Index

[Node.js 3](#_Toc478736070)

[Node.js Application 5](#_Toc478736071)

[npm 5](#_Toc478736072)

[REPL Terminal 6](#_Toc478736073)

[package.json 6](#_Toc478736074)

[Callback 6](#_Toc478736075)

[Event Driven Programming 7](#_Toc478736076)

[EventEmitter 7](#_Toc478736077)

[Difference between Callbacks and Events 8](#_Toc478736078)

[Buffers 8](#_Toc478736079)

[Streams 9](#_Toc478736080)

[File System 10](#_Toc478736081)

[Global Objects 12](#_Toc478736082)

[Utility Modules 13](#_Toc478736083)

[Web Module 13](#_Toc478736084)

[Express Framework 13](#_Toc478736085)

[JSON 14](#_Toc478736086)

[Web Server 14](#_Toc478736087)

[Web Application Architecture 14](#_Toc478736088)

[Web Service 14](#_Toc478736089)

[RESTful Web Services 15](#_Toc478736090)

[REST Architecture 15](#_Toc478736091)

[Five I/O Models 15](#_Toc478736092)

[Framework 15](#_Toc478736093)

[Differences with normal libraries 16](#_Toc478736094)

[MVC 16](#_Toc478736095)

[Observer Pattern 16](#_Toc478736096)

[Gulp 17](#_Toc478736097)

[Build System 17](#_Toc478736098)

[AngularJS 18](#_Toc478736099)

[Directive 18](#_Toc478736100)

[Custom Directive 18](#_Toc478736101)

[Module 19](#_Toc478736102)

[Programming Questions 20](#_Toc478736103)

[UI Choices 20](#_Toc478736104)

[Angular Material (for Angular1) 20](#_Toc478736105)

[PrimerNG (for Angular2) 20](#_Toc478736106)

[Shell Script 20](#_Toc478736107)

[Commands 21](#_Toc478736108)

[Both 21](#_Toc478736109)

[Bash 21](#_Toc478736110)

[Command Prompt 22](#_Toc478736111)

[AWK 23](#_Toc478736112)

[Java 23](#_Toc478736113)

[Java Servlet 24](#_Toc478736114)

[Apache Tomcat 24](#_Toc478736115)

[Catalina 24](#_Toc478736116)

[Apache Maven 24](#_Toc478736117)

[Database 24](#_Toc478736118)

[SQL 24](#_Toc478736119)

[CRUD 24](#_Toc478736120)

[ECMAScript 25](#_Toc478736121)

[NoSQL database 25](#_Toc478736122)

[MongoDB 25](#_Toc478736123)

[Highchart 26](#_Toc478736124)

[Supported Chart Types 26](#_Toc478736125)

[Course Notes 27](#_Toc478736126)

[Computational Thinking 27](#_Toc478736127)

[Design of Computer Programs 27](#_Toc478736128)

[Technical Interview Questions 27](#_Toc478736129)

[Java Core 27](#_Toc478736130)

[1. HashTable v.s. HashMap? 27](#_Toc478736131)

[2. Synchronized? 28](#_Toc478736132)

[3. StringBuilder v.s. StringBuffer? 28](#_Toc478736133)

[4. CheckedException v.s. RuntimeException (unchecked exception)? 28](#_Toc478736134)

[5. ArrayList v.s. LinkedList? 28](#_Toc478736135)

[6. Abstract Class v.s. Interface? 29](#_Toc478736136)

[7. What is Java generic type? Example? 29](#_Toc478736137)

[8. How to create a new thread? 30](#_Toc478736138)

[9. Explain any key interface under concurrent package? 30](#_Toc478736139)

[10. Concurrency? 30](#_Toc478736140)

[11. Concurrent Programming 30](#_Toc478736141)

[12. ReentrantLock v.s. Synchronization? 31](#_Toc478736142)

[13. What is try-with-resources statement? Example? 31](#_Toc478736143)

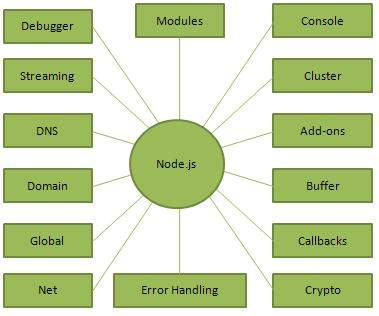
[XML 31](#_Toc478736144)

[Useful Links: 31](#_Toc478736145)

[Books 31](#_Toc478736146)

[Learning 31](#_Toc478736147)

# Node.js

* Developed by Ryan Dahl in 2009
* A very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine.
* Used to develop
  + I/O intensive web apps
  + Data Streaming Apps (E.g. video streaming sites)
  + Data Intensive Real Time Apps (DIRT)
  + JSON APIs based Apps
  + Single Page Apps
* NOT advised to use for: CPU intensive apps
* JavaScript that is executed on the server side.
* Uses an **event-driven**, **non-blocking** [I/O model](#_5_I/O_Models) that makes it lightweight and efficient, perfect for **data-intensive real-time** apps that run across **distributed devices**.
* Features:
  + Asynchronous & Event Driven
  + Very Fast in code execution
  + **Single Threaded** (but supports concurrency via concept of event and callbacks) but Highly Scalable
  + No Buffering
  + Released under the MIT license
* Functions:
  + Output: console.log(“Hello”);
  + Call a JavaScript: node main.js
* Normally JavaScript is confined to a browser, but NodeJS allows you to run JavaScript on your machine by taking Google V8 engine. This enables you to do things you normally can’t do with JavaScript.
* What you can do now:
  + Access file on your computer with JavaScript
  + Listen to network traffic on your computer
  + Listen to HTTP request your machine gets and send back a file
  + Access your databases directly

## Node.js Application

* In Node Application, any asynchronous function accepts a [callback](#_Callback) as a last parameter and the callback function accepts error as a first parameter.
* Three steps/parts:
  + Import required modules: we use require directive to load a Node.js module
  + Create server: a server which listens to clients’ requests (similar to Apache HTTP Server)
  + Read request and return response

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Import required modules

var http = require("http");

http.createServer(function (request, response) {

// Send the HTTP header

// HTTP Status: 200 : OK

// Content Type: text/plain

response.writeHead(200, {'Content-Type': 'text/plain'});

// Send the response body as "Hello World"

response.end('Hello World\n');

}).listen(8081);

// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');

//This is an application which response all HTTP requests with Hello World!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## npm

* Node Package Manager
* Provides online repositories for node.js packages/modules which are searchable on search.nodejs.org
* Provides command line utility to install Node.js packages, do version management and dependency management of Node.js packages
* Install modules: npm install <Module name> (Then you can make use of this module in your js file as following: var someModule = require(‘<Module name>’);
* Global installation: Packages/dependencies are stored in system directory. Such dependencies can be used in CLI (Command Line Interface) function of any node.js but cannot be imported using require() in Node application directly.
* Local installation: Installed in the Node application folder that you are in. Packages are accessible via require() method.
* Commands:
  + Install: npm install <Module name>
  + Uninstall: npm uninstall <Module name>
  + Look at the content of directory: npm ls
  + Update: npm update <Module name>
  + Search: npm search <Module name>
  + Create a module:
    - npm init
    - npm adduser
    - npm publish

## REPL Terminal

* Read: reads users’ input, parse the input into JavaScript data-structure and stores in memory
* Eval: takes and evaluates the data structure
* Print: prints the result
* Loop: loops the above command until user press ctrl+c twice
* Can be started in cmd using: node
* Underscore: \_ means to get the last result (E.g. var x = 10 var y = 20 x + y var sum = \_)
* Commands:
  + Ctrl + c: terminate the current round
  + Ctrl + c twice: terminate the Node REPL
  + Ctrl + d: terminate the Node REPL
  + Up/Down Keys: see command history and modify previous commands
  + Tab key: list of current commands
  + .help: list of all commands
  + .break: exit from multiline expression
  + .clear: exit from multiline expression
  + .save filename: save current Node REPL session to a file
  + .load filename: load file content in current Node REPL session

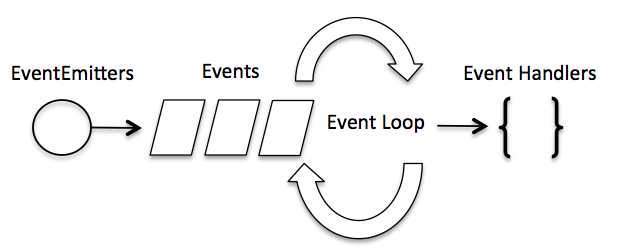
## package.json

* Defines the properties of a package, present in the root directory of the Node application/module.
* Attributes:
  + name
  + version
  + description
  + homepage
  + author
  + contributors
  + dependencies
  + repository
  + main
  + keywords

## Callback

* An asynchronous equivalent for a function.
* Called at the completion of a given task.
* A user makes a request to the server for data in database. The server keeps doing other things until the database calls it back (notifies it that the data is ready).
* A function that is usually passed as an argument to another function, usually invoked after some kind of event (E.g. setTimeout(callback, 5000))

## Event Driven Programming

* In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.
* To make use of events:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Import events module

var event = require(‘events’);

//Create an eventEmitter object

var eventEmitter = new events.EventEmitter();

//Bind event and event handler as follows

eventEmitter.on(‘eventName’, eventHandler);

//Fire an event

eventEmitter.emit(‘eventName’);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## EventEmitter

* Many objects in Node emit events. (E.g. a net.Server emits an event each time a peer connects to it, a fs.readStream emits an event when the file is opened)
* All objects which emit events are instances of events.EventEmitter.
* EventEmitter class lies in events module
* Examples of events emitted:
  + When an EventEmitter instance faces any error, it emits an ‘error’ event.
  + When new listener is added, it fires an ‘newListener’ event.
  + When a listener is removed, it fires an ‘removeListener’ event.
* Provides multiple properties:
  + on: used to bind a function with the event
  + emit: used to fire an event
* Methods:
  + addListener(event, listener)
  + on(event, listener)
  + once(event, listener)
  + removeListener(event, listener)
  + removeAllListeners(event)
  + setMaxListeners(n)
  + listeners(event)
  + emit(event, [arg1], [arg2], […])

## Difference between Callbacks and Events

* Callback functions are called when an asynchronous function returns its result.
* Event handling works on the [observer pattern](#_Observer_Pattern), functions which listens to events act as Observers.

## Buffers

* Can be constructed in several ways:
  + Create an uninitiated Buffer of 10 octets[[1]](#footnote-1):

var buf = new Buffer(10);

* + Create a Buffer from a given array:

var buf = new Buffer([10, 20, 30, 40, 50]);

* + Create a Buffer from a given string and optionally encoding type:

var buf = new Buffer(“Simply Easy Learning”, “utf-8”);

* Writing to Buffers

buf.write(string[, offset][, length][, encoding])

* + string: the string data to be written to buffer
  + offset: the index of the buffer to start writing at, defaults to 0
  + length: the number of bytes to write, defaults to buffer.length
  + encoding: defaults to utf-8
* Reading from Buffers

buf.toString([encoding][, start][, end])

* + encoding: defaults to utf-8
  + start: beginning index to start reading, defaults to 0
  + end: end index to end reading, defaults to complete buffer
  + returns a String
* Convert Buffer to JSON

buf.toJSON()

* + Returns a JSON-representation of the Buffer (in Decimal)
  + E.g. Simply Easy Learning = [ 83, 105, 109, 112, 108, 121, 32, 69, 97, 115, 121, 32, 76, 101, 97, 114, 110, 105, 110, 103 ]
* Concatenate Buffers

buf.concat(list[, totalLength])

* + list: array list of Buffer objects to be concatenated
  + totalLength: the total length of the buffers when concatenated
* Compare Buffers

buf.compare(anotherBuffer)

* Copy Buffer

buf.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

* + targetBuffer: Buffer object to contain the copied Buffer
  + targetStart: Defaults to 0
  + sourceStart: Defaults to 0
  + sourceEnd: Defaults to buffer.length
* Slice Buffer

buf.silce([start][, end])

* Buffer Length

buf.length

## Streams

* Is an instance of EventEmitter
* Commonly used events:
  + data: fired when there is data available to read
  + end: fired when there is no more data to read
  + error: fired when there is any error receiving or writing data
  + finish: fired when all data has been flushed to underlying system
* Enables you to read data from a source or write data to a destination in continuous fashion
* 4 Types:
  + Readable
  + Writable
  + Duplex: Both read and write operation
  + Transform: A type of duplex stream where the output is computed based on input
* Functions:
  + Read from stream
  + Write to stream
  + Piping[[2]](#footnote-2) streams

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

var fs = require("fs");

// Create a readable stream

var readerStream = fs.createReadStream('input.txt');

// Create a writable stream

var writerStream = fs.createWriteStream('output.txt');

// Pipe the read and write operations

// read input.txt and write data to output.txt

readerStream.pipe(writerStream);

console.log("Program Ended");

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + Chaining[[3]](#footnote-3) streams

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

var fs = require("fs");

var zlib = require('zlib');

// Compress the file input.txt to input.txt.gz

fs.createReadStream('input.txt')

.pipe(zlib.createGzip())

.pipe(fs.createWriteStream('input.txt.gz'));

console.log("File Compressed.");

// Decompress the file input.txt.gz to input.txt

fs.createReadStream('input.txt.gz')

.pipe(zlib.createGunzip())

.pipe(fs.createWriteStream('input.txt'));

console.log("File Decompressed.");

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## File System

* Node implements File I/O using simple wrappers around standard POSIX[[4]](#footnote-4) functions.
* Import Node File System module: var fs = require(“fs”);
* Every method in fs module have synchronous as well as asynchronous form.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

var fs = require("fs");

// Asynchronous read

fs.readFile('input.txt', function (err, data) {

if (err) {

return console.error(err);

}

console.log("Asynchronous read: " + data.toString());

});

// Synchronous read

var data = fs.readFileSync('input.txt');

console.log("Synchronous read: " + data.toString());

console.log("Program Ended");

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Open a file

fs.open(path, flags[, mode], callback);

* + path: a string having file name including path
  + flags: tell the behavior of the file to be opened

|  |  |
| --- | --- |
| **Flag** | **Description** |
| r | Open file for reading. An exception occurs if the file does not exist. |
| r+ | Open file for reading and writing. An exception occurs if the file does not exist. |
| rs | Open file for reading in synchronous mode. |
| rs+ | Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution. |
| w | Open file for writing. The file is created (if it does not exist) or truncated (if it exists). |
| wx | Like 'w' but fails if path exists. |
| w+ | Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists). |
| wx+ | Like 'w+' but fails if path exists. |
| a | Open file for appending. The file is created if it does not exist. |
| ax | Like 'a' but fails if path exists. |
| a+ | Open file for reading and appending. The file is created if it does not exist. |
| ax+ | Like 'a+' but fails if path exists. |

* + mode: sets the file mode (permission and sticky bits) but only if the file was created. Defaults to 0666, readable and writable.
  + callback: the callback function which gets two arguments (err, fd)
* Get File information: fs.stat(path, callback);

|  |  |
| --- | --- |
| **Method** | **Description** |
| stats.isFile() | Returns true if file type of a simple file. |
| stats.isDirectory() | Returns true if file type of a directory. |
| stats.isBlockDevice() | Returns true if file type of a block device. |
| stats.isCharacterDevice() | Returns true if file type of a character device. |
| stats.isSymbolicLink() | Returns true if file type of a symbolic link. |
| stats.isFIFO() | Returns true if file type of a FIFO. |
| stats.isSocket() | Returns true if file type of asocket. |

* Write to File: fs.writeFile(filename, data[, options], callback);
* Reading File: fs.read(fd, buffer, offset, length, position, callback);
* Closing File: fs.close(fd, callback);
* Truncate File: fs.ftruncate(fd, len, callback);
* Delete File: fs.unlink(path, callback);
* Create Directory: fs.mkdir(pat[, mode], callback);
* Read Directory: fs.readdir(path, callback);
* Remove Directory: fs.rmdir(path, callback);

## Global Objects

* Available in all modules, do not need to include these objects in our application, can use them directly.
* \_\_filename (two ’\_’): represents the filename of the code being executed – the resolved absolute path of this code file.
* \_\_dirname: represents the name of the directory that the currently executing script resides in
* setTimeout(cb, ms): a global function used to run callback cb after at least ms milliseconds[[5]](#footnote-5).
* clearTimeout(t):
  + used to stop a timer that was previously created with setTimeout(), t is the timer returned by setTimeout() function.
  + Can be used in a cancel button
* setInterval(cb, ms): used to run callback cb repeatedly after at least ms milliseconds. Returns an opaque value that represents the timer which can be used to clear the timer using clearInterval(t).
* clearInterval(t)
* console: a global object used to print different levels of messages to stdout and stderr.

|  |  |
| --- | --- |
| **Method** | **Description** |
| console.log([data][, ...]) | Prints to stdout with newline. This function can take multiple arguments in a printf()-like way. |
| console.info([data][, ...]) | Prints to stdout with newline. This function can take multiple arguments in a printf()-like way. |
| console.error([data][, ...]) | Prints to stderr with newline. This function can take multiple arguments in a printf()-like way. |
| console.warn([data][, ...]) | Prints to stderr with newline. This function can take multiple arguments in a printf()-like way |
| console.dir(obj[, options]) | Uses util.inspect on obj and prints resulting string to stdout. |
| console.time(label) | Mark a time. |
| console.timeEnd(label) | Finish timer, record output. |
| console.trace(message[, ...]) | Print to stderr 'Trace :', followed by the formatted message and stack trace to the current position. |
| console.assert(value[, message][, ...]) | Similar to assert.ok(), but the error message is formatted as util.format(message...). |

* process: a global object, an instance of EventEmitter and emits following events:

|  |  |
| --- | --- |
| **Event** | **Description** |
| exit | Emitted when the process is about to exit. There is no way to prevent the exiting of the event loop at this point, and once all exit listeners have finished running the process will exit. |
| beforeExit | This event is emitted when node empties its event loop and has nothing else to schedule. Normally, node exits when there is no work scheduled, but a listener for 'beforeExit' can make asynchronous calls, and cause node to continue. |
| uncaughtException | Emitted when an exception bubbles all the way back to the event loop. If a listener is added for this exception, the default action (which is to print a stack trace and exit) will not occur. |
| Signal Events | Emitted when the processes receives a signal such as SIGINT, SIGHUP, etc. |

## Utility Modules

|  |  |
| --- | --- |
| **Module** | **Description** |
| OS Module | Provides basic operating-system related utility functions. |
| Path Module | Provides utilities for handling and transforming file paths. |
| Net Module | Provides both servers and clients as streams. Acts as a network wrapper. |
| DNS Module | Provides functions to do actual DNS lookup as well as to use underlying operation system name resolution functionalities. |
| Domain Module | Provides way to handle multiple different I/O operations as a single group. |

## Web Module

* Node.js provides http module which can be used to create either HTTP client or server.

## Express Framework

* A minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications.
* Core features:
  + Allows to set up middlewares to respond to HTTP Requests
  + Defines a routing table which is used to perform different action based on HTTP Method and URL
  + Allows to dynamically render HTML Pages based on passing arguments to templates
  + Need to install:
    - npm install express –save
    - npm install body-parser[[6]](#footnote-6) --save
    - npm install cookie-parser[[7]](#footnote-7) --save
    - npm install multer[[8]](#footnote-8) --save
  + Basic Routing
  + Serving Static Files
  + GET method: process\_get
  + POST method: process\_get
  + File Upload
  + Cookies Management

# JSON

* Javascript object notation

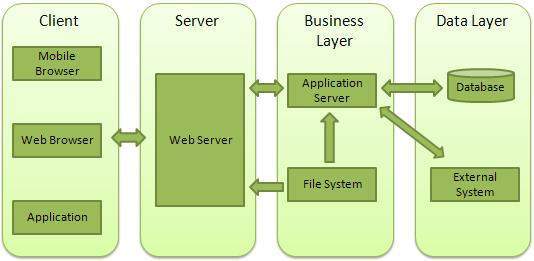
# JHipster

* yo jhipster
* yo jhipster:entity EntityName
* mvn
* mvn liquibase:clearCheckSums

# Web Server

* A software application which handles HTTP requests sent by the HTTP client (e.g. web browsers) and returns web pages in response to the clients.
* Usually delivers html documents along with images, style sheets and scripts.
* Most of them support server side scripts using scripting language or redirect to application server which perform the specific task of getting data from database, perform complex logic, etc. and then sends a result to the HTTP client through the Web server.
* Apache web server is one of the most commonly used web server, it’s an open source project.

# Web Application Architecture

* Usually divided into 4 layers
  + Client: consists of browsers and applications which can make HTTP request to the web server.
  + Server: consists of web server which can intercepts the request made by clients and pass them the response.
  + Business: consists of application server which is utilized by web server to do required processing, interacts with data layer via database or external programs.
  + Data: consists of databases or any source of data.

# Web Service

* A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
* Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.
* This interoperability[[9]](#footnote-9) is due to the use of open standards.

## RESTful Web Services

* Web services based on REST Architecture

# REST Architecture

* Representational State Transfer
* Web standards based architecture uses HTTP protocol.
* First introduced by Roy Fielding in 2000.
* A REST server simply provides access to resources.
* REST clients access and modify the resources using HTTP protocol.
* Each resource is identified by URIs/global IDs.
* Uses various representation to represent a resource (e.g. text, JSON[most popular], XML)

# Five I/O Models

* (User space I/O, including 2 steps(wait for the data to come & copy from kernel to user space))
* (For socket: 1. Wait for data to be transferred from the Internet and copy them to the buffer in kernel; 2. Copy the data from the buffer in kernel to the buffer in the process)
* I/O Multiplexing: Use select (can wait for more than one descriptor) to test if the data is ready and then use recvfrom, also waits for data and the process stuck. [Can handle more connections, but might not be faster than multi-threading + blocking I/O]
* Polling: Set the socket to be non-blocking and keeps calling to check again and again in an iteration.
* Blocking I/O: After you call recvfrom[[10]](#footnote-10), it waits until data comes and returns the value. If no data comes in, your process stuck at this point.
* Non-blocking[[11]](#footnote-11) I/O: Set the socket to be “O\_NONBLOCK”, keeps asking if the data is ready, if not, skip it.
* Signal driven I/O (SIGIO): Signal handler process continues executing while waiting for data, only starts blocking after we know the data is ready and start calling functions to read and copy it.
* Asynchronous I/O (the POSIX aio\_functions): Use aio\_read to pass the descriptor, buffer pointer, buffer size to the core, system immediately returns the result. Signal is given to us when our data is already copied to a buffer.
* Synchronous I/O: I/O operation blocks the process until I/O finished. Including blocking, non-blocking, signal driven, multiplexing.

# Framework

* A software providing generic functionality which can be selectively changed by additional user-written code, thus providing application-specific software.
* Frameworks can include:
  + Libraries: shareable, reusable bits of low-level code in each language.
  + APIs: facilitate access to the database back end.
  + Scaffolding: a technique some [MVC](#_MVC) frameworks employ that strengthens how a database can be accessed, make your site more powerful with better leverage of the database.
  + AJAX[[12]](#footnote-12): some JavaScript frameworks are embedded into larger frameworks, such as some incorporate AJAX into a site’s functionality
  + Caching: cuts back on server workload
  + Security: via authentication and authorization frameworks
  + Compilers or Just-in-Time compilers
* Types:
  + Software frameworks: A reusable environment that’s part of a larger software platform that are geared toward facilitating the development of software applications.
  + Web application frameworks: are software frameworks that are used to streamline web app and website development, web service and web resources. (E.g. [MVC](#_MVC) architecture)

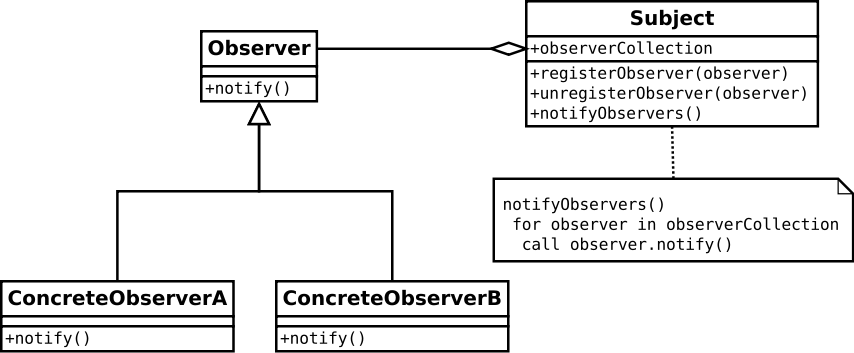
## Differences with normal libraries

* Inversion of control: In a framework, the overall program’s flow of control is not dictated by the caller, but the framework. While when using libraries, your program is in complete control of the flow.
* Default behavior: A framework has a default behavior which must be some useful behavior instead of a series of no-ops[[13]](#footnote-13).
* Extensibility: A framework can be extended by the user by selective overriding, or can be specialized by user code to provide specific functionality.
* Non-modifiable framework code: The framework code is not supposed to be modified while accepting user-implemented extensions.
* You write your code “within” a framework, external to libraries.

# MVC

* Model-View-Controller
* A software architectural pattern which separates data and its representation.

# Observer Pattern

* A software design pattern
* An object (subject) maintains a list of its dependents (observers) and notifies them automatically of any state changes usually by calling one of their methods.
* After notification each observer carries out its task via a separate thread s to prevent blocking.
* Mainly used in distributed event handling systems.
* A key part of [MVC](#_MVC) architectural pattern. (Still arguing)
* Implemented in many programming libraries and systems, including almost all GUI toolkit.
* Structure:

# Gulp

* A task runner that uses Node.js as a platform.
* Builds system automated tasks like CSS & HTML minification, concatenating library files, compiling SASS files.
* Tools similar: Grunt

# Build System

* Collection of tasks which automate the repetitive work.
* Usage
  + Compilation of preprocess CSS and JavaScript.
  + Minification of files to reduce its size.
  + Concatenation of files into one.
  + Triggering the server for automatic reloading.
  + Creation of deployment builds to store the resulting files in one location.
* Works with 3 components”
  + Package manager
  + Preprocessors
  + Task runners and build tools

Package Manager

* Used to automate the installation upgrade, removal of required dependencies, clean libraries and packages used in the development environment.
* Examples: bower, npm

Preprocessors

# GIT

# Markdown

* A way to style text on the web.

# AngularJS

* A client-side/frontend JavaScript framework
* Create responsive websites, plays well with jQuery, easy to test
* Use directives to link js with html:

<body ng-controller=”StoreController”></body>

function StoreController() {}

* BDD: behavior driven development
* TDD: test driven development
* Automated Testing

## Directive

* A marker on a HTML tag that tells Angular to run or reference some JavaScript code.

## Custom Directive

* Invoking methods:
  + Element name: <w3-test-directive></w3-test-directive>
  + Attribute: <div w3-test-directive></div>
  + Class: <div class="w3-test-directive"></div>
  + Comment: <!-- directive: w3-test-directive -->
* Options:
  + Restrictions *[string]* (Default: EA):
    - E: for Element name
    - A: for Attribute
    - C: for Class
    - M: for Comment
  + Priority *[number]*: Smaller numbers have higher priorities
  + Template *[string]*: An element using this directive will be replaced by the HTML template
  + TemplateUrl *[string]*: ‘myTemplate.html’
  + Replace *[bool]*:
    - True: replace the directive with template
    - False: insert template into directive
  + Transclude *[bool]*:
    - True: insert HTML elements in directive into template
    - False:
  + Scope *[bool || plain object]*:
    - True: inherit father scope
    - False: new scope
    - @attr: binds to a matching DOM attribute’s evaluated string value
    - =attr: binds to a matching DOM attribute’s scope property
    - &attr: binds to a matching DOM attribute’s scope function
    - @
    - =
    - &
  + Controller *[function controllerName($scope, $element, $attrs) {…})]*: to define a controller for this directive (mostly for having interactions with other directives)
  + Require *[string]*: to name the directives this directive needs to interact with
  + Link *[function($scope, $element, $attrs) {…}]*: can add watcher methods (e.g. $scope.$watch) or do initializations
  + Compile *[function(telement, tattrs, transclude) {return function(scope, element, attrs) {…}}]*: will be executed before link, used for dynamically modify template

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

angular.module(“myApp”, []).directive(“theName”, function() {

return {

restrict: ‘’,

template: ‘’

}

});

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Module

* Where we write pieces of our Angular application.
* Makes our code more maintainable, testable and readable.
* Where we define dependencies for our app.
* Modules can user other Modules
* To create a module (This means to run the module called)

var app = angular.module(‘store’, [])

(app name, dependencies)

Create server:

var http = require('http');

http.createServer(function(request, response) {

response.writeHead(200);

response.write("Hello, this is Lencia!");

response.end();

}).listen(8080);

Non-blocking:

var fs = require('fs');

fs.readFile('index.html', function(error, contents) {

console.log(contents);

});

Combining properties of the response:

http.createServer(function(request, response) {

response.writeHead(200, {

‘Content-Type’: ‘text/html’

});

response.end("Hello, this is Lencia!");

}).listen(8080);

## Programming Questions

1. Q: How is *this* and *$scope* different in AngularJS controllers?

A:

*this*:

* + when the controller constructor function is called, *this* is the controller
  + when a function defined on a $scope project is called, this is the “scope in effect when the function was called”. This may (or may not) be the $scope that the function is defined on. So, inside the function, this and $scope may not be the same.

# UI Choices

## Angular Material (for Angular1)

## PrimerNG (for Angular2)

# Shell Script

* Make use of shell and related instructions, no need for compiling
* Comment: #
* Print every command as it is executed (useful for debug): set -x
* Execute:
  + Direct:
    - Absolute path: e.g. /home/dmtsai/shell.sh
    - Relative path: e.g. if the work directory is currently in /home/dmtsai/ then we can call ./shell.sh
    - Variable path: put shell.sh into a directory that PATH points to e.g. ~/bin/
  + Execute as a bash program: bash shell.sh or sh shell.sh
* To write a script:
  + Use vim instead of vi: vim has extra grammar checking schema
  + Declare the shell to use for this script in the first line
    - #!/bin/bash: declare that this file uses the bash grammar so that it loads the corresponding environment settings for bash when it’s executed.
  + Use #comments to list information about this program
    - Function
    - Version
    - Author and contact
    - Copyright declaration
    - Special commands e.g. using absolute path
    - Requirement of environment variables
  + Declare environmental variables:

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin

export PATH

* + Main body
  + Define the return value: exit n
* Read a input and then output it:

read -p "Pleas input your name: " name

echo "your name is ${name}"

## Commands

### Both

* Change directory
  + cd /: change to root directory
  + cd ..: change to parent directory
* Make directory: mkdir [dirname]
* Unzip .gzip or .tar file
  + tar cvf zippedName folderToZip: zip a file to a tar file
  + gzip –d: extract .gzip file
  + tar xvf: extract .tar file
  + gzip –d < file.tar.gz | tar xvf -:extract a tar.gz file
* Check Disk Space: df
* Reallocate filesystem: mount

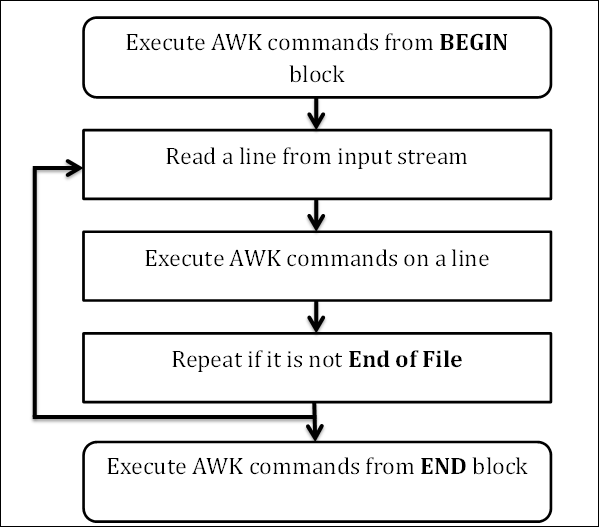
### Bash

* Check process
  + ps -A
* Change directory
  + cd -: back to previous directory
  + cd or cd ~: back to home directory
* chmod u+x program\_name
* Global Regular Expression Print
* Word Count
  + Default: print three numbers: lines, words, characters
  + wc –l PLVAR\_CSA\_20161003.txt: count newline
  + wc –w: count words
  + wc –m: count characters
* List:
  + ls –d \*/: list directories only, and only those under this directory
  + ls -a, --all:do not hide entries starting with .
  + ls –l: use a long listing format
  + ls –S: sort by size
* Conditional ecpressions:
  + General syntax: [ parameter FILE ] OR test parameter FILE OR [[ parameter FILE ]]
  + Where parameter can be any one of the following:
    - -e: Returns true value if file exists.
    - -f: Return true value if file exists and regular file.
    - -r: Return true value if file exists and is readable.
    - -w: Return true value if file exists and is writable.
    - -x: Return true value if file exists and is executable.
    - -d: Return true value if exists and is a directory.
* Delete directories:
  + rm –r dirName: present a prompt for approval to delete each of the files
  + rm –rf dirName: dele all without prompt
  + rmdir
* Kill process:
  + kill -9 -1
  + kill -9 <processId>
* Symbolic link:
  + Soft link
    - Def: If you delete the link, the file would still be there; if you delete the file, it becomes a broken link
    - Create: ln –s {/path/to/file-name} {link-name}
    - Unlink: ln -sf
    - Remove: rm {link-name}
  + Hard link
    - Def: If you delete a hard link, your file would still be there; if you delete the file, your data can still be accessible via the hard link file.
    - Create: ln {/path/to/file-name} {link-name}
    - Remove: rm {link-name}
* Less Pager
  + Search Navigation
    - Forward Search: /[pattern]
    - Backward Search: ?[pattern]
    - Next match (in the chosen direction): n
    - Previous match (in the chosen direction): N
  + Screen Navigation
    - Forward 1 window: ctrl + f
    - Backward 1 window: ctrl + b
    - Forward 0.5 window: ctrl + d
    - Backward 0.5 window: ctrl + u
  + Line Navigation
    - Forward 1 line: j
    - Backward 1 line: k
    - Forward 10 lines: 10j
    - Backward 10 lines: 10k
  + Other Navigations
    - Go to the end of file: G
    - Go to the start of file: g
    - Exit the less pager: q or ZZ
* Vim
* Connect to a server: ssh hydra@hkg3vs0002s.hk.hsbc
* Check file size in a directory: du –sh \*

### Command Prompt

* visit a website: curl http://localhost:8080
* check information about an ip address: nslookup [ipaddress]
* get ip address of current machine: ipconfig
* view files in directory: dir

# AWK

* An interpreted programming language
* Typical usage:
  + Text processing
  + Producing formatted text reports
  + Performing arithmetic operations
  + Performing string operations, and many more
* Workflow: Read, Execute, and Repeat

# Java

* Private constructor:
  + To ensure only one instance of the class is active. (To serve singleton classes)
  + To create a utility class[[14]](#footnote-14) which only contains static methods.
* Hashmap:
* .substring(starting index(inclusive), ending index(exclusive))
* use .equals() to compare Strings
* Size and length:
  + array.length [no parentheses!]
  + string.length()

## Java Servlet

* A Java program that extends the capabilities of a server.

# Apache Tomcat

* Open-source web server, developed by the Apache Software Foundation
* Implements several Java EE specifications including Java Servlet, JavaServer Pages(JSP), Java EL and WebSocket.
* Provides a “pure Java” HTTP web server environment in which Java code can run.

## Catalina

Tomcat’s servlet container

.war

* WEB-inf/web.xml

# Apache Maven

Life cycle:

# Database

* A physical container for collections.
* Each database gets its own set of files on the file system.

## SQL

* Select into
  + select \* into MUREXDB.PLVAR\_CSA\_IN\_DBF from MUREXDB.PLVAR\_CSA\_DBF where 1=2
* Insert into
  + insert into MUREXDB.PLVAR\_CSA\_DBF select \* from MUREXDB.PLVAR\_CSA\_IN\_DBF
* Group by, having
  + select \* from PLVAR\_CSA\_IN\_DBF where TRADE\_ID IN (select TRADE\_ID from PLVAR\_CSA\_IN\_DBF group by REPORT\_DATE, TRADE\_ID, CUR having count(1) > 1)

## CRUD

* Create, read, update and delete
* Basic functions of persistent storage

# ECMAScript

* The JavaScript programming language is standardized by ECMA[[15]](#footnote-15) under the name ECMAScript.
* Defines
  + Language syntax: parsing rules, keywords, statements, declarations, operators, etc.
  + Types: boolean, number, string, object, etc.
  + Prototypes and inheritance
  + The standard library of built-in objects and functions: JSON, Math, Array methods, Object introspection methods, etc.

# NoSQL database

* Different from relationship databases where you need to map everything and figure out the exact schema (tables, fields, types of fields)
* Plan the structure but no need to predefine
* Easy to scale
* Much faster in most operations

# MongoDB

* A cross-platform, document oriented database that provides high performance, high availability and easy scalability.
* A [NoSQL database](#_NoSQL_database)
* Written in C++
* Document database: save data into documents
* Use [json](#_JSON) like syntax to save data
* A single MongoDB server typically has multiple database[[16]](#footnote-16)s.
* Collection:
  + A group of MongoDB documents, the equivalent of an RDBMS table.
  + A collection exists within a single database.
  + Collections do not enforce a schema.
  + Documents within a collection can have different fields.
  + Typically, all documents in a collection are of similar or related purpose.
* Document:
  + A document is a set of key-value pairs.
  + Documents have dynamic schema, which means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection’s documents may hold different types of data.
* RDBMS terminology and corresponding MongoDB name

|  |  |
| --- | --- |
| **RDBMS** | **MongoDB** |
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key \_id provided by mongodb itself) |
| **Database Server and Client** | |
| Mysqld/Oracle | mongod |
| mysql/sqlplus | mongo |

* Advantages of MongoDB over RDBMS:
  + Schema less
  + Structure of a single object is clear.
  + No complex joins.
  + Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that’s nearly as powerful as SQL.
  + Tuning.
  + Easy to scale.
  + Conversion/mapping of application objects to database objects not needed.
  + Uses internal memory for storing the (windowed) working set, enabling faster access of data.
* Why MongoDB?
  + Document Oriented Storage: data is stored in the form of JSON style documents.
  + Index on any attribute
  + Replication and high availability
  + Auto-sharding
  + Rich queries
  + Fast in-place updates
  + Professional support by MongoDB
* No concept of relationship
* Usage:
  + Install
  + mongod –directoryperdb –dbpath [some db folder you created] –logpath [some log folder you created\mongo.log] --logappend --rest –install
  + net start MongoDB

# JPA

## Cascade Types

* CASCADETYPE.PERSIST
* CASCADETYPE.REMOVE
* CASCADETYPE.MERGE
* CASCADETYPE.REFRESH

# Highchart

* A pure JavaScript based charting library meant to enhance web applications by adding interactive charting capability.

## Supported Chart Types

|  |  |
| --- | --- |
| 1 | **Line Charts** Used to draw line/spline based charts. |
| 2 | **Area Charts** Used to draw area wise charts. |
| 3 | **Pie Charts** Used to draw pie charts. |
| 4 | **Scatter Charts** Used to draw scattered charts. |
| 5 | **Bubble Charts** Used to draw bubble based charts. |
| 6 | **Dynamic Charts** Used to draw dynamic charts where user can modify charts. |
| 7 | **Combinations** Used to draw combinations of variety of charts. |
| 8 | **3D Charts** Used to draw 3D charts. |
| 9 | **Angular Gauges** Used to draw speedometer type charts. |
| 10 | **Heat Maps** Used to draw heat maps. |
| 11 | **Tree Maps** Used to draw tree maps. |

# Course Notes

## Computational Thinking

* Decomposition
* Pattern Recognition
* Abstraction
* Algorithm Design

## Design of Computer Programs

# Technical Interview Questions

## Java Core

### HashTable v.s. HashMap?

|  |  |
| --- | --- |
| **HashTable** | **HashMap** |
| [Synchronized](#_Q:_Synchronized?) | Unsynchronized, better for non-threaded applications |
| **Doesn’t allow** *null* keys or values | Allows **1** *null key* and **any number** of *null values* |
|  | One of its subclasses is *LinkedHashMap*, can be used where you want predictable iteration order. |

### Synchronized?

* This keyword prevents concurrent access to a block of code or object by multiple threads.
* Only one thread can access the object at a time.
* Prevents memory consistency errors.
* *“Synchronized methods enable a simple strategy for preventing thread interference and memory consistency errors: if an object is visible to more than one thread, all reads or writes to that object's variables are done through synchronized methods”*

### StringBuilder v.s. StringBuffer?

* *StringBuilder* is faster than *StringBuffer* because it’s not [synchronized](#_Q:_Synchronized?).

### CheckedException v.s. RuntimeException (unchecked exception)?

|  |  |
| --- | --- |
| **CheckedException** | **RuntimeException** |
| Checked at compile time, related to resources | Checked during runtime, purely programmatic errors, wrong calculation, null data or even failures in business logic. |
| All classes that inherit from class Exception but not RuntimeException are checked exceptions. | All direct/indirect subclasses of RuntimeException and classes that inherit from class Error are unchecked exception. |
| Java requires checked exceptions to be explicitly caught or declared. | Not required but you can. |

### ArrayList v.s. LinkedList?

* *ArrayList* with *ArrayDeque[[17]](#footnote-17)* are preferable in much more use-cases than *LinkedList*.

|  |  |  |
| --- | --- | --- |
| **Comparison** | **ArrayList<E>** | **LinkedList<E>** |
| get(int index) | O(1) [Main Benefit] | O(n/4) |
| add(E element) | O(1) amortized, O(n) worst-case | O(1) |
| add(int index, E element) | O(n/2) average | O(n/4) average,  But O(1) when index=0 [Main Benefit] |
| remove(int index) | O(n/2) average | O(n/4) average |
| Iterator.remove() | O(n/2) average | O(1) [Main Benefit] |
| ListIterator.add(E element) | O(n/2) average | O(1) [Main Benefit] |
|  | Allow fast random read access, can grab any element in constant time, but adding/removing from anywhere but the end requires shifting all the latter elements over. Also, if you add more elements than the capacity of the underlying array, a new array(1.5 times the size) is allocated and the old one is copied to the new one, so adding is O(n) in the worst case but constant on average. | Allows constant-time insertions/removals using iterators, but only sequential access of elements. When you add/remove from the head of the list, it’s O(1) while *ArrayList* is O(n) in the same situation. |
| Memory Overhead | Maintains indices and element data | Maintains element data and 2 pointers for neighbor nodes. |

### Abstract Class v.s. Interface?

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| For **Modelling** a class hierarchy of similar looking classes (E.g. Animal can be an abstract class and Human, Lion, Tiger can be concrete derived classes) | For **Communication** between 2 classes which does not care about type of the class implementing the interface. (E.g. Height can be an interface property and it can be implemented by Human, Building, Tree) |
| May contain state (data members) and implementation (concrete method) | Can’t have states or implementation |
| Can be inherited without implementing the abstract methods, an “is-a” relationship, an inheritance | Have to be derived with every method implemented, not a relationship of inheritance |
| May **not** be multiple-inherited. | May be multiple-inherited, which means an interface can extends more than one other interfaces, or you can implement more than one interface in your classes. |
|  | If you must define a variable, it has to be *static final*, and it will be *public* by default (Everything in an interface is public) |
| Neither can have any instances. | |

### What is Java generic type? Example?

* A generic class or interface that is parameterized over types.
* Defined with: class name<T1, T2, …, Tn> {…}
* Example:

|  |  |
| --- | --- |
| non-generic Box class | generic Box class |
| public class Box {  private Object object;  public void set(Object object) { this.object = object; }  public Object get() { return object; }  } | public class Box<T> {  // T stands for “Type”  private T t;  public void set(T t) { this.t = t; }  public T get() { return t; }  } |

* A type variable can be any **non-primitive** type you specify.
* Type Parameter Naming Conventions:
  + E - Element (used extensively by the Java Collections Framework)
  + K - Key
  + N - Number
  + T - Type
  + V - Value
  + S, U, V etc. - 2nd, 3rd, 4th types
* Invocation: Box<Integer> integerBox;
* Instantiation:
  + In the past: Box<Integer> integerBox = new Box<Integer> ();
  + In Java SE 7 and later you can do: Box<Integer> integerBox = new Box<> ();

### How to create a new thread?

* An application that creates an instance of Thread must provide the code that will run in that thread, 2 ways to do this:

|  |  |
| --- | --- |
| Provide a *Runnable* object. (More general) | Subclass *Thread*. |
| The *Runnable* interface defines a single method, run, meant to conatin the code executed in the thread. The *Runnable* object is passed to the *Thread* constructor. | The *Thread* class itself implements *Runnable*, though its run method does nothing. An application can subclass *Thread*, providing its own implementation of run. |
| public class HelloRunnable implements Runnable {  public void run() {  System.out.println(“Hello from a thread!”);  }  public static void main(String args[]) {  (new Thread(new HelloRunnable())).start();  }  } | public class HelloThread extends Thread {  public void run() {  System.out.println(“Hello from a thread!”);  }  public static void main(String args[]) {  (new HelloThread()).start();  }  } |
| Both invoke Thread.start in order to start the new thread. | |

### Explain any key interface under concurrent package?

### Concurrency?

* <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>
* Computer users take it for granted that their systems can **do more than one thing at a time**. They assume that they can continue to work in a word processor, while other applications download files, manage the print queue, and stream audio. Even a single application is often expected to do more than one thing at a time. For example, that streaming audio application must simultaneously read the digital audio off the network, decompress it, manage playback, and update its display. Even the word processor should always be ready to respond to keyboard and mouse events, no matter how busy it is reformatting text or updating the display. Software that can do such things is known as concurrent software.

### Concurrent Programming

* 2 basic units of execution:
  + Processes:
  + Threads:

### ReentrantLock v.s. Synchronization?

### What is try-with-resources statement? Example?

## XML

# Useful Links:

## Books

1. <http://techbus.safaribooksonline.com/>
2. <http://www.untag-smd.ac.id/files/Perpustakaan_Digital_1/>

## Learning

1. <http://codingbat.com/>
2. <http://www.radford.edu/~mhtay/CPSC120/VIM_Editor_Commands.htm>
3. <http://www.thegeekstuff.com/2010/02/unix-less-command-10-tips-for-effective-navigation/>
4. http://www.typingstudy.com/

1. A byte, eight bits. [↑](#footnote-ref-1)
2. A mechanism where we provide output of one stream as the input to another stream. Normally used to get data from one stream and to pass output of that stream to another stream. There’s no limit on piping operations. [↑](#footnote-ref-2)
3. A mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. Normally used with piping operations. We can use piping and chaining to compress a file and decompress it. [↑](#footnote-ref-3)
4. Portable Operating System Interface is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. Defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems. [↑](#footnote-ref-4)
5. Actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days. [↑](#footnote-ref-5)
6. A node.js middleware for handling JSON, Raw, Text and URL encoded form data. [↑](#footnote-ref-6)
7. Parse Cookie header and populate req.cookies with an object keyed by the cookie names. [↑](#footnote-ref-7)
8. A middleware for handling multipart/form-data. [↑](#footnote-ref-8)
9. E.g. communication between Java and Python, or Windows and Linux applications [↑](#footnote-ref-9)
10. A function to receive message from sockets. [↑](#footnote-ref-10)
11. An algorithm is called non-blocking if failure or suspension of any thread cannot cause failure or suspension of another thread. [↑](#footnote-ref-11)
12. Asychronous JavaScript and XML [↑](#footnote-ref-12)
13. No Operation: an assembly language instruction, programming language statement, or computer protocol command that does nothing. [↑](#footnote-ref-13)
14. Java doesn't support what in C# (for example) is known as a "static class" - in other words, a utility class. A utility class is a helper class that's supposed to contain only static members. (Math and System are such cases in Java.) It doesn't make sense for them to be instantiated in any way. In C#, making a class static makes it implicitly both final/sealed and abstract. In Java, there is no such keyword and you can't make a class final and abstract. So if you had such a utility class, you'd make it final and give it a private constructor that's never called. [↑](#footnote-ref-14)
15. A standards body like W3C. [↑](#footnote-ref-15)
16. Database is physical container for collections. Each database gets its own set of files on the file system. [↑](#footnote-ref-16)
17. May be a good alternative to *LinkedList* for adding/removing from the head, but it is not a *List*. [↑](#footnote-ref-17)