1. 21 Monday
   1. Command line basics:
      1. cd – change directory
      2. ls – list files
      3. mv – move
      4. cp – copy files
      5. echo – output the strings being passed as arguments
      6. cat – display a file, create file on terminal editor
   2. Java Virtual Machine – JVM -

a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode

* 1. Java Runtime Environment – JRE –

The java Runtime environment is a freely available software distribution containing a standalone JVM, the Java Standard Library, a config tool, and a browser plugin.

* 1. Java Development Kit – JDK –

An implementation of either one of the Java Standard, Enterprise, or Micro editions.

* 1. Class-

An extensible program-code-template for creating objects, providing initial values for states and behavior

* 1. Object-

The particular instances of a class, where the object can be a combination of variables, functions, and data structures

* 1. Data types –
     1. Primitives
        1. Byte – 1 byte
        2. Short – 2 bytes
        3. Int – 4 bytes
        4. Long – 8 bytes
        5. Float – 4 bytes
        6. Double – 8 bytes
        7. Boolean – 1 bit
        8. Char – 2 bytes
     2. Nonprimitives
        1. String
        2. Array
        3. Class
        4. Interface
  2. Casting
     1. Implicit – (automatic)

Byte🡪short🡪char🡪int🡪long🡪float🡪double

* + 1. Explicit – (manual)

Double🡪float🡪long🡪int🡪char🡪byte

e.g. –

double myDouble = 9.78;

int myInt = (int) myDouble; // manual casting: double 🡪 int

1. 22 Tuesday
   1. Constructors -

ctor – a special type of subroutine called to create an object

You can apply public, private, protected to a constructor, *but not* static, final, synchronized, native, or abstract

Constructors cannot return void

* 1. Var-args –

Variable-length arguments – a method that takes a variable number of arguments

* 1. Methods –

Procedure or behaviors associated with a message and an object

Parameter –

Special kind of variable used in a subroutine to refer to one of the pieces of data provided as input to the subroutine

* 1. Array –

An array type is a data type that represents a collection of elements (values or variables), each selected by one or more indices (identifying keys) that can be computed in runtime during program execution

String[] cars = {“Volvo,” “BMW”, “Tesla”, “Nissan};

int [] myNumbers = { { 1, 2, 3, 4}, {5, 6, 7} };

for (int i = 0; i < myNumbers.length; ++i) {

for(int j = 0; j < myNumbers[i].length; ++j) {

System.out.println(myNumbers[i][j]);

Output:

1

2

3

4

5

6

7

* 1. Control flow –

The order in which individual statements, instructions, or function calls of an imperative program are executed or evaluated

* + 1. For loop – for(*initialization, condition, afterthought*)
    2. While loop – while(condition) {block to execute}

**while** (true) {

// do complicated stuff

**if**(someCondition)

**break**;

//more stuff

}

* + 1. Do-while loop – like a while loop but executes at least once
    2. Switch statement – selects one of many code blocks to be executed
       - 1. switch(*expression*) {
         2. case x:
         3. *// code block*
         4. break;
         5. case y:
         6. *// code block*
         7. break;
         8. default:
         9. *// code block*
         10. }
    3. if-else statement –
       1. if – use if to specify a block of code to be executed if a condition is met
       2. else – use else to specify a block of code to be executed if the condition is false
       3. else if – use else if to specify a new condition if the first condition is false
    4. Arithmetic operators

|  |  |  |
| --- | --- | --- |
| + | Addition | Adds together two values |
|  | Subtraction | Subtracts one value from another |
| \* | Multiplication | Multiplies two values |
| / | Division | Divides one value by another |
| % | Modulus | Returns the division remainder |
| ++ | Increment | Increased the value of a variable by 1 |
| -- | Decrement | Decreased the value of a variable by 1 |

Relational operators

|  |  |
| --- | --- |
| == | Equal to |
| != | Not equal |
| > | Greater than |
| < | less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

Short-Circuit operators

|  |  |  |
| --- | --- | --- |
| && | Logical and | Returns true if both statements are true |
| || | Logical or | Returns true if one of the statements is true |
| ! | Logical not | Reverse the result |

