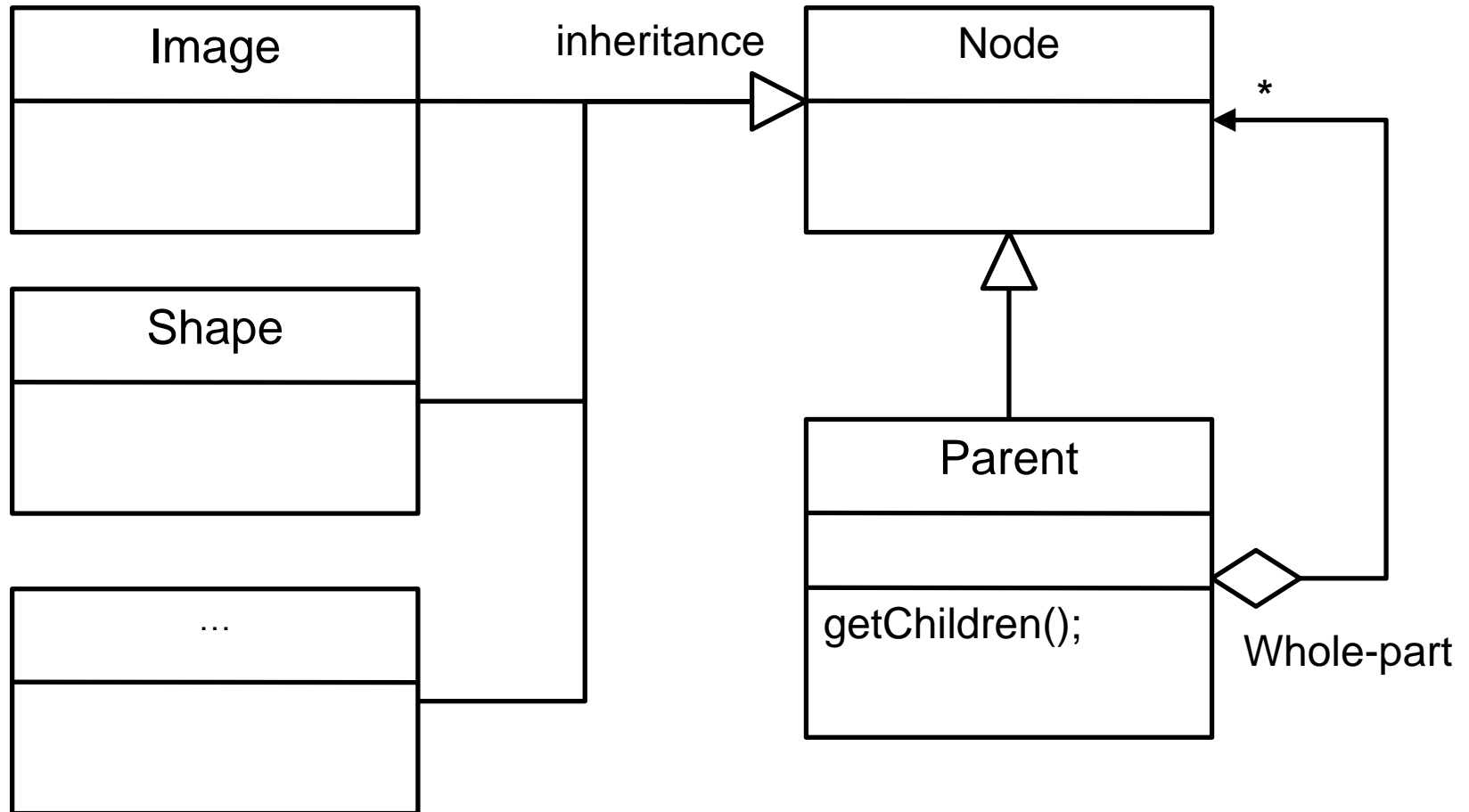
- JavaFX uses the "composite" design pattern when composing UIs.
- Composite design pattern
 - Merge objects into **tree structures** to represent whole-part hierarchies. The pattern allows clients to uniformly treat both individual objects and aggregates.
 - Composite is used to join objects into tree structures.
- It enables **uniform handling** of GUI components (e.g. `javafx.scene.control.CheckBox`) and their compositions.
- It is easy to add more GUI components.

Composite: Example

- Combined graphics objects
- Common operations: **draw()** , **move()** , **delete()** , **scale()**



Design pattern "Composite" in JavaFX



- Parent (composite) could have many nodes (components)

Everything is a Node!

- All GUI components in JavaFX inherit from class **Node**.
- GUI components can be nested using the **Parent** composite.
- For this reason, it is possible to add more GUI components to each JavaFX GUI component.
- **Example:** text box inside a button



GUI components in JavaFX

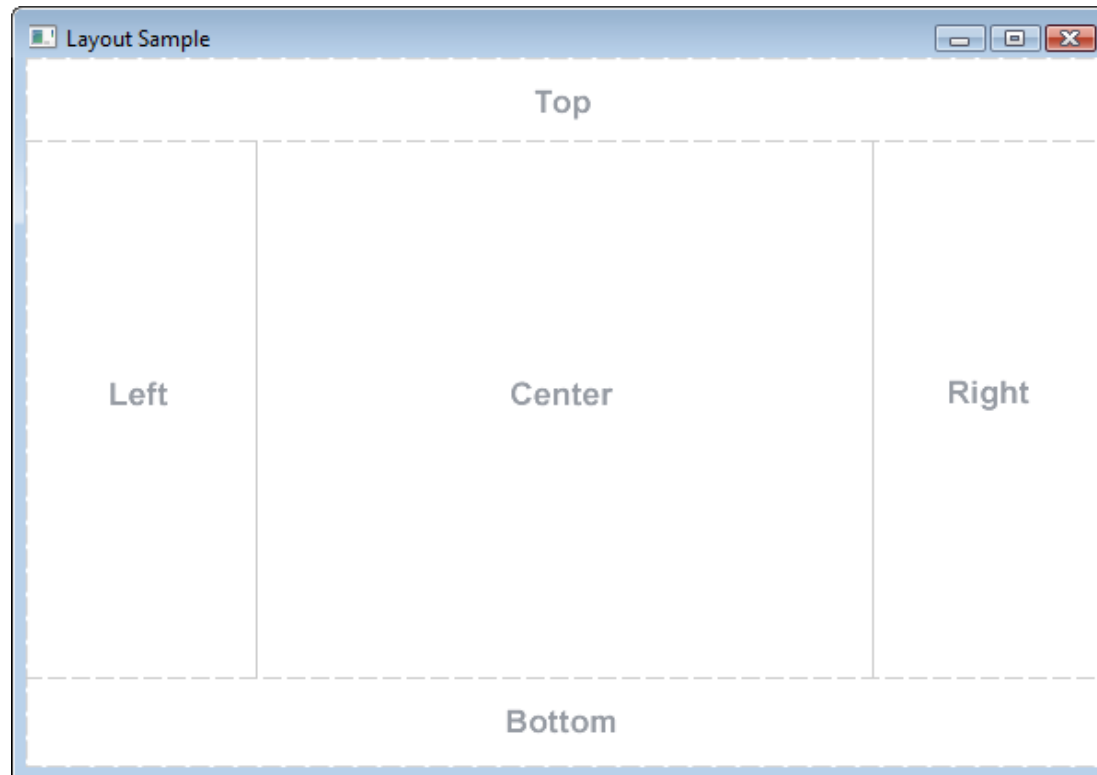
- JavaFX provides a variety of GUI components.
- With a few exceptions (e.g. the windows) all GUI component are lightweight.
- GUI components are added to a node using `getChildren().add(Node)` method.
- For the arrangement of components on a container JavaFX provides different **Panes** ready.

Panes with arrangement (order)

- These include simple placement panes, such as the **FlowPane**, where the GUI components are arranged row by row until the entire width of the window is exploited, and then the GUI components fill the next row.
- By nesting of several arrangement panes complex arrangements of GUI components are possible.
 - **Important:** The developer has to define the suitable panes arrangements.
 - Depending on the GUI component, JavaFX gives different arrangements as default.
 - **Example:** **JPane1** uses **FlowLayout**, where a **JContentPane** uses **BorderLayout**
- <http://docs.oracle.com/javafx/2/api/javafx/scene/layout/Pane.html>
- http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

Panes with arrangement (order) – Example

- **BorderPane:** The BorderPane layout pane provides five regions in which to place nodes: top, bottom, left, right, and center.



What's next?

- So far, you can create a pane using **nodes**, **placement panes**, and **GUI components**.
- But how do you know if the user clicked a button or moved the mouse?
- Java offers **events** and **observers (listeners)** → event handling
 - **Event-driven programming** → **Event-driven architecture**

Events in JavaFX: Two types

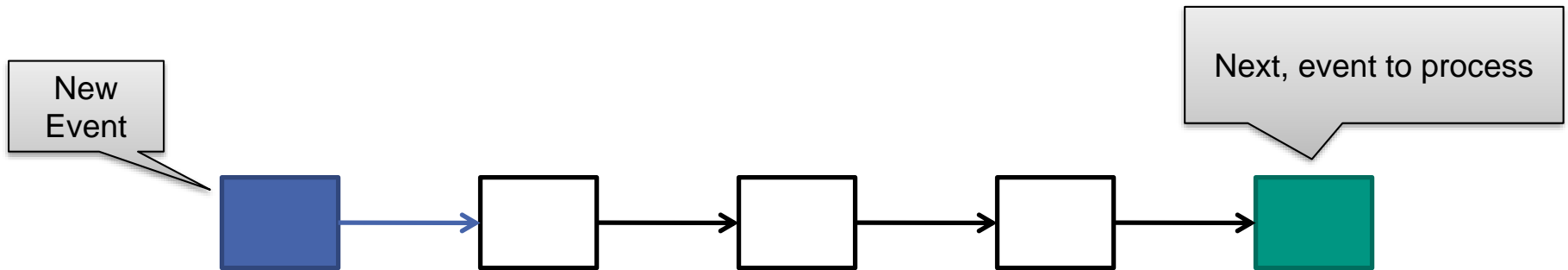
- Low-level events (primitive)
 - Events at the level of the operating system
 - **Examples:** Mouse movement, focus on a component
- High-level events (semantic)
 - In JavaFX, these events are often triggered by primitive events that are transformed into semantic events.
 - **Examples:** Click on a button, highlight text in a text box

Handling events in Java

1. User moves the mouse, presses the keyboard
2. The operating system then generates a low-level event (mouse movement Δx and Δy , which button was pressed) and passes it to the Java platform event handler.
3. Event handler transforms the low-level event into an event object (**EventObject** or subclass of it).
 - The event object contains at least the **event source**, i.e. the object to which the event is attributed.
 - For graphical user interfaces, this is usually a graphical object.

Handling events in Java – Event-driven architecture

4. Event handler appends the event to the **event list** (a queue).



5. The JavaFX GUI-process periodically checks for new events in the event list and handles one event after another:

- Pick up event
- Read event source
- Observers (listeners) are registered at the event source and waiting for the event
 - They determine the appropriate reactor method and call it with the event object as a parameter.

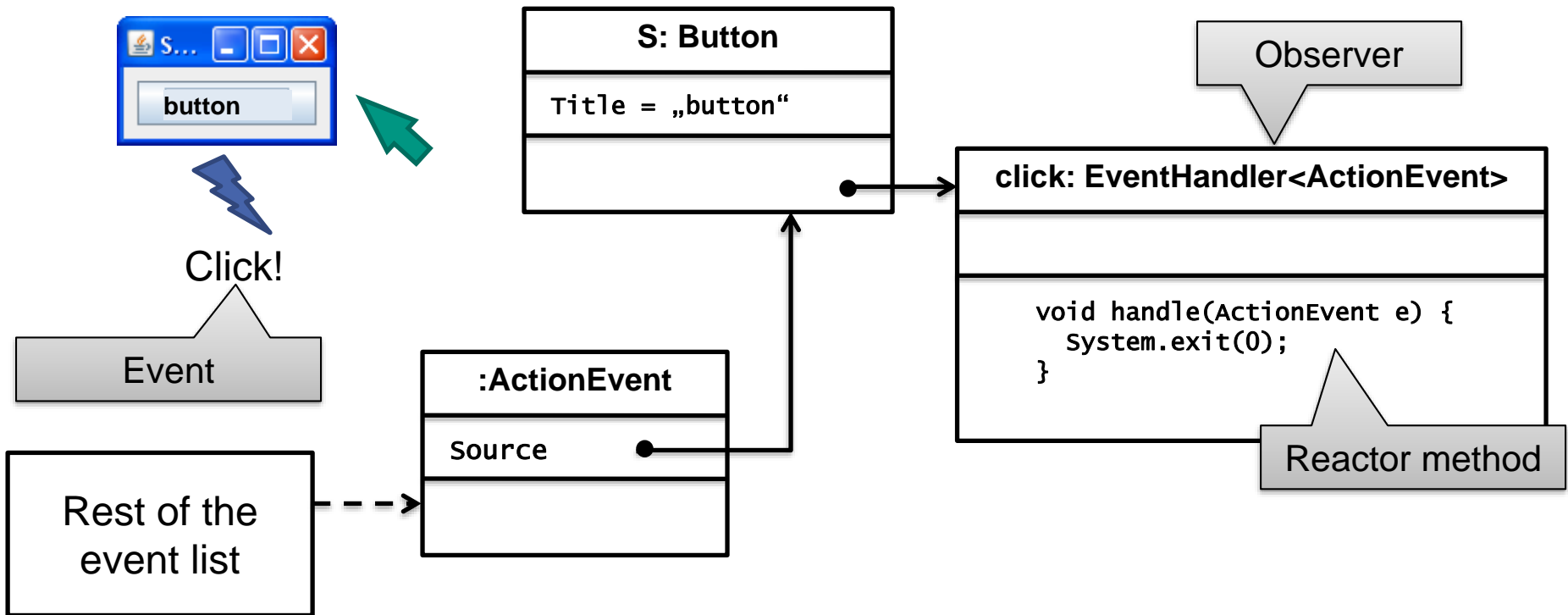
Difference to swing!

