# Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group

Click to Join
Telegram Group



## For More E-Notes

Join Our Community to stay Updated

**TAP ON THE ICONS TO JOIN!**

| | codewitharrays.in freelance project available to buy contact on 8007592194 | |
|---|---|---|
| SR.NO | Project NAME | Technology |
| 1 | Online E-Learning Platform Hub | React+Springboot+MySql |
| 2 | PG Mates / RoomSharing / Flat Mates | React+Springboot+MySql |
| 3 | Tour and Travel management System | React+Springboot+MySql |
| 4 | Election commition of India (online Voting System) | React+Springboot+MySql |
| 5 | HomeRental Booking System | React+Springboot+MySql |
| 6 | Event Management System | React+Springboot+MySql |
| 7 | Hotel Management System | React+Springboot+MySql |
| 8 | Agriculture web Project | React+Springboot+MySql |
| 9 | AirLine Reservation System / Flight booking System | React+Springboot+MySql |
| 10 | E-commerce web Project | React+Springboot+MySql |
| 11 | Hospital Management System | React+Springboot+MySql |
| 12 | E-RTO Driving licence portal | React+Springboot+MySql |
| 13 | Transpotation Services portal | React+Springboot+MySql |
| 14 | Courier Services Portal / Courier Management System | React+Springboot+MySql |
| 15 | Online Food Delivery Portal | React+Springboot+MySql |
| 16 | Muncipal Corporation Management | React+Springboot+MySql |
| 17 | Gym Management System | React+Springboot+MySql |
| 18 | Bike/Car ental System Portal | React+Springboot+MySql |
| 19 | CharityDonation web project | React+Springboot+MySql |
| 20 | Movie Booking System | React+Springboot+MySql |

| | | |
|---|---|---|
| | freelance_Project available to buy contact on 8007592194 | |
| 21 | Job Portal  web project | React+Springboot+MySql |
| 22 | LIC Insurance Portal | React+Springboot+MySql |
| 23 | Employee Management System | React+Springboot+MySql |
| 24 | Payroll Management System | React+Springboot+MySql |
| 25 | RealEstate Property Project | React+Springboot+MySql |
| 26 | Marriage Hall Booking Project | React+Springboot+MySql |
| 27 | Online Student Management portal | React+Springboot+MySql |
| 28 | Resturant management System | React+Springboot+MySql |
| 29 | Solar Management Project | React+Springboot+MySql |
| 30 | OneStepService LinkLabourContractor | React+Springboot+MySql |
| 31 | Vehical Service Center Portal | React+Springboot+MySql |
| 32 | E-wallet Banking Project | React+Springboot+MySql |
| 33 | Blogg Application Project | React+Springboot+MySql |
| 34 | Car Parking booking Project | React+Springboot+MySql |
| 35 | OLA Cab Booking  Portal | React+NextJs+Springboot+MySql |
| 36 | Society management Portal | React+Springboot+MySql |
| 37 | E-College Portal | React+Springboot+MySql |
| 38 | FoodWaste Management Donate System | React+Springboot+MySql |
| 39 | Sports Ground Booking | React+Springboot+MySql |
| 40 | BloodBank mangement System | React+Springboot+MySql |

| | | |
|---|---|---|
| 41 | Bus Tickit Booking Project | React+Springboot+MySql |
| 42 | Fruite Delivery Project | React+Springboot+MySql |
| 43 | Woodworks Bed Shop | React+Springboot+MySql |
| 44 | Online Dairy Product sell Project | React+Springboot+MySql |
| 45 | Online E-Pharma medicine sell Project | React+Springboot+MySql |
| 46 | FarmerMarketplace Web Project | React+Springboot+MySql |
| 47 | Online Cloth Store Project | React+Springboot+MySql |
| 48 | Train Ticket Booking Project | React+Springboot+MySql |
| 49 | Quizz Application Project | JSP+Springboot+MySql |
| 50 | Hotel Room Booking Project | React+Springboot+MySql |
| 51 | Online Crime Reporting Portal Project | React+Springboot+MySql |
| 52 | Online Child Adoption Portal Project | React+Springboot+MySql |
| 53 | online Pizza Delivery System Project | React+Springboot+MySql |
| 54 | Online Social Complaint Portal Project | React+Springboot+MySql |
| 55 | Electric Vehical management system Project | React+Springboot+MySql |
| 56 | Online mess / Tiffin management System Project | React+Springboot+MySql |
| 57 | | React+Springboot+MySql |
| 58 | | React+Springboot+MySql |
| 59 | | React+Springboot+MySql |
| 60 | | React+Springboot+MySql |

# Spring Boot + React JS + MySQL Project List

| Sr.No | Project Name | YouTube Link |
|---|---|---|
| 1 | Online E-Learning Hub Platform Project | https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW |
| 2 | PG Mate / Room sharing/Flat sharing | https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp |
| 3 | Tour and Travel System Project Version 1.0 | https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12 |
| 4 | Marriage Hall  Booking | https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7 |
| 5 | Ecommerce Shopping project | https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq |
| 6 | Bike Rental System Project | https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H |
| 7 | Multi-Restaurant management system | https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB |
| 8 | Hospital management system Project | https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw |
| 9 | Municipal Corporation system Project | https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF |
| 10 | Tour and Travel System Project version 2.0 | https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ |

| Sr.No | Project Name | YouTube Link |
|---|---|---|
| 11 | Tour and Travel System Project version 3.0 | https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug |
| 12 | Gym Management system Project | https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX |
| 13 | Online Driving License system Project | https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn |
| 14 | Online Flight Booking system Project | https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh |
| 15 | Employee management system project | https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H |
| 16 | Online student school or college portal | https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD |
| 17 | Online movie booking system project | https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm |
| 18 | Online Pizza Delivery system project | https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM |
| 19 | Online Crime Reporting system Project | https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO |
| 20 | Online Children Adoption Project | https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N |

# SQL Cheat Sheet

Java Concept Of The Day

## Introduction

**What is SQL?**

SQL stands for Structured Query Language. It is a programming language used to store and manipulate the data in relational databases.

## SQL Database

**CREATE DATABASE :** It creates a new SQL database with specified name.

```
CREATE DATABASE database_name;
```

**DROP DATABASE :** It is used to delete an existing SQL database.

```
DROP DATABASE database_name;
```

**BACKUP DATABASE :** It is used to create full back up of an existing SQL database.

```
BACKUP DATABASE database_name TO DISK
= 'filepath';
```

**BACKUP DATABASE WITH DIFFERENTIAL:** It creates differential back up of an existing database. Differential back up backs up only those parts of the database which have been changed since last back up.

```
BACKUP DATABASE database_name TO DISK
= 'filepath' WITH DIFFERENTIAL;
```

## SQL Constraints

SQL constraints are used to specify the rules for the columns of a table.

**NOT NULL :** A column declared with NOT NULL can't have null values.

**UNIQUE :** A column declared as UNIQUE can't have duplicate values.

**DEFAULT :** It specifies the default value for a column if no value is provided.

**PRIMARY KEY :** It declares a column as primary key.

**FOREIGN KEY :** It declares a column as foreign key.

**CHECK :** It ensures that values in a column must satisfy the given condition.

## SQL Operators

| Operators | Symbols |
|---|---|
| Arithmetic Operators | Add (+), Subtract (-), Multiply (*), Divide (/), Modulus (%) |
| Bitwise Operators | Bitwise AND (&), Bitwise OR ( \| ), Bitwise exclusive OR (^) |
| Comparison Operators | Equal To (=), Smaller Than (<), Greater Than (>), Smaller than or equal to (<=), Greater than or equal to (>=), Not equal to (<>) |
| Compound Operators | Add equals (+=), Subtract equals (-=), Multiply Equals (*=), Divide Equals (/=), Modulus Equals (%=), Bitwise AND equals (&=), Bitwise OR Equals (\|*=), Bitwise exclusive OR equals (^-=) |
| Logical Operators | AND, OR, NOT, ALL, ANY, BETWEEN, IN, EXISTS, LIKE, SOME |

## SQL Comments

```
--Single Line Comment

/*  Multi
    Line
    Comments  */
```

## SQL Views

SQL views are nothing but the virtual tables based on a result set returned by a SQL statement.

**CREATE VIEW :** It is used to create view.

```
CREATE VIEW view_name AS
SELECT column1, column2 ...
FROM table_name WHERE condition;
```

**CREATE OR REPLACE VIEW :** This statement is used to update an already existing view.

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2 ...
FROM table_name WHERE condition;
```

**DROP VIEW :** It is used to remove an already existing view.

```
DROP VIEW view_name;
```

## SQL Tables

**CREATE TABLE :** It is used to create a new table in a database.

```
CREATE TABLE table_name (column1 datatype,
column2 datatype, column3 datatype .... );
```

**INSERT INTO :** It is used to insert new records into a table.

```
INSERT INTO table_name (column1, column2,
column3 ... ) VALUES (value1, value2, value3
... );
```

**DROP TABLE :** It is used to delete an existing table from a database.

```
DROP TABLE table_name;
```

**TRUNCATE TABLE :** It deletes all the data from a table but not the table itself.

```
TRUNCATE TABLE table_name;
```

**ALTER TABLE :** It is used to add, delete and modify table columns.

```
/* Add a column to a table */
ALTER TABLE table_name
ADD column_name datatype;

/* Delete a column from a table */
ALTER TABLE table_name
DROP COLUMN column_name;

/* Rename a column of a table */
ALTER TABLE table_name
RENAME COLUMN old_name to new_name;

/* Change the datatype of a column */
ALTER TABLE table_name
MODIFY column_name datatype;
```

**UPDATE :** It is used to modify or update table records.

```
UPDATE table_name
SET column1 = value1, column2 = value2 ...
WHERE condition;
```

**DELETE :** It is used to delete records from a table.

```
/* Delete all the rows from a table */
DELETE FROM table_name;

/* Delete the rows with condition */
DELETE FROM table_name WHERE condition;
```

**SELECT :** It is used to retrieve data from a table.

```
/* Select all data from a table */
SELECT * FROM table_name;

/* Select data from specific columns */
SELECT column1, column2 ...
FROM table_name;
```

**SELECT DISTINCT :** It selects only distinct values from a table.

```
SELECT DISTINCT column1, column2 ...
FROM table_name;
```

## SQL Functions

**COUNT() :** It returns the number of rows which satisfy the given condition.

```
SELECT COUNT(column_name)
FROM table_name WHERE condition;
```

**AVG() :** It returns average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name WHERE condition;
```

**SUM() :** It returns sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name WHERE condition;
```

**MIN() :** It returns minimum of a specified column.

```
SELECT MIN(column_name)
FROM table_name WHERE condition;
```

**MAX() :** It returns maximum of a specified column.

```
SELECT MAX(column_name)
FROM table_name WHERE condition;
```

**ROUND() :** It is used to round a numeric field.

```
SELECT ROUND(column_name, decimals)
FROM table_name;
```

**NOW() :** It returns current date and time.

```
SELECT NOW() FROM table_name;
```

## SQL Clauses

**WHERE :** It is used to retrieve or update or delete the records based on some condition. This clause can be used with SELECT, UPDATE and DELETE statements.

```
/* SELECT With WHERE */
SELECT column1, column2 ...
FROM table_name
WHERE condition;

/* UPDATE With WHERE */
UPDATE table_name
SET column1 = value1, column2 = value2 ...
WHERE condition;

/* DELETE With WHERE */
DELETE FROM table_name WHERE condition;
```

**ORDER BY :** It is used to sort the records in ascending or descending order.

```
SELECT column1, column2 ...
FROM table_name
ORDER BY column1, column2 ... ASC | DESC;
```

**GROUP BY :** This clause is often used with aggregate functions like SUM(), COUNT(), AVG()... to group the result set by one or two columns.

```
SELECT column_name(s),
aggregate_function_name(column_Name)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

**HAVING :** This clause is added to SQL because WHERE can't be used with aggregate functions.

```
SELECT column_name(s),
aggregate_function_name(column_Name)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition;
```

## SQL Joins

SQL joins are used to combine two or more tables based on a common column between them.

**INNER JOIN :** It selects the records which are common in both the tables.

```
SELECT column_name(s)
FROM table1 INNER JOIN table2 ON
table1.column_name=table2.column_name;
```

**LEFT JOIN :** It returns all the records from left table and matching records from right table.

```
SELECT column_name(s)
FROM table1 LEFT JOIN table2 ON
table1.column_name  = table2.column_name;
```

**RIGHT JOIN :** It returns all the records from right table and matching records from left table.

```
SELECT column_name(s)
FROM table1 RIGHT JOIN table2 ON
table1.column_name  = table2.column_name;
```

**OUTER JOIN or FULL OUTER JOIN :** It returns all the records from both the tables.

```
SELECT column_name(s)
FROM table1 FULL OUTER JOIN table2 ON
table1.column_name  = table2.column_name;
```

**UNION :** It is used to combine the result set of two or more select statements.

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

## SQL Stored Procedures

SQL stored procedure is a group of pre-compiled SQL statements forming one logical unit and they are stored in a database server and can be called whenever required without compiling again and again.

**CREATE PROCEDURE :** This statement is used to create stored procedures.

```
CREATE PROCEDURE procedure_name
@parameter_name data_type ....
AS
  BEGIN
    -- SQL statements
  END
```

**EXEC :** It is used to call stored procedures.

```
EXEC procedure_name;
```

## SQL Miscellaneous

**AND, OR And NOT :** WHERE clause can be used with AND, OR and NOT operators to filter the records with more than one condition.

```
/* AND */
SELECT column1, column2 ...
FROM table_name
WHERE condition1 AND condition2 AND
condition3 ...;

/* OR */
SELECT column1, column2 ...
FROM table_name
WHERE condition1 OR condition2 OR
condition3 ...;

/* NOT */
SELECT column1, column2 ...
FROM table_name
WHERE NOT condition;
```

**EXISTS :** It is used to test for existence of any records in a sub query.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name
WHERE condition);
```

**AS :** It is used to give temporary name called aliases to a table or to a column in a table.

```
/* Alias Column */
SELECT column_name AS alias_name
FROM table_name;

/* Alias Table */
SELECT column_name(s)
FROM table_name AS alias_name;
```

**LIKE :** It is used with WHERE clause to search for a specified pattern in a column.

```
SELECT column1, column2 ...
FROM table_name
WHERE columnN LIKE pattern;
```

**IN :** It is used along with WHERE to specify multiple values in WHERE condition.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2 ...);
```

**BETWEEN :** It used along with WHERE to filter the values within a specified range.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1
AND value2;
```

**IS NULL And IS NOT NULL :** These are used to test for null values.

```
/* IS NULL */
SELECT column_names
FROM table_name
WHERE column_name IS NULL;

/* IS NOT NULL */
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

## SQL Indexes

SQL indexes are used to speed up search queries in the database tables.

**CREATE INDEX :** It is used create indexes on the database tables.

```
CREATE INDEX index_name
ON table_name (column1, column2 ...);
```

**ALTER INDEX RENAME TO :** It is used to rename already existing index.

```
ALTER INDEX old_index_name
RENAME TO new_index_name;
```

**DROP INDEX :** It is used to remove an already existing index on a table.

```
DROP INDEX Index_Name;
```

# HTML Cheat Sheet

## HTML Basic Tags

<!DOCTYPE> : Defines type of the document.
<html> ... </html> : Root of an HTML document.
<head> ... </head> : Container for all the metadata about an HTML document
<title> ... </title> : Defines title for the document.
<meta /> : Defines metadata like character set, viewport, keywords, page description, author etc...
<base /> : Specifies default URL for all links on a page.
<link /> : Defines link to external sources.
<style> ... </style> : Defines style for a document.
<script> ... </script> : Defines client-side scripts.
<noscript> ... </noscript> : Specifies an alternate content to be displayed if the browser doesn't support scripts.
<body> ... </body> : Represents the main body of a document.

## HTML Tables

<table> ... </table> : Table
<tr> ... </tr> : Table Row
<th> ... </th> : Table Header Cell
<td> ... </td> : Table Data
<caption> ... </caption> : Table Caption
<thead> ... </thead> : Group of header contents
<tbody> ... </tbody> : Group of body contents
<tfoot> ... </tfoot> : Group of footer contents
<colgroup> ... </colgroup> : Group of columns
<col /> : Column in a <colgroup>

## HTML Lists

<ul> ... </ul> : Unordered List
<li> ... </li> : List Item
<ol> ... </ol> : Ordered List
<dl> ... </dl> : Description List
<dt> ... </dt> : Terms of the description list
<dd> ... </dd> : Describe each term of the description list

## HTML Frames, Audio, Video & Others

<iframe> ... </iframe> : Inline Frame
<audio> ... </audio> : Audio
<video> ... </video> : Video
<source /> : Resources for <audio>, <video> and <picture> elements
<track /> : Text track for <audio> and <video> elements
<embed /> : Container for external applications
<object> ... </object> : Embedded Object
<param /> : Parameter for <object>

## HTML Text Formatting

 ...  : Paragraph
<br /> : Single line break
<hr /> : Horizontal Rule
 ...  : Bold Text
 ...  : Italic Text
<small> ... </small> : Smaller Text
<mark> ... </mark> : Marked / Highlighted Text
<del> ... </del> : Deleted Text
<ins> ... </ins> : Inserted Text
 ...  : Important Text
 ...  : Emphasized Text
<sub> ... </sub> : Subscript
<sup> ... </sup> : Superscript
<blockquote> ... </blockquote> : Block Quotation
<q> ... </q> : Inline Quotation
<abbr> ... </abbr> : Abbreviations / Acronyms.
<address> ... </address> : Address
<cite> ... </cite> : Citation
<bdo> ... </bdo> : Changes text direction.
<code> ... </code> : Programming Code
<pre> ... </pre> : Preformatted Text
<samp> ... </samp> : Sample output of a program
<kbd> ... <kbd> : Keyboard Input
<var> ... </var> : Variable
<dfn> ... </dfn> : Definition
<meter> ... </meter> : Scalar Measurement
<progress> ... </progress> : Progress bar
<u> ... </u> : Underlined Text

## HTML Forms

<form> ... </form> : HTML form
<input /> : Input Field
<label> ... </label> : Label for <input> element
<button> ... </button> : Clickable button
<textarea> ... </textarea> : Multiline Text Input
<select> ... </select> : Drop down list
<option> ... </option> : An option in drop down list
<optgroup> ... </optgroup> : Groups related options in dropdown list
<fieldset> ... </fieldset> : Groups related elements in a form
<legend> ... </legend> : Caption for <fieldset> element
<datalist> ... </datalist> : List of options
<output> ... </output> : Result

## HTML Headings

<h1> ... </h1> : Heading One
<h2> ... </h2> : Heading Two
<h3> ... </h3> : Heading Three
<h4> ... </h4> : Heading Four
<h5> ... </h5> : Heading Five
<h6> ... </h6> : Heading Six

## HTML Comments

<!-- Single Line comment -->

```
<!--
    Multiple
    Lines
    comment
-->
```

## HTML Links

<a> ... </a> : Hyperlink

## HTML Images

<img /> : Image
<map> ... </map> : Image Map with clickable area
<area /> : Clickable Area in <map>
<picture> ... </picture> : Group of pictures
<figure> ... </figure> : Figures / Diagrams / Illustrations
<figcaption> ... </figcaption> : Caption for <figure>
<canvas> ... </canvas> : Graphics
<svg> ... </svg> : SVG graphics

## HTML Layouts

 ...  : Division / Section / Block Container for HTML elements
 ...  : Inline container for HTML elements
<header> ... </header> : Header content of a document
<main> ... </main> : Main content of a document
<footer> ... </footer> : Footer content of a document
<nav> ... </nav> : Navigation Links
<section> ... </section> : Section in a document
<article> ... </article> : Article in a document
<aside> ... </aside> : Side content in a document
<details> ... </details> : Additional details that user can hide or view on demand.
<summary> ... </summary> : Heading for <details> element
<dialog> ... </dialog> : Dialog Box

# CSS Cheat Sheet (javaconceptoftheday.com)

## CSS Introduction

- CSS stands for Cascading Style Sheets.
- CSS used to style HTML pages.
- CSS determines how HTML elements will be displayed on the screen.
- CSS files end with .css extension.

## CSS Syntax

```
Selector
{
    Property1 : Value1;
    Property2 : Value2;
    ...
    ...
}
```

## CSS Insertion

**External CSS :** Linking external CSS file with an HTML file using <link> tag.
**Ex :** <link rel="stylesheet" href="style.css">

**Internal CSS :** Defining page specific style using <style> element within an HTML page.
**Ex :**
```
<style>
    body {color: black;}
    h1 {color: red;}
</style>
```

**Inline CSS :** Defining element specific style using style attribute of an HTML element.
**Ex :** Inline CSS

## CSS Comments

```
/* Single Line Comment */

/*Multi
  Line
  Comment */
```

## CSS Margins

| | |
|---|---|
| margin<br>margin-top<br>margin-bottom<br>margin-left<br>margin-right | length \| auto \| initial \| inherit |

## CSS Padding

| | |
|---|---|
| padding<br>padding-top<br>padding-bottom<br>padding-left<br>padding-right | length \| initial \| inherit |

## CSS Visibility

| | |
|---|---|
| visibility | visible \| hidden \| collapse \| initial \| inherit |
| display | inline \| block \| grid \| contents \| flex \| inline-block \| inline-flex \| inline-grid \| inline-table \| list-item \| run-in \| table \| none \| initial \| inherit |

## CSS Outline

| | |
|---|---|
| outline-style | dotted \| dashed \| solid \| double \| groove \| ridge \| inset \| outset \| hidden \| none \| initial \| inherit |
| outline-color | invert \| color \| initial \| inherit |
| outline-width | medium \| thick \| thin \| length \| initial \| inherit |
| outline-offset | length \| initial \| inherit |
| outline | outline-width outline-style outline-color \| initial \| inherit |

## CSS Table

| | |
|---|---|
| table-layout | auto \| fixed \| initial \| inherit |
| caption-side | top \| bottom \| initial \| inherit |
| empty-cells | show \| hide \| initial \| inherit |
| border-spacing | length \| initial \| inherit |
| border-collapse | separate \| collapse \| initial \| inherit |

## CSS Selectors

| | | | |
|---|---|---|---|
| Name Selector | P { } | Pseudo Element Selectors | ::after, ::before, ::first-letter, ::first-line, ::marker, ::selection |
| ID Selector | #myID { } | Attribute Selectors | [attribute], [attribute=value], [attribute~=value], [attribute\|=value], [attribute^=value], [attribute$=value], [attribute*=value] |
| Class Selector | . myClass { } | | |
| Universal Selector | * { } | | |
| Group Selector | p, div, span { } | | |
| Descendant Selector | div a { } | | |
| Child Selector | div > a { } | | |
| Adjacent Sibling Selector | div + a { } | General Sibling Selector | div ~ a { } |
| Pseudo Class Selectors | :active, :focus, :hover, :link, :visited, :target, :checked, :disabled, :enabled, :empty, :first-child, :last-child, :only-child, :first-of-type, :last-of-type, :valid, :invalid, :in-range, :out-of-range, :required, :optional, :read-only, :read-write, :root, :lang(language), :not(selector), :nth-child(n), :nth-last-child(n), :nth-of-type(n), :nth-last-of-type(n) | | |

## CSS Background

| | |
|---|---|
| background-color | color \| transparent \| initial \| inherit |
| background-image | URL \| none \| initial \| inherit |
| background-repeat | repeat \| repeat-x \| repeat-y \| no-repeat \| initial \| inherit |
| background-position | left top \| left center \| left bottom \| right top \| right center \| right bottom \| center top \| center center \| center bottom \| x% y% \| xpos ypos \| initial \| inherit |
| background-attachment | scroll \| fixed \| local \| initial \| inherit |
| background-size | auto \| length \| cover \| contain \| initial \| inherit |
| background-clip | border-box \| padding-box \| content-box \| initial \| inherit |
| background-origin | padding-box \| border-box \| content-box \| initial \| inherit |
| background-blend-mode | normal \| multiply \| screen \| overlay \| darken \| lighten \| color-dodge \| saturation \| color \| luminosity |
| background | bg-color bg-image position/bg-size bg-repeat bg-origin bg-clip bg-attachment initial \| inherit |

## CSS Dimensions

| | |
|---|---|
| height<br>width | auto \| length \| % \| initial \| inherit |
| max-height<br>max-width | none \| length \| % \| initial \| inherit |
| min-height<br>min-width | length \| % \| initial \| inherit |

## CSS Lists

| | |
|---|---|
| list-style-type | disc \| circle \| square \| decimal \| decimal-leading-zero \| georgian \| hebrew \| hiragana \| hiragana-iroha \| katakana \| katakana-iroha \| lower-alpha \| lower-greek \| lower-latin \| lower-roman \| upper-alpha \| upper-greek \| upper-latin \| upper-roman \| armenian \| none \| initial \| inherit |
| list-style-image | none \| URL \| initial \| inherit |
| list-style-position | inside \| outside \| initial \| inherit |
| list-style | list-style-type list-style-position list-style-image \| initial \| inherit |

## CSS Borders

| | |
|---|---|
| border<br>border-top<br>border-bottom<br>border-left<br>border-right | border-width border-style border-color \| initial \| inherit |
| border-style<br>border-top-style<br>border-bottom-style<br>border-left-style<br>border-right-style | dotted \| dashed \| solid \| double \| groove \| ridge \| inset \| outset \| hidden \| none \| inherit \| initial |
| border-width<br>border-top-width<br>border-bottom-width<br>border-left-width<br>border-right-width | medium \| thick \| thin \| length \| initial \| inherit |
| border-color<br>border-top-color<br>border-bottom-color<br>border-left-color<br>border-right-color | color \| transparent \| initial \| inherit |
| border-radius<br>border-top-left-radius<br>border-top-right-radius<br>border-bottom-left-radius<br>border-bottom-right-radius | length \| % \| initial \| inherit |
| border-image | source slice width outset repeat \| initial \| inherit |
| border-image-source | none \| URL \| initial \| inherit |
| border-image-width | length \| % \| auto \| initial \| inherit |
| border-image-repeat | stretch \| repeat \| round \| space \| initial \| inherit |
| border-image-slice | number \| % \| fill \| initial \| inherit |
| border-image-outset | length \| number \| initial \| inherit |

## CSS Positions

| | |
|---|---|
| position | static \| absolute \| fixed \| relative \| sticky \| initial \| inherit |
| top<br>bottom<br>left<br>right | auto \| length \| % \| initial \| inherit |
| float | none \| left \| right \| initial \| inherit |
| clear | none \| left \| right \| both \| initial \| inherit |
| clip | auto \| shape \| initial \| inherit |
| z-index | auto \| number \| initial \| inherit |

## CSS Shadow

| | |
|---|---|
| box-shadow | none \| h-offset v-offset blur spread color \| inset \| initial \| inherit |
| text-shadow | h-shadow v-shadow blur-radius color \| none |

## CSS Text

| | |
|---|---|
| color | color \| initial \| inherit |
| font-family | family-name \| generic-family \| initial \| inherit |
| font-size | medium \| xx-small \| x-small \| small \| large \| x-large \| xx-large \| smaller \| larger \| length \| initial \| inherit |
| font-style | normal \| italic \| oblique \| initial \| inherit |
| font-variant | normal \| small-caps \| initial \| inherit |
| font-weight | normal \| bold \| bolder \| lighter \| number \| initial \| inherit |
| font | font-style font-variant font-weight font-size/line-height font-family \| caption \| icon \| menu \| message-box \| small-caption \| status-bar \| initial \| inherit |
| text-align | left \| right \| center \| justify \| initial \| inherit |
| vertical-align | baseline \| length \| sub \| super \| top \| text-top \| middle \| bottom \| text-bottom \| initial \| inherit |
| text-align-last | auto \| left \| right \| center \| justify \| start \| end \| initial \| inherit |
| direction | ltr \| rtl \| initial \| inherit |
| text-decoration-line | none \| underline \| overline \| line-through \| initial \| inherit |
| text-decoration-style | solid \| double \| dotted \| dashed \| wavy \| initial \| inherit |
| text-decoration-color | color \| initial \| inherit |
| text-decoration-thickness | auto \| from-font \| length \| percentage \| initial \| inherit |
| text-decoration | text-decoration-line text-decoration-color text-decoration-style text-decoration-thickness \| initial \| inherit |
| text-transform | none \| capitalize \| uppercase \| lowercase \| initial \| inherit |
| text-indent | length \| initial \| inherit |
| line-height | normal \| number \| length \| initial \| inherit |
| letter-spacing | normal \| length \| initial \| inherit |
| word-spacing | normal \| length \| initial \| inherit |
| white-space | normal \| nowrap \| pre \| pre-line \| pre-wrap \| initial \| inherit |
| text-shadow | h-shadow v-shadow blur-radius color \| none \| initial \| inherit |
| word-break | normal \| break-all \| keep-all \| break-word \| initial \| inherit |
| word-wrap | normal \| break-word \| initial \| inherit |
| writing-mode | horizontal-tb \| vertical-rl \| vertical-lr |
| text-overflow | clip \| ellipsis \| string \| initial \| inherit |
| text-justify | auto \| inter-word \| inter-character \| none \| initial \| inherit |
| font-size-adjust | number \| none \| initial \| inherit |
| font-stretch | ultra-condensed \| extra-condensed \| condensed \| semi-condensed \| normal \| semi-expanded \| expanded \| extra-expanded \| ultra-expanded |

## CSS Overflow

| | |
|---|---|
| overflow | visible \| hidden \| clip \| scroll \| auto \| initial \| inherit |
| overflow-x<br>overflow-y | visible \| hidden \| scroll \| auto \| initial \| inherit |
| overflow-wrap | normal \| anywhere \| break-word \| initial \| inherit |

## CSS Transitions

| | |
|---|---|
| transition-property | none \| all \| property \| initial \| inherit |
| transition-duration | time \| initial \| inherit |
| transition-delay | time \| initial \| inherit |
| transition-timing-function | linear \| ease \| ease-in \| ease-out \| ease-in-out \| step-start \| step-end \| steps(int,start \| end) \| cubic-bezier(n,n,n,n) \| initial \| inherit |
| transition | property duration timing-function delay \| initial \| inherit |

## CSS 2D / 3D

| | |
|---|---|
| transform | none \| transform-functions \| initial \| inherit |
| transform-origin | x-axis y-axis z-axis \| initial \| inherit |
| transform-style | flat \| preserve-3d \| initial \| inherit |
| perspective | length \| none |
| perspective-origin | x-axis y-axis \| initial \| inherit |
| backface-visibility | visible \| hidden \| initial \| inherit |

## CSS Animation

| | |
|---|---|
| animation-name | keyframename \| none \| initial \| inherit |
| animation-duration | time \| initial \| inherit |
| animation-delay | time \| initial \| inherit |
| animation-direction | normal \| reverse \| alternate \| alternate-reverse \| initial \| inherit |
| animation-iteration-count | number \| infinite \| initial \| inherit |
| animation-play-state | paused \| running \| initial \| inherit |
| animation-fill-mode | none \| forwards \| backwards \| both \| initial \| inherit |
| animation-timing-function | linear \| ease \| ease-in \| ease-out \| ease-in-out \| step-start \| step-end \| steps(int,start\|end) \| cubic-bezier(n,n,n,n) \| initial \| inherit |
| animation | name duration timing-function delay iteration-count direction fill-mode play-state |

## CSS Flexible

| | |
|---|---|
| order | number \| initial \| inherit |
| flex-basis | number \| auto \| initial \| inherit |
| flex-grow | number \| initial \| inherit |
| flex-shrink | number \| initial \| inherit |
| flex-wrap | nowrap \| wrap \| wrap-reverse \| initial \| inherit |
| flex-direction | row \| row-reverse \| column \| column-reverse \| initial \| inherit |
| flex-flow | flex-direction flex-wrap \| initial \| inherit |
| flex | flex-grow flex-shrink flex-basis \| auto \| initial \| inherit |
| align-content | stretch \| center \| flex-start \| flex-end \| space-between \| space-around \| space-evenly \| initial \| inherit |
| align-items | stretch \| center \| flex-start \| flex-end \| baseline \| initial \| inherit |
| align-self | auto \| stretch \| center \| flex-start \| flex-end \| baseline \| initial \| inherit |
| justify-content | flex-start \| flex-end \| center \| space-between \| space-around \| space-evenly \| initial \| inherit |

## CSS Columns

| | |
|---|---|
| column-count | number \| auto \| initial \| inherit |
| column-gap | length \| normal \| initial \| inherit |
| column-width | auto \| length \| initial \| inherit |
| column-span | none \| all \| initial \| inherit |
| column-fill | balance \| auto \| initial \| inherit |
| column-rule-style | none \| hidden \| dotted \| dashed \| solid \| double \| groove \| ridge \| inset \| outset \| initial \| inherit |
| column-rule-color | color \| initial \| inherit |
| column-rule-width | medium \| thin \| thick \| length \| initial \| inherit |
| column-rule | column-rule-width column-rule-style column-rule-color \| initial \| inherit |
| columns | auto \| column-width column-count \| initial \| inherit |

## CSS Grids

| | |
|---|---|
| row-gap<br>column-gap | length \| normal \| initial \| inherit |
| gap | row-gap column-gap |
| grid-row-start | auto \| row-line |
| grid-row-end | auto \| row-line \| span n |
| grid-row-gap | length |
| grid-row | grid-row-start \| grid-row-end |
| grid-column-start | auto \| span n \| column-line |
| grid-column-end | auto \| span n \| column-line |
| grid-column-gap | length |
| grid-column | grid-column-start \| grid-column-end |
| grid-gap | grid-row-gap grid-column-gap |
| grid-area | grid-row-start / grid-column-start / grid-row-end / grid-column-end \| itemname |
| grid-auto-rows<br>grid-auto-columns | auto \| max-content \| min-content \| length |
| grid-auto-flow | row \| column \| dense \| row dense \| column dense |
| grid-template-rows<br>grid-template-columns | none \| auto \| max-content \| min-content \| length \| initial \| inherit |
| grid-template-areas | none \| itemnames |
| grid-template | none \| grid-template-rows / grid-template-columns \| grid-template-areas \| initial \| inherit |
| grid | none \| grid-template-rows / grid-template-columns \| grid-template-areas \| grid-template-rows / [grid-auto-flow] grid-auto-columns \| [grid-auto-flow] grid-auto-rows / grid-template-columns \| initial \| inherit |

## CSS Miscellaneous

| | |
|---|---|
| all | Initial \| inherit \| unset |
| box-sizing | content-box \| border-box \| initial \| inherit |
| break-after<br>break-before<br>break-inside | auto \| all \| always \| avoid \| avoid-column \| avoid-page \| avoid-region \| column \| left \| page \| recto \| region \| right \| verso \| initial \| inherit |
| image-rendering | auto \| smooth \| high-quality \| crisp-edges \| pixelated \| initial \| inherit |
| object-fit | fill \| contain \| cover \| scale-down \| none \| initial \| inherit |
| object-position | left \| center \| right \| length \| % \| initial \| inherit |
| opacity | number \| initial \| inherit |
| resize | none \| both \| horizontal \| vertical \| initial \| inherit |
| tab-size | number \| length \| initial \| inherit |
| user-select | auto \| none \| text \| all |

# HTTP Status Codes

javaconceptoftheday.com

| 1xx : Informational Purpose | |
|---|---|
| 100 | Continue |
| 101 | Switching Protocols |
| 102 | Processing |
| 103 | Early Hints |

| 2xx : Success | |
|---|---|
| 200 | Ok |
| 201 | Created |
| 202 | Accepted |
| 203 | Non-Authoritative Information |
| 204 | No Content |
| 205 | Reset Content |
| 206 | Partial Content |
| 207 | Multi Status |
| 208 | Already Reported |
| 226 | IM Used |

| 3xx : Redirection | |
|---|---|
| 300 | Multiple Choices |
| 301 | Moved Permanently |
| 302 | Found |
| 303 | See Other |
| 304 | Not Modified |
| 305 | Use Proxy |
| 306 | No Longer Used |
| 307 | Temporary Redirect |
| 308 | Moved Permanently |

| 4xx : Client Errors | |
|---|---|
| 400 | Bad Request |
| 401 | Unauthorized |
| 402 | Payment Required |
| 403 | Forbidden |
| 404 | Not Found |
| 405 | Method Not Allowed |
| 406 | Not Acceptable |
| 407 | Proxy Authentication Is Required |
| 408 | Request Time Out |
| 409 | Conflict |
| 410 | Gone |
| 411 | Length Required |
| 412 | Precondition Failed |
| 413 | Payload Too Large |
| 414 | URI Too Long |
| 415 | Unsupported Media Type |
| 416 | Range Not Satisfiable |
| 417 | Expectation Failed |
| 421 | Misdirect Request |
| 422 | Unprocessable Entity |
| 423 | Locked |
| 424 | Failed Dependency |
| 425 | Too Early |
| 426 | Upgrade Required |
| 428 | Precondition Required |
| 429 | Too Many Requests |
| 431 | Request Header Fields Too Large |
| 451 | Unavailable For Legal Reasons |

| 5xx : Server Errors | |
|---|---|
| 500 | Internal Server Error |
| 501 | Not Implemented |
| 502 | Bad Gateway |
| 503 | Service Unavailable |
| 504 | Gateway Timeout |
| 505 | HTTP Version Not Supported |
| 507 | Insufficient Storage |
| 508 | Loop Detected |
| 510 | Not Extended |
| 511 | Network Authentication Required |

# Java Keywords Cheat Sheet (javaconceptoftheday.com)

| Data Types | |
|---|---|
| byte | Used to declare primitive byte type of variables. |
| short | Used to declare primitive short type of variables. |
| int | Used to declare primitive integer type of variables. |
| long | Used to declare primitive long type of variables. |
| float | Used to declare primitive float type of variables. |
| double | Used to declare primitive double type of variables. |
| char | Used to declare primitive character type of variables. |
| boolean | Used to declare primitive boolean type of variables. |
| var (From Java 10) | Used to declare a variable of any type. |
| **Control Flow Statements** | |
| if | Used to define if condition statements or blocks. |
| else | Used in if-else blocks. |
| for | Used to define for loops. |
| while | Used to define while loops. |
| do | Used in do-while loops. |
| switch | Used to define switch blocks or switch expressions (From Java 12). |
| case | Used to define case labels in a switch block. |
| break | Used to break a loop or a block. |
| continue | Stops current iteration and starts next iteration in a loop. |
| default | Used for default case label in a switch block and also used for default methods (From Java 8). |
| yield (From Java 13) | Used in switch expressions. |
| **Threads** | |
| synchronized | Used to define synchronized blocks. |
| volatile | Used to define a volatile field whose value is always read from the main memory. |
| **Java 9 Modules** | |
| module | Used to define a module. |
| exports | Used to export all public members of a package in a module. |
| requires | Used to specify required libraries inside a module. |
| open | Used to create an open module. An open module grants reflective access of all of its packages to other modules. |
| opens | Used to expose specific packages for reflective access by other modules. |
| uses | It specifies the services consumed by the current module. |
| provides | It specifies services provided by the current module. |
| **Java 17 Sealed Classes & Interfaces** | |
| sealed | Used to define sealed classes and interfaces. |
| non-sealed | Used to define non-sealed classes and interfaces. |
| permits | Used to specify the sub classes that can extend the sealed class directly. |

| OOP Concepts | |
|---|---|
| class | Used to define a class. |
| new | Used while instantiating a class. |
| static | Used to define static members of a class. |
| interface | Used to define an interface. |
| extends | Used to extend a class. |
| implements | Used to implement an interface. |
| super | Used to access super class members inside a sub class. |
| this | Used to access other members of the same class. |
| abstract | Used to define abstract classes and abstract methods. |
| final | Used to define final classes and final methods. |
| package | Used to specify a package for the current file. |
| enum | Used to define enum types. |
| **Access Modifiers** | |
| private | Used to define private fields, private methods and private constructors. |
| protected | Used to define protected fields, protected methods and protected constructors. |
| public | Used to define public classes, public fields, public methods and public constructors. |
| **Exception Handling** | |
| try | Used to define a try block. |
| catch | Used to define a catch block. |
| finally | Used to define a finally block. |
| throw | Used to throw an exception manually. |
| throws | Used to specify the exceptions which may be thrown by the current method. |
| **Others** | |
| void | Used to indicate that method returns nothing. |
| return | Used to return a value from a method or a block. |
| transient | Used in serialization. A variable which is declared as transient will not be eligible for serialization. |
| strictfp | Used to implement the strict precision of floating point calculations on different platforms. |
| import | Used to import external resources into current Java file. |
| instanceOf | Used to check whether an object is of specified type. |
| native | Used with a method to indicate that a particular method is implemented in native code using Java Native Interfaces(JNI). |
| record (From Java 14) | Used to define a special type of classes which just acts as a data carrier. |
| assert | Used in debugging. |
| const | Reserved but not used. |
| goto | Reserved but not used. |
| _ (Underscore) | From Java 9, _ (underscore) has become a keyword and hence can't be used as an identifier anymore. |
| | |

**Note : true, false and null are not the keywords but reserved for literal values and hence can't be used as identifiers.**

# Java Control Flow Statements Cheat Sheet



**What are control flow statements?**

Control flow statements are the statements in a program which control the flow of execution of a program.

There are three types of control flow statements in Java.

1. Decision Making Statements
2. Looping Statements
3. Branching Statements

## Decision Making Statements

Decision making statements determine which statements to execute depending upon the outcome of a supplied condition or an expression.

There are two different decision making statements available in Java.

1. if statements
2. switch statement

**1) if Statements**

**a) Simple if Statement**

```
if (condition)
{
    //Set of statements
    //These are executed only if condition is TRUE
}
```

**b) if-else statement**

```
if (condition)
{
    //Set of statements
    //These are executed when condition is TRUE
}
else
{
    //Set of statements
    //These are executed when condition is FALSE
}
```

**c) if-else-if Statement**

```
if (condition_1)
{
    //Set of statements
    //These are executed if condition_1 is TRUE
}
else if (condition_2)
{
    //Set of statements
    //These are executed if condition_2 is TRUE
}
else
{
    //Set of statements
    //These are executed if all conditions are FALSE
}
```

**2) switch Statement**

```
switch (key)
{
    case value_1 : Statement_1;
                   break;

    case value_2 : Statement_2;
                   break;
                .
                .
                .
                .
          default : Default_Statement;
                    break;
}
```

## Looping Statements

Looping statements are used to execute a set of statements repeatedly until supplied condition becomes FALSE.

There are four types of loop statements in Java.

1. for loop
2. while loop
3. do-while loop
4. for-each loop

**1) for Loop**

```
for (initialization; termination; increment / decrement)
{
    //Set of statements
}
```

**2) while loop**

```
while (condition)
{
    //Set of statements
    //These are executed until condition is TRUE
}
```

**3) do-while loop**

```
do
{
    //Set of statements
    //These are executed until condition is TRUE
}
while (condition);
```

**4) for-each loop / advanced for loop**

```
for (data_type variable_Name : array / collection)
{
    //Set of statements
}
```

## Branching Statements

Branching statements or jump statements are used to transfer the control of execution to some other part of the program.

There are three branching statements in Java.

1. break statement
2. continue statement
3. return statement

**1) break statement**

```
while (condition)
{
    //Some statements

    if (condition to break)
    {
        break;
    }

    //Some statements
}
```

**2) continue Statement**

```
while (condition)
{
    //Some statements

    if (condition to continue)
    {
        continue;
    }

    //Some statements
}
```

**3) return Statement**

```
return_Type anymethod(Pass_arguments_Here)
{
    //Some statements

    return anyThing or Nothing;
}
```

# Java OOP Concepts Cheat Sheet

## Inheritance

- ✓ Inheritance, as name itself suggests, is used to inherit properties from parent class to child class.
- ✓ Using inheritance, you can reuse existing tried and tested code.
- ✓ Using inheritance, you can also add more features to existing class without modifying it by extending it through its subclass.
- ✓ In Java, inheritance is implemented by using extends keyword.
- ✓ An example :

```
class SuperClass
{
    String superClassField = "Super_Class_Field";

    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}

class SubClass extends SuperClass
{
    String subClassField = "Sub_Class_Field";

    void subClassMethod()
    {
        System.out.println("Sub_Class_Method");
    }
}

public class JavaOOPConcepts
{
    public static void main(String[] args)
    {
        SubClass subClass = new SubClass();

        subClass.subClassMethod();
        System.out.println(subClass.subClassField);

        //SuperClass properties are inherited to
SubClass

        subClass.superClassMethod();
        System.out.println(subClass.superClassField);
    }
}
```

## Abstraction

- ✓ In computer science terms, abstraction means separating ideas from their actual implementations.
- ✓ Using abstraction, you define only ideas in one class so that those ideas can be implemented by its subclasses according to their requirements.
- ✓ In Java, abstraction is implemented by abstract classes and interfaces.
- ✓ An abstract Class example :

```
abstract class AbstractClass
{
    abstract void anIdea();
}

class SubClassOne extends AbstractClass
{
    @Override
    void anIdea()
    {
        System.out.println("An idea is implemented
according to SubClassOne requirement");
    }
}

class SubClassTwo extends AbstractClass
{
    @Override
    void anIdea()
    {
        System.out.println("An idea is implemented
according to SubClassTwo requirement");
    }
}
```

✓ An interface example :

```
interface Interface
{
    void anIdea();
}

class ClassOne implements Interface
{
    @Override
    public void anIdea()
    {
        System.out.println("An idea is implemented
according to ClassOne requirement");
    }
}

class ClassTwo implements Interface
{
    @Override
    public void anIdea()
    {
        System.out.println("An idea is implemented
according to ClassTwo requirement");
    }
}
```

## Polymorphism

- ✓ Poly means many and morphs means forms. So, anything which has multiple forms is called as polymorphism.
- ✓ In computer science terms, any entity like operator or method or constructor which takes many forms and can be used for multiple tasks is called as polymorphism.
- ✓ For example, '+' operator can be used for addition of two numbers as well as for concatenation of two strings.
- ✓ In Java, there are two types of polymorphism - static polymorphism and dynamic polymorphism.
- ✓ Any entity which shows polymorphism during compilation is called static polymorphism.
- ✓ Operator overloading, method overloading and constructor overloading are best examples of static polymorphism.

```
class AnyClass
{
    int i;
    String s;

    //Constructor Overloading

    public AnyClass()
    {
        this.i = 1;
        this.s = "";
    }

    public AnyClass(int i, String s)
    {
        this.i = i;
        this.s = s;
    }

    //Method Overloading

    void anyMethod(int i)
    {
        System.out.println(i+this.i);   //Here, '+' is
used to add two numbers
    }

    void anyMethod(String s)
    {
        System.out.println(s+this.s);   //Here, '+' is
used to concatenate two strings
    }
}
```

- ✓ Any entity which shows polymorphism at run time is called as dynamic polymorphism.
- ✓ Method overriding is the best example of dynamic polymorphism.

```
class SuperClass
{
    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}

class SubClass extends SuperClass
{
    @Override
    void superClassMethod()
    {
System.out.println("Super_Class_Method_Is_Overri
dden");
    }
}

public class JavaOOPConcepts
{
    public static void main(String[] args)
    {
        SuperClass superClass = new SuperClass();

        superClass.superClassMethod();
//Output : Super_Class_Method

        superClass = new SubClass();

        superClass.superClassMethod();
//Output : Super_Class_Method_Is_Overridden
    }
}
```

## Encapsulation

- ✓ Bundling of data and operations to be performed on that data into single unit is called as encapsulation.
- ✓ Encapsulation in Java can be achieved by including both variables (data) and methods (operations) which act upon those variables into a single unit called class.
- ✓ Encapsulation is often used to hide important information from outside the world. It is called data hiding. This can be achieved by declaring all important variables as private and providing public getter and setter methods.

```
class Customer
{
    private int custID;
    private String name;
    private String address;

    //Getter and setter for custID

    public int getCustID()
    {
        return custID;
    }

    public void setCustID(int custID)
    {
        this.custID = custID;
    }

    //Getter and setter for name

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    //Getter and setter for address

    public String getAddress()
    {
        return address;
    }

    public void setAddress(String address)
    {
        this.address = address;
    }
}
```

# Java Array Cheat Sheet

Java Concept Of The Day

## Array Definition

Array is a fixed size index based data structure containing similar type of objects.

**For example :**

**int[] a = new int[10]**
(It is an array of 10 integers)

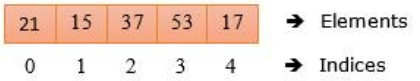
**char[] c = new char[15]**
(It is an array of 15 characters)

**String[] s = new String[20]**
(It is an array of 20 strings)

## Array Structure

Arrays in Java use zero-based indexing to store the elements where first element is stored at $0^{th}$ index, second element at $1^{st}$ index, third element at $2^{nd}$ index and so on.

**int[] intArray = {21, 15, 37, 53, 17}**




## Array Declaration

There are two ways to declare arrays in Java.

```
int[] intArray;
```

OR

```
int intArray[];
```

## Array Initialization

There are three ways to initialize array elements.

**1) Initializing individual elements**

```
int[] intArray = new int[5];
intArray[0] = 21;
intArray[1] = 15;
intArray[2] = 37;
intArray[3] = 53;
intArray[4] = 17;
```

**2) Passing all elements at a time with new operator**

```
int[] intArray = new int[] {21, 15, 37, 53, 17};
```

**3) Passing all elements at a time without new operator**

```
int[] intArray = {21, 15, 37, 53, 17};
```

## Array Traversal

There are two ways to traverse an array.

```
int[] intArray = {21, 15, 37, 53, 17};
```

**1) Using for loop :**

```
for (int i = 0; i < intArray.length; i++)
{
    System.out.println(i);
}
```

**2) Using advanced for loop :**

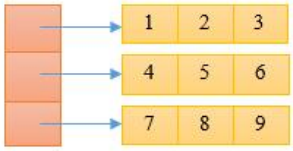
```
for (int i : intArray)
{
    System.out.println(i);
}
```

## Multidimensional Arrays

Multidimensional arrays can be defined as arrays of arrays.

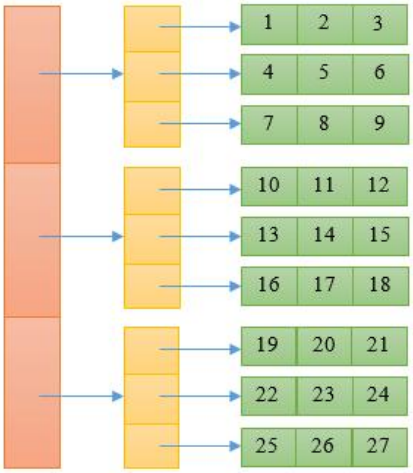
**Two Dimensional Array :**

```
int[][] twoDimensionalArray = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```




**Three Dimensional Array :**

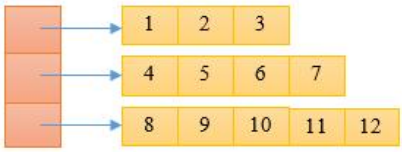
```
int[][][] threeDimensionalArray = { { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} },
                                    { {10, 11, 12}, {13, 14, 15}, {16, 17, 18} },
                                    { {19, 20, 21}, {22, 23, 24}, {25, 26, 27} } };
```




**Jagged Arrays :**

Jagged arrays are also multidimensional arrays containing arrays of different length.

```
int[][] jaggedArray = { {1, 2, 3},
                        {4, 5, 6, 7},
                        {8, 9, 10, 11, 12} };
```




## java.util.Arrays Class

java.util.Arrays class is an utility class which contains many static methods to perform basic operations on an array.

**1) sort() :** Used to sort an array.

**2) stream() :** Returns a stream containing all elements of an array.

**3) spliterator() :** Returns spliterator of an array.

**4) setAll() :** Initializes all elements of an array.

**5) fill() :** Fills the given array with the given value.

**6) copyOf() :** Creates copy of an array.

**7) asList() :** Returns a list containing all elements of an array.

**8) binarySearch() :** It is used to search an array for the given value.

## Array To List

```
List<String> list =
Arrays.asList("One", "Two", "Three", "Four",
"Five");
```

## Array To Stream

```
IntStream stream =
Arrays.stream(new int[] {1, 2, 3, 4, 5});
```

## Array Length

```
int[] intArray = new int[] {1, 2, 3, 4, 5};

System.out.println(intArray.length);
//Output : 5
```

## Anonymous Array

Anonymous array is an array without reference or name.

```
new int[] {1, 2, 3, 4, 5}
```

## Array Pros & Cons

**Pros :**

- Easy to use and implement
- Can hold both primitive types as well as reference types
- Faster data retrieval

**Cons :**

- Fixed size
- Not type safe
- No in-built methods

# Java Strings Cheat Sheet

Java Concept Of The Day

## Strings Basics

**What are strings?**

Strings are nothing but the sequence of characters enclosed within double quotes. For example, "ABC", "xyz", "123" etc.

**How strings are represented in Java?**

In some other languages, strings are represented as array of characters. But in Java, strings are represented as objects of java.lang.String class.

**How do you create string objects in Java?**

There are two ways to create string objects in Java.

1) Using String Literals

```
String s1 = "ABC";
String s2 = "123";
```

2) Using new Operator

```
String s1 = new String("ABC");
String s2 = new String("123");
```

**How string objects are stored in the memory?**

Whenever you create string objects using string literals, those objects will be stored in string constant pool and whenever you create string objects using new operator, such objects will be stored in normal heap memory.

String constant pool is a part of heap memory which is especially dedicated to store string objects. JVM allocates pool space to an object depending upon its content. There will be no two objects in the string constant pool with same content.

Whenever you create a string object using string literal, JVM first checks content of an object to be created. If there exist an object in the pool with same content, then it returns reference of that object. It doesn't create new object. If the content is different from the existing objects then only it creates new object.

**java.lang.String objects are immutable :**

java.lang.String objects, either created using string literals or using new operator, are immutable in nature. That means, once you create a string object, you can't modify the contents of that object. If you try to modify the contents of a string object, a new string object will be created with modified content.

## java.lang.String Methods

charAt()
compareTo()
concat()
contains()
contentEquals()
copyValueOf()
endsWith()
startsWith()
equals()
equalsIgnoreCase()
format()
indexOf()
lastIndexOf()
intern()
isEmpty()
length()
matches()

replace()
replaceAll()
replaceFirst()
split()
subsequence()
substring()
toCharArray()
toLowerCase()
toUpperCase()
trim()
valueOf()

**Java 8 :**
join()

**Java 9 :**
chars()
codePoints()

**Java 11 :**
isBlank()
lines()
repeat()
strip()
stripLeading()
stripTrailing()

**Java 12 :**
indent()
transform()
describeConstable()
resolveConstantDesc()

**Java 15 :**
formatted()
stripIndent()
translateEscapes()

## java.lang.StringBuffer Class

java.lang.StringBuffer class is used to create mutable and thread-safe string objects. In other terms, this class is same as java.lang.String class except its objects are mutable.

It is not possible to create StringBuffer objects using string literals. You have to use new operator to create StringBuffer objects.

**Important Methods :** append(), insert(), replace(), delete(), reverse(), length(), charAt() and substring().

## java.lang.StringBuilder Class

java.lang.StringBuilder class is used to create mutable and non thread-safe string objects. In other terms, this class is same as java.lang.StringBuffer class except its objects are not thread-safe.

It is also not possible to create StringBuilder objects using string literals. You have to use new operator to create StringBuilder objects.

**Important Methods :** append(), insert(), replace(), delete(), reverse(), length(), charAt() and substring().

## String Vs StringBuffer Vs StringBuilder

| String | StringBuffer | StringBuilder |
|---|---|---|
| Immutable | Mutable | Mutable |
| Thread-safe | Thread-safe | Not thread-safe |
| Objects can be created either through string literal or through new operator. | Objects can be created only through new operator. | Objects can be created only through new operator. |
| Objects are stored in string constant pool as well as heap memory. | Objects are stored in heap memory only. | Objects are stored in heap memory only. |
| Slower | Slower | Faster |

## String Intern

String intern refers to string object in the string constant pool.

Interning is the process of creating a string object in String Constant Pool which will be exact copy of string object in heap memory.

intern() method of java.lang.String class is used to perform interning i.e. creating an exact copy of heap string object in string constant pool.

# Java Exception Handling Cheat Sheet

(javaconceptoftheday.com)

## Basics

**What is exception?**

Exception is an abnormal condition which occurs during execution of a program and disrupts the normal flow of a program.

**Ex :** NumberFormatException, ArithmaticException, ArrayIndexOutOfBoundsException, ClassCastException, NullPointerException, StackOverflowError, OutOfMemoryError etc...

**Exception Handling In Java :**

Exceptions in Java are handled using try, catch and finally blocks.

```
try
{
   This block contains statements which may
   throw exceptions during run time.
}
catch(Exception e)
{
   This block handles the exceptions thrown by the
   try block.
}
finally
{
   This block is always executed whether an
   exception is thrown or not and thrown exception
   is caught or not.
}
```

**Rules To Follow While Writing try-catch-finally Blocks :**

- ✓ try, catch and finally blocks form one unit. There must be one try block and one or more catch blocks. finally block is optional.
- ✓ There should not be any statements in between the blocks.
- ✓ If there are multiple catch blocks, the order of catch blocks must be from most specific to general ones. i.e. lower classes in the hierarchy of exceptions must come first and higher classes later.

If try-catch-finally blocks are supposed to return a value :

- ✓ If finally block returns a value then try and catch blocks may or may not return a value.
- ✓ If finally block does not return a value then both try and catch blocks must return a value.
- ✓ finally block overrides return values from try and catch blocks.
- ✓ finally block will be always executed even though try and catch blocks are returning the control.

## Frequently Occurring Exceptions

**1) NullPointerException** occurs when your application tries to access null object.

**2) ArrayIndexOutOfBoundsException** occurs when you try to access an array element with an invalid index i.e index greater than the array length or with a negative index.

**3) NumberFormatException** is thrown when you are trying to convert a string to numeric value like integer, float, double etc..., but input string is not a valid number.

**4) ClassNotFoundException** is thrown when an application tries to load a class at run time but the class with specified name is not found in the classpath.

**5) ArithmeticException** is thrown when an abnormal arithmetic condition arises in an application.

**6) SQLException** is thrown when an application encounters with an error while interacting with the database.

**7) ClassCastException** occurs when an object of one type can not be casted to another type.

**8) IOException** occurs when an IO operation fails in your application.

**9) NoClassDefFoundError** is thrown when Java Runtime System tries to load the definition of a class which is no longer available.

**10) StackOverflowError** is a run time error which occurs when stack overflows. This happens when you keep calling the methods recursively.

## Types Of Exceptions

There are two types of exceptions in Java.

1. **Checked Exceptions** are the exceptions which are checked during compilation itself.
2. **Unchecked Exceptions** are the exceptions which are not checked during compilation. They occur only at run time.

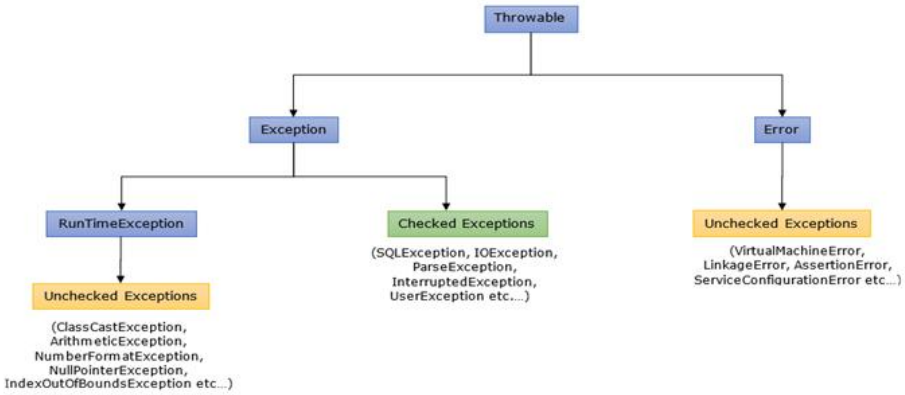
| Checked Exceptions | Unchecked Exceptions |
|---|---|
| They are checked at compile time. | They are not checked at compile time. |
| They are compile time exceptions. | They are run time exceptions. |
| These exceptions must be handled properly either using try-catch blocks or using throws clause, otherwise compiler will throw error. | If these exceptions are not handled properly, compiler will not throw any error. But, you may get error at run time. |
| All the sub classes of java.lang.Exception (except sub classes of java.lang.RunTimeException) are checked exceptions. | All the sub classes of java.lang.RunTimeException and all the sub classes of java.lang.Error are unchecked exceptions. |
| Ex : FileNotFoundException, IOException, SQLException, ClassNotFoundException | Ex : NullPointerException, ArithmeticException, ClassCastException, ArrayIndexOutOfBoundsException |

## Hierarchy Of Exceptions

**java.lang.Throwable** is the super class for all type of errors and exceptions in Java.

It has two sub classes.

1. **java.lang.Error :** It is the super class for all types of errors in Java.
2. **java.lang.Exception :** It is the super class for all types of exceptions in Java.




## throw Keyword

throw keyword is used to throw an exception explicitly.

```
try
{
      throw InstanceOfThrowableType;
}
catch(InstanceOfThrowableType)
{

}
```

where, InstanceOfThrowableType must be an object of type Throwable or subclass of Throwable.

## throws Keyword

throws keyword is used to specify the exceptions that may be thrown by the method.

```
return_type method_name(parameter_list) throws
exception_list
{
    //some statements
}
```

where, exception_list is the list of exceptions that method may throw. Exceptions must be separated by commas.

## Try-with Resources

Try with resources blocks are introduced from Java 7. In these blocks, resources used in try blocks are auto-closed. No need to close the resources explicitly. But, Java 7 try with resources has one drawback. It requires resources to be declared locally within try block. It doesn't recognize resources declared outside the try block. That issue has been resolved in Java 9.

**Before Java 7**

```
//Declare resources here

try
{
      //Use resources here
}
catch (Exception e)
{
      //Catch exceptions here if any
}
finally
{
      //Close resources here
}
```

**After Java 7**

```
try (Declare resources here OR ELSE use local
variable referring to a declared resource)
{
      //Use resources here
}
catch (Exception e)
{
      //Catch exceptions here if any
}

//Resources are auto-closed
//No need to close resources explicitly
```

**After Java 9**

```
//Declare resources here

try (Pass reference of declared resources
here)
{
      //Use resources here
}
catch (Exception e)
{
      //Catch exceptions here if any
}

//Resources are auto-closed
//No need to close resources explicitly
```

# Java Collections Cheat Sheet

Java Concept Of The Day

## Basics

**What is Java Collection Framework?**

Java Collection Framework is a framework which provides some predefined classes and interfaces to store and manipulate the group of objects. Using Java collection framework, you can store the objects as a List or as a Set or as a Queue or as a Map and perform basic operations like adding, removing, updating, sorting, searching etc... with ease.

**Why Java Collection Framework?**

Earlier, arrays are used to store the group of objects. But, arrays are of fixed size. You can't change the size of an array once it is defined. It causes lots of difficulties while handling the group of objects. To overcome this drawback of arrays, Java Collection Framework is introduced from JDK 1.2.

**Java Collections Hierarchy :**

All the classes and interfaces related to Java collections are kept in java.util package. List, Set, Queue and Map are four top level interfaces of Java collection framework. All these interfaces (except Map) inherit from java.util.Collection interface which is the root interface in the Java collection framework.

## List

**Intro :**

- List is a sequential collection of objects.
- Elements are positioned using zero-based index.
- Elements can be inserted or removed or retrieved from any arbitrary position using an integer index.

**Popular Implementations :**

- ArrayList, Vector And LinkedList

**Internal Structure :**

- **ArrayList :** Internally uses re-sizable array which grows or shrinks as we add or delete elements.
- **Vector :** Same as ArrayList but it is synchronized.
- **LinkedList :** Elements are stored as Nodes where each node consists of three parts – Reference To Previous Element, Value Of The Element and Reference To Next Element.

**Null Elements :**

- **ArrayList :** Yes
- **Vector :** Yes
- **LinkedList :** Yes

**Duplicate Elements :**

- **ArrayList :** Yes
- **Vector :** Yes
- **LinkedList :** Yes

**Order Of Elements :**

- **ArrayList :** Insertion Order
- **Vector :** Insertion Order
- **LinkedList :** Insertion Order

**Synchronization :**

- **ArrayList :** Not synchronized
- **Vector :** Synchronized
- **LinkedList :** Not synchronized

**Performance :**

- **ArrayList :** Insertion -> O(1) (if insertion causes restructuring of internal array, it will be O(n)), Removal -> O(1) (if removal causes restructuring of internal array, it will be O(n)), Retrieval -> O(1)
- **Vector :** Similar to ArrayList but little slower because of synchronization.
- **LinkedList :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(n)

**When to use?**

- **ArrayList :** Use it when more search operations are needed then insertion and removal.
- **Vector :** Use it when you need synchronized list.
- **LinkedList :** Use it when insertion and removal are needed frequently.

## Queue

**Intro :**

- Queue is a data structure where elements are added from one end called tail of the queue and elements are removed from another end called head of the queue.
- Queue is typically FIFO (First-In-First-Out) type of data structure.

**Popular Implementations :**

- PriorityQueue, ArrayDeque and LinkedList (implements List also)

**Internal Structure :**

- **PriorityQueue :** It internally uses re-sizable array to store the elements and a Comparator to place the elements in some specific order.
- **ArrayDeque :** It internally uses re-sizable array to store the elements.

**Null Elements :**

- **PriorityQueue :** Not allowed
- **ArrayDeque :** Not allowed

**Duplicate Elements :**

- **PriorityQueue :** Yes
- **ArrayDeque :** Yes

**Order Of Elements :**

- **PriorityQueue :** Elements are placed according to supplied Comparator or in natural order if no Comparator is supplied.
- **ArrayDeque :** Supports both LIFO and FIFO

**Synchronization :**

- **PriorityQueue :** Not synchronized
- **ArrayDeque :** Not synchronized

**Performance :**

- **PriorityQueue :** Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(1)
- **ArrayDeque :** Insertion -> O(1) , Removal -> O(n), Retrieval -> O(1)

**When to use?**

- **PriorityQueue :** Use it when you want a queue of elements placed in some specific order.
- **ArrayDeque :** You can use it as a queue OR as a stack.

## Set

**Intro :**

- Set is a linear collection of objects with no duplicates.
- Set interface does not have its own methods. All its methods are inherited from Collection interface. It just applies restriction on methods so that duplicate elements are always avoided.

**Popular Implementations :**

- HashSet, LinkedHashSet and TreeSet

**Internal Structure :**

- **HashSet :** Internally uses HashMap to store the elements.
- **LinkedHashSet :** Internally uses LinkedHashMap to store the elements.
- **TreeSet :** Internally uses TreeMap to store the elements.

**Null Elements :**

- **HashSet :** Maximum one null element
- **LinkedHashSet :** Maximum one null element.
- **TreeSet :** Doesn't allow even a single null element

**Duplicate Elements :**

- **HashSet :** Not allowed
- **LinkedHashSet :** Not allowed
- **TreeSet :** Not allowed

**Order Of Elements :**

- **HashSet :** No order
- **LinkedHashSet :** Insertion order
- **TreeSet :** Elements are placed according to supplied Comparator or in natural order if no Comparator is supplied.

**Synchronization :**

- **HashSet :** Not synchronized
- **LinkedHashSet :** Not synchronized
- **TreeSet :** Not synchronized

**Performance :**

- **HashSet :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **LinkedHashSet :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **TreeSet :** Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(log(n))

**When to use?**

- **HashSet :** Use it when you want only unique elements without any order.
- **LinkedHashSet :** Use it when you want only unique elements in insertion order.
- **TreeSet :** Use it when you want only unique elements in some specific order.

## Map

**Intro :**

- Map stores the data in the form of key-value pairs where each key is associated with a value.
- Map interface is part of Java collection framework but it doesn't inherit Collection interface.

**Popular Implementations :**

- HashMap, LinkedHashMap And TreeMap

**Internal Structure :**

- **HashMap :** It internally uses an array of buckets where each bucket internally uses linked list to hold the elements.
- **LinkedHashMap :** Same as HashMap but it additionally uses a doubly linked list to maintain insertion order of elements.
- **TreeMap :** It internally uses Red-Black tree.

**Null Elements :**

- HashMap : Only one null key and can have multiple null values
- LinkedHashMap : Only one null key and can have multiple null values.
- TreeMap : Doesn't allow even a single null key but can have multiple null values.

**Duplicate Elements :**

- **HashMap :** Doesn't allow duplicate keys but can have duplicate values.
- **LinkedHashMap :** Doesn't allow duplicate keys but can have duplicate values.
- **TreeMap :** Doesn't allow duplicate keys but can have duplicate values.

**Order Of Elements :**

- **HashMap :** No Order
- **LinkedHashMap :** Insertion Order
- **TreeMap :** Elements are placed according to supplied Comparator or in natural order of keys if no Comparator is supplied.

**Synchronization :**

- **HashMap :** Not synchronized
- **LinkedHashMap :** Not Synchronized
- **TreeMap :** Not Synchronized

**Performance :**

- **HashMap :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **LinkedHashMap :** Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **TreeMap :** Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(log(n))

**When to use?**

- **HashMap :** Use it if you want only key-value pairs without any order.
- **LinkedHashMap :** Use it if you want key-value pairs in insertion order.
- **TreeMap :** Use it when you want key-value pairs sorted in some specific order.

# Java Thread Cheat Sheet

javaconceptoftheday.com

## Basic Definitions

**What is thread?**

Thread is a smallest executable unit of a process. Thread has its own path of execution in a process. A process can have multiple threads.

**What is process?**

Process is an executing instance of an application. For example, when you double click MS Word icon in your computer, you start a process that will run MS word application.

**What is application?**

Application is a program which is designed to perform a specific task. For example : MS Word, Google Chrome, a video or audio player etc.

**What is multithreaded programming?**

In a program or in an application, when two or more threads execute their task simultaneously then it is called multithreaded programming. Java supports multithreaded programming.

## Types Of Threads

There are two types of threads in Java.

**1) User Threads :**

User threads are threads which are created by the application or user. They are high priority threads. JVM will not exit until all user threads finish their execution. JVM wait for user threads to finish their task. These threads are foreground threads.

**2) Daemon Threads :**

Daemon threads are threads which are mostly created by the JVM. These threads always run in background. These threads are used to perform some background tasks like garbage collection. These threads are less priority threads. JVM will not wait for these threads to finish their execution. JVM will exit as soon as all user threads finish their execution.

## Thread Priority

**MIN_PRIORITY :**

It defines the lowest priority that a thread can have and it's value is 1.

**NORM_PRIORITY :**

It defines the normal priority that a thread can have and it's value is 5.

**MAX_PRIORITY :**

It defines the highest priority that a thread can have and it's value is 10.

The default priority of a thread is same as that of it's parent. We can change the priority of a thread at any time using setPriority() method.

## Thread States

There are six thread states - NEW, RUNNABLE, BLOCKED, WAITING, TIMED _WAITING and TERMINATED. At any point of time, a thread will be in any one of these states.

**NEW :** A thread will be in this state before calling start() method.

**RUNNABLE :** A thread will be in this state after calling the start() method.

**BLOCKED :** A thread will be in this state when a thread is waiting for object lock to enter into synchronized method/block or a thread will be in this state if deadlock occurs.

**WAITING :** A thread will be in this state when wait() or join() method is called.

**TIMED_WAITING :** A thread will be in this state when sleep() or wait() with timeOut or join() with timeOut is called.

**TERMINATED :** A thread will be in this state once it finishes it's execution.

## How do you create threads in Java?

There are two ways to create threads in Java.

**1) By extending java.lang.Thread class**

```
class MyThread extends Thread
{
   @Override
   public void run()
   {
      //Keep the task to be performed here
   }
}

//Creating and starting MyThread
MyThread myThread = new MyThread();
myThread.start();
```

**2) By implementing java.lang.Runnable interface**

```
class MyRunnable implements Runnable
{
   @Override
   public void run()
   {
      //Keep the task to be performed here
   }
}

//Creating and starting MyRunnable
Thread t = new Thread(new MyRunnable());
t.start();
```

## Thread Synchronization

Through synchronization, we can make the threads to execute a particular method or block in sync not simultaneously. Synchronization in Java is achieved using **synchronized** keyword.

When a method or block is declared as synchronized, only one thread can enter into that method or block.

The synchronization in Java is built around an entity called object lock or monitor.

Any thread wants to enter into synchronized methods or blocks of any object, they must acquire object lock associated with that object and release the lock after they are done with the execution.

```
synchronized void synchronizedMethod()
{
      //Synchronized Method
}
```

## Deadlock

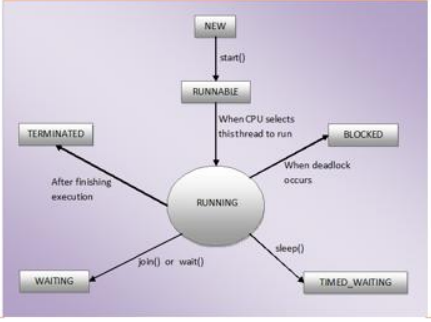
Deadlock in Java is a condition which occurs when two or more threads get blocked waiting for each other for an infinite period of time to release the resources (Locks) they hold.

Lock ordering and lock timeout are two methods which are used to avoid the deadlock in Java.

**Lock Ordering :** In this method of avoiding the deadlock, some predefined order is applied for threads to acquire the locks they need.

**Lock Timeout :** It is another deadlock preventive method in which we specify the time for a thread to acquire the lock. If it fails to acquire the specified lock in the given time, then it should give up trying for a lock and retry after some time.

## Thread Life Cycle




## Java.lang.Thread Methods

**start() :**

It starts execution of a thread.

**run() :**

It contains main task to be performed by the thread.

**sleep() :**

It makes the currently executing thread to pause it's execution for a specified period of time. When the thread is going for sleep, it does not release the locks it holds.

**join() :**

Using this method, you can make the currently executing thread to wait for some other threads to finish their task.

**yield() :**

It causes the currently executing thread to temporarily pause its execution and allow other threads to execute.

**wait() :**

It makes the currently executing thread to release the lock of this object and wait until some other thread notifies it.

**notify() :**

It wakes up one thread randomly which is waiting for this object's lock.

**notifyAll() :**

It wakes up all thread which are waiting for this object's lock. But, only one thread will acquire lock of this object depending upon the priority.

**isAlive() :**

It checks whether a thread is alive or not.

**isDaemon() :**

It checks whether a thread is daemon thread or user thread.

**setDaemon() :**

It sets daemon status of a thread.

**currentThread() :**

It returns a reference to currently executing thread.

**interrupt() :**

It is used to interrupt a thread.

**isInterrupted() :**

It checks whether a thread is interrupted or not.

**getId() :**

It returns ID of a thread.

**getState() :**

It returns current state of a thread.

**getName() and setName() :**

Getter and setter for name of a thread.

**getPriority() and setPriority() :**

Getter and setter for priority of a thread.

**getThreadGroup() :**

It returns a thread group to which this thread belongs to.

## Inter Thread Communication

Threads in Java communicate with each other using wait(), notify() and notifyAll() methods.

**wait() :** This method tells the currently executing thread to release the lock of this object and wait until some other thread acquires the same lock and notify it using either notify() or notifyAll() methods.

**notify() :** This method wakes up one thread randomly that called wait() method on this object.

**notifyAll() :** This method wakes up all the threads that called wait() method on this object. But, only one thread will acquire lock of this object depending upon the priority.

# Java 8 Interview Sample Coding Questions

Java Concept Of The Day

### Separate Odd And Even Numbers

```
listOfIntegers.stream()
              .collect(Collectors.partitioningBy(i -> i % 2 == 0));
```

### Frequency Of Each Character In String

```
inputString.chars()
           .mapToObj(c -> (char) c)
           .collect(Collectors.groupingBy(Function.identity(),
      Collectors.counting()));
```

### Sort The List In Reverse Order

```
anyList.stream().sorted(Comparator.reverseOrder()).forEach(System.
out::println);
```

### Print Multiples Of 5 From The List

```
listOfIntegers.stream()
              .filter(i -> i % 5 == 0).forEach(System.out::println);
```

### Merge Two Unsorted Arrays Into Single Sorted Array

```
IntStream.concat(Arrays.stream(a),Arrays.stream(b))
.sorted().toArray();
```

### Merge Two Unsorted Arrays Into Single Sorted Array Without Duplicates

```
IntStream.concat(Arrays.stream(a),Arrays.stream(b))
.sorted().distinct().toArray();
```

### Three Max & Min Numbers From The List

```
//Min 3 Numbers
listOfIntegers.stream().sorted().limit(3).forEach(System.out::println);

//Max 3 Numbers
listOfIntegers.stream().sorted(Comparator.reverseOrder()).limit(3).fo
rEach(System.out::println);
```

### Sort List Of Strings In Increasing Order Of Their Length

```
listOfStrings.stream().sorted(Comparator.comparing(String::length)).
forEach(System.out::println);
```

### Sum & Average Of All Elements Of An Array

```
//Sum
Arrays.stream(inputArray).sum();

//Average
Arrays.stream(inputArray).average().getAsDouble();
```

### Reverse An Integer Array

```
IntStream.rangeClosed(1, array.length)
         .map(i -> array[array.length - i])
         .toArray();
```

### Palindrome Program In Java 8

```
IntStream.range(0, str.length()/2)
.noneMatch(i -> str.charAt(i) != str.charAt(str.length() - i -1));
```

### Last Element Of An Array

```
listOfStrings.stream().skip(listOfStrings.size()-1).findFirst().get();
```

### Age Of Person In Years

```
LocalDate birthDay = LocalDate.of(1985, 01, 23);
LocalDate today = LocalDate.now();
System.out.println(ChronoUnit.YEARS.between(birthDay, today));
```

### Remove Duplicate Elements From List

```
listOfStrings.stream().distinct().collect(Collectors.toList());
```

### Frequency Of Each Element In An Array

```
anyList.stream().collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));
```

### Join List Of Strings With Prefix, Suffix And Delimiter

```
listOfStrings.stream().collect(Collectors.joining("Delimiter", "Prefix",
"Suffix"));
```

### Maximum & Minimum In A List

```
listOfIntegers.stream().max(Comparator.naturalOrder()).get();

listOfIntegers.stream().min(Comparator.naturalOrder()).get();
```

### Anagram Program In Java 8

```
s1=Stream.of(s1.split("")).map(String::toUpperCase).sorted().collect
(Collectors.joining());

s2=Stream.of(s2.split("")).map(String::toUpperCase).sorted().collect
(Collectors.joining());

If s1 and s2 are equal, then they are anagrams.
```

### Sum Of All Digits Of A Number

```
Stream.of(String.valueOf(inputNumber).split(""))
      .collect(Collectors.summingInt(Integer::parseInt));
```

### Second Largest Number In An Integer Array

```
listOfIntegers.stream().sorted(Comparator.reverseOrder()).skip(1)
.findFirst().get();
```

### Common Elements Between Two Arrays

```
list1.stream().filter(list2::contains).forEach(System.out::println);
```

### Reverse Each Word Of A String

```
Arrays.stream(str.split(" "))
      .map(word -> new StringBuffer(word).reverse())
      .collect(Collectors.joining(" "));
```

### Sum Of First 10 Natural Numbers

```
IntStream.range(1, 11).sum();
```

### Find Strings Which Start With Number

```
listOfStrings.stream()
             .filter(str -> Character.isDigit(str.charAt(0)))
             .forEach(System.out::println);
```

### Find Duplicate Elements From An Array

```
listOfIntegers.stream()
              .filter(i -> ! set.add(i))
              .collect(Collectors.toSet());
```

### Fibonacci Series

```
Stream.iterate(new int[] {0, 1}, f -> new int[] {f[1], f[0]+f[1]})
      .limit(10)
      .map(f -> f[0])
      .forEach(i -> System.out.print(i+" "));
```

# Java JDBC Cheat Sheet

Java Concept Of The Day

## Basics

**What is JDBC?**

JDBC - Java Database Connectivity - is an API which is used by the Java applications to interact with the database management systems.

It consists of several classes and interfaces - written entirely in Java - which can be used to establish connection with the database, send the queries to the database and process the results returned by the database.

**What are JDBC Drivers?**

JDBC API doesn't directly interact with the database. It uses JDBC driver of that particular database with which it wants to interact.

JDBC drivers are nothing but the implementations of classes and interfaces provided in the JDBC API. These implementations are provided by a particular database vendor and supplied along with the database. These implementations are used by the JDBC API to interact with that database.

## Types Of JDBC Drivers

There are four types of JDBC drivers.

**1) Type 1 JDBC Drivers / JDBC-ODBC Bridge Drivers**

This type of drivers translates all JDBC calls into ODBC calls and sends them to ODBC driver which interact with the database.

These drivers just acts as a bridge between JDBC and ODBC API and hence the name JDBC-ODBC bridge drivers.

They are partly written in Java.

**2) Type 2 JDBC Drivers / Native API Drivers**

This type of drivers translates all JDBC calls into database specific calls using native API of the database.

They are also not entirely written in Java.

**3) Type 3 JDBC Drivers / Network Protocol Drivers**

This type of drivers make use of application server or middle-tier server which translates all JDBC calls into database specific network protocol and then sent to the database.

They are purely written in Java.

**4) Type 4 JDBC Drivers / Native Protocol Drivers**

This type of JDBC drivers directly translate all JDBC calls into database specific network protocols without a middle tier.

They are most popular of all 4 type of drivers. They are also called thin drivers. They are entirely written in Java.

## JDBC API

JDBC API is comprised of two packages - java.sql and javax.sql. Below are the some important classes and interfaces of JDBC API.

**java.sql.DriverManager (Class) :**

It acts as a primary mediator between your Java application and the driver of the database you want to connect with. Driver class of every database you want to connect with first has to get registered with this class before you start interacting with the database.

**java.sql.Connection (Interface) :**

It represents a session between Java application and a database. All SQL statements are executed and results are returned within the context of a Connection object. It is mainly used to create Statement, PreparedStatement and CallableStatement objects. You can also use it to retrieve the metadata of a database like name of the database product, name of the JDBC driver, major and minor version of the database etc...

**java.sql.Statement (Interface) :**

It is used to execute static SQL queries.

**java.sql.PreparedStatement (Interface) :**

It is used to execute parameterized or dynamic SQL queries.

**java.sql.CallableStatement (Interface) :**

It is used to execute SQL stored procedures.

**java.sql.ResultSet (Interface) :**

It contains the data returned from the database.

**java.sql.ResultSetMetaData (Interface) :**

This interface provides quick overview about a ResultSet object like number of columns, column name, data type of a column etc...

**java.sql.DatabaseMetaData (Interface) :**

It provides comprehensive information about a database.

**java.sql.Date (Class) :**

It represents a SQL date value.

**java.sql.Time (Class) :**

It represents a SQL time value.

**java.sql.Blob (Interface) :**

It represents a SQL BLOB (Binary Large Object) value. It is used to store/retrieve image files.

**java.sql.Clob (Interface) :**

It represents a SQL CLOB (Character Large Object) value. It is used to store/retrieve character files.

## Database Connection Using JDBC API

**Step 1 : Updating the class path with JDBC Driver**

Add JDBC driver of a database with which you want to interact in the class path. JDBC driver is the jar file provided by the database vendors along with the database. It contains the implementations for all classes and interfaces of JDBC API with specific to that database.

**Step 2 : Registering the driver class**

```
Class.forName("Pass_Driver_Class_Here");
```

**Step 3 : Creating the Connection object.**

```
Connection con = DriverManager.getConnection(URL, username, password);
```

**Step 4 : Creating the Statement Object**

```
Statement stmt = con.createStatement();
```

**Step 5 : Execute the queries.**

```
ResultSet rs = stmt.executeQuery("select * from AnyTable");
```

**Step 6 : Close the resources.**

Close ResultSet, Statement and Connection objects.

## Transaction Management

A transaction is a group of operations used to perform a particular task.

A transaction is said to be successful only if all the operations in a transaction are successful. If any one operation fails, the whole transaction will be cancelled.

In JDBC, transactions are managed using three methods of a Connection interface.

**setAutoCommit() :** It sets the auto commit mode of this connection object. By default it is true. It is set to false to manually manage the transactions.

**commit() :** It is called only when all the operations in a transaction are successful.

**rollback() :** It is called if any one operation in a transaction fails.

## Batch Processing

Batch processing allows us to group similar queries into one unit and submit them all at once for execution. It reduces the communication overhead significantly and increases the performance.

Three methods of Statement interface are used for batch processing.

**addBatch() :** It is used to add SQL statement to the batch.

**executeBatch() :** It executes all SQL statements of a batch and returns an array of integers where each integer represents the status of a respective SQL statement.

**clearBatch() :** It removes all SQL statements added in a batch.

## executeQuery() Vs executeUpdate() Vs execute()

| executeQuery() | executeUpdate() | execute() |
|---|---|---|
| This method is used to execute the SQL statements which retrieve some data from the database. | This method is used to execute the SQL statements which update or modify the database. | This method can be used for any kind of SQL statements. |
| This method returns a ResultSet object which contains the results returned by the query. | This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing. | This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing. |
| This method is used to execute only select queries. | This method is used to execute only non-select queries. | This method can be used for both select and non-select queries. |
| Ex: SELECT | Ex:<br>DML → INSERT, UPDATE and DELETE<br>DDL → CREATE, ALTER | This method can be used for any type of SQL statements. |

## Statement Vs PreparedStatement Vs CallableStatement

| Statement | PreparedStatement | CallableStatement |
|---|---|---|
| It is used to execute normal SQL queries. | It is used to execute parameterized or dynamic SQL queries. | It is used to call the stored procedures. |
| It is preferred when a particular SQL query is to be executed only once. | It is preferred when a particular query is to be executed multiple times. | It is preferred when the stored procedures are to be executed. |
| You cannot pass the parameters to SQL query using this interface. | You can pass the parameters to SQL query at run time using this interface. | You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT. |
| This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc. | It is used for any kind of SQL queries which are to be executed multiple times. | It is used to execute stored procedures and functions. |
| The performance of this interface is very low. | The performance of this interface is better than the Statement interface (when used for multiple execution of same query). | The performance of this interface is high. |


<CODEWITHARRAY'S/>


https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project