**CORE JAVA Notes and some topics covered from ADVANCE JAVA-**

**By – PARAS JAIN**

**Introduction to Java:**

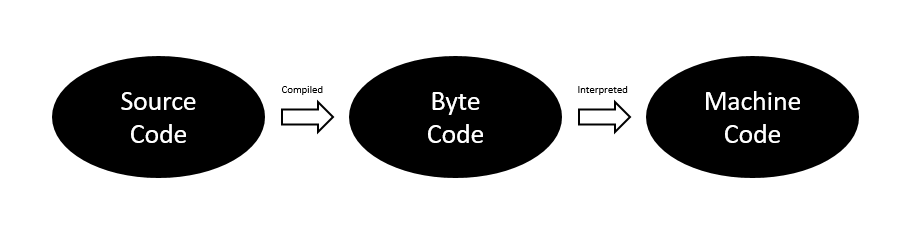
Java is an object-oriented programming language developed by Sun Microsystems of the USA in 1991.

It was originally called Oak by James Goslin. He was one of the inventors of Java.

Java = Purely Object-Oriented

**How Java Works?**

Java is compiled into the bytecode and then it is interpreted to machine code.



**Java Installation:**

* Go to Google & type “Install JDK” (Installs Java JDK)
* Go to Google & type “Install IntelliJ Idea” (Installs Java IDE)

**JDK –**Java Development Kit = Collection of tools used for developing and running java programs.

**JRE –**Java Runtime Environment = Helps in executing programs developed in JAVA.

**Naming Conventions:**

* For classes, we use Pascal Convention. First and Subsequent characters from a word are capital letters (uppercase)

      Example: Main, MyScanner, MyEmployee, CodeWithHarry

* For functions and variables, we use camelCaseConvention. Here the first character is lowercase and the subsequent characters are uppercase like below:

      Main, myScanner, myMarks, CodeWithHarry

### Java Tutorial: Variables and Data Types in Java Programming

Just like we have some rules that we follow to speak English (the grammar), we have some rules to follow while writing a Java program. The set of these rules is called syntax. [It’s like Vocabulary and Grammar of Java :)]

#### Variables

A variable is a container that stores a value.

This value can be changed during the execution of the program.

Example: int number = 8; (Here, int is a data type, number is the variable name and 8 is the value that it contains/stores)

##### **Rules for declaring a variable name:**

We can choose a name while declaring a Java variable if the following rules are followed:

* Must not begin with a digit. (E.g. 1arry is an invalid variable)
* Name is case sensitive. (harry and Harr are different)
* Should not be a keyword (like Void)
* White Space not allowed. (int Code With Harry is invalid)
* Can contain alphabets, $character, \_character, and digits if the other conditions are met.

#### Data Types:

Data types in Java fall under the following categories

1. Primitive Data Types (Intrinsic)
2. Non-Primitive Data Types (Derived)

#### Primitive Data Types:

Java is statically typed i.e. variables must be declared before use.

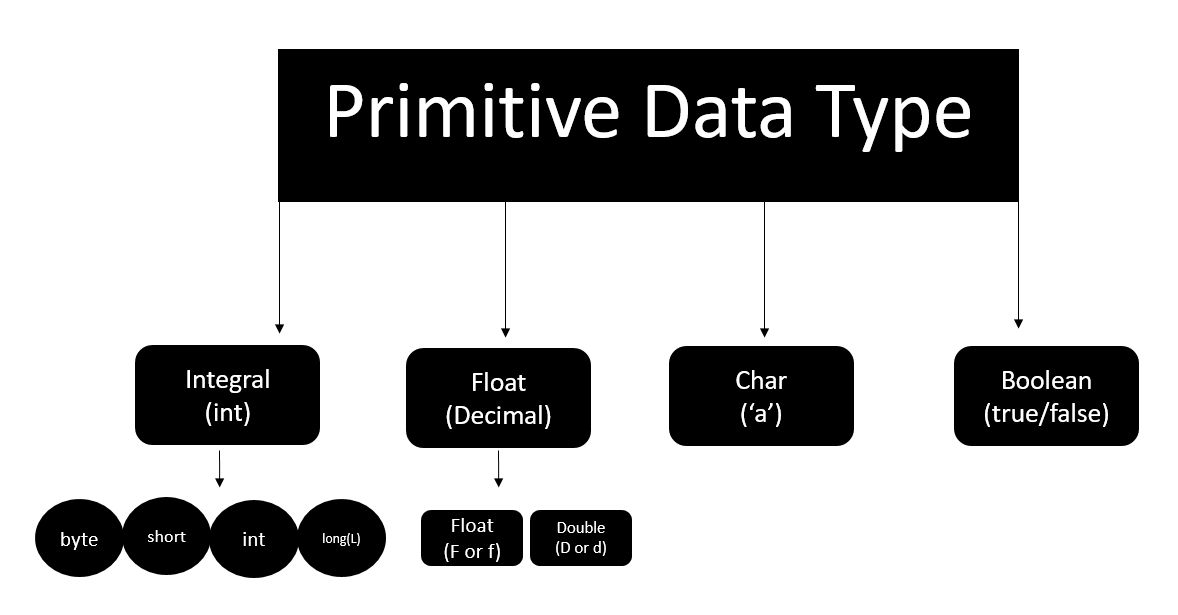
There are 8 primitive data types supported by Java:

1. byte     ➼   •value ranges from -128 to 127  
                     •takes 1 byte   
                     •default value is 0
2. short    ➼   •value ranges from -(216)/2  to  (216)/2 - 1  
                      •takes 1 byte   
                      •default value is 0
3. int        ➼    •value ranges from -(232)/2 to (232)/2 - 1  
                       •takes 2 byte   
                       •deafault value is 0
4. float      ➼    •value ranges from (see Docs)  
                       •takes 4 byte  
                       •deafault value is 0.0f
5. long      ➼    •value ranges from -(264) /2 to (264)/2 - 1  
                       •takes 8 byte   
                       •default value is 0
6. double   ➼  •value ranges from (see Docs)  
                      •takes 8 byte   
                      •deafault value is 0.0d
7. char       ➼  •value ranges from 0 to 65535(216 - 1)  
                       •takes 2 byte   
                       •default value is
8. boolean ➼   •value can be true or false  
                       •size depend on JVM  
                       •default value is false

Quick Quiz: Write a Java program to add three numbers,

#### How to choose data types for our variables

In order to choose the data type we first need to find the type of data, we want to store. After that, we need to analyze the min & max value we might use.



**Literals in Java:**

A constant value that can be assigned to the variable is called a literal.

* 101 – Integer literal
* 10.1f – float literal
* 10.1 – double literal (default type for decimals)
* ‘A’ – character literal
* true – Boolean literal
* “Harry” – String literal

**Keywords:**

Words that are reserved and used by the Java compiler. They cannot be used as an Identifier.

package com.company;

public class CWH\_04\_literals {

public static void main(String[] args) {

byte age = 34;

int age2 = 56;

short age3 = 87;

long ageDino = 5666666666666L;

char ch = 'A';

float f1 = 5.6f;

double d1 = 4.66;

boolean a = true;

System.out.print(age);

String str = "Harry";

System.out.println(str);

}

}

### Java Tutorial: Getting User Input in Java

Reading data from the Keyboard

In order to read data from the keyboard, Java has a scanner class.

Scanner class has a lot of methods to read the data from the keyboard.

Scanner S = new Scanner(System.in); //(Read from the keyboard)

int a = S.nextInt(); //(Method to read from the keyboard)

#### 

package com.company;

import java.util.Scanner;

public class CWH\_05\_TakingInpu {

public static void main(String[] args) {

System.out.println("Taking Input From the User");

Scanner sc = new Scanner(System.in);

// System.out.println("Enter number 1");

// int a = sc.nextInt();

// float a = sc.nextFloat();

// System.out.println("Enter number 2");

// int b = sc.nextInt();

// float b = sc.nextFloat();

// int sum = a +b;

// float sum = a +b;

// System.out.println("The sum of these numbers is");

// System.out.println(sum);

// boolean b1 = sc.hasNextInt();

// System.out.println(b1);

// String str = sc.next();

String str = sc.nextLine();

**Practice Set:**

1. Write a program to sum three numbers in Java.
2. Write a program to calculate CGPA using marks of three subjects (out of 100)
3. Write a Java program that asks the user to enter his/her name and greets them with “Hello <name>, have a good day” text.
4. Write a Java program to convert Kilometers to miles.
5. Write a Java program to detect whether a number entered by the user is an integer or not.
6. package com.company;
7. import java.util.Scanner;
8. public class CWH\_Ch1\_PS {
9. public static void main(String[] args) {
10. // Question1
11. // int a = 4;
12. // int b = 17;
13. // int c =6;
14. // int sum = a + b+c;
15. // System.out.println(sum);
16. // Question2
17. // float subject1 = 45;
18. // float subject2 = 95;
19. // float subject3 = 48;
20. // float cgpa = (subject1 + subject2 +subject3)/30;
21. // System.out.println(cgpa);
22. // Question 3
23. // System.out.println("What is your name");
24. // Scanner sc = new Scanner(System.in);
25. // String name = sc.next();
26. // System.out.println("Hello " + name + " have a good day!");
27. // Question 5
28. System.out.println("Enter your number");
29. Scanner sc = new Scanner(System.in);
30. System.out.println(sc.hasNextInt());
31. }
32. }

**Java Tutorial: Operators, Types of Operators & Expressions in Java**

Operators are used to perform operations on variables and values.

7 + 11 = 18

(Here, 7 and 11 are operand and + is the operator and 18 is the result)

**Types of operators:**

* Arithmetic Operators => [+, -, \*, /, %, ++, --]
* Assignment Operators => [=, +=]
* Comparison Operators => [==, >=, <=]
* Logical Operators => [&&, ||, !]
* Bitwise Operators => [&, |] (operates bitwise)

Arithmetic operators cannot work with Booleans.

% operator can work on floats and doubles.

**Precedence of operators:**

The operators are applied and evaluated based on precedence. For example (+, -) has less precedence compared to (\*, /). Hence \* and / are evaluated first.

In case we like to change this order, we use parenthesis ().

package com.company;

public class CWH\_Ch2\_Operators {

public static void main(String[] args) {

// 1. Arithmetic Operators

int a = 4;

// int b = 6 % a; // Modulo Operator

// 4.8%1.1 --> Returns Decimal Remainder

// 2. Assignment Operators

int b = 9;

b \*= 3;

System.out.println(b);

// 3. Comparison Operators

// System.out.println(64<6);

// 4. Logical Operators

// System.out.println(64>5 && 64>98);

System.out.println(64>5 || 64>98);

// 5. Bitwise Operators

System.out.println(2&3);

// 10

// 11

// ----

// 10

}

}

### Associativity of Operators in Java:

#### Associativity

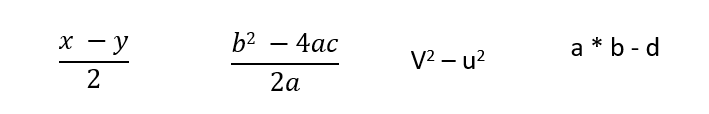
Associativity tells the direction of the execution of operators. It can either be left to right or vice versa.

/ \* -> L to R

+ - -> L to R

++, = -> R to L

Quick Quiz: How will you write the following expression in Java?



package com.company;

public class cwh\_09\_ch2\_op\_pre {

public static void main(String[] args) {

// Precedence & Associativity

//int a = 6\*5-34/2;

/\*

Highest precedence goes to \* and /. They are then evaluated on the basis

of left to right associativity

=30-34/2

=30-17

=13

\*/

//int b = 60/5-34\*2;

/\*

= 12-34\*2

=12-68

=-56

\*/

//System.out.println(a);

//System.out.println(b);

// Quick Quiz

int x =6;

int y = 1;

// int k = x \* y/2;

int b = 0;

int c = 0;

int a = 10;

int k = b\*b - (4\*a\*c)/(2\*a);

System.out.println(k);

}

}

**Data Type of Expressions & Increment/Decrement Operators:**

Resulting data type after arithmetic operation

* Result = byte + short -> integer
* Result = short + integer -> integer
* Result = long + float -> float
* Result = integer + float -> float
* Result = character + integer -> integer
* Result = character + short -> integer
* Result = long + double -> double
* Result = float + double -> double

**Increment and Decrement operators:**

* a++, ++a (Increment Operators)
* a--, --a (Decrement Operators)

These will operate on all data types except Booleans.

**Quick Quiz:**Try increment and decrement operators on a Java variable

* a++ -> first use the value and then increment
* ++a -> first increment the value then use it

**Quick Quiz:**What will be the value of the following expression(x).

1. int y=7;
2. int x = ++y\*8;
3. value of x?
4. char a = ‘B’;
5. a++; (a is not ‘C’)

package com.company;

public class cwh\_10\_resulting\_data\_type {

public static void main(String[] args) {

/\* byte x = 5;

int y = 6;

short z = 8;

int a = y + z;

float b = 6.54f + x;

System.out.println(b); \*/

// Increment and Decrement Operators

int i = 56;

// int b = i++; // first b is assigned i (56) then i is incremented

int j = 67;

int c = ++j; // first j is incremented then c is assigned j (68)

System.out.println(i++);

System.out.println(i);

System.out.println(++i);

System.out.println(i);

int y = 7;

System.out.println( ++y \*8);

char ch = 'a';

System.out.println(++ch);

}

}

**Write a program to calculate the percentage of a given student in the CBSE board exam. His marks from 5 subjects must be taken as input from the keyboard. (Marks are out of 100)**

package com.company;

import java.util.Scanner;

public class cwh\_11\_ex1\_sol {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter your Physics marks : ");

int physics = scan.nextInt();

System.out.println("Enter your English marks : ");

int English = scan.nextInt();

System.out.println("Enter your Chemistry marks : ");

int chemistry = scan.nextInt();

System.out.println("Enter your Mathematics marks : ");

int mathematics = scan.nextInt();

System.out.println("Enter your Computer Science marks : ");

int computer = scan.nextInt();

float percentage = ((physics + English + chemistry + mathematics + computer)/500.0f)\*100;

System.out.println("percentage : ");

System.out.println(percentage);

}

}

**Practice Set (Java Practice Questions):**

1. What will be the result of the following expression:

float a = 7/4 \* 9/2

1. Write a java program to encrypt a grade by adding 8 to it. Decrypt it to show the correct grade.
2. Use comparison operators to find out whether a given number is greater than the user entered number or not.
3. Write the following expression in a java program:

v2-u2/2as

1. Find the value of the following expression:

int x = 7

int a = 7\*49/7 + 35/7

Value of a?

Code Solution:

package com.company;

public class cwh\_12\_ps2\_pr01 {

public static void main(String[] args) {

float a = 7/4.0f \* 9/2.0f;

System.out.println(a);

}

}

package com.company;

public class cwh\_12\_ps2\_pr02 {

public static void main(String[] args) {

char grade = 'B';

grade = (char)(grade + 8);

System.out.println(grade);

// Decrypting the grade

grade = (char)(grade - 8);

System.out.println(grade);

}

}

package com.company;

import java.util.Scanner;

public class cwh\_12\_ps2\_pr\_03 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int a = sc.nextInt();

System.out.println(a>8);

System.out.println(7\*49/7+35/7);

}

}

### Introduction to Strings:

A string is a sequence of characters. A string is instantiated as follows:

String name;

name = new String(“Harry”);

String is a class but can be used as a data type.

Strings are immutable and cannot be changed.

String name = "Harry"

(Here, name is a reference, and “Harry” is an object.)

Different ways to print in Java

We can use the following ways to print in Java:

* System.out.print() // No newline at the end
* System.out.println() // Prints a new line at the end
* System.out.printf()
* System.out.format()

System.out.printf("%c",ch)

* %d for int
* %f for float
* %c for char
* %s for string

package com.company;

import java.util.Scanner;

public class cwh\_13\_strings {

public static void main(String[] args) {

// String name = new String("Harry");

// String name = "Harry";

// System.out.print("The name is: ");

// System.out.print(name);

int a = 6;

float b = 5.6454f;

System.out.printf("The value of a is %d and value of b is %8.2f", a, b);

//System.out.format("The value of a is %d and value of b is %f", a, b);

Scanner sc = new Scanner(System.in);

// String st = sc.next();

// String st = sc.nextLine();

// System.out.println(st);

}

}

**String Methods in Java:**

String Methods operate on Java Strings. They can be used to find the length of the string, convert to lowercase, etc.

Some of the commonly used String methods are:

String name = “Harry”;

(Indexes of the above string are as follows: 0-H, 1-a, 2-r, 3-r, 4-y)

1. length() – Returns the length of String name. (5 in this case)
2. toLowerCase() – Returns a new String which has all the lowercase characters from the String name.
3. toUpperCase() – Returns a new String which has all the uppercase characters from the String name.
4. trim() – Returns a new String after removing all the leading and trailing spaces from the original string.
5. substring(int start) – Returns a substring from start to the end. Substring(3) returns “ry”. [**Note**that index starts from 0]
6. substring(int start, int end) – Returns a substring from the start index to the end index. The start index is included and the end is excluded.
7. replace(‘r’, ‘p’) – Returns a new string after replacing r with p. Happy is returned in this case. (This method takes char as argument)
8. startsWith(“Ha”) – Returns true if name starts with string “Ha”. (True in this case)
9. endsWith(“ry”) – Returns true if name ends with string “ry”. (True in this case)
10. charAt(2) – Returns the character at a given index position. (r in this case)
11. indexOf(“s”) – Returns the index of the given string.

For e.g. name.indexOf(“ar”) returns 1 which is the first occurrence of ar in string “Harry”, -1 otherwise.

* indexOf(“s”, 3) – Returns the index of the given String starting from index 3(int). (-1 is returned in this case)
* lastIndexOf(“r”) – Returns the last index of the given string. (3 in this case)
* lastIndexOf(“r”,2) – Returns the last index of the given string before index 2.
* equals(“Harry”) – Returns true if the given string is equal to “Harry” false otherwise [Case sensitive]
* equalsIgnoreCase(“harry”) – Returns true if two strings are equal ignoring the case of characters.

**Escape Sequence Characters:**

The sequence of characters after backslash ‘\’ = Escape Sequence Characters

Escape Sequence Characters consist of more than one character but represent one character when used within the strings.

Examples: \n (newline), \t (tab), \’ (single quote), \\ (backslash), etc.

package com.company;

public class cwh\_14\_string\_methods {

public static void main(String[] args) {

String name = "Harry";

// System.out.println(name);

int value = name.length();

//System.out.println(value);

//String lstring = name.toLowerCase();

//System.out.println(lstring);

//String ustring = name.toUpperCase();

//System.out.println(ustring);

//String nonTrimmedString = " Harry ";

//System.out.println(nonTrimmedString);

//String trimmedString = nonTrimmedString.trim();

//System.out.println(trimmedString);

**Practice Questions on Strings:**

**Practice Set on Java Strings (Must Solve!)**

Today we will solve some of the best problems in Java related to strings! Here is the Chapter 3 – Practice Set

1. Write a Java program to convert a string to lowercase.
2. Write a Java program to replace spaces with underscores.
3. Write a Java program to fill in a letter template which looks like below:

// letter = “Dear <|name|>, Thanks a lot”

// Replace <|name|> with a string (some name)

1. Write a Java program to detect double and triple spaces in a string.
2. Write a program to format the following letter using escape sequence characters.

Letter = “Dear Harry, This Java Course is nice. Thanks”

package com.company;

public class cwh\_15\_ps3 {

public static void main(String[] args) {

// Problem 1

//String name = "Jack Parker";

//name = name.toLowerCase();

//System.out.println(name);

// Problem 2

//String text = "To My Friend";

//text = text.replace(" ", "\_");

//System.out.println(text);

// Problem 3

String letter = "Dear <|name|>, Thanks a lot!";

letter = letter.replace("<|name|>", "Sachin");

System.out.println(letter);

// Problem 4

String myString = "This string contains double and triple spaces";

System.out.println(myString.indexOf(" "));

System.out.println(myString.indexOf(" "));

// Problem 5

String myLetter = "Dear Harry,\n\tThis Java Course is Nice.\nThanks!";

System.out.println(myLetter);

}

}

**Java Conditionals: If-else Statement in Java:**

Sometimes we want to watch comedy videos on YouTube If the day is Sunday.

Sometimes, we order junk food if it is our friend’s birthday in the hostel.

You might want to buy an umbrella if it’s raining and you have the money.

You order the meal if aloo or your favorite bhindi is listed on the menu.

All these are decisions which depends on a certain condition being met. In java, we can execute instructions on a conditional being met.

**Decision-making instructions in Java:**

* If-Else Statement
* Switch Statement

**If-Else Statement:**

The syntax of an if-else statement in C looks like that of C++ and JavaScript. Java has a similar syntax too. It looks like:

/\* if (condition-to-be-checked) {

statements-if-condition-true;

}

else {

statements-if-condition-false;

} \*/

**Code Example:**

int a = 29;

if (a>18) {

System.out.println(“You can drive”);

}

### Relational and Logical Operators in Java:

#### Relational Operators in Java:

Relational operators are used to evaluate conditions (true or false) inside the if statements.

Some examples of relational operators are:

* == (equals)
* >= (greater than or equals to)
* > (greater than)
* < (less than)
* <= (less than or equals to)
* != (not equals)

**Note**: ‘=’ is used for assignment whereas ‘==’ is used for equality check.

The condition can be either true or false.

#### Logical Operators

&&, || and ! are the most commonly used logical operators in Java.

These are read as:

* && - AND
* || - OR
* ! – NOT

The above logical operators are used to provide logic to our Java programs.

##### **AND Operator:**

Evaluates to true if both the conditions are true.

* Y && Y = Y
* Y && N = N
* N && Y = N
* N && N = N

Convention: # Y – True and N - False

##### **OR Operator:**

Evaluates to true when at least one of the conditions is true.

* Y || Y = Y
* Y || N = Y
* N || Y = Y
* N || N = N

Convention: # Y – True and N - False

##### **NOT Operator:**

Negates the given logic (true becomes false and vice-versa)

* !Y = N
* !N = Y

#### else-if clause:

Instead of using multiple if statements, we can also use else if along with if thus forming an if-else-if-else ladder.

Using such kind of logic reduces indents. Last else is executed only if all the conditions fail.

/\* if (condition1) {

//Statements;

else if {

// Statements;

}

else {

//Statements

} \*/

package com.company;

public class cwh\_17\_logical {

public static void main(String[] args) {

System.out.println("For Logical AND...");

boolean a = true;

boolean b = false;

// if (a && b){

// System.out.println("Y");

// }

// else{

// System.out.println("N");

// }

System.out.println("For Logical OR...");

// if (a || b){

// System.out.println("Y");

// }

// else{

// System.out.println("N");

// }

System.out.println("For Logical NOT");

System.out.print("Not(a) is ");

System.out.println(!a);

System.out.print("Not(b) is ");

System.out.println(!b);

}

}

**Switch Case Statements in Java:**

**Switch Case-Control Instruction:**

**Switch-Case** is used when we have to make a choice between the number of alternatives for a given variable.

/\* Switch(var) {

Case C1:

//Code;

break;

Case C2:

//Code;

break;

Case C3:

//Code

break;

default:

//Code

} \*/

Var can be an integer, character, or string in Java.

* A switch can occur within another but in practice, this is rarely done.

package com.company;

import java.util.Scanner;

public class cwh\_18\_elseif {

public static void main(String[] args) {

String var = "Saurabh";

switch (var) {

case "Shubham" -> {

System.out.println("You are going to become an Adult!");

System.out.println("You are going to become an Adult!");

System.out.println("You are going to become an Adult!");

}

case "Saurabh" -> System.out.println("You are going to join a Job!");

case "Vishaka" -> System.out.println("You are going to get retired!");

default -> System.out.println("Enjoy Your life!");

}

System.out.println("Thanks for using my Java Code!");

/\*

int age;

System.out.println("Enter Your Age");

Scanner sc = new Scanner(System.in);

age = sc.nextInt();

if (age>56){

System.out.println("You are experienced!");

}

else if(age>46){

System.out.println("You are semi-experienced!");

}

else if(age>36){

System.out.println("You are semi-semi-experienced!");

}

else{

System.out.println("You are not experienced");

}

if(age>2){

System.out.println("You are not a baby!");

}

\*/

}

**Practice Questions on Conditionals & Switch Case:**

**Chapter 4– Practice Set:**

1. What will be the output of this program:

int a = 10;

if (a=11)

System.out.println(“I am 11”);

else

System.out.println(“I am not 11”);

1. Write a program to find out whether a student is pass or fail; if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.
2. Calculate income tax paid by an employee to the government as per the slabs mentioned below:

|  |  |
| --- | --- |
| **Income Slab** | **Tax** |
| 2.5L – 5.0L | 5% |
| 5.0L – 10.0L | 20% |
| Above 10.0L | 30% |

Note that there is not tax below 2.5L. Take the input amount as input from the user.

1. Write a Java program to find out the day of the week given the number [1 for Monday, 2 for Tuesday … and so on!]
2. Write a Java program to find whether a year entered by the user is a leap year or not.
3. Write a program to find out the type of website from the URL:

* .com – commercial website
* .org – organization website
* .in – Indian website

package com.company;

import java.util.Scanner;

import java.util.Random;

public class cwh\_19\_ch4\_ps {

public static void main(String[] args) {

// Question 1:

// int a = 11;

// if(a=11){

//

// }

// Question 2

// byte m1, m2, m3;

// Scanner sc = new Scanner(System.in);

// System.out.println("Enter your marks in Physics");

// m1 = sc.nextByte();

//

// System.out.println("Enter your marks in Chemistry");

// m2= sc.nextByte();

//

// System.out.println("Enter your marks in Mathematics");

// m3 = sc.nextByte();

// float avg = (m1+m2+m3)/3.0f;

// System.out.println("Your Overall percentage is: " + avg);

// if(avg>=40 && m1>=33 && m2>=33 && m3>=33){

// System.out.println("Congratulations, You have been promoted");

// }

// else{

// System.out.println("Sorry, You have not been promoted! Please try again.");

// }

// Question 3

// Scanner sc = new Scanner(System.in);

// System.out.println("Enter your income in Lakhs per annum");

// float tax = 0;

// float income = sc.nextFloat();

// if(income<=2.5){

// tax = tax + 0;

// }

// else if(income>2.5f && income <= 5f){

// tax = tax + 0.05f \* (income - 2.5f);

// }

// else if(income>5f && income <= 10.0f){

// tax = tax + 0.05f \* (5.0f - 2.5f);

// tax = tax + 0.2f \* (income - 5f);

// }

// else if(income>10.0f){

// tax = tax + 0.05f \* (5.0f - 2.5f);

// tax = tax + 0.2f \* (10.0f - 5f);

// tax = tax + 0.3f \* (income - 10.0f);

// }

//

// System.out.println("The total tax paid by the employee is: " + tax);

// Question 4:

// Scanner sc = new Scanner(System.in);

// int day = sc.nextInt();

//

// switch (day){

// case 1 -> System.out.println("Monday");

// case 2 -> System.out.println("Tuesday");

// case 3 -> System.out.println("Wednesday");

// case 4 -> System.out.println("Thursday");

// case 5 -> System.out.println("Friday");

// case 6 -> System.out.println("Saturday");

// case 7 -> System.out.println("Sunday");

// }

// Question 6

// Scanner sc = new Scanner(System.in);

// String website = sc.next();

// if(website.endsWith(".org")){

// System.out.println("This is an organizational website");

// }

// else if(website.endsWith(".com")){

// System.out.println("This is a Commercial website");

// }

// else if(website.endsWith(".in")){

// System.out.println("This is an Indian website");

// }

Random r = new Random();

int a = r.nextInt();

System.out.println(a);

}

}

**While Loops in Java:**

Sometimes we want our program to execute a few sets of instructions over and over again.

For example: print 1 to 1000, print multiplication table of 7, etc.

Loops make it easy for us to tell the computer that a given set of instructions need to be executed repeatedly.

Types of Loops

Primarily, there are three types of loops in Java:

1. While loop
2. do-while loop
3. for loop

We will look into these, one by one.

While loops:

/\*

while (Boolean condition)

{

// Statements -> This keeps executing as long as the condition is true.

}

\*/

If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

**Quick Quiz:**Write a program to print natural numbers from 100 to 200.

package com.company;

public class cwh\_21\_ch5\_loops {

public static void main(String[] args) {

System.out.println(1);

System.out.println(2);

System.out.println(3);

System.out.println("Using Loops:");

int i = 100;

while(i<=200){

System.out.println(i);

i++;

}

System.out.println("Finish Running While Loop!");

// while(true){

// System.out.println("I am an infinite while loop!");

// }

}

}

### The do-while loop in Java:

#### do-while loop:

This loop is similar to a while loop except for the fact that it is guaranteed to execute at least once.

/\* do {

//code

} while (condition); //Note this semicolon \*/

while – checks the condition & executes the code

do-while – executes the code & then checks the condition

Quick Quiz: Write a program to print first n natural numbers using a do-while loop.

package com.company;

public class cwh\_22\_ch4\_do\_while {

public static void main(String[] args) {

// int a = 0;

// while(a<5){

// System.out.println(a);

// a++;

// }

int b = 10;

do {

System.out.println(b);

b++;

}while(b<5);

int c = 1;

do{

System.out.println(c);

c++;

}while(c<=45);

}

}

### The for Loop in Java:

#### For loop:

The syntax of a for loop looks like this:

/\* for (initialize; check\_bool\_expression; update){

//code;

} \*/

A for loop is usually used to execute a piece of code for a specific number of times.

Quick Quiz: Write a program to print first n odd numbers using a for loop.

##### **Decrementing for loop:**

for (i=7; i!=0; i--){

System.out.println(i);

}

This for loop keeps running until i becomes 0.

**Quick Quiz:** Write a program to print first n natural numbers in reverse order.

package com.company;

public class cwh\_23\_for\_loop {

public static void main(String[] args) {

// for (int i=1; i<=10; i++){

// System.out.println(i);

// }

// 2i = Even Numbers = 0, 2, 4, 6, 8

// 2i+1 = Odd Numbers = 1, 3, 5, 7, 9

//int n = 3;

//for (int i =0; i<n; i++){

// System.out.println(2\*i+1);

//}

for(int i=5; i!=0; i--){

System.out.println(i);

}

}

}

**break and continue in Java:**

**break statement:**

The break statement is used to exit the loop irrespective of whether the condition is true or false.

Whenever a ‘break’ is encountered inside the loop, the control is sent outside the loop.

**continue statement:**

The continue statement is used to immediately move to the next iteration of the loop. The control is taken to the next iteration thus skipping everything below ‘continue’ inside the loop for that iteration.

**In a Nut Shell …**

1. break statement completely exits the loop
2. continue statement skips the particular iteration of the loop.

package com.company;

public class cwh\_24\_break\_and\_continue {

public static void main(String[] args) {

// Break and continue using loops!

// for (int i=0;i<50;i++){

// System.out.println(i);

// System.out.println("Java is great");

// if(i==2){

// System.out.println("Ending the loop");

// break;

// }

// }

// int i=0;

// do{

// System.out.println(i);

// System.out.println("Java is great");

// if(i==2){

// System.out.println("Ending the loop");

// break;

// }

// i++;

// }while(i<5);

// System.out.println("Loop ends here");

// for(int i=0;i<50;i++){

// if(i==2){

// System.out.println("Ending the loop");

// continue;

// }

// System.out.println(i);

// System.out.println("Java is great");

// }

int i=0;

do{

i++;

if(i==2){

System.out.println("Ending the loop");

continue;

}

System.out.println(i);

System.out.println("Java is great");

}while(i<5);

System.out.println("Loop ends here");

}

}

**Practice Questions on Loops:**

1. Write a program to print the following pattern

\*\*\*\*

\*\*\*

\*\*

\*

1. Write a program to sum first n even numbers using a while loop.
2. Write a program to print the multiplication table of a given number n.
3. Write a program to print a multiplication table of 10 in reverse order.
4. Write a program to find factorial of a given number using for loops.
5. Repeat problem 5 using a while loop.
6. Repeat problem 1 using for/while loop.
7. What can be done using one type of loop can also be done using the other two types of loops - True or False.
8. Write a program to calculate the sum of the numbers occurring in the multiplication table of 8.

* A do-while loop is executed:

       At least once

       At least twice

      At most once

* Repeat problem 2 using for loop.

package com.company;

public class cwh\_25\_practice\_set\_5 {

public static void main(String[] args) {

// Practice Problem 1

// int n = 4;

// for (int i=n; i>0; i--){

// for(int j=0;j<i;j++){

// System.out.print("\*");

// }

// System.out.print("\n");

// }

// Practice Problem 2

// int sum=0;

// int n=4;

// for(int i=0;i<n;i++){

// sum = sum + (2\*i);

// }

// System.out.print("Sum of even numbers is: ");

// System.out.println(sum);

// First 4 even numbers are - 0 2 4 6

// Practice Problem 3

// int n = 5;

// //for(int i=0; i<10; i++) - Goes from i=0 to i=9

// for(int i=1;i<=10;i++){

// System.out.printf("%d X %d = %d\n", n, i, n\*i);

// }

// Practice Problem 4

// int n = 10;

// //for(int i=0; i<10; i++) - Goes from i=0 to i=9

// for(int i=10;i>=1;i--){

// System.out.printf("%d X %d = %d\n", n, i, n\*i);

// }

// Practice Problem 6

// int n = 5;

// // What is factorial n = n \* n-1 \* n-2 ..... 1

// // 5! = 5\*4\*3\*2\*1 = 120

// int i = 1;

// int factorial = 1;

// while(i<=n){

// factorial \*= i;

// i++;

// }

// System.out.println(factorial);

// Practice Problem 9

// int n = 8;

// int sum = 0;

// //for(int i=0; i<10; i++) - Goes from i=0 to i=9

// for(int i=1;i<=10;i++){

// sum += n\*i;

// }

// System.out.println(sum);

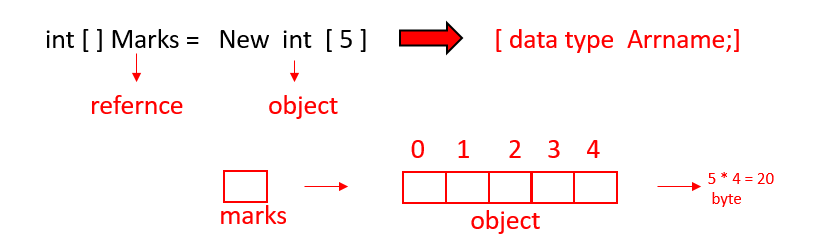
}

}

**Introduction to Arrays:**

An **array** is a collection of similar types of data.

Use Case: Storing marks of 5 students

****

**Accessing Array Elements**

Array elements can be accessed as follows,

/\* marks[0] = 100 //Note that index starts from 0

marks[1] = 70

.

.

marks[4] = 98 \*/

So in a nut shell, this is how array works:

1. int[] marks; //Declaration!

marks = new int[5];           //Memory allocation!

1. int[] marks = new int[5]; //Declaration + Memory allocation!
2. int[] marks = {100,70,80,71,98} // Declare + Initialize!

Array indices start from 0 and go till (n-1) where n is the size of the array.

**Array length**

Arrays have a length property which gives the length of the array.

marks.length //Gives 5 if marks is a reference to an array with 5 elements

**Displaying an Array**

An array can be displayed using a for loop:

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

{

Sout(marks[i]); //Array Traversal

}

**Quick Quiz:**Write a Java program to print the elements of an array in reverse order.

package com.company;

public class cwh\_26\_arrays {

public static void main(String[] args) {

/\* Classroom of 500 students - You have to store marks of these 500 students

You have 2 options:

1. Create 500 variables

2. Use Arrays (recommended)

\*/

// There are three main ways to create an array in Java

// 1. Declaration and memory allocation

// int [] marks = new int[5];

// 2. Declaration and then memory allocation

// int [] marks;

// marks = new int[5];

// Initialization

// marks[0] = 100;

// marks[1] = 60;

// marks[2] = 70;

// marks[3] = 90;

// marks[4] = 86;

// 3. Declaration, memory allocation and initialization together

int [] marks = {98, 45, 79, 99, 80};

// marks[5] = 96; - throws an error

System.out.println(marks[4]);

}

}

### For Each Loop in Java:

Array elements can also be traversed as follows:

/\* for (int element:Arr) {

Sout(element); //Prints all the elements

} \*/

package com.company;

public class cwh\_27\_arrays {

public static void main(String[] args) {

/\*

float [] marks = {98.5f, 45.5f, 79.5f, 99.5f, 80.5f};

String [] students ={"Harry", "Rohan", "Shubham", "Lovish"};

System.out.println(students.length);

System.out.println(students[2]);

\*/

int [] marks = {98, 45, 79, 99, 80};

// System.out.println(marks.length);

// Displaying the Array (Naive way)

System.out.println("Printing using Naive way");

System.out.println(marks[0]);

System.out.println(marks[1]);

System.out.println(marks[2]);

System.out.println(marks[3]);

System.out.println(marks[4]);

// Displaying the Array (for loop)

System.out.println("Printing using for loop");

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

System.out.println(marks[i]);

}

// Quick Quiz: Displaying the Array in Reverse order (for loop)

System.out.println("Printing using for loop in reverse order");

for(int i=marks.length -1;i>=0;i--){

System.out.println(marks[i]);

}

// Quick Quiz: Displaying the Array (for-each loop)

System.out.println("Printing using for-each loop");

for(int element: marks){

System.out.println(element);

}

}

}

### Multidimensional Arrays in Java:

Multidimensional Arrays are an Array of Arrays. Each elements of an M-D array is an array itself. Marks in the previous example was a 1-D array.

#### Multidimensional 2-D Array:

A 2-D array can be created as follows:

int [][] flats = new int[2][3] //A 2-D array of 2 rows + 3 columns

We can add elements to this array as follows

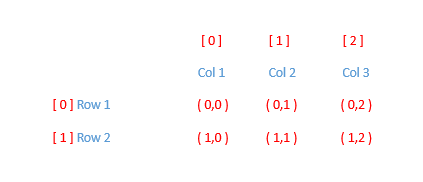
flats[0][0] = 100

flats[0][1] = 101

flats[0][2] = 102

// … & so on!

This 2-D array can be visualized as follows:

****

Similarly, a **3-D array** can be created as follows:

String[][][] arr = new String [2][3][4]

package com.company;

public class cwh\_28\_multi\_dim\_arrays {

public static void main(String[] args) {

int [] marks; // A 1-D Array

int [][] flats; // A 2-D Array

flats = new int [2][3];

flats[0][0] = 101;

flats[0][1] = 102;

flats[0][2] = 103;

flats[1][0] = 201;

flats[1][1] = 202;

flats[1][2] = 203;

// Displaying the 2-D Array (for loop)

System.out.println("Printing a 2-D array using for loop");

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

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

System.out.print(flats[i][j]);

System.out.print(" ");

}

System.out.println("");

}

}

}

**Practice Questions on Arrays in Java:**

1. Create an array of 5 floats and calculate their sum.
2. Write a program to find out whether a given integer is present in an array or not.
3. Calculate the average marks from an array containing marks of all students in physics using a for-each loop.
4. Create a Java program to add two matrices of size 2x3.
5. Write a Java program to reverse an array.
6. Write a Java program to find the maximum element in an array.
7. Write a Java program to find the maximum element in a Java array.
8. Write a Java program to find whether an array is sorted or not.

**Solution with code:**

package com.company;

public class cwh\_29\_Practice\_Set\_6 {

public static void main(String[] args) {

// Practice Problem 1

/\* float [] marks = {45.7f, 67.8f, 63.4f, 99.2f, 100.0f};

float sum = 0;

for(float element:marks){

sum = sum + element;

}

System.out.println("The value of sum is " + sum);

// Practice Problem 2

float [] marks = {45.7f, 67.8f, 63.4f, 99.2f, 100.0f};

float num = 45.57f;

boolean isInArray = false;

for(float element:marks){

if(num==element){

isInArray = true;

break;

}

}

if(isInArray){

System.out.println("The value is present in the array");

}

else{

System.out.println("The value is not present in the array");

}

// Practice Problem 3

float [] marks = {45.7f, 67.8f, 63.4f, 99.2f, 100.0f};

float sum = 0;

for(float element:marks){

sum = sum + element;

}

System.out.println("The value of average marks is " + sum/marks.length);

// Practice Problem 4

int [][] mat1 = {{1, 2, 3},

{4, 5, 6}};

int [][] mat2 = {{2, 6, 13},

{3, 7, 1}};

int [][] result = {{0, 0, 0},

{0, 0, 0}};

for (int i=0;i<mat1.length;i++){ // row number of times

for (int j=0;j<mat1[i].length;j++) { // column number of time

System.out.format(" Setting value for i=%d and j=%d\n", i, j);

result[i][j] = mat1[i][j] + mat2[i][j];

}

}

// Printing the elements of a 2-D Array

for (int i=0;i<mat1.length;i++){ // row number of times

for (int j=0;j<mat1[i].length;j++) { // column number of time

System.out.print(result[i][j] + " ");

result[i][j] = mat1[i][j] + mat2[i][j];

}

System.out.println(""); // Prints a new line

}

// Practice Problem 5

int [] arr = {1, 21, 3, 4, 5, 34, 67};

int l = arr.length;

int n = Math.floorDiv(l, 2);

int temp;

for(int i=0; i<n; i++){

// Swap a[i] and a[l-1-i]

// a b temp

// |4| |3| ||

temp = arr[i];

arr[i] = arr[l-i-1];

arr[l-i-1] = temp;

}

for(int element: arr){

System.out.print(element + " ");

}

// Practice Problem 6

int [] arr = {1, 2100, 3, 455, 5, 34, 67};

int max = Integer.MIN\_VALUE;

for(int e: arr){

if(e>max){

max = e;

}

}

System.out.println("the value of the maximum element in this array is: "+ max);

// Practice Problem 6

System.out.println(Integer.MIN\_VALUE);

System.out.println(Integer.MAX\_VALUE);

\*/

// Practice Problem 7

boolean isSorted = true;

int [] arr = {1, 12, 3, 4, 5, 34, 67};

for(int i=0;i<arr.length-1;i++){

if(arr[i] > arr[i+1]){

isSorted = false;

break;

}

}

if(isSorted){

System.out.println("The Array is sorted");

}

else{

System.out.println("The Array is not sorted");

}

}

}

### Methods in Java:

Sometimes our program grows in size and we want to separate the logic of the main method to other methods.

For instance – if we are calculating the average of a number pair 5 times, we can use methods to avoid repeating the logic. [DRY – Don’t Repeat Yourself]

#### Syntax of a Method:

A method is a function written inside a class.

Since Java is an object-oriented language, we need to write the method inside some class.

dataType name() {

//Method body

}

The following method returns the sum of two numbers

int mySum(int a, int b) {

int c = a+b;

return c; //Return value

}

In the above method/function int before mySum is the return type of the function.

#### Calling a Method:

A method can be called by creating an object of the class in which the method exists followed by the method call:

Calc obj = new Calc(); //Object Creation

obj.mySum(a , b); //Method call upon an object

The values from the method call (a and b) are copied to the a and b of the function mySum. Thus even if we modify the values a and b inside the method, the values in the main method will not change.

#### Void return type:

When we don’t want our method to return anything, we use void as the return type.

#### Static keyword:

The static keyword is used to associate a method of a given class with the class rather than the object. The static method in a class is shared by all the objects.

#### Process of method invocation in Java:

Consider the method Sum:

int Sum(int a, int b)

{

return a+b;

}

The method is called like this:

Calc obj = new Calc();

c = obj.Sum(2,3)

The values 2 and 3 are copied to a and b and then a+b = 2+3 = 5 is returned in C which is an integer.

Note: In the case of Arrays, the reference is passed. The same is the case for object passing to methods.

package com.company;

public class cwh\_31\_methods {

static int logic(int x, int y){

int z;

if(x>y){

z = x+y;

}

else {

z = (x +y) \* 5;

}

x = 566;

return z;

}

public static void main(String[] args) {

int a = 5;

int b = 7;

int c;

// Method invocation using Object creation

//cwh\_31\_methods obj = new cwh\_31\_methods();

//c = obj.logic(a, b);

c = logic(a, b);

System.out.println(a + " "+ b);

int a1 = 2;

int b1 = 1;

int c1;

c1 = logic(a1, b1);

System.out.println(c);

System.out.println(c1);

}

}

### Method Overloading in Java:

Two or more methods can have the same name but different parameters. Such methods are called Overloaded methods.

void foo()

void foo(int a) //Overloaded function foo

int foo(int a, int b)

Method overloading cannot be performed by changing the return type of methods.

package com.company;

public class cwh\_32\_method\_overloading {

static void foo(){

System.out.println("Good Morning bro!");

}

static void foo(int a){

System.out.println("Good morning " + a + " bro!");

}

static void foo(int a, int b){

System.out.println("Good morning " + a + " bro!");

System.out.println("Good morning " + b + " bro!");

}

static void foo(int a, int b, int c){

System.out.println("Good morning " + a + " bro!");

System.out.println("Good morning " + b + " bro!");

}

static void change(int a){

a = 98;

}

static void change2(int [] arr){

arr[0] = 98;

}

static void tellJoke(){

System.out.println("I invented a new word!\n" +

"Plagiarism!");

}

public static void main(String[] args) {

// tellJoke();

// Case 1: Changing the Integer

//int x = 45;

//change(x);

//System.out.println("The value of x after running change is: " + x);

// Case 1: Changing the Array

// int [] marks = {52, 73, 77, 89, 98, 94};

// change2(marks);

// System.out.println("The value of x after running change is: " + marks[0]);

// Method Overloading

foo();

foo(3000);

foo(3000, 4000);

// Arguments are actual!

}

}

**Variable Arguments (VarArgs) in Java:**

A function with Vararg can be created in Java using the following syntax:

/\*

public static void foo(int … arr)

{

// arr is available here as int[] arr

}

\*/

foo can be called with zero or more arguments like this:

* foo(7)
* foo(7,8,9)
* foo(1,2,7,8,9)

We can also create a function bar like this

public static void bar(int a, int arr)

{

//Code

}

bar can be called as bar(1), bar(1,2), bar(1,7,9,11), etc.

### Recursion in Java:

A function in Java can call itself. Such calling of function by itself is called recursion.

Example: Factorial of a number

//factorial(n) = n\*factorial(n-1) [n >= 1]

Quick Quiz: Write a program to calculate (recursion must be used) factorial of a number in Java?

package com.company;

public class cwh\_34\_recursion {

// factorial(0) = 1

// factorial(n) = n \* n-1 \*....1

// factorial(5) = 5 \* 4 \* 3 \* 2 \* 1 = 120

// factorial(n) = n \* factorial(n-1)

static int factorial(int n){

if(n==0 || n==1){

return 1;

}

else{

return n \* factorial(n-1);

}

}

static int factorial\_iterative(int n){

if(n==0 || n==1){

return 1;

}

else{

int product = 1;

for (int i=1;i<=n;i++){ // 1 to n

product \*= i;

}

return product;

}

}

public static void main(String[] args) {

int x = 0;

System.out.println("The value of factorial x is: " + factorial(x));

System.out.println("The value of factorial x is: " + factorial\_iterative(x));

}

}

**Practice Questions on Java Methods:**

1. Write a Java method to print the multiplication table of a number n.
2. Write a program using functions to print the following pattern:

      \*

     \*\*

    \*\*\*

   \*\*\*\*

1. Write a recursive function to calculate the sum of first n natural numbers.
2. Write a function to print the following pattern:

    \*\*\*\*

    \*\*\*

    \*\*

    \*

1. Write a function to print the nthterm of the Fibonacci series using recursion.
2. Write a function to find the average of a set of numbers passed as arguments.
3. Repeat problem 4 using Recursion.
4. Repeat problem 2 using Recursion.
5. Write a function to convert Celsius temperature into Fahrenheit.
6. Repeat problem 3 using an iterative approach.

package com.company;

public class cwh\_35\_practice\_set\_on\_methods {

static void multiplication(int n) {

for (int i = 1; i <= 10; i++) {

System.out.format("%d X %d = %d\n", n, i, n \* i);

}

}

static void pattern1(int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < i + 1; j++) {

System.out.print("\*");

}

System.out.println();

}

}

static void pattern1\_rec(int n) {

if (n > 0) {

pattern1\_rec(n - 1);

for (int i = 0; i < n; i++) {

System.out.print("\*");

}

System.out.println();

}

}

// pattern1\_rec(3)

// pattern1\_rec(2) + 3 times star and new line

// pattern1\_rec(1) + 2 times star and new line + 3 times star and new line

// pattern1\_rec(0) + 1 times star and new line + 2 times star and new line + 3 times star and new line

// sum(n) = 1 + 2 + 3... + n

// sum(n) = 1 + 2 + 3... + n-1 + n

// sum(n) = sum(n-1) + n

// sum(3) = 3 + sum(2)

// sum(3) = 3 + 2 + sum(1)

// sum(3) = 3 + 2 + 1

static int sumRec(int n) {

// Base condition

if (n == 1) {

return 1;

}

return n + sumRec(n - 1);

}

static int fib(int n) {

/\* if(n==1){

return 0;

}

else if(n==2){

return 1;

} \*/

if (n == 1 || n == 2) {

return n - 1;

} else {

return fib(n - 1) + fib(n - 2);

}

}

public static void main(String[] args) {

// Problem 1

// multiplication(7);

// Problem 2

// pattern1(9);

// Problem 3

// int c = sumRec(4);

// System.out.println(c);

// Problem 4

// fibonacci series - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

// int result = fib(7);

// System.out.println(result);

// Problem 8

pattern1(9);

}

}

### Introduction to Object Oriented Programming:

Object-Oriented Programming tries to map code instructions with real-world making the code short and easier to understand.

#### What Is Object-Oriented Programming?

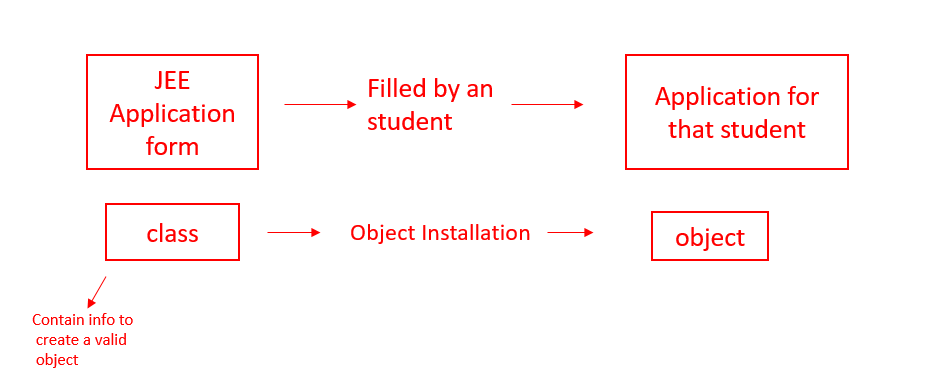
Solving a problem by creating objects is one of the most popular approaches in programming. This is called Object-Oriented Programming.

#### What is DRY?

DRY stands for – Do no-repeat yourself (It focuses on code reusability)

##### **Class:**

A class is a blueprint for creating objects.

****

##### **Object:**

An object is an instantiation of a class. When a class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

#### How to model a problem in OOPs

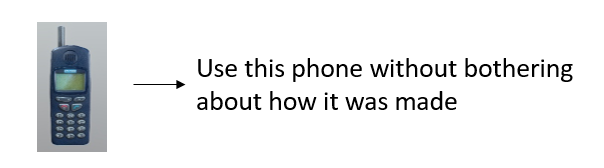
We identify the following:

* Noun              - Class                        - Employee
* Adjective        - Attributes                  - name, age, salary
* Verb               - Methods                    - getSalary(), increment()

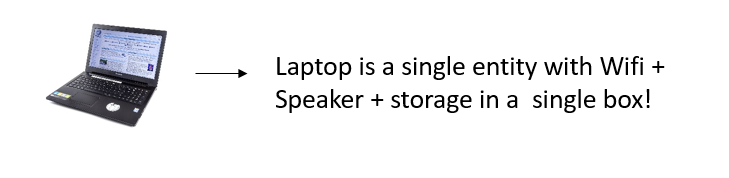
**Basic Terminologies in Object Oriented Programming:**

**OOPs Terminology:**

1. **Abstraction –**Hiding internal details [show only essential info!]

****

1. **Encapsulation –**The act of putting various components together (in a capsule).

****

In Java, encapsulation simply means that sensitive data can be hidden from the users.

1. **Inheritance –**The act of deriving new things from existing things.

* Rickshaw      →        E-Rickshaw
* Phone           →        Smart Phone

1. **Polymorphism –**One entity many forms

* Smartphone → Phone                   Smartphone → Calculator

C**reating Our Own Java Class:**

Writing a Custom Class -

We can write a custom class as follows:

public class Employee {

int id; // Attribute 1

String name; // Attribute 2

}

* Any real-world object        = Properties + Behavior
* Object in OOPs                 = Attributes + Methods

**A Class with Methods:**

We can add methods to our class Employee as follows:

/\*

public class Employee {

public int id;

public String name;

public int getSalary(){

//code

}

public void getDetails(){

//code

}

};

\*/

package com.company;

class Employee{

int id;

int salary;

String name;

public void printDetails(){

System.out.println("My id is " + id);

System.out.println("and my name is "+ name);

}

public int getSalary(){

return salary;

}

}

public class cwh\_38\_custom\_class {

public static void main(String[] args) {

System.out.println("This is our custom class");

Employee harry = new Employee(); // Instantiating a new Employee Object

Employee john = new Employee(); // Instantiating a new Employee Object

// Setting Attributes for Harry

harry.id = 12;

harry.salary = 34;

harry.name = "CodeWithHarry";

// Setting Attributes for John

john.id = 17;

john.salary = 12;

john.name = "John Khandelwal";

// Printing the Attributes

harry.printDetails();

john.printDetails();

int salary = john.getSalary();

System.out.println(salary);

// System.out.println(harry.id);

// System.out.println(harry.name);

}

}

**Basic Questions on Object Oriented Programming:**

1. Create a class Employee with the following properties and methods:

* Salary (property) (int)
* getSalary (method returning int)
* name (property) (String)
* getName (method returning String)
* setName (method changing name)

1. Create a class cellphone with methods to print “ringing…”, “vibrating…”, etc.
2. Create a class Square with a method to initialize its side, calculating area, perimeter etc.
3. Create a class Rectangle & problem 3.
4. Create a class TommyVecetti for Rockstar Games capable of hitting (print hitting…), running, firing, etc.
5. Repeat problem 4 for a circle.

**Code with solution :**

package com.company;

class Employee{

int salary;

String name;

public int getSalary(){

return salary;

}

public String getName(){

return name;

}

public void setName(String n){

name = n;

}

}

class CellPhone{

public void ring(){

System.out.println("Ringing...");

}

public void vibrate(){

System.out.println("Vibrating...");

}

public void callFriend(){

System.out.println("Calling Mukul...");

}

}

class Square{

int side;

public int area(){

return side\*side;

}

public int perimeter(){

return 4\*side;

}

}

class Tommy{

public void hit(){

System.out.println("Hitting the enemy");

}

public void run(){

System.out.println("Running from the enemy");

}

public void fire(){

System.out.println("Firing on the enemy");

}

}

public class cwh\_39\_ch8ps {

public static void main(String[] args) {

/\*

// Problem 1

Employee harry = new Employee();

harry.setName("CodeWithHarry");

harry.salary = 233;

System.out.println(harry.getSalary());

System.out.println(harry.getName());

// Problem 2

CellPhone asus = new CellPhone();

asus.callFriend();

asus.vibrate();

//asus.ring();

// Problem 3

Square sq = new Square();

sq.side = 3;

System.out.println(sq.area());

System.out.println(sq.perimeter());

\*/

// Problem 5

Tommy player1 = new Tommy();

player1.fire();

player1.run();

player1.hit();

}

}

**Access modifiers, getters & setters in Java:**

**Access Modifiers:**

Access Modifiers specifies where a property/method is accessible there are four types of access modifier in java:

1. private
2. default
3. protected
4. public

**Getters and Setters:**

* Getter ➼   Returns the value  [accessors]
* setter ➼    Sets / updates the value  [mutators]

Source code as described in the video:

package com.company;

class MyEmployee{

private int id;

private String name;

public String getName(){

return name;

}

public void setName(String n){

this.name = n;

}

public void setId(int i){

this.id = i;

}

public int getId(){

return id;

}

}

public class cwh\_40\_ch9 {

public static void main(String[] args) {

MyEmployee harry = new MyEmployee();

// harry.id = 45;

// harry.name = "CodeWithHarry"; --> Throws an error due to private access modifier

harry.setName("CodeWithHarry");

System.out.println(harry.getName());

harry.setId(234);

System.out.println(harry.getId());

}

}

### Constructors in Java:

#### Constructors in java:

A number function used to initialy an object while creating it.

Employee harry = new Employee ();

harry.setName ("Harry bhai ")

In order to write our own constructor , we define a method with name same as class name

public Employee () {

name = " your name " ) ;

}

#### Constructor Overloading in Java :

constructors can be overloaded just like other method in java. We can overload the Employe constructor like below:

public Employee (String n)

name = n;

}

Note:

1. Constructors can take parameters without being overloaded
2. there can be more than two overloaded constructors

Quick quiz: overloaded the employes constructor to initialize the salary to Rs 10,000

package com.company;

class MyMainEmployee{

private int id;

private String name;

public MyMainEmployee(){

id = 0;

name = "Your-Name-Here";

}

public MyMainEmployee(String myName, int myId){

id = myId;

name = myName;

}

public MyMainEmployee(String myName){

id = 1;

name = myName;

}

public String getName(){

return name;

}

public void setName(String n){

this.name = n;

}

public void setId(int i){

this.id = i;

}

public int getId(){

return id;

}

}

public class cwh\_42\_constructors {

public static void main(String[] args) {

//MyMainEmployee harry = new MyMainEmployee("ProgrammingWithHarry", 12);

MyMainEmployee harry = new MyMainEmployee();

//harry.setName("CodeWithHarry");

//harry.setId(34);

System.out.println(harry.getId());

System.out.println(harry.getName());

}

}

### JAVA EXERCISE : Guess the Number (OOPs Edition)

Create a class Game, which allows a user to play "Guess the Number" game once.

* Game should have the following methods:
* Constructor to generate the random number
* takeUserInput() to take a user input of number
* isCorrectNumber() to detect whether the number entered by the user is true
* getter and setter for noOfGuesses

Use properties such as noOfGuesses(int), etc to get this task done!

**Solution:**

package com.company;

import java.util.Random;

import java.util.Scanner;

class Game{

public int number;

public int inputNumber;

public int noOfGuesses = 0;

public int getNoOfGuesses() {

return noOfGuesses;

}

public void setNoOfGuesses(int noOfGuesses) {

this.noOfGuesses = noOfGuesses;

}

Game(){

Random rand = new Random();

this.number = rand.nextInt(100);

}

void takeUserInput(){

System.out.println("Guess the number");

Scanner sc = new Scanner(System.in);

inputNumber = sc.nextInt();

}

boolean isCorrectNumber(){

noOfGuesses++;

if (inputNumber==number){

System.out.format("Yes you guessed it right, it was %d\nYou guessed it in %d attempts", number, noOfGuesses);

return true;

}

else if(inputNumber<number){

System.out.println("Too low...");

}

else if(inputNumber>number){

System.out.println("Too high...");

}

return false;

}

}

public class cwh\_50\_ex3sol {

public static void main(String[] args) {

/\*

Create a class Game, which allows a user to play "Guess the Number"

game once. Game should have the following methods:

1. Constructor to generate the random number

2. takeUserInput() to take a user input of number

3. isCorrectNumber() to detect whether the number entered by the user is true

4. getter and setter for noOfGuesses

Use properties such as noOfGuesses(int), etc to get this task done!

\*/

Game g = new Game();

boolean b= false;

while(!b){

g.takeUserInput();

b = g.isCorrectNumber();

} } }

**Exercise on Access Modifiers and Constructors:**

1. create a class cylinder and use getter and setters to set its radius and height
2. use ➊ to calculate surface and volume of the cylinder
3. Use a constructor and repeat ➊
4. Overload a constructor used to initialize a rectangle of length and breath 5 for using custom parameters
5. Repeat ➊ for a sphere

**Code solution:**

package com.company;

class Cylinder{

private int radius;

private int height;

public Cylinder(int radius, int height) {

this.radius = radius;

this.height = height;

}

public int getRadius() {

return radius;

}

public void setRadius(int radius) {

this.radius = radius;

}

public int getHeight() {

return height;

}

public void setHeight(int height) {

this.height = height;

}

public double surfaceArea(){

return 2\* Math.PI\* radius \* radius + 2\*Math.PI\*radius\*height;

}

public double volume(){

return Math.PI \* radius \* radius \* height;

}

}

class Rectangle{

private int length;

private int breadth;

public Rectangle() {

this.length = 4;

this.breadth = 5;

}

public Rectangle(int length, int breadth) {

this.length = length;

this.breadth = breadth;

}

public int getLength() {

return length;

}

public int getBreadth() {

return breadth;

}

}

public class cwh\_44\_ps09 {

public static void main(String[] args) {

/\*

// Problem 1

Cylinder myCylinder = new Cylinder(9, 12);

//myCylinder.setHeight(12);

System.out.println(myCylinder.getHeight());

//myCylinder.setRadius(9);

System.out.println(myCylinder.getRadius());

// Problem 2

System.out.println(myCylinder.surfaceArea());

System.out.println(myCylinder.volume());

\*/

// Problem 3

Rectangle r = new Rectangle(12, 56);

System.out.println(r.getLength());

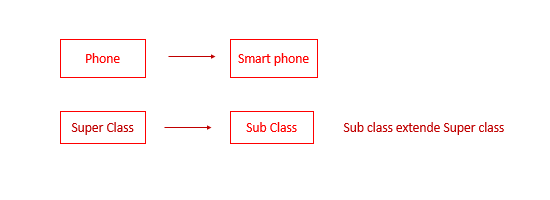
System.out.println(r.getBreadth());

}

}

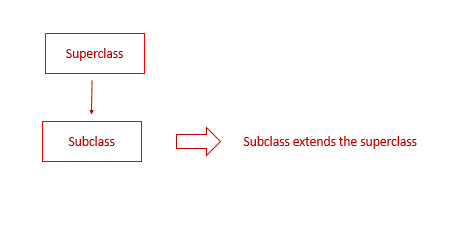
### Inheritance in Java:

Inheritance is used to borrow properties & method from an existing class

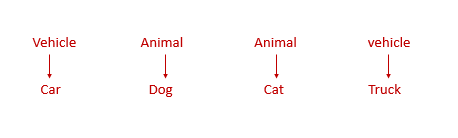
****

#### Declaring Inheritance in Java:

Inheritance in java is declared using extends keyword

****

#### More Examples:

****

When a class inherits from a superclass , it inherits part of superclass method and field java doesn’t support multiple inheritance i.e. two class cannot be super class for a subclass

#### Code Example:

Inheritance in java is declared using extends keywords

public class dog extends Animal {

// code

Quick quiz: Create a class Animal and Derive another class dog from it

package com.company;

class Base{

public int x;

public int getX() {

return x;

}

public void setX(int x) {

System.out.println("I am in base and setting x now");

this.x = x;

}

public void printMe(){

System.out.println("I am a constructor");

}

}

class Derived extends Base{

public int y;

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

}

public class cwh\_45\_inheritance {

public static void main(String[] args) {

// Creating an Object of base class

Base b = new Base();

b.setX(4);

System.out.println(b.getX());

// Creating an object of derived class

Derived d = new Derived();

d.setY(43);

System.out.println(d.getY());

}

}

**Constructors in Inheritance in Java:**

**Constructors in Inheritance -**

When a derived class is extended from the base class the constructor of the base class is executed first followed by the constructor of the derived class for the following Inheritance hierarchy, the constructors are executed in the order:

1. C1- Parent
2. C2 - Child
3. C3 - Grand child

**Constructors during constructor overloading -**

When there are multiple constructors in the parent class, the constructor without any parameters is called from the child class

If we want to call the constructor with parameters from the parent class, we can use super keywords

super(a, b) calls the constructor from the parent class which takes 2 variables

Source code as described in the video:

package com.company;

class Base1{

Base1(){

System.out.println("I am a constructor");

}

Base1(int x){

System.out.println("I am an overloaded constructor with value of x as: " + x);

}

}

class Derived1 extends Base1{

Derived1(){

//super(0);

System.out.println("I am a derived class constructor");

}

Derived1(int x, int y){

super(x);

System.out.println("I am an overloaded constructor of Derived with value of y as: " + y);

}

}

class ChildOfDerived extends Derived1{

ChildOfDerived(){

System.out.println("I am a child of derived constructor");

}

ChildOfDerived(int x, int y, int z){

super(x, y);

System.out.println("I am an overloaded constructor of Derived with value of z as: " + z);

}

}

public class cwh\_46\_constructors\_in\_inheritance {

public static void main(String[] args) {

// Base1 b = new Base1();

// Derived1 d = new Derived1();

// Derived1 d = new Derived1(14, 9);

// ChildOfDerived cd = new ChildOfDerived();

ChildOfDerived cd = new ChildOfDerived(12, 13, 15);

}

}

**this and super keyword in Java:**

**this keyword -**

this is a way for us to reference an object of the class which is being created / referenced.

In this.area = 2   --> this is a reference to current object.

**Super keyword -**

A reference variable used to refer immediate parent class object

* can be used to refer immediate parent class instance variable
* can be used to invoke parent class method
* can be used to invoke parent class constructors

package com.company;

import javax.print.Doc;

class EkClass{

int a;

public int getA() {

return a;

}

EkClass(int a){

this.a = a;

}

public int returnone(){

return 1;

}

}

class DoClass extends EkClass{

DoClass(int c){

super(c);

System.out.println("I am a constructor");

}

}

public class cwh\_47\_this\_super {

public static void main(String[] args) {

EkClass e = new EkClass(65);

DoClass d = new DoClass(5);

System.out.println(e.getA());

}

}

### Method Overriding in Java:

#### Method overriding -

if the child class implements the same method present in the parent class again, it is know as method overriding

#### Source code as described in the video:

package com.company;

class A{

public int a;

public int harry(){

return 4;

}

public void meth2(){

System.out.println("I am method 2 of class A");

}

}

class B extends A{

@Override

public void meth2(){

System.out.println("I am method 2 of class B");

}

public void meth3(){

System.out.println("I am method 3 of class B");

}

}

public class cwh\_48\_method\_overriding {

public static void main(String[] args) {

A a = new A();

a.meth2();

B b = new B();

b.meth2();

}

}

### Dynamic Method Dispatch in Java:

package com.company;

class Phone{

public void showTime(){

System.out.println("Time is 8 am");

}

public void on(){

System.out.println("Turning on Phone...");

}

}

class SmartPhone extends Phone{

public void music(){

System.out.println("Playing music...");

}

public void on(){

System.out.println("Turning on SmartPhone...");

}

}

public class cwh\_49\_dynamic\_method\_dispatch {

public static void main(String[] args) {

// Phone obj = new Phone(); // Allowed

// SmartPhone smobj = new SmartPhone(); // Allowed

// obj.name();

Phone obj = new SmartPhone(); // Yes it is allowed

// SmartPhone obj2 = new Phone(); // Not allowed

obj.showTime();

obj.on();

// obj.music(); Not Allowed

}

}

**Java Tutorial: Exercise & Practice Questions on Inheritance:**

1. Create a class circle and use inheritance to create another class cylinder from it
2. Create a class rectangle and use inheritance to create another class cuboid. try to keep it as close to real world scenario as possible
3. Create method for area and volume in 1
4. create methods for area & volume in 2 also create getters and setters
5. What is the order of constructor execution for the following inheritance hierarchy   
                       Base   
                                     
                      Derived 1  
                                         
                      Derived 2  
     
   Derived obj = new Derived 2( );   
   Which constructor(s) will be executed & in what order ?

package com.company;

class Circle{

public int radius;

Circle(){

System.out.println("I am non param of circle");

}

Circle(int r){

System.out.println("I am circle parameterized constructor");

this.radius = r;

}

public double area(){

return Math.PI\*this.radius\*this.radius;

}

}

class Cylinder1 extends Circle{

public int height;

Cylinder1(int r, int h){

super(r);

System.out.println("I am cylinder1 parameterized constructor");

this.height = h;

}

public double volume(){

return Math.PI\*this.radius\*this.radius\*this.height;

}

}

public class cwh\_52\_ch10ps {

public static void main(String[] args) {

// Problem 1

// Circle objC = new Circle(12);

Cylinder1 obj = new Cylinder1(12, 4);

}

}

### Abstract Class & Abstract Methods:

#### what does Abstract (class) mean?

Abstract in English means existing in through or as an idea without concrete existence

#### Abstract method:

A method that is declared without an implementation abstract void move to ( double X , double Y )

#### Abstract class:

If a class includes abstract methods , then the class itself must be declared abstract , as in:

public abstract class phone Model {

abstract void switch off ();

|| more code

}

when an abstract class is subclassed , the subclass usually provides implementation for all of the methods in parent class If it must be declared abstract

An example:

                           Shape   
    circle             rectangle         Rombus

Note: it is possible to create reference of an abstract class it is not possible to create an object of an abstract class we can also assign reference of an object of a concrete subclass.

package com.company;

abstract class Parent2{

public Parent2(){

System.out.println("Mai base2 ka constructor hoon");

}

public void sayHello(){

System.out.println("Hello");

}

abstract public void greet();

abstract public void greet2();

}

class Child2 extends Parent2{

@Override

public void greet(){

System.out.println("Good morning");

}

@Override

public void greet2(){

System.out.println("Good afternoon");

}

}

abstract class Child3 extends Parent2{

public void th(){

System.out.println("I am good");

}

}

public class cwh\_53\_abstract {

public static void main(String[] args) {

//Parent2 p = new Parent2(); -- error

Child2 c = new Child2();

//Child3 c3 = new Child3(); -- error

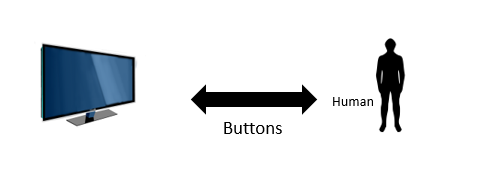
}

}

### Introduction to Interfaces:

#### Interfaces in java:

Interface in English is a point where two system meet and interact

****

In java interface is a group of related methods with empty bodies

interface Bicycle {

void apply brake ( int decrement );

void speed up ( int increment );

}

class Avon cycle implements Bicycle {

int speed = 7 ;

void apply brake ( int decrement ) {

speed = speed - decrement ;

}

void speedup ( int increment ){

speed = speed + increment ;

}

package com.company;

interface Bicycle{

int a = 45;

void applyBrake(int decrement);

void speedUp(int increment);

}

interface HornBicycle{

int x = 45;

void blowHornK3g();

void blowHornmhn();

}

class AvonCycle implements Bicycle, HornBicycle{

//public int x = 5;

void blowHorn(){

System.out.println("Pee Pee Poo Poo");

}

public void applyBrake(int decrement){

System.out.println("Applying Brake");

}

public void speedUp(int increment){

System.out.println("Applying SpeedUP");

}

public void blowHornK3g(){

System.out.println("Kabhi khushi kabhi gum pee pee pee pee");

}

public void blowHornmhn(){

System.out.println("Main hoon naa po po po po");

}

}

public class cwh\_54\_interfaces {

public static void main(String[] args) {

AvonCycle cycleHarry = new AvonCycle();

cycleHarry.applyBrake(1);

// You can create properties in Interfaces

System.out.println(cycleHarry.a);

System.out.println(cycleHarry.x);

// You cannot modify the properties in Interfaces as they are final

// cycleHarry.a = 454;

//System.out.println(cycleHarry.a);

cycleHarry.blowHornK3g();

cycleHarry.blowHornmhn();

}

}

### Abstract Classes Vs Interfaces:

#### Abstract class vs Interfaces

We can’t extend multiple abstract class but we can implement multiple interface at a time interface are meant for dynamic method dispatch and run time polymorphism.

**Why multiple inheritance is not supported in java?**

**IS multiple inheritance allow in java?**

Multiple inheritance face problem when there exist method with same signature in both the super classes

Due to such problem , java does not support multiple inheritance directly but the similar concept can be achived using interfaces

A class can implement multiple interfaces and extend a class at the same time

**Note:**

1. Interfaces in java is a bit like the class but with a significant different
2. An Interface can only have method signatures field and default method
3. the class implementing an interface needs to on declare the methods (not field)
4. you can create a reference of interface but not the object
5. interface method are public by default

### Java Interfaces Example & Default Methods:

#### Default method:

An interface can have static and default methods default methods enable us to add new functionality to existing interfaces

This feature was introduce in java 8 to ensure backward compatibility while updating an inter faces

Class implementing the interface need not implement the default methods

Interfaces can also include private methods for default methods to use

package com.company;

interface MyCamera{

void takeSnap();

void recordVideo();

private void greet(){

System.out.println("Good Morning");

}

default void record4KVideo(){

greet();

System.out.println("Recording in 4k...");

}

}

interface MyWifi{

String[] getNetworks();

void connectToNetwork(String network);

}

class MyCellPhone{

void callNumber(int phoneNumber){

System.out.println("Calling "+ phoneNumber);

}

void pickCall(){

System.out.println("Connecting... ");

}

}

class MySmartPhone extends MyCellPhone implements MyWifi, MyCamera{

public void takeSnap(){

System.out.println("Taking snap");

}

public void recordVideo(){

System.out.println("Taking snap");

}

// public void record4KVideo(){

// System.out.println("Taking snap and recoding in 4k");

// }

public String[] getNetworks(){

System.out.println("Getting List of Networks");

String[] networkList = {"Harry", "Prashanth", "Anjali5G"};

return networkList;

}

public void connectToNetwork(String network){

System.out.println("Connecting to " + network);

}

}

public class cwh\_57\_default\_methods {

public static void main(String[] args) {

MySmartPhone ms = new MySmartPhone();

ms.record4KVideo();

// ms.greet(); --> Throws an error!

String[] ar = ms.getNetworks();

for (String item: ar) {

System.out.println(item);

}

}

#### Inheritance in Interfaces:

Interfaces can extend another interfaces:

public interface Interface 1 {

void meth1 ();

}

public interface Interface 2 extends Interface 1 {

void meth 2( );

}

Remember that interface cannot implement another interface only classes can do that!

package com.company;

interface sampleInterface{

void meth1();

void meth2();

}

interface childSampleInterface extends sampleInterface{

void meth3();

void meth4();

}

class MySampleClass implements childSampleInterface{

public void meth1(){

System.out.println("meth1");

}

public void meth2(){

System.out.println("meth2");

}

public void meth3(){

System.out.println("meth3");

}

public void meth4(){

System.out.println("meth4");

}

}

public class cwh\_58\_inheritance\_interfaces {

public static void main(String[] args) {

MySampleClass obj = new MySampleClass();

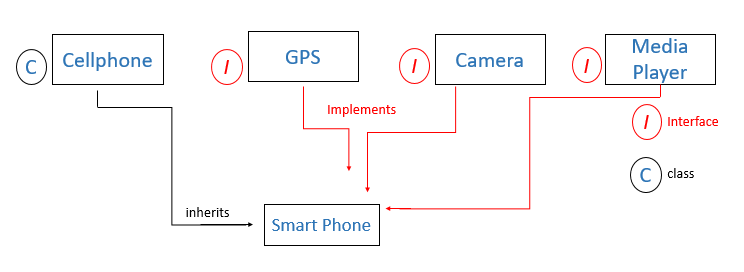
obj.meth1();

obj.meth2();

obj.meth3();

}}

### Polymorphism in Interfaces:

****

 GPS g                =    new Smartphone ( );  can only use GPS method Smartphones    =    new Smartphone ( );   can only use smartphone methods

Implementing an Interface force method implementation

package com.company;

interface MyCamera2{

void takeSnap();

void recordVideo();

private void greet(){

System.out.println("Good Morning");

}

default void record4KVideo(){

greet();

System.out.println("Recording in 4k...");

}

}

interface MyWifi2{

String[] getNetworks();

void connectToNetwork(String network);

}

class MyCellPhone2{

void callNumber(int phoneNumber){

System.out.println("Calling "+ phoneNumber);

}

void pickCall(){

System.out.println("Connecting... ");

}

}

class MySmartPhone2 extends MyCellPhone2 implements MyWifi2, MyCamera2{

public void takeSnap(){

System.out.println("Taking snap");

}

public void recordVideo(){

System.out.println("Taking snap");

}

// public void record4KVideo(){

// System.out.println("Taking snap and recoding in 4k");

// }

public String[] getNetworks(){

System.out.println("Getting List of Networks");

String[] networkList = {"Harry", "Prashanth", "Anjali5G"};

return networkList;

}

public void connectToNetwork(String network){

System.out.println("Connecting to " + network);

}

public void sampleMeth(){

System.out.println("meth");

}

}

public class cwh\_59\_polymorphism {

public static void main(String[] args) {

MyCamera2 cam1 = new MySmartPhone2(); // This is a smartphone but, use it as a camera

// cam1.getNetworks(); --> Not allowed

// cam1.sampleMeth(); --> Not allowed

cam1.record4KVideo();

MySmartPhone2 s = new MySmartPhone2();

s.sampleMeth();

s.recordVideo();

s.getNetworks();

s.callNumber(7979);

}

}

**Java Practice Questions on Abstract Classes & Interfaces :**

1. Create an abstract class pen with methods write () and refill () as abstract methods
2. Use the pen class from Q1 to create a concrete class fountain pen with additional method change Nib ()
3. Create a class monkey with jump ( ) and bite ( ) methods Create a class human which inherits this monkey class and implements basic animal interface with eat ( ) and sleep methods
4. Create a class telephone with ( ) , lift ( ) and disconnected ( ) methods as abstract methods create another class smart telephone and demonstrate polymorphism
5. Demonstrate polymorphism using using monkey  class from Q3
6. Create an interface TVremote and use it to inherit another interface smart TVremote
7. Create a class TV which implements TVremote interface from Q6

package com.company;

abstract class Pen{

abstract void write();

abstract void refill();

}

class FountainPen extends Pen{

void write(){

System.out.println("Write");

}

void refill(){

System.out.println("Refill");

}

void changeNib(){

System.out.println("Changing the nib");

}

}

class Monkey{

void jump(){

System.out.println("Jumping...");

}

void bite(){

System.out.println("Biting...");

}

}

interface BasicAnimal{

void eat();

void sleep();

}

class Human extends Monkey implements BasicAnimal{

void speak(){

System.out.println("Hello sir!");

}

@Override

public void eat() {

System.out.println("Eating");

}

@Override

public void sleep() {

System.out.println("Sleeping");

}

}

public class cwh\_60\_ch11ps {

public static void main(String[] args) {

// Q1 + Q2

FountainPen pen = new FountainPen();

pen.changeNib();

// Q3

Human harry = new Human();

harry.sleep();

// Q5

Monkey m1 = new Human();

m1.jump();

m1.bite();

// m1.speak(); --> Cannot use speak method because the reference is monkey which does not have speak method

BasicAnimal lovish = new Human();

// lovish.speak(); --> error

lovish.eat();

lovish.sleep();

}

}

**Exercise 4 - Online Library :**

You have to implement a library using Java Class "Library"

* Methods: addBook(), issueBook(), returnBook(), showAvailableBooks()
* Properties: Array to store the available books,
* Array to store the issued books

**Code Solution:**

package com.company;

class Library{

String[] books;

int no\_of\_books;

Library(){

this.books = new String[100];

this.no\_of\_books = 0;

}

void addBook(String book){

this.books[no\_of\_books] = book;

no\_of\_books++;

System.out.println(book+ " has been added!");

}

void showAvailableBooks(){

System.out.println("Available Books are:");

for (String book : this.books) {

if (book == null){

continue;

}

System.out.println("\* " + book);

}

}

void issueBook(String book){

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

if (this.books[i].equals(book)){

System.out.println("The book has been issued!");

this.books[i] = null;

return;

}

}

System.out.println("This book does not exist");

}

void returnBook(String book){

addBook(book);

}

}

public class cwh\_61\_ex4sol {

public static void main(String[] args) {

// You have to implement a library using Java Class "Library"

// Methods: addBook, issueBook, returnBook, showAvailableBooks

// Properties: Array to store the available books,

// Array to store the issued books

Library centralLibrary = new Library();

centralLibrary.addBook("Think and grow Rich");

centralLibrary.addBook("Algorithms");

centralLibrary.addBook("C++");

centralLibrary.showAvailableBooks();

centralLibrary.issueBook("C++");

centralLibrary.showAvailableBooks();

centralLibrary.returnBook("C++");

centralLibrary.showAvailableBooks();

}

}

**Interpreted vs Compiled Languages:**

**Interpreter Vs Compliler -**

Interpreter translates one statement at a time into machine code

compiler scans the entire program and translates whole of in into machine code

**Interpreter -**

1. one statement at a time
2. Interpreter is needed everytime
3. Partial execution if error
4. Easy for programmers

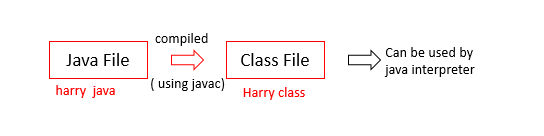
**Compiler -**

1. Entire program at a time
2. once compiled it is not needed
3. no execution if an error occurs
4. Usually not as easy as interpreted once

**Is Java interpreted or compiled?**

**Is java compiled or interpreted:**

Java is a hybrid language   both compiled as well as interpreted



* A JVM can be used to interprete this bycode
* This bytecode can be taken to any platform ( win/ mac / linux ) for education
* Hence java is platform independent ( write once run everywhere )

**Executing a java program:**

java Harry java - compiles

java Harry class - Interpreted

so far the execution of our program was being managed by intellij idea

we can download a source code like VS code to compile & execute our java programs

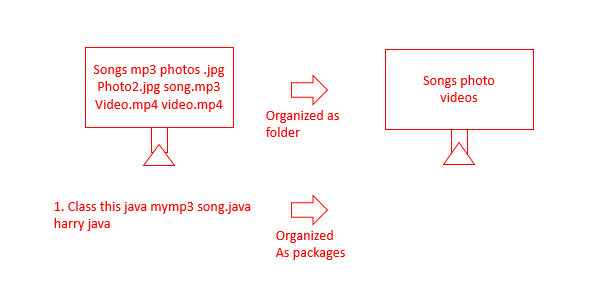
**Packages in Java:**

**Packages in java-**

A package is used to group related class packages help in avoidind name conflits

  There are two type of packages:

* Build in packages - java API
* User defined packages - Custom packages



**Using a java package-**

import java lang \* - import

import java string -  import string from java long

s= new java long string ( " Harry " ) -  use without importing

### Creating Packages in Java-

java Harry java -> creates Harry class

javac -d Harry java -> creates a packages folder   
                                                                              we can keep adding class to  
                                                                               a package like this

we can also create inner packages by adding packages inner as package name

These packages once created can be used by other class

package com.company;

import java.util.Scanner;

//import java.util.\*;

public class cwh\_65\_packages {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// java.util.Scanner sc = new java.util.Scanner(System.in);

int a = sc.nextInt();

System.out.println("This is my scanner taking input as " + a);

}

}

### Access Modifiers in Java:

Access modifiers determine whether other class can use a particular field or invoke a particular method can be public, private, protected or default (no modifier)

Modifier          class          Package          Subclass          World

Public               Y                   Y                        Y                      Y

protected          Y                   Y                        Y                      N

Default (no)       Y                   Y                        N                      N

Private                Y                   N                       N                     N

package com.company;

class C1{

public int x = 5;

protected int y =45;

int z = 6;

private int a = 78;

public void meth1(){

System.out.println(x);

System.out.println(y);

System.out.println(z);

System.out.println(a);

}

}

public class cwh\_66\_access\_modifiers {

public static void main(String[] args) {

C1 c = new C1();

// c.meth1();

System.out.println(c.x);

System.out.println(c.y);

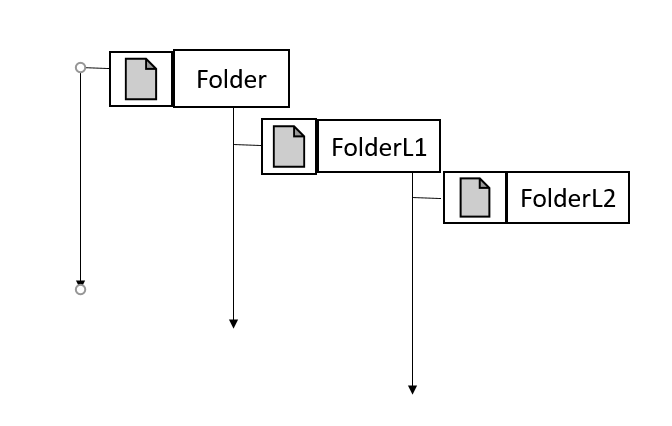
System.out.println(c.z);

// System.out.println(c.a);

}

}

**Practice Set on Java Package & Access Modifiers:**

1. Create three classes calculator, Sc calculator and Hybridcalculator and group them into a package
2. Use a build-in package in java to write a class which displays a message ( by using sout ) after taking input from the user
3. Create a package in class with three package levels folder, folderL1, folderL2   
                                             ****
4. prove that you cannot access default property but can access protected properly from the subclass.

### Creating a Custom Package:

 You have to create a package named com.codewithharry.shape  
 This package should have individual classes for Rectangle, Square, Circle, Cylinder, Sphere  
 These classes should use inheritance to properly manage the code!  
 Include methods like volume, surface area and getters/setters for dimensions

package com.company;

/\*

\*\*\* WRITE THIS CODE IN NOTEPAD \*\*\*

You have to create a package named com.codewithharry.shape

This package should have individual classes for Rectangle, Square, Circle, Cylinder, Sphere

These classes should use inheritance to properly manage the code!

Include methods like volume, surface area and getters/setters for dimensions

\*/

public class cwh\_68\_ex5 {

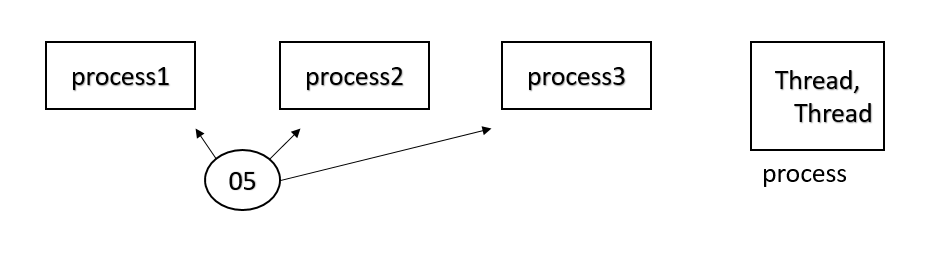
public static void main(String[] args) {

}

}

**Multithreading in Java:**

Multiprocessing and multithreading both are used to achieve multitasking

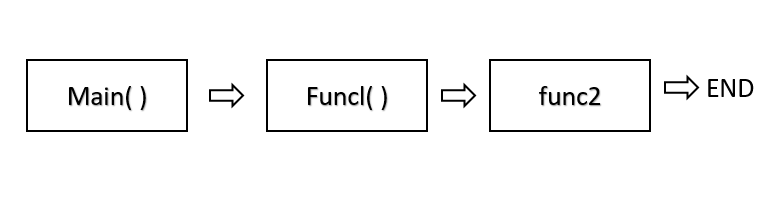
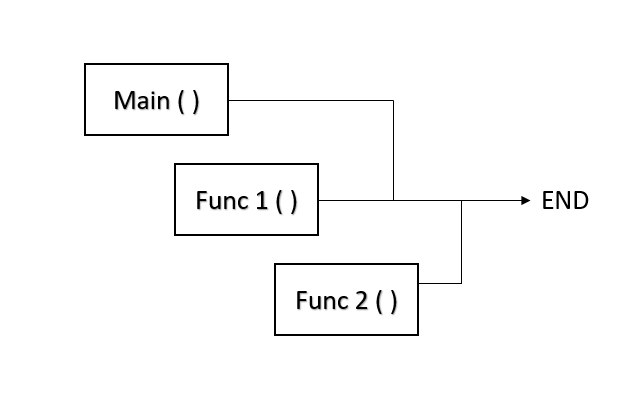


**In a nut shell.......**

* threads use shared memory area
* threads = faster content switching
* Athread is light-weight where a process is heavyweight

   for example = Aword processor can have one thread running in foreground as an editor and another in the background auto saving the document!

**Flow of control in java**

* Without  threading:  
    
  
* With threading:   
    
  

**Creating a Threading:**

   There are two ways to create a thread in java

1. By extending thread class
2. By implementing Runnable interface

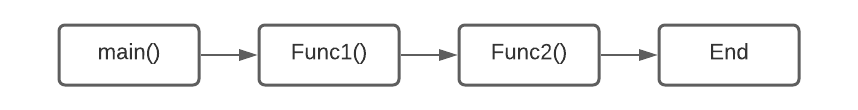
### Creating a Thread by Extending Thread class

#### Multithreading In Java:

* Used to maximize the CPU utilization.
* We don't want our CPU to be in a free state, for example, Func1() comes into the memory and demands any input/output process. In such a condition, the CPU will need to wait for unit Func1() to complete its input/output operation. But, while Func1() completes its I/O operation, the CPU is free and not executing any thread. So, the efficiency of the CPU is decreased in the absence of multithreading.
* In the case of multithreading, if a thread demands any I/O operation, then the CPU will let the thread perform its I/O operation but it will start the execution of a new thread parallelly. So, in this case, two threads are executing at the same time.

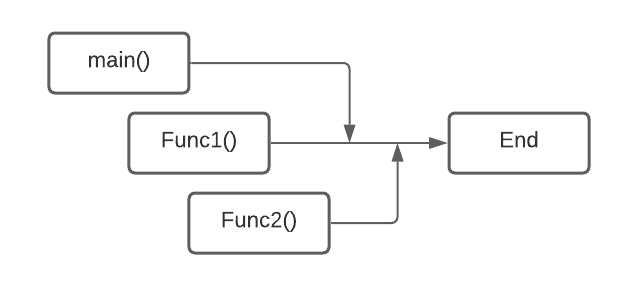
#### Flow Of Control in Java:

##### **Without threading:**



In the above image, you can see that Func1() and Func2() are called inside the main() function. But the execution of Func2() will start only after the completion of the Func1().

##### **2. With threading:**



Again, Func1() and Func2() are called inside the main function, but none of the two functions is waiting for the execution of the other function. Both the functions are getting executed concurrently.

#### Ways To Create a Thread In Java:

1. By extending the thread class
2. By implementing Runnable interface

Let's see how we can create a thread by extending the thread class.

##### **Extending Thread Class:**

To create a thread using the thread class, we need to extend the thread class. Java's multithreading system is based on the thread class.

class MyThread extends Thread{

@Override

public void run(){

//code that we want to get executed on running the thread

}

}

* In the above code, first, we're inheriting the Thread class and then we're overriding the run() method.
* The code that you want to execute on the execution of the thread goes inside the run() method.

class MyThread extends Thread{

@Override

public void run(){

int i =0;

while(i<40000){

System.out.println("My Cooking Thread is Running");

System.out.println("I am happy!");

i++;

}

}

}

public class cwh\_70 {

public static void main(String[] args) {

MyThread t1 = new MyThread();

t1.start();

}

}

In order to execute the thread, the start() method is used. start() is called on the object of the MyThread class. It automatically calls the run() method and a new stack is provided to the thread. So, that's how you easily create threads by extending the thread class in Java.

package com.company;

class MyThread1 extends Thread{

@Override

public void run(){

int i =0;

while(i<40000){

System.out.println("My Cooking Thread is Running");

System.out.println("I am happy!");

i++;

}

}

}

class MyThread2 extends Thread{

@Override

public void run(){

int i =0;

while(i<40000){

System.out.println("Thread 2 for Chatting with her");

System.out.println("I am sad!");

i++;

}

}

}

public class cwh\_70 {

public static void main(String[] args) {

MyThread1 t1 = new MyThread1();

MyThread2 t2 = new MyThread2();

t1.start();

t2.start();

}

}

### Creating a Java Thread Using Runnable Interface:

In the previous tutorial, I told you that there are two ways to create a thread in java:

1. By Extending Thread Class
2. By implementing Runnable interface

We've already seen in this tutorial, we'll see how to create a Java thread by using a runnable interface.

#### Steps to Create a Java Thread Using Runnable Interface:

1. Create a class and implement the Runnable interface by using the implements keyword.
2. Override the run() method inside the implementer class.
3. Create an object of implementer class in the main() method.
4. Instantiate the Thread class and pass the object to the Thread constructor.
5. Call start() on the thread. start()will call the run()method.

#### Example:

class t1 implements Runnable{

@Override

public void run(){

System.out.println("Thread is running");

}

}

public class ClassName{

public static void main(String[] args) {

t1 obj1 = new t1();

Thread t = new Thread(obj1);

t.start();

}

}

1. class t1 is implementing the Runnable interface.
2. Overriding of the run() method is done inside the t1 class.
3. In the main() method, obj1 which is an object of the t1 class is created.
4. The constructor of the Thread class accepts the Runnable instance as an argument so obj1 is passed to the constructor of the Thread class.
5. Finally, the start()method is called on the thread which will call the run() method internally and the execution of the thread will begin.

#### Runnable Interface Vs Extending Thread Class:

Since we've discussed both the ways to create a thread in Java. There might be a question in your mind that should we use the Runnable interface or Thread class to implement a thread in Java. Let me answer this question for you. The Runnable interface is preferred over extending the Thread class because of the following reasons:

1. As multiple inheritance is not supported in Java so it is not possible to extend the Thread class if your class had already extended some other class.
2. While implementing Runnable, we do not modify or change the thread's behavior.
3. More memory is required while extending the Thread class because each thread creates a unique object.
4. Less memory is required while implementing Runnable because multiple threads share the same object.

package com.company;

class MyThreadRunnable1 implements Runnable{

public void run(){

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

System.out.println("I am a thread 1 not a threat 1");

}

}

class MyThreadRunnable2 implements Runnable{

public void run(){

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

System.out.println("I am a thread 2 not a threat 2");

}

}

public class cwh\_71\_runnable {

public static void main(String[] args) {

MyThreadRunnable1 bullet1 = new MyThreadRunnable1();

Thread gun1 = new Thread(bullet1);

MyThreadRunnable2 bullet2 = new MyThreadRunnable2();

Thread gun2 = new Thread(bullet2);

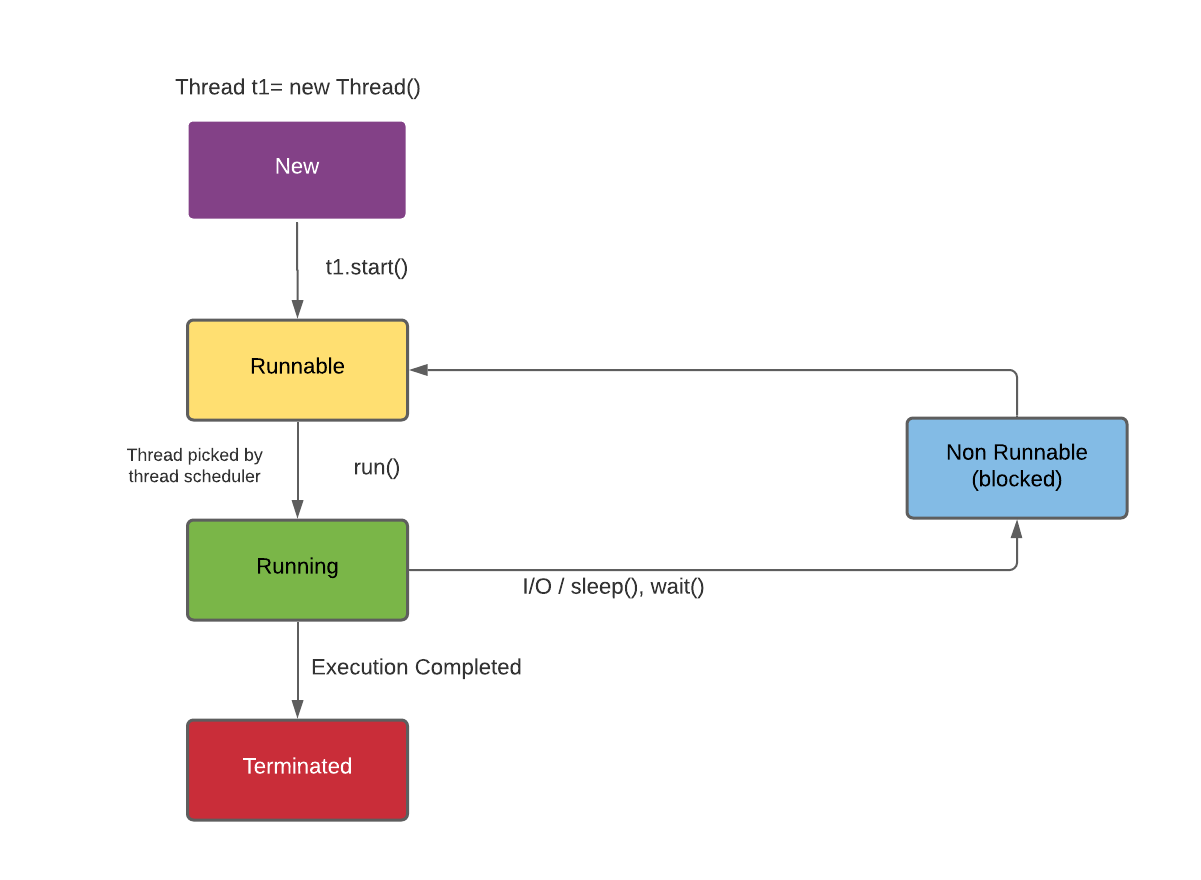
gun1.start();

gun2.start();

}

}

### Java Thread Life Cycle:



1. New - Instance of thread created which is not yet started by involving start(). In this state, the thread is also known as the born thread.
2. Runnable - After invocation of start()& before it is selected to be run by the scheduler.
3. Running - After the thread scheduler has selected it.
4. Non-runnable - thread alive, not eligible to run.
5. Terminated - run() method has exited.

**Constructors from Thread class in Java:**

**The Thread class:**

Below are the commonly used constructors of thread class:

1. Thread ()
2. Thread (string)
3. Thread (Runnable r)
4. Thread (Runnable r, String name)

package com.company;

class MyThr extends Thread{

public MyThr(String name){

super(name);

}

public void run(){

int i = 34;

System.out.println("Thank you");

// while(true){

// System.out.println("I am a thread");

// }

}

}

public class cwh\_73\_thread\_constructor {

public static void main(String[] args) {

MyThr t1 = new MyThr("Harry");

MyThr t2 = new MyThr("Ram Candr");

t1.start();

t2.start();

System.out.println("The id of the thread t is " + t1.getId());

System.out.println("The name of the thread t is " + t1.getName());

System.out.println("The id of the thread t is " + t2.getId());

System.out.println("The name of the thread t is " + t2.getName());

}

}

### Java Thread Priorities:

In a Multithreading environment, all the threads which are ready and waiting to be executed are present in the Ready queue. The thread scheduler is responsible for assigning the processor to a thread. But the question is on what basis the thread scheduler decides that a particular thread will run before other threads. Here comes the concept of priority in the picture.

1. Every single thread created in Java has some priority associated with it.
2. The programmer can explicitly assign the priority to the thread otherwise JVM automatically assigns priority while creating the thread.
3. In Java, we can specify the priority of each thread relative to other threads. Those threads having higher priorities get greater access to the available resources than lower priorities threads.
4. Thread scheduler will use priorities while allocating processor.
5. The valid range of thread priorities is 1 to 10 (but not 0 to 10) where 1 is the least priority and 10 is the higher priority.
6. If there is more than one thread of the same priority in the queue then the thread scheduler picks any one of them to execute.
7. The following static final integer constants are defined in the Thread class:

* **MIN\_PRIORITY**: Minimum priority that a thread can have. Value is 1.
* **NORM\_PRIORITY**: This is the by default priority automatically given by JVM to a thread if a programmer does not explicitly set the priority of that thread. Value is 5.
* **MAX\_PRIORITY**: Maximum priority that a thread can have. Value is 10.

#### Priority Methods in Java:

##### **setPriority(): This method is used to set the priority of a thread. IllegalArgumentException is thrown if the priority given by the user is out of the range [1,10].**

**Syntax:**

public final void setPriority(int x) // x is the priority [1,10] that is to be set for the thread.

  2. **getPriority():** This method is used to display the priority of a given thread.

**Syntax:**

t1.getPriority() // Will return the priortity of the t1 thread.

#### Example:

class cwh\_Priority extends Thread{

public void run(){

System.out.println("I'm thread : "+Thread.currentThread().getName());

System.out.println("I'm thread :"+Thread.currentThread().getPriority());

}

public static void main(String args[]){

cwh\_Priority t1=new cwh\_Priority();

cwh\_Priority t2= new cwh\_Priority();

t1.setPriority(Thread.MIN\_PRIORITY); // setting priority of t1 thread to MIN\_PRIORITY (1)

t2.setPriority(Thread.MAX\_PRIORITY); // setting priority of t2 thread to MAX\_PRIORITY (10)

t1.start();

t2.start();

}

}

#### Output:

I'm thread : Thread-0

I'm thread : Thread-1

I'm thread :10

I'm thread :1

package com.company;

class MyThr1 extends Thread{

public MyThr1(String name){

super(name);

}

public void run(){

int i = 34;

while(true){

// System.out.println("I am a thread");

System.out.println("Thank you: " + this.getName());

}

}

}

public class cwh\_74\_thread\_priorities {

public static void main(String[] args) {

// Ready Queue: T1 T2 T3 T4 T5

MyThr1 t1 = new MyThr1("Harry1");

MyThr1 t2 = new MyThr1("Harry2");

MyThr1 t3 = new MyThr1("Harry3");

MyThr1 t4 = new MyThr1("Harry4");

MyThr1 t5 = new MyThr1("Harry5 (most Important)");

t5.setPriority(Thread.MAX\_PRIORITY);

t1.setPriority(Thread.MIN\_PRIORITY);

t2.setPriority(Thread.MIN\_PRIORITY);

t3.setPriority(Thread.MIN\_PRIORITY);

t4.setPriority(Thread.MIN\_PRIORITY);

t5.setPriority(Thread.MIN\_PRIORITY);

t1.start();

t2.start();

t3.start();

t4.start();

t5.start();

}

}

### Java Thread Methods:

#### Join() method In Java :

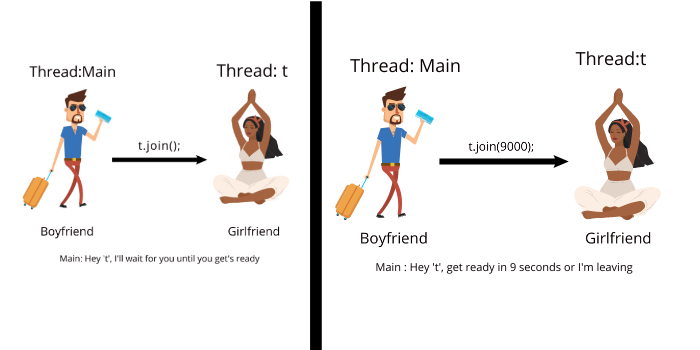
* The join() method in Java allows one thread to wait until the execution of some other specified thread is completed.
* If t is a Thread object whose thread is currently executing,t.join(); causes the current thread to pause execution until t's thread terminates.
* Join() method puts the current thread on wait until the thread on which it is called is dead.

##### **Syntax:**

public final void join()

You can also specify the time for which you need to wait for the execution of a particular thread by using the Join() method. Syntax :

public final void join(long millis)



#### Sleep() Method :

* The sleep() method in Java is useful to put a thread to sleep for a specified amount of time.
* When we put a thread to sleep, the thread scheduler picks and executes another thread in the queue.
* Sleep() method returns void.
* sleep() method can be used for any thread including the main() thread also.

##### **Syntax:**

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

##### **Parameters Passed To Sleep() Method :**

1. long milisecond: Time in milliseconds for which thread will sleep.
2. nanos : Ranges from [0,999999]. Additional time in nanoseconds.

##### **Example:**

import java.io.\*;

import java.lang.Thread;

public class cwh {

public static void main(String[] args)

{

try {

for (int i = 1; i <=5; i++) {

Thread.sleep(2000);

System.out.println(i);

}

}

catch (Exception e) {

System.out.println(e);

}

}

}

In the above example, the main() method will be put to sleep for 2 seconds every time the for loop executes.

##### **Output:**

1

2

3

4

5

#### Interrupt() method :

package com.company;

class MyNewThr1 extends Thread{

public void run(){

int i = 0;

while(true){

// System.out.println("I am a thread");

System.out.println("Thank you: ");

try {

Thread.sleep(455);

} catch (InterruptedException e) {

e.printStackTrace();

}

i++;

}

}

}

class MyNewThr2 extends Thread{

public void run(){

while(true){

// System.out.println("I am a thread");

System.out.println("My Thank you: ");

}

}

}

public class cwh\_75\_thread\_methods {

public static void main(String[] args){

MyNewThr1 t1 = new MyNewThr1();

MyNewThr2 t2 = new MyNewThr2();

t1.start();

// try{

// t1.join();

// }

// catch(Exception e){

// System.out.println(e);

// }

t2.start();

}

}

### Solutions Practice Questions on Thread:

package com.company;

class Practice13 extends Thread{

public void run(){

while(true){

System.out.println("Good Morning!");

}

}

}

class Practice13b extends Thread{

public void run(){

// while(false){

// try {

// Thread.sleep(200);

// }

// catch (Exception e){

// System.out.println(e);

// }

// System.out.println("Welcome");

// }

}

}

public class cwh\_76\_practice13 {

public static void main(String[] args) {

Practice13 p1 = new Practice13();

Practice13b p2 = new Practice13b();

// p1.setPriority(6);

// p2.setPriority(9);

System.out.println(p1.getPriority());

System.out.println(p2.getPriority());

System.out.println(p2.getState());

// p1.start();

p2.start();

System.out.println(p2.getState());

System.out.println(Thread.currentThread().getState());

}

}

### Errors & Exception in Java:

No matter how smart we are, errors are our constant comparisons.

With practice, we keep getting better at finding 2 correcting them.

There are three types of errors in java.

1) Syntax errors

2) Logical errors

3) Runtime errors- also called Exceptions

#### Syntax Errors:

When complex finds something wrong with our program,

it throws a syntax error

int a = 9 // No semicolon, syntax errors!

a = a + 3;

d = 4; // Variable not declared, syntax errors

#### Logical errors:

A logical error or a bug occurs when a program

compiles and round but does the wrong thing.

- Message delivered wrongly

- Wrong time of chats being displayed

* Incorrect redirects!

#### Runtime errors:

Java may sometimes encounter an error while the program is running.

These are also called Exceptions!

These are encountered due to circumstances like bad input and (or) resource constraints.

Ex: User supplies 'S' + 8 to a program which adds 2 numbers.

Syntax errors and logical errors are encountered by the programmers where as Runtime errors are encountered by the users.

package com.company;

public class cwh\_78\_errors {

public static void main(String[] args) {

int a = 5;

int b = 9;

System.out.println(a+b);

}

}

### Syntax Errors, Runtime Errors & Logical Errors in Java (Demo):

package com.company;

import java.util.Scanner;

public class cwh\_79\_errorsdemo {

public static void main(String[] args) {

// SYNTAX ERROR DEMO

// int a = 0 // Error: no semicolon!

// b = 8; // Error: b not declared!

// LOGICAL ERROR DEMO

// Write a program to print all prime numbers between 1 to 10

System.out.println(2);

for (int i=1; i<5; i++){

System.out.println(2\*i+1);

}

// RUNTIME ERROR

int k;

Scanner sc = new Scanner(System.in);

k = sc.nextInt();

System.out.println("Integer part of 1000 divided by k is "+ 1000/k);

}}

### Exceptions & Try-Catch Block in Java:

#### Exceptions in Java:

An exception is an event that occurs when a program is executed dissented the normal flow of instructions.

There are mainly two types of exceptions in java:

1) Checked exceptions - compile time exceptions (Handle by compiler)

2) Unchecked exceptions - Runtime exceptions

#### Commonly Occurring Exceptions:

Following are few commonly occurring exceptions in java:

1) Null pointer exception

2) Arithmetic Exception

3) Array Index out of Bound exception

4) Illegal Argument Exception

5) Number Format Exception

package com.company;

public class cwh\_80\_try {

public static void main(String[] args) {

int a = 6000;

int b = 0;

// Without Try:

// int c = a / b;

// System.out.println("The result is " + c);

// With Try:

try {

int c = a / b;

System.out.println("The result is " + c);

}

catch(Exception e) {

System.out.println("We failed to divide. Reason: ");

System.out.println(e);

}

System.out.println("End of the program");

}

}

### Handling Specific Exceptions in Java:

#### Handling specific Exceptions:

In java, we can handle specific exceptions by typing multiple catch blocks.

try (

// code

)

Catch (To Exception e) - Handle all Exceptions of type IO Exception

// code

)

Catch (Exception e) - Handle all Exceptions of type Arithmetic Exception

// code

)

Catch (Exception e) - Handle all other Exceptions

// code

)

package com.company;

import java.util.Scanner;

public class cwh\_81 {

public static void main(String[] args) {

int [] marks = new int[3];

marks[0] = 7;

marks[1] = 56;

marks[2] = 6;

Scanner sc = new Scanner(System.in);

System.out.println("Enter the array index");

int ind = sc.nextInt();

System.out.println("Enter the number you want to divide the value with");

int number = sc.nextInt();

try{

System.out.println("The value at array index entered is: " + marks[ind]);

System.out.println("The value of array-value/number is: " + marks[ind]/number);

}

catch (ArithmeticException e){

System.out.println("ArithmeticException occured!");

System.out.println(e);

}

catch (ArrayIndexOutOfBoundsException e){

System.out.println("ArrayIndexOutOfBoundsException occured!");

System.out.println(e);

}

catch (Exception e){

System.out.println("Some other exception occured!");

System.out.println(e);

}

}

}

### Nested Try-Catch in Java:

#### Nested try – catch:

We can nest multiple try - catch blocks as follows:

package com.company;

import java.util.Scanner;

public class cwh\_82\_nested\_try\_catch {

public static void main(String[] args) {

int [] marks = new int[3];

marks[0] = 7;

marks[1] = 56;

marks[2] = 6;

Scanner sc = new Scanner(System.in);

boolean flag = true;

while(flag) {

System.out.println("Enter the value of index");

int ind = sc.nextInt();

try {

System.out.println("Welcome to video no 82");

try {

System.out.println(marks[ind]);

flag = false;

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Sorry this index does not exist");

System.out.println("Exception in level 2");

}

} catch (Exception e) {

System.out.println("Exception in level 1");

}

}

System.out.println("Thanks for using this program");

}

}

Similarly, we can further nest try - catch blocks inside the nested try catch blocks.

**Quick Quiz:** Write a java program that allows to keep accessing an array until a valid index is given by the user.

### The Exception class in Java:

We can write our custom Exceptions using Exception class in java.

public class MyException extends Exception {

// Overridden methods

}

The Exception class has following important methods:

1) Strings toString() executed when sout (e) is ran

2) Void printStackTrace() - prints Stack trace

3) String getMessage() - prints the exception message

package com.company;

import java.util.Scanner;

class MyException extends Exception{

@Override

public String toString() {

return "I am toString()";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class MaxAgeException extends Exception{

@Override

public String toString() {

return "Age cannot be greater than 125";

}

@Override

public String getMessage() {

return "Make sure that the value of age entered is correct";

}

}

public class cwh\_83\_exception\_class {

public static void main(String[] args) {

int a;

Scanner sc = new Scanner(System.in);

a = sc.nextInt();

if (a<9){

try{

// throw new MyException();

throw new ArithmeticException("This is an exception");

}

catch (Exception e){

System.out.println(e.getMessage());

System.out.println(e.toString());

e.printStackTrace();

System.out.println("Finished");

}

System.out.println("Yes Finished");

}

} }

### Throw vs Throws in Java:

#### The Throw Keyword:

The throw keyword is used to throw an exception explicitly by the programmer

if ( b===0 ) {

throw new ArithmeticException("Div by 0");

}

else{

return a/b ;

}

In a similar manner, we can throw user defined exceptions:

throw new My Exception ( "Exception throw" );

#### The throw Keyword:

Throws java throws keyword is used to declare an exception.

This gives an information to the programmer that there might be an exception so its better to be prepared with a try catch block!

public void calculate (int a, int b) throws IOException {

// code

}

package com.company;

class NegativeRadiusException extends Exception{

@Override

public String toString() {

return "Radius cannot be negative!";

}

@Override

public String getMessage() {

return "Radius cannot be negative!";

}

}

public class cwh\_84\_throw\_throws {

public static double area(int r) throws NegativeRadiusException{

if (r<0){

throw new NegativeRadiusException();

}

double result = Math.PI \* r \* r;

return result;

}

public static int divide(int a, int b) throws ArithmeticException{

// Made By Harry

int result = a/b;

return result;

}

public static void main(String[] args) {

// Shivam - uses divide function created by Harry

try{

// int c = divide(6, 0);

// System.out.println(c);

double ar = area(6);

System.out.println(ar);

}

catch(Exception e){

System.out.println("Exception");

}

}

}

### Finally Block in Java & Why is it needed!

#### Java finally block:

Finally block contains the code which is always executed whether the exception is handled or not.

It is used to exception is handled or not.

It is used to execute code containing instructions to release the system resources, close a connection etc.

#### Sample Code Demonstrating Finally

package com.company;

public class cwh\_85\_finally {

public static int greet(){

try{

int a = 50;

int b = 10;

int c = a/b;

return c;

}

catch(Exception e){

System.out.println(e);

}

finally {

System.out.println("Cleaning up resources...This is the end of this function");

}

return -1;

}

public static void main(String[] args) {

int k = greet();

System.out.println(k);

int a = 7;

int b = 9;

while(true){

try{

System.out.println(a/b);

}

catch (Exception e){

System.out.println(e);

break;

}

finally{

System.out.println("I am finally for value of b = " + b);

}

b--;

}

try{

System.out.println(50/3);

}

finally {

System.out.println("Yes this is finally");

}

}

}

### Practice Set on Errors & Exceptions:

#### Chapter- 14 - Practice set:

1) Write a java program to demonstrate syntax, logical 2 runtime errors.

2) Write a java program that prints "HaHa" during Arithmetic exception and "HeHe" during an Illegal argument exception.

3) Write a program that allows you to given. If max retries exceed 5 print "errors".

4) Modify program in Q3 to throw a custom Exception if max retries are reached.

5) Wrap the program in Q3 inside a method which throws your custom Exception.

package com.company;

import java.util.Scanner;

public class cwh\_86\_ps14 {

public static void main(String[] args) {

// Problem 1

// Syntax Error - int a = 7

int age = 78;

int year\_born = 2000-78; // Logical error

// System.out.println(6/0);

// Problem 2

try{

int a = 666/0;

}

catch (IllegalArgumentException e){

System.out.println("HeHe");

}

catch (ArithmeticException e){

System.out.println("Haha");

}

// Problem 3

boolean flag = true;

int [] marks = new int[3];

marks[0] = 7;

marks[1] = 56;

marks[2] = 6;

Scanner Sc = new Scanner(System.in);

int index;

int i = 0;

while(flag && i<5){

try {

System.out.println("Enter the value of index");

index = Sc.nextInt();

System.out.println("The value of marks[index] is " + marks[index]);

break;

}

catch (Exception e) {

System.out.println("Invalid Index");

i++;

}

}

if(i>=5){

System.out.println("Error");

}

}

}

### Java Exercise: Custom Calculator | Java Practice Question-

Exercise: You have to create a custom calculator with following operations:  
1. + -> Addition  
2. - -> Subtraction  
3. \* -> Multiplication  
4. / -> Division  
which throws the following exceptions:  
1. Invalid input Exception ex: 8 & 9  
2. Cannot divide by 0 Exception  
3. Max Input Exception if any of the inputs is greater than 100000  
4. Max Multiplier Reached Exception - Don't allow any multiplication input to be greater than 7000

### Java Exercise: Solution | Custom Calculator-

Below is the solution of exercise number that I gave you all in:

package com.company;

class InvalidInputException extends Exception{

@Override

public String toString() {

return "Cannot add 8 and 9";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class MaxInputException extends Exception{

@Override

public String toString() {

return "Input cant be greater than 100000";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class CannotDivideByZeroException extends Exception{

@Override

public String toString() {

return "Cannot divide by 0";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class MaxMultiplyInputException extends Exception{

@Override

public String toString() {

return "Input cant be greater than 7000 while multiplying";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class CustomCalculator {

double add(double a, double b) throws InvalidInputException, MaxInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

if(a==8 || b==9) {

throw new InvalidInputException();

}

return a + b;

}

double subtract(double a, double b) throws MaxInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

return a - b;

}

double multiply(double a, double b)throws MaxInputException, MaxMultiplyInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

else if(a>7000 || b>7000){

throw new MaxMultiplyInputException();

}

return a \* b;

}

double divide(double a, double b) throws CannotDivideByZeroException, MaxInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

if(b==0){

throw new CannotDivideByZeroException();

}

return a / b;

}

}

public class cwh\_103\_ex6sol {

public static void main(String[] args) throws InvalidInputException,

CannotDivideByZeroException, MaxInputException, MaxMultiplyInputException {

CustomCalculator c = new CustomCalculator();

// c.add(8, 9);

// c.divide(6, 0);

// c.divide(600000000, 40);

c.multiply(5, 9888);

/\*

Exercise 6: You have to create a custom calculator with following operations:

1. + -> Addition

2. - -> Subtraction

3. \* -> Multiplication

4. / -> Division

which throws the following exceptions:

1. Invalid input Exception ex: 8 & 9

2. Cannot divide by 0 Exception

3. Max Input Exception if any of the inputs is greater than 100000

4. Max Multiplier Reached Exception - Don't allow any multiplication input to be greater than 7000

\*/

}}

### Advance Java:

### Java Collections Framework:

**Collection Framework-**

A collection represents a group of object Java collections provide classes and Interfaces for us to be able to write code interfaces for us to be able to write code quickly and efficiently

Why do we need collections

We need collections for efficient storage and better manipulation of data in java

For ex: we use arrays to store integers but what if we want to

* Resize this array?
* Insert an element in between?
* Delete an elements in Array?
* Apply certain operations to change this array?

package com.company;

import java.util.ArrayList;

import java.util.Set;

import java.util.TreeSet;

public class cwh\_89\_collections {

public static void main(String[] args) {

// ArrayList

// Set

// TreeSet

}

}

**Collections Hierarchy in Java:**

**How are collections available-**

Collections in java are available as class and interfaces Folling are few commonly used collections in java:

* ArrayList -> For variables size collections
* Set -> For distinct collection
* Stack-> A LIFO data structure
* HashMap -> For strong key - value pairs

Collections class is available in java util package collection class also provides static methods for sorting , searching etc.

package com.company;

import java.util.ArrayList;

import java.util.Set;

import java.util.TreeSet;

public class cwh\_89\_collections {

public static void main(String[] args) {

// ArrayList

// Set

// TreeSet

}

}

**ArrayList in Java: Demo & Methods:**

* The ArrayList class is the dynamic array found in the java.util package.
* The size of the normal array can not be changed, but ArrayList provides us the ability to increase or decrease the size.
* ArrayList is slower than the standard array, but it helps us to manipulate the data easily.

How to declare an ArrayList :

ArrayList<Integer> l1 = new ArrayList<>(); /\*Creates an ArrayList object of

integer type \*/

package com.company;

import java.util.\*;

public class cwh\_92\_linkedlist {

public static void main(String[] args) {

LinkedList<Integer> l1 = new LinkedList<>();

LinkedList<Integer> l2 = new LinkedList<>();

l2.add(15);

l2.add(18);

l2.add(19);

l1.add(6);

l1.add(7);

l1.add(4);

l1.add(6);

l1.add(0, 5);

l1.add(0, 1);

l1.addAll(0, l2);

l1.addLast(676);

l1.addFirst(788);

System.out.println(l1.contains(27));

System.out.println(l1.indexOf(6));

System.out.println(l1.lastIndexOf(6));

//l1.clear();

l1.set(1, 566);

for(int i=0; i<l1.size(); i++){

System.out.print(l1.get(i));

System.out.print(", ");

}

}

}

#### Performing various operations on ArrayList:

ArrayList comes with a number of methods that can be used to manipulate the data of the ArrayList. Let’s take a look at some of the important methods of ArrayList:

##### **Adding an element:**

* + add() method is used to insert an element in the ArrayList.
  + add(Object): Inserts an element at the end of the ArrayList.
  + add(Index,Object) : Inserts an element at the given index.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayList<Integer> l1 = new ArrayList<>();

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(4);

l1.add(6);

l1.add(5,5); // inserts 5 at the 5th index in l1

System.out.println(l1);

}

}

##### **Output:**

[1, 2, 3, 4, 6, 5]

#####  **Removing an Element:**

* remove() method is used to delete or remove an element at a given index from the ArrayList.

##### **Syntax:**

L1.remove(index number)

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayList<Integer> l1 = new ArrayList<>();

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(4);

l1.add(6);

l1.add(5,5); // inserts 5 at the 5th index in l1

System.out.println("Array list before : "+ l1);

l1.remove(0);

System.out.println("ArrayList after removing the value at index 0 :" + l1);

}

}

##### **Output:**

Array list before : [1, 2, 3, 4, 6, 5]

ArrayList after removing the value at index 0 : [2, 3, 4, 6, 5]

#####  **Checking if an ArrayList contains a specific value or not :**

* contains() method is used to check if an ArrayList contains a specified element or not. This method returns the boolean value.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayList<Integer> l1 = new ArrayList<>();

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(4);

l1.add(5);

l1.add(6);

System.out.println("Array list : "+ l1);

System.out.println("L1 list contains 7 : " + l1.contains(7));

System.out.println("L1 list contains 4 : " + l1.contains(4));

}

}

##### **Output:**

Array list before : [1, 2, 3, 4, 5, 6]

L1 list contains 7 : false

L1 list contains 4 : true

#####  **Merging two ArrayLists:**

* The elements of an ArrayList can be merged into another Arraylist with the help of the addAll() method.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayList<Integer> l1 = new ArrayList<>();

ArrayList<Integer> l2 = new ArrayList<>();

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(4);

l1.add(5);

l1.add(6);

l2.add(11);

l2.add(12);

l2.add(13);

l2.add(14);

System.out.println("L1 Array list : "+ l1);

System.out.println("L2 Array list : " +l2);

l1.addAll(l2);

System.out.println("L1 Array list after merging: "+ l1);

System.out.println("L2 Array list : " +l2);

}

}

##### **Output:**

L1 Array list : [1, 2, 3, 4, 5, 6]

L2 Array list : [11, 12, 13, 14]

L1 Array list after merging: [1, 2, 3, 4, 5, 6, 11, 12, 13, 14]

L2 Array list : [11, 12, 13, 14]

* You can add the elements of l2 at the starting of l1 by typing :

L1.addAll(0,l2);

#####  **Finding the first occurrence of a specified number in the ArrayList :**

* Indexof() method prints the index of the first occurrence of a particular number. Returns -1 if the element is not present in the ArrayList.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayList<Integer> l1 = new ArrayList<>();

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(4);

l1.add(5);

l1.add(6);

l1.add(3);

System.out.println("L1 Array list : "+ l1);

System.out.println("The first occurrence of 3 in l1 is at index : " + l1.indexOf(3));

}

##### **Output:**

L1 Array list : [1, 2, 3, 4, 5, 6, 3]

The first occurrence of 3 in l1 is at index : 2

Similarly, you can also find the index of the last occurrence of an element with the help of the lastIndexOf() method.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayList<Integer> l1 = new ArrayList<>();

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(4);

l1.add(5);

l1.add(3);

l1.add(6);

l1.add(3);

System.out.println("L1 Array list : "+ l1);

System.out.println("The last occurrence of 3 in l1 is at index : " + l1.lastIndexOf(3));

}

}

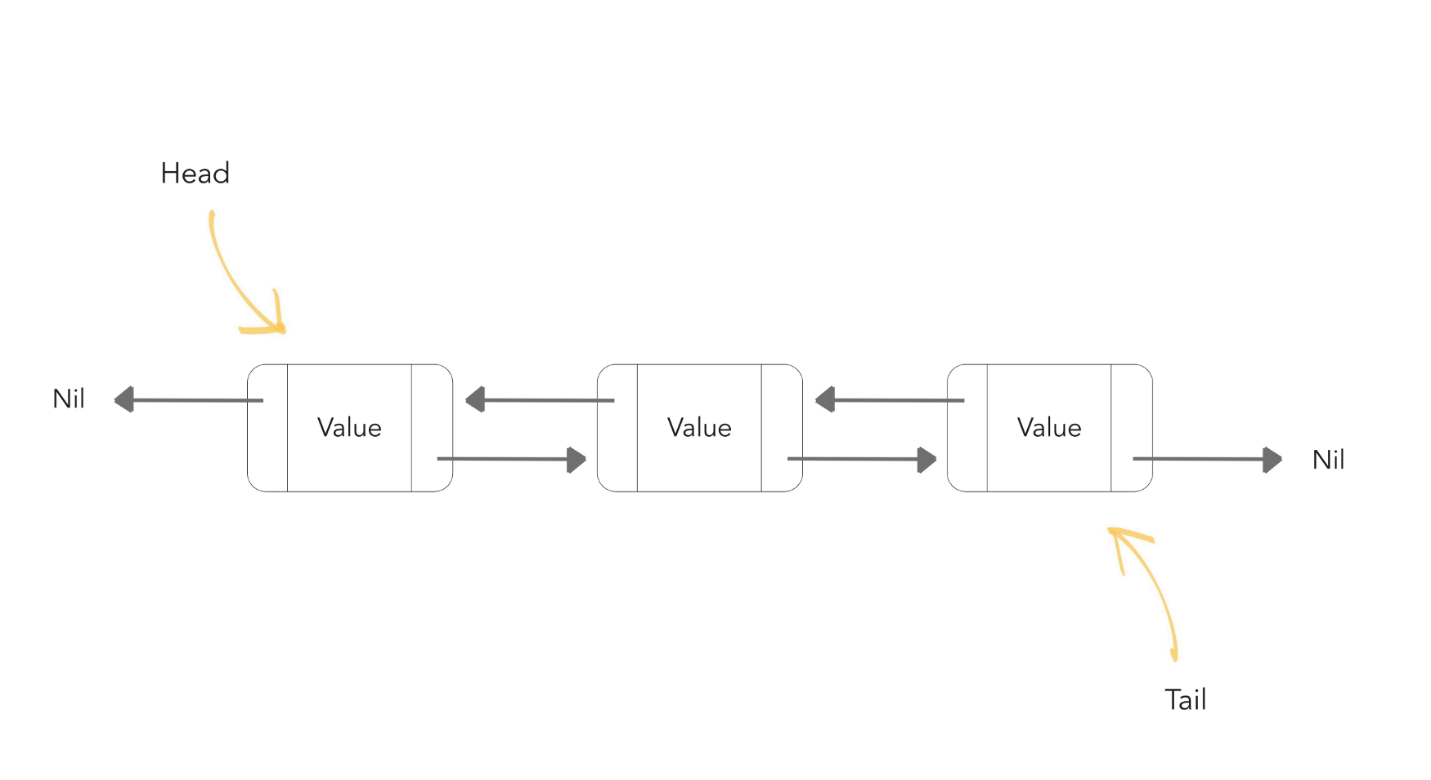
##### **Output:**

L1 Array list : [1, 2, 3, 4, 5, 3, 6, 3]

The last occurrence of 3 in l1 is at index : 7

### LinkedList in Java: Demo & Methods:

* The LinkedList class in Java provides us with the doubly linked list data structure.
* Each element of the linked list is known as a node.
* Each node points to the address of the next node & its previous node.



* Linked lists are preferred over the Array list because the insertion & deletion in the linked lists can be done in a constant time. But, in arrays, if we want to add or delete an element in between then, we need to shift all the other elements.
* In a linked list, it is impossible to directly access an element because we need to traverse the whole linked list to get the desired element.

#### ArrayList Vs. LinkedList:

Although ArrayList & LinkedList both implement the List interface and have the same methods, it is important to understand when to use which one.

* The insertion & deletion can be done in constant time in Linked List, so it is best to use the linked list when you need to add or remove elements frequently.
* Use ArrayList when you want to access the random elements frequently, as it can’t be done in a linked list in constant time.

#### Performing various operations on LinkedList:

##### **Adding Element in LinkedList:**

* + Similar to ArrayList, add() method is used to add elements in a linked list.
  + add(Object): Inserts an element at the end of the ArrayList.
  + add(Index,Object) : Inserts an element at the given index.Example :
  + import java.util.\*;
  + public class CWH extends Thread{
  + public static void main(String[] args) {
  + LinkedList<Integer> l1 = new LinkedList<>();
  + l1.add(11);
  + l1.add(22);
  + l1.add(33);
  + l1.add(44);
  + l1.add(55);
  + l1.add(77);
  + l1.add(5,77); // Inserts 77 at index 5
  + System.out.println("L1 Linked list : "+ l1);
  + }

}

##### **Output:**

L1 Linked list : [11, 22, 33, 44, 55, 77, 77]

##### **Removing an element from the LinkedList:**

* + remove() method is used to remove an element from the linked list.

#### Example:

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LinkedList<Integer> l1 = new LinkedList<>();

l1.add(11);

l1.add(22);

l1.add(33);

l1.add(44);

l1.add(55);

l1.add(77);

l1.add(5,77);

System.out.println("L1 Linked list before: "+ l1);

l1.remove(2); //removes element present at 2nd index

System.out.println("L1 Linked list after: " + l1);

}

}

##### **Output:**

L1 Linked list before: [11, 22, 33, 44, 55, 77, 77]

L1 Linked list after: [11, 22, 44, 55, 77, 77]

##### **Changing An Element Of Linked List:**

* + set() method is used to change an already existing element of a linked list.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LinkedList<Integer> l1 = new LinkedList<>();

l1.add(11);

l1.add(22);

l1.add(33);

l1.add(44);

l1.add(55);

l1.add(66);

System.out.println("L1 Linked list before: "+ l1);

l1.set(2,10); //changes element present at 2nd index (33 changed to 10)

System.out.println("L1 Linked list after: " + l1);

}

}

##### **Output:**

L1 Linked list before: [11, 22, 33, 44, 55, 66]

L1 Linked list after: [11, 22, 10, 44, 55, 66]

##### **Inserting an element at the last of the linked list:**

* + addlast() method is used to insert an element at the start of the linked list.

##### **Example:**

F. import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LinkedList<Integer> l1 = new LinkedList<>();

l1.add(11);

l1.add(22);

l1.add(33);

l1.add(44);

l1.add(55);

l1.add(66);

System.out.println("L1 Linked list before: "+ l1);

l1.addLast(100); //Inserting 100 at the end of L1

System.out.println("L1 Linked list after inserting element at last index: " + l1);

}

}

##### **Output:**

L1 Linked list before: [11, 22, 33, 44, 55, 66]

L1 Linked list after inserting element at last index: [11, 22, 33, 44, 55, 66, 100]

##### **Inserting an element at the start of the linked list:**

* + addFirst() method is used to insert an element at the start of the linked list.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LinkedList<Integer> l1 = new LinkedList<>();

l1.add(11);

l1.add(22);

l1.add(33);

l1.add(44);

l1.add(55);

l1.add(66);

System.out.println("L1 Linked list before: "+ l1);

l1.addFirst(0); //Inserting 0 at the starting of L1

System.out.println("L1 Linked list after: " + l1);

}

}

Output:

L1 Linked list before: [11, 22, 33, 44, 55, 66]

L1 Linked list after: [0, 11, 22, 33, 44, 55, 66]

package com.company;

import java.util.\*;

public class cwh\_92\_linkedlist {

public static void main(String[] args) {

LinkedList<Integer> l1 = new LinkedList<>();

LinkedList<Integer> l2 = new LinkedList<>();

l2.add(15);

l2.add(18);

l2.add(19);

l1.add(6);

l1.add(7);

l1.add(4);

l1.add(6);

l1.add(0, 5);

l1.add(0, 1);

l1.addAll(0, l2);

l1.addLast(676);

l1.addFirst(788);

System.out.println(l1.contains(27));

System.out.println(l1.indexOf(6));

System.out.println(l1.lastIndexOf(6));

//l1.clear();

l1.set(1, 566);

for(int i=0; i<l1.size(); i++){

System.out.print(l1.get(i));

System.out.print(", ");

}

}

}

### ArrayDeque in Java:

* ArrayDeque = Resizable array + Deque interface.
* ArrayDeque implements the Queue & Deque interface.
* There are no capacity restrictions for ArrayDeque, and it provides us the facility to add or remove any element from both sides of the queue.
* Also known as Array Double Ended Queue.
* It is faster than Linked list and stack.

#### Constructors of ArrayDeque class:

1. **ArrayDeque():** Used to create an empty array deque that has the capacity to hold 16 elements.
2. **ArrayDeque(int numElements):**Used to create an empty array deque that has the capacity to hold the specified number of elements.
3. **ArrayDeque(Collection<? extends E> c):**Used to create an array deque containing all the elements of the specified collections.

#### Performing Various Operation On ArrayDeque():

##### **Inserting an element :**

* + **Insertion at front**: add(), offerFirst() and addFirst() methods are used to insert an element at front of an array deque.

##### **Example :**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.addFirst(5);

ad1.offerFirst(10);

System.out.println(ad1);

}

}

##### **Output :**

[10, 5, 6, 56, 9]

* + **Insertion At End:** addLast() and offerLast() methods are used to insert an element at the end of the array deque.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.addLast(5);

ad1.offerLast(10);

System.out.println(ad1);

}

}

##### **Output :**

[6, 56, 9, 5, 10]

##### **Accessing an element:**

* + **Accessing an element from the head of the deque array:**getFirst() & peekFirst() methods are used to get the first element of the deque array.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.add(10);

ad1.add(91);

ad1.add(19);

System.out.println(ad1.getFirst());

System.out.println(ad1.peekFirst());

}

}

##### **Output:**

6

6

* + **Accessing the last element:**getLast() or peekLast() methods are used to print the last element of the deque array.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.add(10);

ad1.add(91);

ad1.add(19);

System.out.println(ad1.getLast());

System.out.println(ad1.peekLast());

}

}

##### **Output:**

19

19

##### **Removing an element:**

* + **Removing the first element:**removeFirst() & pollFirst() methods are used to delete an element from the head of the queue.
  + removeFirst() throws an exception if the queue is empty.
  + pollFirst() returns null if the queue is empty.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.add(10);

ad1.add(91);

ad1.add(19);

ad1.pollFirst(); //deletes 6

ad1.removeFirst(); //deletes 56

System.out.println(ad1);

}}

##### **Output:**

[9, 10, 91, 19]

* + **Removing the last element:**removeLast() & pollLast() methods are used to delete an element from the tail of the queue.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.add(10);

ad1.add(91);

ad1.add(19);

ad1.pollLast(); //deletes 19

ad1.removeLast(); //deletes 91

System.out.println(ad1);

}

}

##### **Output:**

[6, 56, 9, 10]

package com.company;

import java.util.ArrayDeque;

public class cwh\_93\_arraydeque {

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.addFirst(5);

System.out.println(ad1.getFirst());

System.out.println(ad1.getLast());

}

}

package com.company;

import java.util.ArrayDeque;

public class cwh\_93\_arraydeque {

public static void main(String[] args) {

ArrayDeque<Integer> ad1 = new ArrayDeque<>();

ad1.add(6);

ad1.add(56);

ad1.add(9);

ad1.addFirst(5);

System.out.println(ad1.getFirst());

System.out.println(ad1.getLast());

}

}

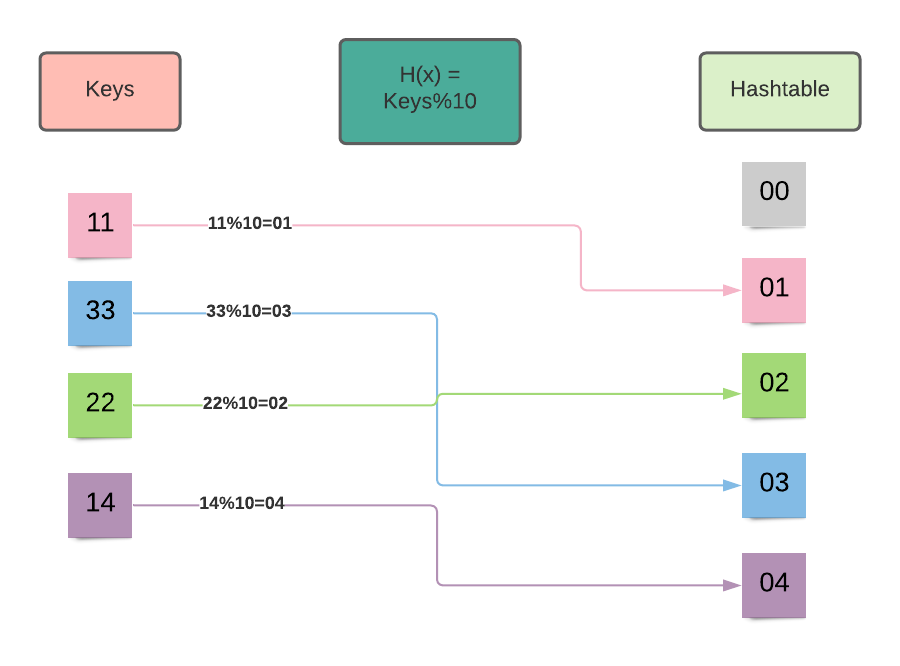
### Hashing in Java:

Hashing is the technique to convert the range of key-value pairs to a range of indices. In hashing, we use hash functions to map keys to some values.

##### **Example:**

Let arr=[11,33,22,14]

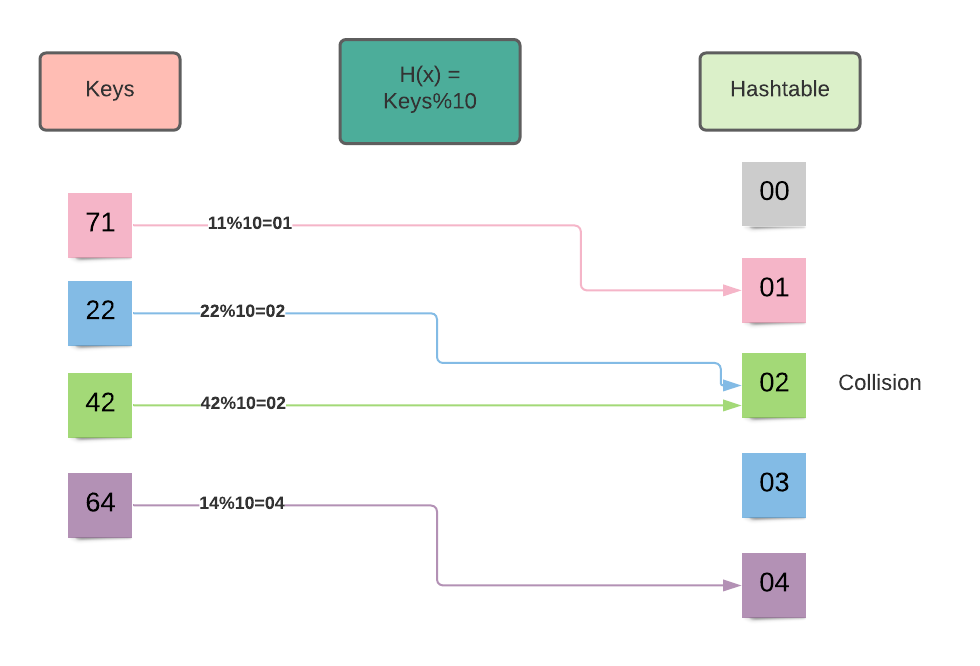
hashIndex = (key %10)



Collision: The hash function may map two key values to a single index. Such a situation is known as a collision.

Example : Let 1l=[22,42,64,71]

H(x) = keys%10



In the above image, you can see that the 22 and 44 are mapped to the index number 2. Therefore we need to avoid the collision. Following techniques are used to avoid collision in hashing :

* Open addressing
* Chaining

### HashSet in Java:

* HashSet class uses a hash table for storing the elements.
* It implements the set interface.
* Duplicate values are not allowed.
* Before storing any object, the hashset uses the hashCode() and equals() method to check any duplicate entry in the hash table.
* Allows null value.
* Best suited for search operations.

#### Constructors Of HashSet:

1. **HashSet():** This constructor is used to create a new empty HashSet that can store 16 elements and have a load factor of 0.75.
2. **HashSet(int initialCapacity):** This constructor is used to create a new empty HashSet which has the capacity to store the specified number of elements and having a load factor of 0.75.
3. **HashSet(int initialCapacity, float loadFactor):**This constructor is used to create a new empty HashSet with the capacity & load factor equal to specified integer and float value.
4. **HashSet(Collection<? extends E> c):** This constructor is used to create a HashSet using the elements of collection c.

#### Performing Various Operations On HashSet:

##### **Inserting elements:**

* + add() method is used to add elements in HashSet.
  + The insertion order of the elements does not remains preserved in HashSet.
  + All the duplicate elements are ignored because the set contains only unique values.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

myHashSet.add(6);

myHashSet.add(8);

myHashSet.add(3);

myHashSet.add(11);

myHashSet.add(11); // This element will be ignored

System.out.println(myHashSet);

}}

##### **Output:**

[8, 3, 11, 6]

##### **Removing elements from the HashSet:**

* + remove() method is used to delete the specified element from the HashSet.
  + This method does not throws any exception if the specified element is not present in the HashSet.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

myHashSet.add(6);

myHashSet.add(8);

myHashSet.add(3);

myHashSet.add(11);

myHashSet.add(11); // This element will be ignored

System.out.println("myHashSet before removing any element : " + myHashSet);

myHashSet.remove(3); //deletes 3 from the hashset

System.out.println("myHashSet after removing a element : " + myHashSet);

}

}

##### **Output:**

myHashSet before removing any element : [8, 3, 11, 6]

myHashSet after removing a element : [8, 11, 6]

##### **Checking if the HashSet is empty or not:**

* + - isEmpty() method is used to check if there is any object in the HashSet or not.
    - This method returns a boolean value.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

myHashSet.add(6);

myHashSet.add(8);

myHashSet.add(3);

myHashSet.add(11);

HashSet<Integer> myHashSet1 = new HashSet<>();

System.out.println(myHashSet.isEmpty());

System.out.println(myHashSet1.isEmpty());

}

}

##### **Output:**

false

true

##### **Removing all the elements from the HashSet:**

* + clear() method is used to remove all the elements from the HashSet at once.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

myHashSet.add(16);

myHashSet.add(33);

myHashSet.add(78);

myHashSet.add(19);

myHashSet.add(29);

myHashSet.add(10);

System.out.println("myHashSet before : " + myHashSet);

myHashSet.clear(); //deletes all the elements from the hashset

System.out.println("myHashSet after : " + myHashSet);

}

}

##### **Output:**

myHashSet before : [16, 33, 19, 10, 29, 78]

myHashSet after : []

##### **Printing the size of the HashSet:**

* + size() method is used to get the size of the HashSet.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

myHashSet.add(16);

myHashSet.add(33);

myHashSet.add(78);

myHashSet.add(19);

myHashSet.add(29);

myHashSet.add(10);

System.out.println("The size of myHashSet is : " + myHashSet.size());

}

}

##### **Output:**

The size of myHashSet is : 6

#### Code as described/written in the video:

package com.company;

import java.util.HashSet;

public class cwh\_95\_set {

public static void main(String[] args) {

HashSet<Integer> myHashSet = new HashSet<>(6, 0.5f);

myHashSet.add(6);

myHashSet.add(8);

myHashSet.add(3);

myHashSet.add(11);

myHashSet.add(11);

System.out.println(myHashSet);

}

}

### Date and Time in Java:

java time -> package for date & time in java from java onwards

Before java 8, java util package used to hold the date time class now these classes are deprecated

### How java stores a Date?

Date in java is stored in the form of a long number. This long number holds the number of milliseconds passed since 1 January 1970

Java assumes that 1900 is the start year which means it calculates years passed since 1900 whenever We ask it for years passed

System current Time Millis () returns no of seconds passed Once number of milliseconds are calculated, we can calculate minutes, seconds & years passed.

package com.company;

public class cwh\_96\_date {

public static void main(String[] args) {

System.out.println(System.currentTimeMillis()/1000/3600/24/365);

}

}

### The Date Class in Java:

##### **Question: Is it safe to store the number of milliseconds in a variable of type long?**

**Answer:** Yes, it is absolutely safe to store the number of milliseconds in a variable of type long because the maximum value that can be stored in long is 9223372036854775807. You can see that the maximum value of long is huge. Therefore, we do not need to worry about the value of milliseconds. Notice the output of the below code; the value of current time in millisecond is 10^6 times smaller than the maximum value of long data type.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

System.out.println("The maximum value of long is :" + Long.MAX\_VALUE);

System.out.println("The value of current time in ms : " + System.currentTimeMillis());

}

}

##### **Output:**

The maximum value of long is :9223372036854775807

The value of current time in ms : 1621708466975

#### Date Class in Java:

* -Date class in java is available in java.util package.
* This class provides the instant in time with precision of millisecond.

#### Constructors of the date class:

1. **Date():**This constructor is used when we need an object of current date and time.
2. **Date(long milliseconds):** This constructor creates a date object from the number of milliseconds passed since January 1, 1970.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Date d= new Date();

System.out.println("The current date is : " + d);

Date d1= new Date(1621709639111l);

System.out.println("The date calculated form miliseconds is : " + d1);

}

}

##### **Output:**

The current date is : Sun May 23 00:24:17 IST 2021

The date calculated form miliseconds is : Sun May 23 00:23:59 IST 2021

#### Methods of date class:

##### **compareTo() :**

* + Checks for the equality of the two dates.
  + Returns 0 if the dates are equal; else, returns 1.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Date d= new Date();

Date d1= new Date(2021,12, 24); //both dates are different

System.out.println(d1.compareTo(d));

}

}

##### **Output:**

1

##### **getTime() :**

* + This method returns the number of milliseconds passed since the midnight of January 1, 1970.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Date d= new Date(2021,5,23);

System.out.println("The number of milliseonds passed since Jan 1, 1970 :" +d.getTime() );

}

}

##### **Output:**

The number of milliseonds passed since Jan 1, 1970 :61582530600000

1. getYear(): Prints the current year.  
   getDate(): Prints the current date.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Date d= new Date();

System.out.println("The current date is : "+ d.getDate());

System.out.println("The current year is : "+ d.getYear()); //19

}

}

##### **Output:**

The current date is : 23

The current year is : 121

package com.company;

import java.util.Date;

public class cwh\_97\_date\_class {

public static void main(String[] args) {

// System.out.println(Long.MAX\_VALUE);

// System.out.println(System.currentTimeMillis());

Date d = new Date();

System.out.println(d);

System.out.println(d.getTime());

System.out.println(d.getDate());

System.out.println(d.getSeconds());

System.out.println(d.getYear());

}

}

### Calendar Class in Java:

* The calendar class in java provides the methods that helps in converting date between a specific instant in time.
* It is an abstract class.
* Since it is an abstract class, we cannot create an instance of this class with the help of a constructor.
* We use the static method Calender.getInstance() in order to implement a sub-class.

##### **Example to demonstrate the getInstance() method :**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println(c.getCalendarType()); //getCalendarType() returns the type of the calendar

}

}

##### **Output:**

gregory

#### Constructors of the Calendar class:

1. **Calendar():**This constructor is used to construct a calendar with the default time zone & locale.
2. **Calendar(Time zone, Locale locale):**This constructor is used to construct a calendar with the specified time zone & locale.

#### Methods of the Calendar class:

##### **get(int field) :**

* + This method returns the value of the specified calendar field.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println("Current year is :"+ c.get(Calendar.YEAR));

System.out.println("Current month is :"+ c.get(Calendar.MONTH)); //The indexing for month field ranges from [0,11]

System.out.println("Current day is :"+ c.get(Calendar.DAY\_OF\_WEEK));

System.out.println("Current hour is :"+ c.get(Calendar.HOUR\_OF\_DAY));

System.out.println("Current minute is :"+ c.get(Calendar.MINUTE));

System.out.println("Current second is :"+ c.get(Calendar.SECOND));

}

}

##### **Output:**

Current year is :2021

Current month is :4

Current day is :1

Current hour is :12

Current minute is :3

Current second is :3

##### **add(int field, int amount):**

* + This method is useful for calculating the time after or before of a specified calendar field.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println("Current date is : " + c.getTime());

c.add(Calendar.YEAR, 4);

System.out.println("After 4 years : "+ c.getTime());

c.add(Calendar.YEAR, -12);

System.out.println("Before 12 years : "+ c.getTime());

c.add(Calendar.MONTH,2);

System.out.println("After 2 months : "+ c.getTime());

}

}

##### **Output:**

Current date is : Sun May 23 12:14:24 IST 2021

After 4 years : Fri May 23 12:14:24 IST 2025

Before 12 years : Thu May 23 12:14:24 IST 2013

After 2 months : Tue Jul 23 12:14:24 IST 2013

##### **getWeeksInWeekYear():**

* + Returns the number of weeks.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println(c.getWeeksInWeekYear());

}

}

##### **Output:**

52

##### **getMaximum(int field):**

* + Returns the maximum value for the specified calendar field.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println(" The maximum no. of weeks in a year : " + c.getMaximum(Calendar.WEEK\_OF\_YEAR));

}

}

##### **Output:**

53

### GregorianCalendar class & TimeZone in java:

* GregorianCalendar class is the concrete sub-class of the Calendar class.
* This class supports both the Julian and Gregorian calendar systems.

#### Difference between Calendar and GregorianCalendar class:

The calendar class is an abstract class. So, the instance of this class cannot be instantiated. Therefore, we need to use the static method Calendar.getInstance() to initialize the object of the Calendar class:

Calendar c = Calendar.getInstance();

Since the GregorianCalendar class is a concrete subclass, it can be initialized as:

GregorianCalendar gcal = new GregorianCalendar();

#### Constructors of the GregorianCalendar class:

1. **GregorianCalendar():** This constructor is used to initialize an object with the current time in the default time zone.
2. **GregorianCalendar(int year, int month, int day):**This constructor is used to initialize an object with the date-set specified as parameters in the default time zone and default locale.
3. **GregorianCalendar(int year, int month, int day, int hours, int minutes):**This constructor initializes an object with the given date and time set in the default locale and time zone.
4. **GregorianCalendar(int year, int month, int day, int hours, int minutes, int seconds):**This constructor initializes an object with the more specific time and date-set passed as a parameter in the default locale and time zone.
5. **GregorianCalendar(Locale locale):**Initializes a GregorianCalendar object with the current date and time in the default time zone and the specified locale.
6. **GregorianCalendar(TimeZone timeZone):**Initializes a GregorianCalendar object with the current date and time in the default locale and the specified time zone.
7. **GregorianCalendar(TimeZone timeZone, Locale locale):**Initializes an object with the locale and timezone passed as parameters.

#### Methods of the Gregorian class:

##### **isLeapYear(int year):**

* + Checks if the year passed as a parameter is a leap year or not.
  + This method returns a boolean value.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

GregorianCalendar cal = new GregorianCalendar();

System.out.println(cal.isLeapYear(2000));

System.out.println(cal.isLeapYear(2021));

}

}

##### **Output:**

true

false

##### **roll(int field, boolean up) :**

* + - This method adds/subtracts a single unit of time from the specified time field.
    - true = rolls up the value by 1.
    - false = rolls down the value by 1.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

GregorianCalendar c = new GregorianCalendar();

System.out.println("Date before rolling : " + c.getTime());

c.roll(Calendar.MONTH, true);

c.roll(Calendar.DATE, false);

c.roll(Calendar.YEAR, true);

System.out.println("Date after rolling : " + c.getTime());

}

}

##### **Output:**

Date before rolling : Wed May 26 07:53:24 IST 2021

Date after rolling : Sat Jun 25 07:53:24 IST 2022

##### **hashcode():**

* + - This method returns the hashcode of the calendar object.

##### **Example:**

import java.util.\*;

public class CWH extends Thread{

public static void main(String[] args) {

GregorianCalendar c = new GregorianCalendar();

System.out.println("Calendar : " + c.getTime());

System.out.println("The hashcode for this calendar is : "+ c.hashCode());

}

}

##### **Output:**

Calendar : Wed May 26 08:08:33 IST 2021

The hashcode for this calendar is : 1358707903

package com.company;

import java.util.Calendar;

import java.util.GregorianCalendar;

import java.util.TimeZone;

public class cwh\_99\_gregorian {

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println(c.getTime());

System.out.println(c.get(Calendar.DATE));

System.out.println(c.get(Calendar.SECOND));

System.out.println(c.get(Calendar.HOUR));

System.out.println(c.get(Calendar.HOUR\_OF\_DAY) + ":" + c.get(Calendar.MINUTE) + ":" + c.get(Calendar.SECOND));

GregorianCalendar cal = new GregorianCalendar();

System.out.println(cal.isLeapYear(2018));

System.out.println(TimeZone.getAvailableIDs()[0]);

System.out.println(TimeZone.getAvailableIDs()[1]);

System.out.println(TimeZone.getAvailableIDs()[2]);

}

}

### java.time API - Classes & Methods:

Date and time features in Java is primarily supported by two packages:

* java.util
* java.time

The package java.time was added with the release of Java 8 with the aim of solving problems faced by Java developers while handling date and time with java.util package such as representing a date without time, etc.

#### Classes of Java.time:

##### **Clock class:**

* + This class provides access to the current instant, date and time zone using a time-zone.
  + Clock class is an abstract class therefore it is not possible to create instance of the clock class.

##### **Some methods of the clock class :**

* + **abstract ZoneId getZone() :** This methods returns the time zone being used to create date and time objects.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Clock cl = Clock.systemDefaultZone();

System.out.println(cl.getZone());

}

}

##### **Output:**

Asia/Calcutta

* + **abstract Instant instant() :** This methods returns the current instant of the clocks.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Clock cl = Clock.systemUTC();

System.out.println(cl.instant());

}

}

##### **Output:**

2021-05-26T06:43:05.064640700Z

##### **Duration class:**

* + This class is used to measure time in seconds and nano seconds.
  + This class is immutable.

##### **Some Methods of the duration class:**

* + **boolean isNegative() :**This method is used to check if the duration is negative.  
    Example :
  + import java.time.\*;
  + public class CWH extends Thread{
  + public static void main(String[] args) {
  + Duration d1 = Duration.between(LocalTime.MIN,LocalTime.NOON); //LocalTime.MIN = '00:00' , LocalTime.NOON = '12:00'
  + System.out.println(d1.isNegative());
  + Duration d2 = Duration.between(LocalTime.MAX,LocalTime.MIN); //LocalTime.MAX = '23:59:59.999999999' , LocalTime.MIN = '00:00'
  + System.out.println(d2.isNegative());
  + }
  + }

##### **Output:**

false

true

* + **isZero() :** This method is used to check if the duration is zero. Returns boolean value.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

Duration d1 = Duration.between(LocalTime.MIN,LocalTime.MIDNIGHT); //LocalTime.MIN = '00:00' , LocalTime.NOON = '00:00'

System.out.println(d1.isZero());

Duration d2 = Duration.between(LocalTime.MAX,LocalTime.MIN); //LocalTime.MAX = '23:59:59.999999999' , LocalTime.MIN = '00:00'

System.out.println(d2.isZero());

}

}

##### **Output:**

true

false

##### **LocalDate class:**

* + This class is useful for representing the dates in the year-month-day format.
  + With the help of LocalDate class, dates can be represented without time.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDate d = LocalDate.now();

System.out.println(d);

}

}

##### **Output:**

2021-05-26

##### **Some methods of the LocalDate class:**

* + **compareTo() :**This method compares the equality of the two dates. Returns boolean value.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDate d = LocalDate.parse("2021-05-27");

LocalDate d1= LocalDate.parse("2021-05-26");

LocalDate d2= LocalDate.parse("2021-05-26");

System.out.println(d1.equals(d));

System.out.println(d2.equals(d1));

}

##### **Output:**

false

true

* + **withYear(int Year) :** This method returns a copy of the LocaleDate but alters the year with the value of year passed as argument.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDate d = LocalDate.parse("2021-05-27");

System.out.println(d.withYear(2001));

}

}

##### **Output:**

2001-05-27

##### **LocalTime class:**

* + This class helps us to represent the time without the dates.
  + Instances of LocalTime class are mutable.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LocalTime t = LocalTime.now();

System.out.println(t);

}

}

##### **Output:**

13:13:36.198479100

##### **Some methods of the LocalTime class:**

* + **LocalTime plusHours(long hoursToAdd) :**This method returns a copy of the LocalTime but with the specified number of hours added.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LocalTime t = LocalTime.of(13,18,29);

System.out.println("Time before : " + t);

LocalTime t1= t.plusHours(5);

System.out.println("Time after adding 5 hours : " + t1);

}

}

##### **Output:**

Time before : 13:18:29

Time after adding 5 hours : 18:18:29

* + **LocalTime minusMinutes(long minutesToSubtract) :**This method returns a copy of the LocalTime but with the specified number of minutes subtracted.

##### **Example:**

import java.time.\*;

public class CWH extends Thread{

public static void main(String[] args) {

LocalTime t = LocalTime.of(15,28,19);

System.out.println("Time before : " + t);

LocalTime t1= t.minusMinutes(8);

System.out.println("Time after subtracting 8 minutes : " + t1);

}

}

##### **Output:**

Time before : 15:28:19

Time after subtracting 8 minutes : 15:20:19

#### Code as described/written in the video:

package com.company;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

public class cwh\_100\_java\_time {

public static void main(String[] args) {

LocalDate d = LocalDate.now();

System.out.println(d);

LocalTime t = LocalTime.now();

System.out.println(t);

LocalDateTime dt = LocalDateTime.now();

System.out.println(dt);

}

}

### DateTimeFormatter in Java:

* This class helps us to print and parse date and time in our desired format.
* The format() method of the DateTimeFormatter class is used to format the dates using our desired format.

##### **Syntax:**

public String format(DateTimeFormatter formatter)

##### **Parameter:**

The object of the formatter to be used is passed, and it can not be null.

##### **Exception:**

This method throws DateTimeException.

##### **Return Value:**

Returns the string in the format specified by the user.

##### **Example:**

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDateTime dt = LocalDateTime.now(); System.out.println("The current date is : " + dt);

DateTimeFormatter df = DateTimeFormatter.ofPattern("dd.MM.yyyy"); // This is the format

String myDate = dt.format(df);

// Creating date string using date and format

System.out.println("Date after formattin : "+ myDate);

}

}

##### **Output:**

The current date is : 2021-05-26T18:15:42.554864400

Date after formattin : 26.05.2021

* In addition to the format, formatters can be created with desired Locale, Chronology, ZoneId, and DecimalStyle.

#### Programs to illustrate some of the predefined formatters of the DateTimeFormatter class:

##### **ISO\_LOCAL\_DATE:**

* + Formats the date according to the International Standard for the representation of dates.

##### **Example:**

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDateTime dt = LocalDateTime.now();

DateTimeFormatter df = DateTimeFormatter.ISO\_LOCAL\_DATE;

// Formatting the date in the ISO format

String myDate = dt.format(df);

// Creating date string using date and format

System.out.println("Date in ISO format : "+ myDate);

}

}

##### **Output:**

Date in ISO format : 2021-05-26

##### **ISO\_WEEK\_DATE:**

* + - Returns the number of weeks and year.

##### **Example:**

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDateTime dt = LocalDateTime.now();

DateTimeFormatter df =DateTimeFormatter.ISO\_WEEK\_DATE;

String myDate = dt.format(df);

System.out.println("Date in ISO\_WEEK\_DATE Format : "+ myDate);

}

}

##### **Output:**

Date in ISO\_WEEK\_DATE Format : 2021-W21-3

##### **ISO\_ORDINAL\_DATE:**

* + - Returns the year and day of the year.

##### **Example:**

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CWH extends Thread{

public static void main(String[] args) {

LocalDateTime dt = LocalDateTime.now();

DateTimeFormatter dt = DateTimeFormatter.ISO\_ORDINAL\_DATE;

String myDate = dt.format(df);

System.out.println("Date in ISO\_WEEK\_DATE Format : "+ myDate);

}

}

##### **Output:**

Date in ISO\_ORDINAL\_DATE Format : 2021-146**Patterns for formatting and parsing :**

Pattern letters for all the alphabets(capital as well as small) are defined as follows:

Symbol Meaning Presentation Examples

------ ------- ------------ -------

G era text AD; Anno Domini; A

u year year 2004; 04

y year-of-era year 2004; 04

D day-of-year number 189

M/L month-of-year number/text 7; 07; Jul; July; J

d day-of-month number 10

Q/q quarter-of-year number/text 3; 03; Q3; 3rd quarter

Y week-based-year year 1996; 96

w week-of-week-based-year number 27

W week-of-month number 4

E day-of-week text Tue; Tuesday; T

e/c localized day-of-week number/text 2; 02; Tue; Tuesday; T

F week-of-month number 3

a am-pm-of-day text PM

h clock-hour-of-am-pm (1-12) number 12

K hour-of-am-pm (0-11) number 0

k clock-hour-of-am-pm (1-24) number 0

H hour-of-day (0-23) number 0

m minute-of-hour number 30

s second-of-minute number 55

S fraction-of-second fraction 978

A milli-of-day number 1234

n nano-of-second number 987654321

N nano-of-day number 1234000000

V time-zone ID zone-id America/Los\_Angeles; Z; -08:30

z time-zone name zone-name Pacific Standard Time; PST

O localized zone-offset offset-O GMT+8; GMT+08:00; UTC-08:00;

X zone-offset 'Z' for zero offset-X Z; -08; -0830; -08:30; -083015; -08:30:15;

x zone-offset offset-x +0000; -08; -0830; -08:30; -083015; -08:30:15;

Z zone-offset offset-Z +0000; -0800; -08:00;

p pad next pad modifier 1

' escape for text delimiter

'' single quote literal '

[ optional section start

] optional section end

# reserved for future use

{ reserved for future use

} reserved for future use

#### Code as described/written in the video:

package com.company;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class cwh\_101\_datetimeformatter {

public static void main(String[] args) {

LocalDateTime dt = LocalDateTime.now(); // This is the date

System.out.println(dt);

DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy -- E H:m a"); // This is the format

DateTimeFormatter df2 = DateTimeFormatter.ISO\_LOCAL\_DATE;

String myDate = dt.format(df); // Creating date string using date and format

System.out.println(myDate);

}

}

### Advanced Java Practice Set:

**Question 1:**Create an ArrayList and store the names of ten students inside it. Print it using a for each loop.

**Answer 1:**As we discussed in the[ArrayList tutorial](https://www.codewithharry.com/videos/java-tutorials-for-beginners-91)of this playlist, add() method is used to insert element in an ArrayList. Below is the required program:

import java.util.ArrayList;

public class CWH{

public static void main(String[] args) {

ArrayList ar = new ArrayList();

ar.add("Student 1");

ar.add("Student 2");

ar.add("Student 3");

ar.add("Student 4");

ar.add("Student 5");

ar.add("Student 6");

ar.add("Student 7");

ar.add("Student 8");

ar.add("Student 9");

ar.add("Student 10");

for(Object o: ar){

System.out.println(o);

}

}

}

##### **Output:**

Student 1

Student 2

Student 3

Student 4

Student 5

Student 6

Student 7

Student 8

Student 9

Student 10

**Question 2:**Use the Date class in Java to print the time in the following format: **21:47:02.**

**Answer 2:** In the[Date class tutorial](https://www.codewithharry.com/videos/java-tutorials-for-beginners-97), we saw that how get() method can be used to print the dates in our desired format. Below is required program :

Date d = new Date();

System.out.println(d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds());

##### **Output:**

19:13:17

**Question 3:** Repeat question number 2 using the Calendar class.

**Answer 3:**In the [Calendar class tutorial](https://www.codewithharry.com/videos/java-tutorials-for-beginners-98), we saw how we can use the Calendar.getInstance() static method to initialize an object of the Calendar class. After creating object, use the get() method to print the date in the desired formate. Below is the required program :

import java.util.Calendar;

import java.util.Date;

public class CWH{

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println(c.get(Calendar.HOUR\_OF\_DAY) + ":" + c.get(Calendar.MINUTE) + ":" + c.get(Calendar.SECOND));

}

}

##### **Output:**

19:15:26

**Question 4:** Repeat question number 2 using java.time API.

**Answer 4:**In the[DateTimeFormatter](https://www.codewithharry.com/videos/java-tutorials-for-beginners-101) class tutorial, I told you to use the format() method in order to change the date and time in the desired format.

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CWH{

public static void main(String[] args) {

LocalDateTime dt = LocalDateTime.now(); // This is the date

DateTimeFormatter df = DateTimeFormatter.ofPattern("H:m:s"); // This is the format

String myDate = dt.format(df); // Creating date string using date and format

System.out.println(myDate);

}

}

##### **Output:**

19:27:59

**Question 5:**Create a Set in java. Try to store the duplicate values elements inside this set and verify that only one instance is stored.

**Answer** **5:** In the [Hashset tutorial](https://www.codewithharry.com/videos/java-tutorials-for-beginners-95), we saw that only unique elements can be stored inside a Hashset. Below is the required code:

import java.util.HashSet;

public class CWH{

public static void main(String[] args) {

HashSet<Integer> s = new HashSet();

s.add(5);

s.add(6);

s.add(46);

s.add(60);

s.add(9);

s.add(6);

System.out.println(s);

}

}

##### **Output:**

[5, 6, 9, 60, 46]

### EXERCISE: CUSTOM CALCULATOR:

Exercise: You have to create a custom calculator with following operations:

1. + -> Addition

2. - -> Subtraction

3. \* -> Multiplication

4. / -> Division

which throws the following exceptions:

1. Invalid input Exception ex: 8 & 9

2. Cannot divide by 0 Exception

3. Max Input Exception if any of the inputs is greater than 100000

4. Max Multiplier Reached Exception - Don't allow any multiplication input to be greater than 7000

### 

### Solution: Custom Calculator:

### solution of exercise:

package com.company;

class InvalidInputException extends Exception{

@Override

public String toString() {

return "Cannot add 8 and 9";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class MaxInputException extends Exception{

@Override

public String toString() {

return "Input cant be greater than 100000";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class CannotDivideByZeroException extends Exception{

@Override

public String toString() {

return "Cannot divide by 0";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class MaxMultiplyInputException extends Exception{

@Override

public String toString() {

return "Input cant be greater than 7000 while multiplying";

}

@Override

public String getMessage() {

return "I am getMessage()";

}

}

class CustomCalculator {

double add(double a, double b) throws InvalidInputException, MaxInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

if(a==8 || b==9) {

throw new InvalidInputException();

}

return a + b;

}

double subtract(double a, double b) throws MaxInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

return a - b;

}

double multiply(double a, double b)throws MaxInputException, MaxMultiplyInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

else if(a>7000 || b>7000){

throw new MaxMultiplyInputException();

}

return a \* b;

}

double divide(double a, double b) throws CannotDivideByZeroException, MaxInputException{

if(a>100000 || b>100000){

throw new MaxInputException();

}

if(b==0){

throw new CannotDivideByZeroException();

}

return a / b;

}

}

public class cwh\_103\_ex6sol {

public static void main(String[] args) throws InvalidInputException,

CannotDivideByZeroException, MaxInputException, MaxMultiplyInputException {

CustomCalculator c = new CustomCalculator();

// c.add(8, 9);

// c.divide(6, 0);

// c.divide(600000000, 40);

c.multiply(5, 9888);

}

}

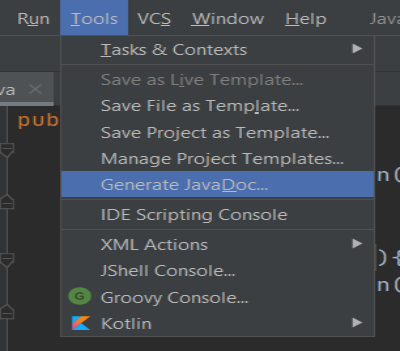
**Generating our own JavaDocs for our Package:**

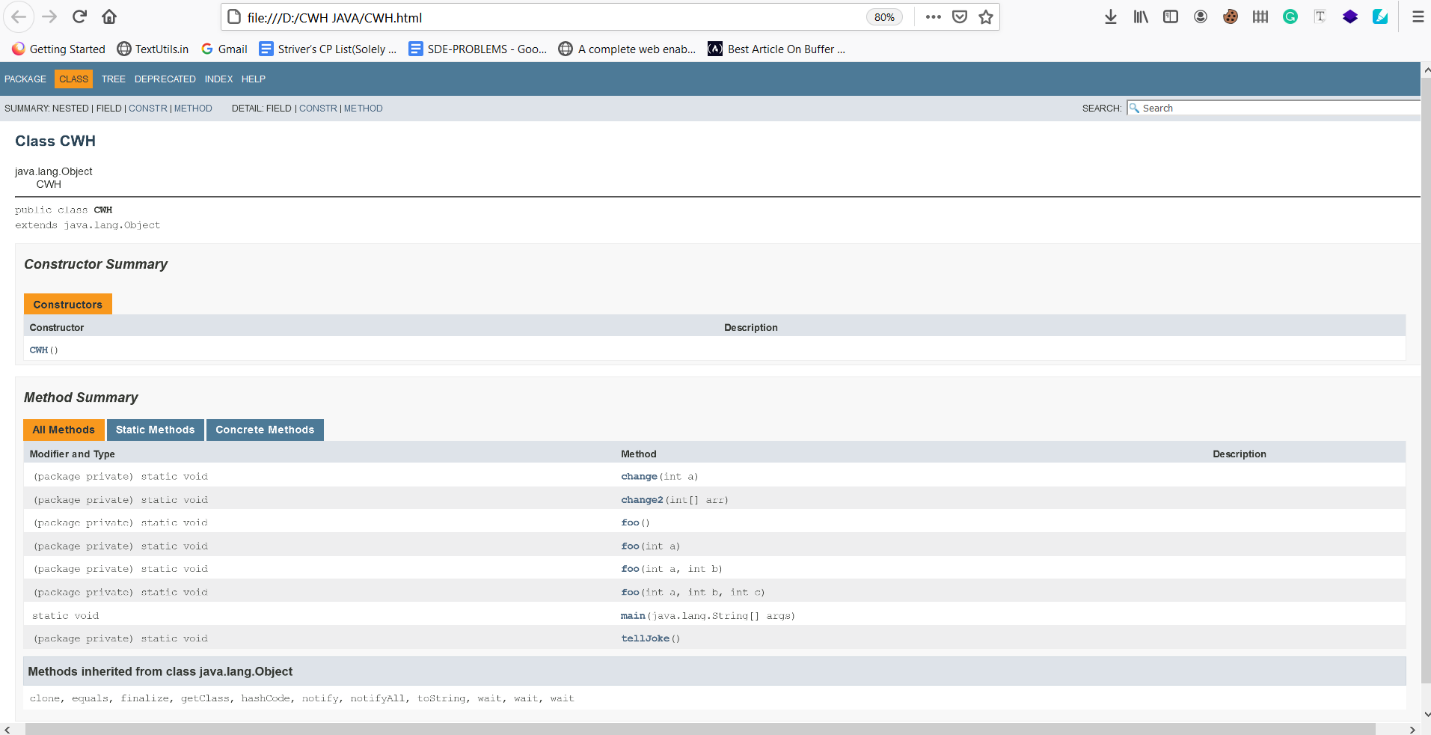
 A reliable documentation is a must for a developer. Just imagine how difficult it would be to write code in a language you are not familiar with, without proper documentation. Documentation helps us to develop, maintain and transfer knowledge to other developers. As you are already aware that there is a proper documentation available for Java. But have you ever thought of creating your own documentation? In this tutorial, I will tell you how you can automatically generate documentation for your Java packages with the help of JavaDoc.

What is JavaDoc?

* It is a tool that automatically generates standard documentation in HTML format from the Java source code.
* This tool comes built-in with JDK( Java Development Kit).

Steps to create documentation with JavaDoc in IntellijIdea:

1. Open your java program in IntellijIdea.
2. Click on **Tools**and then select **Generate JavaDoc**.  
   
3. A pop window will open. Select the project and packages for which you want to create the JavaDoc.
4. Select the classes for which you want to generate the documentation. By default, documentation will be created for all the classes.
5. Select the location where you want to save your JavaDoc by clicking on the **Output directory**.
6. Click on **Ok,**and your documentation will be saved to your specified location.

Screenshot of the generated JavaDoc:  


This is all for this tutorial, and in the next tutorial, we will deal with the tags for documenting classes in JavaDocs.

Code as described/written in the video:

package com.company;

public class cwh\_105\_javadoc {

static void foo(){

System.out.println("Good Morning bro!");

}

static void foo(int a){

System.out.println("Good morning " + a + " bro!");

}

static void foo(int a, int b){

System.out.println("Good morning " + a + " bro!");

System.out.println("Good morning " + b + " bro!");

}

static void foo(int a, int b, int c){

System.out.println("Good morning " + a + " bro!");

System.out.println("Good morning " + b + " bro!");

}

static void change(int a){

a = 98;

}

static void change2(int [] arr){

arr[0] = 98;

}

static void tellJoke(){

System.out.println("I invented a new word!\n" +

"Plagiarism!");

}

public static void main(String[] args) {

// tellJoke();

// Case 1: Changing the Integer

//int x = 45;

//change(x);

//System.out.println("The value of x after running change is: " + x);

// Case 1: Changing the Array

// int [] marks = {52, 73, 77, 89, 98, 94};

// change2(marks);

// System.out.println("The value of x after running change is: " + marks[0]);

// Method Overloading

foo();

foo(3000);

foo(3000, 4000);

// Arguments are actual

}}

### Javadocs: Tags for Documenting Classes:

#### Below is the list of the JavaDoc tags:

|  |  |  |
| --- | --- | --- |
| **Tag** | **Syntax** | **Description** |
| @author | @author name-text | Describes the author of a class. |
| @version | @version version-number | Adds a "Version" heading which specifies the current version  of the release or file. |
| @since | @since release-date | Adds a "Since" heading that tells about the release date. |
| @see | @see <a href="reference"></a> | Adds a "See Also" heading that refers to the other element  of the documentation. |
| @return | @return return-description | Adds a "Return" description that tells about the return  value of the method. |
| @param | @param param-description | Provides the information about the method parameters in  the "Parameters" section. |
| @throws | @exception exception-name description | Displays the exception that can be thrown by a method  (same as @exception) |
| {@code} | {@code text} | Displays text in code font without interpreting the text as  HTML markup or nested javadoc tags. |
| @deprecated | @deprecated deprecatedtext | Adds a "Deprecated" heading indicating that this API  should no longer be used. |

#### Comments In JavaDoc:

Like Java programs, we can also include comments in Java documentation for a better understanding. A JavaDoc comment is known as doc comment in general.

##### **Syntax:**

/\*\*Documentation comment \*/

#### Including HTML inside the JavaDoc:

You can include HTML tags in the JavaDoc. Example:

/\*\*

\* <i>This is a simple documentation to show that HTML elements can be included in JavaDoc.</i>

\* @see <a href="https://www.codewithharry.com/videos/java-tutorials-for-beginners-1">Best Java Course Available For <b>Free</b></a>

\* \*/

public class CWH {

/\*\* This method is used to return the divison of two numbers

\* @throws ArithmeticException if divided by 0

\* @return Integer

\* @param a First parameter - Integer

\* @param b Second parameter - Integer

\* \*/

public int divison(int a, int b){

return a/b;

}

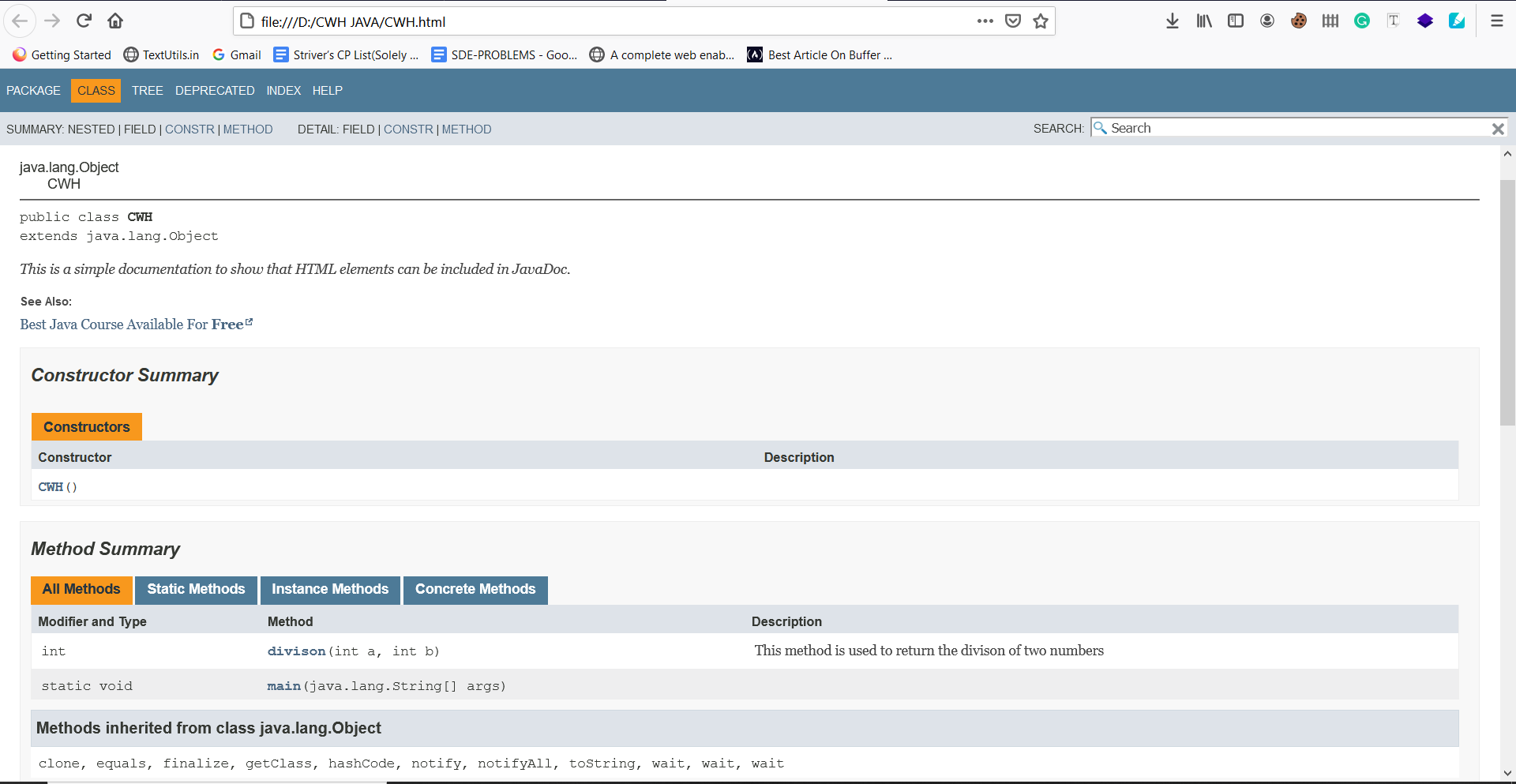
public static void main(String[] args) {

System.out.println("This is my main method");

}

}

##### **Screenshot of the generated JavaDoc:**



#### Code as described/written in the video :

package com.company;

/\*\*

\* This class is to demonstrate what javadoc is and how it is used in the java industry

\* This is <i>italic</i> word<p>this is a new paragraph</p>

\* @author Harry (CodeWithHarry)

\* @version 0.1

\* @since 2002

\* @see <a href="https://docs.oracle.com/en/java/javase/14/docs/api/index.html" target="\_blank">Java Docs</a>

\*/

public class cwh\_106\_javadoc {

public void add(int a, int b){

System.out.println("The sum is: " + a+b);

}

public static void main(String[] args) {

System.out.println("This is my main method");

}

}

### Javadocs: Method Tags For Generating java Documentation:

#### JavaDoc tags:

|  |  |  |
| --- | --- | --- |
| **Tag** | **Syntax** | **Description** |
| @author | @author name-text | Describes the author of a class. |
| @version | @version version-number | Adds a "Version" heading which specifies the current version of the  release or file. |
| @since | @since release-date | Adds a "Since" heading that tells about the release date. |
| @see | @see <a href="reference"></a> | Adds a "See Also" heading that refers to the other element of the  documentation. |
| @return | @return return-description | Adds a "Return" description that tells about the return value of the method. |
| @param | @param param-description | Provides the information about the method parameters in the  "Parameters" section. |
| @throws | @exception exception-name description | Displays the exception that can be thrown by a method  (same as @exception) |
| {@code} | {@code text} | Displays text in code font without interpreting the text as HTML  markup or nested javadoc tags. |
| @deprecated | @deprecated deprecatedtext | Adds a "Deprecated" heading indicating that this API should  no longer be used. |

package com.company;

/\*\*

\* This is a good class

\*/

public class cwh\_107\_method\_tags {

/\*\*

\*

\* @param args These are arguments supplied to the command line

\*/

public static void main(String[] args) {

System.out.println("I am main method");

}

/\*\*

\* Hello this is a method and this is the most beautiful method of this class

\* @param i This is the first number to add

\* @param j This is the second number to add

\* @return Sum of two numbers as an integer

\* @throws Exception if i is 0

\* @deprecated This method is deprecated please use + Operator

\*/

public int add(int i, int j) throws Exception{

if(i==0){

throw new Exception();

}

int c;

c= i+ j;

return c;

}

}

### Annotations in Java:

* Annotations provides metadata to class/methods.
* Annotations start with '@'.
* Annotations are helpful for detecting erros. Example : @override annotations will make sure that there are no typos while overriding a method.

#### Important Annotations In Java:

##### **@Override:**

* + This annotation makes sure that the sub class method is successfully overiding the parent class method.
  + While overriding a class, there is a chance of typing errors or spelling mistakes. In such cases, the method will not get overridden and you will get an error.
  + Override exception helps us to encounter such situtations by extracting a warning from the compiler.

##### **Example:**

class KeyPadPhone{

void sendMessage(){

System.out.println("Text message sent!");

}

}

class AndroidPhone extends KeyPadPhone{

@Override

void sendMessage(){

System.out.println("Message sent via WhatsApp!");

}

}

public class CWH{

public static void main(String args[]){

AndroidPhone Samsung = new AndroidPhone();

Samsung.sendMessage();

}

}

##### **Output:**

Message sent via WhatsApp!

##### **@Deprecated:**

* + This annotation is used to mark a deprecated method.
  + If developer uses the deprecated method then the compiler generated a warning.
  + There high chance of removal of deprecated methods in future versions therefore it is better to not use them.

##### **Example:**

class KeyPadPhone{

@Deprecated

void sendMessage(){

System.out.println("Text message sent!");

}

}

class AndroidPhone extends KeyPadPhone{

@Override

void sendMessage(){

System.out.println("Message sent via WhatsApp!");

}

}

public class CWH{

public static void main(String args[]){

AndroidPhone Samsung = new AndroidPhone();

Samsung.sendMessage();

}

}

##### **Build Output:**

java: sendMessage() in KeyPadPhone has been deprecated

##### **@SupressWarnings:**

* + This annotation helps us to supress some warnings that are being generated by compiler.

##### **Example:**

class KeyPadPhone{

@Deprecated

void sendMessage(){

System.out.println("Text message sent!");

}

}

class AndroidPhone extends KeyPadPhone{

@Override

void sendMessage(){

System.out.println("Message sent via WhatsApp!");

}

}

public class CWH{

public static void main(String args[]){

@SuppressWarnings("deprecation")

AndroidPhone Samsung = new AndroidPhone();

Samsung.sendMessage();

}

}

##### **Build Output:**

This time no warning is generated because we've suppressed the deprecation warning.

##### **@FunctionInterface:**

* + An interface which contains only one abstract method is known as functional interface.
  + @FunctionInterface annotation helps us to make sure that a functional interface is not having more than one abstract method.

##### **Example:**

@FunctionalInterface

interface myFunctionalInterface {

void method1();

void methodd2();

}

public class CWH{

public static void main(String args[]){

System.out.println("Functonal interface annotation");

}

}

##### **Output:**

java: Unexpected @FunctionalInterface annotation

myFunctionalInterface is not a functional interface

multiple non-overriding abstract methods found in interface myFunctionalInterface

The above code generates error because the **myFunctionInterface**is containing more than one abstract method.

package com.company;

@FunctionalInterface

interface myFunctionalInteface{

void thisMethod();

// void thisMethod2();

}

class NewPhone extends Phone{

@Override

public void showTime(){

System.out.println("Time is 8PM");

}

@Deprecated

public int sum(int a, int b){

return a+b;

}

}

public class cwh\_108\_java\_annotations {

@SuppressWarnings("deprecation")

public static void main(String[] args) {

NewPhone phone = new NewPhone();

phone.showTime();

phone.sum(5, 6);

}

}

### Java Anonymous Classes & Lambda Expressions:

* Anonymous class is nothing but a class without any name.
* They are used to override a class method or interface.
* Anonymous classes in Java help us to write more concise and readable code.

##### **Syntax:**

// Demo can be interface or abstract class.

Demo t = new Demol()

{

// data members and methods

public void Demo\_method()

{

........

........

}

#### How Anonymous class helps us to write concise code?

Take a look at the code given below:

@FunctionalInterface

interface Animal{

void bark();

}

class Dog implements Animal{

@Override

public void bark() {

System.out.println("Dog barks!");

}

}

class AnonDemo{

public static void main(String[] args) {

Dog Bruno = new Dog();

Bruno.bark();

}}

* In the above example, **Animal**is a FunctionalInterface containing a **bark()** method inside it.
* Class **Dog**implements the **Animal**interface and overrides the **bark()** method.
* **Bruno**is an object of Dog class on which we are running the **bark()** method. Now, let's see the output of the above code :

##### **Output:**

Dog barks!

 The same output can be generated without creating the **Dog**class. This is the scenario where the Anonymous class comes into the picture. With the help of the Anonymous class, we can declare and instantiate a class at the same time. Let's see how it is done:

@FunctionalInterface

interface Animal{

void bark();

}

class AnonDemo{

public static void main(String[] args) {

Animal Bruno = new Animal() {

@Override

public void bark() {

System.out.println("Dog barks!");

}

};

Bruno.bark();

}

}

In the above code, we've created the **Bruno**object by referencing the **Animal**interface. So, that's how we have overridden the **bark()**method without creating any separate class.

#### Ways to create an Anonymous Java class:

The Anonymous class in Java can be created by two ways:

1. By extending a class
2. By implementing an interface

Let's take an example for both ways listed above.

##### **By extending a class:**

* abstract class Vehicle{
* abstract void drive();
* }
* class AnonDemoByClass{
* public static void main(String[] args) {
* Vehicle car = new Vehicle() {
* @Override
* void drive() {
* System.out.println("I'm driving a car.");
* }
* };
* car.drive();
* }

}

##### **Output:**

I'm driving a car.

##### **By implementing an interface:**

* @FunctionalInterface
* interface Human{
* void walk();
* }
* class AnonDemo{
* public static void main(String[] args) {
* Human John = new Human() {
* @Override
* public void walk() {
* System.out.println("John walks.");
* }
* };
* John.walk();
* }

}

##### **Output:**

John walks.

#### Lambda Expressions:

* Lambda expressions were introduced in Java 8.
* They are similar to methods, but they don't need a name.

##### **Syntax:**

(parameter1, parameter2) -> { code to be executed }

Take a look at the example given below:

@FunctionalInterface

interface LambaExp{

void meth1(int a, int b);

}

class LambaExpDemo{

public static void main(String[] args) {

LambaExp obj =(a,b)->{

System.out.println("The value of a and b is : "+ a + "," + b);

};

obj.meth1(5,10);

}

}

##### **Output:**

The value of a and b is : 5,10

package com.company;

@FunctionalInterface

interface DemoAno{

void meth1(int a);

// void meth2();

}

//

//class HarryFunc implements DemoAno{

// @Override

// public void meth1() {

// System.out.println("This is method 1");

// }

//}

//class AnonyDemo implements DemoAno{

// public void display(){

// System.out.println("Hello");

// }

//

// @Override

// public void meth1() {

// System.out.println("I am meth1");

// }

//

// @Override

// public void meth2() {

// System.out.println("I am meth2");

// }

//}

public class cwh\_109\_lambda {

public static void main(String[] args) {

// DemoAno obj = new AnonyDemo();

// obj.meth1();

// Anonymous Class

// DemoAno obj = new DemoAno() {

// @Override

// public void meth1() {

// System.out.println("I am meth1");

// }

//

// @Override

// public void meth2() {

// System.out.println("I am meth2");

// }

// };

// obj.meth1();

// Lambda Expressions

// DemoAno obj = new HarryFunc();

// obj.meth1();

DemoAno obj = (a)->{System.out.println("I am method 1 from this lambda " + a);};

obj.meth1(6);

}

}

### Java Generics:

* Introduced from JDK 5.0 onwards.
* The Java Generics helps us to deal with the compiler time type-safety.
* With the help of the Generics, we can write a single method and call it with different argument types(integer, strings, etc.).

#### Advantages of Generics:

##### **Bugs can be detected at compile-time:**

* + While developing any application or program, it is always better to catch the bug/problem at the compile-time instead of runtime so that we can provide a smooth experience to the user.
  + Let's take an example to see how Java Generics helps us to detect problems at compile-time:

##### **Example:**

import java.util.ArrayList;

public class CWH {

public static void main(String[] args) {

// Without Java Generics :

ArrayList myArrayList = new ArrayList();

myArrayList.add(10); //Integer value

myArrayList.add("Harry Bhai!"); //String value

myArrayList.add(20.4); //Double value

System.out.println(myArrayList);

}

}

##### **Output:**

[10, Harry Bhai!, 20.4]

In the above code, notice that we can store any type of object in a collection without Generics. But, this is not the case with the Generics. It allows us to store only one type of object. Take a look at the example given below:

import java.util.ArrayList;

public class CWH {

public static void main(String[] args) {

// With Java Generics :

ArrayList<Integer> myArrayList = new ArrayList();

myArrayList.add(10); //Integer value

myArrayList.add("Harry Bhai!"); //String value

myArrayList.add(20.4); //Double value

System.out.println(myArrayList);

}

}

##### **Output:**

java: incompatible types: java.lang.String cannot be converted to java.lang.Integer

java: incompatible types: double cannot be converted to java.lang.Integer

The same code produces the **Incompatible type** error because we can only store the integer object type.

##### **Type-casting not required:**

* + Let's suppose you created an ArrayList(without using Generics), and you want to store the value at index 0 into an integer variable named "x." Are you allowed to do this in Java? The answer is a big NO! This is because the ArrayList returns an object, but we're storing the value in an integer variable. In such cases, we need to type-cast the object into our desired data type. But, if we use Generics, then there is no need to typecast. Take a look at the below example to get a better understanding:
  + import java.util.ArrayList;
  + public class CWH {
  + public static void main(String[] args) {
  + // Without Java Generics :
  + ArrayList myArrayList = new ArrayList();
  + myArrayList.add(10); //Integer value
  + myArrayList.add("Harry Bhai!"); //String value
  + myArrayList.add(20.4); //Double value
  + int x = myArrayList.get(0);
  + System.out.println(x);
  + }

}

##### **Output:**

java: incompatible types: java.lang.Object cannot be converted to int

The above code produces an error because we've not typecasted the object into the integer type. Now, let's typecast and see the results:

import java.util.ArrayList;

public class CWH {

public static void main(String[] args) {

// Without Java Generics :

ArrayList myArrayList = new ArrayList();

myArrayList.add(10); //Integer value

myArrayList.add("Harry Bhai!"); //String value

myArrayList.add(20.4); //Double value

int x = (int) myArrayList.get(0); //b=object typecasted into integer

System.out.println(x);

}

}

##### **Output:**

10

Now, let's see how we can get the desired results with the help of the Generic:

import java.util.ArrayList;

public class CWH {

public static void main(String[] args) {

// With Java Generics :

ArrayList<Integer> myArrayList = new ArrayList();

myArrayList.add(10);

myArrayList.add(20);

myArrayList.add(30);

myArrayList.add(40);

int x = myArrayList.get(0);

System.out.println(x);

}

}

##### **Output:**

10

package com.company;

import java.util.ArrayList;

import java.util.Scanner;

class MyGeneric<T1, T2>{

int val;

private T1 t1;

private T2 t2;

public MyGeneric(int val, T1 t1, T2 t2) {

this.val = val;

this.t1 = t1;

this.t2= t2;

}

public T2 getT2() {

return t2;

}

public void setT2(T2 t2) {

this.t2 = t2;

}

public int getVal() {

return val;

}

public void setVal(int val) {

this.val = val;

}

public T1 getT1() {

return t1;

}

public void setT1(T1 t1) {

this.t1 = t1;

}

}

public class cwh\_110\_generics {

public static void main(String[] args) {

ArrayList<Integer> arrayList = new ArrayList();

// ArrayList<int> arrayList = new ArrayList(); -- this will produce an error

// arrayList.add("str1");

arrayList.add(54);

arrayList.add(643);

// arrayList.add(new Scanner(System.in));

int a = (int) arrayList.get(0);

// System.out.println(a);

MyGeneric<String, Integer> g1 = new MyGeneric(23, "MyString is my string ", 45);

String str = g1.getT1();

Integer int1 = g1.getT2();

System.out.println(str + int1);

}

}

### File Handling in Java:

* File handling is a crucial part of any programming language.
* In Java, file handling is done with the help of the File class of the java.io package.

#### Common file handling operations:

1. Creating a new file.
2. Writing in a file.
3. Reading an existing file.
4. Deleting a file.

To perform any of the above operations on a file in Java, we need to create an object of the **File** class as shown in the below code:

import java.io.File; // Importing the File class

File obj = new File("filename.txt"); // Specify the name of the file

Now, let's see how we can perform the above operation on a file in Java.

#### Creating a new file:

* + **createNewFile()** method is used to create a new file. Take a look at the below example:
  + import java.io.File;
  + import java.io.FileWriter;
  + public class CWH {
  + public static void main(String[] args) {
  + File myFile = new File("CWH\_file1.txt");
  + try {
  + myFile.createNewFile();
  + System.out.println("File created successfully.");
  + } catch (IOException e) {
  + e.printStackTrace();
  + }
  + }

}

Output:

File created successfully.

#### Writing to a file:

* + **FileWriter**class is used with its **write()**method to write some content in a file.
  + Do not forget to use the **close()**method when you're finished writing to a file. Take a look at the below example :
  + import java.io.File;
  + import java.io.FileWriter;
  + import java.io.IOException;
  + public class CWH {
  + public static void main(String[] args) {
  + File myFile = new File("CWH\_file1.txt");
  + try {
  + FileWriter fileWriter = new FileWriter("CWH\_file1.txt");
  + fileWriter.write("CodeWithHarry is one step solution for your all programming problems.\nKeep learning, Keep coding!");
  + fileWriter.close();
  + } catch (IOException e) {
  + e.printStackTrace();
  + }
  + }

}

##### **Output of the CWH\_file1.tx:**

CodeWithHarry is one step solution for your all programming problems.

Keep learning, Keep coding!

#### Reading a file:

* + The**Scanner**class is used to read a file.
  + It is important to enclose the method in a try-catch block to handle the IOException.

##### **Example:**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class CWH {

public static void main(String[] args) {

File myFile = new File("CWH\_file1.txt");

try {

Scanner sc = new Scanner(myFile);

while(sc.hasNextLine()){

String line = sc.nextLine();

System.out.println(line);

}

sc.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

##### **Output:**

CodeWithHarry is one step solution for your all programming problems.

Keep learning, Keep coding!

#### Deleting a file:

* + The**delete()**method is used to delete a file in Java.

##### **Example:**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class CWH {

public static void main(String[] args) {

File myFile = new File("CWH\_file1.txt");

if(myFile.delete()){

System.out.println("I have deleted: " + myFile.getName());

}

else{

System.out.println("Some problem occurred while deleting the file");

}

}

}

##### **Output:**

I have deleted: CWH\_file1.txt

#### Code as described/written in the video:

package com.company;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class cwh\_111\_file {

public static void main(String[] args) {

// Code to create a new file

/\*

File myFile = new File("cwh111file.txt");

try {

myFile.createNewFile();

} catch (IOException e) {

System.out.println("Unable to create this file");

e.printStackTrace();

}

// Code to write to a file

try {

FileWriter fileWriter = new FileWriter("cwh111file.txt");

fileWriter.write("This is our first file from this java course\nOkay now bye");

fileWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

// Reading a file

File myFile = new File("cwh111file.txt");

try {

Scanner sc = new Scanner(myFile);

while(sc.hasNextLine()){

String line = sc.nextLine();

System.out.println(line);

}

sc.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

\*/

// Deleting a file

File myFile = new File("cwh111file.txt");

if(myFile.delete()){

System.out.println("I have deleted: " + myFile.getName());

}

else{

System.out.println("Some problem occurred while deleting the file");

}

}

}

### Advanced Java 2 - Practice Set:

**Question 1:** Create a class and a method with deprecated annotation. What is its effect on program execution?

**Answer 1:** We discussed the deprecated annotation Below is the required program:

class MyDeprecated{

@Deprecated

void meth1(){

System.out.println("I am method 1");

}

}

public class CWH {

public static void main(String[] args) {

MyDeprecated d = new MyDeprecated();

d.meth1();

}

}

##### **Output:**

I am method 1

There is no as such special effect on the program of deprecated annotation. The only thing is that compiler generated a waning if we use deprecated method or class in our program.

**Question 2:**Suppress the warning generated in question number 2.

**Answer 2:**We saw how compiler-generated warnings could be suppressed by using [@SuppressWarnings annotation](https://www.codewithharry.com/java-tutorials-for-beginners-108). The required program is given below:

class MyDeprecated{

@Deprecated

void meth1(){

System.out.println("I am method 1");

}

}

public class CWH {

public static void main(String[] args) {

@SuppressWarnings("deprecation")

MyDeprecated d = new MyDeprecated();

d.meth1();

}

}

**Question 3:**Create an interface and generate an instance from it.

**Answer 3:**

interface MyInt{

void display();

}

public class CWH {

public static void main(String[] args) {

MyInt i = () -> System.out.println("I am display");

i.display();

}

}

##### **Output :**

I am display

**Question 4:**Write a Java program to generate a multiplication table of a given number and write it to a file.

**Answer 4:**The following program saves the multiplication table of 19 into a file named "MultiplicationTable.txt".

import java.io.FileWriter;

import java.io.IOException;

public class CWH {

public static void main(String[] args) {

int i = 19;

String table = "";

for (int j = 0; j < 10; j++) {

table += i + "X"+(j+1) + "=" + i\*(j+1);

table += "\n";

}

try {

FileWriter fileWriter = new FileWriter("MultiplicationTable.txt");

fileWriter.write(table);

fileWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

### JAVA EXERCISE: LIBRARY MANAGEMENT SYSTEM-

Create a library management system which is capable of issuing books to the students.

Book should have info like:

1. Book name

2. Book Author

3. Issued to

4. Issued on

User should be able to add books, return issued books, issue books

Assume that all the users are registered with their names in the central database

### Exercise: Solution-

Below is the solution of exercise number 7 that I gave you:

package com.company;

import java.util.ArrayList;

class Book{

public String name, author;

public Book(String name, String author) {

this.name = name;

this.author = author;

}

@Override

public String toString() {

return "Book{" +

"name='" + name + '\'' +

", author='" + author + '\'' +

'}';

}

}

class MyLibrary{

public ArrayList<Book> books;

public MyLibrary(ArrayList<Book> books) {

this.books = books;

}

public void addBook(Book book){

System.out.println("The book has been added to the library");

this.books.add(book);

}

public void issueBook(Book book, String issued\_to){

System.out.println("The book has been issued from the library to " + issued\_to);

this.books.remove(book);

}

public void returnBook(Book b){

System.out.println("The book has been returned");

this.books.add(b);

}

}

public class cwh\_113\_ex7sol {

public static void main(String[] args) {

// Exercise 7 Solution

ArrayList<Book> bk = new ArrayList<>();

Book b1 = new Book("Algorithms", "CLRS");

bk.add(b1);

Book b2 = new Book("Algorithms2", "CLRS2");

bk.add(b2);

Book b3 = new Book("Algorithms3", "CLRS3");

bk.add(b3);

Book b4 = new Book("Algorithms4", "CLRS4");

bk.add(b4);

MyLibrary l = new MyLibrary(bk);

l.addBook(new Book("algo4", "myAuthor"));

System.out.println(l.books);

l.issueBook(b3, "Harry");

System.out.println(l.books);

}

}

----- Thanks for using the notes, I hope these notes help you a lot ----