Handling Events in Java: The Event List (Queue)

- The event handler can combine multiple events in the event list.
 - **Example:** Several mouse movements in a row can be combined into one event.
- The event list can also be manipulated from your own program.
- The event list buffers events if they can not be handled fast enough.

Handling Events in Java: Passing an Event Object

- The handling of the event is delegated from the event source to the observers (listeners).



Handling Events in Java: Advantages of Delegation

- Graphical **representation** and **application core** can be **separated** cleanly.
 - Enables event handling to be handled by objects other than the ones that generate the events.
- The assignment of events is very simple
 - Event source is determined automatically
- Performs much better in applications where more events are generated.
 - Does not have to repeatedly process unhandled events

Events in JavaFX

- Almost every JavaFX component provides events
 - The events triggered by the component can be intercepted and reused by the developer
- The most important event classes are:
 - **DragEvent**: move, enlarge, ... the component
 - **KeyEvent**: key pressed, key released, ...
 - **MouseEvent**: mouse button pressed, mouse is moved ...
 - **ContextMenuEvent**: context menu requested
 - **GestureEvent**: gesture input
 - **InputMethodEvent**: Text input is made or changed
 - **TouchEvent**: touch (e.g. on touch screen)

Events in JavaFX (2)

- Depending on the GUI component, various events are triggered, which can be picked up by an observer (listener).
- Depending on the type of event to be intercepted, different observers (listener) are needed.

Events in JavaFX – Listeners

- The main listeners (observers) are:
 - The **ActionListener** responds to any ActionEvent triggered by the user.
 - **Example:** The user clicks a button.
 - The **ChangeListener** responds to changes to the associated object.
 - **Example:** The user changes the value of a scroll bar.
 - The **KeyListener** responds to the user's keystrokes.
 - **Example:** The user presses the keyboard shortcut [Ctrl] + [Esc]
 - The **MouseListener** responds to actions that are taken with the mouse.
 - **Example:** The user moves the mouse into a window.
 - The **WindowListener** responds to changes in the window state.
 - **Example:** The user minimizes the window.

Event filter

- Event filters control the flow of events to nodes.
- Nodes register event filters.
- Event filters implement the **EventHandler** interface.

Registering an event filter

```
EventHandler filter = new EventHandler(<InputEvent>() {  
    public void handle(InputEvent event) {  
        System.out.println(event.getEventType() + "consumed");  
        event.consume();  
    }  
} );
```

The two nodes use the same event filter.

```
myNode1.addEventFilter(MouseEvent.MOUSE_PRESSED, filter);  
myNode2.addEventFilter(MouseEvent.MOUSE_PRESSED, filter);
```

Sample application in JavaFX

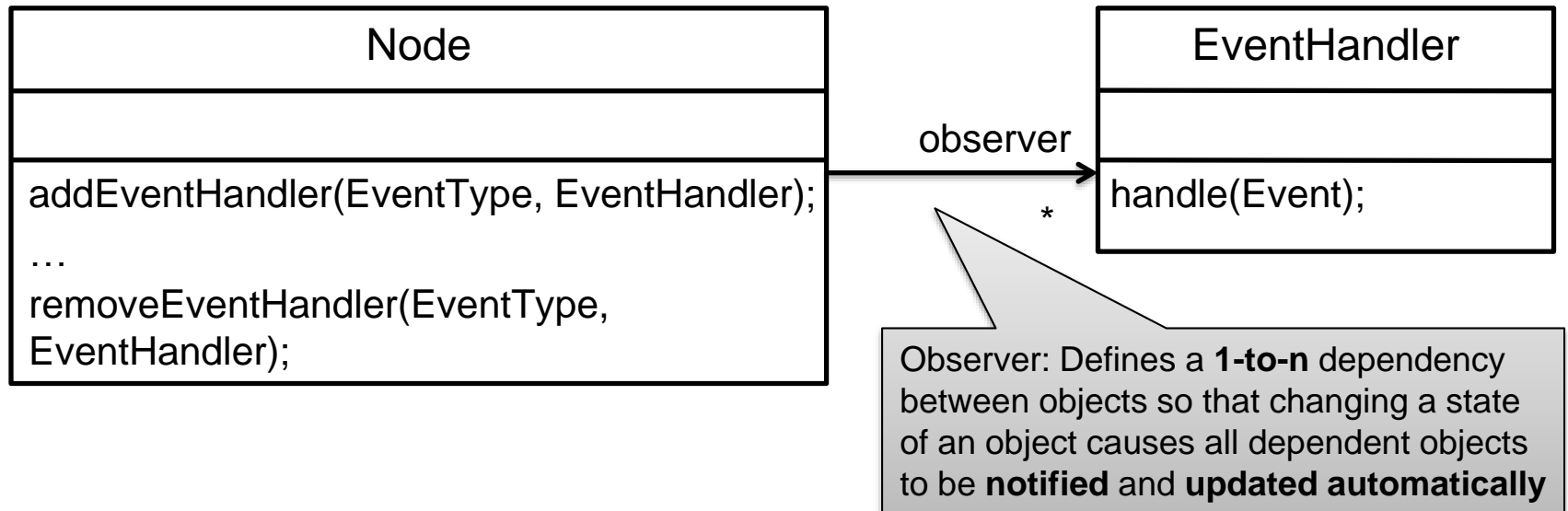
```
public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello world!");
        Button btn = new Button();
        btn.setText("01 + 01 = ?");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                Stage dialogStage = new Stage();
                dialogStage.initModality(Modality.WINDOW_MODAL);
                VBox vbox = new VBox(); vbox.setAlignment(Pos.CENTER);
                vbox.getChildren().add(new Text("Answer:"));
                vbox.getChildren().add(new Button("01 + 01 = 10"));
                dialogStage.setScene(new Scene(vbox));
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

Registering an event filter

Specify what should happen when an event arrives.

Events in JavaFX: The observer design pattern

- In JavaFX, the design pattern "observer" is used in slightly different
- **Example:** `javafx.scene.Node` and `KeyListener`

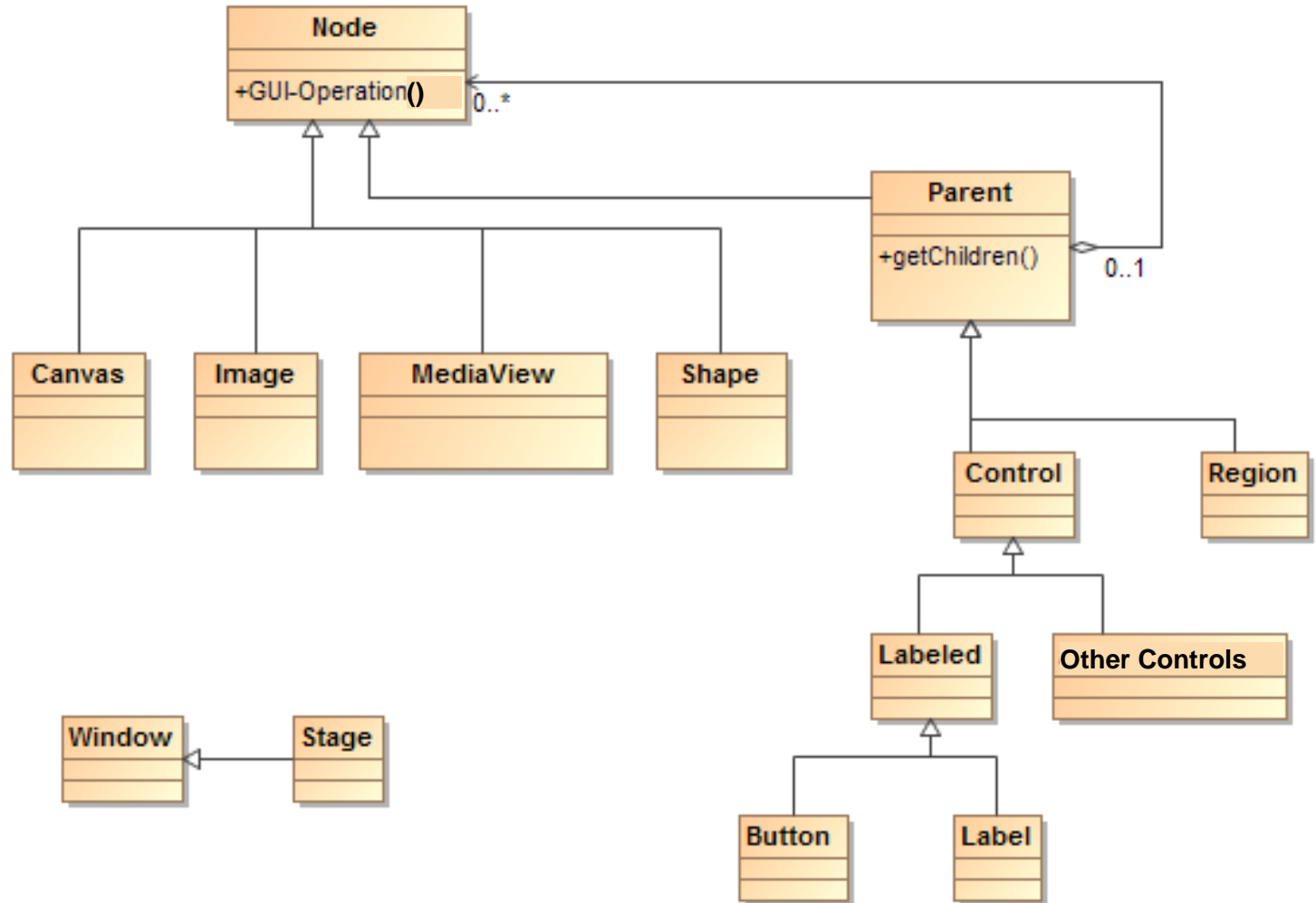


- When an event is taken from the event-list queue, for **all** registered observers the corresponding methods of the component are called, while the event is delivered.

Implementation of observers

- An observer can be implemented in Java in different ways.
- A separate observer class (also as an inner class)
 - `class MyObserver implements EventHandler { //... }`
- Use anonymous classes
 - `setOnAction(new EventHandler<ActionEvent>() {
 public void handle(ActionEvent e) { //... }
});`

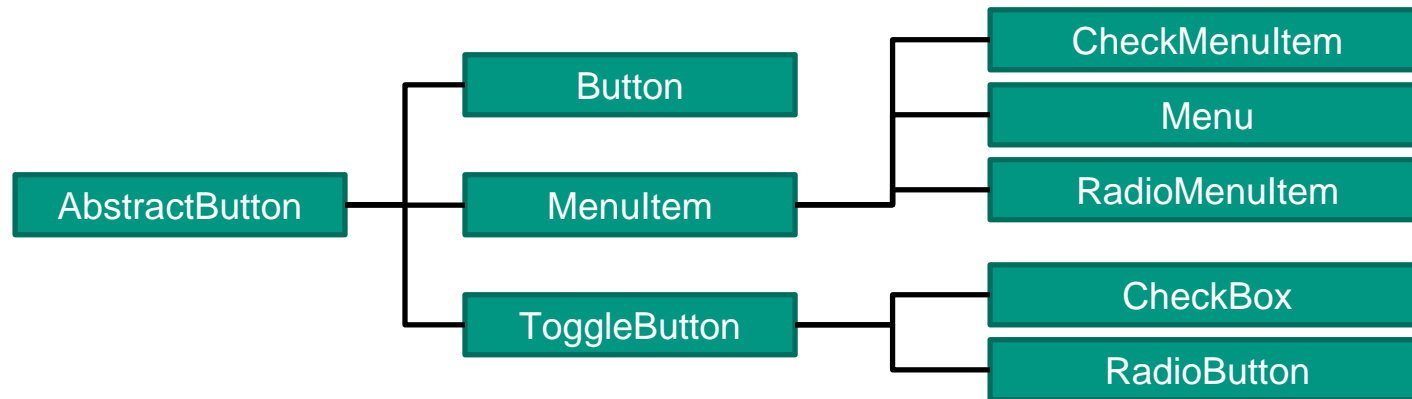
Components overview in JavaFX



Source: <http://docs.oracle.com/javafx/2/api/overview-tree.html>

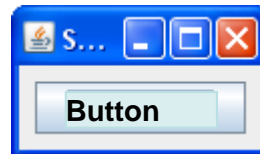
Buttons

- JavaFX offers several types of buttons:
 - **Simple** buttons (**Button**)
 - Buttons for **menus** (**MenuItem**)
 - With and without switchable status
 - Buttons with **switchable status** (**ToggleButton**)



Buttons: Button

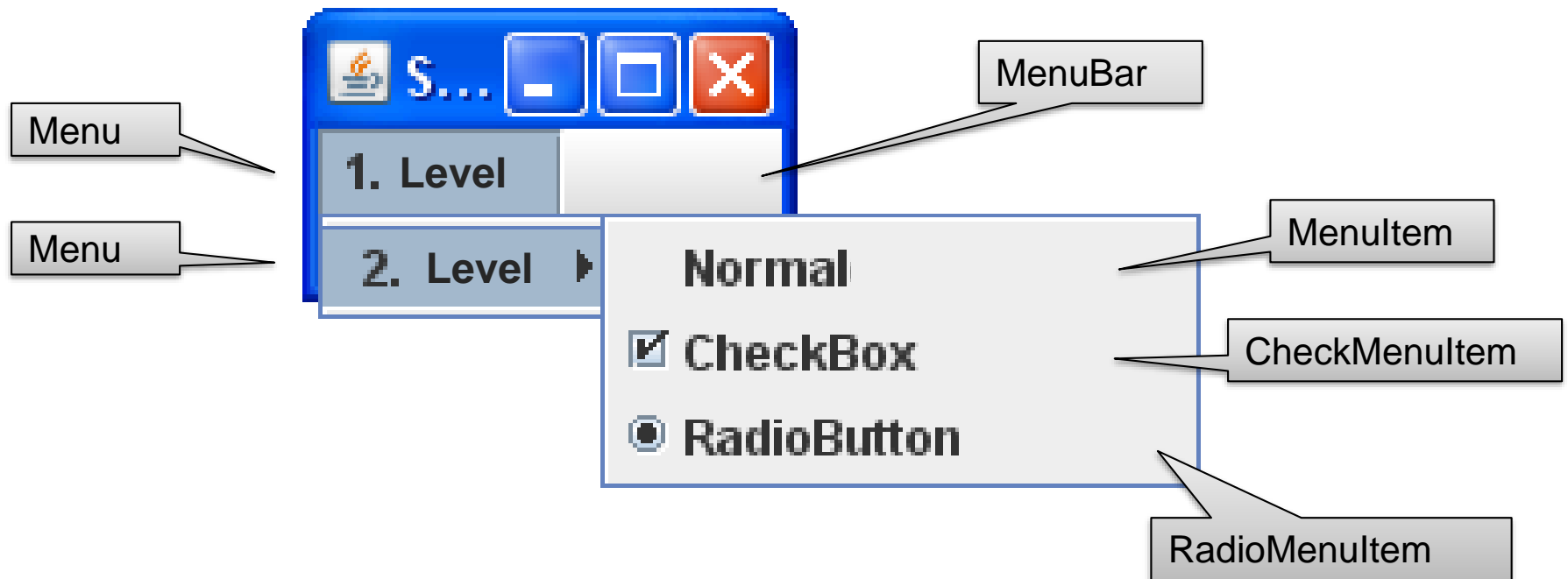
- Objects of class **Button** represent simple buttons.
- When a control of this class is activated (e.g. by clicking with the left mouse button), the button triggers an event (in this case an **ActionEvent**).



Menu entry: MenuItem (1)

- GUI objects of this class or derived classes represent an **entry** in a menu (e.g. **MenuBar**).
- Derived from this class
 - **CheckMenuItem**
 - **Menu**: A button that displays a pop-up menu when clicked
 - **RadioMenuItem**
- Objects of class **CheckMenuItem** and **RadioMenuItem** can accept the two states " Selected " or "Not selected".
- Objects of class **RadioMenuItem** are used if the offered options are **mutually exclusive**.

Menu entry: MenuItem (2)

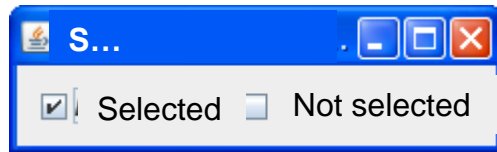


Switch: ToggleButton (1)

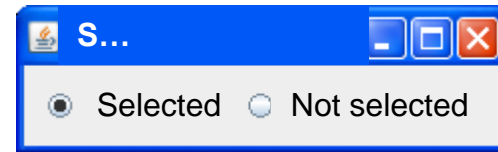
- GUI components of this type can be toggled between multiple states.
- The components inherit from this class
 - **Selection box:** `CheckBox`
 - **Button:** `RadioButton`
- Objects of the class `CheckBox` and `RadioButton` can assume the two states "Selected" or "Not selected".
- Objects of class `RadioButton` are used, if the offered options are mutually exclusive.

Switch: ToggleButton (2)

CheckBox



RadioButton



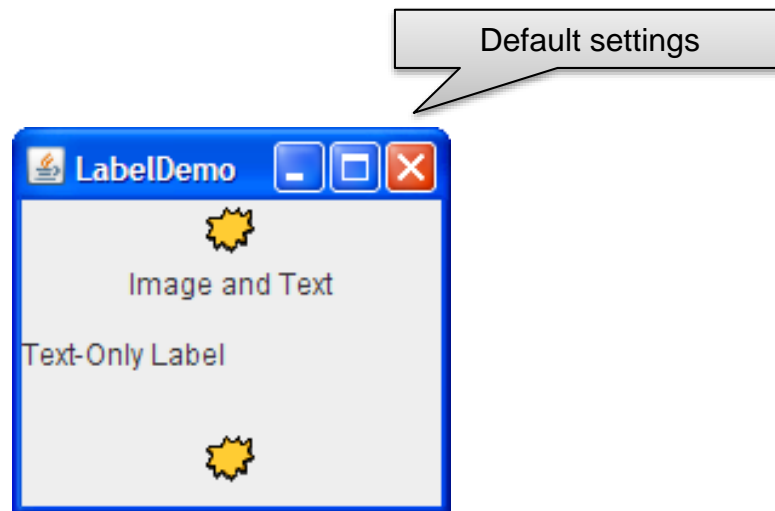
Note:

RadioButton-Component must be part of a **ButtonGroup**, which ensures mutual exclusion.

```
ToggleGroup tg = new ToggleGroup();  
  
RadioButton rb1 =  
new RadioButton("Selected", true);  
RadioButton rb2 =  
new RadioButton("Not selected",  
false);  
  
rb1.setToggleGroup(tg);  
rb2.setToggleGroup(tg);
```

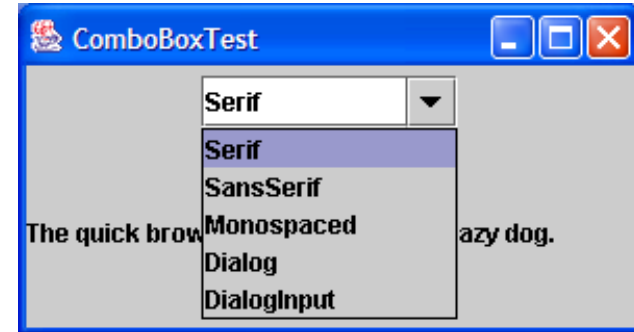

Labels: Label

- `Label` can be used to display an image, a short text or to display text and image.
- It can be specified how text and image are aligned in the `Label`.
- Often used for the representation of table cells.



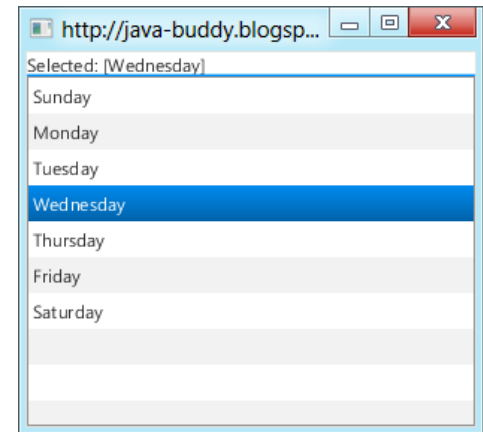
Selection list: ComboBox

- A **ComboBox** consists of a button or editable field and a drop-down list.
- In a **ComboBox** , only one entry can be selected at a time.
- In **ComboBox** , an entry can be selected by pressing a button.
- The developer can specify whether the user is allowed to enter his own values in addition to the values in the selection list.
- Access to the selected entry via `getSelectionMode()`, `GetSelectedItem()`



List view: ListView

- In GUI components of the `ListView` class, the user can select a particular subset of entries from a given set of entries.
- The developer can define how a selection of entries may look like.
- Entries in a `ListView` can be edited
- A `ListView` does not provide methods for adding, inserting, or deleting individual entries. For all at once this can be done with the method `ListView.setItems (ObservableList <>) .`



Text fields: `TextInputControl` (1)

- `TextInputControl` GUI components, or controls derived from them, allow users to enter text.
- The components inherit from this class
 - `Textarea`
 - Allows **multi-line** input with a font
 - `TextField`
 - Allows **a line of text** with a font
 - The `PasswordField` control is used to enter a one-line password.

Text fields: TextInputControl (2)



TextField



PasswordField



TextArea

Images: ImageView

```
Image image = new Image("flower.png");  
ImageView iv1 = new ImageView();  
iv1.setImage(image);
```

Load picture in
JavaFX picture
frame

```
Group root = new Group();  
Scene scene = new Scene(root);  
HBox box = new HBox();
```

Create window with
arrangement

```
box.getChildren().add(iv1);  
root.getChildren().add(box);  
stage.setScene(scene);
```

Put picture frames
on the window

BufferedImage (AWT) vs. Image (JavaFX)

- ... are different data structures!
- ... are interconvertible via
`javafx.embed.swing.SwingFXUtils`
- ... have different ways of getting an image file into the data structure:
 - BufferedImage: Can not do that by "itself"; needs e.g. methods from `javax.imageio.ImageI()`.
 - Image (JavaFX): Can read image files by itself, e.g. In the constructor.

Self-Study: JavaFX

- The following sections of the Java documentation may be of interest to you:
- `javafx.stage.Stage`
- `javafx.event.ActionEvent` , `javafx.event.EventHandler`
- `javafx.scene.layout.BorderPane` ,
`javafx.scene.layout.FlowPane` , ...
- `javafx.embed.swing.JFXPanel` ,
`javafx.scene.control.Button` , ...
- Code samples often provide helpful information if the Java documentation does not work.

Self-Study: Java API

- Refer to links provided in the references (next pages):
 - Description of the provided functionality (possibly with code examples).
 - Overview of all provided constants, constructors and methods.
 - Description of constants, constructors and methods, as well as their parameters and return values.

Literature – JavaFX

- Information about JavaFX in the latest Java documentation:
<http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>
- A tutorial on JavaFX can be found at:
http://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm
- Many helpful code examples: <http://www.java2s.com/>
- Documentation of the Java API:
<http://download.oracle.com/javase/8/docs/>



Summary

- Event-Driven Programming
 - Event handling
 - Event-driven architecture
 - Asynchronous programming, etc.
 - Event-Driven Programming with JavaScript (Node.js)
 - Frameworks and API
 - Web development roadmap
 - Event-Driven Programming with JavaFX