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..! |

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 | |
|  |
|  |
|  |
|  |
|  |
| |  | | --- | | **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** | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |