|  |
| --- |
| COMPUTER   * Computer is an electronic device which is composed of software and hardware components. Primarily used to store, process and retrive data * Examples of software components consist of 🡪Os,VM,device drivers,firmwares,programing softwares, middleware * Examples of hardware components consist of🡪Processing units, input device, output device |

**Java notes/questions /programs**

|  |
| --- |
| SOFTWARE   * Set of programs which works in co-ordination with each other to perform a specific task In order to solve real world problem is known a software * E.g:power point presentation,excel,Pubg etc |

|  |
| --- |
| HARDWARE  Set of electronic components on which applications runs are known as hardware. E.g::CPU,GPU,SSD,HDD |

|  |
| --- |
| CONVENTIONS IN JAVA   * *Conventions are nothing but industry standereds whicha are oprtional but highky recommended to follow.While excuting coventions are not checked by compiler* * *For class name and interface name use : Camelcase* * *For variable name and method name use : Pascal case* * *Explain pascal vs camal case* |

|  |
| --- |
| PROGRAMING LANGUAGE:   * Language :Language is medium of communication * Programing language: Programing language are the languages used to communicate with Computer * Technically we can say that,   The languages by using which we command or provide instructions to the computer in order to perform task from computer is known as Programing language   * <difference between coding and programing>. |

|  |
| --- |
| PROGRAMING   * *<what is programing ? why we do programing>* * *<Difference between coding and programing>* |

|  |
| --- |
| TYPES OF PROGRAMING LANGUAGES:   * LLL:LLL are also known as machine level languages. The Programing languages which are directly understandable to computer are known as LLL * MLL:The Programing languages which consist of predefined code words which are understandable to computer   through intermediate software 'assembler' called as middle level language.  Assembler is an intermediate software used to covert assembly code native machine code.   * HLL:   The language close to human language and easily readable , understandable and instructible by programmer to computer is known as High Level Language.  E.g: c++, Java, python, Perl, Scala, ruby, php, kotlin, c# |

|  |
| --- |
| What is interpreter?  Interpreter is an compur=ter software which executes human readable HLL language code line by line enabling program to run of specific hardware or environment without creating intermediate file |

|  |
| --- |
| JDK ARCHITECTURE CONSIST OF JDK ,JRE AND JVM   * JDK: * JDk stands for 'Java Development Kit' is a package given oracle community. It is used to develop as well run java application. * it internally consist of JRE and JVM * Technically we can define it as -> JRE along with development tool is termed as JDK. * JRE: * JRE stands for Java Runtime Environment. It is used to run java applications . * It internally consist of JVM * Technically we can define it as ->JVM along with built in Libraries termed as JRE * JVM * JVM stand for Java Virtual Machine . * It is an interpreter of java language which converts byte code which can be generated by any platform into corresponding platforms machine code ,which will lead to achieve platform independency feature.   Summery  Jdk=JRE+Development tools(used to devlop an run java applications)  Jre=jvm+built in Libraries(used to run hava applications)  Jvm=java interpreter used to achieve platform independency  (used to convert byte code to corresponding platform's native machine code and execution of code line by line) |

|  |
| --- |
| About Java Language:   * Java is a high-level, object-oriented programming language. * It was developed by James Gosling and his team at Sun Microsystems in 1995. It was initially called 'oak' and later renamed Java. * The purpose of its creation was to develop a language suitable for embedded systems and to address the limitations   previous languages of C and C++ languages.[MEMORY MANAGAGEMENT.POINTER Arithmetics,platform independency,inbuilt support of built-in support for exception handling,inbuilt support of multithreading ]   * Java is popular for its features like platform independence (thanks to the Java Virtual Machine) and,. * It is a statically typed and strongly coupled language. * Its standardized support for exception handling makes it a robust language. |

|  |
| --- |
| History of java:   * It was developed by James Gosling and his team at Sun Microsystems in 1995. It was initially called 'oak' and later renamed Java. * The purpose of its creation was to develop a language suitable for embedded systems and to address the limitations of C and C++ languages. * It's first version was released in 1996 * in 2010 Java was owned by oracle * stable version of java language is jdk 1.8 which was released in 2014 |

|  |
| --- |
| What is compiler?  Compiler is a software which coverts human redable HLL source code into corresponding lower-level representation of code which will be understandiable by computer and enables computer program to run/execute on hardware or envirnoment |

|  |
| --- |
| Platform:   * The combination of s/w and h/w component on which computer software runs is known as platform * Platform dependent software: * The softwares which run on only that type of platform on which they made I.e native type of platform are known as platform dependent softwares * E.g: software made using cand c++ * Platform independent software: * The softwares which run on any type of platform irrespective of platform on which they made are known as platform independent softwares * E.g:software made using java |

|  |
| --- |
| Platform dependency in java :   * Platform dependency in c and c++ * When we write code in c++,save with .cpp extension. After successfull compilation of code native code is generated. Native code is the language code which will execute only platform type on which they made. * Due to this s/w made using c and c++ are platform dependent * Platform independency in java * When we write code java save it with .java extension. After successful compilation of code will generates class file also known as byte code * This byte code is neither understandable to programmer as well as computer. * But this byte code can be converted into corresponding native machine code using JVM * JVM is Java Virtual Machine is a interpreter ,is a platform dependent s/w designed for each platform * Due to this java code can run on any platform irrespective of platform on which it is made * Eventually this results in achieving platform independency * So .software made using java are platform independent |

|  |
| --- |
| Dynamic input using CLI / *How to execute java program using cmd/terminal?/ How to give input to java program through cmd?* |
| // package dynamicInputsFromUser.usingCmd;  public class Driver1 {  public static void main(String[] args) {  System.out.println("multiplication of inputs from CLI :"+(Integer.parseInt(args[0])\*Integer.parseInt(args[1])));  }  } |
| *javac Driver1.java*  *java Driver1 6 7*  *multiplication of inputs from CLI :42* |

|  |
| --- |
| EXPLAIN STRUCTURE OF JAVA LANGUAGE   * Package name * import statements * class block * method blocks * each method consist of set of instruction   Execution of class starts from main method, all other methods are directly or indirectly called by main method, and execution ends at main method |

|  |
| --- |
| PACKAGES IN JAVA:   * package in java is mechanism of grouping and organizing related classes and interfaces together so they will work as a single entity to achieve common goal or functionality. * package creation statement must be first statement of class * packages are of two types, * A]Built in:(def)e.gjava.lang/util/sql/math/io/awt   The packages created by creators of java language which can be directly used by programmer are known as built in packages   * B]user defined:(def)To build software entity programmer can creates it's own packages such packages can be termed as user defined packages * Advantages of package: * modularity of s/w increases * maintainability of s/w increases * security of s/w increases * code reusability can be achieved |

|  |
| --- |
| Tokens in java:   * Building blocks of programing language by using which we build instructions of language * In other way we can define it as smallest unit of programing language by using which we creat/write instructions in language * In java we have several types of tokens and they are * Literals * separators * operators * comments * keywords * identifiers |

|  |
| --- |
| *Tokens in java* |
| public class Driver1 {  public static void main(String[] args) {  //addition code  System.out.println("Addition :”+(4+6));  }  } |
| |  |  | | --- | --- | | Token in above program |  | | Literals | “addition” | 4 |6 | | Separator | {} |()(?????) | | | Operator | [] | + | | | Keyword | Public | class | void | main (built in?????) | String | | identifiers | Driver1 | args built in?????) |System |out|println | | comments | //addition code | | |

|  |
| --- |
| LITERALS IN JAVA:   * Literals are nothing but kinds of data we are going to deal while programing in java. * In java Literals are of two types   A)primitive Literals   * Primitive Literals are the single value data. This Literals are further classified into 2 types * a)numerical primitive Literals   This Literals deal with number format values. Again this Literals are classified in two types and they are   * a.1)integer literals: numerical Literals without point values * a.2)floating point literals: numerical Literals with point values * b)non numerical primitive literals * b1)Character Literals: all characters on keyboard are comes under character literals. Again this Literals is divided into 3 types and that types are * c1)Alphabets: these literal consist of English alphabets from ‘a’ to ‘z'   They are aging divided in to three types and that types are   * c.11)uppercase character * c.12)lowercase character * c2) digits: there total 10 digits from 0 9 * c3) special characters: characters other than digits and alphabets   B)Non primitive literals  These are multiple valued a data.Different non primitive Literals are   * A)class * B)strings * C)array * D)interface * E) enum |

|  |
| --- |
| Explain keywords in java  Give rules of naming identifiers  KEYWORDS:   * One of the tokens of java language * keywords are reserve words. These are compiler aware words whose meaning already known to compiler used to build instruction. * Programmer can't alter keywords * Always in lowercase * We can't use them as identifier and programmer can alter behavior of keywords * java language has 51+ keywords (!!!)   Eg::static,public,private,int,long,short,byte,float,for ,continue,break |

|  |
| --- |
| Difference between = and == operator   * A]used for:= assignment operator used to assign value to variable while == is comparison operator used to compare two primitive type values * B]= is assignment operator while == is comparison operator * C]return type of = is respective numerical literal datatype and of == Boolean * D]not used for decision making while == used for design making |

|  |
| --- |
| Difference between == operator and .equal() method   * A]used with :primitive and non-primitive data * B]do comparison of :primitive value and objects(ref/content)   C]WRT string do comparison: based on ref and based on content |

|  |
| --- |
| CONDITIONAL OPERATOR   * ternary operator used to for decision making * Syntax   (condition)?(operand 1):(Operand 2); (with explanation)   * result type depends on operand 2 and operand 3 |

|  |
| --- |
| Miscillanious operator  dot operator,instanceOf(),new operator  instanceOf() :   * to check object reference belong to hierarchy of given object type[??????????] |

|  |
| --- |
| RELATIONAL OPERATOR:   * binary operator used to compare primitive values(operands) * [== >= <= > < !=] * return type is 'Boolean' used for decision making * we can use relational operators other than (>,<,<=,>=) with Boolean values i.e:t(==,!=relational operators with Boolean value |

|  |
| --- |
| UNARY OPERATOR:   * unary operator used to update variable value by one : * 4 types unary operator * pre increment operator : increases value before use * post increment operator: increases value after use * pre decrement operator :decreases value before use * post decrement operator: decreases value after use |

|  |
| --- |
| EXPLAIN DATATYPES IN JAVA:   * Datatypes are the keyword which are used to declare variable and it specifies which kind of literals it will be stored,   what will be its size and default value .   * E.g.: byte a ; * Here byte is keyword and datatype which specifies declaring variable datatype will of integer literal type and it will store value within range [128 to 127 ]and default value will be 0. * We have total 8 datatypes in java and they are * byte : This datatype is of primitive integral type, stores 1 byte data ,will store value from[ 2^7 to 2^7 ( 1) ] * short : This datatype is of primitive integral type, stores 2 byte data ,will store value from[ 2^15 to 2^15 ( 1)] * int : This datatype is of primitive integral type, stores 4 byte data ,will store value from[ 2^31to 2^31 ( 1) ] * long : This datatype is of primitive integral type, stores 8 byte data ,will store value from[ 2^63 to 2^63 ( 1)] * float : This datatype will store floating point integral type with less number of precisions and of size 4 bytes. * double : Float :this datatype will store floating point integral type with more number of precisions and of size 8 bytes * char : The char datatype stores character literal type value 0 to 2^16,and of size 2 byte * Boolean: Size can't determine as it depend on code. Stores boolean literal type of value. |

|  |
| --- |
| VARIABLES IN JAVA:   * Variables are the container which stores value and is created with the help of datatypes.   In simple word we can define variable as name given memory address for storsge of value   * As java is statically typed language without specifying dataypes of variable we can't declare variable. * Variable declaration is the process of declaring variable along with it's datatype * Syntax of variable declaration is , * <datype keword> <variable identifier>; * Variable initialization the process of assigning value to declared variable * When we declare and initialize variable simultaneously then we call it as variable declaration and initialization * We can reassign value to declared variable an this process is known as reinitialization * ------------------------------------------------------------------------------------------------------------------------------- * We have certain rules to naming variables and that are ,(Rules of identifiers) * 1)variable name should not start with number * 2) it should not consist of special characters other than $ and \_ * 3)should not consist of blank spaces * 4)keyword can't be used as an identifier * ----------------------------------------------------------------------------------------------------------------- * In java variables are mainly classified into two types and that are , * A]Local variables * B]Global variables * ----------------------------------------------------------------------------------------------------------------- * A]local variables :   Variables declared in any method block are known as local variable of that block and can't be access out of that block .  I.e. local variable's scope is limited to block only in which they are declared.   * We have certain rules for declaring local variables and that are, * We can't have multiple variable of same in same block * But can have multiple variable of same name in different blocks * Scope of local variable is limited to block in which they are declared .We can't use local variable without initialization .I.e before use local variables in * They must be initialized with some value explicitly. * ---------------------------------------------------------------------------------------------------- * B]Global variables : * Variables declared in class block not in method block are known as global variables. They can be accessed within class or other class anywhere. * Global variables are further classified into two sub types and that are,   a]static variables  b]non static variables  a]static variables :   * The global variables which are prefixed with 'static' keyword (which work as non access modifier) are known as static variables. Static variables are also known as class variables.   b]non static variables :   * The global variables which are not prefixed with 'static' keyword are known as non static variables.Non static variables are also known as 'Instance variables'. |

|  |
| --- |
| Difference between local and global variables   * A]DECLARATION: local variables are declared in method block while global variable declared in class block * B]ACCESSIBLITY OUT OF BLOCK: local variables can't be accessed out side block while global variables can be access outside block using class or object name as reference. * c]ACCESSIBLITY OUT OF CLASS: local variables can't be used in other class while global variable can be used in other class * D]DEFAULT VALUE: Before use, initialization of local variable is compulsory but in case of global variable * E]MEMORY:static and non-static variables are stored in method area and stack area respectively   it is not compulsory as they are initialized with default values implicitly if programmer do not to initialize them. |

|  |
| --- |
| Difference between static and non-static variables   * A]Named as: static variables are also known as class variable or class member while non-static variables are also known as instance variable * B]Access :we can access static variable using class name as reference while non-static variables can be accessed using object name as reference * c] Use: we use static global variables when we want do programing without using objectwhile non-static global variables are used to do programing using objects * D]static :stack memory |non-static heap memory |

|  |
| --- |
| TYPECASTING :   * A]primitive (I. widening II. Narrowing ) * B]non primitive (I. Upcasting II. downcasting)   TYPECASTING:   * process of converting one type of data into another type * Types of typecasting A]primitive typecasting | B] non-primitive typecasting * A]primitive Typecasting: process of converting one type of primitive data into another primitive type .To do primitive typecasting literal type should be in primitive data type only * Types of primitive typecasting * A]widening:   primitive typecasting in which datatype with less size converted into datatype with large size  as widening do not leads to data loss it is done implicitly by compiler   * B]narrowing:   primitive typecasting in which datatype with larger size converted into datatype with smaller size  as narrowing do leads to data loss it is done explicitly by programmer.   * B]non primitive Typecasting:   process of converting one type of non primitive data into another type of non primitive  to do non primitive typecasting class should be in 'is a ' relation   * upcasting : * non primitive typecasting in which derived class type object is converted/upcasted to it's base class type * we do upcasting to achieve generalization * doing upcasting we can override methods of base class * after doing upcasting through derived class we can use members of base class only * downcasting * non primitive typecasting in which upcasted derived class type object converted back to it's original derived class type * we do downcasting to achieve specialization * downcasting is solution for problem occurred due to upcasting * after doing downcasting through derived class we can use it's own as well as base class members also |

|  |
| --- |
| CONTROL FLOW STATEMENTS :   * control flow statements types : A]conditional statements | B]loop statements * A]conditional statements * if else statements * 'if' statement * 'if else' statement * 'if elseif' ladder * 'if elseif' ladder followed by else statement * switch case statements * When to go for if-stattement,if-else statement and if-eleif ladder ? * B]loop statements (each loop with flow and working) * LOOPS * while * do while * for * for each * Control flow statements :   The statement used control the execution flow of program are known as control flow statements   * control flow statements are of two types and they are , * A]conditional statements: * B]loop statements: * A]conditional statements:   The control flow statements which are used for decision making while execution of program are known as conditional statement.  B]loop statements: The control flow statement which are used for to do repetitive task are known as looping |

|  |
| --- |
| PROGRAMES  -try usin all loops  -try for each loop and for loop  -Using java 8 |
| * Voter validation based on age * Using if-else statement * Using ternary operator * Using if statement * Program to check which kind of character is entered(uppercase alphabet, lowercase alphabet ,digit, special symbol) * Using character operands * Using ASCII value * program to demonstrate use of break and continue keyword * swapping programs * swapping of 2 variables using 3rd variable * swapping of 2 variables without using 3rd variable * using + and – operator * using \* and / operator * using bitwise operator * all above without using else statement * even/odd number * using modulus operator * without using modulus operator * all above without using else statement * divide/multiply all number within range by specific number and print result without using / and \* operator respectively * create variable which can store all kind of primitive literals * create variable which can store all kind of literals * give the sum/product of first natural number * using loop * without using loops * password application (strong, weak ,medium) * game application * password verification within limit * leap year program * extract digits from number(with and without using string format) * digit’s addition /multiplication in given number(with and without using string format) * even /odd digits count in given number(with and without using string format) * twisted even /odd number * count of digits in a given number(with and without using string format) * palindrome number(within range | next) * palindromic even /odd * reverse number (with and without using string format) * check duck number * factorial of a number (using recursion| using loop) * digit’s factorial addition /multiplication * base race to index value(using inbuilt function | using loop) * factors of a number * factors addition /multiplication of digit of a number   -perfect number   * Tech number * Check sum and product of digit is same or not * Check factoral’s/factors/ multiplication and addition n is same or not * Print fibonascii series * Check elemnt is fibonascii or not * Make element fibonacii * Print nth fibonascii element * Fimd sum and product check they ar eequal * LCM and HCF * Prime number(in range) * check all digits are prime or consonant or mix * find digits sum/product .check it Is prime or consonant * prime factor’s multiplication and addition n is same or not * co-prime number * check twisted prime number * twin prime in range * strong number(within range) * Armstrong number(within range)[Armstrong prime number] * Automorphic number(within range) * Pronic * Happy number in range * Buzz * Neon * Rotate number * Unique number in range * niven number in range * ramanuj number in range * smith number in range * ugly number in range * Adam number in range * bouncy number * Defficient abudent number in range * Facscinatin number in range * Good number in range * Kaith number * Strontio * Sunny number * Xylem phloem number * Catlon number * fermats number * goldbatch number * hailstone series * lucas number * lychral numberpell number series * spenic number * ALL PATTEREN PROGRAMS * pascal triangle |

|  |
| --- |
| METHOD BINDING PROCESS:   * Method binding is the process of linking method signature to it's method body. This method binding is done by compiler   so it is known as compile time binding or compile time polymorphism   * method binding using method signature follows several steps to bind method body to it's method call statement and they are, * first compiler search for method name(if not found) * if multiple names of same method are there it further checks total number of formal arguments * if doesn’t get any method matching number of arguments it throws compile time error   and if number of formal and actual number matches   * if datatypes of formal and actual argument matches that method body will binded to method call statement * if datatypes do not matches again we have two possibilities * widening happens: method body is binded to method call statement * widening is not possible: it will throw compile time error |

|  |
| --- |
| Methods in java :   * what is method? * why we use ? * syntax explanation(m,am,nam,rt r,mn,ms,mb) * methods classification based on different 3 criteria's * method overloading * characteristics of methods   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  Methods in java:   * method is a set of instruction which is used to perform specific task * Methods are reusable block of code which may or may not return value depending on it's return type . * Methods are used to achieve code modularity * code modularity is nothing but dividing larger task in to sub task and designing reusable methods for each sub module. * Code modularity has several benefits and they are, * code readability increases * code reusability is achieved * easy bug fixing * independent logic checking * Talking about structure/syntax of method, * method syntax consist of two components * method prototype followed by method body. * method prototype's syntax is <access modifier><non access modifier><return type><method name><formal arguments in parenthesis> * method name along with formal arguments in parenthesis is also known as method signature * return type of method specifies that method is return value or not ,If method is returning value what will be it's type. * modifiers: are the keyword used to alter the behavior of program * non access modifier will specifies customized behavior for method. * for example , * static non access modifier will specifies that declaring method is class member, * final non access modifier will specifies that method will be inherited by derived class but restricted to override * Access modifiers are the modifiers which alter the behavior of program as per visibility. * Access modifier defines the scope of java member within class & packages. * for example : java members which are prefixed with protected access modifier can be accessible by any class and can be accessible by other package's sub class * talking about method body, * method body consist of set of instruction and a return statement. * return statement is has 3 functionality, * return value back to calling method if return type is other than void, * terminates frame execution of method from stack area of a memory * if return type is void method will not return any value and use of return statement is not compulsory otherwise it will return value and use of return statement is compulsory * method call statement: method signature along with actual arguments is known as method call statement * Methods has several types depending on different categories, * method type based on creator of method are, * built in method * user defined method * method type based on arguments * no argument methods * parameterized methods * method type based on method body are , * abstract method * concrete methods * Method overloading: * a class having more than one method of which one of will executed during runtime is known as method overloading * certain rules for method overloading????????? * why it comes under compile tile poly morphism or compile time binding??? * we have certain characteristics of method and that are, * in class we can create any number of methods * we can call method using method call statement * methods get executed when they called * we can call method any number of time * local variables of method can't be used outside that method |

|  |
| --- |
| What is recursion? Why we use recursion?  RECURSION :   * Method calling itself is called as recursion * Recursion is used to do repetitive task   Recursion is faster alternative for loops |

|  |
| --- |
| Explain return statement in java  RETURN STATEMENT :   * it is control transfer statement in java. * it has 3 functionalities: * it terminates execution frame of called method from stack memory. * if called method's return type is other than void return statement will return value to calling method. * transfer control from called method to calling method * for methods having return type void use of 'return' statement is optional otherwise it is compulsory . * we can't write instruction after return statement .it we write we get compile time error |

|  |
| --- |
| Difference between actual vs formal arguments:  A]formal arguments: variable declared in method signature  B]actual arguments: are variable passed to method call statement  FORMAL AND ACTUAL ARGUMENTS:   * formal arguments/parameters :   Variable which are declared in method signature of method are known as formal arguments   * actual arguments:   Variables which are passed to calling statement of any method are known as actual arguments |

|  |
| --- |
| What are static members?Use of static members  STATIC JAVA MEMBERS:   * Static is keyword and a non access modifier. * Static modifier can be prefixed with variable ,method ,block and inner class only. * It is non access modifier which specifies that declaring java member is static member/class member.   I.e. When we declare any java member which is prefixed with 'static' modifier then that member is called as static member.   * Static members are also called as class member as they can called anywhere in same class other class. * We can call class member in same class directly by calling it's name and in other class using class name as reference * Static members are loaded in memory and static initializers executed from top to bottom * Which will lead to completion of loading process of class. After that main method execution starts * Static members of class are used to do programing without using objects * Static variables: * The variable which are prefixed with static keyword/ or modifier is known as static variable. * As static variables are prefixed with 'static' keyword they becomes static members of class. * Static variables are also called as class variables * We can access class variable within same class anywhere using variable name directly and * If we can also access class variables in other class by calling name with class name as reference * Static variable which are not initialized will be implicitly initialized with default values during loading process of class * If static variable is declared as well as initialized it will become static initializer of class and will be executed * During loading process of class.(can be executed from top to bottom if class has multiple such static initializers) * Static methods: * The methods which are prefixed with static keyword/ or modifier is known as static methods. * As static methods are prefixed with 'static' keyword they becomes static members of class. * Static methods are also called as static context * We can access static within same class anywhere using name of method directly and * If we can also access static context in other class by calling name along with class name as reference * Static blocks: * The black which is prefixed with static modifier is known as static block. * Static blocks are static initializers which executes during loading processes of class * They don't have return type and identifier ,so programmer can't call anywhere explicitly. * They are generally used to perform task/execute task before start of main method . * For example: loading database drive ,for creating connection, setting up environment etc. * Static initializers: * These are the java members which are get executed during loading process of class * Static variable declaration with initialization and static blocks are called as static initializers * After loading all static member in memory ,static initializers get executed from top to bottom |

|  |
| --- |
| What are non static members?Use of static members  NON STATIC JAVA MEMBERS :   * The members which are declared in class block and do not prefixed with any * Non access modifier are termed as non static member of class * We can make variable ,method ,block and inner class as non static member. * Non static members are also called as instance member as they can called anywhere in same class an inOther class using object reference. * Non static members of class are used to do programing using objects * non static variables: * The variable which are not prefixed with static keyword/ or modifier is known as non static variable. * Non static variables are also called as instance variables * We can use instance class in same class or other class using object reference. * Non static variable which are not initialized will be implicitly initialized with default values during loading process of object * If non static variable is declared as well as initialized it will become non static initializer and can be executed   During loading process of object.(can be executed from top to bottom if class has multiple such non static initializers)   * non static methods: * The methods which are not prefixed with static keyword/ or modifier is known as non static methods. * Non static methods are also called as nonstatic??????????? Context * We can use non static methods in same class or other class using object reference. * non static blocks: * The black which is not prefixed with any modifier and declared in class is known as non static block. * non static block is non static initializers which executes during loading processes of object * they don't have return type and identifier, modifier, so programmer can't call anywhere explicitly. * non static initializers: * These are the java members which are get executed during loading process of object * Non static verifiable declaration with initialization and non static blocks are non static initializers   After loading all non static member in heap memory ,non static initializers get executed from top t |

|  |
| --- |
| Explain object loading process  LOADING PROCESS OF OBJECT:   * New keyword create block of memory in heap area and return unique reference for newly created object * Constructor is called. Constructor starts loading process of object * All non static members of class will get loaded in object * Non static variable with which are no initialized will be initialized with default values implicitly * All non static initializers will executed from top to bottom * All instructions in constructor will executed. * Complete execution of constructor lead to completion of loading process of object |

|  |
| --- |
| Difference between class and object  CLASS VS OBJECT:   * Class is blueprint/templet and object is real world entity * Object is real world entity which has existence in physical world while class is template or blueprint to real world object in programing |

|  |
| --- |
| How to create object in java?  HOW WE CREATE OBJECT IN JAVA?   * we create object using new keyword along with constructor of class. * new keyword is unary operator which create block of memory at runtime .assigns unique reference to that memory * which will be assigned to object reference variable . * constructor loads all non static members of class to newly created object . * this will lead to creation of object in heap memory at runtime |

|  |
| --- |
| What is new? Why we use it?  NEW KEYWORD:   * new is a keyword and unary operator * it is used to for creation of object * to create object we use new keyword along with constructor of object representing class * it has two functionality * it will create block of memory in heap area at runtime * assign unique reference to block memory and returns this reference back to object reference variable |

|  |
| --- |
| Explain class  CLASS:   * To create object of real world first we have to create template/design of real world entity ,which we create using class. * Class in java is user defined non primitive datatype which consist of all specifications of object. * Non static variables of class represent attributes of object while non static methods of class represents behaviors of object   [ref:CWH]  How we model real word entity in OOP?   * [NOUN ]| Entity name |Class name | {employee} * [ADJECTIVE]| Entity properties/details |non static variable/instance variables | {name ,age,sal} * [VERB ]| Entity behavior/Actions |non static method | {getSalary,getname} |

|  |
| --- |
| Explain types inner class and it’s type  WHAT IS INNER CLASS?   * A class inside another class is known as inner class. * scenarios where we use inner class, * if i want to create class which will use private member of other class(other class=outer class) * inner class is logically belongs to outer class * types of inner class * non static inner class * static inner class * local inner class * anonymous inner class |

|  |
| --- |
| ***JVM Architecture – Understanding JVM Internals***  [July 12, 2019](https://harshitjain.home.blog/2019/07/12/jvm-architecture-understanding-jvm-internals/) by [HARHSIT JAIN](https://harshitjain.home.blog/author/hjain5791/), posted in [JAVA](https://harshitjain.home.blog/category/java/)  Every Java developer knows that bytecode will be executed by **JRE** (Java Runtime Environment). But many doesn’t know the fact that **JRE** is the implementation of JAVA VIRTUAL MACHINE (JVM), which analyzes the bytecode, interprets the code and executes it. It is very important as a developer we should know the Architecture of JVM, this enables us to write code more efficiently. In this JVM architecture in Java with diagram article, we will learn more deeply about **JVM architecture** in Java and **different components** of a JVM.  ***What is the JVM?***  A **Virtual Machine** is a software implementation of a physical machine. Java was developed with the concept of **WORA (*Write Once Run Anywhere*)**,which runs on a **VM**. The**compiler** compiles the Java file into a Java **.class** file, then that .class file is input into the JVM, which Loads and executes the class file. Below is a diagram of the Architecture of the JVM.  ***How Does the JVM Work?***  As shown in the above architecture diagram, the JVM is divided into three main subsystems:   1. Class Loader Subsystem 2. Memory Area (Runtime Data Access) 3. Execution Engine   ***1. Class Loader Subsystem:***  Java’s dynamic class loading functionality is handled by the class loader subsystem. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time.  ***1.1 Loading:***  Classes will be loaded by this component. BootStrap class Loader, Extension class Loader, and Application class Loader are the three class loader which will help in achieving it.   1. **Boot Strap ClassLoader–** Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar.**Highest priority will be given to this loader. 2. **Extension ClassLoader** – Responsible for loading classes which are inside the ext folder **(jre\lib).** 3. **Application ClassLoader** –Responsible for loading Application Level Classpath, path mentioned Environment Variable etc.   The above Class Loaders will follow Delegation Hierarchy Algorithm while loading the class files.  ***1.2 Linking:***   1. **Verify** – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error. 2. **Prepare** – For all static variables memory will be allocated and assigned with default values. 3. **Resolve** – All symbolic memory references are replaced with the original references from Method Area.   ***1.3 Initialization:***  This is the final phase of Class Loading, here all static variables will be assigned with the original values, and the static block will be executed.  ***2. Memory Area:***  The Memory Area is divided into 5 major components:   1. **Method Area** – All the class level data will be stored here, including static variables. **There is only one method area per JVM, and it is a shared resource.** 2. **Heap Area** – All the Objects and their corresponding instance variables and arrays will be stored here. **There is also one Heap Area per JVM**. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe. 3. **Stack Area** – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three sub entities:    1. **Local Variable Array** – Related to the method how many local variables are involved and the corresponding values will be stored here.    2. **Operand stack** – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.    3. **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data. 4. **PC Registers** – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction. 5. **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.   ***3. Execution Engine:***  The bytecode which is assigned to the **Memory Area** will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.   1. **Interpreter** – The interpreter interprets the bytecode faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required. 2. **JIT Compiler** – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in interpreting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.    1. **Intermediate Code generator** – Produces intermediate code.    2. **Code Optimizer** – Responsible for optimizing the intermediate code generated above.    3. **Target Code Generator** – Responsible for Generating Machine Code or Native Code.    4. **Profiler** – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not. 3. **Garbage Collector**: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.   ***Java Native Interface (JNI):***  JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.  ***Native Method Libraries:***  This is a collection of the Native Libraries which is required for the Execution Engine. |

|  |
| --- |
| Explain class loading process  LOADING PROCESS OF CLASS:   * Memory is allocated for class * All methods are stored in method area * Static members are loaded in memory * Static variables which are not initialized will be initialized with default values * Static initializers will get executed from top to bottom leads to completion of loading process of class.   What is class loading?/how jvm works/JVM archtecture  ===========================CLASS LOADING SUB-SYSTEM====================================   * Class loading is nothing but Stroing class specific information(data and methods present in class) in in JVM memory * When we run java .class file class loading happens * JVM is invoked which creats memeory for class called as JVM memory * When JVM is invoked non-demon thrad i.e main thread and daemon thread garbage collector.Garbage collector has responsibility to remove unreferenced variable from memory so that memory again reclaimed for further execution * Main thread has responsibility to load .class file into JVM memory which is done by with help of three types of class loader and they are, * Bootstrap class loader:loads all nessesary inbuilt classes from rt.jar file which is byte code of inbuilt classes of java language present in JDK/JRE/Lib path * Extension loader :loads byte code from .exe folder of which is present in ??(extends bootstrap class loader) * Application loader: loads byte code from ?? folder of which is present in ??(extends Extension loader class loader) * After successfull loading of class file it’s linking process happens is through three sub-processes and that are, * Verification:in this process byte code of class is verified * Prepare:?? * Resolve:   ===============================================RUNTIME DATA AREAS====================================================   * After successful completion of linking process initialization done which completes class loading is complted and different areas are created in memory which are used for further process and that are, * Method area:consist of all methods of class and also consist of class static area where all static members are loaded during loading process of class in tabular format with unique refrence.?? * Stack area:Stack area is responsible for execution of java programe by creating a frames for each methods execution and terminating them after complete execution of of each method ?   Each frame further consist of three kinds of information which are ,   * Local variable array * Operand? * Frame data * ALL ARE USED FOR EFFICIENT MEMORY MANAGEMENT * Heap area:loads all instance member during loading process from that non-static initializers are excuted form top to bottom and before that uninitilized instance variables are initialized with default values * For each thread in stack area a pc register is allocated for performing operation * Native method stack area:used to excute code by calling native methods which are written in other programing language like c.   ===========================================EXECUTION ENGINE============================================================   * This system internally interpreter which uses ***jit compiler*** and ***profiler*** for efficient execution of program * *Use of native method interface* * *Use of native method library:Used to allocate program to processor* |

|  |
| --- |
| Explain multiple class file in java and rules associated with it? Explain public class in multiple class file  MULTIPLE CLASS AND IT'S RULES:   * We can create java file with multiple class * if file consist of a public class that class name must be given to java file * Otherwise we can give any name * multiple class allowed to have only one public class ,if there are more than one public class then it will lead to compile time error during execution. |

|  |
| --- |
| IMP NOTE:WHEN PRGRAM RUNS IT IS CONVERTED TO PROCESS |

|  |
| --- |
| What is constructor?  CONSTRUCTOR:   * what is constructor? * constructor is special kind of method whose name is same as class a. * It has no return type and never be prefixed with static modifier as it is non static member. * USE: * we use constructor in order to load non static members of object representing class in to newly created object * Also using constructor we can initialize object while creation of object * talking about instructions inside constructor body, we have several kinds of instructions and they are , * it may or may not consist of formal arguments depending on it's type. * constructor body consist of several types of instructions and they are, * instructions to load non static members * all non static initializers will be inserted in side constructor body during loading process * user defined instruction * There are two type of constructor , * A]no argument: do not contain any formal arguments and used to execute standard set of instruction or user defined instruction and   load all non static members of class into newly created object   * A]parameterized constructor: contains formal arguments and used to initialize attributes of an object during creation of object   and also to load all non static members of class into newly created object   * RULEs: * Having at least one constructor to each class is compulsory. * When programmer fails to add constructor in class ,compiler will implicitly add no argument constructor known as default constructor * It is compulsory to programmer to add no argument constructor in case when program declares any parameterized constructor * constructor chaining: * constructor calling other constructor for execution is known as constructor chaining. * we do constructor chaining in order to avoid duplication of code in multiple constructor * constructor overloading: * class can have multiple constructor from which one of will executed during object creation is known as constructor overloading. * we do constructor overloading in order to customize loading of attributes of object during creation of object. * (remember rules for constructor overloading...) |

|  |
| --- |
| Method overloading and overriding  Difference between method overloading and method overriding   * A]Polymorphism type: method overloading is compile time polymorphism while method overriding is run time polymorphism * B]member invocation decision: member invocation decision depends in compiler in method overloading while member invocation depends on type of object created. * C]member selection time: member selection done at compile time while member selection done at run time * D]binding type: method overloading comes under early binding while method overriding comes under lazy binding |

|  |
| --- |
| Explain method chaining and it’s rules  METHOD CHAINING :   * The process in which one method call another method for execution using object reference in known as method chaining.we use when we want to call multiple methods one by one without using object reference. |

|  |
| --- |
| Oops pricipals  OOP PRINCIPAL:   * Oops helps programmer to design s/w entity effectively and efficiently. * There are 4 types of oops principal and that are, * A]encapsulation * B]inheritance * C]polymorphism * D]abstraction |
| |  | | --- | | OOPs in java | | What is oops?  It is one of the approach to do programing. It deals with classes and objects.  In this we write programs related to real world entity to build software entity.  In simple word we can say that oops is one of the approach of programing in which we write program which are related to real world entity .It mainly deals with classes and objects  Class are used to blueprint for an object .These are user defined non-primitive datatypes which consist of all specifications of object.  Object are the real world entity who has exestance in real world.For example information about pen like pen type-pen ,pen-name,pen-price etc  Technically we can define objectas it is block of memory in heap area created at runtime which represent real world entity  To create object first we have to creat blueprint of it which we creat using classe  Object has two thhongs… | | Oops principals | |  | | What is oops? |  |  | | --- | | Relations in java | | What is relation?  Association between classes  Types of relations  IS-A vs HAS-A  (aggregation association)  Benefits(code reusablility| |reduce code duplication) | |  | |  |  |  | | --- | | package –interface-class | | Java.lang  -iterable(I) | |

|  |
| --- |
| Why we use OOPs principal  OBJECT ORIENTED PROGRAMING:   * what is object oriented programing ?   Writing or creating programs which are related to real world entity is known as object oriented programing   * why we use OOP? * We do object oriented programing to solve real world problem by modeling real world entities. * OOP follows DRY principal which makes programing easy and efficient * Object help us to model real problem and solve problem easily and efficiently. * What is Object? * Object is real world entity which has existence in real world. * Technically, we can define object as ,   it is block of memory which is created at runtime in heap area which is representation of real world entity.   * object= instance of a class * Object has two things , * A]Attributes/properties/details/features(represented by non static variables) * B]actions/behaviors(represented by non static methods) |

|  |
| --- |
| What is encapsulation?  A] Encapsulation:   * Wrapping attributes and behavior of object together is known as 'Encapsulation'. * As a programmer should not give direct access to object's attribute to the user   But the able should be able to read or modify the attribute of object through behavior   * therefore, it is good practice programmer to design a class. Such that we have methods to read as well as update attributes * Advantages:   Since we provide an indirect access to attribute it is secure and we can have validation steps before reading or updating and attribute.   * Encapsulation is the process is used to achieve data hiding. * What is data hiding? * it is process of restricting direct access to the attributes of an object and providing secured indirect access to the attribute/properties of object is known as data hiding. * several steps for achieve data hiding using encapsulation are * step01:makes the attributes private. * step02:Design getter and setter methods for reading and updating the attributes respectively * step03:This get flexibility to design the following, * we can make attribute readable but not modifiable * we can make attribute modifiable but not readable * we can make attribute both readable and modifiable * we can make attribute nether readable not modifiable |

