Runnable :

* Comparable and Comparator are predefined functional interface used for used for sorting of custom object
* functional interface having only one method i.e run() method
* prototype
* present in java.lang package
* introduced in jdk 1.0

===================================================================================================================================

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.

===================================================================================================================================

*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.*

===================================================================================================================================

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>*

===================================================================================================================================

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) |  |  |

===================================================================================================================================

|  |
| --- |
|  |
| 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..! |

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`

|  |
| --- |
| ***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();  // } |  |  | |
|  |

|  |
| --- |
| ***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 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 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

|  |
| --- |
| **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..! |

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 reefrence |

|  |
| --- |
| **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:

* 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..! |

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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

===================================================================================================================================

**Constructor reference :**

Questions and notes:

|  |
| --- |
| Technical questions |
|  |
| |  | | --- | | **Basic terminologies in java** | | What is computer?Why we use?What are its componenets | | What is ‘software” ? | | What is ‘hardware’? | |
|  |
| |  | | --- | | **Conventions in java** | | Explain conventions in java | | Pascal case vs camal case | |
|  |
| |  | | --- | | **Programing in java** | | What is ‘programing’? | | Difference between ‘coding’ and ‘programing’ | | What is ‘programing language’ | | Explain levels of programing languages | |  | |
|  |
|  |
| |  | | --- | | **Code executing software’s** | | What is compiler? | | What is interpreter? | | Difference between compiler and interpreter?[work,error,code execution,use-case] | | Give the difference between JDK,JRE and JVM. | | Explain JDK architecture | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| |  | | --- | | **About java** | | Tell me about java | | Explain features of java | | Give history of java | | Purpose of creating java language | | Which limitations of cand c++ are overcome by java language?[platform dependency|standerdixed exception handling no inbuilt support of multithreading] | | Why java is not purely OOP language? | |
|  |
|  |
|  |
|  |
| |  | | --- | | **Platform dependency** | | Explain term ‘platform’ | | Explain platform based types of software | | ***Explain platform dependency in java ?*** | | ***Why s/w made using c and c++ are platform dependent?*** | | ***Why s/w made using java are platform dependent?*** | |  | |
|  |
|  |
| |  | | --- | | **Program execution** | | Explain steps to execute java programs | | *How to execute java program using cmd/terminal?* | | Explain structure of java program? | | Packages in java? Types of packages?advantages of packages | | Give me examples of builtin packages in java(math,lang,time,io,awt,net) | |
|  |
|  |
|  |
| |  | | --- | | **Tokens in java** | | Explain tokens in java | | Explain types of tokens in java | | Explain ’Literals ’ in java | | Explain keywords in java | | Explain identifiers in java | | Give rules of naming identifiers | | Explain comments in java | | Explain types of comment | | Explain operators in java | | Explain Arithmetic operator and its return type | | When we get arithmeticException(2 examples of arithmetic operations) | | Explain assignment operator and its return type | | = vs == [initialization/comparision,assign value/return Boolean value,operator type] | | = operator vs equals() method | | Explain compound assignment operator and its return type | | Explain relational operators and its return type | | Explain bitwise operator and its return type | | Explain increment and decrement operator | | Explain types of increment decrement operator | | Explain relational operator and its return type | | Explain logical operators and its return type | | Explain separator in java | | Explain dot operator in java | | Explain typeOf() operator | |  | |  | |  | |
|  |
|  |
| |  | | --- | | **Bit Manipulation** | | What is bit manipulation in java? | |  | |  | |
|  |
|  |
| |  | | --- | | **Datatypes and variables** | | What is datatype? | | Explain different datatypes used in java? | | What is variable? | | Explain types of variables in java | | What are local variables and rules of declaring variable in java ? | | What are global 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 variables | | Static vs non-static members in java | | Give the rules of creating variables in java | |
|  |
|  |
| |  | | --- | | **Explain typecasting in java** | | What is type casting? | | 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** | | What are control-flow statement? | | 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 | | **PROGRAMES**  **-try usin all loops**  **-try for each loop and for loop** | | * 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 | |
| |  | | --- | | **Methods in java** | | Explain 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 | | What is recursion? Why we use recursion? | | Explain return statement in java | | 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** | | What are static members?Use of static members | | Tell me about static keyword? | | Explain accessibility of static java member within and out of class | | In memory where ststic 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? | |  | | What are non static members?Use of static members | | 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? | | Explain object loading process | |
|  |
|  |
|  |
|  |
| |  | | --- | | **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 | |  | |
|  |
|  |
| |  | | --- | | **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? | | Difference between class and object | | How to create object in java? | | New keyword and constructors impact in memeory? | | What is new?why we use it? | |  | |  | |  | | Explain class | | Explain types inner class and it’s type | | Explain class loading process | | Explain multiple class file in java and rules associated with it?Explain public class in multiple class file | |  | | What is constructor? | | Can we call constructor anywhrer explicitly?if no explain | | Explain purpose of constructor | | What is default constructor | | Content of constructor body | | Constructor’s type | | Rules for creation of parametorized constructor | | Is it compulsory to every class must have constructor? | | What is default constructor? | | Purpose of no-arument and default constructor | | Constructor never be \*\*? | | Explain constructor overloading | | Rules of constructor overloading | | Purpose of constructor overloading | | Explain method chaining and it’s rules | | Note for constructor | |  | | Why we use OOPs principal | | What is encapsulation? | | What is data hiding ?how to achieve using encapsulation | | Realtions in oops | | 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 | |  | | What is inheritance? | | 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 is polymorphism? | | 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 derived typecasting | | Explain method overriding | | Explain what is abstraction therotivally and technically | | Which java members ar used to achive abstraction | | Abstract method vs concrete methods | | Abstract class and concrete class | | Why we use abstract methods | |  | | What is object class ? why we use it? | | 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 handling? | | 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 | |
|  |
|  |
|  |
| |  | | --- | | **Wrapper class** | | What are wrapper class ? why we use it ? | | Boxing vs unboixing | | Wrapper classes overrides which 3 methods of object class | | Value() method of each wrapper class | | Autoboxing vs Autounboxing | | Modifiers and it’s type | | Modifiers accesiblity | | Members prefixed with modifier | | Final modifier with all members variable,methods and class | |  | |  | |  | |  | |  | |  | |  | |  | |
|  |
|  |
|  |
|  |
| |  | | --- | | **Arrays** | | What is array why we use it?featues an of array | | Explain syntax of array | | Tell me about array 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** | | What is string ? | | How we dreate 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 accomodate | | Give me difference between string builder and buffer | |  | |  | | **Programs** | |  | |
|  |
|  |
| |  | | --- | | **Collections in java** | | 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? | | What is difference between linkedlist and arraylist? | | What is difference between map and collection? | | Write code to check given number is palindrome or not? | |
|  |
|  |
| |  | | --- | | **JAVA 1.8 FEATURES** | |  | |  | |  | |  | |  | | Why java 1.8 introduced | | 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 method refrence?How method reference reducing code duplication | | What will happens when number arguments are different? | | Difference static method and non-static mrthod refrence | |
|  |
|  |
|  |
|  |
|  |
| |  | | --- | | **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? | |  | |  | |  | |  | |
|  |
|  |
|  |
| |  | | --- | | **Keywords ,excepton,packages,errors,modifiers** | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| Technical questions |
|  |
| |  | | --- | | **Basic terminologies in java** | | What is computer?Why we use?What are its componenets   |  | | --- | | WHAT IS 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 | | | What is ‘software” ?   |  | | --- | | WHAT IS SOFTWARE?   * Set of programs which works in co-ordination with each other to perform a specific task In order to solve real world problem are together known as software * E.g:power point presentation,excel,Pubg etc | | | What is ‘hardware’?   |  | | --- | | WHAT ARE HARDWARE ?   * Set of electronic components on which applications runs are known as hardware. E.g::CPU,GPU,SSD,HDD | | |
|  |
| |  | | --- | | **Conventions in java** | | Explain conventions in java | | Pascal case vs camal case | |
|  |
| |  | | --- | | **Programing in java** | | What is ‘programing’? | | Difference between ‘coding’ and ‘programing’ | | What is ‘programing language’   |  | | --- | | 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 | | | Explain levels of programing languages   |  | | --- | | 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 understandable and instructible by programmer to computer is known as High Level Language.  E.g: c++, Java, python, Perl, Scala, ruby, php, kotlin, c# | | |  | |
|  |
|  |
| |  | | --- | | **Code executing software’s** | | What is compiler? | | What is interpreter? | | Difference between compiler and interpreter?[work,error,code execution,use-case] | |  | | * Explain JDK architecture * Give the difference between JDK,JRE and JVM.  |  | | --- | | * 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** | | Tell me about java   |  | | --- | | 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.   * Java is popular for its features like platform independence (thanks to the Java Virtual Machine) and built-in support for multithreading. * It is a statically typed and strongly coupled language. * Its standardized support for exception handling makes it a robust language. | | | Explain features of java | | Give history of java   |  | | --- | | 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 | | | Purpose of creating java language | | Which limitations of c and c++ are overcome by java language?[platform dependency|standerdixed exception handling no inbuilt support of multithreading] | | Why java is not purely OOP language? | |
|  |
|  |
|  |
|  |
| |  | | --- | | **Platform dependency** | | * Explain term ‘platform’ * Explain platform based types of software  |  | | --- | | Platform:.   * The combination of s/w and h/w component on which computer software runs is known as platfor * 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 | | |  | | ***Explain platform dependency in java ?*** | | ***Why s/w made using c and c++ are platform dependent?***   |  | | --- | | * 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 * 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 | | | ***Why s/w made using java are platform dependent?*** | |  | |
|  |
|  |
| |  | | --- | | **Program execution** | | Explain steps to execute java programs | | *How to execute java program using cmd/terminal?* | | Explain structure of java program?   |  | | --- | | 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? Types of packages?advantages of packages | | Give me examples of builtin packages in java(math,lang,time,io,awt,net) | |
|  |
|  |
|  |
| |  | | --- | | **Tokens in java** | | Explain tokens in java | | Explain types of tokens in java | | Explain ’Literals ’ in java   |  | | --- | | 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 * c)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   |  | | --- | | 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 * java language has 51+ keywords (!!!)   Eg::static,public,private,int,long,short,byte,float,for ,continue,break | | | Explain identifiers in java   |  | | --- | | IDENTIFIERS:   * One of the token of java language. * Identifiers in java are nothing but names given to java members such as method,variable,interface and class * we have certain rules to for creating Identifiers and that are * 1)Identifiers should not start with digits * 2)Identifiers should not consist of special characters except \_ and $ * 3)it should not consist of blank spaces * 4)keywords can't be used as an identifier * we follow pascals case convention for class and interface Identifiers while for variable and method identifier we follow camalcase convention | | | Give rules of naming identifiers | | Explain comments in java | | Explain types of comment   |  | | --- | | COMMENTS:   * It is the information or note written by programmer related to program in order to understand program in better way. * Comments are ignored during execution * there are 2 types of comments * single line comment * multi line comments * official * unofficial????????? | | | Explain seperators in java   |  | | --- | | SEPARATORS IN JAVA:   * Special characters used to define and organize and puncuate java program. * They are used to , * separate block of related statement * defines the scope of variable * separate multiple declared variable in same line - * separate java members * E.g.: * {} * Will represent block of related statement * And the scope of variable declared inside block is limited to that block inly * E.g.: braces, parenthesis, round bracket, square bracket, comma, semicolon, | | | Explain operators in java   |  | | --- | | OPERATORS: [MAACCBIRL]   * These are special symbols which perform operations on operand (values) to get final result. * The final result is dependent on type of operand used * For example of arithmetic operators will result integer or float value * Use of logical and comparison operator will result in Boolean value * We have various type of operators in java * Operator types based on number of operands operator takes to perform operation * Unary * Binary * Ternary * Operation based operator(use of each operator mentioned in square bracket) * Arithmetic operator * Assignment operator * Compound assignment operator * Logical operator * relational operator * conditional operator * Bitwise operator * increment /decrement operator * miscellaneous operator | | | Explain operand based operators   |  | | --- | | OPERAND BASED OPERATOR   * Unary : operator perform operation on single operand * Binary : operator perform operation on two operands * Ternary: operator perform operation 3 operands | | | Explain Arithmetic operator and its return type   |  | | --- | | ARITHMETIC OPERATOR:   * binary operator used to perform mathematical operations like * [+,-,\*,/,%] | | | When we get ArithmeticException (2 examples of arithmetic operations) | | What is compound assignment operator?   |  | | --- | | COMPOUND ASSIGNMENT OPERATOR: | | | Explain assignment operator and its return type   |  | | --- | | ASSIGNMENT OPERATOR :   * binary operator used to assign/store value to a variable * LHS ,RHs must be variable and value respectively * value type and variable type must be same . * if types are not same either winding happens or will throw compile time error | | | = vs == [initialization/comparision,assign value/return Boolean value,operator type]   |  | | --- | | 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 | | | = operator vs equals() method   |  | | --- | | 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 | | | Explain compound assignment operator and its return type | | Conditional operator   |  | | --- | | CONDITIONAL OPERATOR   * ternary operator used to for decision making * Syntax()?(operand 1):(Operand 2); (with explanation) * result type depends on operand 2 and operand 3 | | | 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 | | Miscillanious operator   |  | | --- | | [dot operator,instanceOf(),new operator] | | | Explain relational operator and its return 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 | | | Explain logical operators and its return type   |  | | --- | | LOGICAL OPERATOR:   * used for logical design making. * used for decision making based on multiple conditions * it may be unary and binary operator | | | Explain separator in java | | Explain dot operator in java | | Explain typeOf() operator   |  | | --- | | instanceOf() :   * to check object reference belong to hierarchy of given object type[??????????] | | | Explain Unary operator   |  | | --- | | UNARY OPERATOR:   * unary operator used to update variable value by one : * 4 types unary operator * pre increment operator : increases value after use * post increment operator: increases value before use * pre decrement operator :decreases value after use * post decrement operator: decreases value before use * miscellaneous operator | | |  | |  | |
|  |
|  |
| |  | | --- | | **Bit Manipulation** | | What is bit manipulation in java? | |  | |  | |
|  |
|  |
| |  | | --- | | **Datatypes and variables** | | What is datatype?   |  | | --- | | 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. | | | Explain different datatypes used in java? | | What is variable? | | Explain types of variables in java | | What are local variables and rules of declaring variable in java ? | | What are global variables? | | Difference local and global 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 * it is not compulsory as they are initialized with default values implicitly if programmer do not to initialize them. | | | 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 variables   |  | | --- | | 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 | | | Static vs non-static members in java | | Give the rules of creating variables in java | |
|  |
|  |
| |  | | --- | | **Explain typecasting in java** | | What is type casting? | | 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** | | What are control-flow statement? | | 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 | | **PROGRAMES**  **-try usin all loops**  **-try for each loop and for loop** | | * 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 | |
| |  | | --- | | **Methods in java** | | Explain 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 | | What is recursion? Why we use recursion? | | Explain return statement in java | | Difference between return and return type | | Difference between formal and actual argument   |  | | --- | | Difference between actual vs formal arguments:  A]formal arguments: variable declared in method signature  B]actual arguments: are variable passed to method call statement | | |  | | 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** | | What are static members?Use of static members | | Tell me about static keyword? | | Explain accessibility of static java member within and out of class | | In memory where ststic 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? | |  | | What are non static members?Use of static members | | 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? | | Explain object loading process | |
|  |
|  |
|  |
|  |
| |  | | --- | | **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 | |  | |
|  |
|  |
| |  | | --- | | **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? | | Difference between class and object | | How to create object in java? | | New keyword and constructors impact in memory? | | What is new? Why we use it? | |  | |  | |  | | Explain class | | Explain types inner class and it’s type | | Explain class loading process | | Explain multiple class file in java and rules associated with it? Explain public class in multiple class file | |  | | What is constructor? | | 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 | | Constructor 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 | | | Purpose of constructor overloading | | Explain method chaining and it’s rules | | Note for constructor | |  | | Why we use OOPs principal | | What is encapsulation? | | What is data hiding ?how to achieve using encapsulation | | Realtions in oops | | 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 | |  | | What is inheritance? | | 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 is polymorphism? | | 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 derived typecasting | | Explain method overriding | | Explain what is abstraction therotivally and technically | | Which java members ar used to achive abstraction | | Abstract method vs concrete methods | | Abstract class and concrete class | | Why we use abstract methods | |  | | What is object class ? why we use it? | | 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 handling? | | 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 | |
|  |
|  |
|  |
| |  | | --- | | **Wrapper class** | | What are wrapper class ? why we use it ? | | Boxing vs unboixing | | Wrapper classes overrides which 3 methods of object class | | Value() method of each wrapper class | | Autoboxing vs Autounboxing | | Modifiers and it’s type | | Modifiers accesiblity | | Members prefixed with modifier | | Final modifier with all members variable,methods and class | |  | |  | |  | |  | |  | |  | |  | |  | |
|  |
|  |
|  |
|  |
| |  | | --- | | **Arrays** | | What is array why we use it?featues an of array | | Explain syntax of array | | Tell me about array 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** | | What is string ? | | 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 accomodate | |  | | Give me difference between string builder and buffer   |  | | --- | | Difference between Buffer and builder   * A]synchronized/ Asynchronized * B]slower/faste * C]consumes memory-less/more | | |  | |  | | **Programs** | | |  | | --- | | 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 | | |
|  |
|  |
| |  | | --- | | **Collections in java** | | 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? | | What is difference between linkedlist and arraylist? | | What is difference between map and collection? | | Write code to check given number is palindrome or not? | |
|  |
|  |
| |  | | --- | | **JAVA 1.8 FEATURES** | |  | |  | |  | |  | |  | | Why java 1.8 introduced | | 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 method refrence?How method reference reducing code duplication | | What will happens when number arguments are different? | | Difference static method and non-static mrthod refrence | |
|  |
|  |
|  |
|  |
|  |
| |  | | --- | | **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? | |  | |  | |  | |  | |
|  |
|  |
|  |
| |  | | --- | | **Keywords ,excepton,packages,errors,modifiers** | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |