Welcome to Java

October 26th, 2020

Week 1 Overview

1. What is Java?
2. Development Environment Setup
3. First Program
4. Data Types & Variables
5. Arrays
6. Control Flow (if-else & switch statements)
7. Methods
8. The 4 Pillars of Object Oriented Programing
9. Memory
10. Strings & Loops
11. Exception Handling
12. The Collections Framework

History of Java

- 1991
- James Gosling, Sun Microsystems
- C-Based
- Originally called “Oak”
- Acquired by **Oracle** in **2010**

ORACLE

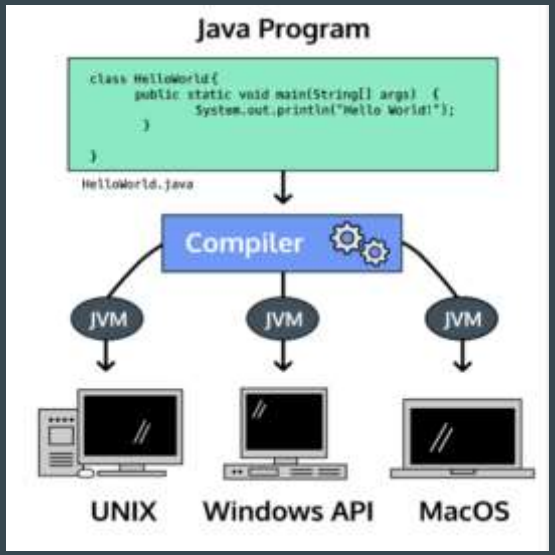
What's So Great About Java 8?

Java is high-level, object-oriented computer programming language that is multi-threaded, secure, strongly-typed and specifically designed to have as few implementation dependencies as possible.

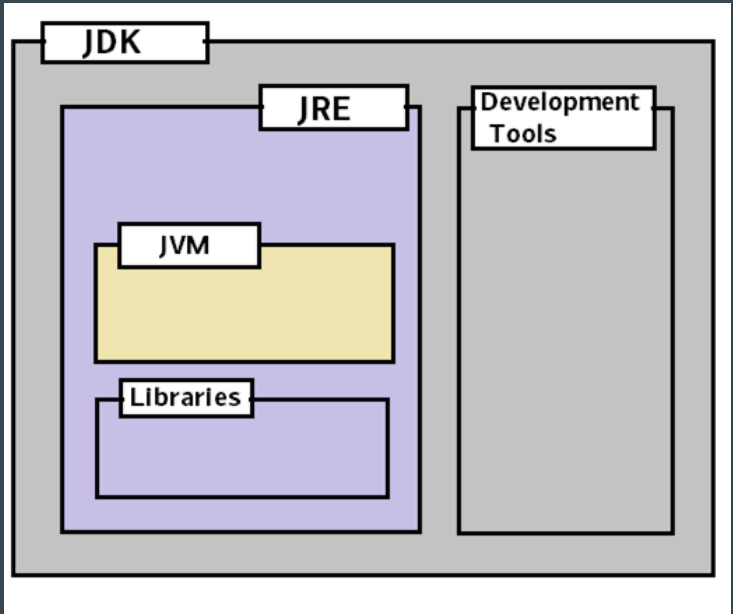
It is intended to let application developers **"write once, run anywhere" (WORA)**, meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

...JAVA IS → PLATFORM INDEPENDENT!

- Lambda Expressions
- Functional Interfaces
- Stream API
- Date & Time API
- ...many new features in Java 8...

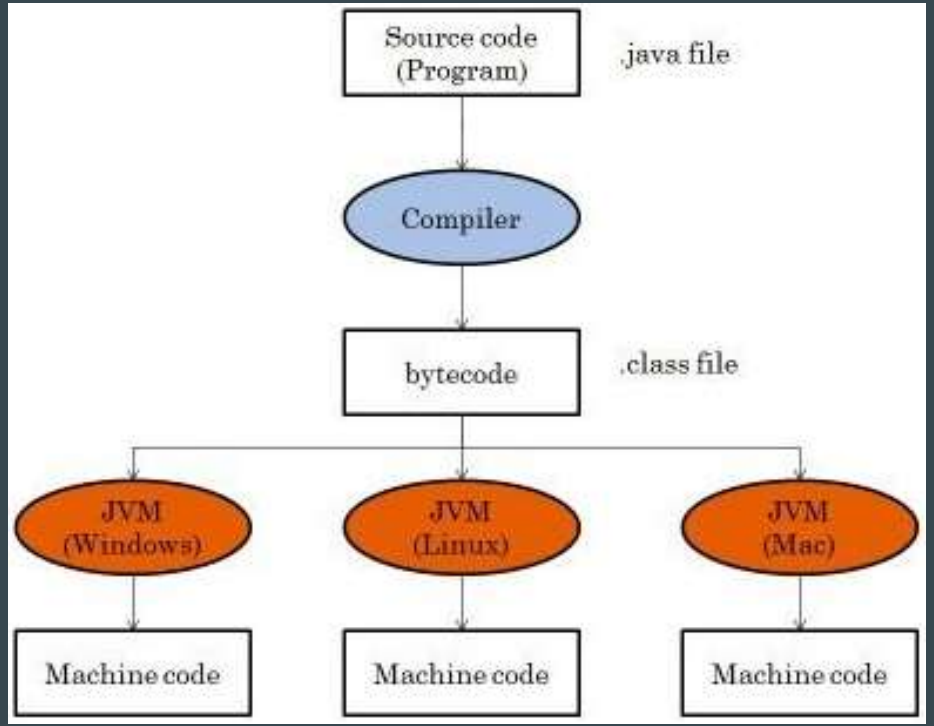


JDK, JVM, JRE - How Java Works Under the Hood



JDK = JRE + Development Tool
JRE = JVM + Library Classes

JDK



Primitive Data Types

Bytes are used to measure data.

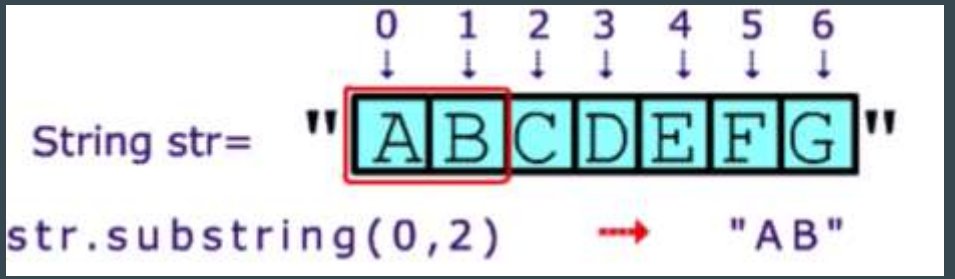
8 bits = 1 byte

Primitive type	Size	Description
boolean	not specified (JVM-dependent)	represents true and false values
byte	8-bit	numerical, integral value
short	16-bit	signed numerical, integral value
char	16-bit	unsigned numerical, Unicode character
int	32-bit	numerical, integral value
long	64-bit	numerical, integral value
float	32-bit	floating point value
double	64-bit	floating point value

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

String Class

- a String = an array of chars
- a String is an **object**
- Index of the last character is = to the string.length() **- 1**

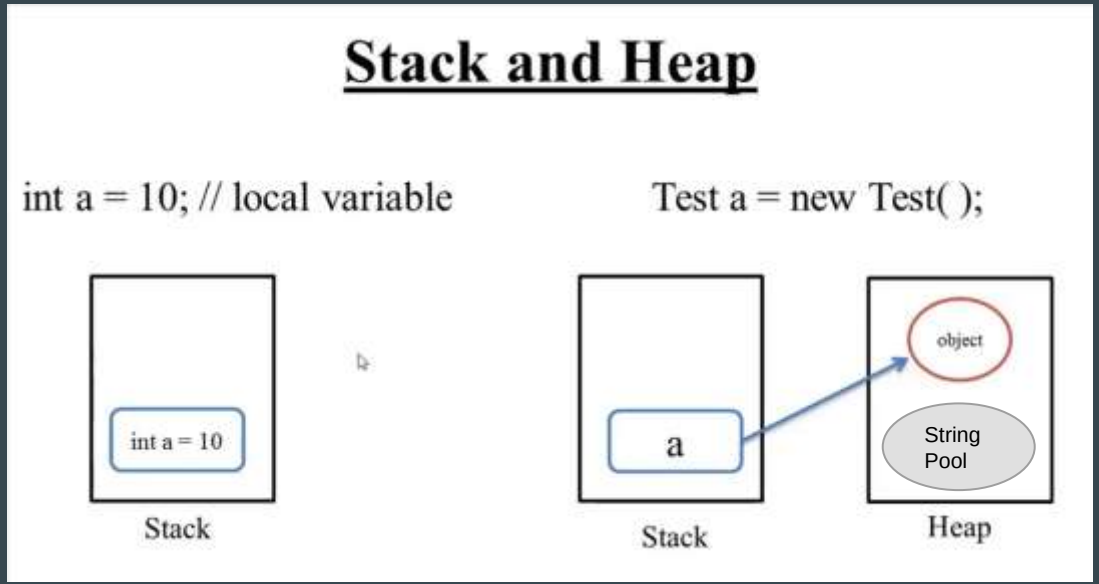


Memory: Stack and Heap

What happens when you click run?

- 2 memory locations are reserved for your app:
  1. Stack (methods + ref. variables)
  2. Heap (Objects + String pool)
- Garbage Collection:

- A process that runs in the heap
- Non-deterministic gc.run()
- Cleans up space
- Deletes unreferenced objects



Access Modifiers

Access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

4 Pillars of OOP

- OOP is a way of organizing our code (by splitting into multiple files)

Abstraction

Polymorphism

Inheritance

Encapsulation

Abstraction: the process of handling complexity by hiding unnecessary details from the user

*(i.e : **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class)*

Polymorphism : the ability to process objects differently depending on their data type or class, specifically by refining methods for derived classes

*(i.e : **Overloading** & **Overriding** a method)*

Inheritance: the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

*(i.e : **a child class** inherits the properties and methods of a **parent** or **super class**)*

Encapsulation: the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components.

*(i.e : we use **access modifiers** to control who can access data within a class)*

Interface

- An interface is a **contract**
- Java doesn't support Multiple-Inheritance
- *Can* implement **multiple interfaces**
- **Abstract Methods** (no body)
- cannot be instantiated

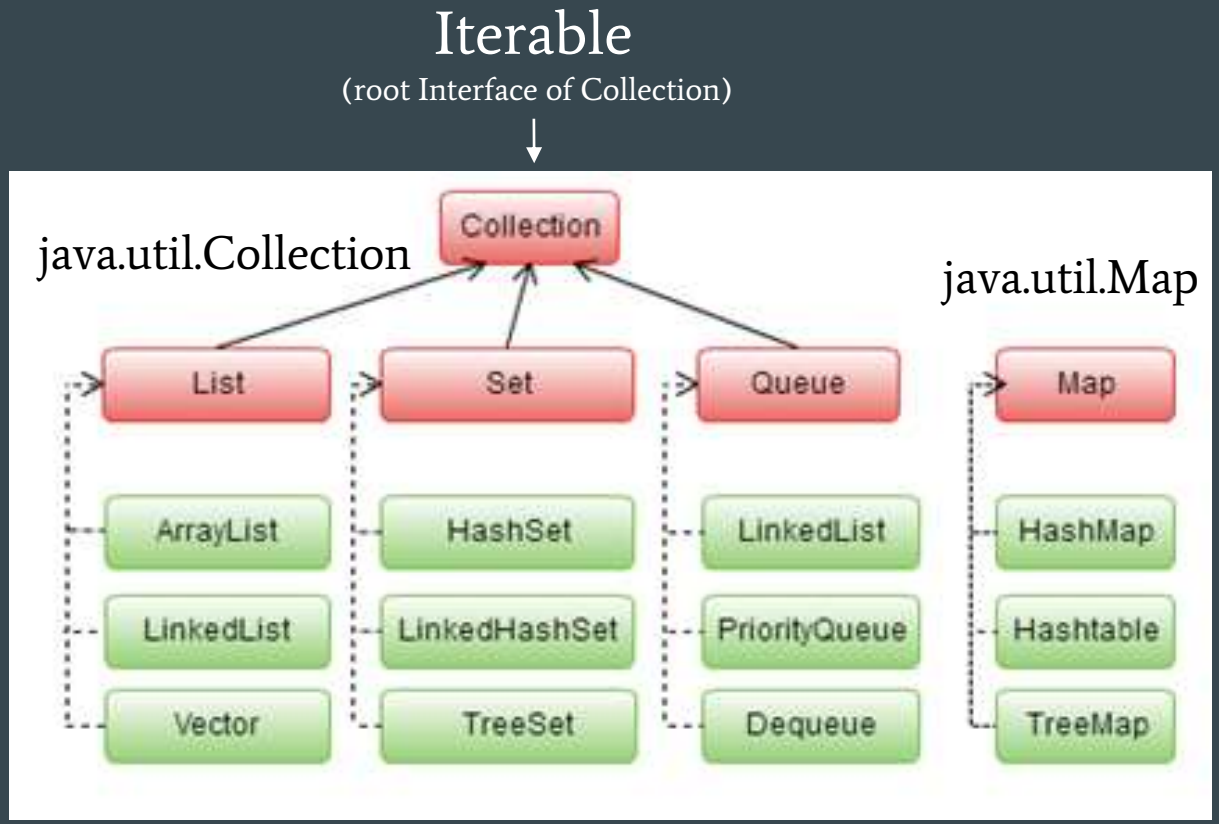
Abstract Class

- Almost a regular class...
- But *cannot* be instantiated
- **Abstract methods**
- Can only be a **parent** (extended)

Collections Framework (API)

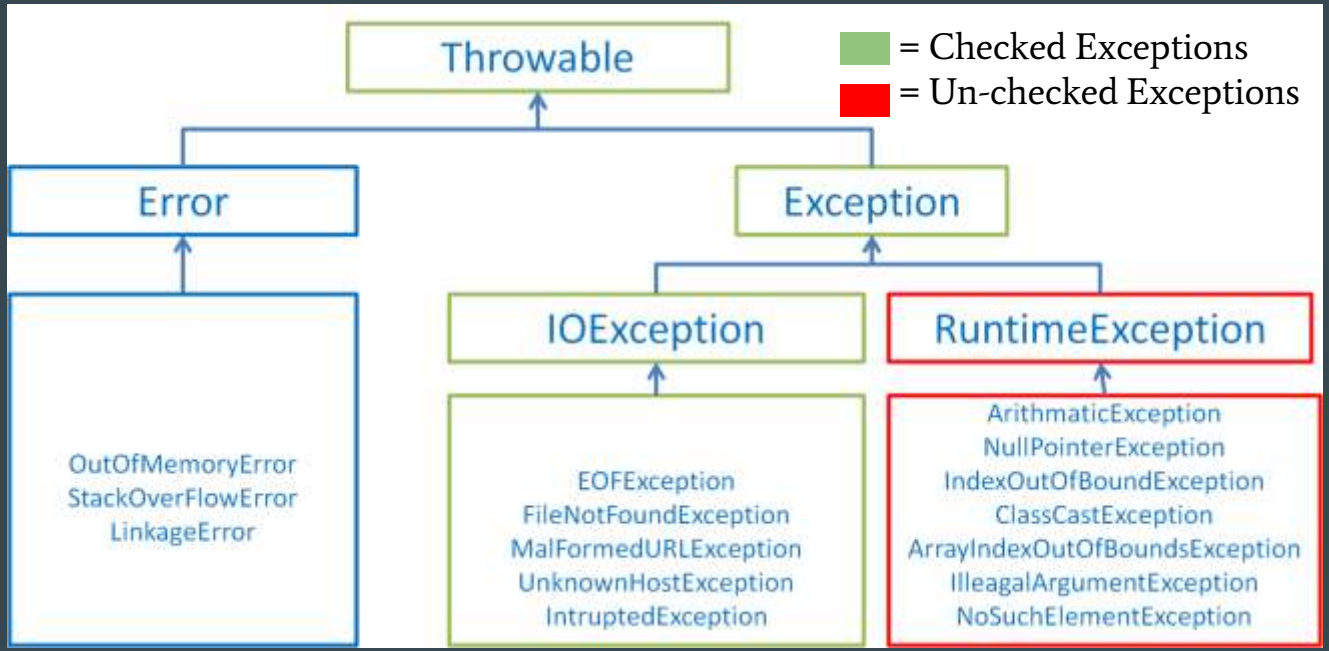
= Interface

= Class



Exceptions

- *An event which occurs during the execution of an application that disrupts the normal flow of the program's instructions*



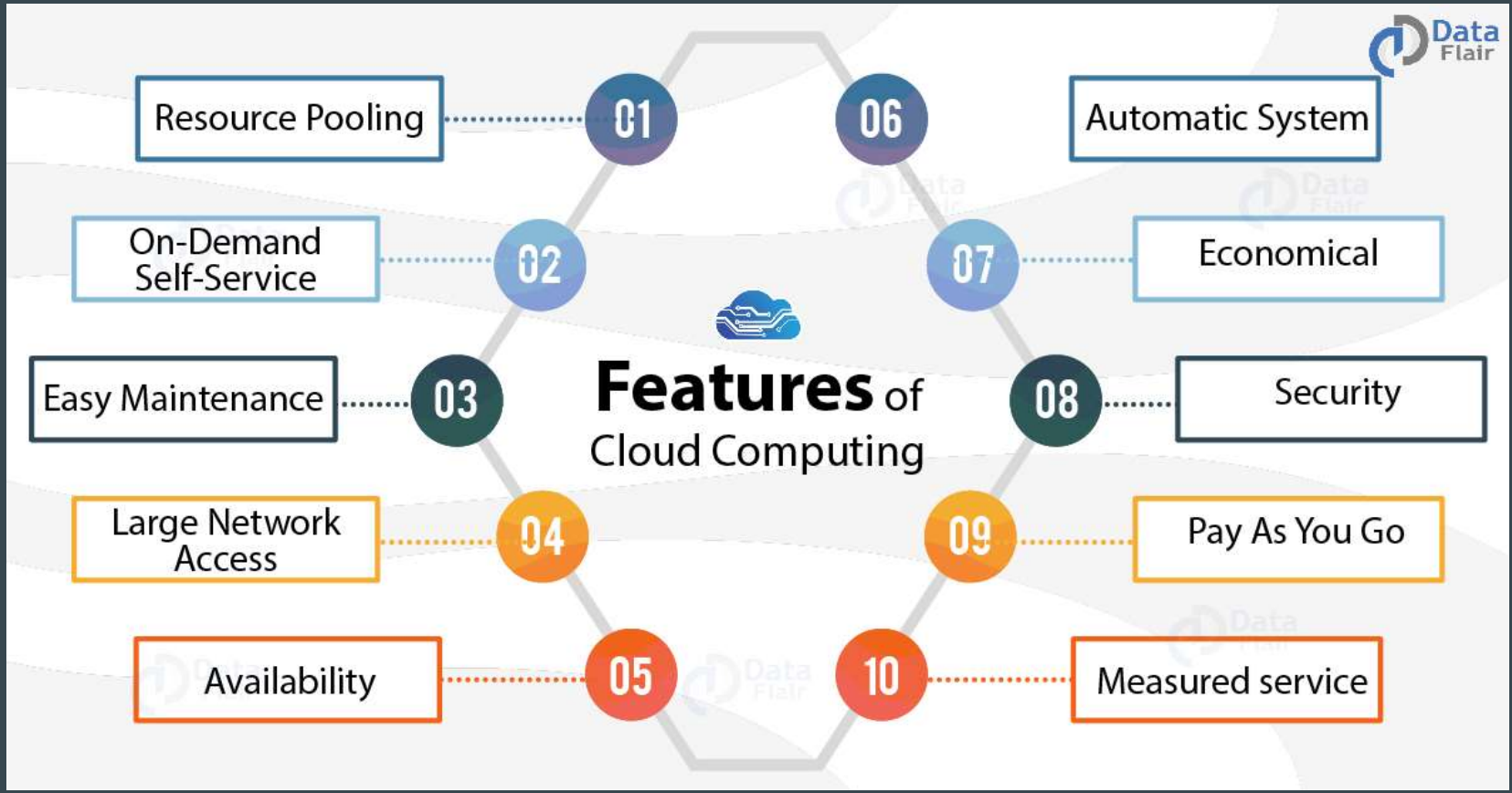
AWS & Cloud Computing

Week 2 - Part 1

What is Cloud Computing?

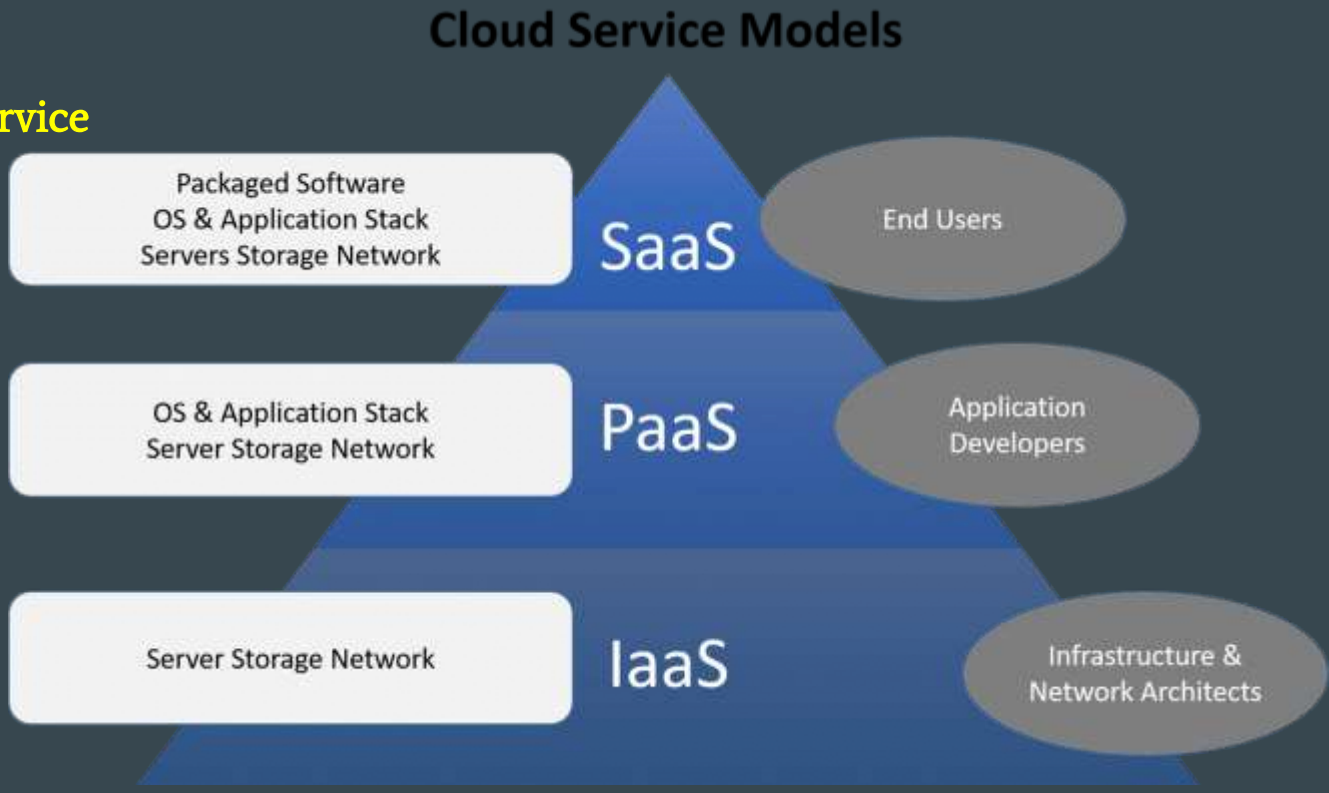
“Cloud computing is the on-demand delivery of compute power, database storage, applications and other IT resources through a cloud services platform via the Internet with pay-as-you-go pricing.”

Benefits of Cloud Computing

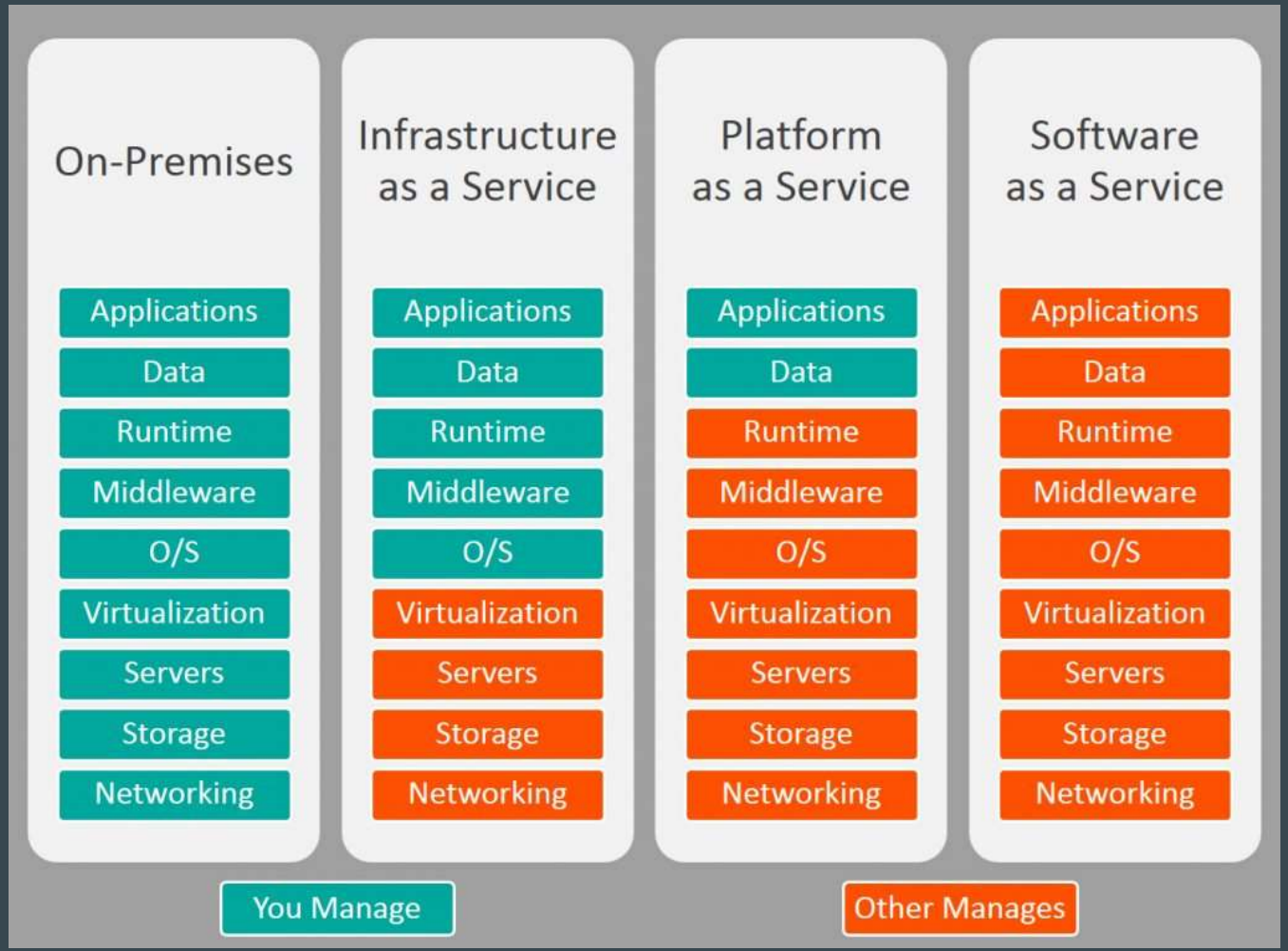


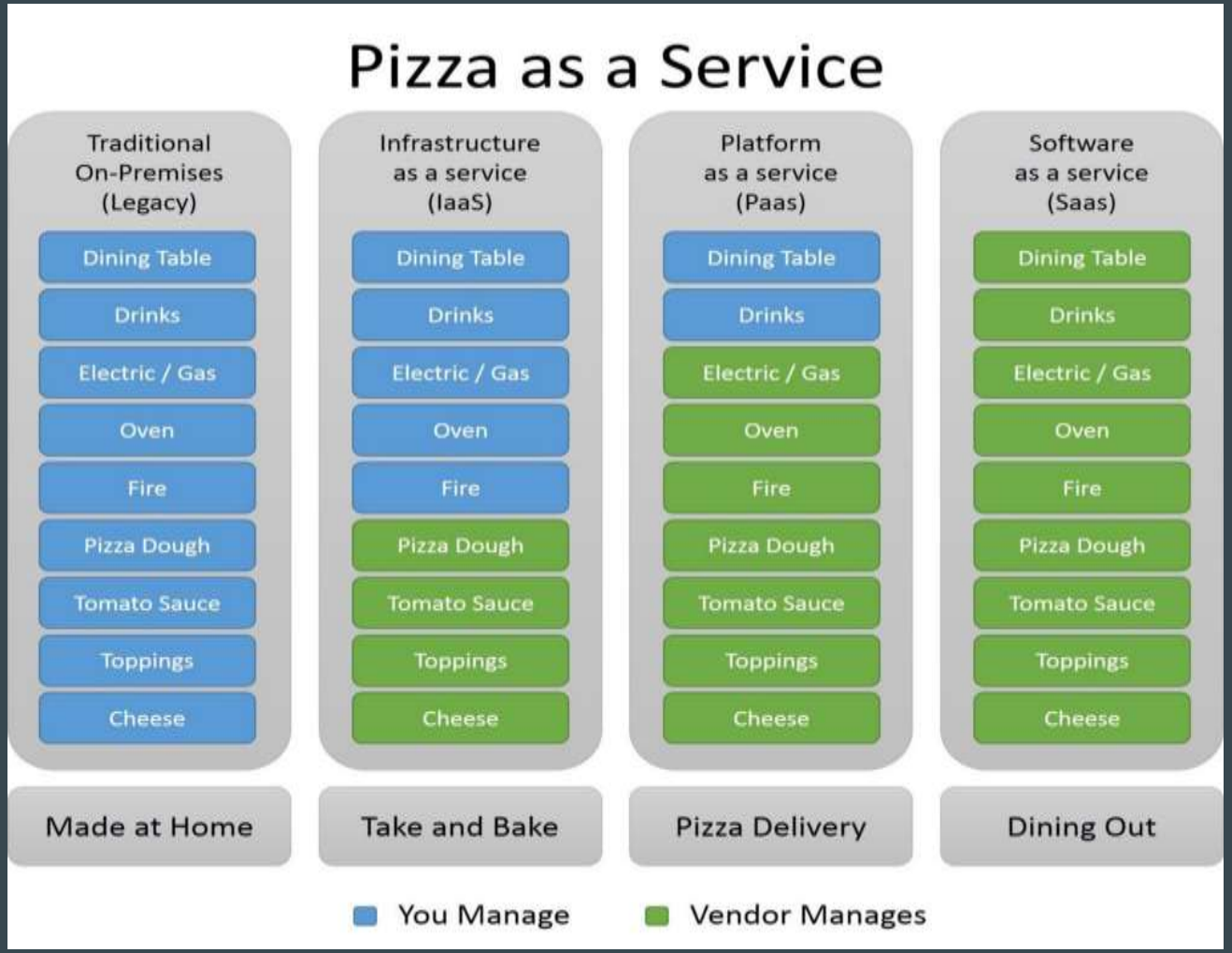
3 Models of Cloud Computing:

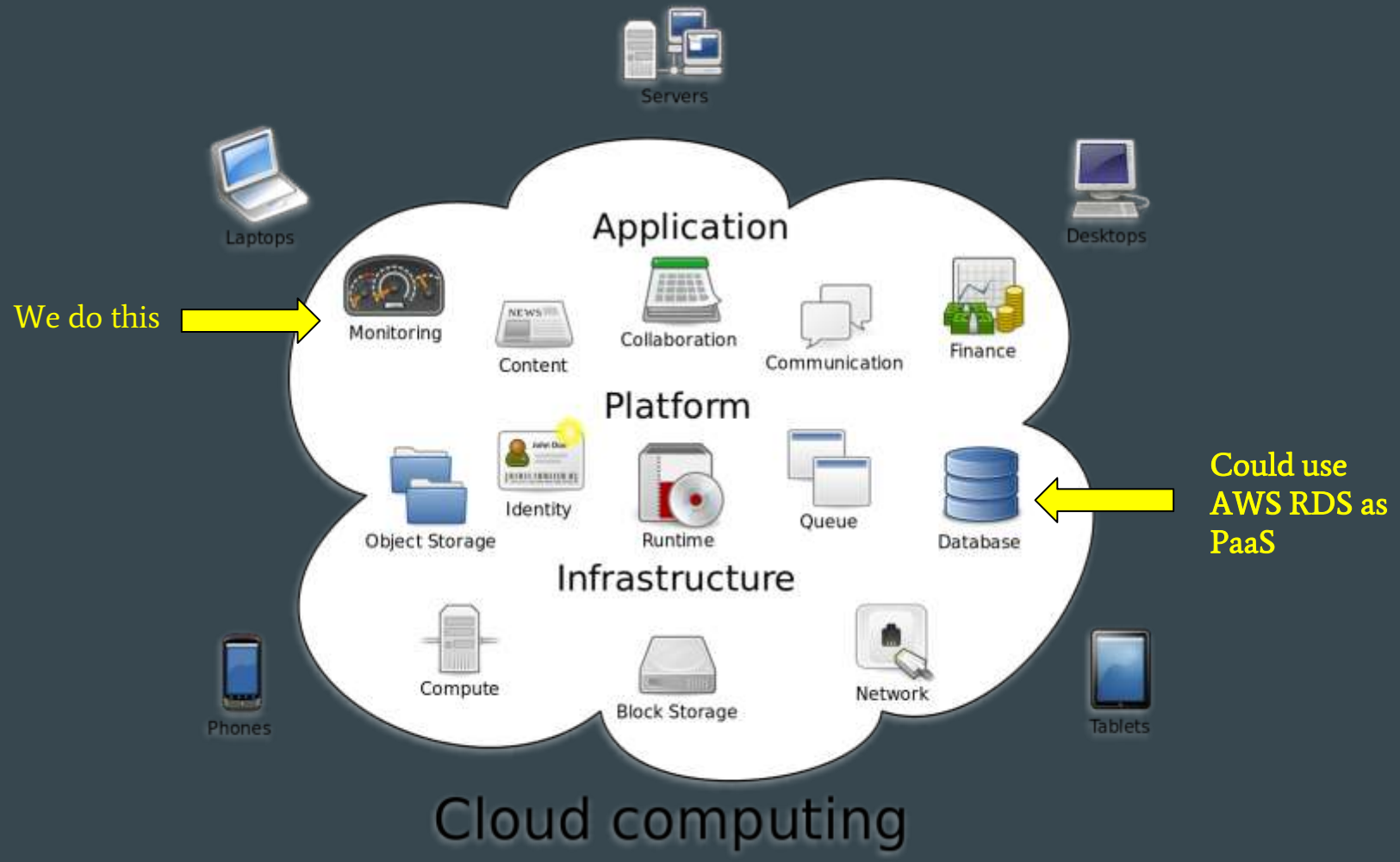
- Software As A Service
- Platform As A Service
- Infrastructure As A Service



Think AWS RDS and AWS Elastic Beanstalk!


On-Premises
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Infrastructure as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Platform as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Software as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You Manage
Other Manages


Pizza as a Service
Traditional On-Premises (Legacy)
Infrastructure as a service (IaaS)
Platform as a service (Paas)
Software as a service (Saas)
Dining Table
Drinks
Electric / Gas
Oven
Fire
Pizza Dough
Tomato Sauce
Toppings
Cheese
Made at Home
Take and Bake
Pizza Delivery
Dining Out
You Manage
Vendor Manages


Servers
Laptops
Desktops
We do this
Application
Monitoring
Content
Collaboration
Communication
Finance
Platform
Identity
Object Storage
Runtime
Queue
Database
Could use AWS RDS as PaaS
Infrastructure
Compute
Block Storage
Network
Phones
Tablets
Cloud computing

Amazon RDS

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud.

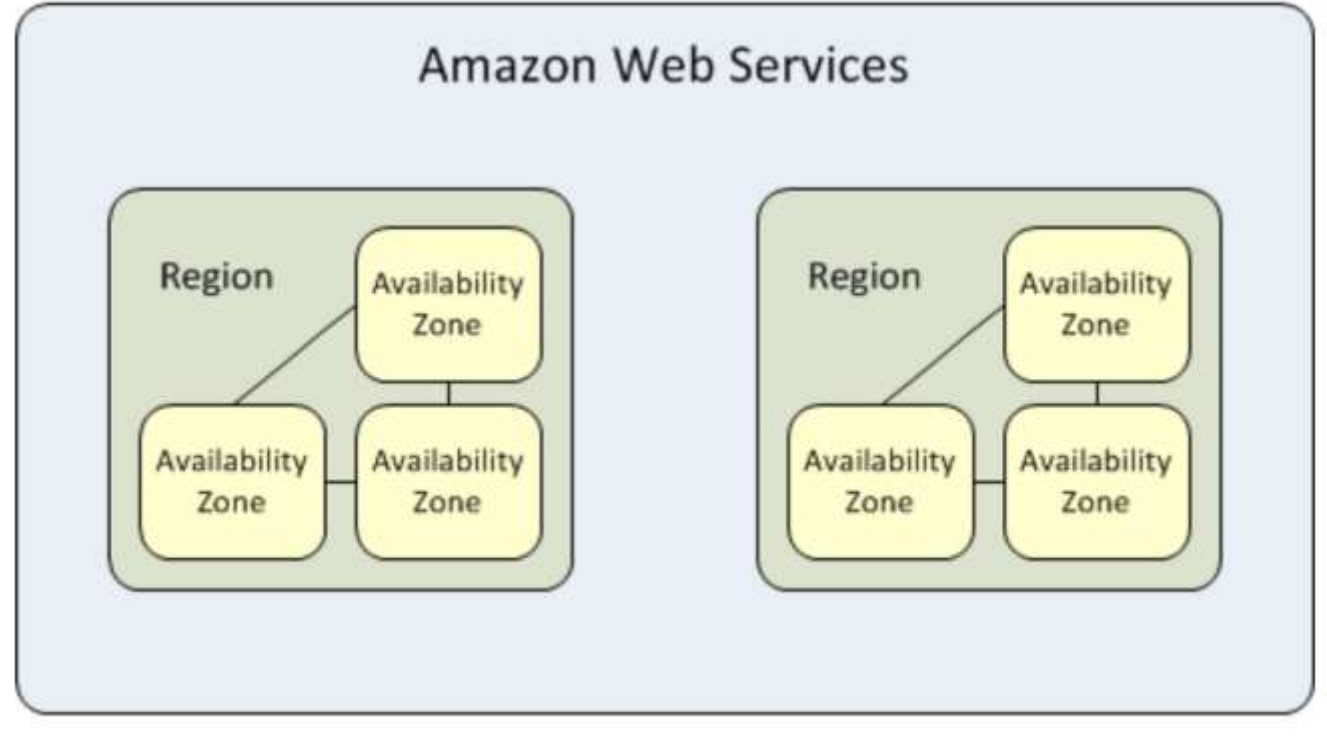
It's a PaaS!

Regions = is a highly available data center that houses Amazon cloud computing resources in different areas of the world (24 in total)

Availability Zones = Each AWS Region contains multiple distinct locations called Availability Zones, or AZs.

Each Availability Zone is engineered to be isolated from failures in other Availability Zones.

Fault-tolerance defines the ability for a system to remain in operation even if some of the components used to build the system fail.



SQL, RDBMS & JDBC

...

Database Overview

- A database is a collection of related data.

- **A DBMS (Database Management System) is a collection of *programs* used to create and maintain a database.**

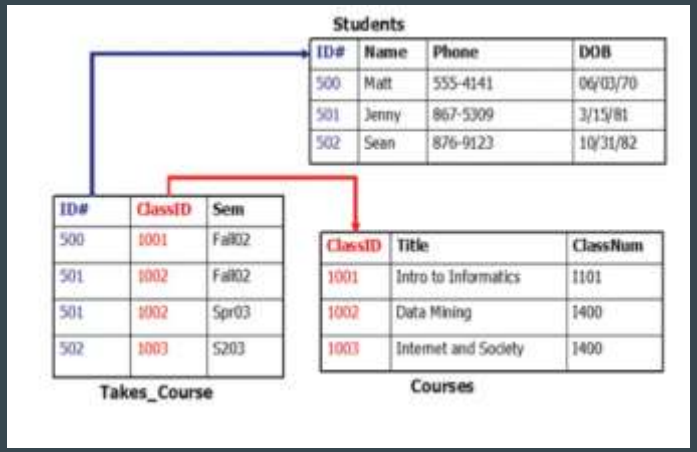
DBMS is useful for:

- Storing data in a structured format
- Allow **concurrent** use of data (many users)
- Control access to the data (admin privileges)
- Maintaining data integrity (maintain consistency and accuracy of data)
- Data backup and recovery

Types of DBMS (x4)

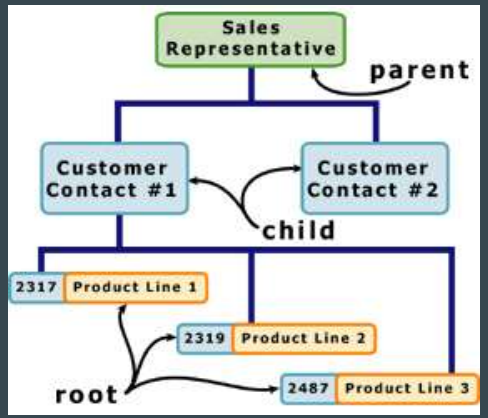
Relational Database Management Systems (RDBMS)

A relational database refers to a database that stores data in a structured format consisting of tables, **rows** and **columns**.



Hierarchical Database Management Systems

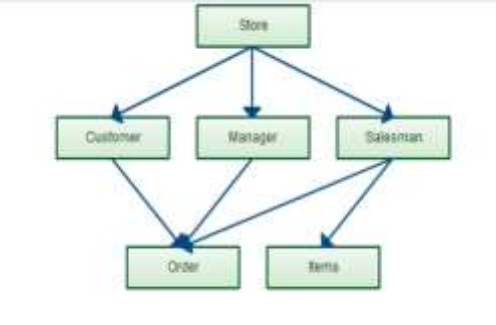
A hierarchical database is a design that uses a one-to-many relationship for data elements. Hierarchical database models use a tree structure that links a number of disparate elements to one "**owner**," or "**parent**," primary record.



Types of DBMS (x4)

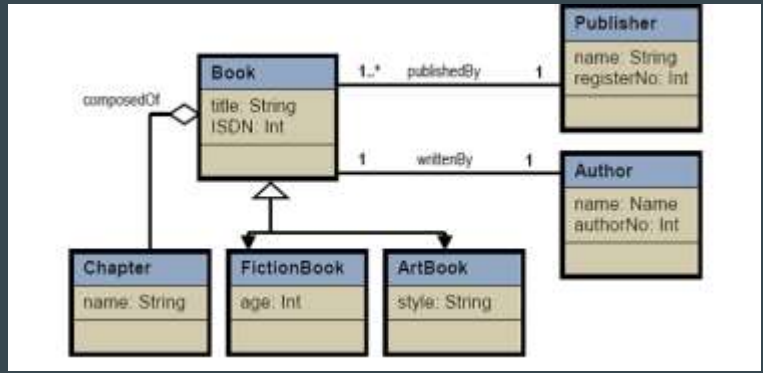
Network Database Management Systems

Network database management systems (Network DBMSs) use a network structure to create relationship between entities.



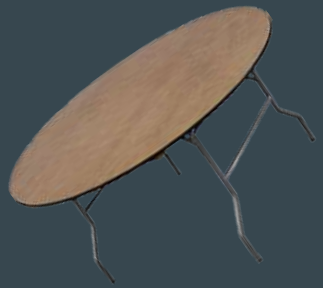
Object-oriented Database Management Systems

Object oriented DBMS support creation and modelling of data as objects. An object is a person, place, or thing.



What is a Relational Database?

- A Relational Database is a collection of data organized into *tables*.
- *Tables* contain:
  - **Columns** of data categories
  - **Rows** with particular instances of that data category



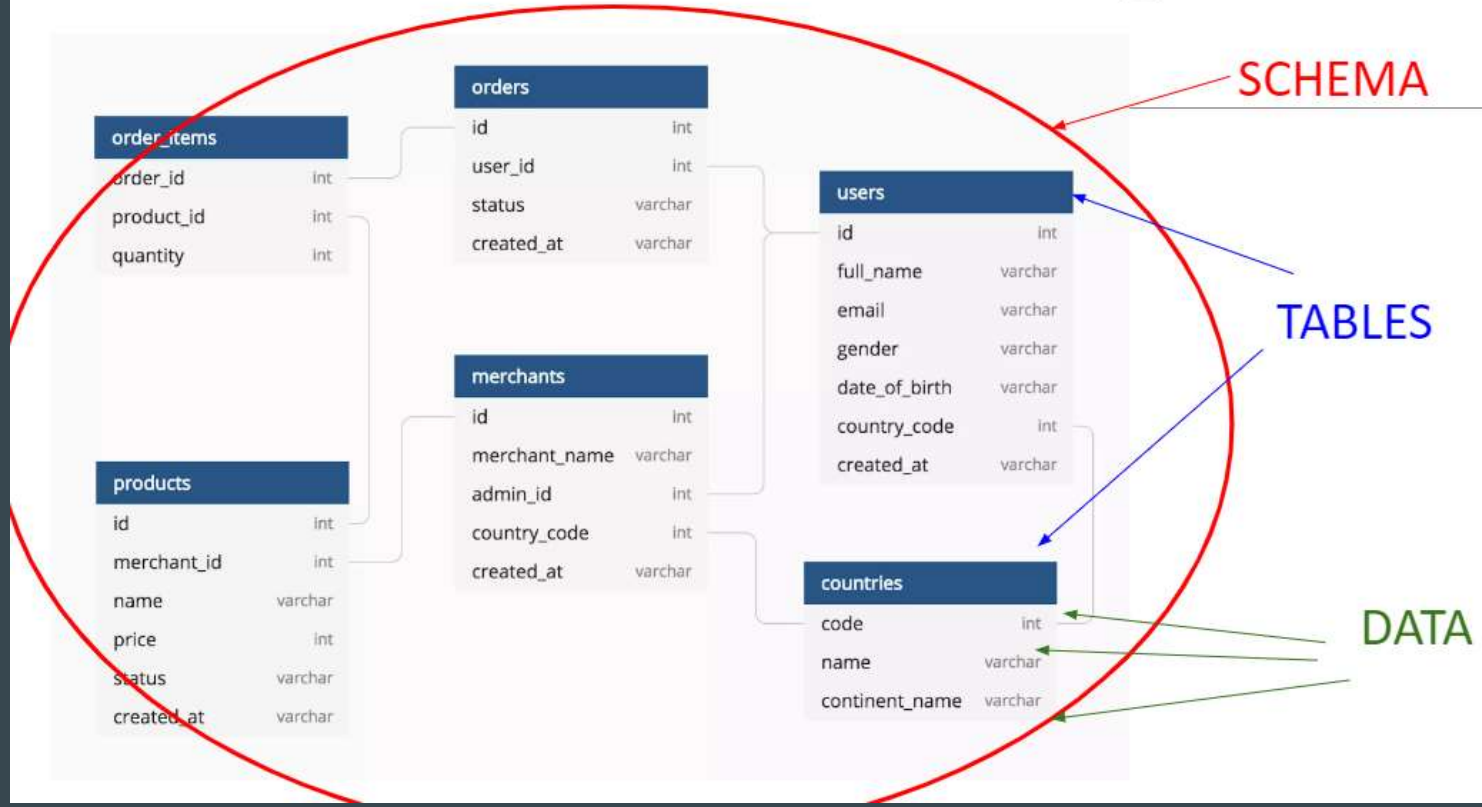
id	first_name	last_name	gender
1	Chris	Martin	M
2	Emma	Law	F
3	Mark	Watkins	M
4	Daniel	Williams	M
5	Sarah	Taylor	F

RDBMS Vendors

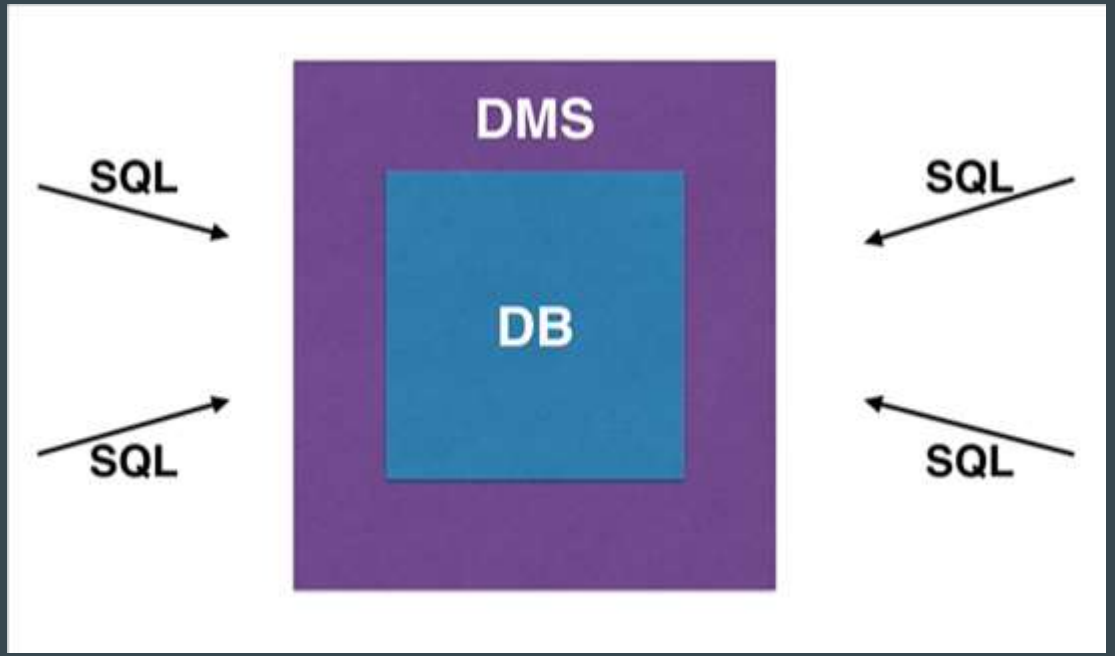
- **Oracle DB**
- MySQL
- Mircrosoft SQL Server
- **PostgreSQL (Postgres)**
- MariaDB
- SQLite

MariaDB

How do we create and manage..... in RDBMS?



Databases allow us to access and interact with data with **Structured Query Language (SQL)**



SQL is used to perform tasks on a database

...With SQL: Structured Query Language

Statements

The code below is a SQL statement. A *statement* is text that the database recognizes as a valid command.

`table_name` refers to the name of the table that the command is applied to.

`CREATE TABLE` is a *clause*. Clauses perform specific tasks in SQL.

```
CREATE TABLE table_name (
    column_1 data_type,
    column_2 data_type,
    column_3 data_type
);
```

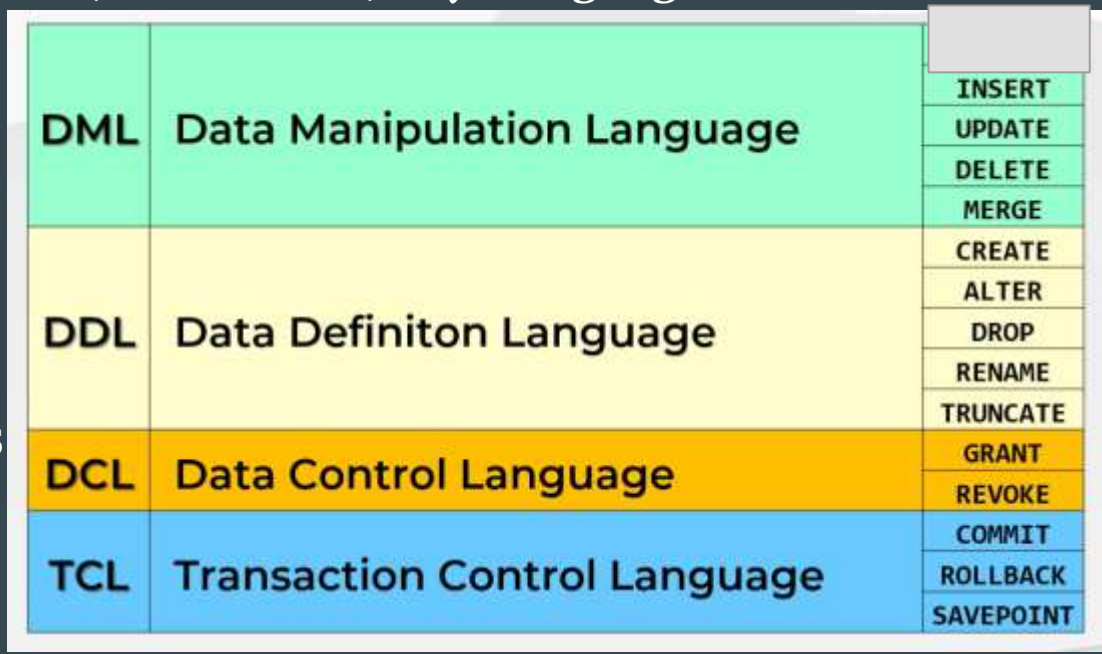
`(column_1 data_type, column_2 data_type, column_3 data_type)` is a *parameter*. A parameter is a list of columns, data types, or values that are passed to a clause as an argument.

SQL Sub-Languages

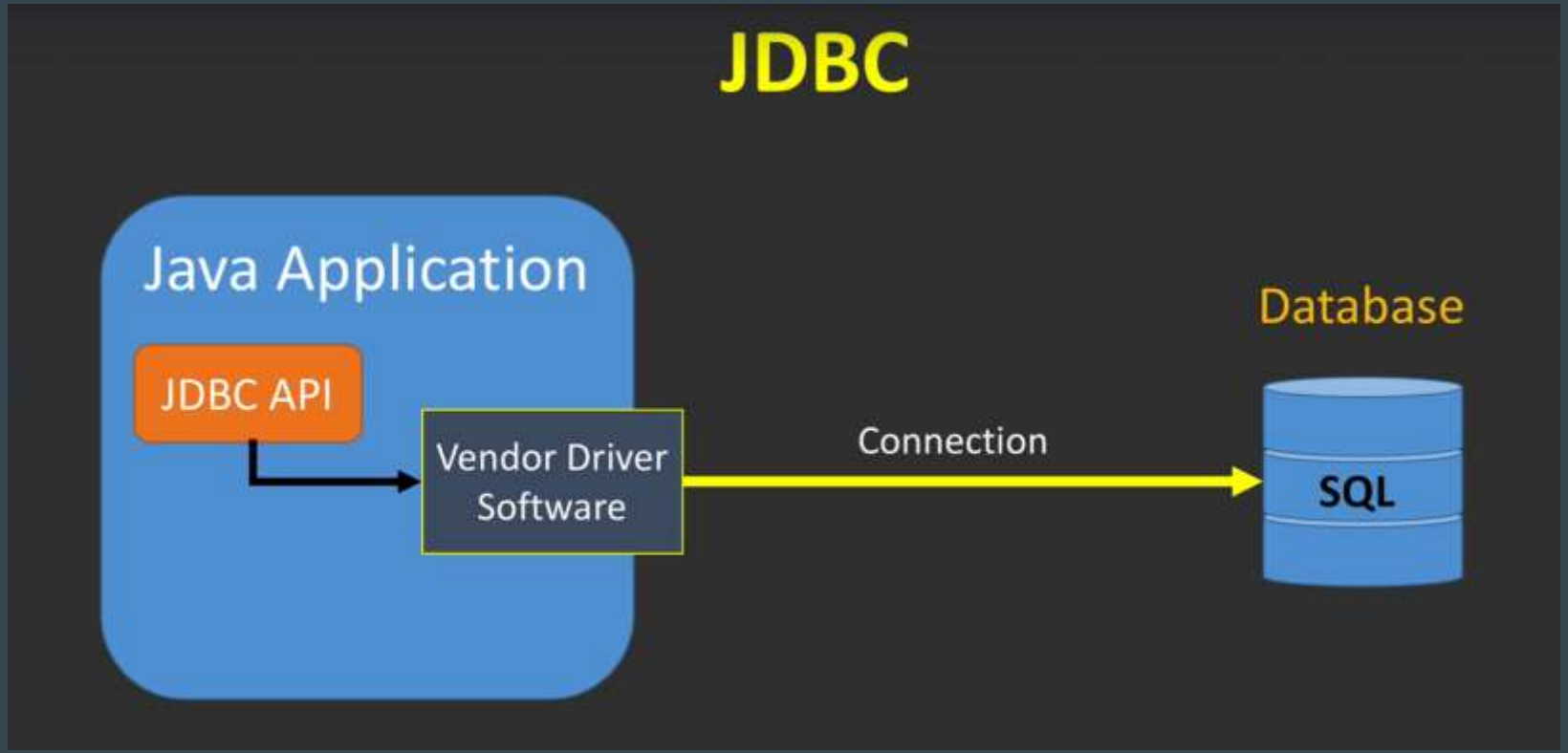
Remember this slide!

- **DDL**: Data Definition Language
  ....manages DB structure - the SCHEMA - tables
- **DML**: Data Manipulation Language
  ....manages data WITHIN the tables- cells
- **DQL**: Data Query Language
  ....**RETRIEVES data (SELECT)**
- **DCL**: Data Control Language
  ....controls **SECURITY** and **ACCESS**
- **TCL**: Transaction Control Language.
  ....manages **changes in RDBMS**

DQL → Data Query Language SELECT

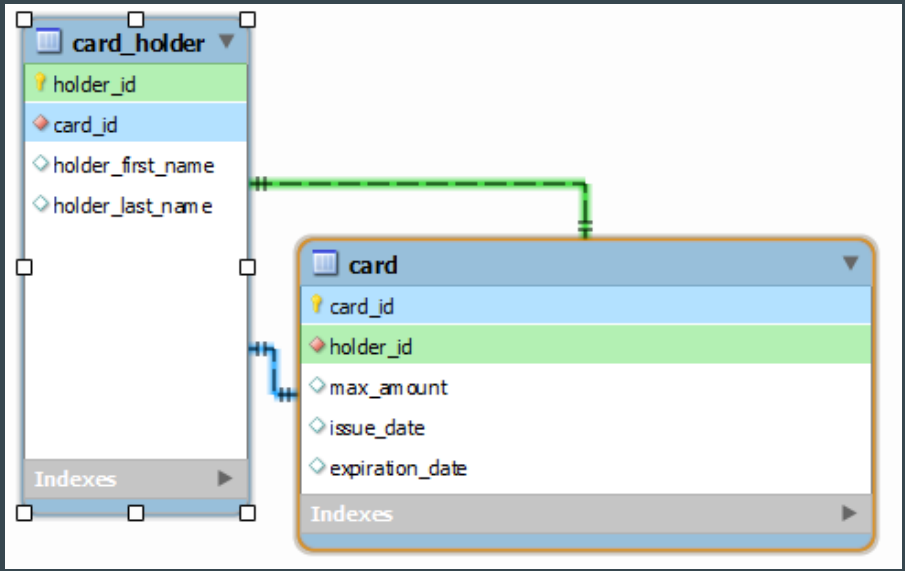


Java Database Connectivity API



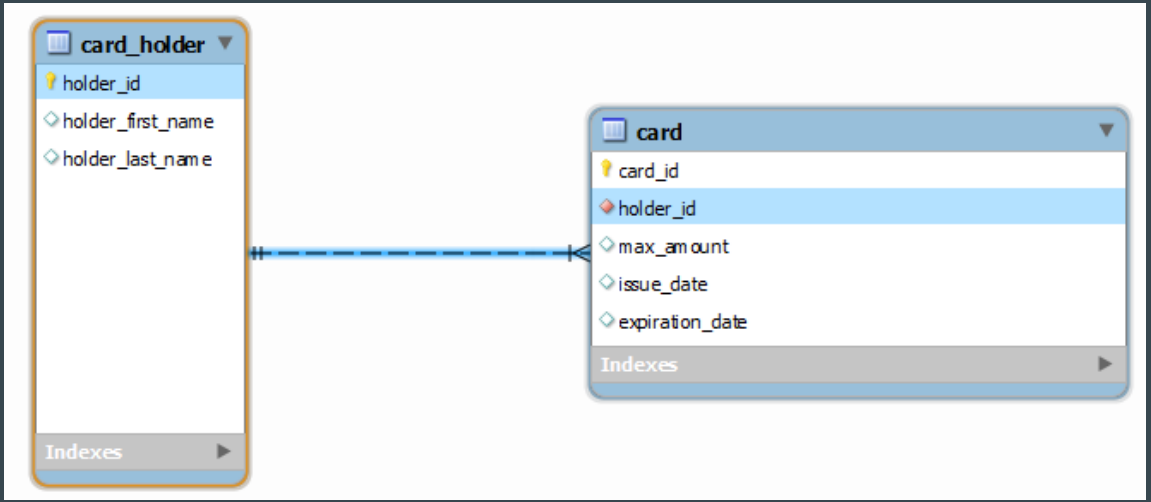
Cardinality: 1-to-1 Relationship (1:1)

Think of a credit card company that has two tables: a table for the person who gets the card and a table for the card itself.



Cardinality: 1-to-many Relationship (1:n)

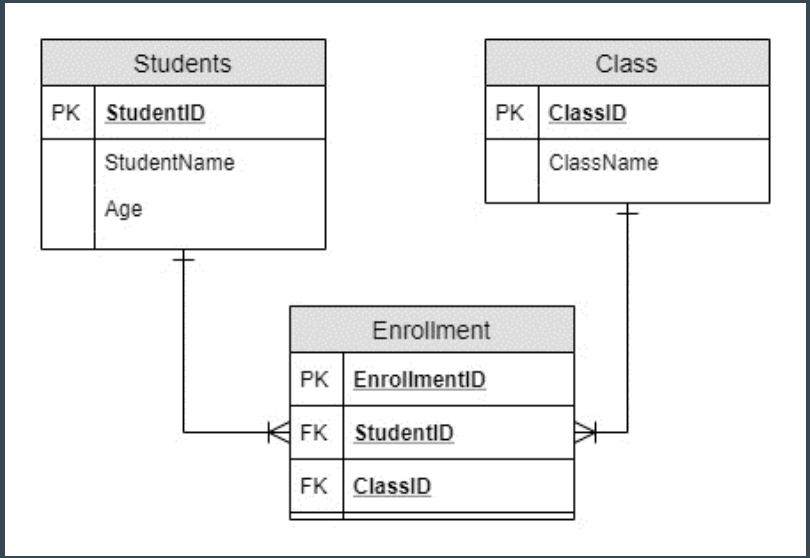
If the card holder can have multiple cards it would be a one to many relationship:



Many to Many - n:n

Best represented using a joins table

- Records in both tables are associated with many other records
- Think of many students to many classes
- Many classes have many students



Normalization - a technique of organizing data into multiple related tables to minimize data redundancy and ensure referential integrity

1. 0 Normal Form - 0NF : chaos!

This is NOT YET in 1NF because the color column has multiple values...

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

1st Normal Form - 1NF

All data should be ATOMIC and have a unique identifier....

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

0NF

TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

1NF

2nd Normal Form - 2NF

- It must be in 1NF
- All values are identified by a single column
- No partial dependencies - (all non-key attributes are full functional dependent on the primary key)

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

1NF

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

2NF

3rd Normal Form

- Must be in 2NF first
- NO transitive dependencies -> (this means that no column is dependent on another column that ISN'T primary key.)

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

2NF

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

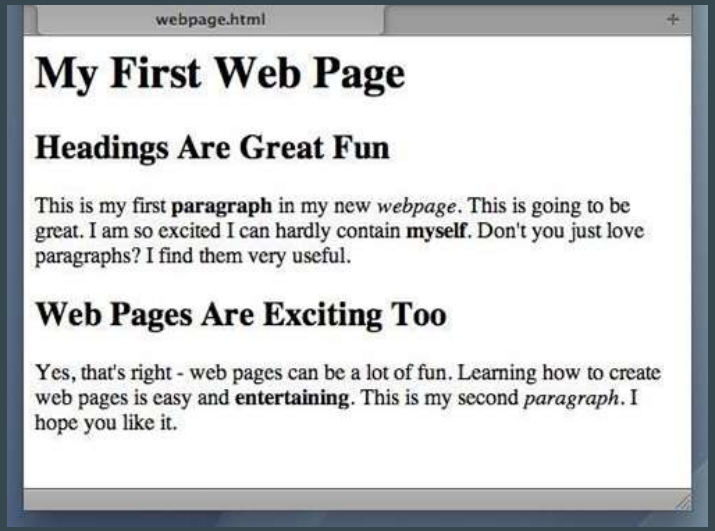
Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

3NF

HTML, CSS, JAVASCRIPT

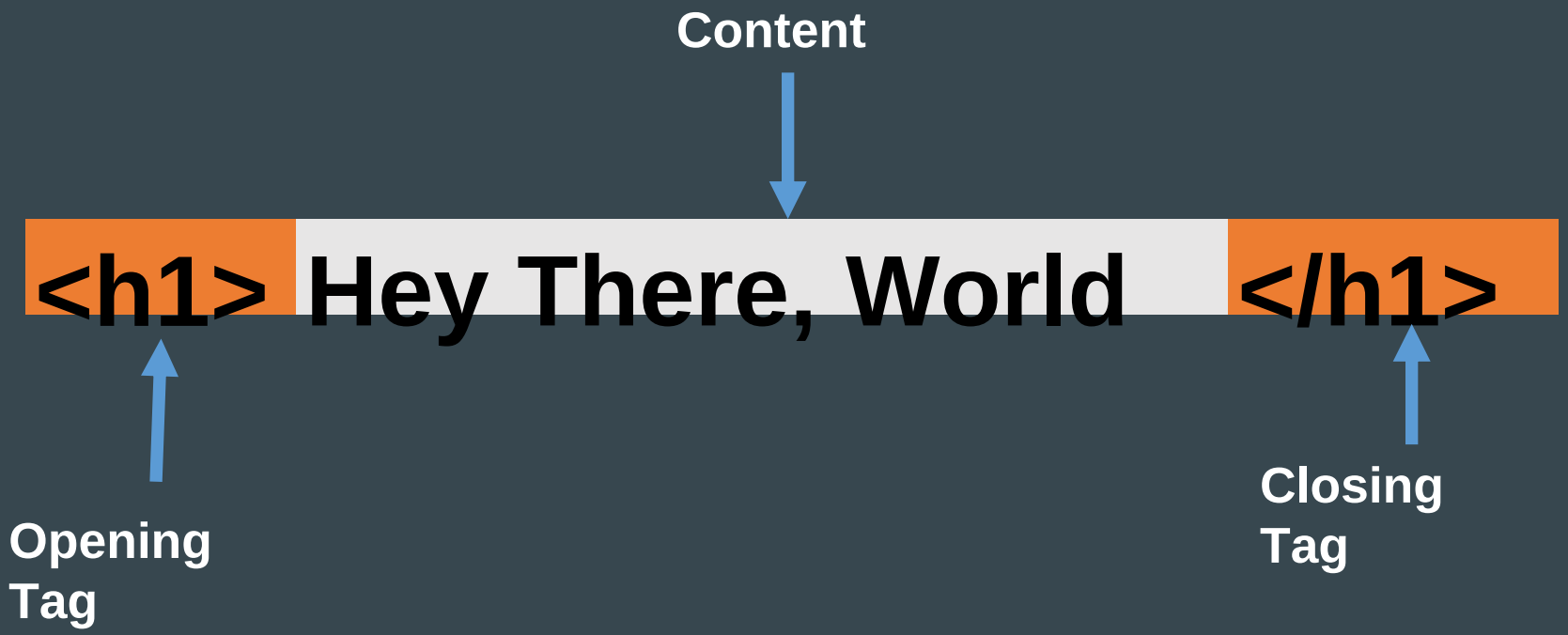
...

<title> Intro to HTML </title>

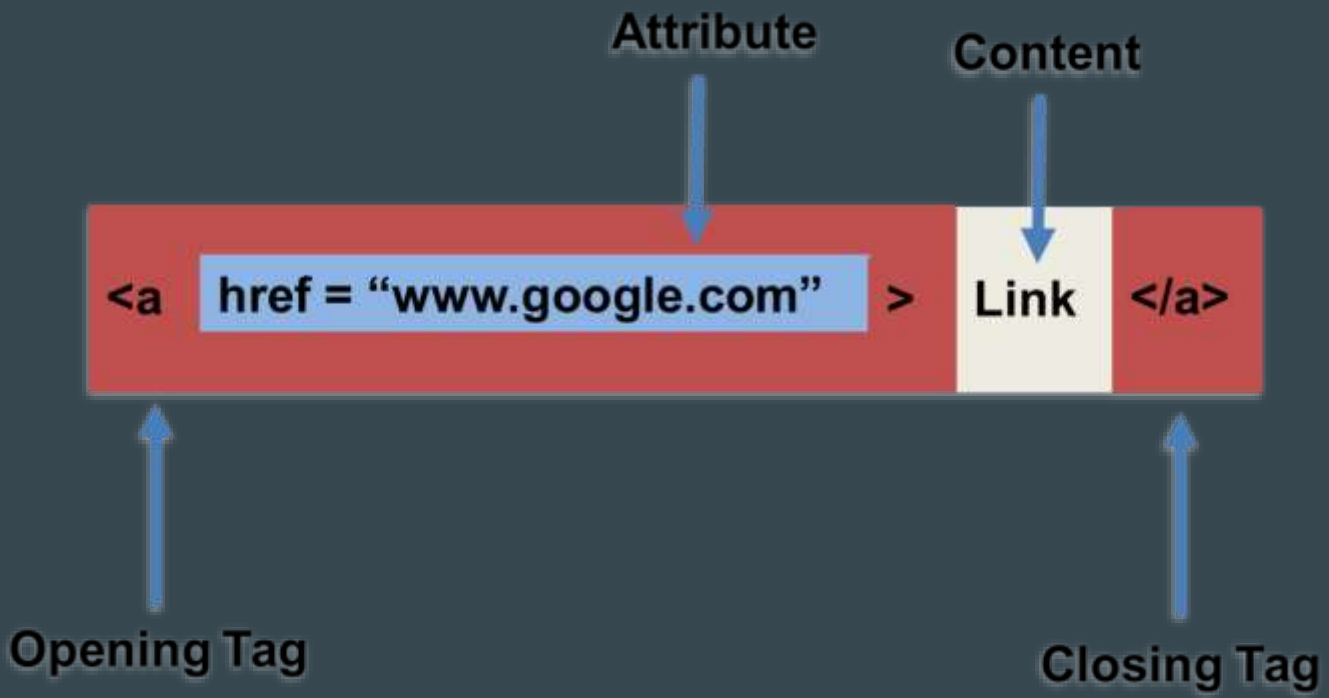


- **HTML** is one of the three base languages behind every single website.
- It defines all of the basic content and a *bit* of formatting.
- HTML elements tell the browser how to display the content.

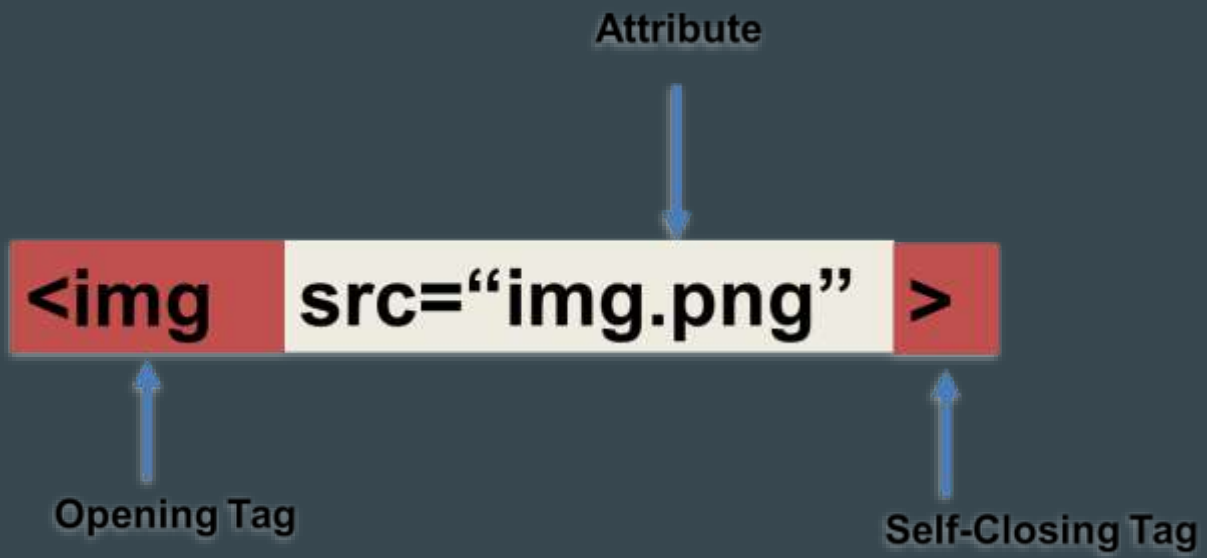
HTML Syntax (Basic)



HTML Syntax (with Attribute)



Tricky Tags (Self-Closing)



Important Common Tags

Headings:

- **<h1> </h1>** - Heading 1 (Largest Heading)
- **<h2> </h2>** - Heading 2 (Next Largest Heading)
- **<h3> </h3>** - Heading 3
- **…**

Containers:

- **<html> </html>** - Wraps the entire page
- **<head> </head>** - Wraps the header of the page
- **<body> </body>** - Wraps the main content
- ** ** - Logical Container ***
- ** ** - Wraps individual Paragraphs

Others:

- **** (bold), **** (emphasis)
- **<img>** (images)**, <a href>** (links)**, ** (list items) **, <title>** (title), **
** (line break), **<table>** (tables), **<!-- -->** (comments)

Less Common Tags

- **All HTML Tags are listed here:** http://www.w3schools.com/tags/

- Don’t try to memorize them! Simply refer back to documentation as needed.

- Other tags:
  - <video> for Videos
  - <audio> for Audio files
  - <embed> for Embedded files
  - <code> for including computer code
  - <header> for headers
  - <nav> for navigation bars
  - <footer> for footers

HTML for Forms

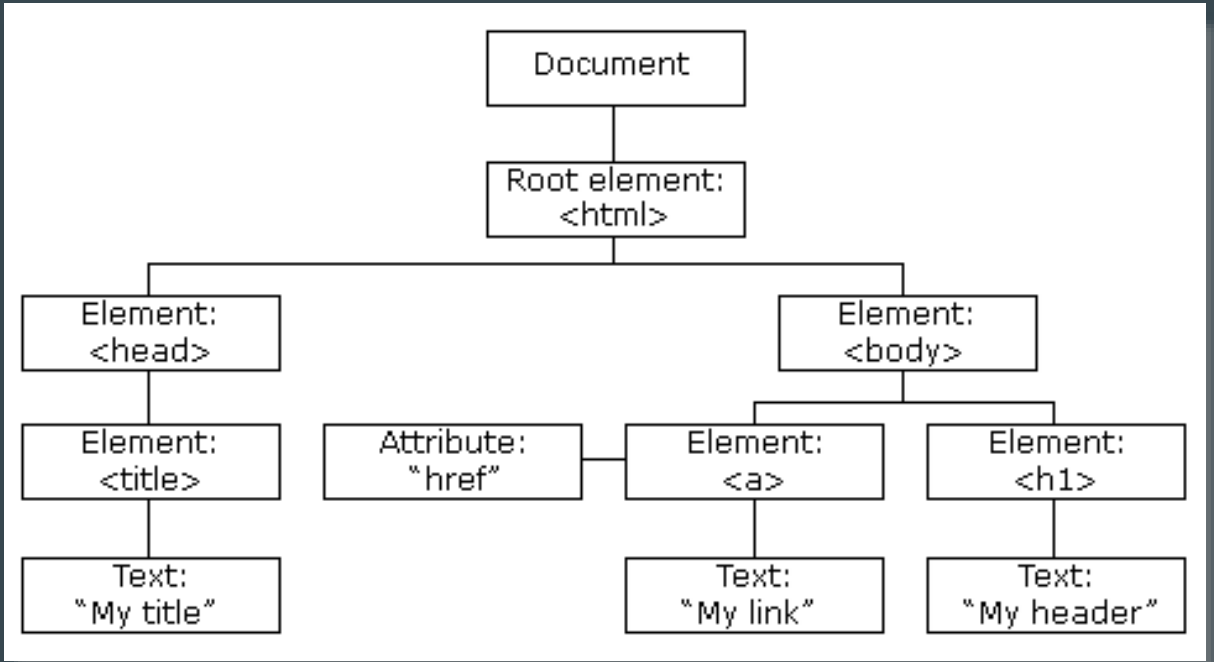
Common UI (User Interface) Form Elements:

- **<form>** - Creates a form section in HTML
- **<input>** - Input boxes
- **<label>** - Labels for boxes
- **<button>** - Button
- **<textarea>** - Large textbox

DOM: Document Object Model

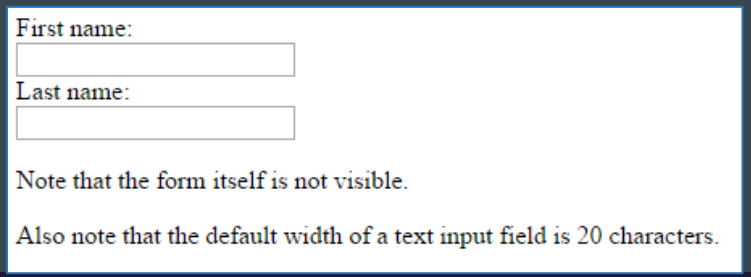
The Document Object Model is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document.

- When a web page is loaded, the browser creates a **Document Object Model** of the page.
- The **HTML DOM** is constructed as a tree of **Objects**
- We can access these properties and stylize them with CSS or code their behavior with JS

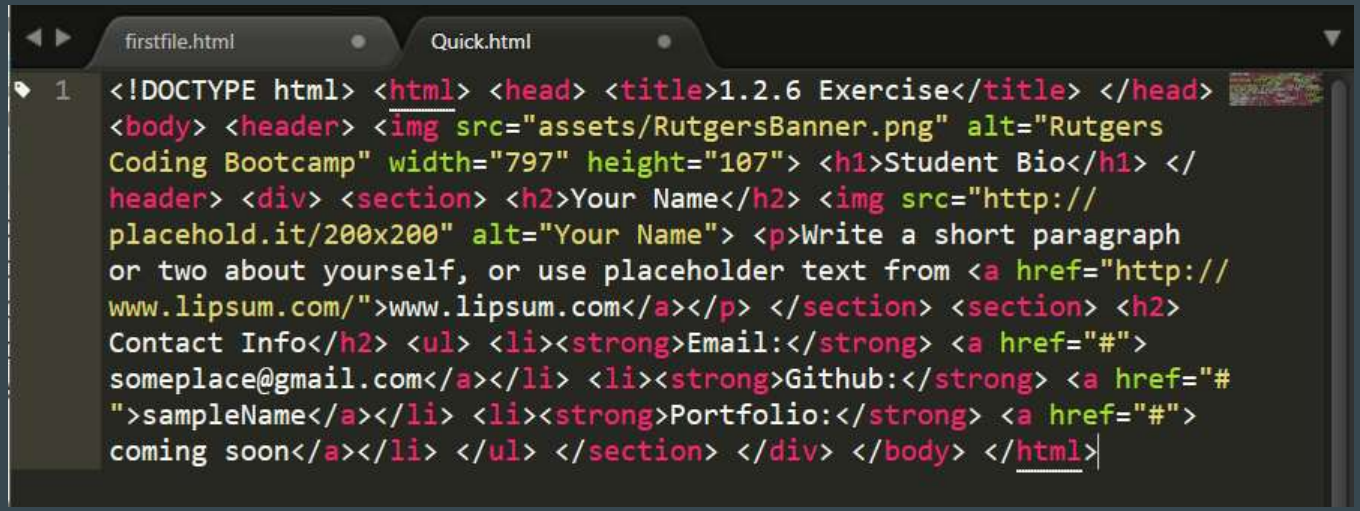


HTML for Forms

```
<!DOCTYPE html>
<html>
<body>

<form>
  First name:<br>
  <input type="text" name="firstname">
  <br>
  Last name:<br>
  <input type="text" name="lastname">
</form>

<p>Note that the form itself is not visible.</p>

<p>Also note that the default width of a text input field is 20 characters.</p>

</body>
</html>
```



On Ugly HTML



- Don’t do this… Use proper indentation and sectioning.
- Readable code is easier to maintain.
- Invest time to get better about this now. It will pay dividends!

CSS - Cascading Style Sheets

...

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

HTML / CSS Definitions

- **HTML:** Hypertext Markup Language – (Content)
- **CSS:** Cascading Style Sheets – (Appearance)
- **HTML/CSS are the "languages of the web."** Together they define both the content and the aesthetics of a webpage – handling everything from the layouts, colors, fonts and content placement. (JavaScript is the third – handling logic, animation, etc.)

HTML / CSS Analogy

HTML Alone

- Like writing papers in “Notepad.”
- Can only write unformatted text.

HTML / CSS

- Like writing papers in Microsoft Word.
- Can format text, page settings, alignment, etc. based on “highlighting” and menu options.

Basic HTML Page

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>My First Website!</title>
</head>

<body>

    <h1>Awesome Header</h1>
    <h2>Smaller Awesome Header</h2>
    <h3>Even Smaller Header</h3>

    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
        incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
        exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</p>
    <img src="https://vignette.wikia.nocookie.net/Bill_Murray.jpeg/" alt="bill murray" width="25%" >

    <h3>Menu Links</h3>
    <ul>
        <li><a href="http//www.google.com">Google</a></li>
        <li><a href="http//www.facebook.com">Facebook</a></li>
        <li><a href="http//www.twitter.com">Twitter</a></li>
    </ul>

</body>
</html>
```

Basic HTML Page - Result

Awesome Header

Smaller Awesome Header

Even Smaller Header

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quidem consequatur unde aut dolores odio hic, accusamus recusandae ipsam illum enim voluptatibus obcaecati totam tempora eum quod sapiente. Corporis, quidem, culpa?

Menu Links

- Google
- Facebook
- Twitter

Basic HTML Page - Result

Awesome Header

Smaller Awesome Header

Even Smaller Header

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quidem consequatur unde aut dolores odio hic, accusamus recusandae ipsam illum enim voluptatibus obcaecati totam tempora eum quod sapiente. Corporis, quidem, culpa?

Menu Links

- Google
- Facebook
- Twitter

Very Boring…

Enter CSS

```
<style>
    h1 {
        font-size: 60px;
        text-align: center;
        margin-bottom: 15px;
        text-decoration: underline;
        background-color: black;
        color: white;
    }

    h2 {
        font-size: 40px;
        text-align: center;
        margin-top: 15px;
        margin-bottom: 15px;
    }

    h3 {
        font-size: 20px;
        text-align: center;
        margin-top: 15px;
    }
```

```
    img {
        display: block;
        margin-left: auto;
        margin-right: auto;
    }

    p {
        text-align: center;
        font-size: 20px;
        font-weight: bold;
    }

    ul {
        text-align: center;
        font-size: 35px;
        list-style-position: inside;
        border-style: solid;
        border-width: 5px;
    }
</style>
```

Enter CSS - Result

Awesome Header

Smaller Awesome Header

Even Smaller Header

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quidem consequatur unde aut dolores odio hic, accusamus recusandae ipsam illum enim voluptatibus obcaecati totam tempora eum quod sapiente. Corporis, quidem, culpa?

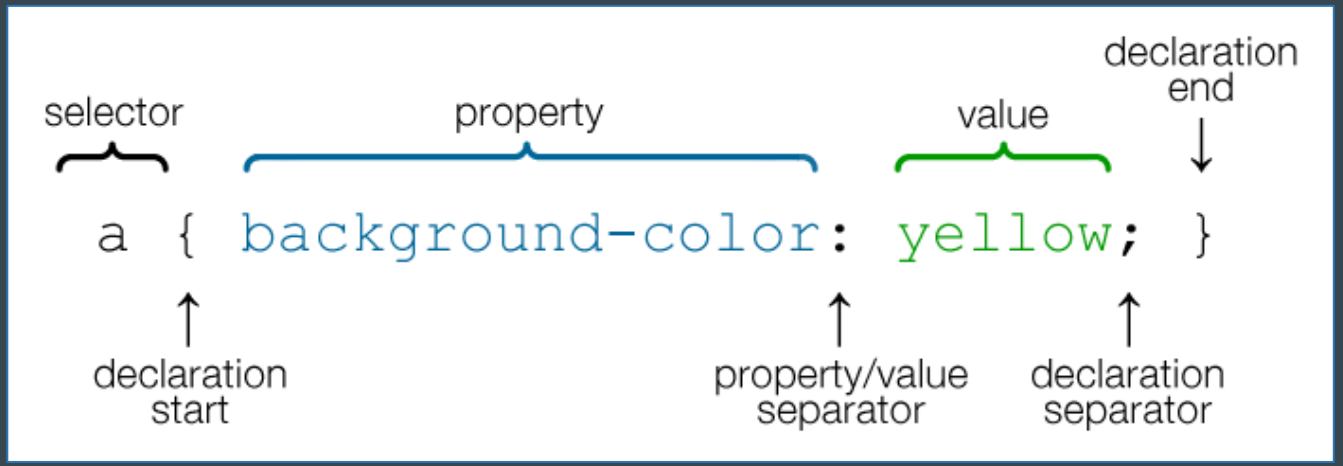
Menu Links

- Google
- Facebook
- Twitter

CSS Syntax

- CSS works by hooking onto **selectors** added into HTML using **classes** and **identifiers.**

- Once hooked, we apply **styles** to those HTML elements using CSS.



CSS Example

- In the below example the “Header” would be turned blue and MUCH larger because of the CSS.

- We can incorporate an element’s class or ID to apply a CSS style to a particular part of the document.
  - Just remember to include the necessary symbol before the CSS: “.” for class, “#” for ID.

Example (HTML):

```
<p class=“bigBlue”>Header</p>
```

Example (CSS):

```
.bigBlue
{
        font-size: 100px;
        color: blue;
}
```

Key CSS Attributes

Font / Color:

- **color**: Sets color of text.
- **font-size**: Sets size of the font.
- **font-style**: Sets italics.
- **font-weight**: Sets bold.

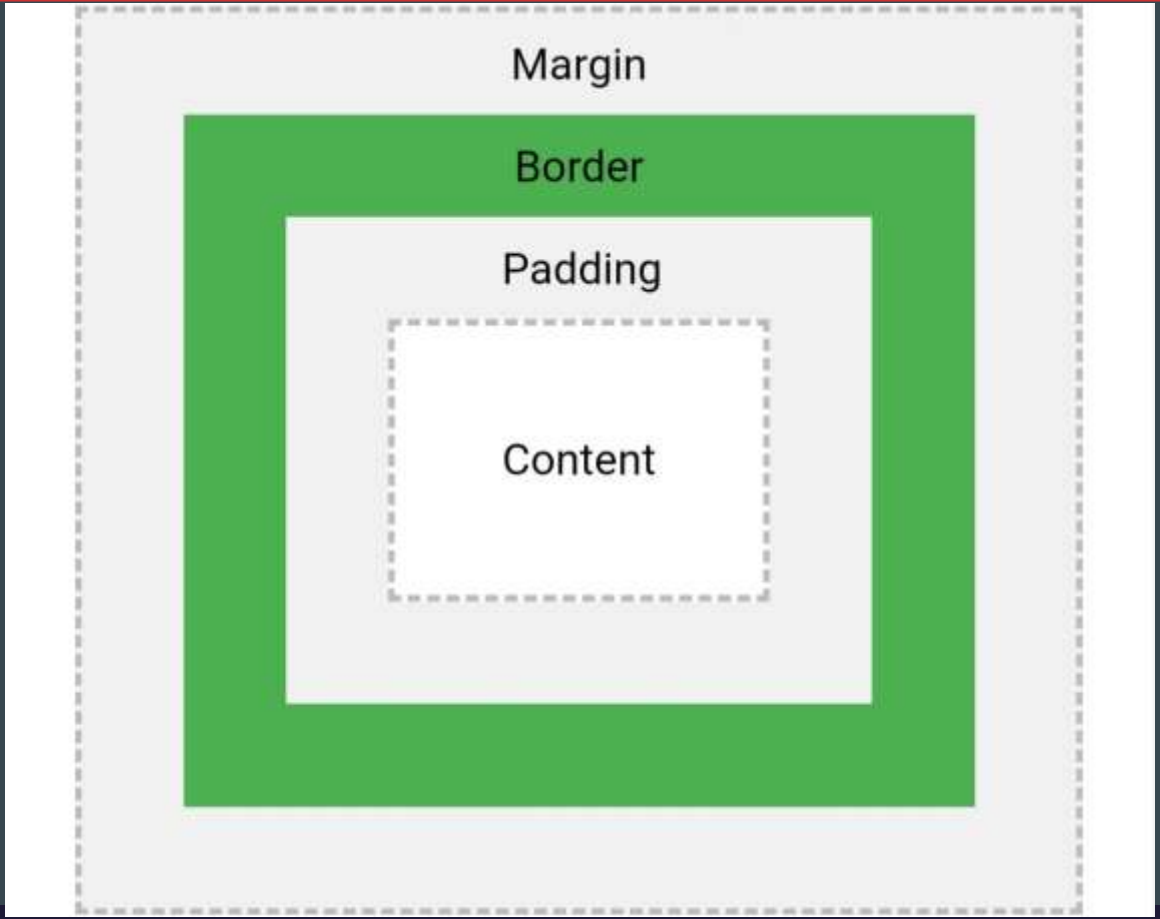
Alignment / Spacing:

- **padding (top/right/bottom/left):** Adds space between element and its own border.
- **margin (top/right/bottom/left):** Adds space between element and surrounding elements.
- **float:** Forces elements to the sides, centers, or tops.

Background:

- **background-color:** sets background color.
- **background-image:** sets background image.

CSS Box Model



Amazon Simple Storage Service (S3)

...

Amazon S3 is object storage built to store and retrieve any amount of data from anywhere on the Internet.

S3 Use Cases

- You can store any type of file in S3
- S3 is designed to deliver 99.999999999% durability

- Typical use-cases include:
  - Backup and Storage - provide data backup and storage services for others
  - Application Hosting - provide services that deploy, install, and manage web apps
  - Media Hosting - build a redundant, scalable, and highly available infrastructure that hosts video
  - Software Delivery - Host your software applications which customers can download
  - Static Websites - configure a static website to run from an S3 bucket.

S3 Buckets

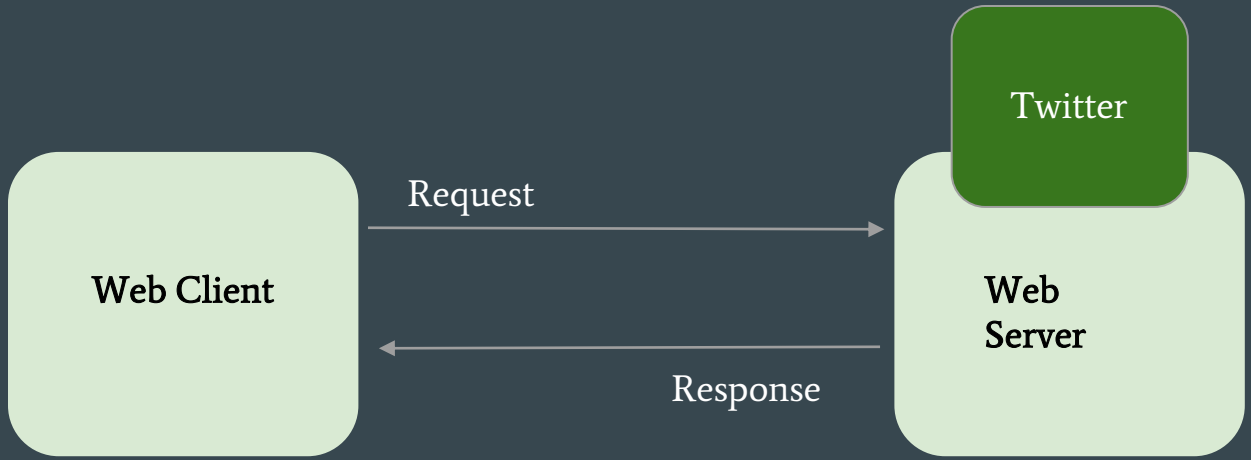
- Files are stored in **buckets**
- **Buckets are root level folders**
- Buckets store objects, which consist of data and its descriptive metadata.
- Unlimited storage available
- You create buckets within a REGION

- Objects consist of:
  - Key (name of obj)
  - Value (data made up of a sequence of bytes)
  - Version ID (used for versioning)
  - Metadata (data about the data that is stored

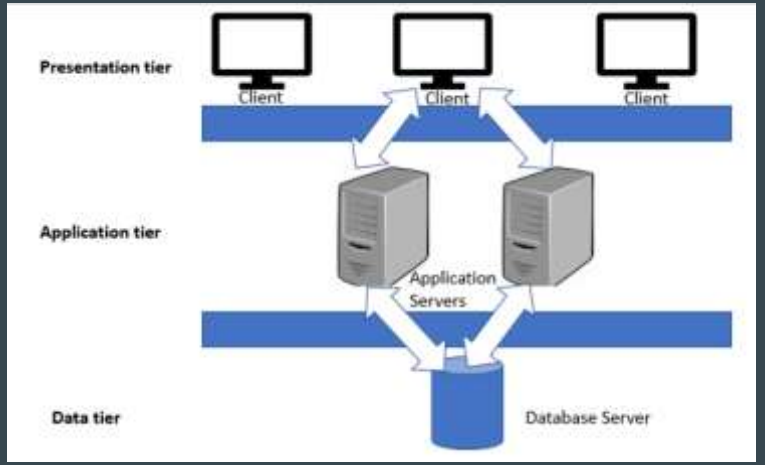
Client/Server Communication & Servlets

•••

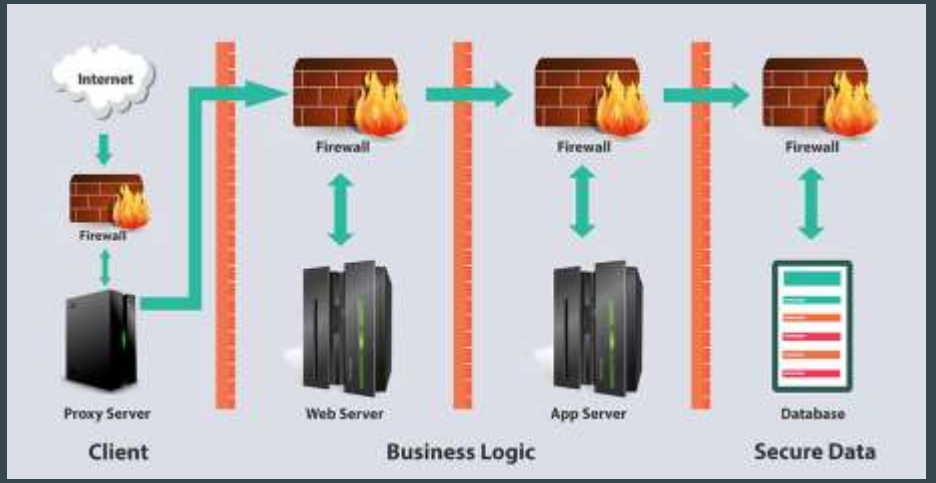
Web Application



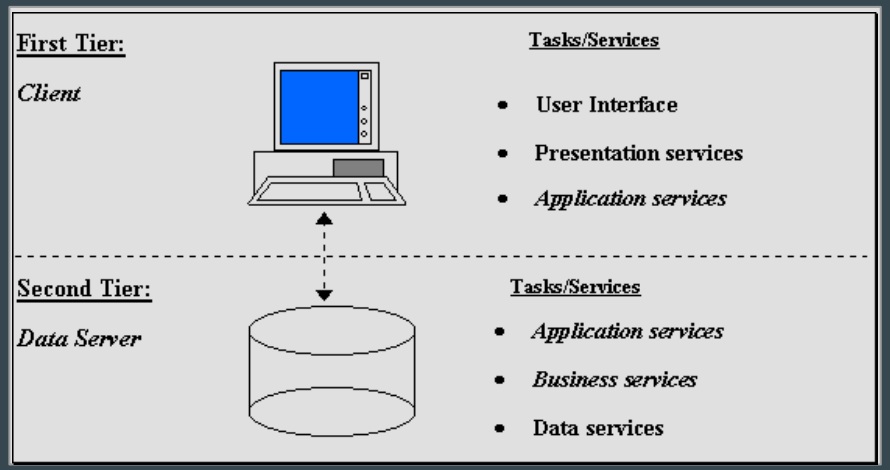
Tiered Client/Server Architecture



3 Tier Architecture



n - Tier Architecture



2 Tier Architecture

URL

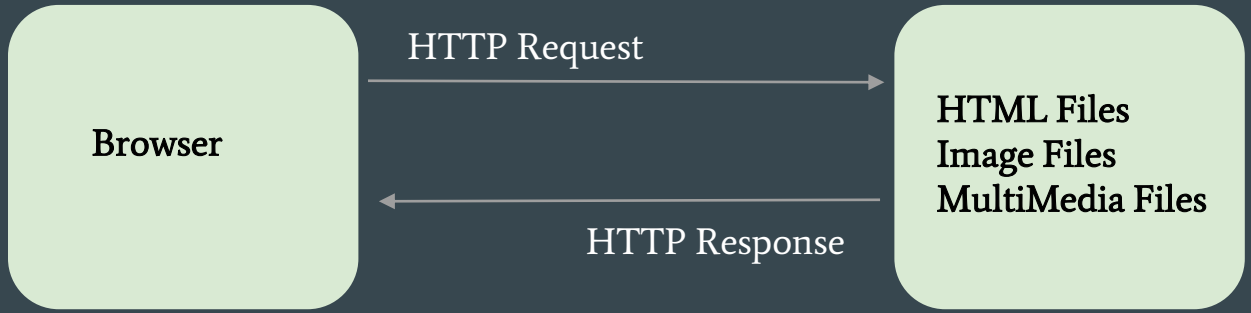
Universal Resource Locator

URL: http://www.twitter.com:80

protocol domain name port

Static Web Application

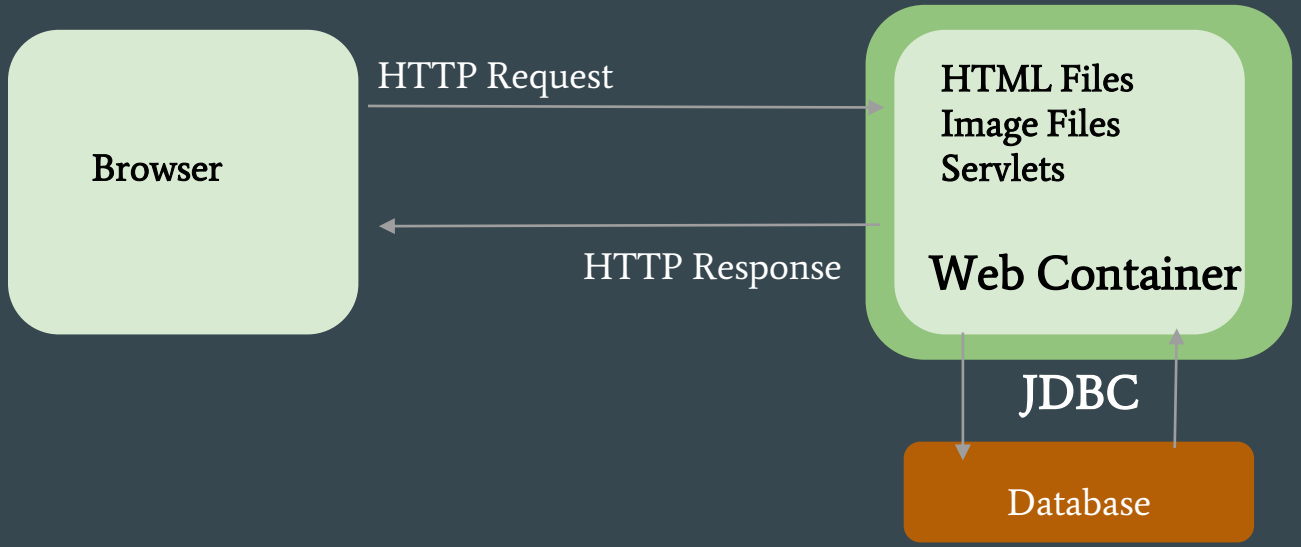
A **Static Web Application** is non-interactive and provides generic information for the client that's accessing it.



Dynamic Web Application

A **Dynamic Web Application** is comprised of Servlets and server-side components

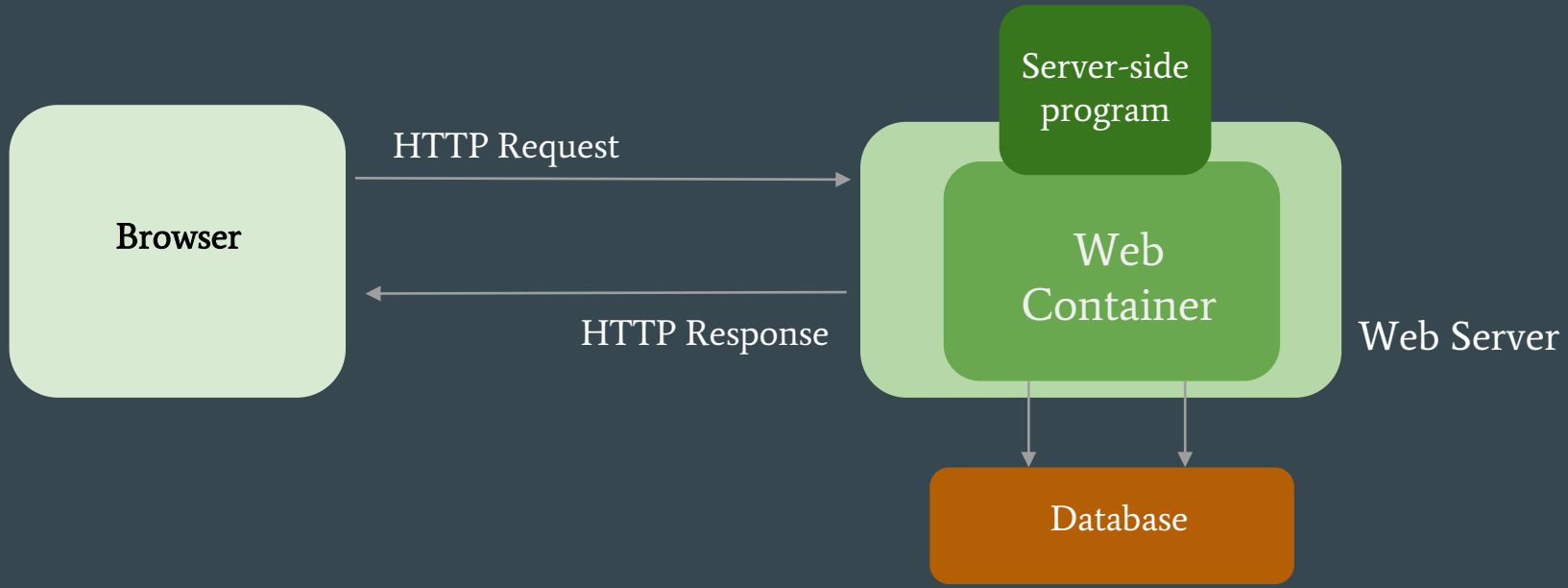
- Runs on a web container
- Server-side components generate pages *on the fly* by contacting a database



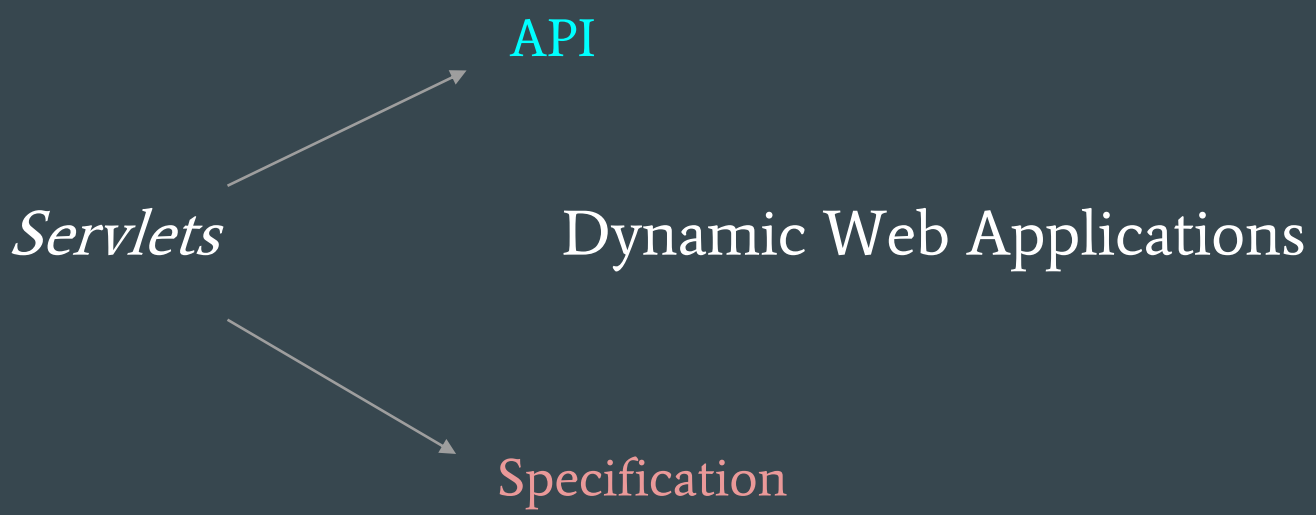
Server-Side Programming

Duties of a Server Side program:

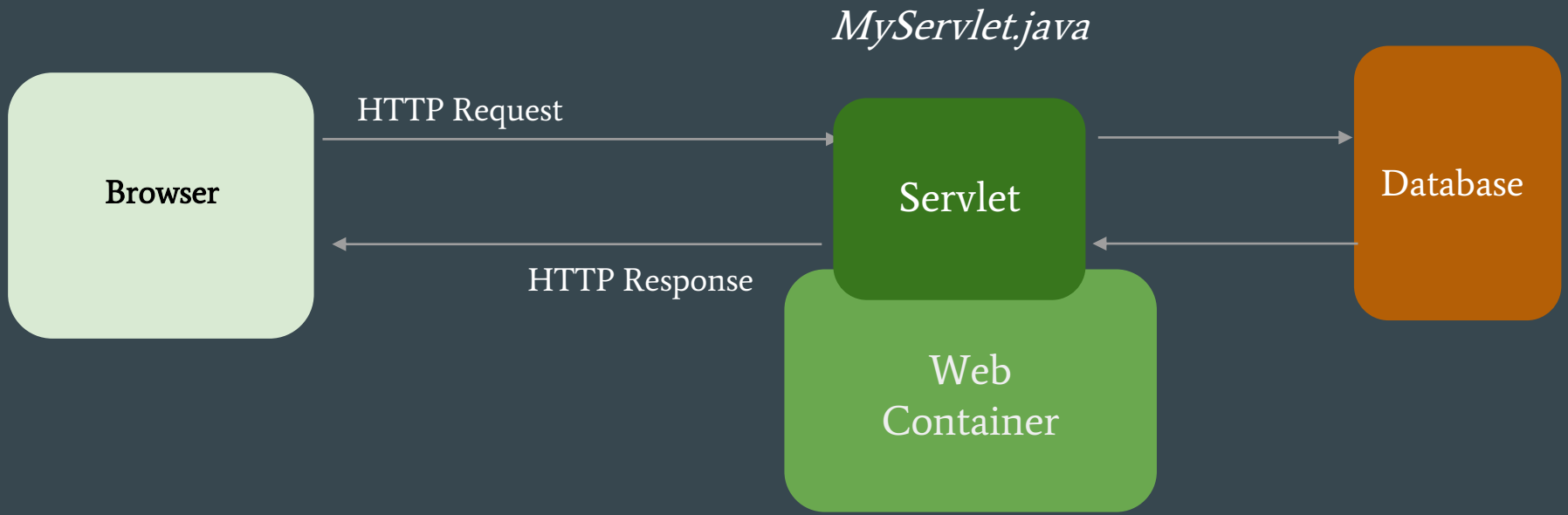
1. Handle the client request by capturing the user-input
2. Connects and communicates with Database
3. Processes Data
4. Produce the response page (a dynamic HTML response page, creating HTML on the fly!)
5. Then, it hands this response page back to the web server, which sends it as a response to the client (browser)



Servlet API and Specifications



What's a Servlet?



Life Cycle Methods and Phases

3 Life Cycle Methods of a Servlet:
(*these methods are called by the container...like Tomcat)*

1. init()
2. service()
3. destroy()

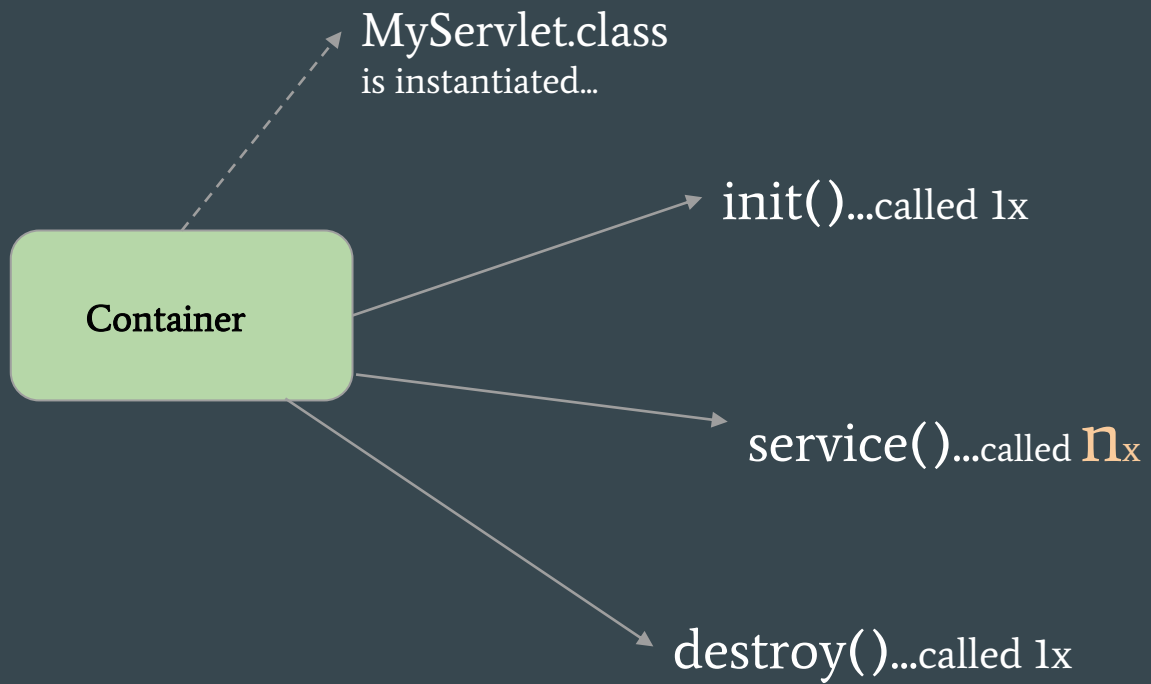
4 Life Cycle Phases associated with these methods:

1. Instantiation
2. Initialization
3. Servicing
4. Destruction

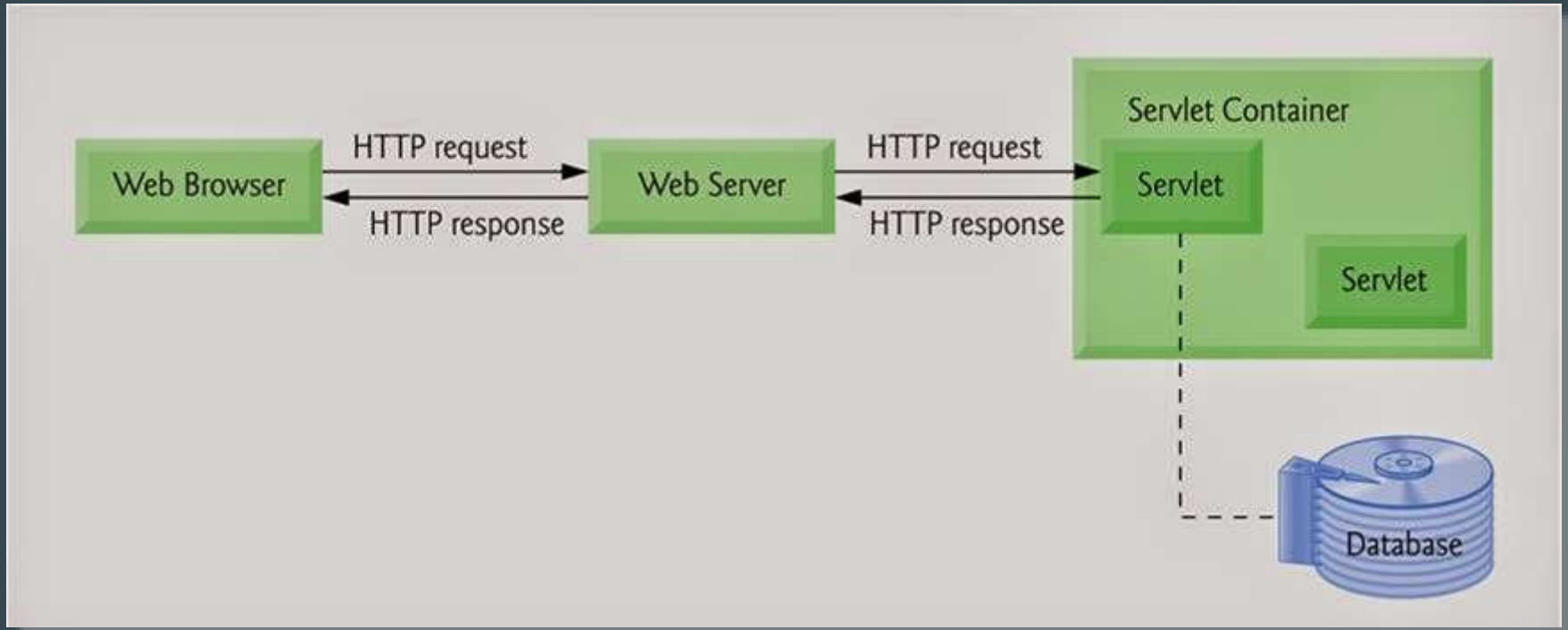
Life Cycle Methods and Phases

Life Cycle Phases:

1. Instantiation
1. Initialization
1. Servicing
1. Destruction



How do Servlets Work? (*this is the same diagram in another light)*

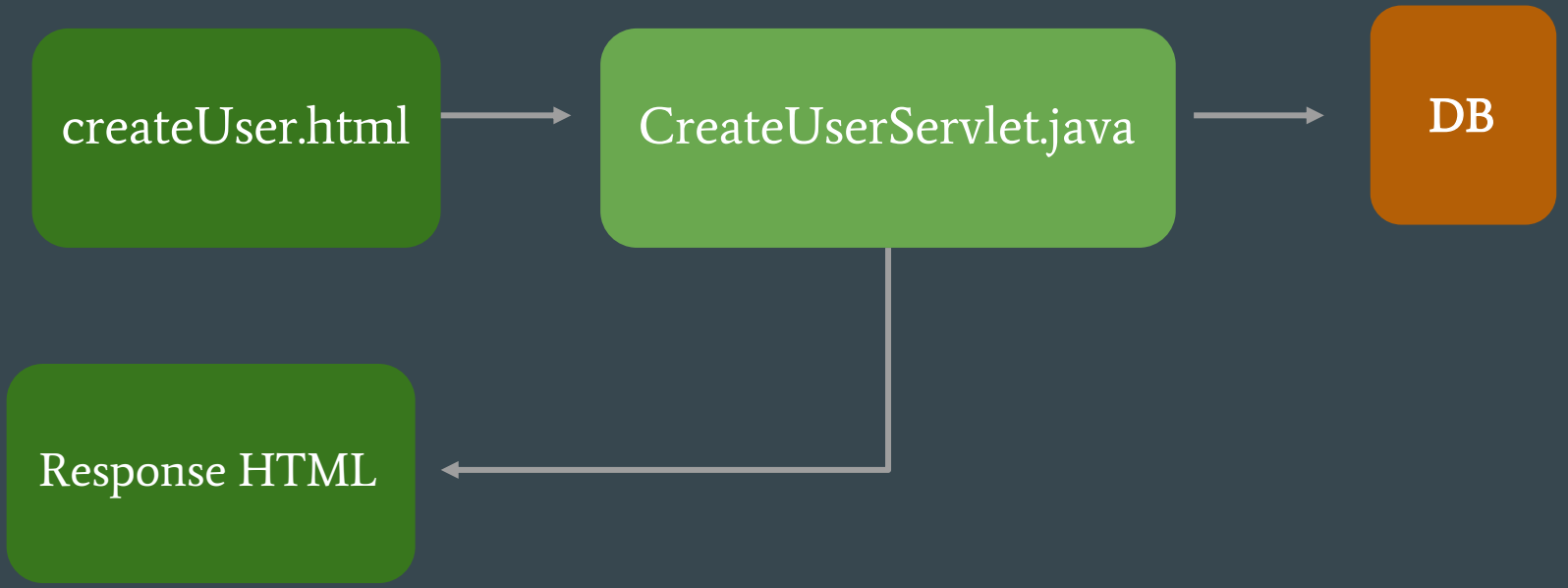


3 Steps to Create a Servlet Class:

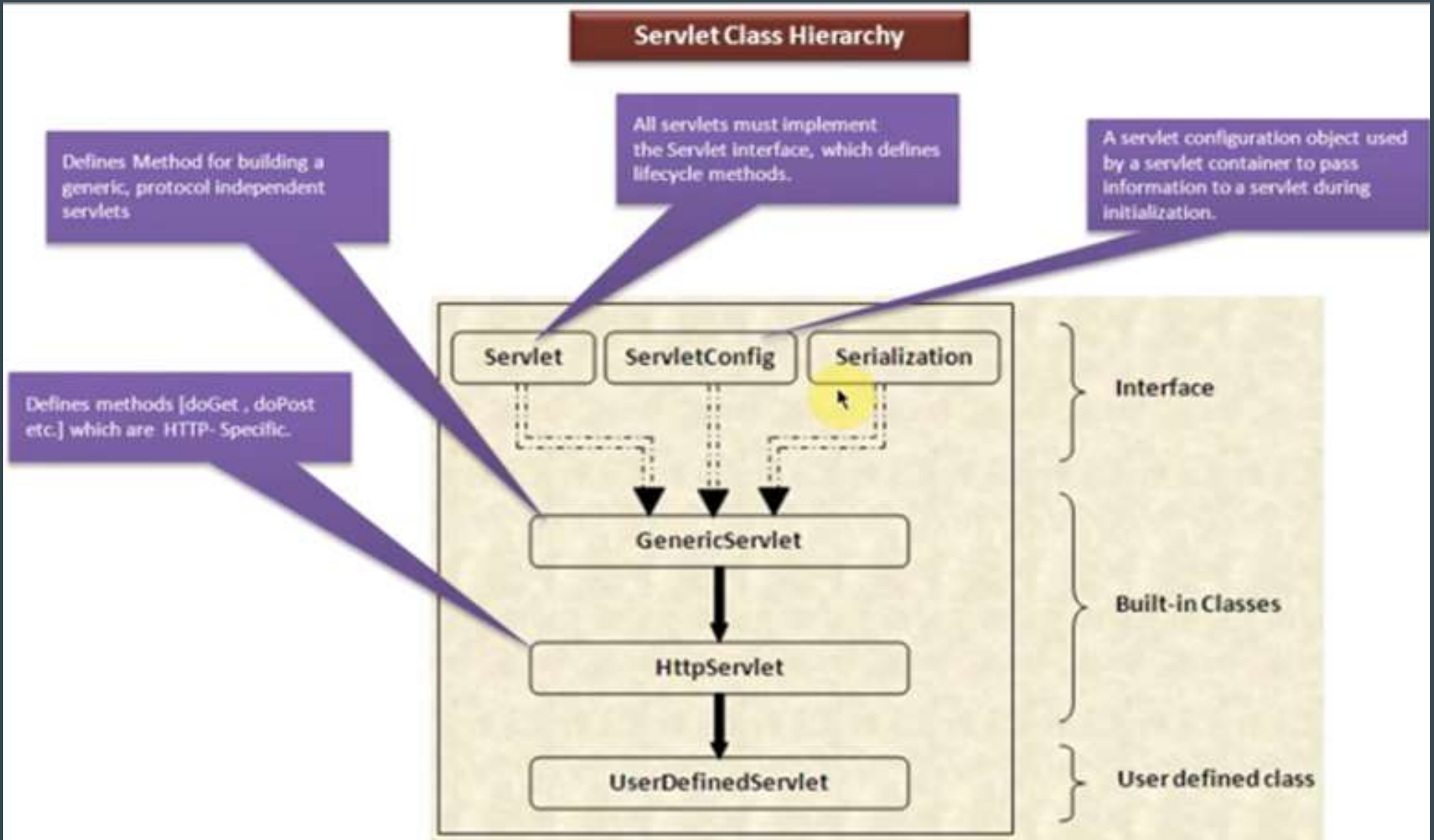
1. Create Project
2. Web Resource Development
3. Create the Deployment Descriptor

web.xml

Create User Servlet



Servlet Class Hierarchy



HTTP Methods

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

GET
POST

VS.

- “Gets” data (reads)
- No body
- Query string
- No sensitive data here!
- Data size limit
- **Idempotent**

default

- “Posts” data (Inserts/updates)
- Body
- Can be sent
- No restriction on data
- NOT idempotent(changes stuff in the DB

Mentioned explicitly:

<form method=”post”...

Initialization:

Lazy (*default*) vs. Eag

- Servlet is initialized only when the client sends a request

- Servlet is initialized before startup through configuration in web.xml

```
<servlet>
        <servlet-name>...
        <servlet-class>
        <load-on-startup> 1 </load-on-start
</servlet>

<servlet>
        <servlet-name>...
        <servlet-class>
        <load-on-startup> 2 </load-on-start
</servlet>
```

```
@WebServlet(urlPatterns = "/myServlet", loadOnStartup = 1)
```

InterServlet Communication: RequestDispatcher

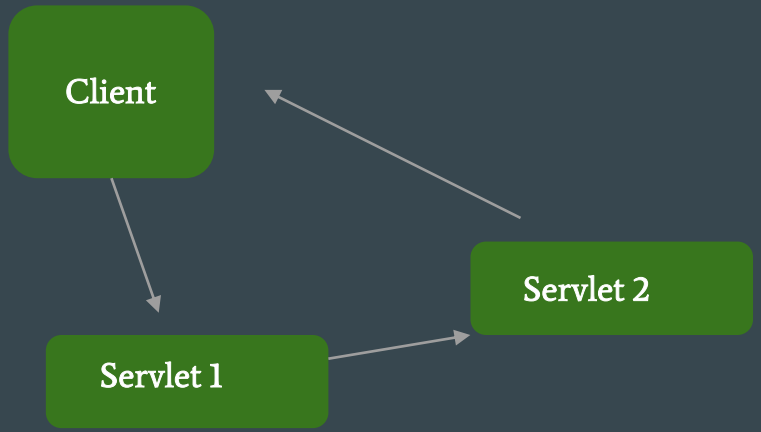
- When a servlet is done with its work it *dispatches* the request to another Servlet which can do more work
- We use RequestDispatcher interface from Servlet API
- Retrieve the RequestDispatcher using *request.getRequestDispatcher("/homepage")*
- *The (uri) = where we want to go next.*

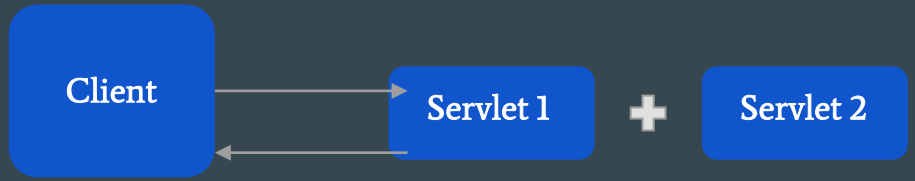
2 Ways Request Dispatching can be done:

forward(req, res)

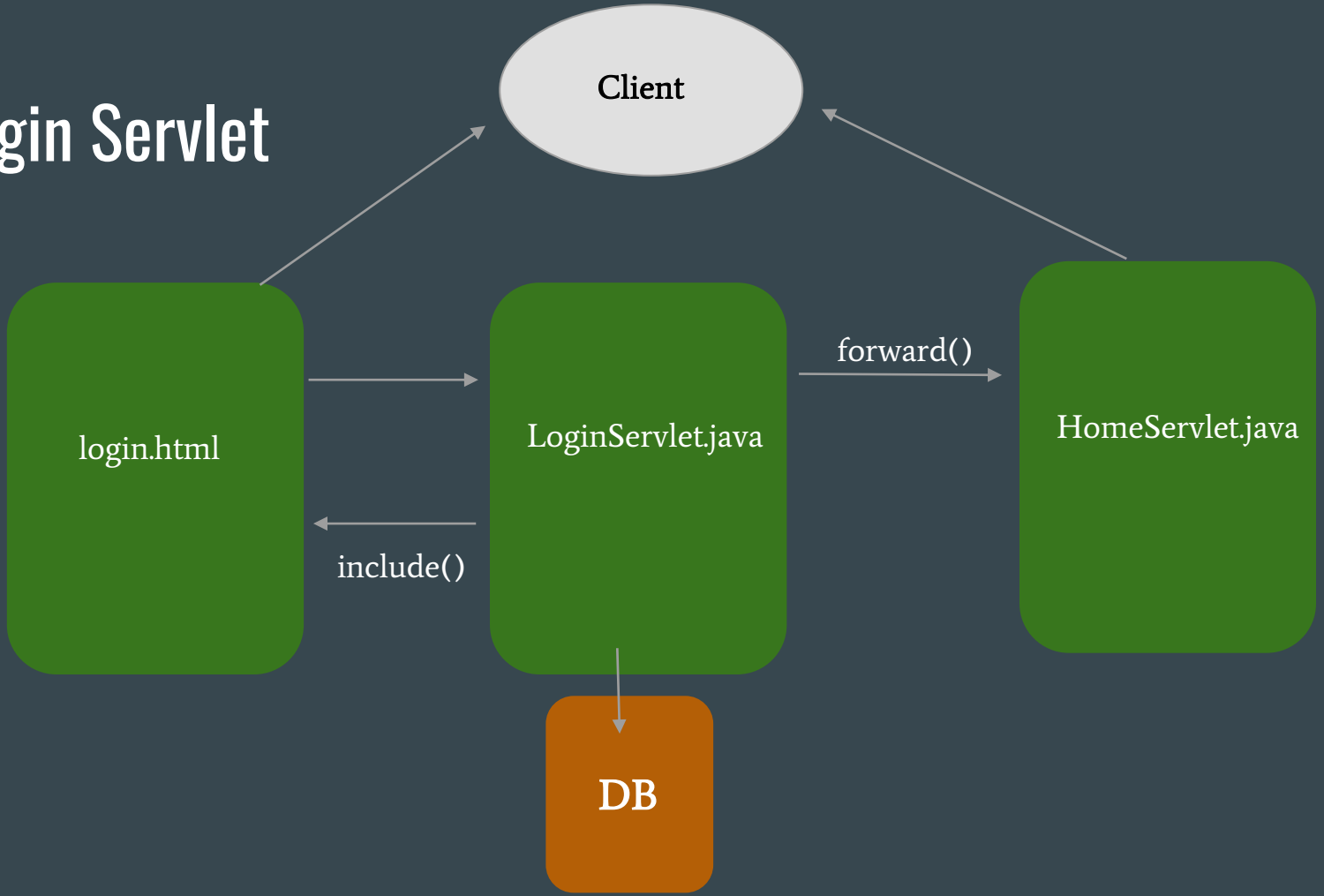
include(req, res)

vs.

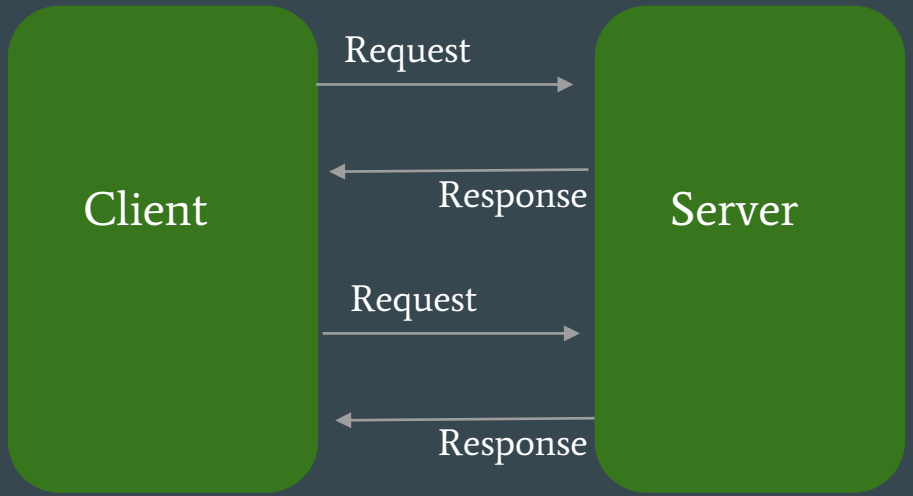




Login Servlet



HTTP is stateless !



- Connection is closed as soon as response is sent back
- Server does not remember the previous request

Session Management:

1. Create the Session using HttpSession obj

```
HttpSession session = req.getSession();
```

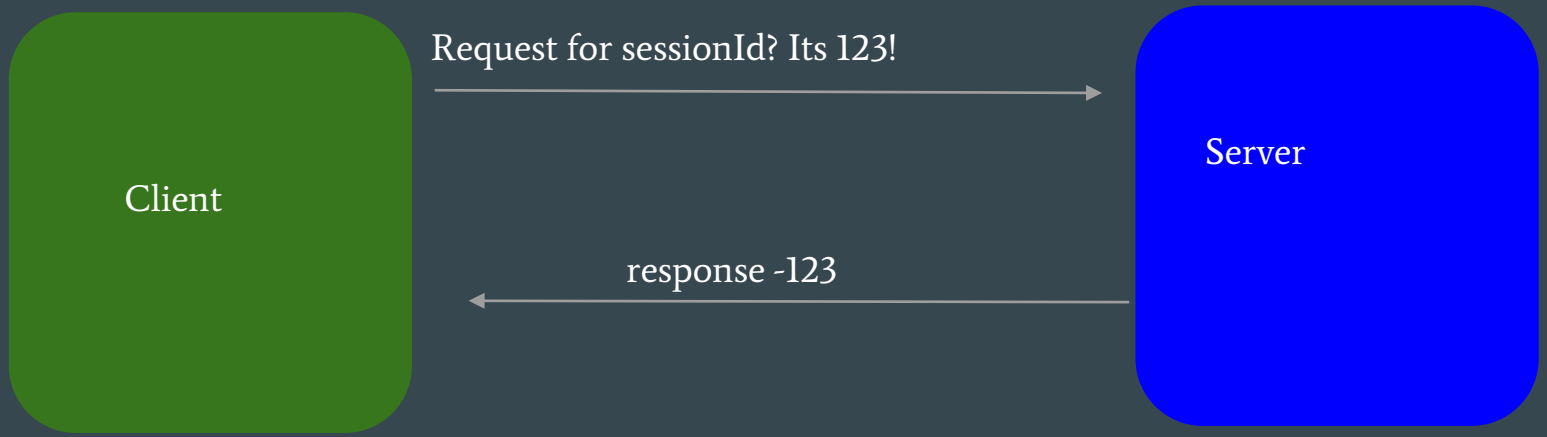
1. Maintain the data using the 4 HttpSession methods()....

```
session.setAttribute("user", username);
```

Server looks for `sessionId`

1. End Session

How it works:



SESSION ID	HttpSession
123	map1

KEY	VALUE

Session Tracking

- To maintain the State despite the Statelessness of HTTP

  (example: think of a shopping cart on a website)

- Tracks User Interaction

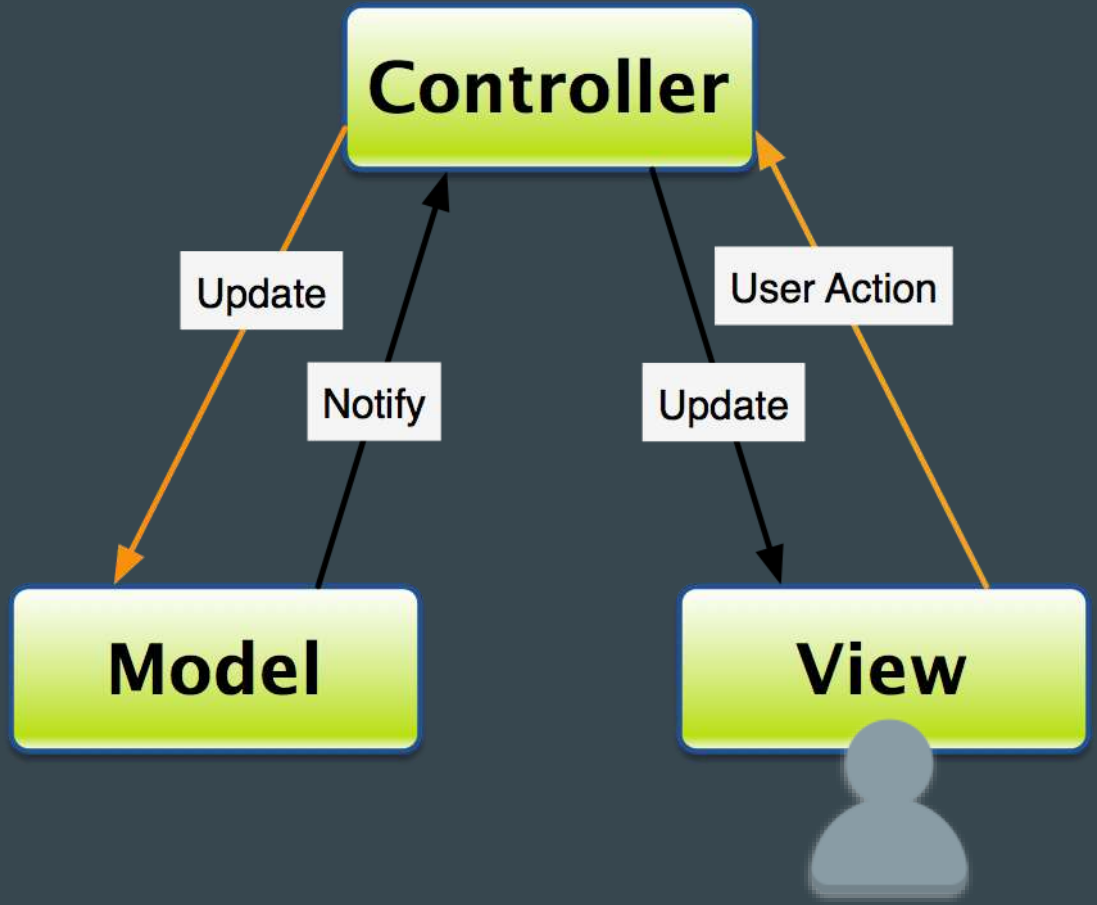
- javax.servlet.HttpSession
- Cookies (client-side files!)

MVC Design Pattern

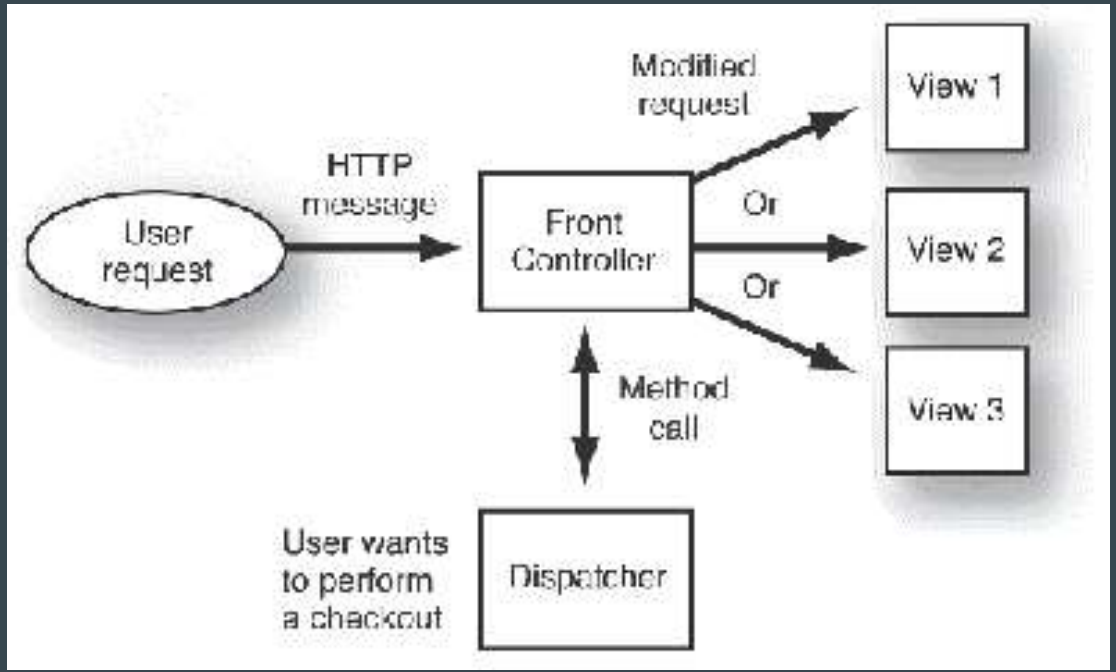
Model - Model used to represent the business layer of the application.

View - The view presents the model's data to the user. The view is a presentation layer that represents the user interface. It displays the data fetched from the model layer by the controller and presents the data to the user whenever requests.

Controller - A controller is an intermediary between the model & a view. It receives the user requests from the view layer and processes them.



Front Controller Design Pattern (it's an MVC)



The front controller design pattern means that all requests that come for a resource in an application will be handled by a single handler and then dispatched to the appropriate handler for that type of request.

The front controller may use other helpers to achieve the dispatching mechanism.

REST != HTTP

REST
REpresentational **S**tate **T**ransfer

...

It is...**an architectural style** for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation

REST API

- Architectural style for designing networked applications
- Relies on **stateless, client/server protocol -- almost always HTTP**
- Treats Server objects as resources that can be created or destroyed
    - Think CRUD!
    - HTTP Verbs

An API is a messenger that abides by a contract. The contract states:

"Give me this information formatted in a particular way, and I will give you a response."

An API is like a "waiter" between client (guest) and server (kitchen)

Each URL is called a **request** while the data sent back to you is called a **response**.

The Anatomy Of A Request

It's important to know that a request is made up of four things:

01 The endpoint

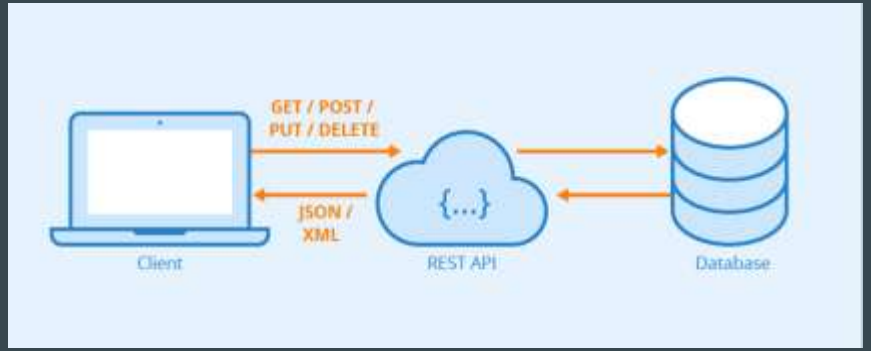
02 The method

03 The headers

04 The data (or body)

The endpoint (or route) is the url you request for. It follows this structure:

```
root-endpoint/?
```



HTTP Methods

Endpoints - The URL/URI where the api/service can be accessed by a client application

- **GET**: Retrieve data from a specified resource
- **POST**: Submit data to be processed to a specified resource
- **PUT**: Update a specified resource
- **DELETE**: Delete a specified resource

https://site.com/api/users OR https://site.com/api/users/1

https://site.com/api/users (add)

https://site.com/api/users/1

https://site.com/api/users/1

Other type of request methods that exist but are rarely used:

- **HEAD**: Same as GET but does not return a body
- **OPTIONS**: Returns the supported HTTP methods
- **PATCH**: Update partial resources

REST Constraints (Principles)...*part 1/1*

x6!

1. **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

REST Constraints (Principles)...*part 2/2*

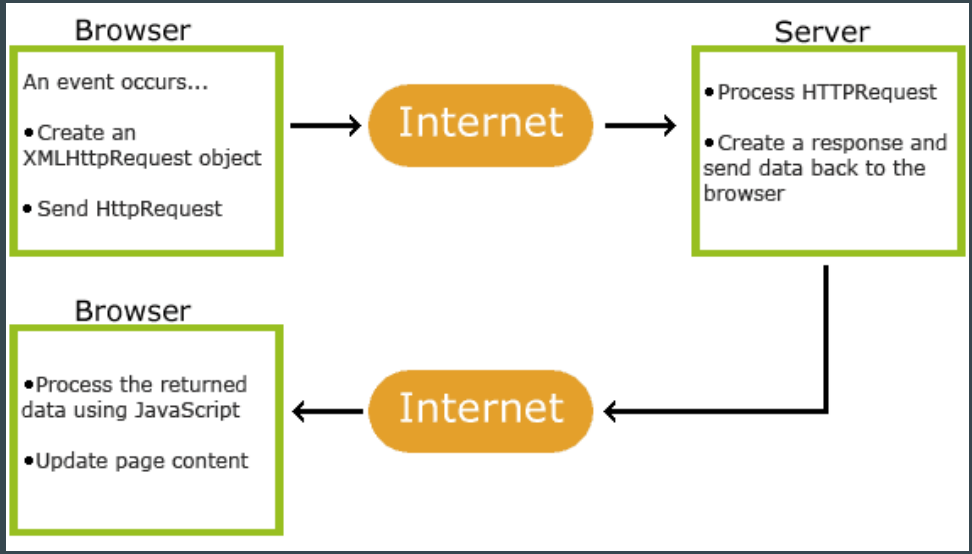
4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.

6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

AJAX - Asynchronous JavaScript And XML

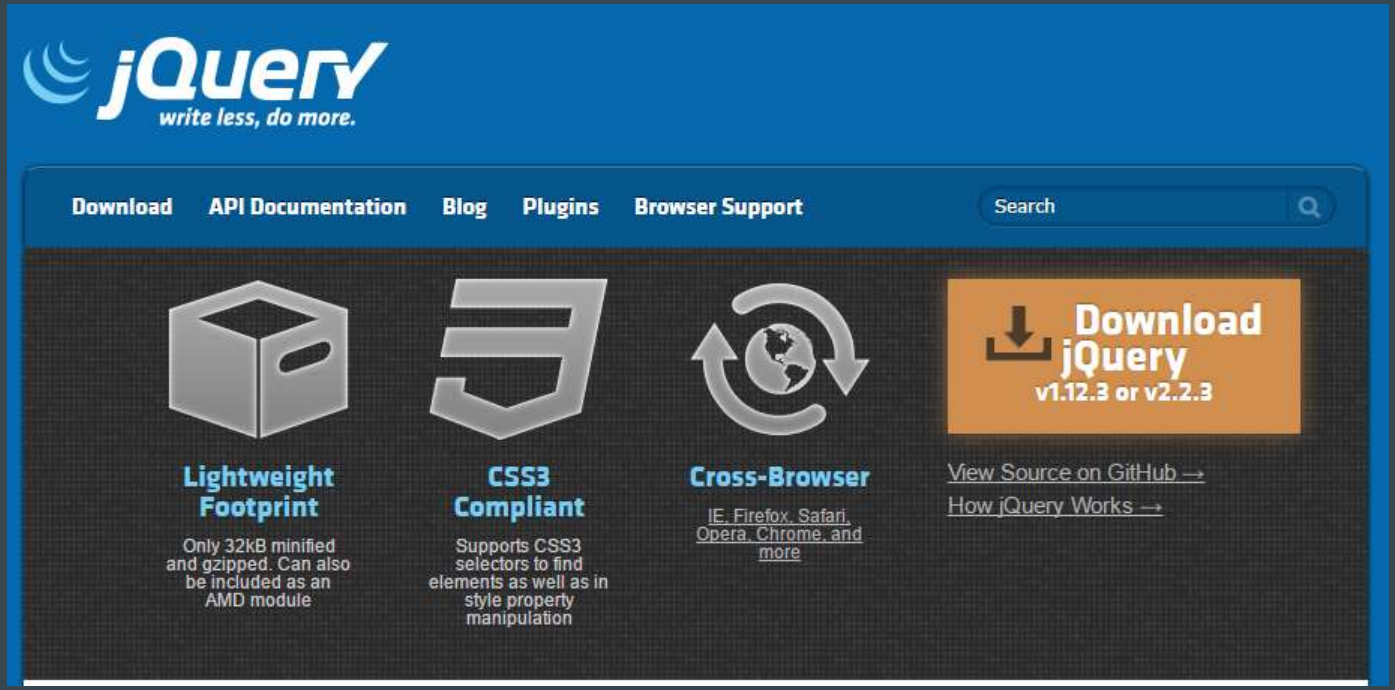
Rather than loading the data through the typical browser HTTP Request initiated via the URL bar...



AJAX requests use JavaScript's **XMLHttpRequest** Object to make requests to the server "behind the scenes"

Intro to jQuery

https://jquery.com/



jQuery is a cross-platform **Javascript library** for easier client-side HTML scripting.

jQuery Helper Library

jQuery can be useful for tasks like:

- Dynamically Inserting, Updating, or Removing HTML
- Registering click or other change events
- Animating HTML elements
- Download data from databases
- And much more!

Working with jQuery generally involves…

1. Including a CDN Link to the jQuery script…

```
<!-- Added link to the jQuery Library -->
<script src="https://code.jquery.com/jquery-2.2.3.js" integrity="sha256-
laXWtGydpwqJ8JA+X9x2miwmaiKhn8tVmOVEigRNtP4=" crossorigin="anonymous"></script>
```

2. Utilizing the jQuery specific ($) selector…

```
$("#clickMe")
```

3. Then applying jQuery methods on the selected elements.

```
$("#clickMe").on("click", function(){

    // Trigger an alert.
    alert("I've been clicked!");
})
```

Docker

...

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called *containers*.

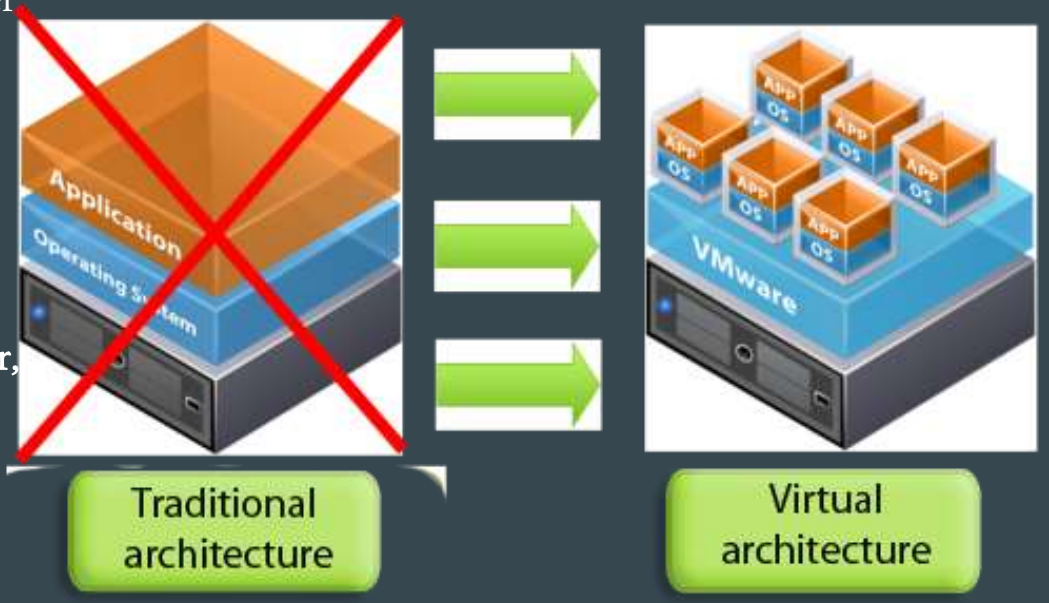
Life Before Virtualization

$$$$$$

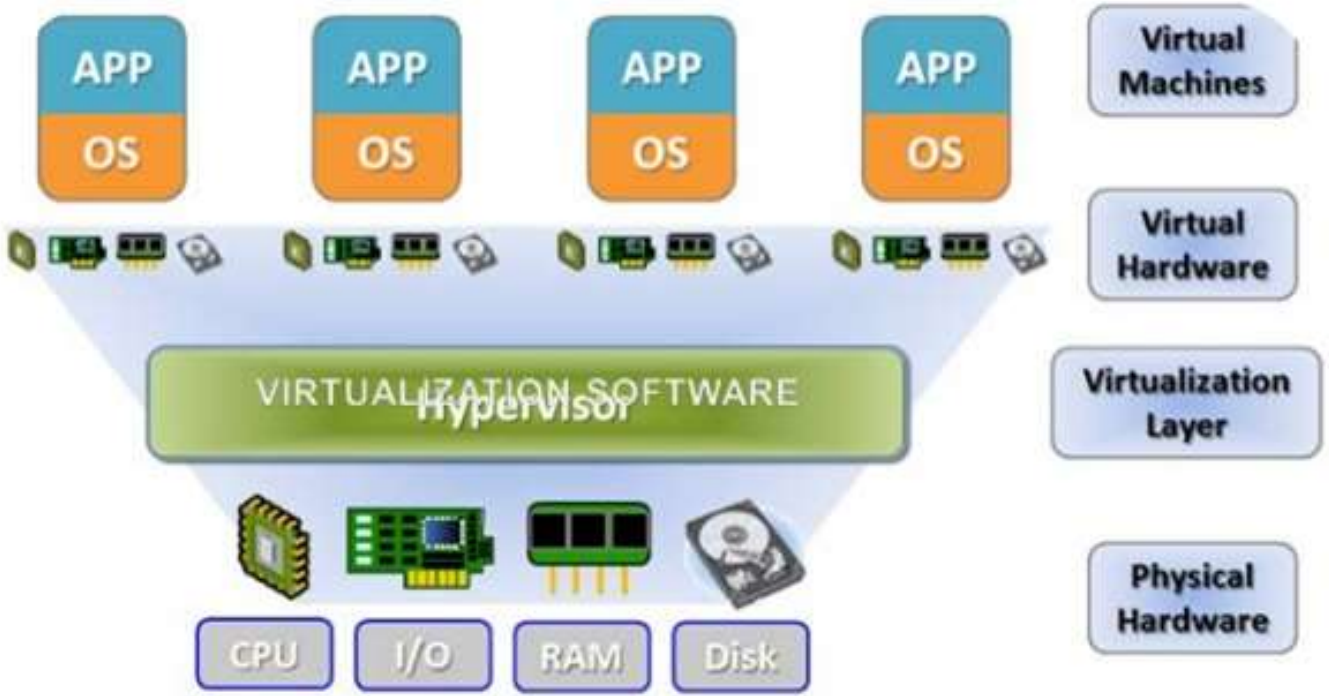
Virtualization

A virtual computer system is known as a “virtual machine” (VM): a tightly isolated software container with an operating system and application inside.

- Each self-contained VM is completely independent.
- Putting multiple VMs on a single computer enables several operating systems and applications **to run on just one physical server, or “host.”**
- A thin layer of software called a **“hypervisor”** decouples the virtual machines from the host and dynamically allocates computing resources to each virtual machine as needed.



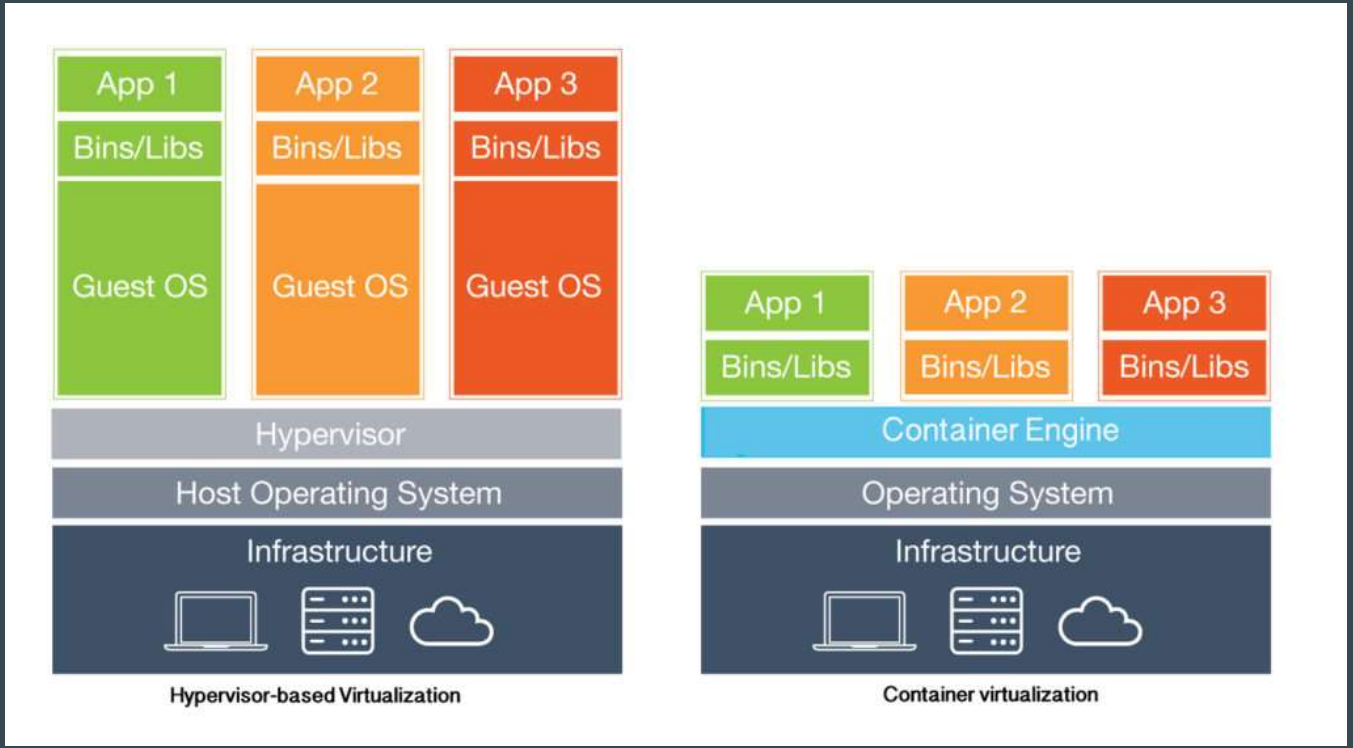
What is Virtualization?



Containerization (...it's better!)

With containers, instead of virtualizing the underlying computer like a virtual machine (VM), just the OS is virtualized.

In contrast to VMs, all that a container requires is enough of an operating system, supporting programs and libraries, and system resources to run a specific program.

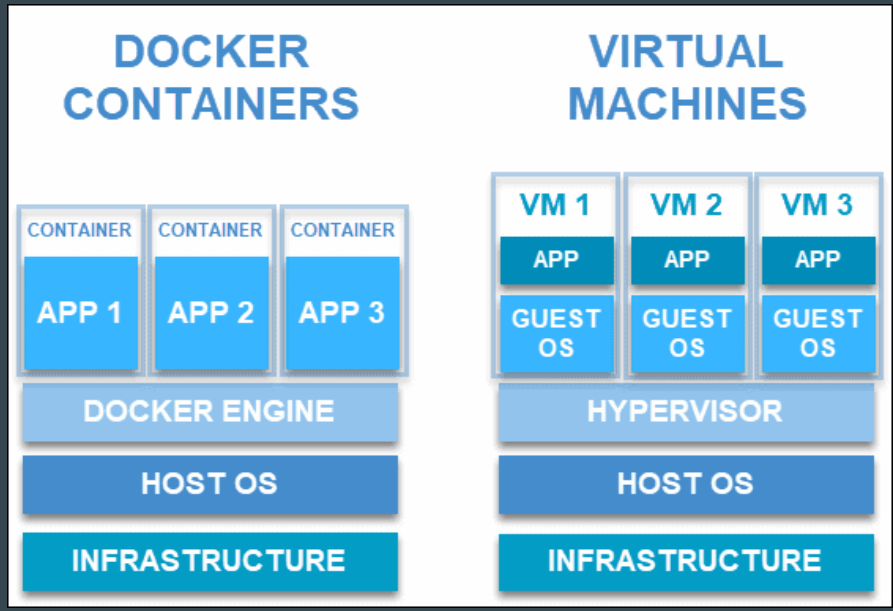


Docker Containers

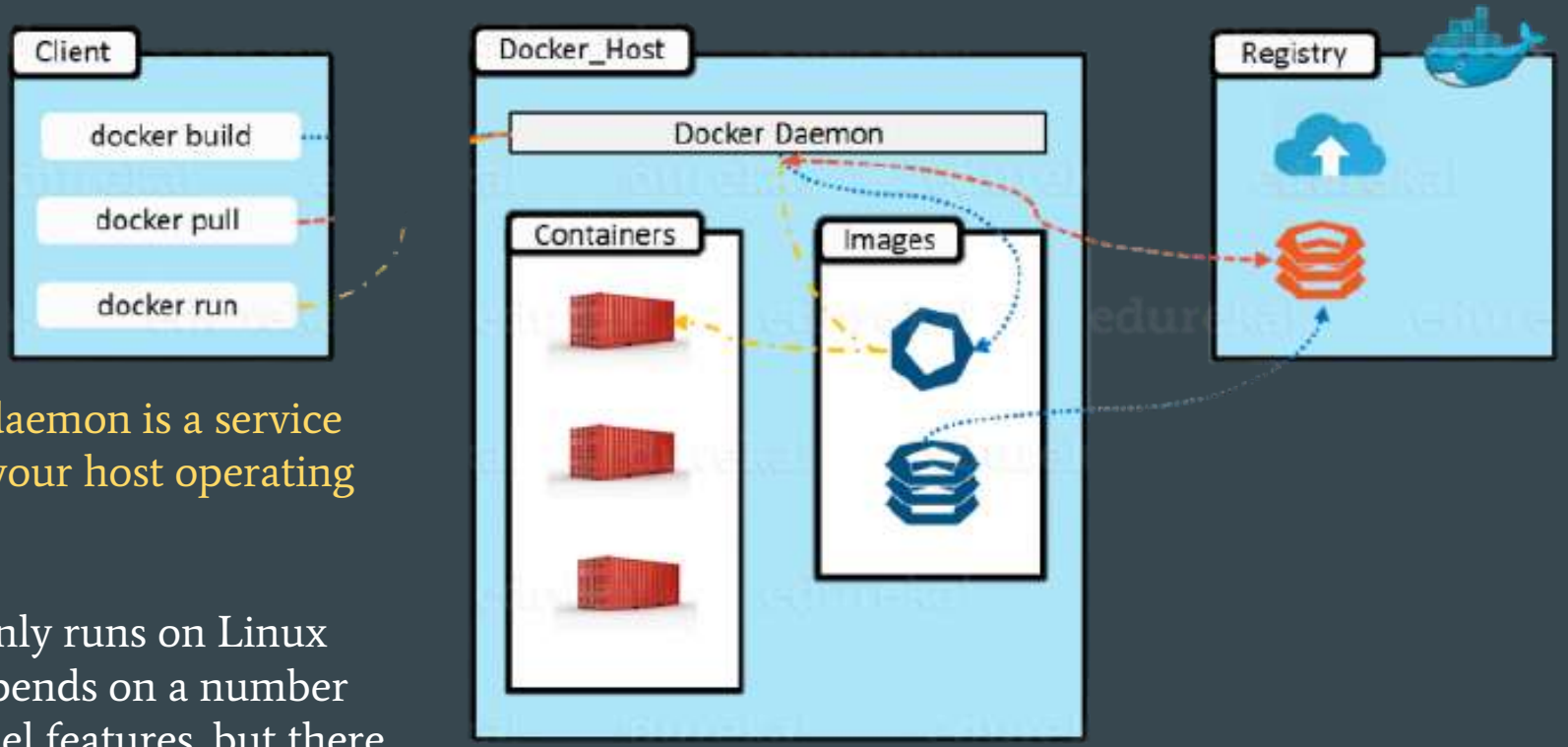
A Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment.

It could be an Ubuntu container, CentOs container, etc. to fulfill the requirement from an operating system point of view.

- Also, it could be an application oriented container like CakePHP container or a Tomcat-Ubuntu container etc.



Docker Daemon



The Docker daemon is a service that runs on your host operating system.

It currently only runs on Linux because it depends on a number of Linux kernel features, but there are a few ways to run Docker on MacOS and Windows too.

Docker Daemon cont'd...

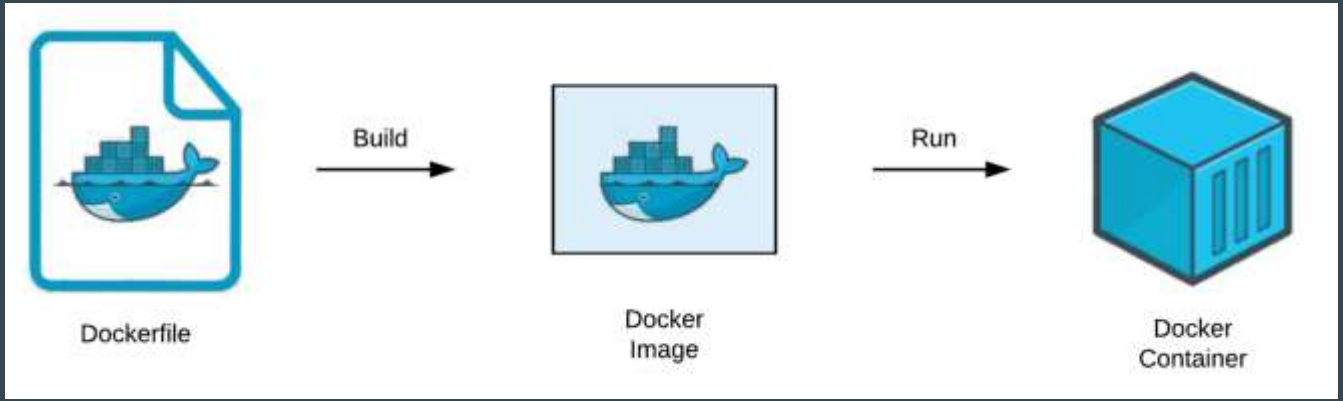
The Docker daemon itself exposes a REST API. From here, a number of different tools can talk to the daemon through this API.

The most widespread tool is the Docker CLI. It is a command line tool that lets you talk to the Docker daemon. When you install Docker, you get both the Docker daemon and the Docker CLI tools together.

If you don't want to use a system utility to manage the Docker daemon, or just want to test things out, you can manually run it using the **dockerd** command.

You may need to use sudo, depending on your operating system configuration.

Dockerfile → Image → Container



A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.

A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform.

An image is a blueprint for building container (the instructions for your container)

A container is a running instance of your image

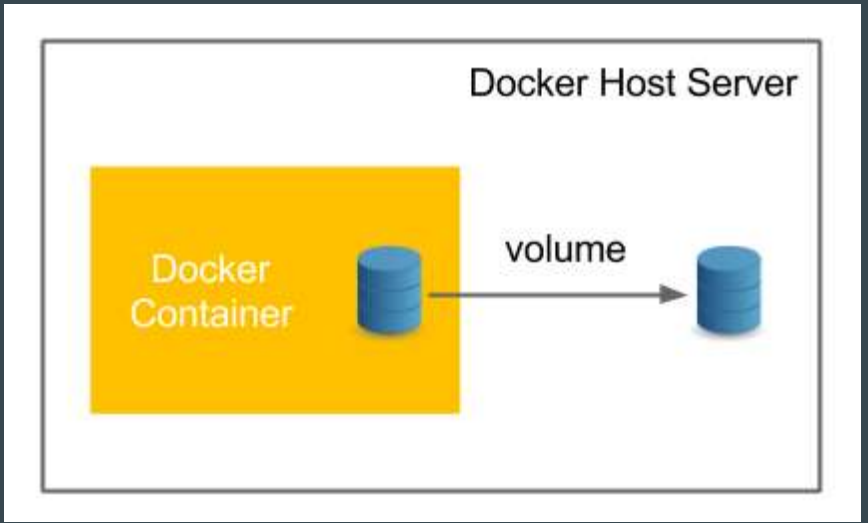
Docker **Volumes**

Volumes: Created and managed by Docker.

You can create a volume explicitly using the **docker volume create** command

When you create a volume, it is stored within a directory on the Docker host.

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.



Building an Image

```
# Define the parent image
FROM ubuntu

# Install needed programs
RUN apt-get update
RUN apt-get -y dist-upgrade
RUN apt-get -y install default-jdk

# Within the image and thus container, set the working directory to the new directory
example
WORKDIR /example

# Create Hello World Java program and save it in the appropriate file
RUN echo 'public class HiWorld{ public static void main(String[] args)
{System.out.println("Hi world");}}'> HiWorld.java

# Compile the Java program, creating the file that the JVM can actually run
RUN javac HiWorld.java

# Run the HelloWorld program in the container
CMD ["java", "HiWorld"]
```

Introducing ORM

...

Reviewing JDBC

- Java Database Connectivity (JDBC):
  - Is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
  - Provides a complete set of interfaces that allows for portable access to an underlying database.
  - Provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.
  - Library includes APIs for the following tasks that are commonly associated with database usage:
    - Making a connection to a database.
    - Creating SQL or MySQL statements.
    - Executing SQL or MySQL queries in the database.
    - Viewing & Modifying the resulting records.

Reviewing JDBC(Contd.)

Identifying the Need of Hibernate

While using traditional methods to create Web applications, you need to:	To store data in a relational database, you need to:	To retrieve data from a relational database, you need to:
• Implement the business logic in the application for storing and retrieving data from the database.	• Convert the various properties of a Java class into primitive data types of the relational database. • Map the class properties with the columns of a table in the database.	• Convert the database values back into the data types of the class properties.

Identifying the Need of Hibernate(Contd.)

- You need to write additional code for handling conversion of values between the data types of database and class properties.
- This increases the application-development time and makes the application error prone.
- In addition, the following problems may occur in the application:
  - Scattered business logic and data manipulation code.
  - Difficulty in fixing compile-time or run-time errors.
  - Difficulty in managing the database connections.

Introducing ORM

- Most of the object-oriented applications use relational databases to store and manage the application data.
- The relational databases represent the data in tables whereas the data in object-oriented applications is encapsulated in a class.
- You can access a class by using its objects. However, to access the tabular data, you need to use a query language.
- As a result, it is not possible to directly store the objects in a relational database.
- These differences between object-oriented and relational database paradigms are called impedance mismatch.

Introducing ORM(Contd.)

- Impedance mismatch exists at the following points:
  - **Granularity**:
    - Refers to the mismatch in the number of classes that are mapped with a certain number of database tables. For example, consider the following code snippet:

```
public class Employee
{
private String name;
private ConactNo contactno;
//Setters and getters
}
public class ConactNo   {
private String MbNo.
private String LandlineNo.
//Setters and getters
}
```

    - The EMPLOYEE table that stores employee information in the database contains the columns as NAME and CONTACTNO.
    - It is evident from the preceding code that a EMPLOYEE table is represented in more than one class.
    - Adding a new data type is easier in an object model than creating a new user-defined column. Therefore, the object model is more granular than the relational model.

Introducing ORM(Contd.)

- **Subtypes (inheritance)**:
  - Refers to the difference in the relationship among the classes in an application and the tables in the database.
  - Classes in the application are commonly related to each other through an inheritance hierarchy. However, the tables in the database cannot be represented through an inheritance hierarchy.
- **Identity**:
  - Refers to how the objects are differentiated in an application and in a relational database.
  - The relational database distinguishes an object instance on the basis of their primary key. However, an object model distinguishes an object from the rest of the objects on the basis of the object identity and object equality.
- **Association**:
  - Refers to the difference in the linking of classes in an application and the linking of tables in the database.
  - In an object model, two classes are linked by association. However, in relational databases, the linking of tables is achieved with the help of foreign keys.

Introducing ORM(Contd.)

- To remove the impedance mismatch problem you need to use process called Object-relational mapping (ORM).
- In computing term ORM is a programming technique for converting data between incompatible type systems in object-oriented programming languages.
- It helps in implementing data persistence by storing the values of class properties in a database table. For example, consider the following code snippet:

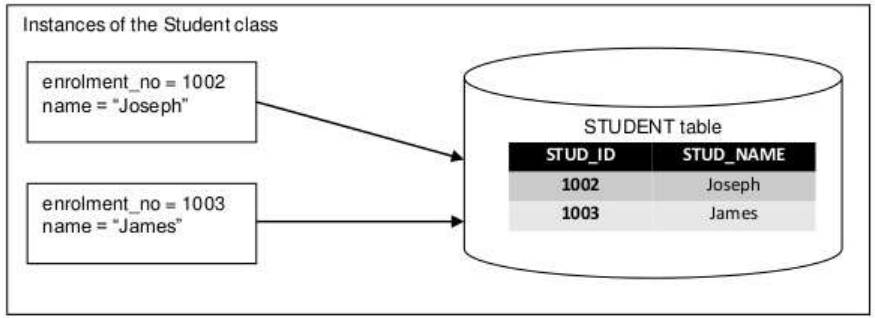
```
CREATE TABLE STUDENT
(
  STUD_ID INT PRIMARY KEY NOT NULL,
  STUD_NAME VARCHAR(20)
)
```

- The `Student` class that is mapped with the `STUDENT` table can be created by using the following code snippet:

```
class Student
{
 int enrolment_no;
 String name;
}
```

Introducing ORM(Contd.)

- The following figure shows the mapping of the `Student` class with the `STUDENT` table.



Introducing ORM(Contd.)

- Java ORM Frameworks:
  - There are several persistent frameworks and ORM options in Java. A persistent framework is an ORM service that stores and retrieves objects into a relational database. Some of the ORM options are:
    - Enterprise JavaBeans Entity Beans
    - Java Data Objects
    - Spring DAO
    - Hibernate
    - Castor
    - TopLink

Introduction to Hibernate

...

Identifying Hibernate

- Hibernate is an ORM solution for JAVA introduced by Gavin King in 2001.
- It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.
- Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.
- Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.

Advantages of Hibernate

- Some of the common advantage of hibernate are:
  - **Open source and Lightweight**: Hibernate framework is open source under the Lesser General Public License (LGPL) license and lightweight.
  - **Fast performance**: The performance of hibernate framework is fast because cache is internally used in hibernate framework.
  - **Database Independent query**: HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries.
  - **Automatic table creation**: Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.
  - **Simplifies complex join**: To fetch data form multiple tables is easy in hibernate framework.
  - **Provides query statistics and database status**: Hibernate supports Query cache and provide statistics about query and database status.

Difference between JDBC and Hibernate

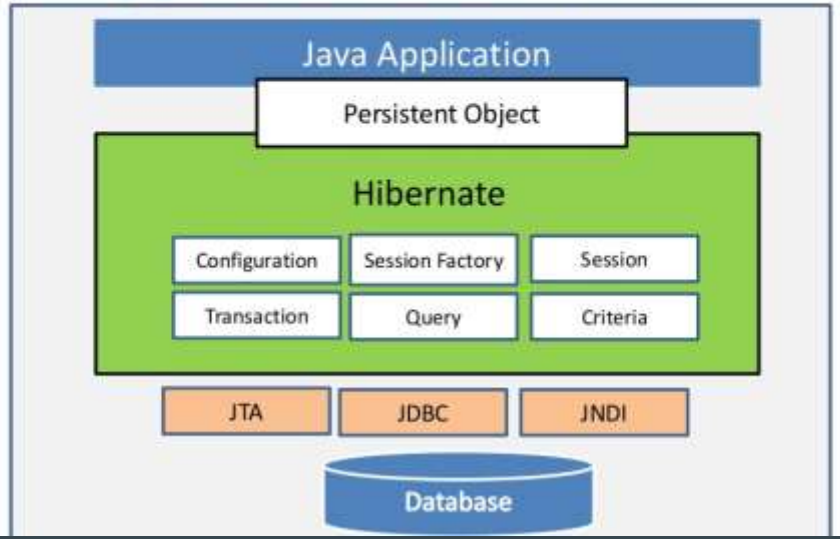
JDBC	Hibernate
Developers needs to write code to map an object model's data representation to a relational data model and its corresponding database schema.	Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files.
JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database.	Hibernate provides a powerful query language Hibernate Query Language that is expressed in a familiar SQL like syntax.
Automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code.	Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS.

Hibernate Architecture

...

Exploring Hibernate Architecture

- Hibernate is a collection of various constituent components that work together to communicate with the database to ensure data integrity and consistency.
- The following figure depicts the architecture of hibernate:



Exploring Hibernate Architecture(Contd.)

- To persist data in the database, the applications communicate with the Hibernate layer that contains the following core classes and interfaces of the Hibernate API:
  - `Configuration` **class**:
    - An instance of the `Configuration` class is used to represent the properties of the configuration file of Hibernate.
    - The instance of this class is created once during the initialization of the Hibernate application.
    - Hibernate uses this instance to read and parse the properties required to connect to a database and application.
    - An Instance of this class is used to create a `SessionFactory` instance. Once the `SessionFactory` instance is created in the application, the `Configuration` object is discarded.
    - The `Configuration` object provides two keys components:
      - **Database Connection**: This is handled through one or more configuration files supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.
      - **Class Mapping Setup**: This component creates the connection between the Java classes and database tables..

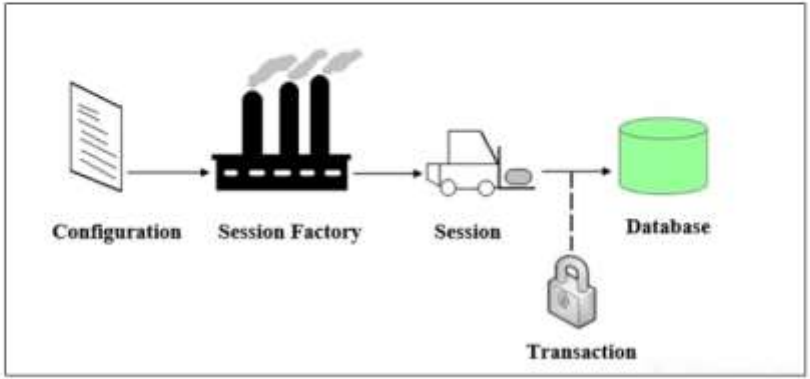
Exploring Hibernate Architecture(Contd.)

- `SessionFactory` **interface:**
  - An instance of the `SessionFactory` interface is created by using an object of the `Configuration` class.
  - This instance is used to create and open a session to communicate with a database.
  - A `SessionFactory` object is created and configured separately for each database being connected.
- `Session` **interface**:
  - An instance of the `Session` interface is created with the help of the `SessionFactory` object. It is used to communicate with a database.
- `Transaction` **interface**:
  - An object of the `Transaction` interface is created by using a `Session` object and used to perform a logical unit of work from the database.
  - The Hibernate framework uses the transaction implementation of various available APIs, such as JDBC, JTA, and JNDI.

Exploring Hibernate Architecture(Contd.)

- `Query` interface: An object of the Query interface is used to create a query that retrieves or stores data into the database.
- `Criteria` interface: An object of the Criteria interface is used to create a query that retrieves or stores data based on multiple conditions.

- The following figure shows the working of the Hibernate classes and interfaces.



Configuring Hibernate

...

Configuring Hibernate

- To connect with a database in Hibernate application, you need to set various properties regarding driver class, user name, and password in the hibernate.cfg.xml file
- The structure of the hibernate.cfg.xml file is given in the following code snippet:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">
<hibernate-configuration>
    <session-factory>
    <property name="connection.username">SYSTEM</property>
              .......
              .......
              .......
        <mapping resource="Mapped_File.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

Configuring Hibernate(Contd.)

- In the preceding code snippet, the hibernate.cfg.xml file contains the following tags:
  - `<?xml>`: Defines the version and encoding type used for the XML document.
  - `<DOCTYPE>`: Specifies the Document Type Definition (DTD) for the XML elements. DTD specifies the grammar rule for the XML document. For example, DTD for hibernate.cfg.xml file is specified as http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd.
  - `<hibernate-configuration>`: Specifies all the configuration details that the application uses to communicate with the underlying database.
  - `<session-factory>`: Contains the database and application specific properties that a Session object uses to establish a communication link between the application and the database.
  - `<property>`: Defines the various properties that are required to connect with the database.
  - `<mapping>`: Specifies the name of the Hibernate mapping file that defines the mapping of a database table to a class.

Configuring Hibernate(Contd.)

- The following properties are commonly set under the `<property>` tag:
  - **JDBC properties**: The JDBC properties are used to connect with a relational database. The following JDBC properties are commonly used in a Hibernate configuration file:
    - `hibernate.connection.driver_class`: Specifies the database specific driver name.
    - `hibernate.connection.url`: Specifies the complete path along with the port number and name of a database that needs to be connected with the application.
    - `hibernate.connection.username`: Specifies the user name that is used to connect with a particular database.
    - `hibernate.connection.password`: Specifies the password that is used to connect with a particular database.

Configuring Hibernate(Contd.)

- **Hibernate configuration properties**: The Hibernate configuration properties control the behavior of Hibernate at runtime. The following configurations are commonly used in the Hibernate configuration file:
  - `hibernate.dialect`: Specifies the database that is used to communicate with the application. It accepts a class name corresponding to the database used in the application. For example, to set the dialect property for the Oracle database, you need to use the following code snippet:

```
<property name="dialect">
org.hibernate.dialect.OracleDialect </property>
```

  - `hibernate.show_sql`: Specifies that the SQL statements are written on the console during the execution of the application. It helps in identifying errors during execution and improving the query performance. The following code snippet is used to set the hibernate.show_sql property:

```
<property name = "hibernate.show_sql">
true</property>
```

Configuring Hibernate(Contd.)

- The following code illustrates how to configure Hibernate with MySQL database:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
 <property name="hibernate.dialect"> org.hibernate.dialect.MySQLDialect </property>
 <property name="hibernate.connection.driver_class"> com.mysql.jdbc.Driver </property>
 <!-- Assume test is the database name -->
 <property name="hibernate.connection.url"> jdbc:mysql://localhost/test </property>
 <property name="hibernate.connection.username"> root </property>
 <property name="hibernate.connection.password"> root123 </property>
 <!-- List of XML mapping files -->
 <mapping resource="Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Creating Hibernate Session

- In a Hibernate application, a session acts as a pipeline between the application and the database.
- To store the application data in the database, you need to create a `Session` object.
- A session object in an application is created by using a `SessionFactory` object.
- To create a `SessionFactory` object, the application needs various database specific configuration settings.
- Hibernate specifies the configuration settings and information about the mapping document in an XML file named hibernate.cfg.xml.
- Hibernate provides the `org.hibernate.cfg.Configuration` class that allows the application to specify the configuration properties and mapping documents to create a `SessionFactory` object.

Creating Hibernate Session(Contd.)

- The `Configuration` class provides the following methods to specify the configuration properties and mapping documents to create a `SessionFactory` object.
  - `configure()`: Loads the hibernate.cfg.xml file and initializes the object of the `Configuration` class with the mappings and configuration properties specified in this file.
  - `getProperties()`: Is used to fetch the properties for the configuration that are configured using the `configure()` method.
  - `applySettings()`: Uses the `ServiceRegistryBuilder` object to apply the settings fetched using the `getProperties()` method. It takes the output of the `getProperties()` method as its parameter.
  - `buildSessionFactory()`: Uses the `Configuration` object returned by the `configure()` method to instantiate a new `SessionFactory` object. It takes `ServiceRegistry` object as its parameter. This object carries the configuration settings that are to be applied.

Creating Hibernate Session(Contd.)

- The following code snippet illustrates how to create object of the `SessionFactory` interface:

```
private static final SessionFactory sessionFactory;
Configuration configuration = new Configuration().configure();
ServiceRegistryBuilder registry = new ServiceRegistryBuilder();
registry.applySettings(configuration.getProperties());
ServiceRegistry serviceRegistry = registry.buildServiceRegistry();
sessionFactory=configuration.buildSessionFactory(serviceRegistry);
```

- You can use the `getSessionFactory()` method to access the Hibernate session in your application, as shown in the following code snippet:


```
Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
```

Creating Hibernate Session(Contd.)

- The main function of the `Session` object is to offer create, read, and delete operations for instances of mapped entity classes. Instances may exist in one of the following three states at a given point in time:
  - **transient**: A new instance of a persistent class which is not associated with a `Session` and has no representation in the database and no identifier value is considered transient by Hibernate.
  - **persistent**: You can make a transient instance persistent by associating it with a `Session`. A persistent instance has a representation in the database, an identifier value and is associated with a `Session`.
  - **detached**: Once we close the Hibernate `Session`, the persistent instance will become a detached instance.

Creating Hibernate Session(Contd.)

- The following table contains some of the common methods available in the `Session` object.

Method	*Description*
`Transaction beginTransaction()`	Begin a unit of work and return the associated Transaction object.
`void cancelQuery()`	Cancel the execution of the current query.
`void clear()`	Completely clear the session.
`Connection close()`	End the session by releasing the JDBC connection and cleaning up.
`Serializable getIdentifier(Object object)`	Return the identifier value of the given entity as associated with this session.
`Query createFilter(Object collection, String queryString)`	Create a new instance of Query for the given collection and filter string.
`Query createQuery(String queryString)`	Create a new instance of Query for the given HQL query string.
`SQLQuery createSQLQuery(String queryString)`	Create a new instance of SQLQuery for the given SQL query string.
`void delete(Object object)`	Remove a persistent instance from the datastore.

Creating Hibernate Session(Contd.)

Method	Description
`void delete(String entityName, Object object)`	Remove a persistent instance from the datastore.
`Session get(String entityName, Serializable id)`	Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.
`SessionFactory getSessionFactory()`	Get the session factory which created this session.
`Transaction getTransaction()`	Get the Transaction instance associated with this session.
`boolean isConnected()`	Check if the session is currently connected.
`boolean isOpen()`	Check if the session is still open.
`Serializable save(Object object)`	Persist the given transient instance, first assigning a generated identifier.
`void saveOrUpdate(Object object)`	Either save(Object) or update(Object) the given instance.
`void update(Object object)`	Update the persistent instance with the identifier of the given detached instance.
`void update(String entityName, Object object)`	Update the persistent instance with the identifier of the given detached instance.

Creating Hibernate Session(Contd.)

- A `Session` instance is serializable if its persistent classes are serializable. A typical transaction should use the following idiom:

```
Session session = factory.openSession();
Transaction tx = null;
try {
    tx = session.beginTransaction();
    // do some work
    ...
    tx.commit();
}
catch (Exception e) {
    if (tx!=null) tx.rollback();
    e.printStackTrace();
}finally {
    session.close();
}
```

Configuring Mapping Properties

- Hibernate allows you to map the classes and their properties with the tables of the database in a configuration file.
- The name of this mapping file has the following syntax: `<persistent_class_name>.hbm.xml`.
- The Hibernate mapping file overcomes the problem of difference between the data types used in classes and columns of the tables.
- The Hibernate mapping file contains mapping information of the class data types with the database specific data types.
- The Hibernate mapping file acts as a medium of communication between the persistent classes and the database tables.
- The Hibernate mapping file enables the communication by mapping the data types used in the persistent classes with the database specific data types by using the Hibernate types.

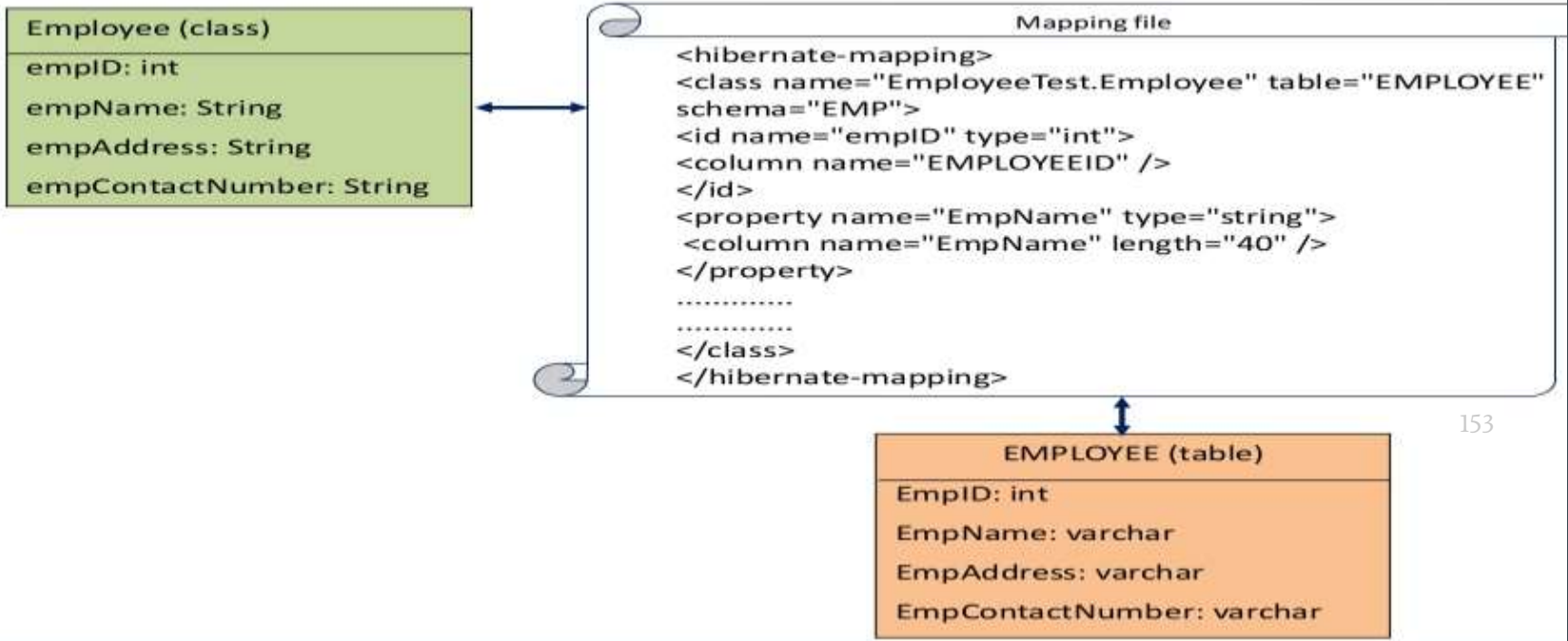
Configuring Mapping Properties(Contd.)

- Hibernate Types:
  - Provide an abstract representation of the underlying database types.
  - Allow you to develop the application without worrying about the target database and the data types supported by it.
  - Provide you the flexibility to change the database without changing the application code.
  - The following table lists some of the Hibernate built-in types along with the corresponding Java and SQL data types.

Hibernate Type	Java Type	SQL Type
integer, long, short	Integer, int, long short	NUMERIC, NUMBER, INT, or other vendor specific type.
character	char	CHAR
big_decimal	java.math.BigDecimal	NUMERIC, NUMBER
float, double	float, double	FLOAT, DOUBLE
boolean, yes_no, true_false	java.lang.Boolean, boolean	BOOLEAN, INT
string	java.lang.String	VARCHAR, VARCHAR2
date, time, timestamp	java.util.Date	DATE, TIME, and TIMESTAMP

Configuring Mapping Properties(Contd.)

- The following figure shows the communication of a persistent class with a database table through the Hibernate mapping file.



Configuring Mapping Properties(Contd.)

- While mapping the persistent class objects with the tables in the database, Hibernate classifies objects in the following types:
  - **Entity type**: Is an independent entity and has its own primary key in a database table.
  - **Value type**: A value type object does not have an identifier. Therefore, a value type object depends on an entity type object for its existence.

Mapping Beans with the Database

- Hibernate provides you with the following mapping techniques to create and map the persistent classes of the application with the database tables to retrieve and store data:
  - Using the Hibernate mapping file
  - Using annotations

Using the Hibernate Mapping File

- In this technique, you need to set values for the various mapping elements in the Hibernate mapping file.
- The Hibernate mapping file contains the following commonly used elements:
  - `<DOCTYPE>`: Specifies the type of the current XML document and the name of the Document Type Definition (DTD).
  - `<hibernate-mapping>`: Refers to the root mapping element that contains other mapping elements, such as <class>, <id>, and <generator>.
  - `<class>`: Refers to the Java class that is mapped with a database table and it contains the following commonly used attributes:
    - `name`: Specifies the fully qualified class name of the persistent class.
    - `table`: Specifies the name of the database table to which the Java class is mapped.
    - `Mutable`: Specifies whether the Java class is mutable. A class whose objects are declared as read-only objects is referred to as the mutable class.
    - `schema`: Specifies the schema of the database being mapped with the Java class.

Using the Hibernate Mapping File(Contd.)

- <id>: Refers to the primary key column of a database table. The commonly used attributes of the <id> element are:
  - name: Specifies the property name of the Java class that is mapped with the primary key column of the database table.
  - type: Specifies the Hibernate type that is used to map a property of the Java class with the primary key column of a database table.
  - column: Specifies the name of the primary key column of the database table.
- <generator>: Refers to a Java class used to generate unique identifiers for the instances of the persistent class in a database table. This element defines the class name with the help of the class attribute. The commonly used aliases as the value of the class attribute are:
  - assigned:
  - Increment
  - Identity
  - Sequence
  - native

Using the Hibernate Mapping File(Contd.)

- `<property>`: Is used to map the properties of the Java class with the specific columns of a database table. This element has the following commonly used attributes:
  - `name`: Specifies the name of a property of the Java class that is being mapped with a column of the database table.
  - `type`: Specifies the Hibernate type that is used to map a property of the Java class with a column of the database table.
  - `column`: Specifies the name of a column of the database table that is mapped to a property of the Java class.
  - `length`: Specifies the maximum number of characters that a column, which is being mapped with a property of the Java class, can store.
- `<column>`: Is used to specify a column that is to be mapped with a property of the Java class. It is used within the `<property>` tag. However, this tag can be replaced by using the column attribute of the `<property>` tag. The `<column>` tag has the following commonly used attributes:
  - `name`: Specifies the name of the column in the database table.
  - `length`: Specifies the maximum number of characters that a column can store.

Using the Hibernate Mapping File(Contd.)

- The following code snippet map the `Employee` class with the `EMPLOYEE` table in a database.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE" schema="EMP">
      <id name="empid" type="int">
         <column name="EMPID" />
         <generator class="assigned" />
      </id>
      <property name="empname" type="string">
         <column name="EMPNAME" length="20" />
      </property>
      <property name="empaddress" type="string">
         <column name="EMPADDRESS" length="40" />
      </property>
   </class>
</hibernate-mapping>
```

Using Annotations

- In this technique, you map the elements of the table in the database with the bean using annotations. The following annotations are commonly used while mapping:
  - `@Entity`: Is used to specify a class as an entity bean. It is contained inside the `javax.persistence` package.
  - `@Table`: Is used to specify the name of the table.
  - `@Column`: Is used to specify the column name. It contains the following attributes:
    - `name`: Specifies the name of the column.
    - `length`: Specifies the length of the column.
    - `nullable`: Specifies the column to be NOT NULL.
    - `unique`: Specifies the column to contain unique values.
  - `@Id`: Is used to define the column as an identifier or a primary key.
  - `@GeneratedValue`: This annotation is used to specify the identifier generation strategy. The identifier generation strategies are of the following types:
    - `AUTO`
    - `IDENTITY`
    - `SEQUENCE`
    - `TABLE`

Using Annotations(Contd.)

- Consider a scenario where you are developing a Web page that accepts the employee id of an employee from the user and displays the details about that employee. The employee details need to be fetched from the EMPLOYEE table stored in the database.
- The Employee table displays the details in the database, as shown in the following figure.

EmpID	EmpName	EmpAddress	EmpContact
238	JAMES JONES	California	7728374859
239	JONAH DOMES	Boston	7528384559
240	RICHARD PARKER	New York	7238495867

Using Annotations(Contd.)

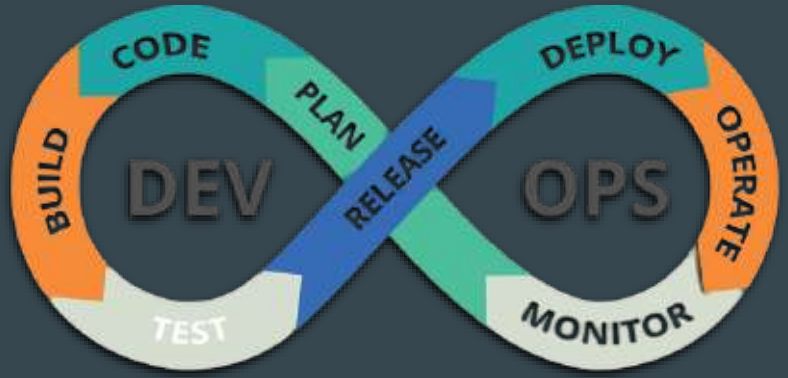
- The following code snippet displays the EMPLOYEE table mapped with the managed bean.

```
import java.io.Serializable;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.persistence.*;
@Named("employee")
@RequestScoped
@Entity
@Table(name="Employee")
public class Employee implements Serializable
{
@Id @GeneratedValue
@Column(name="EmpID")
private int empID;
@Column(name="EmpName")
private String empName;
@Column(name="EmpAddress")
private String empAddress;
@Column(name="EmpContact")
private long empContact;
.....
.....
public Employee() {
    }
}
```

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity

Thus...evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes.

This speed enables organizations to better serve their customers and compete more effectively in the market.



DevOps SDLC - Software Development LifeCycle

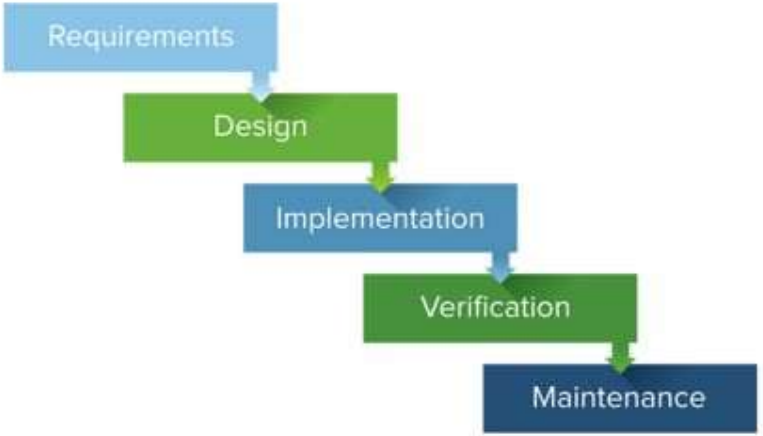
A process for creating testing, and displaying an information system.

1. Requirement Phase : existing system is evaluated to determine flaws (done by Business Analysts
2. Analytics Phase : new system requirements are defined and deficiencies are addressed with proposal for improvement (by B.A's and Senior Members of the team)
3. Design Phase : The proposed system is designed along with product features (junior & senior devs do this) ...still no coding involved
4. Development Phase : software is built and code is written (senior and junior devs)
5. Testing Phase : software is tested to ensure that it functions as expected
6. Deployment & Maintenance Phase : product is delivered to customers and maintenence is kept up (operations team + devs)

Agile

- Continuous cycles
- Small, high-functioning, collaborative teams
- Multiple methodologies
- Flexible/continuous evolution
- Customer involvement

Waterfall



- Sequential/linear stages
- Upfront planning and in-depth documentation
- Contract negotiation
- Best for simple, unchanging projects
- Close project manager involvement

Continuous Integration (CI)

This is the practice of merging code **frequently** into a repository. This should happen several times a day. → It's meant to prevent large errors from accumulating --> allows for immediate correction of errors while they're still small

Continuous Delivery (CD)

Automating the testing of your code so that the release process is available at the push of a button.

Continuous Deployment (CD)

Every change that passes all stages of the product pipeline is released to the customer immediately. There is no "release day".... This is valuable because it speeds up the process of customer feedback

Agile Scrum Framework at a Glance