* + 1. Operator precedence

|  |  |  |  |
| --- | --- | --- | --- |
| **16** | [] . () | access array element access object member parentheses | left to right |
| **15** | ++ -- | unary post-increment unary post-decrement | not associative |
| **14** | ++ -- + - ! ~ | unary pre-increment unary pre-decrement unary plus unary minus unary logical NOT unary bitwise NOT | right to left |
| **13** | () new | cast object creation | right to left |
| **12** | \* / % | multiplicative | left to right |
| **11** | + - + | additive string concatenation | left to right |
| **10** | << >> >>> | shift | left to right |
| **9** | < <= > >= instanceof | relational | not associative |
| **8** | == != | equality | left to right |
| **7** | & | bitwise AND | left to right |
| **6** | ^ | bitwise XOR | left to right |
| **5** | | | bitwise OR | left to right |
| **4** | && | logical AND | left to right |
| **3** | || | logical OR | left to right |
| **2** | ?: | ternary | right to left |
| **1** | =   +=   -= \*=   /=   %= &=   ^=   |= <<=  >>= >>>= | assignment | right to left |

viii.

Basic Git --

**Configurations:**

$ git config --global.user.name “FirstName LastName”

$ git config --global user.email “your-email@email-provider.com”

$ git config --global color.ui true

$ git config --list

**Starting a repository**

$ git init

$ git status

**Adding –**

$ git add <file-name>

$ git add <file-name> <another file-name><yet-another-file-name>

$ git add

$ git add --all

$ git add -A

$ git rm --cached <file-name>

$ git reset <file-name>

**Committing**

$ git commit -m “Add three files”

$ git reset --soft HEAD^

$ git commit --amend -m <enter message>

**Pulling & Pushing**

$ git remote add origin <link>

$ git push -u origin master

$ git clone <clone>

$ git pull

**Branches** – branches in Git are paths. Base branch is called the master branch.

1. 9/23 Wednesday
   1. Annotations – a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code that they annotate. @Override, placed above an overridden method in the child class, is an example of annotation. Among the uses of annotation are:
      1. **Informing the compiler**
      2. **Compile-time and deployment-time processing**
      3. **Runtime processing**
      4. Packages – a grouping of related types providing access protection and name space management. **Types** here refers to **classes**, **interfaces**, enumerations, and annotation types. Enumerations and annotation types are special kinds of classes and interfaces. The types that comprise a package are known as *package members*
      5. Imports – to import a specific member into a current file, put an *import* statement at the beginning of the file before any type definitions but after the package statement.
   2. Object class –

The root of the class hierarchy. Every class has *Object* as its superclass. All objects, including arrays, implement the methods of the Object class.

Methods: clone(), equals(Object obj), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait()

* 1. Wrapper class –

Provide a way to use primitive data types as objects

|  |  |
| --- | --- |
| **Primitive Data Type** | **Wrapper Class** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |
| char | Character |

* 1. Static keyword –

Static fields are associated with the class, rather than any object. Attributes and methods belong to the class, rather than an object

* 1. Scopes of a variable – the part of the program where the variable is accessible.
     1. Member variable - class-level scope

These variables must be declared inside the class (outside of any function). They can be directly accessed anywhere in class. Member variables can be accessed outside a class following these rules:

**Modifier Package Subclass World**

public Yes Yes Yes

protected Yes Yes No

Default (no

modifier) Yes No No

private No No No

* + 1. Local variable – method-level scope

Variables declared within a method have method-level scope and cannot be access outside the method

* + 1. Loop variable – block-level scope

A variable declared inside a pair of brackets “{“ and “}” in a method has scope within the brackets only

* 1. The final keyword –

The final keyword indicates that the method cannot be overridden by subclasses. A class that is declared final cannot be subclassed. This is particularly useful, for example, when creating an immutable class like the *String* class.

* 1. Single and multiline comments –
     1. // this is a single line comment
     2. /\* this is a

Multiline comment \*/

* 1. String API

The string class represents character strings. All string literals in java programs, such as “abc”, are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable, they can be shared.

* 1. Format specifiers
     1. %n – inserts a newline character
     2. %d – decimal integer
     3. %s – String
  2. StringBuilder

This class provides an API compatible with StringBuffer, designed for use as a drop-in replacement for StringBuffer in places where the buffer was being used by a single thread. **Append** and **inset** are the main methods.

* 1. StringBuffer

Like StringBuilder but thread safe (safe for multithreaded environments)

1. 9/24 Thursday
   1. Exceptions class hierarchy
      1. Error
      2. Checked Exception – all exceptions *checked* at compile time

IO Exception, ClassNotFoundException, FileNotFoundException

* + 1. Unchecked exception – Error and RuntimeException classes

Arithmetic, ClassCast, IllegalArgument,IndexOutofBounds,NullPointer

* 1. Custom exception –

Customized exception which extends the *Exception* class

* 1. Try/catch/finally
     1. Try – the *try* statement allows you to define a block of code to be tested for errors while it is being executed
     2. Catch – the *catch* statement allows you to define a block of code to be executed if an error occurs in the try block
     3. Finally – the *finally* statement lets you execute code, after try…catch, regardless of the results
  2. Throw

The throw statement allows you to create a custom error message

* 1. Throws

Indicates what exception type may be thrown by a method

1. 9/25 Friday
   * 1. Abstraction  
        the proves of hiding certain details and only showing essential information to the user
        1. Abstract class: a restricted class that cannot be used to create objects (can only be inherited from another class)
        2. Abstract method: can only be used in abstract classes, and has no body.
     2. Polymorphism

Polymorphism uses inherited methods to perform different tasks than original programmed (method overriding)

* + 1. Encapsulation

Keeps sensitive information hidden from users. To achieve this you must:

* + - 1. Declare class variables/attributes as private
      2. Provide public get and set methods to access the value of a private variable
    1. Inheritance

The possibility of inheriting attributes and methods from one class to another

* + - 1. Subclass (child) - the class that inherits from another class
      2. Superclass (parent) – the class being inherited from
    1. Method overloading –

Multiple methods in the same class can have the same name with different parameters

* + 1. Method overriding –

How Java achieves polymorphism. A feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one its super-classes or parent classes.

* + 1. Access modifiers
       1. Class-level – public/default
       2. Attribute-, method-, and constructor-level – public/private/default/protected
    2. Non-access modifiers
       1. Class-level – finals/abstract
       2. A,M,C-level – final/static/abstract/transient/synchronized/volatile

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Levels** | | | | |
| **Modifier** | **Class** | **Package** | **Subclass** | **World** |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *default* | Y | Y | N | N |
| private | Y | N | N | N |

* + 1. Abstract classes

A restricted class that cannot be used to create objects (to access it, it must be inherited to another class)

* + 1. Interfaces

An interface is a completely “abstract class” that is used to group related methods with empty bodies

6)

1. RDMBS – software system that enables users to define, create, maintain, and control access to the database / database management system based on the relational model

7)

i) SQL Data types (Mysql):

String – CHAR, VARCHAR, BINARY, VARBINARY, TINYBLOB, TINYTEXT, TEXT, BLOB (Binary Large Object – images, audio, multimedia object), MEDIUMTEXT, MEDIUMBLOB, LONGTEXT, LONGBLOB, ENUM (set of named values called elements, members, enumeral, or enumerators, SET

Numeric – BIT, TINYINT, BOOL, BOOLEAN, SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT, FLOAT, FLOAT, DOUBLE, DOUBLE PRECISION, DECIMAL, DEC

Date and Time – DATE, DATETIME, TIMESTAMP, TIME, YEAR

(sqlServer) Other – sql\_variant, uniqueidentifier, xml, cursor, table

1. DML - data manipulation language –

INSERT – used to insert data into a table

UPDATE – used to update existing data within a table

DELETE – used to delete records from a database table

MERGE – runs insert, update, or delete operations on a target table from the results of a join with a source table

CAL

EXPLAIN PLAN – displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements

LOCK TABLE – locks one or more tables, table partitions, or table subpartitions in a specified mode. The lock manually overrides automatic locking and permits or denies access to a table of view by other users for the duration of your operation

DDL – data definition language –

CREATE – used to create database or objects

ALTER – is used to alter structure of database

DROP – is used to delete objects from the database

RENAME – used to rename object existing in the database

TRUNCATE – used to remove all records from a table (deletes the data but not the table)

COMMENT – used to add comments to the data dictionary

DQL – data query language –

SELECT – used to retrieve data from the database

DCL - Data Control Language –

GRANT – gives user’s access privileges to database

REVOKE – withdraw user’s access privileges given by ‘GRANT’

TCL – transaction control language –

COMMIT – commits a transaction

ROLLBACK – rollbacks a transaction to last COMMIT

SAVEPOINT – sets a SAVEPOINT within a transaction for rollbacks

SET TRANSACTION – specify characteristics for the transaction

Transaction – groups a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any task fails, the transaction fails. Therefore, transaction results are binary: **success** or **failure**.

1. Basic SQL queries – SELECT *column1, column2*… FROM *tableName* WHERE *condition*

8)

i)

ISOLATION LEVELS – controls how transaction integrity is visible to others users and systems. For example, when a user is creative a purchase order and has created the header, but not the purchase order lines, is the header available for other systems or users to see? Lower isolation = greater visibility. Higher isolation = reduced concurrency effects + more resource utilization.

Serializable – highest isolation level

Read uncommitted – lowest isolation level (dirty reads allowed)

Repeatable reads – read- and write- locked (phantoms allowed)

Read committed – read-locked (non-repeatable reads, lost updates, phantoms allowed)

‘+’ – not possible

‘-‘ – possible

![Scatter chart

Description automatically generated]()

READ PHENOMENA –

Dirty reads – occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed

Non-repeatable reads – occurs when a row is retrieved twice and the values within the row differ between reads during the course of a transaction

Phantom reads – occurs when new rows are added or removed by another transaction to the records being read in the course of a transaction

Lost updates – occurs when two processes read the same data and then try to update the data with a different value

ii)

Data Query Language DQL –

WHERE – used to filter records, used to extract only those records that fulfil a specific condition

= , > , < , >= , <=, <> (not equal), BETWEEN, LIKE (search for a pattern), IN (multiple possible values for a column)

HAVING – like where, but can be used with *aggregate functions*

ORDER BY – used to sort the result-set in ascending (default) or descending order. To sort by descending, use the DESC keyword (can use multiple columns)

GROUP BY – groups rows that have the same values into summary rows, like “find the number of customers in each country”

Often used with aggregate functions (ie COUNT, MAX, MIN, SUM, AVG)

iii)

Schema – a collection of database objects linked with a particular database username. One schema per database, each database can have single or multiple schemas. The design of the database – physical (structural), logical (administrators), and view (end-user interaction).

1. Multiplicity –

One-to-one: each entity instance is related to a single instance of another entity

One-to-many: An entity instance can be related to multiple instances of the other entities

Many-to-one: Multiple instances of an entity can be related to a single instance of the other entity

Many-to-many: The entity instances can be related to multiple instances of each other

1. **SQL constraints** are used to specify rules for data in a table. Can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement
2. **NOT NULL –** ensures that a column cannot have a null value
3. **UNIQUE –** ensures that all values in a column are different
4. **PRIMARY KEY –** a combination of NOT NULL and UNIQUE. Uniquely identifies each row in a table
5. **FOREIGN KEY –** uniquely identifies a row/record in another table
6. **CHECK** – ensures that all values in a column satisfies a specific condition
7. **DEFAULT –** sets a default value for a column when no value is specified
8. **INDEX** – used to create and retrieve data from the database very quickly

9)

i) Aliases (AS) – used to give a table or a column in a table a temporary name, often to make column names more readable. Aliases only exist for the duration of the query, used with the AS keyword.

SELECT column\_name AS alias\_name

FROM table\_name;

ii) *Scalar functions – based on user input, these return single values*

1. UCASE()
2. LCASE()
3. MID(column\_name, start, length) extracts texts from the text field
4. LEN() – returns the length
5. ROUND() – rounds to nearest even integer (e.g. 5.5 = 6.5)
6. NOW() – current system date and time
7. FORMAT(column\_name,format) – used to format how a field is to be displayed

iii) *Aggregate functions -*

1. AVG()
2. COUNT()
3. FIRST()
4. LAST()
5. MAX()
6. MIN()
7. SUM()

iv)Intro to JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity. Part of JSE. Provides methods to query and update data in a database and is oriented toward relational databases.

v) JDBC classes & interfaces:

| **Class/interface** | **Description** |
| --- | --- |
| **DriverManager ©** | This **class** manages the JDBC drivers. You need to register your drivers to this. It provides methods such as *registerDriver()* and *getConnection().* |
| **Driver (i)** | This **interface** is the Base interface for every driver class i.e. If you want to create a JDBC Driver of your own, you need to implement this interface. If you load a Driver class (implementation of this interface), it will create an instance of itself and register with the driver manager. |
| **Statement (i)** | This interface represents a static SQL statement. Using the Statement object and its methods, you can execute an SQL statement and get the results of it. It provides methods such as *execute()*, *executeBatch(), executeUpdate()* etc to execute the statements. |
| **PreparedStatement** | This represents a precompiled SQL statement. An SQL statement is compiled and stored in a prepared statement and you can later execute this multiple times. You can get an object of this interface using the method of the Connection interface named *prepareStatement()*. This provides methods such as *executeQuery()*, *executeUpdate()*, and *execute()* to execute the prepared statements and *getXXX(), setXXX()* (where XXX is the datatypes such as **long int float** etc..) methods to set and get the values of the bind variables of the prepared statement. |
| **CallableStatement (i)** | Using an object of this **interface** you can execute the stored procedures. This returns single or multiple results. It will accept input parameters too. You can create a CallableStatement using the *prepareCall*() method of the Connection interface. Just like Prepared statement, this will also provide *setXXX*() and *getXXX*() methods to pass the input parameters and to get the output parameters of the procedures. |
| **Connection (i)** | This **interface** represents the connection with a specific database. SQL statements are executed in the context of a connection. This **interface** provides TCL methods such as *close*(), *commit*(), *rollback*(), *createStatement*(), *prepareCall*(), *prepareStatement*(), *setAutoCommit*() *setSavepoint*() etc. |
| **ResultSet (i)** | This **interface** represents the database result set, a table which is generated by executing statements. This interface provides getter and update methods to retrieve and update its contents, respectively. |
| **ResultSetMetaData (i)** | This **interface** is used to get the information about the result set such as, number of columns, name of the column, data type of the column, schema of the result set, table name, etc. It provides methods such as *getColumnCount*(), *getColumnName*(), *getColumnType*(), *getTableName*(), *getSchemaName*() etc. |

vi) JDBC Connection String – ie *jdbc:postgresql://HOST/DATABASE  
 org.postgresql.Driver*

vii) Design Pattern : Data Access Object (**DAO**)

pattern used to separate low level data accessing API or operations from high level business services.

BusinessObject – represents the data client (*obtains / modifies*)

DataAccessObject (**DAO**) – abstracts the underlying data access implementation for the BusinessObject to enable transparent access to the data source (*creates / uses*)

DataSource – represents a data source implementation (RDBMS, XML repository, *encapsulates*)

TransferObject – used by **DAO** to return data to the client

viii) Parametrization – creating different data sets for different users in the same test script. For example, running multiple users with different credentials in the same script.

10)

i) database joins:

INNER JOIN: Returns records that have matching values in both tables

LEFT JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT JOIN: Returns all records from the right table, and the matched records from the left table

FULL OUTER JOIN: Returns all records when there is a match in either left or right table

ii) set operators

UNION – combines the results of two SQL queries in a single table of all matching rows. The two queries must result in the same number of columns and compatible data types in order to unite

UNION ALL – (does not eliminate duplicates)

INTERSECT – takes the results of two queries and returns only rows that appear in both result sets. Does not distinguish between NULLs. Removes duplicate rows from the final result set

EXCEPT – takes the distinct rows of one query and returns the rows that do not appear in a second result set

Oracle (MINUS)

iii)

Referential integrity – a property of data stating that all its references are valid. In the context of relational databases, it requires that if a value of one attribute (column) of a relation (table) references a value of another attribute, then the referenced value must exist.

Any column in a base table that is declared a foreign key can only contain either null values or values from a parent table’s primary key or a candidate key.

The adjective ‘referential’ describes the action that a foreign key performs, ‘referring’ to a linked column in another table. In simple terms, ‘referential integrity’ guarantees that the target ‘referred’ to will be found.

11)

1. Client-server communication – how client makes request and server receives/processes these requests, apply some logic, send back appropriate response
2. JAVA EE database access, remote method invocation (RMI), messaging (JMS), web services, XML processing

Standard APIs for Enterprise JavaBeans, Servlets, Portlets, and Java Server Pages (JSP)

1. HTTP Protocol & methods

HTTP is CONNECTIONLESS, MEDIA INDEPENDENT, STATELESS

URI – uniform resource identifier

Application layer protocol for distributed, collaborative, hypermedia information systems. The foundation of data communication for the WWW

1. **GET** – requests representation of specified resource / to retrieve data, default method
2. HEAD – same as GET, but without response body / for meta content
3. **POST** – the server accepts the entity enclosed in the request as a new subordinate of the web resource identified by the URI (annotation, message, newsgroup, mailing list, comment thread; process, save, and change data
4. **PUT** – requests that the enclosed entity be stored under the supplied URI; used for updating
5. **DELETE-** used to delete a document of the target resource
6. TRACE – used when the client wants to see if any change has been done by the intermediate server
7. OPTIONS – returns to the HTTP methods which are supported by the server
8. CONNECT
9. PATCH – used for updating

Request Header

Content-type header

Request body

| Security of HTTP methods | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **HTTP method** | **RFC** | **Request has Body** | **Response has Body** | **Safe** | **Idempotent** | **Cacheable** |
| **GET** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Optional | Yes | Yes | Yes | Yes |
| **HEAD** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Optional | No | Yes | Yes | Yes |
| **POST** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Yes | Yes | No | No | Yes |
| **PUT** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Yes | Yes | No | Yes | No |
| **DELETE** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Optional | Yes | No | Yes | No |
| **CONNECT** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Optional | Yes | No | No | No |
| **OPTIONS** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | Optional | Yes | Yes | Yes | No |
| **TRACE** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [7231](https://tools.ietf.org/html/rfc7231) | No | Yes | Yes | Yes | No |
| **PATCH** | [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [5789](https://tools.ietf.org/html/rfc5789) | Yes | Yes | No | No | No |

1. HTTP Status codes

1xx informational response – request received and understood

2xx successful – action requested by the client was received, understood, and accepted

3xx redirection – the client must take additional action to complete the request

4xx client error – error seems to have been caused by the client

5xx server error – server failed to fulfill a request

1. Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser
2. Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human- and machine-readable – applications providing web services communicate through XML
3. HTTPS – Hypertext Transfer Protocol Secure – an extension of the hypertext transfer protocol used for secure communication over a computer network
4. TCP – transmission control protocol – provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network
5. SMTP - Simple mail transfer protocol – communication protocol for electronic mail transmission
6. FTP – file transfer protocol – network protocol used for the transfer of computer files between a client and server on a computer network
7. PoP – point of presence – an artificial demarcation point or interface point between communication entities – internet point of presence, the local access point that allows users to connect to the internet with their internet service provider.
8. BEEP – Blocks Extensible Exchange Protocol – framework for creating network application protocols.

12)

i) Container and lifecycle

**Container** – web container, containerization – the creates an environment to complete web applications on-the-go. A small, server-resident program that typically runs automatically in response to user input

The component of a web server that interacts with Jakarta Servlets. A web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

Creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet-management tasks

Implements the web component contract of the Jakarta EE architecture. Specifies a runtime environment for additional web components, including security, concurrency, lifecycle management, transaction, deployment, and other services

**Lifecycle –**

Three methods are central to the lifecycle of a servlet – these are

1. Class loading
2. Servlet instantiation
3. init() – initialization method called when a servlet instance is created
4. service() – the method called to perform the work. The service() method converts HTTP requests into appropriate doXXX() method calls
5. destroy() – when a servlet instance is taken out of service, the destroy() method is called – only once for that instance

ii) Servlet introduction

Servlet –

• a server-side scripting tool which is capable of reading incoming requests made by the client and generate appropriate responses

• Written in .java file

• We configure URL pattern for the servlet in web.xml

• endpoint request – can configure a unique endpoint

• run on a lifecycle

• handle requests as multithreaded

• commonly known as “Controller” in the popular MVC architecture

• lifecycle executed within the servlet container provided by the web server under the web container

iii) Servlet inheritance hierarchy:

Servlet interface (i) 🡪 Generic Servlet (ac) 🡪 Http Servlet (ac) 🡪 MyServlet

13)

* 1. XML overview
  2. Deployment descriptor (web.xml)
  3. ServletConfig and ServletContext

Configuration objects which are used by servlet container to initialize various parameters of web application. Config is available only within the servlet scope, context is available application-wide

ServletConfig in an interface in the Servlet API and ServletConfig object represents or used to initialize single servlet in web application by servlet container. By using ServletConfig and combination of init-param you can configure any Servlet in J2EE environment

ServletContext object is common for all servlets and they use this object to communicate with the servlet container to get the detail of whole web application or execution environment.

|  |  |
| --- | --- |
| **Servlet Config** | **Servlet Context** |
| Servlet config object represent single servlet | It represents whole web application running on particular JVM and common for all the servlet |
| It’s like local parameter associated with particular servlet | It’s like global parameter associated with whole application |
| It’s a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope | ServletContext has application wide scope so define outside of servlet tag in web.xml file. |
| getServletConfig() method is used to get the config object | getServletContext() method is used to get the context object. |
| for example shopping cart of a user is a specific to particular user so here we can use servlet config | To get the MIME type of a file or application session related information is stored using servlet context object. |

* 1. Request Dispatcher

forward(request,response) – when you want the response of only one page to be sent to the client

include(request,response) – when you want to combine >1 page response to be sent to the client, use the include method

* 1. Redirection vs forwarding

reponse.sendRedirect(“url”)

REDIRECTION creates a whole new request – can provide any URL.

FORWARD/INCLUDE sends your info to a new request

* 1. Exposing/consuming REST API endpoints

1. REST (by Roy Fielding)
2. Representational State and Transfer
3. Used for building WEB services
4. Uses the https protocol
5. Suitable for modular web app dev
6. It is light weight
7. Easy for f/ws like JS and its f/ws to consume these services
8. Communication is done via http/https protocol by using readable XML, JSON, plain text
9. For status check, since it on http/https protocol, the success/failures(exceptions) are shared using http status codes

Six REST architectural constraints

1. Uniform Identifier – endpoints
2. Client – server architecture
3. Layered architecture – separation of concerns
4. Statelessness
5. Cacheable – frequently used data is already stored for later use
6. Code on demand (optional) – send and executable

Providers of REST

i) JEE(JAX-RS) XML, JSON 🡪 Jersey

ii) REST with XML (JAXB)

iii) Spring-RS XML, JSON 🡪 Spring API for RESTful services – rapid application development)

* 1. REST with XML (JAXB)
  2. REST with JSON (Jackson)

14)

i) Front Controller design pattern – 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.

**Controller:** the controller is the initial contact point for handling all requests into the system.

**View:** Represents and displays information to the client.

**Dispatcher:** responsible for view management and navigation

**Helper:** responsible for helping a view or controller complete its processing.

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| • centralized control  • thread safety | • scale limitations  • less efficient processing when handling multiple requests simultaneously |

A picture containing graphical user interface, diagram, application

Description automatically generated

ii) MVC design pattern – an application consist of data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

1. **Model –** contains only the pure application data, no logic (raw data)
2. **View –** presents model data to a user (html)
3. **Controller –** exists between the view and the model – listens to events triggered by the view and executes the appropriate reaction (servlet)

Diagram

Description automatically generated

iii) Handling data with servlets

Filters – **Interface** which can perform pre and post processing logic for the servlet

* Called *interceptors*
* Methods:
* void init(FilterConfig)
* void doFilter(request,response,FilterChain) – used to connect the filter with next filter or to the servlet
* void destroy()

Servlet

Via a filter

15)

16)

17)

* 1. Functional interface – one method

Arrays.sort(ar);

Collections.sort(list);

Internally these make use of Comparable and Comparator interfaces to sort or arrange objects in a certain order

* + 1. Comparable
    2. Comparator