|  |
| --- |
| Realtions in oops  RELATIONSHIP:   * Association between two class is known as Relationship * relationship is categorized into two types and the types are, * A]'has a' relation * B]'is a' relation * A]'has a' relation: One object is dependent on another object * a)composition: Composition represents a "whole part" relationship where the part cannot exist independently of the whole.   If the whole object is destroyed, its parts are also destroyed.   * composition and aggregation using * Early binding/Early instantiation:(implicit sub object creation in main object) * lazy binding/Lazy instantiation :use initializers to create sub type object * b)aggregation: Aggregation represents a "has a" relationship where the parts can exist independently of the whole. * The objects are related but do not have a strict dependency. * B]'is a' relation: parent child relation between object. Child acquires properties of parents |

|  |
| --- |
| What is inheritance?  B] INHERITANCE:   * If classes has 'is a ' relation between them where one class acts as parent * while other acts as child and child can inherit properties of parent by default .Such relation between two classes is known as inheritance * To achieve inheritance between two classes and interface we use 'extends' keyword . * but achieve inheritance between interface and class where interface will be acts as parent class we use 'implements' keyword * Q:INTERFACE NEVER INHERITS PROPERTIES OF CLASS EXCEPT OBJECT CLASS??????? * If two classes in inheritance then parent can use it's properties and behavior but child can use both it's own and parent's properties and behavior * Through relation we can achieve various characteristics few important among them are, * 1)by upcasting we can achieve generalization * 2) by downcasting achieve generalization * 3)we can achieve code reusability * [\*\*\*\*\*\*\*]private members ,static and non static initializers and constructor of class can't be inherited by derived class,   but still we can use them using derived class as constructor of base class is invoked during object creation of derived class through super call statement[?????????]  5 TYPES OF INHERITANCE   * A]Single level inheritance : If there is inheritance between class at single level then it is called as single level inheritance * B]multi level inheritance : If there is inheritance between class ,more than one level * C]Hierarchical inheritance : If base class ha multiple derived class then we can say there is hierarchical inheritance * D]multiple inheritance :If multiple base class having a same derived class then we can sat that there is multiple inheritance * E]Hybrid inheritance: combination of single, multilevel and hierarchical and multiple inheritance is known as hybrid inheritance * DIAMOND PROBLEM: when a derived class try to extends more than one base class then it will lead to ambiguity. The ambiguity is arises because of following two reason and that are * if multiple base classes have member with same identifier then it becomes ambiguity that which base class's member should be invoked. * super() call statement derived has ambiguity that which base class's constructor should be invoked   these ambiguities are termed as Diamond problem.   * Because of diamond problem classes in java allowed to inherit only single class. * so we can't achieve multiple inheritance in java using classes only but it can possible to achieve multiple inheritance using classes and interface   HOW ONE CLASS INHERIT ANOTHER CLASS ?   * Inheritance is achieved by using keywords 'extends' and 'implements', * if we want to achieve inheritance between two classes or two interface we use 'extends' keyword. * if we want to achieve inheritance between interface and class we use 'implements' keyword.   INTERFACE IN JAVA:   * 'interface' is a java member used to achieve pure abstraction and multiple inheritance * members that can be created in interface * by default all members are public[until 1.8] * by default all variables public static final * by default methods are public non static abstract before 1.8 * public static concrete method[since 1.8] * default non static concrete methods[since 1.8] * until 1.8 interface can't have concrete non static method * functional interface <java 1.8> and marker interface * interface can't have constructor ,so that can't be instantiated * *<extend vs implement AND inheritance between class and interface various combination>* * *<explain how multiple inheritance possible with classes and interface>* * *Types of interfaces with use* |

|  |
| --- |
| super() call STATEMENT :   * It is a statement used to invoke base class constructor during object creation of derived class. * It invokes all non static member during derived class object creation * It is used for constructor chaining between two class but the only condition is the classes should be in inheritance   What is super call satatement ?why we use it?  For constructor chaining what we use?  How to pass value from base to derived class  SUPER CALL STATEMENT:   * Use to invoke base class constructor to derived class object creation. Loads all non static members of base class to object of derived class * Pass data from base class to derived class * Used to constructor chaining more than one class * P(s&ns b)ic can't be inherited[3\*]private members ,static and non static initializers and constructor of class can't be inherited by derived class, * But still we can use them using derived class as constructor of base class is invoked during object creation of derived class through super call statement[?????????] * Explain how to pass value from derived class to base class using super call statement .[????????] * Super call statement implicitly added by compiler to the constructor of derived class in order to invoke non static members of base class during object creation of derived class * Non static member not inherited by inheritance[check is it correct???????????] |

|  |
| --- |
| What is polymorphism?  C] POLYMORPHISM :   * 'poly' many 'morphism' form * The ability of java members to have many form which represented by single identifier is known as Polymorphism. * out of these many forms one of the member get executed at a time * The polymorphism is mainly categorized in to two types , * A]compile time polymorphism * the polymorphism in which compiler makes decision at compile time which form of java member should be executed , * then that polymorphism termed as compile time polymorphism * the types of compile time polymorphism are , * method overloading * constructor overloading * method shadowing[!] * variable shadowing[!] * operator overloading * compile time polymorphism is known as compile time binding * It comes under early binding * B]run time polymorphism * Comes under lazy binding * the polymorphism which form of java member should executed at runtime based on object type invoked is known as run time polymorphism. * the types of run time polymorphism are , * A]derived typecasting:{mechanism...} * method overriding :mechanism of giving new implementation method body to non static method in base class through derived class present is known as method overriding * method overriding only applicable to non static method classes should be in 'is a' relationship * for successful method overriding we have to design same non static method as method present in base class with new implementation body and have to upcast derived class into base class(give detail explanation i.e how derived class method is given priority) |

|  |
| --- |
| Explain derived typecasting  \*\*\*DERIVED TYPECASTING\*\*\*   * The typecasting in which we convert one type of non primitive data to another primitive data is known as derived typecasting. * to apply derived typecasting on classes ,the classes should be in is a relationship * i.e they should be in inheritance * There are two types of typecasting and they are ,   A]upcasting :   * The typecasting in which derived type of object is converted into it's base type is known as upcasting . * we do upcasting in order to achieve 'Generalization'[explain term generalization????????????] * when we do upcasting then through derived class object we are allowed to use members of base class only. * Upcasting done by compiler implicitly.   B]Downcasting :   * The typecasting in which upcasted object is converted back to it's original base type is known as downcasting. * downcasting is the solution for problem occurred due to upcasting . * we do downcasting in order to achieve 'specialization' * downcasting done by programmer explicitly. |

|  |
| --- |
| Explain what is abstraction theoretically and technically  D] ABSTRACTION :   * process of hiding implementation body and just showing feature is known as 'Abstraction'.Process of showing features without implementation details=abstraction * we achieve abstraction using 'classes' and 'interface' * *<understand abstract and concrete method and class >* * *<when we prefix abstract to class and method>* |

|  |
| --- |
| What is object class ? why we use it?  OBJECT CLASS: what is 'object class ' why we use object class and non static methods in 'object class' with use   * *Methods used for multithreading purpose* * *Abstract method allowed to have for functional interface* |

|  |
| --- |
| Comparable vs comparator  comparable interface   * comparable functional interface is used for list type of collection for natural as well as customized sorting. * we can use this interface for sorting of customized object * present in package java.lang   compareTo() method   * this value compares attribute and returns value based on comparison * if the first element is smaller returns 1 * if the first element is smaller returns 1 * else return 0 * it takes single argument   comparator interface   * comparator functional interface is used for list type of collection for natural as well as customized sorting.[???????] * we can use this interface for sorting of customized object * present in package java.util * it takes two arguments   compare() method |

|  |
| --- |
| What is exception handling?  EXCEPTIONAL HANDLING USING JAVA :   * *<what is exception?>* * *<types of exception and difference>* * *<exception handling with simple vs complex try catch block>* * *<proper syntax for they block with multiple catch block>* * *<exception flow and AND exception hierarchy>* * *<finally block>*   *<exception object propagation>* |

|  |
| --- |
| What are wrapper class ? why we use it ?  WRAPPER CLASSES IN JAVA :   * *<what are wrapper classes in java with .value()?>* * *<types of exception and difference>* * *< boxing and auto boxing difference and notes>* |

|  |
| --- |
| Modifiers and it’s type  Modifiers and scope :   * *<modifiers and it's type and sub types with explanation>* * *<members can be prefixed with >* * *<final modifier with use>* * *<variable>* * *<method>* * *<class>* |

|  |
| --- |
| What is array why we use it?featues an of array  Array :   * *<Note>* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| What is string ?  STRING IN JAVA :   * *string and how to create string>* * *string is sequence of character enclosed in double coat.* * *in java to create string we have three classes 'String','StringBuilder','StringBuffer'* * *<string class>* * *<constructor of string class>* * *<way of creating string in java>(impact in memory)* * *<immutability of string in java>* * *<string builder and string buffer class and difference>* * *(DIFF:{constructor for conversion to #,asynchronized/synchronized, faster/slower,consumes less/more memory})*  |  |  | | --- | --- | | String comstant pool area | Heap area | | Literals are strogly refrenced throughout the program | Literals are may be weak refrenced it not in use | |

|  |
| --- |
| Give me difference between string builder and buffer  Difference between Buffer and builder   * A]synchronized/ Asynchronized * B]slower/faste * C]consumes memory-less/more |

|  |
| --- |
| Program to check reference in string pool area vs reference in heap area |
| // package javapgms.strings.stringConstantPoolArea;  public class Driver1 {  public static void main(String[] args) {  String inputString1 = "demo1";  String inputString2 = "demo1";  String inputString3 = new String("demo1");  System.out.println(inputString1 == inputString2);// true  System.out.println(inputString2 == inputString3);// false  System.out.println(inputString2.equals(inputString3));// true  System.out.println(inputString1.equals(inputString2));// true  }  } |
| True  false  true  true |

|  |
| --- |
| **COLLECTION HIRARCHY IN JAVA**   * Iterable(I) * Collection(I) * List(I) * ArrayList(C) * LinkedList(C) * Vector(C) * Stack(C) * CopyOnWriteArrayList(C?) * Set(I) * HashSet(C) * LinkedHashSet(C) * SortedSet(I) * NavigableSet(I) * TreeSet(C) * EnumSet(?) * ConcurrentSkipListSet(?) * CopyOnWriteArraySet(C?) * Queue(I) * PrivateQueue(C) * Dequeue(I) * ArrayDequeue(C) * LinkedList(C) * ConcurrentLinkedDeque * BlockingQueue(I) * ArrayBlockingQueue * LinkedBlockingQueue * PriorityBlockingQueue * SynchronusQueue * DelayQueue * ConcurrentLinkedQueue |

|  |
| --- |
| Collection framework in java  Collection framework in java:   * collection framework is set related classes and interface which provides mechanism to to store group of objects together * why we need framework?   To store multiple objects together. It is also possible to store multiple objects using array but array has limitations which are resolved by collections framework.   * limitations of array are : * Array size is fixed * During run time we can't increases or decreases size of an array * Array stores only homogenous data * array manipulation such as inserting element, removal element ,sorting of element requires complex logic * collection framework provides mechanism to perfoform operations such as,   create element,insert element,access element,remove element,replace elemnt,search element,update element,sort element thrpugh related classes interfaces methods and constructor   * collection allows non prmitive values * collection framework consist of two main hierarches and that are, * collection hierarchy * map hierarchy   What is collection framework?  It provides set of classes and interfaces that help In managing group of object  Why collection framework is introduced?   * Before the introduction of collection framework in jdk 1.2,java used to relay on a verity of classes like vector ,Stack ,HashTable and arrays to store and manipulate group of objects * These classes has some drowbacks which includes * Writing logic for group of objects are complex using these classes * Inconsistency:Each class has it’s own style to manage object which leads to confusion * Lack of inter-operability:They are not build to work together * No common interface: As there is absence of common interface we are unable to write algorithm which will operate on different kinds of collection * To solve these problem collection framework is introduced..   Advantages of collection framework are ,   * Unified architecture:consistent set of interfaces are there for all kinds of collection * Inter-operability:collections can be easily interchanged and manipulated in uniform way * Reusability:Generic algorithm can be written that work with any collection * Efficiency:The framework is capable of providing efficient algorithms for basic operations like searching ,sorting and manipulation |

|  |
| --- |
| What is collection in java?  Explanation on collection in java  Collection interface in java\*\*\*  collection is an interface present in java.util package  It is root interface of collection hierarchy which is present in collection framework  provides mechanism to store group of objects together  primitive data is not allowed in collection interface |

|  |
| --- |
| Java generics  Java generics   * Based on types of object stored in collection ,collection is categorized into two types * A] Generic collection(introduced in java 5) * Purpose : achive type safety and compile time error detection,to avoid explicit type casting,maintainable redable code due to type specification(eliminate type casting,avoid class cast exception) * Homogenous collection of object in which each object is stored into it's generic type. * Return type of object in generic collection is always generic type only. * B] non generic collation: * Purpose:used whwere type safty in not primery concern * Heterogenous collection of object in which each object is upcasted to object type from its generic type. * Return type of object stored in Non generic collection is always 'Object' type.  1. What is java generics? 2. Explain generic collection with purpose? 3. Explain non- generic collection with purpose? |

|  |
| --- |
| List (Interface):  List interface in collection   * List is an interface present in java.util package and introduced in jdk 1.2 * It is an sub interface of 'collection interface' * It uses list data structure for storing elements * Allows null and duplicate value   Featurs of list interface? |

|  |
| --- |
| Queue interface in collection  Queue :   * About queue data structure: * 'queue' is a data structure which FIFO data structure * internally uses array * element inserted from tail and extracted from head * About queue interface in java collection: * It is an interface present in java,util package. * it is an sub interface of 'map interface' which is root interface of 'map hierarchy' * methods: * {add(),remove(),element()|offer(),poll(),peek()} |

|  |
| --- |
| CLASS:PriorityQueue  'PriorityQueue' is concrete implementing class of 'Queue' interface which is subInterface of root interface 'Collection' present in java .util package  orders element using min heap datastructure  elements must be comparable type  does not allows null values  it is asynchronized  for synchronized version we can use PriorityBlockingQueue' class  Constructors :  methods: |

|  |
| --- |
| Map interface in collection   * About map data structure: * 'map' is a data structure which stores value in key value pair * keys can't be duplicate but value can be duplicate * one key can be associate with at most one value * we can access value from using key easily * About map interface in java collection: * It is an interface present in java,util package. * it is an sub interface of 'map interface' which is root interface of 'map hierarchy' * for collection of object it uses 'map' data structure * we can create generic map by providing key and values * 'Map' interface provides us 3 views and that are, * collection of values * set of keys * set of key value mapping |

|  |
| --- |
| CLASS:hashmap  'HashMap' is concrete implementing class of 'NavigableMap' interface which is subInterface of root interface 'Map'  hash map synchronized  internal implementayion of 'HashMap' is 'HashTable'  constructors of hashmap:  ??  ??  ?? |

|  |
| --- |
| Set interface in collection   * present in java.??? * It is an sub interface of 'collection interface' which is root interface of collection hierarchy of collection framework in java * It uses 'set' data structure for storing object and also perform operation on it. * duplicates are not allowed * only single null value is allowed * does not supports indexing |

|  |
| --- |
| CLASS:TreeSet  TreeSet is concrete implementing class of 'hashSet' and 'navigable set ' interfaces which is present in java.util.package  collection in tree object will be stored in sorted order by default  the element added to tree set must be of comparable type.  a class must implement comparable interface  the element entering in treeSet must be homoginous .If it is not homogenous will lead to  treeset and hashset internally uses treemap and hashmap respectively  treeset |

|  |
| --- |
| ArrayList class:   * present in java.util and introduced in jdk 1.2 * concrete implementing class of list interface * uses list data structure * -------------------------------------------------------------------------------------------------------------------------------------------------------------------- * allows duplicate elements ??? * allows multiple null values * supports indexing, indexing starts from zero * order of insertion is maintained * not synchronized * -------------------------------------------------------------------------------------------------------------------------------------------------------------------- * internally uses growable array which will increase size to to double of current size based on load factor and the load factor is 0.75 * -------------------------------------------------------------------------------------------------------------------------------------------------------------------- * methods * `.add(object)`:used to add element in array list * [p]ways to iterate list * using for loop * using for each loop * iterator * list iterator * difference between iterator and list iterator |

|  |
| --- |
| Comparator and compaarable |

|  |
| --- |
| Q]ITERATOR VS LISTITERATOR |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Linked list  Internal structure linked list :  WHEN I CAN USE IMPPLEMENTING CLASS MEMBERS WRT REFRENCE VARAIABLE?  LIST VS ARRAYLIST REFRENCE VARAIBLE   |  |  |  | | --- | --- | --- | |  | ArrayList | Linked list | | Memory allocation | contiguous | Random | | Memory requirement | Requires less memory | Requires more memory as it saves in the for of node | | Internal DS | Growable/dynamic array | Doubly Linked list | | Insertion/deletion | Not efficient o(n) | Efficient as o(1) | | Retrival/fetching | Efficient as o(1) | Not efficient o(n) | | Better for | Random element access | Insertion and deletion | | Behaves like | list | Stack and queue | |  | Asynchonised | |   --------------------------------------------------------------------------------------------------------------------------------------------------------------------  Linked list interface:   * concrete implementing class of list interface present in java.util package introduced in jdk 1.8 * It internally uses linked list data structure * Stores elements in node each node has two elements * Data: * Reference * Why complexity for insertion and deletion is o(n) * Synchronised * Consist of nodes * Each node consist od data and refrence of next object * Last and first element will point to null in doubly linked list * What is circular linked list? |

|  |
| --- |
| Vector [implementing class of list-interface]:   * introduced :jdk 1.0(legacy class) * implements list interface * it is synchronized/thread safe unlike arraylist and linked list * as it thread-safe takes more time and memory for processing so in case of single theaded sinario it recommended to use its other alternavtives like AL and LL * in case of thread safe concern we use vector * it’s initial capacity is 10 * unlike arraylist can check capacity using capacity() method * after crossing loadfactor vector size increses by 50 % * we can explicitly decide incrementing capacity using vector constructor * internal implementation is growable array * synchronization and performance(SS) * benefit of vector wrt synchronization (SS)   DIFFERENCE BETWEEN AL ,LL AND V? |

|  |
| --- |
| STACK[implementing class of list]:  Stack extends vector  Follows LIFO principal  We can use stack as well as vector functionalities  Search method of stack is one based indexing  How many ways to implement stack in java  Array.linked list-class , stack class,array-dqque |

|  |
| --- |
| CopyOnWriteArrayList:   * Use case:read insentive things are there (Reading is more than writing)   Internal working of CopyOnWriteArrayList |

|  |
| --- |
| Map:  Map is separate hierarchy   * In whch Map is root hierarchy * Stors value in key-value pair in which key is always unique and value can be duplicate * Do not extends collection interface * Key charecteristics of map interface(SS) |

|  |
| --- |
| HashMap   * Order of insertion is not maintained * Do not follows indexing * Key charecteristics(Ss) * Allow multiple values as null but single key value as null * Internal structure of hashmap:array of linked list[o(n)] or array of self balanceding binary tree/red black tree[0(logn)] after 1.8 * Threshold for conversion from linked list to r-b tree is 8->process termed as treefication * Key charecteristics(Ss) * How data is stored under hashmap * How collisions are handled? * Hashmap resizing (rehashing) after load factor 0.75[hashmap constructor with capacity and loadFactor] [size doubles] * During rehashing 🡪ss * Time complexity 🡪ss 2:40:14 * Use of equals() method for checking element * DiffObject class vs Objects class |
| |  | | --- | | CopyOnWriteArrayList:   * Use case:read insentive things are there (Reading is more than writing) * Internal working of CopyOnWriteArrayList | | Map:  Map is separate hierarchy   * In whch Map is root hierarchy * Stors value in key-value pair in which key is always unique and value can be duplicate * Do not extends collection interface * Key charecteristics of map interface(SS) | | HashMap   * Order of insertion is not maintained * Do not follows indexing * Key charecteristics(Ss) * Allow multiple values as null but single key value as null * Internal structure of hashmap:array of linked list[o(n)] or array of self balanceding binary tree/red black tree[0(logn)] after 1.8 * Threshold for conversion from linked list to r-b tree is 8->process termed as treefication * Key charecteristics(Ss) * How data is stored under hashmap * How collisions are handled? * Hashmap resizing (rehashing) after load factor 0.75[hashmap constructor with capacity and loadFactor] [size doubles] * During rehashing 🡪ss * Time complexity 🡪ss 2:40:14 * Use of equals() method for checking element * DiffObject class vs Objects class | | LinkedHashMap  Unlike hashmap order of insertion is maintained  Internally uses doubly linked list so cosumes more memory and will take more time  Lined hasmap is usedwhen we want to keep order of hashmap  CONSTRUCTOR:initial capacity,ic and load factor,ic lf and access order 3rd constructor 3rd val explanation  Cash eviction strategy 2.48.54 | | LRUcash:  WHEN TO USE COMPARABLE VS WHEN TO USE COMPARATOR ?   |  |  |  | | --- | --- | --- | | POC | comparable | Comparator | |  | Used to do sorting of object based on natural sorting of single attributes | Used to do sorting of object based on multiple attribute of object | |  | compareto () abstract method is used | compare() abstract method is used | |  | We have to write in class of an object | We can’t write in class of an object | | | Sortedmap:  Treemap:  Implementing class of sorted and navigable map  Internal implementation is r-b tree(self balancing binary search tree)[log(n)] | | hashTable:  internally uses array bins which stores value in LL DS only  hastable is synchronized | | Concurrent hashmap –class  In java works :segment based in java 7  In java 8 no segmentation |uses compare and swap approach 🡪no locking is there except resizing orcollesion | |

|  |
| --- |
| LinkedHashMap  Unlike hashmap order of insertion is maintained  Internally uses doubly linked list so cosumes more memory and will take more time  Lined hasmap is usedwhen we want to keep order of hashmap  CONSTRUCTOR:initial capacity,ic and load factor,ic lf and access order 3rd constructor 3rd val explanation  Cash eviction strategy 2.48.54 |

|  |
| --- |
| Sortedmap:  Treemap:  Implementing class of sorted and navigable map  Internal implementation is r-b tree(self balancing binary search tree)[log(n)] |

|  |
| --- |
| hashTable:  internally uses array bins which stores value in LL DS only  hastable is synchronized |

|  |
| --- |
| Concurrent hashmap –class  In java works :segment based in java 7  In java 8 no segmentation |uses compare and swap approach 🡪no locking is there except resizing orcollesion |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LRUcash:  WHEN TO USE COMPARABLE VS WHEN TO USE COMPARATOR ?   |  |  |  | | --- | --- | --- | | POC | Comparable | Comparator | |  | Used to do sorting of object based on natural sorting of single attributes | Used to do sorting of object based on multiple attribute of object | |  | compareto () abstract method is used | compare() abstract method is used | |  | We have to write in class of an object | We can’t write in class of an object | |

|  |
| --- |
|  |
| Multithreading in java |
| Runnable interface in java |
| Multitasking and its type and advantages of type(t) |
| Applications of multithreading(t) |
| What is thread ?Difference between thread and process(t) |
| Ways to create thread ? Which is best way and why?(t) |
| Methods of thread class and constructor with use explanation(t+p) |
| Difference between start() and run() method(t) |
| Uncertainty in o/p with explanation(t) |
| About ‘thread scheduler’(t) |
| Explain threads are independent .Write code to prove(t+p |
| Why we prevent threads ? ways to prevent threads?(t+p) |
| Difference between yield () ,join and sleep() method(t+p) |
| Why we interrupt method? How to interrupt method?(t+p) |
| Explain what is ‘data inconsistency problem’ ? How it is resolved ?(t+p) |
| Explain what is Synchronization?(t+p) |
| Tell me about ‘Synchronized’ keyword(t+p) |
| Give difference between Synchronized block and method(t+p) |
| Explain lock concept?(t) |
| Give difference between object level lock and class level lock(t) |
| Tell me about ‘ interthread communication’(t+p) |
| Tell me about deadlock(t) |
| Tell me about daemon thread(t+p) |
| What is method refrence?How method reference reducing code duplication |
| What will happens when number arguments are different? |
| Difference static method and non-static mrthod refrence |

|  |
| --- |
| Why java 1.8 introduced  \*\*\*java 1.8 features\*\*\*:  introduced in March 18,2014 by 'oracle' company  the main advantage of java 1.8 is to reduce the number of lines of code  concepts introduced in java 1.8 are ,  A] lambda expression[completed]  B] functional interface[completed]  c] static and default methods  D]:: operator (method reference)  E] constructor reference  F] predefined functional interface  G] streams  H] date and time  I] Optional class and other...(these concepts are used frequently)  these concepts leads to conciseness of code |

|  |
| --- |
| WHY WE ARE LEARNING JAVA 1.8?WHY IT IS INTRODUCED ?   * 1995 java 1.0 * 2012-2013 black period for java language .Started to migrate to other language leading to decrease value of java * Reasons are , * Lengthy boiler plate code (number of lines of code) * Requires more time for development * To overcome this problem they java language creators introduced java 1.8 (march 18,2014) * Version which reduce number of lines of code. Version incorporated several concept to achieve code conciseness * Due to this java survived and again people shifted back to java language because features or concepts introduced in java 1.8 * Answer: Most of the java project uses java 1.8 concept for code conciseness. without using java 1.8 code duplication percentage will be reduced. |

|  |
| --- |
| FEATURES INTRODUCED IN JAVA 1.8:   * lambda expression * functional interface * predefined functional interface * static and default methods * :: operator (method reference) * <> diamond operator * constructor reference * streams * date and time * optional class ….*<other features>* |

|  |
| --- |
| Functional interface in java  FUNCTIONAL INTERFACE:   * Interface having single an abstract method is called as functional interface * functional interface=SAM * Feature introduced in java 1.8 * To hold lambda expression we need functional interface reference variable * '@Functionalinterface' annotation is used to specify the functional interface * Inside functional interface we can have object class method as abstract method * We can have tostring(),equals() and hashcode() methods as abstract method * Why other are not abstract method * clone is native method * finalize() depreciated method * other are final , inside a functional interface we have object class method as abstract methods ,inside object class we have 11 methods, * 01) `public String toString()` * 02) `public boolean equals(Object obj)` * 03) `public int hashCode()` * ----------------------------------------------------------------------------------------- * 04) `protected Object clone() throws CloneNotSupportedException` * ----------------------------------------------------------------------------------------- * 05) `protected void finalize() throws Throwable` * ----------------------------------------------------------------------------------------- * 06) `public final Class<?> getClass()` * 07) `public final void notify()` * 08) `public final void notifyAll()` * 09) `public final void wait() throws InterruptedException` * 10) `public final void wait(long timeout) throws InterruptedException` * 11) `public final void wait(long timeout, int nanos) throws InterruptedException` |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lambda expression in java  CONCEPT 01:LAMBADA EXPRESSION:   * Anonymous function * It does not have modifiers * It does not have identifier/name * It does not have return type * To hold lambda expression we need a functional interface * The main objective of lambda of expression is enabling functional programing in java * Lisp is programing language where they used lambda expression for the first time * {functional programing is enabled with lambda expression released in java 1.8} * Functional interface=interface having single abstract method * '@FunctionalInterface' annotation is recommended to use * HOW CODE IS REDUCED? * While writing lambda expression writing datatypes are optional * Parenthesis and curly braces are optional in case of single arument and statement respectively otherwise it is compulsory * When there is single return statement ,should not write return statement otherwise will get error * Lambda expression is one of the way to implement functional interface * Number of argument in lambda expression and abstract method of functional interface must be equal  |  |  |  |  | | --- | --- | --- | --- | | (optional:formal arguments)-->{  //stmt  }; | ()-->{  SYSO("THIS IS MSG")  }; | ()-->SYSO("THIS IS MSG"); | -->SYSO("THIS IS MSG"); | | (int a)-->{return a\*a} | ( a)-->{return a\*a} | a-->{return a\*a} | a--> a\*a | | ( a,b)-->{return a\*b} | ( a,b)-->SYSO(a\*b) |  |  | |

|  |
| --- |
| *Program for checking scenarios of functional interface* |
| // package javapgms.oops.functionalinterface;  public class Driver1 {  public static void main(String[] args) {  }  }   |  |  |  |  | | --- | --- | --- | --- | | @FunctionalInterface  interface Demo1{  void m1();  } | @FunctionalInterface  interface Demo2{  void m1();  boolean equals(Object o);  } | // @FunctionalInterface  // interface Demo3{  // void m1();  // void m2();  // } | @FunctionalInterface  interface Demo4{  void m1();  String toString();  } | | @FunctionalInterface  interface Demo5{  void m1();  int hashCode();  } | // @FunctionalInterface  // interface Demo6{  // void m1();  // void finalize();  // } | // @FunctionalInterface  // interface Demo7{  // void m1();  // void notify();  // } | // @FunctionalInterface  // interface Demo8{  // void m1();  // Object clone();//as this is prefixed with protected  // } | | // @FunctionalInterface  // interface Demo9{  // void m1();  // Class getClass();  // } | // @FunctionalInterface  // interface Demo10{  // void m1();  // int hashCode();  // String toString();  // boolean equals();  // } |  |  | |
|  |

|  |
| --- |
|  |
| public class Driver {  public static void main(String[] args) {  // overiding method in functional interface usinf=g lambda expression  checkNumberEvenOdd var1 = (a) -> {  return (a % 2 == 0) ? true : false;  };  System.out.println(var1.checkNum(7));  // --------------------------------------------------------------------------------------  // no compulsion to use parenthesis when there is single formal argument  // no compulsion to use curly braces when there is single statement  // implesit return type  checkNumberEvenOdd var2 = a -> (a % 2 == 0) ? true : false;  System.out.println(var2.checkNum(6));  }  }  interface checkNumberEvenOdd {  boolean checkNum(int n);  } |
|  |

|  |
| --- |
|  |
| public class Driver3 {  public static void main(String[] args) {  Voter var1= a-> (a>18)?"is eligible for voating":"not eligible for voating";  System.out.println(var1.printVoterOrNot(19));  System.out.println(var1.printVoterOrNot(15));  Discount var2=a-> (a>60)?"Your are eligible for discount":"Your are not eligible for discount";  System.out.println(var2.printDiscountMsg(99));  System.out.println(var2.printDiscountMsg(15));  Authorization var3=a-> (a.toLowerCase().equals("admin"))?"You are Authorized..!":"You are not Authorized..!";  System.out.println(var3.printAuthorizationMsg("Admin"));  System.out.println(var3.printAuthorizationMsg("team lead"));  }  }  @FunctionalInterface  interface Voter {  String printVoterOrNot(int age);  }  @FunctionalInterface  interface Discount {  String printDiscountMsg(int age);  }  @FunctionalInterface  interface Authorization {  String printAuthorizationMsg(String roleName );  } |
| is eligible for voating  not eligible for voating  Your are eligible for discount  Your are not eligible for discount  You are Authorized..!  You are not Authorized..! |

|  |
| --- |
| *AISSIGNMENT ON LAMBDA EXPRESSION: give implementation for comparable ,comparator and runnable interface using lambda expression* |
| *assignment on static methods-TASK1/2* |
| import java.util.ArrayList;  import java.util.Collection;  import java.util.Collections;  import java.util.List;  public class Driver4AssignmentComparableComparator {  public static void main(String[] args) {  List<UserInfo> userInfoList=new ArrayList<>();  userInfoList.add(new UserInfo("AB-name1", 123));  userInfoList.add(new UserInfo("AA-name1", 456));  userInfoList.add(new UserInfo("C-name1", 789));  userInfoList.add(new UserInfo("D-name1", 126));  userInfoList.forEach(System.out::println);  System.out.println("------sorting using comparable-------------------------------------");  Collections.sort(userInfoList);  userInfoList.forEach(System.out::println);  System.out.println("----------sorting using comparator---------------------------------");  userInfoList.sort((o1,o2)->o2.name.compareTo(o1.name));  userInfoList.forEach(System.out::println);  }  }  class UserInfo implements Comparable<UserInfo>{  String name;  int id;  public UserInfo(String name, int id) {  this.name = name;  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  //lambda expression to sort object in desending order..!  Comparable<UserInfo> comparableObject = o -> o.name.compareTo(this.name);  @Override  public int compareTo(UserInfo o) {  return comparableObject.compareTo(o);  }  @Override  public String toString() {  return "id :"+id+" name :"+name;  }  } |
| id :123 name :AB-name1  id :456 name :AA-name1  id :789 name :C-name1  id :126 name :D-name1  ------sorting using comparable-------------------------------------  id :126 name :D-name1  id :789 name :C-name1  id :123 name :AB-name1  id :456 name :AA-name1  ----------sorting using comparator---------------------------------  id :126 name :D-name1  id :789 name :C-name1  id :123 name :AB-name1  id :456 name :AA-name1 |

|  |
| --- |
| *assignment on lambda expression-TASK2/2* |
| public class Driver4Assignment1Task {  public static void main(String[] args) {  Runnable thread1= ()-> {  for (int i = 0; i < 3; i++) {  System.out.println("thread 1..!");  }  };  Thread t1=new Thread(thread1);  t1.start();  Runnable thread2=()-> {  for (int i = 0; i < 3; i++) {  System.out.println("thread 2..!");  }  };  Thread t2=new Thread(thread2);  t2.start();  }  } |
| thread 1..!  thread 1..!  thread 2..!  thread 2..!  thread 2..!  thread 1..! |

|  |
| --- |
| *print collection of treeset in descending order* |
| public TreeSet(Comparator c){  //statement  }  PQ:chang the default sorting (asending order) of treeset collection to descending order |
| import java.util.Comparator;  import java.util.TreeSet;  public class Driver4 {  public static void main(String[] args) {  Comparator<Integer> comparatorObject = (o1, o2) -> o2.compareTo(o1);  TreeSet<Integer> treesetlist = new TreeSet<>(comparatorObject);  treesetlist.add(77);  treesetlist.add(9);  treesetlist.add(67);  treesetlist.add(48);  treesetlist.add(57);  System.out  .println("changing default ordering of treeset collection(desending..!)-----------------------------");  System.out.println(treesetlist);  }  } |
| [77, 67, 57, 48, 9] |

|  |
| --- |
| Static methods in java  STATIC METHODS IN INTERFACE :   * It is introduced in jdk 1.8 * The main purpose of introducing static methods in interface to reduce number of lines of code to * increase the code readability and remove the duplicate code also * we can call this method using method signature and interface name as reference * static methods can't be inherited the child class * the implementation which is common for all the Childs is written in static method in interface * we can have more than one static method inside interface * we can have main method inside a interface |

|  |
| --- |
| *assignment on static methods-TASK1/2* |
| // package javapgms.java8.practicals.StaticMethods\_02;  public class Driver1 {  public static void main(String[] args) {  Car1 ob1 = new Car1();  Bicycal ob2 = new Bicycal();  ob1.fuel();  ob2.fuel();  Vehical.commonActivity();  }  }  interface Vehical {  void fuel();  static void commonActivity() {  start();  run();  stop();  }  static void start() {  System.out.print("START ");  }  static void run() {  System.out.print("RUN ");  }  static void stop() {  System.out.print("STOP ");  }  }  class Car1 implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption:disel/petrol");  }  }  class Bicycal implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption: No fuel");  }  } |
| [COMMON FEATURES ] :calling | SMS | FM  [SMARTPHONES] : video call | play games |install and use apps | can run browser | camera  [TABLETS ] : can VIDEO games | can code |

|  |
| --- |
| *assignment on static methods-TASK2/2* |
| // package javapgms.java8.practicals.StaticMethods\_02;  public class Driver1 {  public static void main(String[] args) {  Car1 ob1 = new Car1();  Bicycal ob2 = new Bicycal();  ob1.fuel();  ob2.fuel();  Vehical.commonActivity();  }  }  interface Vehical {  void fuel();  static void commonActivity() {  start();  run();  stop();  }  static void start() {  System.out.print("START ");  }  static void run() {  System.out.print("RUN ");  }  static void stop() {  System.out.print("STOP ");  }  }  class Car1 implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption:disel/petrol");  }  }  class Bicycal implements Vehical {  @Override  public void fuel() {  System.out.println("fuel consumption: No fuel");  }  } |
| fuel consumption:disel/petrol  fuel consumption: No fuel  START RUN STOP |

|  |
| --- |
| Default methods in java  DEFAULT METHOD INSIDE A INTERFACE:   * we can have default method inside a interface from jdk 1.8 version onwards * a method which is default keyword is known as default keyword * default methods will get inherited to its child class * if you want to call /use default methods then object creation is compulsory * the implementation which is common for most of the classes but inside some class we have to override that method then we write implementation is default method |
| Static and default method difference |

|  |
| --- |
| Default methods in an interface(ClassExapmple1) |
| package defaultsMethod;  public class Driver1 {  public static void main(String[] args) {  new Human().walk();  new Cat().walk();  new Goat().walk();  }  }  interface Animal {  default void walk() {  System.*out*.println("Walking on 4 legs..!");  }  void eat();  void sound();  }  class Human implements Animal {  public void walk() {  System.*out*.println("Walking on 2 legs..!");  }  *@Override*  public void eat() {  System.*out*.println(" eat both..!");  }  *@Override*  public void sound() {  System.*out*.println(" eat both..!");  }  }  class Cat implements Animal {  *@Override*  public void eat() {  System.*out*.println(" eat both..!");  }  *@Override*  public void sound() {  System.*out*.println(" eat both..!");  }  }  class Goat implements Animal {  *@Override*  public void eat() {  System.*out*.println(" eat both..!");  }  *@Override*  public void sound() {  System.*out*.println(" eat both..!");  }  } |
| Walking on 2 legs..!  Walking on 4 legs..!  Walking on 4 legs..! |

|  |  |
| --- | --- |
| Static methods | Default methods |
| Methods prefixed with static modifier | Method which is prefixed with default modifier |
| Static methods can’t be inherited to it’ child class | Default methods can be inherited to its child class |
| We can use interface as a reference name | We can call default method by creating object of implementing child class |
| Use to provide implementation which is common for al implementing child classes | Use to provide implementation which is common for most of implementing child classes ,but is different for very few classes that will be written in default classes |
| Static methods can’t be overridden as not inherited by child class | default methods can be overridden as they can be inherited by child class |

|  |
| --- |
| Default methods in an interface(ClassExapmple2) |
| package defaultsMethod;  public class Driver2 {  public static void main(String[] args) {  new Plant().mobility();  new Animals().mobility();  new Humans().mobility();  new MiccoOrganisms().mobility();  }  }  interface LivingThings {  default void mobility() {  System.*out*.println("can move anywhere..!");  }  }  class Plant implements LivingThings {  public void mobility() {  System.*out*.println("movement is not possible in plant..!");  }  }  class Animals implements LivingThings {  }  class MiccoOrganisms implements LivingThings {  }  class Humans implements LivingThings {  } |
| movement is not possible in plant..!  can move anywhere..!  can move anywhere..!  can move anywhere..! |

|  |
| --- |
| Method refrence in java :  Features of method reference   * Concrete method and abstract methods * access modifiers can be anything * non-access modifier can be anything * method name can be anything * formal arguments same * if abstract method return type is void then concrete method ‘s return type can be anything * if abstract method return type is other than void then concrete method return type must be same as abstract method |

|  |
| --- |
| Static method refrence  Static method reference:   * syntax: class-name::method name ; * by using static method reference we are going to give implementation to abstract mrthod functional interface   Syntax : We can use static method reference for functional interface |

|  |
| --- |
| Method reference classExample1 |
| public class Driver1{  public static void main(String[] args) {  Services s1=()->System.out.println("welcome");  s1.greetingService();  Services s2=Driver1::greetings;  s2.greetingService();  }  static void  greetings(){  System.out.println("welcome using static method refrence..!");  }  }  interface Services{  void greetingService();  } |
| Welcome  welcome using static method reference |

|  |
| --- |
| Method reference classExample2 |
| public class Driver2{  public static void main(String[] args) {  User u1=Driver2::notification;  u1.placedOrderNotification();  }  static void  notification(){  System.out.println(" Your order placed successfully..!");  }  }  interface User{  void placedOrderNotification();  } |
| Your order placed successfully..! |

|  |
| --- |
| Method reference classExample3 |
| public class Driver3{  public static void main(String[] args) {  User u1=Driver3::alertmsg;  u1.sendAlertMsg();  }  static void  alertmsg(){  System.out.println(" Your account is hacked..!");  }  }  interface User{  void sendAlertMsg();  } |
| Your account is hacked..! |
|  |

|  |
| --- |
| Method reference classExample4 |
| public class Driver4{  public static void main(String[] args) {  User u1=Driver4::welcomemsg;  u1.sendAlertMsg();  }  static void  welcomemsg(){  System.out.println(" Thank you for choosing Axis Bank..!..!");  System.out.println(" You will welcome kit soon..!");  }  }  @FunctionalInterface  interface User{  void sendAlertMsg();  } |
| Thank you for choosing Axis Bank..!..!  You will welcome kit soon..! |

|  |
| --- |
| Non-static method reference :  Non-static method reference:   * By using non-static method reference we are going to give the existing non-static method implementation to abstract method of functional interface * If number of argument in abstract method and method present in driver method are different we will get compile time error |

|  |
| --- |
| Method reference classExample5 |
| public class Driver5Nonstatimethodrefrence{  public static void main(String[] args) {  Driver5Nonstatimethodrefrence ob1=new Driver5Nonstatimethodrefrence();  User u1=new Driver5Nonstatimethodrefrence()::otpMsg;  u1.generateandsendOtp();  }  void  otpMsg(){  System.out.println("  OTP sent to registered mobile number..!");  }  }  @FunctionalInterface  interface User{  void generateandsendOtp();  } |
| OTP sent to registered mobile number..! |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| When to go for lambda expression and method reference?   |  |  | | --- | --- | | Lambda expression | Method reference | | If the implementation is already not exist then we will go for lambda expression | If the implementation is already exist then we will go for method refrence | |

|  |
| --- |
| When to use lambda expression |
| Interface I1{  Void m1();  }  Class Driver {  Public static void main(){  I1 i1=()-> soln(“welcome..!”)  }  } |
|  |
|  |

|  |
| --- |
| When to use method refrence |
| Interface I2{  Void m1();  }  Class Driver {  Public static void main(){  I1 i1=new Driver()::print;  I1.m2();  }  Public void print(){  Soln(“msg ..!”);+  }  } |

|  |
| --- |
| When to go for r constructor reference?   * When abstract method of functional interface return type in non-primitive datatype then we will go for constructor refrence * In constructor refrence we are not worring about nuber of arguments also not worring about object creation ,So that instead of going individual class implementation or lambda expression it is better choice to go for constructor refrence * Syntax   Class-name::new;   * To hold constructor reference we need functional interface |

|  |
| --- |
| Individual class implementation Vs lambda expression implementation Vs constryctro refrence |
| public class Driver1 {  public static void main(String[] args) {  // way 2  Employee1Imp1 i1 = () -> {  return new Employee1();  };  i1.m1();  // way3  Employee1Imp1 i2 = Employee1::new;  Employee1 i3 = i2.m1();  }  }  class Employee1 {  Employee1() {  System.out.println("i am employee..!");  }  }  @FunctionalInterface  interface Employee1Imp1 {  Employee1 m1();  }  class Test implements Employee1Imp1 {  public Employee1 m1() {  Employee1 e1 = new Employee1();  return e1;  }  } |
| i am employee..!  i am employee..! |

|  |
| --- |
| Which constructor will be used by reference when we have overloaded constructor?  It depends on number of arguments which we declare in a abstract method present in functional interface |
| // package javapgms.java8.practicals.random;  class Student {  Student() {  System.out.println("student with no-argument constructor..!");  }  Student(int i, int i2) {  System.out.println("student  constructor with 1 argument..!");  }  }  @FunctionalInterface  interface studentInterface {  Student getStudent(int i1, int i2);  }  public class Driver2 {  public static void main(String[] args) {  studentInterface s1 = Student::new;  System.out.println(s1);  }  } |
| Driver2$$Lambda$1/0x0000000100072028@5f8ed237 |

|  |
| --- |
|  |
| //Package  Class MyClass extends Exception{  Public myException(String chat);  }  Interface CustomException{  Public static void main(){  CustomExceptions c=MyException::new;  try{  throw c.creatCustomException(“This is my custom exception..!”);  }catch(Exception e){  e.printStack();  }  }  } |
| Bsjdhbjkh jdiueoiuqioeub eiqp;o qoeihq qoihewf qoiehfe qieuhef liefb qlkiuef kiquewflif lqiuhfqwb qiuhqwh |

|  |
| --- |
| *{*  *predefined functional interface to perform common operation(jdk 1.8)*  *example:CRUD operation*  *instead of creating custom user defined functional interface we can use predefined functional interfacce*  *benefit:code readability and code reusability*  *we can use it directly*  *it makes development faster as programer nned not creat functional interface*  *}*  \*\*\*Predefined functional interface\*\*\*   * Predefined functional interface introduced in jdk 1.8 * whenever we want to perform common operation like arithmetic operations and Boolean type of operation * we explicitly we create a functional interface ,but these operations are common on day to day programmer life * instead of creating our own functional interface to perform common operation we are going to take the help of pre-defined functional interface those are 'predicate', 'function', 'consumer', supplier'..etc * the advantage of predefined functional interface is reducing the development time and it will make code as more readable |

|  |
| --- |
| *{*  *default and static method*  *and | or |not |is equal*  *\*\*\*predicate\*\*\*[use predicate in java.util\* package]*  *when to go for predecate predefine interface?*  *-for boolean operation*  *}*  Predicate:   * It is predefined functional interface * it is present in java.util.function package * it is having only one abstract method i.e test METHOD DECLARATION:public abstract test(T t) T-Type of parameter * introduced in jdk 1.8 * we use when we want to perform boo lean operation |

|  |
| --- |
| -[How functional interface is implemented before 1.8] |
| interface CheckNumber{  boolean chek(int i);  }  class ChildCheckNumber implements CheckNumber{  public boolean check(int i){  return i%2==0;  }  }  class Driver{  main(){  ChildCheckNumber ckn = new ChildCheckNumber();  System.out.println(ckn.check(10) );  }  } |
|  |

|  |
| --- |
| coading questions for presdicate predefined functional interface  p01]WPT check given string length is equal to 5 or not  p01]WPT check given is started with h or not  po3]WPT check given is end with 'A' or not  po3]string is having j or not  p]number is in the range or not(10,30) |
| // package javapgms.java8.practicals.predefinedFunctionalInterface;  import java.util.function.Predicate;  public class Driver1 {  public static void main(String[] args) {  //write a program to check givn number is even or odd using predefined functional interface  Predicate <Integer> isOperationNumber=n->n%2==0;  System.out.println(isOperationNumber.test(6));  //write a program to check givn number is positive or negative using predefined functional interface  isOperationNumber=n->n>0;  System.out.println(isOperationNumber.test(-6));  //write a program to check given string length is 5 or not  Predicate <String> isOperationString=n->n.length()==5;  System.out.println(isOperationString.test("abc"));  //WPT check given is started with h or not  isOperationString=n->n.charAt(0)=='A';  System.out.println(isOperationString.test("Abc"));  //WPT check given is end with 'A' or not  isOperationString=n->n.charAt(n.length()-1)=='H';  System.out.println(isOperationString.test("AbcH"));  //string is having j or not  isOperationString=n->n.contains("j");  System.out.println(isOperationString.test("AbcH"));  //number is in the range or not(10,30)  isOperationNumber=n->n>=10 && n<=30;  System.out.println(isOperationNumber.test(20));  }  } |
| True  false  false  true  true  false  true |

|  |
| --- |
|  |
| // package javapgms.java8.practicals.predefinedFunctionalInterface;  import java.util.function.Predicate;  public class Driver2 {  public static void main(String[] args) {  Predicate<Object> isProperty = s -> s.toString().length() == 4;  String arr[] = { "abcd", "difsd", "abhgcd" };  for (String i : arr) {  if (isProperty.test(i))  System.out.println(i);  }  int arr2[] = { 1, 2, 3, 4, 5, 6, 7, 8 };  isProperty = s -> (Integer) s % 2 == 0;  for (int i : arr2) {  if (isProperty.test(i))  System.out.print(i+" ");  }  }  } |
| Abcd  2 4 6 8 |

|  |
| --- |
| Assignment:5 examples of predicate |
| // package Predicatefucnctionalinterface;  /\*\*  \* Title: Predicate-Based Validation Program  \*  \* Description:  \* This program demonstrates the use of Java's Predicate functional interface  \* for performing various validations and checks. It includes examples such as  \* email validation, password strength checking, phone number validation,  \* discount eligibility, and examination pass criteria.  \*  \* Author: [Sushant shendage]  \* Date: [28/12/24]  \*/  import java.util.function.Predicate;  public class Driver3Assignment {  public static void main(String[] args) {  // Program for email validation  Predicate<Object> isProperty = n -> (n.toString().contains("@gmail.com"))  && (n.toString().length() - 10 > 1)  && (n.toString().charAt(0) > '0' && n.toString().charAt(0) > '9');  System.out.println("1] Valid email :" + isProperty.test("ww@gmail.com"));  // Checking if the object is null or not  isProperty = n -> n != null;  System.out.println("2] Presence of object :" + isProperty.test(null));  // Validating password strength  isProperty = n -> n.toString().length() >= 6;  System.out.println("3] Strong password :" + isProperty.test("123456"));  // Phone number validation (10 digits, starts with 7, 8, or 9)  isProperty = n -> (n.toString().length() == 10)  && (n.toString().charAt(0) == '7' || n.toString().charAt(0) == '8' || n.toString().charAt(0) == '9');  System.out.println("4] Number validation :" + isProperty.test("9123456789"));  // Checking discount eligibility based on user input amount  isProperty = n -> (Integer) n >= 1000;  System.out.println("5] Discount applied :" + isProperty.test(236));  // Determine if a student has passed the exam based on marks  isProperty = n -> (Double) n >= 35.00;  System.out.println("6] Exam passed :" + isProperty.test(35.67));  }  } |
| 1] Valid email :true  2] Presence of object :false  3] Strong password :true  4] Number validation :false  5] Discount applied :false  6] Exam passed :true |

|  |  |  |  |
| --- | --- | --- | --- |
| Assignm | | | |
| POD | Predicate | function | Consumer |
| Use | Used for Boolean operation | Used for other thsn Boolean operation | Used for printing purpose |
| Abstract method | test() | apply() | accept() |
| Return type : | Boolean | Generic | void |
| prototype | Boolean test(T t) | R apply(T t) | Void accept(T t) |
|  | Predicate <T> | Function<T,R> | Consumer<T> |

|  |
| --- |
| Stream API:   * Introduced in jdk 1.8 present in java.util package * Stream API will help us to perform bulk operation on collection object like like filter,manipulation,searching,sorting,min,max,counting etc   Stream:   * it predefined interface present in java.util.stream package * if you want to perform a bulk operation on collection object then we need some predefined methods, all those methods are present inside a stream interface * some of the predefined stream interface methods are filter(),map(),sorted(),sorted(-),count(),min(),max(),distinct()   *WHAT ARE VAR-ARG METHODS?*  *WHAT IS THE MEANING OF … IN VAR-ARG METHOD?*  *WHAT IS THE USE OF STREAM DEFAULT METHOD PRESENT IN COLLECTION INTERFACE?*  *USE OD COLLECT METHOD*  *TELL ME ABOUT COLLECT METHOD*  *TELL ME ABOUT COLLECTOR Interface*  *TELL ME ABOUT COLLECTORS final class* |

|  |
| --- |
| Stream ()method in collecton interface:   * It is a default method present in collection interface * It will help us to covert collection object to stream object ,so that we can apply the stream interface methods on collection object based on the requirement(example select the even element) * METHOD DECLARATION: default Stream<E> stream() |

|  |
| --- |
| Filter   * It is abstract method present inside stream interface * It will take predicate as argument * Whenever we want to perform filter type operation or Boolean type operation then we go for filter() method * METHOD DECLARATION: Stream <T> filter(Predicate p); |

|  |
| --- |
| Collect () method:   * It is abstract method present in stream interface * It will take collector as a argument * It will helps to store/collect the elements after performing the operations on collection object * METHOD DECLARATION : |

|  |
| --- |
| Collectors class:   * It is final predefined class and present in java.util.stream package * It is having so many static methods like toList(),toMap(),toSet() …etc   It will help us to convert that respective collection object |

|  |
| --- |
| Collector interface:   * It is a predefined interface present in java.util.stream package |

|  |  |  |
| --- | --- | --- |
| stream() | Stream –interface |  |
| collect() | Collectors-class | Collector -interface |

|  |
| --- |
| WAP to print only even elements by using stream API  WAP to print element which is greter than 15 |
| // package javapgms.java8.practicals.stream06;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  import java.util.function.Predicate;  import java.util.stream.Collector;  import java.util.stream.Collectors;  import java.util.stream.Stream;  public class Driver1 {  public static void main(String[] args) {  List <Integer> list1= new ArrayList<Integer>() ;  list1.add(1);  list1.add(2);  list1.add(3);  list1.add(4);  list1.add(5);  list1.add(6);  Stream <Integer> streamVar= list1.stream();  Predicate <Integer> p1=n->n%2==0;  Stream <Integer> filter1=streamVar.filter(p1);  List<Integer> collect1=filter1.collect(Collectors.toList());  collect1.forEach(e->System.out.println(e));  list1.stream().filter(n->n%2==0).collect(Collectors.toList()).forEach(e->System.out.println(e));  }  } |
| 2  4  6  2  4  6 |
| // package javapgms.java8.practicals.stream06;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  import java.util.function.Predicate;  import java.util.stream.Collector;  import java.util.stream.Collectors;  import java.util.stream.Stream;  public class Driver2 {  public static void main(String[] args) {  List <Integer> list1= new ArrayList<Integer>() ;  list1.add(1);  list1.add(2);  list1.add(3);  list1.add(44);  list1.add(55);  list1.add(66);  list1.stream().filter(n->n>=15).collect(Collectors.toList()).forEach(e->System.out.println(e));  }  } |
| 44  55  66 |
| // package javapgms.java8.practicals.stream06;  import java.util.ArrayList;  import java.util.List;  public class Driver3 {  public static void main(String[] args) {  ArrayList<Student> studentInfo= new ArrayList<Student>();  studentInfo.add(new Student(101, "Aname1", "add1"));  studentInfo.add(new Student(101, "Aname1", "add2"));  studentInfo.add(new Student(101, "Bname1", "add3"));  studentInfo.add(new Student(101, "Cname1", "add4"));  studentInfo.stream().filter(n->n.name.charAt(0)=='A').forEach(e->System.out.println(e));  }  }  class Student{  int id;  String name;  String addr;  public Student(int id, String name, String add) {  this.id = id;  this.name = name;  this.addr = add;  }  @Override  public String toString() {  return "student [id=" + id + ", name=" + name + ", add=" + addr + "]";  }  }  //HOW TO DO USING ITERATOR |
| student [id=101, name=Aname1, add=add1]  student [id=101, name=Aname1, add=add2] |

|  |
| --- |
| *Creat a employee object and print employees whoes salary is greater than 15000[A:1/5]* |
| import java.util.ArrayList;  import java.util.List;  public class Driver4 {  public static void main(String[] args) {  List<Employee> employeeInfo = new ArrayList<Employee>();  employeeInfo.add(new Employee(723, "a-emp", 45000));  employeeInfo.add(new Employee(923, "b-emp", 35000));  employeeInfo.add(new Employee(323, "c-emp", 8000));  employeeInfo.add(new Employee(823, "d-emp", 56000));  employeeInfo.add(new Employee(223, "e-emp", 10000));  employeeInfo.stream().filter(n -> n.salary >= 15000).forEach(e -> System.out.println(e));  }  }  class Employee {  int id;  String name;  double salary;  public Employee(int id, String name, double salary) {  this.id = id;  this.name = name;  this.salary = salary;  }  @Override  public String toString() {  return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";  }  } |
| Employee [id=723, name=a-emp, salary=45000.0]  Employee [id=923, name=b-emp, salary=35000.0]  Employee [id=823, name=d-emp, salary=56000.0] |

|  |
| --- |
| Print student whose address is banglore 2/5 |
| import java.util.ArrayList;  import java.util.List;  public class Driver5 {  public static void main(String[] args) {  List<StudentDetails> list1 = new ArrayList<>();  list1.add(new StudentDetails(34, "a-name", "banglore"));  list1.add(new StudentDetails(45, "b-name", "X-address"));  list1.add(new StudentDetails(67, "c-name", "banglore"));  list1.add(new StudentDetails(90, "d-name", "E-address"));  list1.add(new StudentDetails(21, "e-name", "w-address"));  list1.stream().filter(n -> n.address.equals("banglore")).forEach(e -> System.out.println(e));  }  }  class StudentDetails {  int id;  String name;  String address;  public StudentDetails(int id, String name, String address) {  this.id = id;  this.name = name;  this.address = address;  }  @Override  public String toString() {  return "StudentDetails [id=" + id + ", name=" + name + ", address=" + address + "]";  }  } |
| StudentDetails [id=34, name=a-name, address=banglore]  StudentDetails [id=67, name=c-name, address=banglore] |

|  |
| --- |
| *Create a teacher class and print teacher object whose name length will be equal to 5 3/5* |
| import java.util.ArrayList;  import java.util.List;  public class Driver6 {  public static void main(String[] args) {  List <Teacher> teacherInfo=new ArrayList<>();  teacherInfo.add(new Teacher(67, "acudb"));  teacherInfo.add(new Teacher(78, "xy"));  teacherInfo.add(new Teacher(90, "prxsq"));  teacherInfo.add(new Teacher(120, "cd"));  teacherInfo.stream().filter(n -> n.teachername.length()==5).forEach(e -> System.out.println(e));  }  }  class Teacher {  int teacherId;  String teachername;  public Teacher(int teacherId, String teachername) {  this.teacherId = teacherId;  this.teachername = teachername;  }  @Override  public String toString() {  return "Teacher [teacherId=" + teacherId + ", teachername=" + teachername + "]";  }  } |
| Teacher [teacherId=67, teachername=acudb]  Teacher [teacherId=90, teachername=prxsq] |

|  |
| --- |
| *Create a developer object and print only the developer’s who is having destination as java developers* *4/5* |
| import java.util.ArrayList;  import java.util.List;  public class Driver7 {  public static void main(String[] args) {  List<DevloperDetails> devInfo = new ArrayList<>();  devInfo.add(new DevloperDetails(67, "java-devloper", "d1"));  devInfo.add(new DevloperDetails(78, "python-devloper", "d2"));  devInfo.add(new DevloperDetails(90, "sql-devloper", "d3"));  devInfo.add(new DevloperDetails(120, "java-devloper", "d4"));  devInfo.stream().filter(n -> n.designation.equals("java-devloper")).forEach(e -> System.out.println(e));  }  }  class DevloperDetails {  int id;  String designation;  String domainName;  public DevloperDetails(int id, String designation, String domainName) {  this.id = id;  this.designation = designation;  this.domainName = domainName;  }  @Override  public String toString() {  return "DevloperDetails [id=" + id + ", designation=" + designation + ", domainName=" + domainName + "]";  }  } |
| DevloperDetails [id=67, designation=java-devloper, domainName=d1]  DevloperDetails [id=120, designation=java-devloper, domainName=d4] |

|  |
| --- |
| map() method:   * It is abstract method present inside a stream() interface * It will take function as a argument * If you want to perform any manipulation operation or other than Boolean operation then we go for a map () method   WHEN WE WILL GO FOR FILTER() METHOD?  Terminal vs non-terminal methods? |

|  |
| --- |
| *Map method practical(CP1)* |
| // package javapgms.java8.practicals.stream06.con2;  import java.util.ArrayList;  import java.util.List;  import java.util.stream.Collectors;  public class Driver1 {  // Approch 1:double the elements in collection using forEach loop/(WITHOUT  // STREAM)  // Approch 2:using stream API  public static void main(String[] args) {  // Approch 2:using stream API  List<Integer> intList = new ArrayList<>();  intList.add(77);  intList.add(9);  intList.add(67);  intList.add(48);  intList.add(57);  intList.add(100);  // streamof integer : intList.stream().map()  // collector argument=collectors.toX()  intList.stream().map(n -> n \* 2).forEach(System.out::println);  System.out.println("--------------------------------");  List<Integer> dobleIntList = intList.stream().map(n -> n \* 2).collect(Collectors.toList());  dobleIntList.forEach(System.out::println);  }  } |
| 154  18  134  96  114  200  --------------------------------  154  18  134  96  114  200 |

|  |
| --- |
| *filter() and map() method practical (c)*  *Different ways to print stream element after performing stream operation* |
| import java.util.ArrayList;  import java.util.List;  import java.util.stream.Collectors;  import java.util.stream.Stream;  public class Driver2 {  public static void main(String[] args) {  List<Integer> intList = new ArrayList<>();  intList.add(77);  intList.add(9);  intList.add(67);  intList.add(48);  intList.add(57);  intList.add(100);  //way -1  Stream<Integer> streamObject = intList.stream();  Stream<Integer> streamObjectFilter = streamObject.filter(n -> n >= 50);  Stream<Integer> streamObjectFiltermap = streamObjectFilter.map(n -> n \* 2);  List<Integer> ansList2 = streamObjectFiltermap.collect(Collectors.toList());  ansList2.forEach(System.out::println);  System.out.println("-------------------------------");  //way-2  List<Integer> ansList = intList.stream().filter(n -> n >= 50).map(n -> n \* 2).collect(Collectors.toList());  ansList.forEach(System.out::println);  }  } |
| 154  134  114  200  -------------------------------  154  134  114  200 |

|  |
| --- |
| *Map and filter method on customize objects* |
| import java.util.ArrayList;  import java.util.List;  import java.util.stream.Collectors;  public class Driver3 {  public static void main(String[] args) {  List<Employee> empInfo = new ArrayList();  empInfo.add(new Employee(1, "emp1", 30000));  empInfo.add(new Employee(1, "emp2", 34000));  empInfo.add(new Employee(1, "emp3", 50000));  empInfo.add(new Employee(1, "emp4", 60000));  List<Employee> ansEmpInfo = empInfo.stream().filter(n -> n.salary >= 50000).map(n -> {  n.salary \*= 1.20;  return n;  }).collect(Collectors.toList());  ansEmpInfo.forEach(System.out::println);  System.out.println("---------------------");  empInfo.stream().filter(n -> n.salary >= 50000).map(n -> {  n.salary \*= 1.20;  return n;  }).forEach(System.out::println);  }  }  class Employee {  int id;  String name;  double salary;  public Employee(int id, String name, double salary) {  this.id = id;  this.name = name;  this.salary = salary;  }  @Override  public String toString() {  return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";  }  } |
| Employee [id=1, name=emp3, salary=60000.0]  Employee [id=1, name=emp4, salary=72000.0]  ---------------------  Employee [id=1, name=emp3, salary=72000.0]  Employee [id=1, name=emp4, salary=86400.0] |

|  |
| --- |
| forEach() method:   * it is abstract method present inside a stream interface * it will take consumer as a argument * once we perform operation on collection object by using stream API directly we can print element on console without storing a collection then we will go for forEach method |

|  |  |
| --- | --- |
| *filter() vs map()* | |
| filter() | map() |
| Used to perform Boolean operation | Used to perform operation other than Boolean operation |
| Takes predicate as argument | Takes function as argument |
| Uses test method internally | Uses apply method internally |
| Filter may cause reduce in number of collection | Opposite |

|  |
| --- |
| process data in decalrative manner  FOR CONFIGURATION WHICH METHODS ARE USED ?  Filter and map  FOR PROCESS WHICH METHODS ARE USED?  Collect ,foreach ,min,max,count,sort  For typesafty purpose we have java generics  We can make class generic  Print filtered collection in single line  Uppercase format of collection |

|  |
| --- |
|  |
| // package java8.practicals.stream06.concept;  import java.util.ArrayList;  import java.util.List;  public class Driver2 {  public static void main(String[] args) {  List <Movie> movieList1=new ArrayList<>();  movieList1.add(new Movie("m1", "l1", 7.5, 2.5));  movieList1.add(new Movie("m2", "l1", 5.5, 3));  movieList1.add(new Movie("m3", "l2", 7.0, 1.5));  movieList1.add(new Movie("m4", "l3", 9.0, 3));  movieList1.add(new Movie("m5", "l2", 6.5, 2.5));  movieList1.stream().map(n->{n.movieName=n.movieName.toUpperCase();  return n;  } ).forEach(System.out::println);  }  }  class Movie{  String movieName;  String language;  double rating;  double duration;  public Movie(String movieName, String language, double rating, double duration) {  this.movieName = movieName;  this.language = language;  this.rating = rating;  this.duration = duration;  }  @Override  public String toString() {  return "Movie [movieName=" + movieName + ", language=" + language + ", rating=" + rating + ", duration="  + duration + "]";  }  } |
| Movie [movieName=M1, language=l1, rating=7.5, duration=2.5]  Movie [movieName=M2, language=l1, rating=5.5, duration=3.0]  Movie [movieName=M3, language=l2, rating=7.0, duration=1.5]  Movie [movieName=M4, language=l3, rating=9.0, duration=3.0]  Movie [movieName=M5, language=l2, rating=6.5, duration=2.5] |

|  |
| --- |
|  |
| import java.util.ArrayList;  import java.util.Comparator;  import java.util.List;  public class Driver1minMaxCountmethod {  public static void main(String[] args) {  List <Integer> intList1=new ArrayList<>();  intList1.add(77);  intList1.add(9);  intList1.add(67);  //comparator object  Comparator<Integer> comp=(o1,o2)->o1.compareTo(o2);  //sorting list using stream  [sorted() method]  intList1.stream().sorted(comp).forEach(System.out::println);  //countmethod  System.out.println("count:"+intList1.stream().count());  //min method  System.out.println("Min :"+intList1.stream().min(comp).get());  //max method  System.out.println("Max :"+intList1.stream().max(comp).get());  }  } |
| 9  67  77  count:3  Min :9  Max :77 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Methods classification in stream interface :  In java8 they have introduced stream API concept inside a stream interface we are having so many abstract methods and they are categorized as,   * Terminal methods * Non-terminal methods  |  |  | | --- | --- | | Non-terminal Methods:  The methods whoes return type is Stream from stream interface are non-terminal methods. | Terminal methods :  The methods whoes return type is other than Stream from stream interface are non-terminal methods. | | filter(),map(),reduce(),sorted(),sorted(Comparator c)[stream] | Collect()[collection],forEach[void],min()[optional],max()[optional],reduce[T],count[long] | | Help to configure data before performing operation on it | Help to print or return data after performing operation | |

|  |
| --- |
| Date and time API:   * Before gdk 1.8 if you want to print date common time then we use class called date present inside or java.util package but that class’s methods are not convenient for programmer as class most of the methods are deprecated * Date date1=new Date();   System.out.println(date1);  Int date/day/hours=date1.getDate()/Day()/Hours()   * WHAT ARE THE PRECATED METHODS?   THE METHODS WHICH ARE UNDER MAINTENANCE AND HAS HIGH CHANCES TO REMOVE OR REPLACE   * In Java it is always recommended to to avoid depricated code to overcome above problem we go for which is introduced in gdk 1.8 * Date and time API is also known as JODA API as it is introduced by JODA organisation .   The classes which are introduced in Joda API or date and time API is present in your time package .  Example: LocalDate() ,LocalTime, LocalDateAndTime()   * If you want to print current system date then we go for a class called LocalDate() * Local date it is a final class * it is a having so many methods like now() and of() etc * if you want to create object of local date class then there are various ways but in that we are going to take the help of method i.e LocalDate() * LocalDate date1= date1; Sysyem.out.hpPrintln(date1); * Once we got a local late object from that local it object we can print only year or we can print only Day date on month with the help of below methods * we are having some non starting method inside local it class getYear(), getMonthValue(), getDayOfMonth() * all these three methods are non static methods with the return type int * by using this 3 methods I can print only year I can print only month and I can print only day from the local date object * LocalDate date2=LocalDate .now()   Int yyyy=date1.getYear()/getMonthValue()/getDayOfMonth();   * Whenever we are printing a local date format year May month but I want to print in my own format then we will go for below by extracting day month and year * To print any data on particular format we will go for a printing statement called we will go for printing setment called system |

|  |
| --- |
| LocalTime:   * It is final class present in java.Time package * It is having so many static and non-static methods like now() method ,getHour() method,getMinute(),getSecond(),getNano() etc. * If you want to print current time then will write a code as below * Once we got a local time object I can print only hour/minute/second with the help of below methods * If we want to print Local time in our own format usinfg printf() statement |

|  |
| --- |
| *Extraction of hour ,minute and second from LocalTime object* |
| // package javapgms.dateAndtimeAPI;  import java.time.LocalTime;  public class Driver1 {  public static void main(String[] args) {  LocalTime time1 = LocalTime.now();  System.out.println(time1);  System.out.println("getHour()   : " + time1.getHour());  System.out.println("getMinute() : " + time1.getMinute());  System.out.println("getSecond() : " + time1.getSecond());  System.out.println("getNano()   : " + time1.getNano());  System.out.printf("%d--%d--%d--%d",time1.getHour(),time1.getMinute(),time1.getSecond(),time1.getNano());  }  } |
| 17:27:56.047  getHour() : 17  getMinute() : 27  getSecond() : 56  getNano() : 47000000  17--27--56—47000000 |

|  |
| --- |
| Period Class[period=Timequntity(how many days/year/month)]   * If you want to represent quantity of date and time in days/year/months then we go for a concept called “Period” * Period is final class present in java.lang package * If you want to creat a object of this class will take help of method between() method [other methods are also there] * between () is static method * once we got Period object then we can print only year/month/days with the help of below methods   int getYears/Days/Months() all are non-static method with return type int |

|  |
| --- |
| LocalDateTime:   * If you want to represent both date and time simultaneously then we will go for a class called as LocalDateTime * It is final class * It is also having some static and non-satic methods inside it * To creat objrct of this class we use static method now() * LocalDateTime().now() method: * Once we got a localTimeDate object then we can print only date/month/year/hour/minutes/seconds with help of following non-static methods * getYear/MonthValue/dayOfMonth/Hour/Minute/Seconds() * if you want to print date and and time object in any format simultaneously then we go for printf() method * printf(String s),printf(Object …variable\_name)   WHAT ARE FINAL METHODS?  StringBuilder, StringBuffer,Scanner Class,Colloctors,System.Localdatetime,Localdate,Localtime |

|  |
| --- |
| Of() method for customized date and time object:   * If we want to print our customized date and time object then we go for a method called as of method which present inside lacaldatatime class * It is a static and overloaded method * After N –days/weeks/hours/minutes/seconds * We have to write in camalCase * All methods will take Long as argument🡪 (plus/minus)Monthes/hours/minutes/day/year/nanos/seconds/weeks * All method’s return type is localDateTime * All are non-static methods |

|  |
| --- |
| *Practical on “Period’-concept* |
| import java.time.LocalDate;  import java.time.Period;  public class Driver3Period\_timeQuantity {  public static void main(String[] args) {  LocalDate startingDate = LocalDate.of(2000,4,6);  LocalDate endingDate = LocalDate.of(2025,1,10);  Period p=Period.between(startingDate, endingDate);  System.out.println(p);  //-----------------------------------  System.out.println("year   : "+p.getYears());  System.out.println("month  : "+p.getMonths());  System.out.println("days   : "+p.getDays());  //-----------------------------------  //printing age using Period-class  System.out.printf("I am %d years old..!",Period.between(LocalDate.of(2000,4,6), LocalDate.of(2025,1,10));  }  } |
| P24Y9M4D  year : 24  month : 9  days : 4  I am 24 years old..! |

|  |
| --- |
| Optional class  {{{  Devloper has to explicitly handle NPE  Writing same boiler plate code lead to writing duplicate code  For above we use Optional class by using which we can avoid NPE  Throws keyword is used for checked exception  }}}  Optional class:   * In java there are so many common exception which will occur at runtime * Among those common exception there is one exception called NullPointerException * NPE:whenever we are performing any operation on NULL then we get a NullPointerException   Problems without optional class:   * If optional class is not introduced then developer has towrite a code a to avoid a NullPonterException each and every time * When developer is writing a code explicitly then he has to check possibilities of NPE in each and every methods,class..etc * When a developer is writing a code again and again to avoid NPE then there might be chance of duplicate code * When we are a writing a duplicate code then it will affect on code redability[boiler plate code=writing duplicate code again and agian] * To avoid above problems oracl peple introduced a concept called Optional class in jdk 1.8   Optional class:   * It is final class * Present in * Main purpose of optional class is to avoid NPE * Creating object of optional class * There are various a way to create a object of optional class inthat we are using of() method and ofNullable() method * Of()method   + Static method   + Take argument as generic * When you are argument as null then of() method will throws NPE * When you are a passing a value which is non-null value then optional calss objecti s created |

|  |
| --- |
| Before optional class-practical |
| // package javapgms.optionalclass;  public class Driver1 {  public static void main(String[] args) {  String inputString=null;  System.out.println(inputString);  if (inputString!=null) {  System.out.println("length :"+inputString.length());  } else {  System.out.println("null string..!");  }  }  } |
| Null  null string..! |
|  |

|  |
| --- |
| ofNullable():   * it is static method * it will take generic as argument * when you are passing null as argument then it will creat empty optional object * when you are passing argument as non-null value then it will create a object of optional class * whenever optional object we creat then the value or non-null value is wrapped inside a Optional class object * optional[non-null value] * methods of optional class * isPresent() method:help to check value is present or not * it is non-static method * it’s return type is Boolean * it will return true when ther is a value otherwise it will return false when there is no value(in case of empty value) * get() method * it is non-static method and it’s return type is generic * it will help us to get value which wrapped inside a optional class object * when optional is empty then if we try to use get method then we will get a exception NoSuchElementException |

|  |
| --- |
|  |
| // package javapgms.optionalclass;  import java.util.Optional;  public class Driver2AfterOPtional {  public static void main(String[] args) {  String inputString = null;  Optional<String> op1 = Optional.of(inputString);  System.out.println(op1);  }  } |
| Exception in thread "main" java.lang.NullPointerException |

|  |
| --- |
|  |
| // package javapgms.optionalclass;  import java.util.Optional;  public class Driver2AfterOPtional {  public static void main(String[] args) {  String inputString = "demo";  Optional<String> op1 = Optional.of(inputString);  System.out.println(op1);  }  } |
| Optional[demo] |

|  |
| --- |
|  |
| // package javapgms.optionalclass;  import java.util.Optional;  public class Driver2AfterOPtional {  public static void main(String[] args) {  String inputString = null;  Optional<String> op1 = Optional.ofNullable(inputString);  System.out.println(op1);  }  } |
| Optional.empty |

|  |
| --- |
|  |
| // package javapgms.optionalclass;  import java.util.Optional;  public class Driver2AfterOPtional {  public static void main(String[] args) {  String inputString = null;  Optional<String> op1 = Optional.ofNullable(inputString);  System.out.println(op1);  if (op1.isPresent()) {  System.out.println("data is present..!");  }else{  System.out.println("data is not present..!");  }  }  } |
| Optional.empty  data is not present..! |

|  |
| --- |
|  |
| // package javapgms.optionalclass;  import java.util.Optional;  public class Driver2AfterOPtional {  public static void main(String[] args) {  String inputString = null;  Optional<String> op1 = Optional.ofNullable(inputString);  System.out.println(op1);  // if (op1.isPresent()) {  //     System.out.println("data is present..!");  // }else{  //     System.out.println("data is not present..!");  // }  if (op1.get()!=null) {  System.out.println("data is got..!");  }else{  System.out.println("data is not present..!");  }  }  } |
| Optional.empty  Exception in thread "main" java.util.NoSuchElementException: No value present |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***isEmpty() :***   * it is non-static method * it’s return typr is Boolean * introduced in jdk 11 * isEmpty() will return true if the optional is empty elese it will return false * example :  |  | | --- | |  | | Soutln(optinalVariable.isEmpty()); | | true | |  | |

|  |  |
| --- | --- |
| isEmpty() | isPresent() |
| True🡪optinal empty |  |
| False🡪optional non-empty |  |

|  |
| --- |
|  |
| Nillable.isPresent()->s.toString() else “student is not present” |
|  |

|  |
| --- |
|  |
| Main(){  String s=null;  Optional <string> nullable=Optional.ofNullable(S);  If(nullable.isPresent())  s.ToUpperCase();  else  Soutln(s is pointing null“);  } |
|  |

|  |
| --- |
| Supplier:   * It is a predefined functional interface present in java.util.function package * It is having only one abstract method i.e get() method * METHOD DECLARATION: T get(); * It will return/produce the value without taking any input * It opposite to consumer |

|  |
| --- |
|  |
| // package javapgms.java8.Supplier;  import java.util.function.Supplier;  public class Driver {  public static void main(String[] args) {  double pi=Math.PI;  Supplier<Double> suppliervariable=()->Math.PI;  System.out.println(suppliervariable);  System.out.println(suppliervariable.get());  Supplier suppliervariable2=()->Math.max(13,56);  System.out.println(suppliervariable2.get());  // Supplier suppliervariable3=()->Math.max(13,56);  }  } |
| Driver$$Lambda$1/0x0000000100071c28@161cd475  3.141592653589793  56 |

|  |
| --- |
| File handling:   * WHY WE USE JDBC?INSTEAD OF FILE? * File:whenever we want to store any data like character data /object data/image/audio/video from java application file then we go for a concept called file handling concept * We are having the all the classes and the methods in order to communicate with files inside a file handling concept * Example :File,fileWriter,FileReader,FileInputStream,FileOutputStream   File:   * File is a named location where we can store some relative information * It is a class present in java.IO package introduced in jdk 1.8 * Constructors of File class * new File(String name) * new File(String f,String name) * new File(File f,String name) * methods of File class * exists() * this method will check weather file is exist or not * it will return true if the file is exist or else return false[return type is Boolean] * it is non-static method |

|  |
| --- |
|  |
| // package javapgms.filehandling.qsfileHandling;  import java.io.File;  public class Driver1 {  public static void main(String[] args) {  File file1=new File("Demo.txt");  System.out.println("file present :"+file1.exists());  }  } |
| file present :false |

|  |
| --- |
| * createNewFile() * it is a non-static method * it will create a file only if the file is not exist in that particular directory/workspace * if the file is already exists in that particular directory then it will not create file * when it will create a file it will return true else it will return false   whenever we are using this method always it will throws checked exception i.e IoException |

|  |
| --- |
| long length()  boolean mkdir():   * this method will help us to create a folder * it will create a folder only if the folder is not exists in that particular folder/directory else it will no create * whenever it will create a folder it will return true else it will return false   isFile():   * it will check wether the refrence is pointing to file or not * if refrence is pointing to file it will return true otherwise it will returns false |

|  |
| --- |
|  |
| // package javapgms.filehandling.qsfileHandling;  import java.io.File;  import java.io.IOException;  public class Driver1 {  public static void main(String[] args) throws IOException{  File folder1= new File("our folder");  if (folder1.mkdir()) {  System.out.println("folder created successfully..!");  }else{  System.out.println("folder already exists..!");  }  }  } |
| folder created successfully..! |
| folder already exists..! |

|  |
| --- |
| delete():   * it will delete the file if the file exists and it will return true otherwise it will return false |

|  |
| --- |
|  |
| // package javapgms.filehandling.qsfileHandling;  import java.io.File;  import java.io.IOException;  public class Driver1 {  public static void main(String[] args) throws IOException{  File folder1= new File("our folder");  if (folder1.delete()) {  System.out.println("folder deleted successfully..!");  }else{  System.out.println("folder not exists..!");  }  }  } |
| folder deleted successfully..! |
| folder not exists..! |

|  |
| --- |
| **New file(String foldername,String filename ):** |
|  |
| // package javapgms.filehandling.qsfileHandling;  import java.io.File;  import java.io.IOException;  public class Driver2 {  public static void main(String[] args) throws IOException {  File folder2=new File("our folder2");  folder2.mkdir();  File file99=new File("our folder2","file99");  file99.createNewFile();  }  } |

|  |
| --- |
| * File Hirarchy/Io-stream * charecterStream * Reader classes * fileReader class * BufferReader class * Writer classes * FileWriter class * BufferWriter class * PrintWriter class * byteStream * inputStream classes * FileInputStream class * ObjectInputStream * outputStream classes * FileOutputStream * ObjectOutputStream |

|  |
| --- |
| **FileWriter** *class* :   * It is process of storing character data into a file with the help of fileWriter * It is class it is present in java.io package * Constructors of FileWriter class: * new FileWriter(String file) * new FileWriter(File f) * new FileWriter(String filename,boolean append) * new FileWriter(File f,boolean append) * methods of FileWriter : * write(int ch) * write(char ch[]) * write(String s) * flush() * close()   USE OF FLUSH METHOD[ALWAYS RECOMMANDED TO USE FLUSH() METHOD]  METHOD:Write(int ch)   * When we want to store only one character inside a file then we will use this method * In this method we have to pass ASCII value of that character   METHOD:flush()   * By using flush method we are giving guarantee that all the chareter data will be stoered inside file   METHOD:close()   * Once we stored character data into file we have to close the file otherwise there is chance of data loss   Constructor: FileWriter(String filename,boolean append)  When we want to store multiple data again and again without overriding previous data |

|  |
| --- |
| ***FileWriter(String filename,boolean append)***  **FileReader:** |
| // package javapgms.filehandling.qsfileHandling;  import java.io.FileWriter;  import java.io.IOException;  public class Driver3 {  public static void main(String[] args) throws IOException{  FileWriter file22=new FileWriter("file22.txt",true);  file22.write(100);  file22.write("this is the sentese");  char ch[]={'a','e','i'};  file22.write(ch);  file22.flush();  }  } |
| EXECUTED 3 TIMES  dthis is the senteseaeidthis is the senteseaeidthis is the senteseaeidthis is the senteseaei |

|  |
| --- |
|  |

|  |
| --- |
| File Reader[Class] :  it is a class present in java.IO package  FileReader is used to read the character data from the file  Constructors of FileReader [class]:   * new FileReader(String name) * new FileReader(File name)   Methods of FileReader[class]:   * Int read() return type is int * Whenever we are reading character data from the file by using read method ,read method always will return the ASCII value of that particular character * Instead of printing ASCII value if we want to print character then we will do typecasting * Once read method all the data from file then it will return -1 * FileReader will read the character data character by charecter |

|  |
| --- |
| ***Reading data from file using FileReader class’s read() method***  ***Use of two constructors new FileReader(String name)/ FileReader(File name)*** |
| import java.io.File;  import java.io.FileReader;  import java.io.IOException;  public class Driver4FileWriter {      public static void main(String[] args) throws IOException {          FileReader fRObj = new FileReader(".//demo.txt");          for (int i = fRObj.read(); i != -1; i = fRObj.read()) {              System.out.print((char) i);          }          System.out.println();          File fileClassObject = new File(".//demo.txt");          FileReader fRObj2 = new FileReader(fileClassObject);          for (int i = fRObj2.read(); i != -1; i = fRObj2.read()) {              System.out.print((char) i);          }      }  } |
| this is demo file  this is demo file |

|  |
| --- |
| Disadvantages of FileWriter   * The main disadvantage of file writer is whenever we want to write the character in the next line then we have to use \n * This \n is different from one system to another system,it means some OS having \r to move to new line /some having \n to move to next line * So it is not convenient to program * To overcome this advantage will go for BufferedWriter –class * The main disadvantage of FileWriter is it will read file character by chaa=recter(consumes more time) * To overcome this we go for BufferedWriter -class |

|  |
| --- |
| BufferedWriter-class:  It is class present in java.io package used to write character data in file  BufferWriter will not communicate with file directly,it will take the help of Writer object  CONSTRUCTORS: new BufferedWriter(Writer w)  METHODS :   * writer(int ch/char ch/charch[]/string s/) * flush() * close() * void newline() |

|  |
| --- |
| Program-pending |

|  |
| --- |
| BufferedReader-class   * It is class present in java.io package * It is used to read character data from file * BufferedReader can’t communicate with file directly,it will take the help of reader object * By using BufferedReader we can read data from the file character by character Or line by line * CONSTRUCTOR:BuffredReader(Reader r) * Read(),clase(),String readLine() * readLine() method will read charecters line by line inside a file * after reading all character for absence/end of file/ character represented by null value * for reading data it is always recommended to to use BufferedReader()   Disadvantages of BufferdWriter-class:   * Whenever we want to write chrecter data in to new line we have to use use newline() method each and every time * It incresese code length and will affect on code readability * Whenever we want to provide single character we have to use ASCII value of corresponding character.remembering ASCII value for each and every character will be quiet difficult job for programer[THEN WHAT ABOUT {‘A’} THIS ] * To overcome this problem we will go for PrintWriter-class |

|  |
| --- |
| Program |

|  |
| --- |
| PrintWriter-class:   * It is a class present In java.io package used to write character data into file * By using printWriter() we can directly erite primitive value in file * CONSTRUCTORS:   new printWriter(String name/File f/Writer r)   * METHODS : * write(int ch/char ch/[]/String s) | flush() |close() * print(char ch/char ch[]/Boolean b/float f) * println(char ch/char ch[]/Boolean b/float f)   it is always recommended …???????????? |

|  |
| --- |
| program |

|  |
| --- |
| Serialization using |

|  |  |  |
| --- | --- | --- |
| Serialization :   * It is a process of writing state/attributes/?? of object into file * If we want to store any object inside a fuile then that object should be a serializable object otherwise we will get a exception * When we want to store object into file then we take help of FileOutputStream class * FileOutputStream class it will store binary data into a file directly but not object * To store object into file ,FileOutputStream will take the help of ObjectOutputStream class  |  | | --- | | File.txt  Object  Java program | | ***Serialization using fileOutputStream for storing java objects into file*** |   Serializable:   * It is marker interface present in java.io package introduced in ?? * When we are making any object as serializable then that objet(class) must and should implements Serializable interface * If you make the object as serializable then that object will be converted into byteStream later we can store inside a file or database |

|  |
| --- |
| ***Program of serialization*** |
| // package FileIpOpstream;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.FileOutputStream;  import java.io.ObjectOutputStream;  import java.io.Serializable;  public class Driver1StoringObjectinFile  {      public static void main(String[] args) throws Exception{              FileOutputStream f1=new FileOutputStream("fileOutputStream.txt") ;            ObjectOutputStream f2=new ObjectOutputStream(f1);          f2.writeObject(new Student("student1",21));          // FileInputStream fs=new FileInputStream("abc.txt");      }  }  class Student implements Serializable{      String name;      int age;      Student(){      }      public Student(String name, int age) {          this.name = name;          this.age = age;      }      @Override      public String toString() {          return "Student [name=" + name + ", age=" + age + "]";      }    } |
| CONSOLE O/P:  object written successfully..! |
| O/P IN FILE:  ��srStudentĖ� ��;�IageLnametLjava/lang/String;xptstudent1 |

|  |
| --- |
| Deserialization:   * It is process of reading state of an object from the file is known as deseralization * To read the state of an object from the file we are going to take the help of FileInputStream but FileInputStream will read binary data directly but not an object * Then it will take the hel of ObjectInputStream * Inside a ObjectInputStream readObject() method and it’s return type in Object * When we are reading the object always we have to do typecasting(downcasting) |

|  |
| --- |
| ***Program of deserialization*** |
| // package FileIpOpstream;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.FileOutputStream;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  import java.io.Serializable;  public class Driver2IS {      public static void main(String[] args) throws Exception{              FileInputStream f1=new FileInputStream("fileOutputStream.txt") ;            ObjectInputStream f2=new ObjectInputStream(f1);          Student s1=(Student)f2.readObject();          System.out.println(s1);  System.out.println("object read successfully..!");          // FileInputStream fs=new FileInputStream("abc.txt");      }  }  class Student implements Serializable{      String name;      int age;      Student(){      }      public Student(String name, int age) {          this.name = name;          this.age = age;      }      @Override      public String toString() {          return "Student [name=" + name + ", age=" + age + "]";      }    } |
| object read successfully..! |

|  |
| --- |
| **transient –keyword :**   * Transient keyword use * Used for non-static variable only(works only for non-static variable/final non-static in other case there will be no impact) * Sometimes we don’t want to store some state of an object in file instead of store original value we want to store default value then we will make that state/attribute of an object transient * We can use transient keyword only for non-static varable but can not be used for static ,static final and non-static final variable * If we try to use also there is no impact * NOTE : @transient-annotaion use :prevent creation of column in for table in database |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | |
| Tokens in java |
| Give me examples of builtin packages in java(math,lang,time,io,awt,net) |
| About java language |
| Difference between compiler and interpreter?[work,error,code execution,use-case] |
| Explain features of java |
| Code executing software’s |
| Purpose of creating java language |
| Why java is not purely OOP language? |
| Which limitations of c and c++ are overcome by java language?[platform dependency|standerdixed exception handling no inbuilt support of multithreading] |
| Program execution |
| Explain steps to execute java programs |
| Explain compound assignment operator and its return type |
| Explain relational operators and its return type |
| Bitwise operator |
| Explain bitwise operator and its return type |
| Explain increment and decrement operator |
| Explain types of increment decrement operator |
| Explain separator in java |
| Explain dot operator in java |
| Bit Manipulation |
| What is bit manipulation in java? |
| Datatypes and variables |
| Why we use static variables? In which memory they are stored? |
| Why we use non-static variables? In which memory they are stored? |
| Static vs non-static members in java |
| Typecasting in java |
|  |
| What are the types of types of typecasting? |
| What is narrowing? |
| What is widening? |
| Difference between narrowing and widning |
| What is upcasting?why we do upcasting? |
| What is downcasting why we do upcasting |
| Difference between upcasting and downcasting |
|  |
| Control flow statements |
|  |
| Explain decision control statement |
| Explain looping statements |
| Explain if-else statements and it’s variations with workflow |
| Explain switch-case statement and its workflow |
| Explain for loop and its workflow? When to use it? |
| Explain while loop and its workflow? When to use it? |
| Explain do while loop and its workflow? When to use it? |
| Explain for-each loop and its workflow? When to use it? |
| Give difference among all conditional statement |
| Do-while vs while |
| Why we use nested loops? |
| Difference between break and continue |
|  |
| Methods in java |
|  |
|  |
| Why we use methods in java? |
| What is code modularity ?what are i’s benefits? |
| Explain types of methods based on (creator, number of arguments, body) |
| Explain syntax of method |
| Explain return type of method |
| What are modifiers? |
| Access modifiers vs non-access modifier |
| Types of access and non-access modifiers |
| Method signature vs Method prototype |
| Explain binding process of method |
| Explain work of main method |
|  |
|  |
| Difference between return and return type |
|  |
| Program |
| * Program for return type * Compile time polymorphism for methods * Main method overloading * Giving inputs through cmd and scanner * Factorial using recursion * . operator in java |
|  |
| Static vs non static |
|  |
| Tell me about static keyword? |
| Explain accessibility of static java member within and out of class |
| In memory where static members are stored? |
| List members to which we ca prefix static keyword |
| Explain static variables in java |
| Explain scope of static variables in java |
| What are static initializers? why we use them? |
| Explain static blocks? |
|  |
| Explain accessibility of non-static java member within and out of class |
| In memory where non-satatic members are stored? |
| List members to which we can be nonstatic member |
| Explain non-static variables in java |
| Explain scope of non-static variables in java |
| What are non- static initializers ?why we use them? |
| Explain non-static blocks?Why we use it? |
|  |
| OOPs |
| What is oops? why we use Oops? |
| What are objects(simple and technical definition)?How objects are defined in programing? Why we use object? |
|  |
| New keyword and constructors impact in memory? |
|  |
| Can we call constructor anywhere explicitly? If no explain |
| Explain purpose of constructor |
| What is default constructor |
| Content of constructor body |
| Constructor’s type |
| Rules for creation of parameterized constructor |
| Is it compulsory to every class must have constructor? |
| What is default constructor? |
| Purpose of no-argument and default constructor |
| Constructor never be \*\*? |
| Explain constructor overloading |
| Rules of constructor overloading |
|  |
| Purpose of constructor overloading |
|  |
| Note for constructor |
|  |
| What is data hiding ?how to achieve using encapsulation |
|  |
| Explain ‘has a’ relation |
| Explain ‘is a’ relation |
| Explain composition for relation |
| Explain aggregation for relation |
| How to achive realation |
| Explain early vs lazy binding |
|  |
| Effect of inheritance in number of java members |
| Which keywords are used to achieve inheritance? |
| Types of inheritance?levels of inheritance in java |
| Explain single level ,multi-level,hierarchical hybrid inheritance in java |
| Explain multiple inheritance in java? |
| Explain diamond problem in java |
| Which java members are not inerited by child class |
|  |
| Why we use supercall statement and this() call statement and there difference |
| Why we use super statement and this statement and there difference |
| This vs this() |
| Super vs super() |
|  |
| What are the types of polymorphism? |
| Explain is compile time polymorphism? |
| Explain runtime polymorphism |
| Explain types of compile time polymorphisms |
| Explain types of runtime polymorphism |
| Explain method ,operator overloading and method and variable shadowing |
|  |
| Explain method overriding |
|  |
| Which java members ar used to achive abstraction |
| Abstract method vs concrete methods |
| Abstract class and concrete class |
| Why we use abstract methods |
|  |
| Explain all methods in object class? |
| Which method in object class can be implicitly added as abstract method in functional interface?Why others are not allowed |
|  |
| Explain is interface in java in detail? |
| Members allowed in interface |
| Explain how multiple inheritance is achived using interface |
| Difference between abstract class and interface |
|  |
| Exception handling |
| What is exception? |
|  |
| What is exception hierarchy? |
| Exception flow in java language programing |
| Difference between exception and error |
| Difference between checked and checked exception |
| Explain try , catch and finally block |
| Explain proper arrangement for try-catch block |
| Explain difference between throw and throws keyword |
| Difference between final, finally and finalize |
| What is exception prapogation ? |
| When we will get NullPointerException ? |
| When we will get ArrayIndexOutOfBoundException ? |
| When we will get StringIndexOutOfBoundException ? |
| What is exceptionPropogation? |
| ClassCastException |
| NullPointerException |
| All exceptions in java   |  | | --- | | ClassCastException:  trying to access class member which does not exist  trying to doencast object which is not upcasted | | NullPointerException:pointing object which is initialized with null | | IllegalArgumentException:passing illegal value to java member(value i.e not supported by java member) | |  | |
| Wrapper class |
|  |
| Boxing vs unboixing |
| Wrapper classes overrides which 3 methods of object class |
| Value() method of each wrapper class |
| Autoboxing vs Autounboxing |
|  |
| Modifiers accessibility |
| Members prefixed with modifier |
| Final modifier with all members variable,methods and class |
|  |
| Arrays |
| Explain syntax of array |
| Tell me about array class |
| Arrays class in java  Arrays class |
| Programs |
| * Replace repeating number from array * Remove repeating number from array * Find 1st ,2ns,3rd max from array * Find sum of array ele * Find sub of array ele * Find mul of array ele * Find div of array ele |
| Strings,builder and buffer |
| How we create string in java? |
| String class is present u=in which package? |
| String class is P\*,f\* and inherits object class |
| String class implemetns which 3 interface? [CCS] |
| Give me ezamples of string class constructors? |
| Using string class in how many way we can create string object? And what will be impact of each way on memory |
| Explain string constant pool area |
| Explain how strings immutability in java? |
| Explain Why strings are immutable in java? |
| How to create immutable strings in java? |
| Give examples of string inbuilt methods() |
|  |
| String builder and buffer |
| Why we use string builder and buffer |
| Which single method is overridden by string builder and buffer \*toString() |
| How many character string builder and buffer will accommodate |
|  |
| Programs |
| Collections in java |
| 1. what is collection? 2. What is collection wrt java? 3. What is collection framework?why we use it(over array advantages)?It uses non-primitive data and dynamic ds?has hierarchies(Explain) 4. Why we use collection framework instead of array ds? 5. Can we use primitive data in collection?if no why? 6. All collections are dynamic nature??????? 7. Explain two hierarchies in collection framework? 8. Explain collection hierarchy? 9. Explain map hierarchy? 10. Basic difference between collection and map hierarchy? |
| Map interface in collection |
| Set interface in collection |
| |  | | --- | |  | | import java.util.ArrayList;  import java.util.List;  public class Driver2 {  public static void main(String[] args) {  // list declaration :way1  List list1 = new ArrayList();  list1.add(45);  list1.add("njd");  // list declaration :way2  List list2 = new ArrayList<>();  list2.add(4565);  list2.add("njdggd");  // list declaration :way3  ArrayList list3 = new ArrayList<>();  list3.add(4895);  list3.add("njdggd");  ArrayList list4 = new ArrayList();  list4.add(67);  list4.add(77);  list4.add(90);  list4.add(67);  list4.add(48);  list4.add(39);  list4.add(11);  list4.add(0, 99);//  // iterating elemt using for loop  System.out.println();  System.out.println("printing list of element by iterating using for loop");  for (int i = 0; i < list4.size(); i++) {  System.out.print(list4.get(i) + "|");  }  // iterating eleement using forEach loop  System.out.println();  System.out.println("printing list of element by iterating using forEach loop");  for (Object i : list4) {  System.out.print(i + " ");  }  // iterating element using foreach and method refrence  System.out.println("\nway 5:printing list element using refrence variable :");  System.out.println(list4);  System.out.println("printing list of element by iterating using method refrence ");  list4.forEach(System.out::println);  // iterating using iterator and list iterator  // geting element by index  System.out.println("\nget element by index :" + list4.get(5));  // geting list size by index  System.out.println("get list size by refrence variable:" + list4.size());  // checking element is present or not in list  System.out.println("checking element in collection is present or not :" + list4.contains(48));  // add()OM,remove()OM  // setting value  list4.set(0, 100);  System.out.println("updating element of list :");  System.out.println(list4);  list4.forEach(System.out::println);  // remove [index][Integer.valueOf()]  System.out.println("remove element  from list(index) :");  list4.remove(0);  list4.forEach(System.out::println);  System.out.println("remove element  from list(index) :");  list4.remove(Integer.valueOf(90));  list4.forEach(System.out::println);  //creating sub list  //list to array conversion  System.out.println("list to array conversion :");  //sorting array-list  list4.sort(null);  list4.forEach(System.out::println);  System.out.println("desending order:");  list4.sort((o1,o2)->(Integer)o2-(Integer)o1);  list4.forEach(System.out::println);  }  } | | printing list of element by iterating using for loop  99|67|77|90|67|48|39|11|  printing list of element by iterating using forEach loop  99 67 77 90 67 48 39 11  way 5:printing list element using refrence variable :  [99, 67, 77, 90, 67, 48, 39, 11]  printing list of element by iterating using method refrence  99  67  77  90  67  48  39  11  get element by index :48  get list size by refrence variable:8  checking element in collection is present or not :true  updating element of list :  [100, 67, 77, 90, 67, 48, 39, 11]  100  67  77  90  67  48  39  11  remove element from list(index) :  67  77  90  67  48  39  11  remove element from list(index) :  67  77  67  48  39  11  list to array conversion :  11  39  48  67  67  77  desending order:  PS C:\Users\susha\OneDrive\Desktop\gitRepos\javapgms\collection\collectionHirarchy\List> cd "c:\Users\susha\OneDrive\Desktop\gitRepos\javapgms\collection\collectionHirarchy\List\" ; if ($?) { javac Driver2.java } ; if ($?) { java Driver2 }  Note: Driver2.java uses unchecked or unsafe operations.  Note: Recompile with -Xlint:unchecked for details.  printing list of element by iterating using for loop  99|67|77|90|67|48|39|11|  printing list of element by iterating using forEach loop  99 67 77 90 67 48 39 11  way 5:printing list element using refrence variable :  [99, 67, 77, 90, 67, 48, 39, 11]  printing list of element by iterating using method refrence  99  67  77  90  67  48  39  11  get element by index :48  get list size by refrence variable:8  checking element in collection is present or not :true  updating element of list :  [100, 67, 77, 90, 67, 48, 39, 11]  100  67  77  90  67  48  39  11  remove element from list(index) :  67  77  90  67  48  39  11  remove element from list(index) :  67  77  67  48  39  11  list to array conversion :  11  39  48  67  67  77  desending order:  77  67  67  48  39  11 | |  | | |  | | --- | |  | | // package javapgms.oops.comparatorAndComparable;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  public class Driver1 {  public static void main(String[] args) {  ArrayList listOfIntegers=new ArrayList<>();  listOfIntegers.add( 12);  listOfIntegers.add( 24);  listOfIntegers.add( 88);  listOfIntegers.add( 34);  listOfIntegers.add( 26);  listOfIntegers.add( 56);  System.out.println("orginal integer list..!");  System.out.println(listOfIntegers);  //implicit sorting in asencending order by sort() method  System.out.println("implicit sorting in asencending order by sort() method");  Collections.sort(listOfIntegers);  System.out.println(listOfIntegers);  //changing default sorting order by sort() and compare method of comparator functinal interface  System.out.println("changing default sorting order by sort() and compare method of comparator functinal interface");  listOfIntegers.sort((i1,i2)->(int)i2-(int)i1);  System.out.println(listOfIntegers);  }  } | | orginal integer list..!  [12, 24, 88, 34, 26, 56]  implicit sorting in asencending order by sort() method  [12, 24, 26, 34, 56, 88]  changing default sorting order by sort() and compare method of comparator functinal interface  [88, 56, 34, 26, 24, 12] |   ArrayList :  Asynchronised | | |  | | --- | | ArrayList basic operations | | // package linkedlist;  import java.util.LinkedList;  import java.util.List;  public class Driver1 {  public static void main(String[] args) {  LinkedList<Integer> intList1 = new LinkedList<>();  intList1.add(77);  intList1.add(9);  intList1.add(67);  intList1.add(22);  intList1.add(20);  System.out.println("original list :" + intList1);  intList1.addFirst(100);  intList1.addLast(900);  intList1.add(1, 10);  System.out.println("after adding first and last element 2nd element :" + intList1);  System.out.println(" getting 1st element :" + intList1.getFirst());  System.out.println(" getting last element :" + intList1.getLast());  System.out.println(" getting 5th element :" + intList1.get(4));  // removeFirst removeLast() remove(index:X)  intList1.remove(2);  intList1.removeLast();  intList1.removeFirst();  System.out.println("after removing first and last element 2nd element :" + intList1);  System.out.println("removing  even element :");  intList1.removeIf(n -> n % 2 == 0);  System.out.println(intList1);  System.out.println("removing  all element :");  intList1.removeAll(intList1);  System.out.println(intList1);  intList1.add(77);  intList1.add(9);  intList1.add(67);  intList1.add(22);  intList1.add(20);  System.out.println("other list  all element :");  LinkedList<Integer> intList2 = new LinkedList<>();  intList2.add(77);  intList2.add(9);  intList2.add(67);  System.out.println(intList2);  intList1.removeAll(intList2);  System.out.println("removing  commonlist element :");  intList1.removeAll(intList2);  System.out.println(intList1);  }  } | | original list :[77, 9, 67, 22, 20]  after adding first and last element 2nd element :[100, 10, 77, 9, 67, 22, 20, 900]  getting 1st element :100  getting last element :900  getting 5th element :67  after removing first and last element 2nd element :[10, 9, 67, 22, 20]  removing even element :  [9, 67]  removing all element :  []  other list all element :  [77, 9, 67]  removing commonlist element :  [22, 20] | | | |
|  |
| Features introduced in java 1.8 |
| Explain lambda expression |
| What is interface? Why we use functional interface |
|  |
| Why we use static methods in interface? |
| Why we use default methods in interface? |
| Explain Default methods/static methods in interface? |
| When we use default methods and static method? |
| Difference between default and static methods |
| How static and default methods are reducing code duplication |
| What is stream API?Why we use stream API? |
| When we will use Filter () method? |
|  | |
| Static and default methods in interface | |
| When we use default methods and static method? | |
| Difference between default and static methods | |
| How static and default methods are reducing code duplication | |
|  | |
|  | |
| UNCATEGORIZED | |
| what are native methods | |
| comparable vs comparator | |
| ways to implement functional interface | |
| how to resolve problem/error occurred while creating implementation class of interface?{T} | |
| "in lambda expression no need to specify datatype of formal argument" why? | |
| what is difference between abstract class and interface? | |
| *SONARQUBE TOOL:TOOL IS USED TO CHEQUE CODE QUALITY*   * *duplicate code (10 to 70 % code is acceptable)* * *check rules and conventions* * *check test cases(min 80% acceptable)*   *while doing project it is mandatory to use/implement java 8 features to reduce code duplication.* | |
|  | |
| * Difference between early and lazy binding * Difference between equals() method and hashcode() method difference * Difference between Abstract class vs interface * Difference between hashmap and hashtable * Difference between arraylist and linked list | |
| Derived typecasting:  upcasting: downcasting:  def def  implicitly explicitly  class cast exception  super keyword  instanceOf() operator | |
| \*\*\*Multiple inheritance using interface and classes\*\*\*  OOPs  instantiation of object  setting value  updating value  doing above two steps using getters and setters  do above using validation | |
| MULTITHREADING:  JAVA 8 FEATURES:  Why java 8 features are introduced?  Advantages of java 8 features?  Enlist features introduced in java 8?  What is functional interface?  Difference between inbuilt and predefined functional interface?[!]  Enlist examples of inbuilt functional interface?  Ways to implement functional interface  Tell me about lambda expression  explain what is lambda expression (definition features, various cases of compact code)  2 purpose of lambda expression  compact syntax for various scenarios | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| All predicted Questions and points   |  |  | | --- | --- | | Shortcut Keys | Use | | Ctrl+Shift+U / Cmd+Shift+U | Make String Uppercase | | Ctrl+. / Cmd+. | Surround with Try-Catch Block | | Ctrl+Shift+P (Search 'Generate Getters and Setters') | Generate Getters and Setters | | Ctrl+Shift+P (Search 'Generate Constructor') | Generate Constructor | | /\*\* + Enter | Add Official Comment | | Ctrl+Shift+A / Cmd+Shift+A | Add Unofficial Multiline Comment | | Shift+Alt+F / Shift+Option+F | Format Code | | Ctrl+Shift+O / Cmd+Shift+O | Automatic Import and Remove Unnecessary Imports | | F2 | Rename Variable Everywhere | | Type 'void' + Tab | Create a Void Method in Class Block | | Define Return Type or Use Code Snippets | Create a Method in Class Block | | Ctrl+Shift+H | Check Call Hierarchy | | Ctrl+Alt+K (Set), Ctrl+Alt+L (Go to) | Navigate from One Point to Another Point | | F12 / Cmd+Option+Down Arrow | Navigate to Method Implementation | |
| |  |  |  | | --- | --- | --- | |  | Eclips shortcut keys | Vs code | | Make string uppercase | Ctrl+shift+X |  | |  | Ctrl+shift+Y |  | | Surround try-catch/s block/s | Altshift+Z |  | | Generate getters and setters | Alt+shift+S,R |  | | Generate constructor | Alt+shift+O,S |  | | Official comment | Ctrl+shift+j |  | | unOfficial multiline comment | Select regin/lines and press ctr+shift+/  (ctrl+/) |  | | Formatting code | Ctrl+shift+f |  | | Automatic import and remove unnecessary import | Ctrl+shift+o |  | | Rename variable at all place | Ctrl+shit+r |  | |  | Ctrl+shift+t |  | |  | Ctrl+shift+t and then ctrl+o |  | | Used to creat void method In class block | methodName+ctrl+space |  | | Used to creat method in class block | Alt+shift+m |  | | To check call hirarchy | Ctrl+alt+H |  | | One point to othr point | Ctrl+shift+p |  | | Navigate to method name to it’e implemntation | Press control and click on method calll |  | |  | Alt+left/right arrow history navigation |  |  * COMPUTER * SOFTWARE * HARDWARE * what is convention ? why we use convention? * what is compiler? * what is interpreter? * differce between compiler and interpreter (def|execute code...|error|usecase) * JDK|JRE|JVM * what is programing?what is the differnce between coding and programing * levels of programing languages * examples of HLL * why pweple shifted from lll to MLL to HLL? * tell me about java language * Purpose of creating java language * Why java is not purely OOP language? * Which limitations of c and c++ are overcome by java language?[platform dependency|standerdixed exception handling no inbuilt support of multithreading] * History of java language * What is platform? Types of s/w based on platform with example ?eExplain platform independency in java. [!examples] * Define packages in java.Advantags of package.Types of packages * ======================================================================================================================== * Explain steps to execute java programs * *How to execute java program using cmd/terminal?* * EXPLAIN STRUCTURE OF JAVA LANGUAGE * Explain tokens in java[p] * Explain keywords in java * Explain identifiers in java * Explain separators in java[!!!] * Explain operators in java[!!!] * Explain comments in java * Explain literals in java * = vs == * = vs .equals() * Explain bit manipulation * Exolain ternery operator * Explain new operator * Explain dot operator * Explain [] operator * Explain typeOf() operator * Explain :: operator * Explain ->operator * Explain datatypes in java * Explain variables in java * Static vs non-static variables * Global vs local variable * Explain typecasting * Narrowing vs widening * When we get ClassCastException * Upcasting vs downcasting * Explain control flow statement * Explain types of control flow statement * Explain conditional statement * Explain looping statement * Explain if-else statement * Explain switch-case statement * Explain for loop statement * Explain while loop statement * Explain do while loop statement * Explain for -each loop statement * For vs fore-each loop * While vs do-while loop * Method bonding process * What are methods why we use it?what is benefit of code modularity * Explain structure of method * What is method signature * What is method prototype * What is modifier. Access modifier vs non-access modifier * Body content of method * Return statement in method * Return type vs return stat * Method call statement * Formal vs actual argument * Types of methods based on 3 categories * Method overloading vs method overriding * Work of main method * Characteristics of methods * What is recursion why we use recursion * Recursion s looping statements * Explain static members of java * Explain non-static members in java * Loading process of an object * What is oops?why we use oops? * Multiple class file and it’s rules * Explain constructor * What is constructor? * Use of constructor * Tell me about content of constructor body * Give me types of constructors * Rules of constructor * Constructor overloading and it’s rules * Constructor chaining * Constructor recursion/why we use constructor recursion * Can we call constructor anywhere explicitly? If no explain * Explain purpose of constructor * What is default constructor * Rules for creation of parameterized constructor * Is it compulsory to every class must have constructor? * Constructor never be \*\*? * Difference between constructor overloading and chaining * Purpose of constructor overloading and chaining * EXPLAIN METHOD CHAING AND IT’S RULES * note for constructor * oops principal with use * explain encapsulation * programmer should not give which kind of access to user? What is the solution for it? * What is data hiding and it’s advantages * Steps to achieve data hiding * WRT object what is realtion * Types of relation * What is a relation * What is has a relation * ‘Is a’ vs ‘has a’ relation * How to achieve ‘is a relation’ * Define composition and aggregation and difference * Early vs lazy binding * Tight vs loose coupling * explain inheritance in java * what is inheritance * how to achieve inheritance(class and interface) * uses of inheritance * explain types of inheritance * explain diamond problem * Which java members are not inherited by child class * Explain interface in java * Use of interface * Members allowed in interface * Explain functional interface and marker interface * How multiple inheritance is achieved in java * Which java members are not inherited by child class * Why we use super call statement and this() call statement and there difference * Why we use super statement and this statement and there difference * This vs this() * Super vs super() * Explain polymorphism in java * What is polymorphism? * Types of polymorphism * Sub-types of types of polymorphism * Explain compile time polymorphism? * Explain runtime polymorphism * Compile time vs runtime polymorphism * Explain method overriding?Explain rule related to accessibility also(access modifier of overriding and overridden method) * What is upcasing/downcasting?why we use it?dofference * What is abstraction? * what we use to achive abstraction? * Abstract method class?concreate method and class?difference * Rules of abstract class * What is object class?why we use object class * Explain 11 methods of abstract class * In functional interface which abstract methods are allowed?why that methods only allowed and other not * Explain comparable and comparator? Why we use ?diff * What is exception and error ?diff * Types of exception in java * Difference between compile time and runtime exception with examples * What is exception handling * Ways to handle exception in java * Exception handling using try –catch block * Try with multiple catch block * Exception flow and hierarchy * Finally block * Diff:finalize,final and finally * Difference between throew and throws keyword * Exception propogation * What is custom exception ?why we creat custom exception * Difference between throw and throws * Explain below exceptions * NullPointerException * ArrayIndexOutOfBoundException * StringIndexOutOfBoundException * StringIndexOutOfBoundException * exceptionPropogation * ClassCastException * NullPointerException * What are wrpapper class?why we useit? * Boxing and unboxing * Value() method of each wrapper-class * Auto-boxing and auto un-boxing * Wrapper class overrides 3 methods of object class * Which 2 interface are implemented? * From which version boxing and unboxing made implicit? * Define modifiers with there scope * Accessibility of memeres within class and package * Enlist access modifiers and non-access modifiers * Accesiblity in desending order * Explain Final modifier * Tell me about final variable * Tell me about final methods * Explain final class * What is array * Give me syntax explanation * Features of array * About array class * Diff: array vs arrays class * Define strings in java * By default nature of string * How and why strings are mutable in java? * Difference classes to create string * Explain package and prototype of string class * 2 ways to create string using String class. Memory usage wrt way(program to check refrence) * Explain which class is inherited by string class and in that which methods? * Which 3 interfaces are implemented by string class? * Constructor and methods of string class * ``````````````````````````````````````````````````````````````` * Difference string builder and buffer * Which single method is overridden by string builder and buffer \*toString() * How many character string builder and buffer will accommodate * Tell me about collection in java * Is it root interface? * It is part of which framework in core java? * It provides which kind of mechanism * How it better than previous concepts * It supports which kin=d of data? * Benefits over array data structure * Hierarchies in collection framework * What is difference between array and arraylist? * What is difference between list and set? * What is difference between comparable and comparator? * What is difference between vector and arraylist? * What is difference between hashset , linkedhashset and treeset? * What is difference between collection and collections? * Explain java generics * What is difference between linkedlist and arraylist? * interface (set | list | queue) * classes(hashmap | treeset | arraylist | collectons) * iterator vs list interator * What is difference between map and collection? * Runnable interface in java * Multitasking and its type and advantages of type(t) * Applications of multithreading(t) * What is thread ?Difference between thread and process(t) * Ways to create thread ? Which is best way and why?(t) * Methods of thread class and constructor with use explanation(t+p) * Difference between start() and run() method(t) * Uncertainty in o/p with explanation(t) * About ‘thread scheduler’(t) * Explain threads are independent .Write code to prove(t+p) * Why we prevent threads ? ways to prevent threads?(t+p) * Difference between yield () ,join and sleep() method(t+p) * Why we interrupt method? How to interrupt method?(t+p) * Explain what is ‘data inconsistency problem’ ? How it is resolved ?(t+p) * Explain what is Synchronization?(t+p) * Tell me about ‘Synchronized’ keyword(t+p) * Give difference between Synchronized block and method(t+p) * Explain lock concept?(t) * Give difference between object level lock and class level lock(t) * Tell me about ‘ interthread communication’(t+p) * Tell me about daemon thread(t+p) * Give difference between object level lock and class level lock(t) * Tell me about ‘ interthread communication’(t+p) * Tell me about deadlock(t) * Tell me about daemon thread(t+p) * Method refrence in java * Static methods in interface * Default methods in interface * why java 1.8 intoduced * why we are learning java 1.8?why it is introduced ? * features introduced in java 1.8: * Functional interface in java * Note on lambda expression * How code is reduced in java using lambda expression * Static vs default methods * When to go for lambda expression and method reference? * When to go for constructor reference? * 3 Types of method reference * What are predefined functional interface ?why we use them?what are the benefits of using predefined function? * When to go for predicate predefined functional interface?tell me about predefined functional interface |