**Core Java- 19/5(Introduction, Features, JDK/JRE )**

**Features**

**• Simple**

**• Platform Independent**

**• High Performance**

**• Java is 100% object-oriented. Everything is inside the classes in Java. You cannnot write main outside the classes.**

**• Java is a compiled and interpreted language.**

**• JIT (Just in Time) Compiler:-**

**o Works just before execution.**

**o It converts some part of the code directly into machine language.**

**• Because of this, the language becomes faster and it reduces the read and convert time.**

**• Security over the internet**

**• Robust (strong/tough)**

**o Automatic Memory Management**

**o Exception Handling**

**• Multithreading (multi-tasking)**

**• Distributed (Remote Server involved)**

**• Dynamic**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**JDK/JRE**

**First Program**

**public class Hello**

**{**

**public static void main(String[] args)**

**{**

**System.out.println("Hello world");**

**}**

**}**

**• When the class is declared as public, the name of the program should be same as the class name.**

**• Static method can be called without object creation thats why it is used in main.**

**• out is the object called for print, it is accessible using System class.**

**• println() is a method which terminates the current line by writing the line separator string.**

**• At least one class should be public to declare main().**

**• Class name always start with a capital letter.**

**• System = Class, out = object, println() = function.**

**• Setting path of java/javac (for setting the path of java and javac you need following steps).**

**• For running program javac programName.**

**• javac -version is used to check the version of java.**

**• \*Bytes code(.class) is always based on class name, if program name and class name are not same then .class file name will be class name not a program name. \***

**javac programName**

**java .classFileName**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**20/5 (OOPs,Major and Minor pillar)**

**Introduction of Object Oriented Programming**

**Procedural Programming**

**\* It is made with the functions.**

**\* Co-ordinate with `main()`.**

**\* Does not work with entities.**

**\* Real-life modelling is not possible.**

**\* Data Security (mainly focuses on procedures not security)**

**\* After some code modification of code is difficult.**

**• To overcome this disadvantage, a new concept i.e OOP was formulated**

**Object Oriented Programming**

**Object**

**• It is an instance of a class.**

**• It represents real life entity. For e.g. - customer, student, account.**

**• Object has some state and behavior.**

**• Every object has some identity and role to be played in system.**

**• Some objects are actually physically present, they are called as tangible entity/object (e.g. student, laptop,hard-disk).**

**• Some objects are not actually physically present, they are called as non-tangible entity/object (e.g. data, account,software)**

**• Object is something which represents the real life tangible and non-tangible entity which have some identity and role to be played.**

**• State and behavior of objects**

**o Set of values given to the common attributes which are used for describing the object.**

**o State is a description of the object.**

**o E.g. - There is a car which has some company name, color, number..etc**

**o So if the object is MyCar then the states of the object are Maruti,Black,mh14p8254..etc**

**o State is the description of object.**

**o There is some functionality/behavior/role of object like car speed in program it is like function.**

**o State = Set of values**

**o Identity is one of the state value which has some unique identity.**

**Four Major Piller of OOP**

**Major Pillar**

**1. Abstraction**

**2. Encapsulation**

**3. Inheritance**

**4. Polymorphism**

**Minor Pillar**

**1. Strong typing**

**2. Concurrency**

**3. Persistance**

**Minor Pillar**

**1) Strong Typing**

**• Particular type of input for the way of typing or writing code.**

**2) Concurrency**

**• Simultaneous Execution eg. Notepad spelling check**

**3) Persistance**

**• Permanent Storage/Database**

**Major Pillar**

**1) Abstraction**

**• Selective ignorance**

**• It helps in reducing the complexity of code.**

**• Process of separation which takes what is needed.**

**•It focuses on what is needed/required and discards everything else.**

**2) Encapsulation**

**State(data) and Behavior(function) are inseparable thats why we use Encapsulation**

**State and behavior will always together bundled into one unit is called Encapsulation.**

**Advantage-**

**1.Data hiding/helps to restrict the access of data**

**2.Data Binding**

**3.Complex functionality can be hidden so that simple UI is used for using of object.**

**4.Hide Complexity and give simple functions to the user**

**• Class is the best example of encapsulation.**

**3) Inheritance**

**• It is about the "is a" relationship.**

**• If the entity has a "is a" relationship then there is Inheritance.**

**• Advantage 1.Reusability/reusing of code.**

**• 2.It creates the basics of polymorphism**

**• Specialize entity from existing generalize entity by adding some extra state and behavior.**

**4) Polymorphism**

**• Polymorphism:- Same massage given to common entity of different object.(different function defication)**

**• Encapsulation Unit**

**class Date**

**{**

**// state and beahvior**

**// data members(private) + member methods(public)**

**private int dd;**

**private int mm;**

**private int yy; //state members**

**public void showDate()**

**{**

**System.out.println(dd+"/"+mm+"/"+yy);**

**}**

**{**

**public class DateDemo**

**{**

**public static void main(String args[])**

**{**

**Date d1; //reference variable decleration**

**d1 = new Date(); //object creation**

**d1.shoowdate();**

**}**

**}**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**21/5 (Data Types, Memory Management, Constructor, this)**

**Data Types**

**1. Value type (premative) int,char,short,double,float eg. char ch;**

**2. Reference type (reference/non-premative(in c/c++)) class,array,interface eg. Date d1;**

**Memory Management in java**

**• main method variable is in stack.**

**• Object are stored in heap.**

**• new is used to create to object in heap**

**• Heap data members(data in heap/object) are always initialized to 0.**

**• Default value of object variable is 0.**

**• Garbage collector is actually removing the unreferenced objects from the heap.**

**• In Java, Switch case works for both int and string.**

**• One advanced for loop(each for) is added in java.**

**• Class(acts as encapsulation unit) never occupies any memory.**

**Constructor**

**• Is the function that has same name as class.**

**• If you do not write constructor, then there is default constractor(zero argument).**

**• It has no any return type.**

**• There is two type of constructor 1. parameterized 2.non-parameterized.**

**• If you do not write any constructor, then the default will be there but if the you pass with argumrent you have to write the constructor.**

**• If you write paramatrized constructor, then the default is not available.**

**• We can write N number of constructors but the parameters should be different. No constructor :- Default constructor call Constructor with parameter :- have to define parametrixed constructor Both (para/non para) :- Both are available but parameterized constructor have to be defined (eg. con(), con(int a, int b))**

**• Constructor follows all the rules of polymorphism.**

**• Constructor assigns the local variable value to instance variable.**

**• Which constructor will be used depends on creation of object.**

**• If you create a object of parameterized constructor, then only parameterized constructor will call**

**this**

**• this refers to the current object(for which constructor is called)**

**• Refers to the last point of constructor.**

**• instance variable = local variable**

**• this.dd = dd;**

**• this.mm = mm;**

**• this.yyyy = yyyy;**

**• We are assigning the local variable to the instance variable.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**22/5 (Garbage Collector, Array)**

**revise**

**Garbage Collector**

**• Static**

**• Static can access only static data**

**• Non static access both static and non static**

**static block,static method,static variable**

**Static , Array covered portion**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**24/05 (Association Relation, Wrapper Classes)**

**reference equality**

**Association Relationship/has a relationship**

**• General rule of Encaplusation is declare Data Member as private and method public.**

**class Person**

**{**

**private String name; //instance/non static-has**

**private Date bdate; //reference variable-has**

**private int age; // value type variable**

**//person has name**

**//person has birthday**

**public Person(String name, int dd,int yy,int mm )**

**{**

**this.name=name;**

**this.bdate=new Date(dd,yy,mm);**

**}**

**}**

**class Date**

**{**

**int dd,yy,mm;**

**public Date(int dd,int yy, int mm)**

**{**

**this.dd=dd;**

**this.mm=mm;**

**this.yy=yy**

**}**

**}**

**class PersonMain()**

**{**

**public static void main(String args [])**

**{**

**Person p1 = new Person("Shree",07,97,08);**

**}**

**}**

**• When there is one object inside/contain other object, it is called has relationship.**

**• Object within object.**

**• Instance of one class contain instance of another class is called has a relationship.**

**• Reusing the class**

**• If you do not assign any value to reference variable it will treat as null.**

**• name and bdate will be null if you dont assign anything to it.**

**• In Value type variable if you do not assign anything then it'll be zero.**

**• int age will be zero if you do not assign anything to it.**

**• You cannot call method on null reference you will get error called null pointer expection.**

**• Person is container and and bdate is component.**

**• Adventage:- reusability.**

**Object Array**

**public class ObjectArray**

**{**

**Date [] datearr;**

**datearr = new Date[3];**

**datearr[0]=new Date();**

**datearr[1]=new Date(11,12,99);**

**datearr[2]=new Date(12);**

**// Short cut**

**//Date[] datearr = {new Date(),new Date(11,12,99),new Date(22)};**

**for(int i=0;i<datearr.lenght;i++)**

**{**

**datearr[i].showdate();**

**}**

**}**

**Wrapper Class**

**• Is the classes which is replaced to treditional variables for using 100% oops concepts.**

**• Example:-**

**int i=10;**

**Integer obj = new Integer(23);**

**Integer obj = i; //autoboxing feature-no need to write new**

**// obj is object**

**//i is value**

**int num = obj; //auto unboxing**

**• It assigns an object to integer(auto unboxing) and an integer to object(auto boxing)**

**• Because of wrapper class java is pure object oriented.**

**• There are 8 premative data type so there are 8 wrapper classes.**

**• Specifically Wrapper classes used in collections.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**25/05 (Packages,IDE)**

**Packages**

**• Packages is a namespace/collection of classes interfaces and sub-classes.**

**• Conceptually, it is a File Manager/File system in our mobile and PC.**

**• Easy to manage the things.**

**• File with same name but in different folder (avoid name conflicts).**

**• Example of build in package java.lang is base and java.io,java.util,java.sql are sub packages.**

**• You can create your own package.**

**• Package name should always start with small letter.**

**• To start a package, you have to just put package package\_name;.**

**• Always start your program with package include.**

**• Sub packages are imported as package package\_name.sub\_package;**

**• When package is complied packages are reflected in folder hierarchy.**

**• Helps in reusability.**

**• Build in package java.lang API.**

**• Package name should be short and small letter.**

**• Command for package compile javac -d dirpath javafilename**

**Eclipse IDE**

**• It gives Development support.**

**• Eclipse is developed in JAVA.**

**• Plugin is available for extra features.**

**• Workspace is a current working directory that has to be selected when eclipse is started.**

**• Perspective is combinition of views and editors that are required for a particular type of application.**

**• Use Ctrl + space for auto complete.**

**• In project, there is option Build Automatically which compiles the code while you type it.**

**• Right click on code then source and then generate const from superclass for creating constructor/with field (parameterized) autumatically.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**26/05(Inheritance,super(),Overriding, Dynamic Binding, Debug, Scanner)**

**Inheritance**

**• It has a is a relationship.**

**• Creating specialized entity from existing generalized entity by adding extra state and behavior.**

**• Person - name and bdate is state - display()**

**• Emp - empid, salary bev - calSalary()**

**• extends is keyword used for inheritance.**

**class Emp extends Person**

**{**

**//person is parent/super class**

**//emp is child/sub class**

**}**

**• It has a is a relationship with each others.**

**• It is used for reusability.**

**• Child class recieve data members and methods.**

**• Private members are restricted within the class so not accessible to the child class.**

**• Super() is used for accessing super class constructor.**

**• We can access any constructor with the help of super()**

**• super() should always be called first in subclass.**

**• super() is a representation of the default constructor of the super class.**

**• So if you do not write it, it will the call default constructor of super class implicitly.**

**• Two methods of same name can be same in super and sub class because they are not in same class this is called method overriding.**

**// Method overriding**

**class Person**

**{**

**display();**

**}**

**class Emp extends Person**

**{**

**int id,salary;**

**public void display()**

**{**

**super.display();**

**syso("ID:-"+id);**

**syso("Salary:-"+salary)**

**}**

**}**

**• Recrusion is calling the ame method from same method.**

**// Recursion**

**class Emp extends Person**

**{**

**public void display()**

**{**

**display();**

**}**

**}**

**• The difference between Inheritance and the import is that in import you cannot access data members of class but in inheritance you can access the data members of super class.**

**debug,scanner**

**Scanner**

**Dynamic Binding**

**• Whenever you create reference of super class you can create object of subclass Person p = new Emp();**

**• If we call p.display();, which display method will be called, to understand this we need to understand static binding and dynamic binding**

**static binding (Not used in JAVA)**

**• Binding takes place at compile time,**

**• It is based on the type of invoking entity.**

**• It will resolve on type of p/Person.**

**Dynmaic binding (used in java)**

**• Binding takes place at run time.**

**• It resolves type of object refered by invoking entity i.e object of p/Employee.**

**• All instance method calls are resolved dynamically so the display() called from Employee.**

**// if you create reference of super class i.e Person p; (Person is super class)**

**// You can create object of subclass i.e p=new Employee(); (Employee is subclass)**

**// You cannot do Employee e = new Person();**

**class Person**

**{**

**display()**

**{**

**System.out.println("Person display");**

**}**

**}**

**class Employee extends Person**

**{**

**dislay()**

**{**

**System.out.println("Employee display");**

**}**

**}**

**pblic class MainClass**

**{**

**public static void main(String args)**

**{**

**//Now see the obj and reference**

**Person p = new Employee();**

**p.display();**

**// so the question is which display will call Employee-> display() or Person-> display();**

**//java used Dynamic Binding so it called Employee display**

**}**

**}**

**> The output of the program**

**> Employee display**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**27/5 (class objects, c-o overiding)**

**Class object**

**• Hierarchy of class object object -> class Person -> class Emp**

**• Super class of Person is object and the object consists of the following methods.**

**• Methods oof class Object**

**public String toString(); public boolean equals(); public void wait(); public int hashCode(); protected void finalize();**

**• By default these methods are present in every object of the class.**

**p s v m(s [] a)**

**{**

**Person p = new Emp();**

**System.out.println(p); //p.toString()**

**//Person class inherited toString() from class Object**

**}**

**• It will print hash map line office.emps.Emp@262323a**

**Class object overiding**

**// in Person class**

**@override**

**public String toString()**

**{**

**return "Name: "+name+" birthdate:- " +bdate;**

**//name.toString(); bdate.toString();**

**}**

**• Output of this code is Name:- shree Birdthday:- utility.Date@5215525a;**

**• To avoid utility.Date@5215525a; on the place of birthdate you need to write return toString() in Date class.**

**• If there is toString() in Emp class then it will call Emp -> toString();**

**• In below code Emp class doesn't contain toString() but its super class contains the toString() so it call Person -> toString();**

**• And for birthdate, the flow is - main -> Emp(not toString()) -> Person(super of Emp) -> toString -> Date -> toString();**

**• It represents an object in string.**

**• It converts object to string.**

**• Purpose of toString() and display() are the same but toString prints directly with the help of object.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**28/5(Multilevel Inheritance,foreach, Abstraction, Access Specificer)**

**Multilevel Inheritance**

**• The flow of Executing constructor is Person()->Employee()->SalesManager()**

**foreach loop**

**Emp [] allemps;**

**allemps = new Employee[3];**

**allemps[0] = new SalesManager();**

**allemps[1] = new Admin();**

**// Shortcut**

**Employee [] allemps = {new SalesManager(), new Admin()};**

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

**// Advanced for loop foreach.**

**for(Employee e : allemps)**

**{**

**System.out.println(e); //e.toString;**

**}**

**// here we achive polymorphism sending same data to every method but every methods acts diffently**

**• foreach is only used for array.**

**• Protected data members are accesssible to its sub classes.**

**• If you do not write access specificer then the scope is package-private.**

**• Thre is 4 access specifier public, private, protected and default**

**• keywords are 3 and specifiers are 4.**

**Abstraction**

**• Abstract methods that are not declared in super class.**

**• Methods that are not declared are used for polymorphism.**

**• If one of the method of a class is declared as abstract then class should be abstract.**

**• Subclass must complete abstract method. If it fails to complete the abstract method, the subclass will be abstract.**

**• Creating instance of abstract class is not allow.**

**• We cannot create object of abstract class.**

**• Non abstract classes are concrete class.**

**• Non abstract classes can be declared as abstract.**

**• e.calsal(); will not called because it is not in Super class Emp.**

**public class Employee**

**{**

**public void display()**

**{**

**System.out.println("Employee Display");**

**}**

**public void accept()**

**{**

**System.out.println("Employee accept");**

**}**

**}**

**public class Manager extends Employee**

**{**

**display();**

**accept();**

**calsal();**

**}**

**public class SalesPerson extends Employee**

**{**

**display();**

**accept();**

**calsal();**

**bonous();**

**}**

**public class WageEmp()**

**{**

**display();**

**accept();**

**calsal();**

**}**

**public class MainClass()**

**{**

**Employee e = new Manager();**

**e.calSale(); //cannot be compile**

**}**

**• In the above code e.calSale() will not compile because it searches the function in super class i.e in Employee.**

**• That is when the abstration concept comes in picture.**

**• So you can declare the method in super class but there is no need to declare that method.**

**• You cannot create object of an abstract class.**

**• We cannot create object that's why we use array.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**29/5(geter/seter,final,Interface)**

**geter/seter**

**• The rule says that class data members should be declared as private.**

**• Because of this its unable to access in subclass.**

**• So there are 2 options to access that data member i.e 1. make it protected so it will accessible to sub class.**

**• And 2nd option is geter and seter methods.**

**• getter() method return private data member.**

**• setter() set new value to private data members.**

**public class Account**

**{**

**private String name;**

**private int no;**

**}**

**public String getName()**

**{**

**return name;**

**}**

**public String setName(String name)**

**{**

**this.name = name;**

**return name;**

**}**

**public int getNo()**

**{**

**return no;**

**}**

**public int setNo(int no)**

**{**

**this.no=no;**

**}**

**final**

**• final is the keyword in java.**

**• It is exactly opposite abstract.**

**• It cannot be modified/change.**

**• it is fully defined.**

**• It can be used in variable,methods and classes.**

**• final is replacement of const in java.**

**• We can only declare final but you have to initilize it afterwords.**

**• fianl variable is declared in all capital letters i.e final String NAME="shree"**

**• final Methods cannot be overriden.**

**• eg.class object methods wait(),notify(),notifyAll() are defined final.**

**• final classs cannot be modified**

**• String class in java library is declared as final.**

**• Because you cannot modify final class, it cannot act as a super class.**

**• final keyword for array.**

**Interfaces**

**• Interfaces are 100% incomplete.**

**• In interface, there are abstract methods and concrete methods.**

**• In Inteface, there is only final variable and abstract methods (for now)**

**• Multiple intheritance through the classes is not allowed but using interface it is allowed.**

**• Anything you write in interface will take public static final by default.**

**interface Drawable**

**{**

**static double PI = 3.14; //public static and final**

**double calArea(); //it can only declared as public andabstract**

**void drawShape(); //methods are bydefault treated as abstract**

**}**

**class Rectangle implements Drawable**

**{**

**privete double lenght,breadth;**

**//constractor**

**// here toString method is available**

**double calArea()**

**{**

**}**

**void drawShape()**

**{**

**}**

**}**

**class MainClass**

**{**

**public static void main(String [] args)**

**{**

**Drawable d = new Circle();**

**//interface reference = implementing class instance**

**//super class reference = sub class instance**

**d.calArea(); // Instance and non static methods resolve using dynamic binding**

**//so it will call from the Circle class.**

**}**

**}**

**• It has a is a relationship.**

**• You cannot create a object but you can create a reference of interface.**

**• You can create only reference of Interface.**

**• There is no any concrete method in interface( upto java 1.7).**

**• Compulsorily, you have to implement all the methods in sub class.**

**interface A**

**{**

**void m1();**

**}**

**interface B extends A**

**{**

**void m2();**

**}**

**class X implements B**

**{**

**void m1(){}**

**void m2(){}**

**}**

**• If two interfaces are extended with each other (A and B)**

**• Then, in implement class i.e (X) needs to implement both interface methods.**

**• Class and Interface relation**

**o class ---> (extends)---> class**

**o class ---> (implements)---> interface**

**o interface ---> (extends)---> interface**

**• Abstract class and Interface cannot create object.**

**• Make difference between class and Interface.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**31/5(typecast, instanceof, interface of java8)**

**Types of methods**

**• Final Methods**

**o Final methods are defined final it will not modified/overrided further.**

**• Concrete Methods**

**o In concrete methods, we have an option to override or not.**

**• Abstract Methods**

**o Abstract methods should be compulsorily overrided in subclass.**

**• Default Methods**

**o Default methods are declared in inheritance from java8 onwards as concrete methods in inheritance.**

**How multiple inheritance replaced by interfaces**

**• We can use extend and implement both on one class.**

**public class SalesManager extends Employee implement ITraveller**

**{**

**//add all unimplemented methods in ITraveller**

**//ITraveller is interface so you have to implement all methods which in ITraveller**

**}**

**typecast**

**• Convert the type of variable/object**

**• double d=4.6; ---> float f=(float)d;**

**Employee [] allEmp;**

**allEmp = new Employee[3];**

**allEmp[0]=new SalesPerson("",,,,);**

**allEmp[1]=new SalesPerson("",,,,);**

**allEmp[2]=new SalesPerson("",,,,);**

**for (Emp e : allEmp)**

**{**

**if(e instanceof ITreaveller) //It checks e can be typecasted toITreaveller so remove error that is showed in typecast**

**{**

**System.out.println("TA: "+((ITraveller)e).calculateTA());**

**}**

**}**

**instanceof and typecast**

**• instanceof is a operator**

**• It returns true and false value.**

**• Refer below code and error ---> It checks e can be typecasted toITreaveller so remove error that is shown in typecast**

**• syntax**

**e instanceof iTreaveller e source, ITreaveller is destination**

**• It avoids error during runtime.**

**• It is intelligent way of checking typecast is valid or not.**

**• Upcasting is not required (super class methods are available for subclass as it is).**

**• Downcast is required (because subclass methods are not available in super class).**

**• Extra ↓**

**• It verified calsalary(); is available in person or not**

**Interface onwards java 8**

**• Previously interface only contained abstract methods.**

**• If suddenly abstract method is added, interface will breakdown existing classes.**

**• When this was realized by the Oracle and other developers, they fixed it.**

**• Now we do not add any abstract method suddenly, instead of that use default keyword.**

**• Also you can add static method in interface.**

**• Now there are three type of methods in interface -**

**o Static (not expect to rewrire or overide)**

**o Abstract (compulsory override)**

**o default (do or don't)**

**interface A**

**{**

**void m1(); // compulsory to implement**

**default void m2() // not compulsory use it or not**

**{}**

**static void m3()**

**{**

**}**

**}**

**class MyClass implements A**

**{**

**void m1(){}**

**void m2(){}**

**}**

**• Interface can be implemented across the hierarchy (Not abstract classes)**

**• Interface decides what to do.**

**• Class decides how to do it.**

**• Download and see code here**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1/6**

**Exception Handling**

**• Handling the error occured during run time**

**• Ther are three type of error**

**Compilation logical run time**

**• Runtime errors are not under the control of the programmer.**

**• Java library has many exception classes are already defined.**

**• Exception object is thrown,**

**• Expection handling revolves around 5 keywords - try, catch, finally, throw and throws.**

**public class ExpectionDemo**

**{**

**public static void main(String [] args)**

**{**

**try**

**{**

**int n1= Integer.parseInt(args[0]);**

**int n2= Integer.parseInt(args[1]);**

**Syso("Division:"+(n1/n2));**

**}**

**catch(Expection e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**try-catch**

**try-catch-catch**

**try-finally**

**try-catch-catch-finally**

**Finally block**

**• Real purpose of finally block is not to handle expection.**

**• Even if there is no expection still finally block always get executed.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**8/7(Java IO, Stream)**

**Java IO(Input Output)**

**• Stream is a sequence of data which travels from source to destination.**

**• Streams are in java.io package**

**• Stream**

**o byte Stream**

**o Character Stream**

** String, char[]**

**• java i/o is based on 4 abstract classes**

**o Byte**

**o Input Stream**

**o Output**

**o Character**

**o Reader**

**o Writer**

**• Three steps every I/O operation should be follow =**

**o Open Stream**

**o Use Stream**

**o Close Stream**

**• printStream class contain print & println and out as object.**

**• In some Stream classes the Source and Destination is not defined.**

**• eg.ObjectOutputStream**

**• To overcome this, we use object chaining**

**• Two streams are together for performing some IO.**

**• FullIOoperation = IO Operation1 + IO operation2**

**• FileOutputStream + ObjectOutputStream**

**// Object chaining**

**Person p = new Person("shree",new DOB(7,8,97));**

**Consol IO**

**• Predefined Stream**

**o System.in**

**o System.out**

**o System.err**

**• .err/.out/.in <-- are the Static data members of System class**

**• .out is object of printStream class**

**BufferReader**

**• BufferReader is the character reader stream**

**• Some important functions of BufferReader**

**• br.read()**

**o It returns a integer single char**

**o int i = br.read();**

**• br.readLine()**

**o It returns a String**

**o String str = br.readLine();**

**// Dummy/psudoCode**

**BufferReader br = new BufferReader(new inputStreamReader(System.in)); // OPEN STREAM**

**int n = br.read(); // USE STREAM**

**System.out.println((char)n);**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**System.out.println("Enter 'q' for exit");**

**while((n=br.read()) != 'q') // USE STREAM**

**{**

**System.out.println((char)n);**

**}**

**br.close(); // CLOSE STREAM**

**// Real Code**

**public static void main(String [] args)**

**{**

**// ISR is passed for creating BR instance is object chaining**

**BufferReader br=null;**

**try**

**{**

**br = new BufferReader(new InputStreamReader(System.in));**

**syso("Enter q to quit");**

**int n;**

**while(n=br.read())!='q')**

**{**

**syso((char)n);**

**}**

**}catch(IOException e)**

**{**

**e.printStackTree();**

**}finally**

**{**

**try{**

**br.close();**

**}catch{**

**}**

**}**

**}**

**• BufferReader return type is char so we need to parse a data as follows -**

**Parsing BufferReader**

**int sid = Integer.parseInt(br.readLine());**

**String name = br.readLine();**

**float marks = Float.parseFloat(br.readLine());**

**double percent = Double.parseDouble(br.readLine());**

**char ch = br.readLine().charAt(0);**

**File IO**

**File Reader**

**• Stream**

**o FileInput -byte read**

**o FileReader - char read**

**o FileOutputStream - byte write**

**o FileWriter - char write**

**• Creating Stream**

**• It needs information of file**

**• We give information as String path or file object**

**• File object contains metadata (the data about data)**

**o file size, readable,extra path contain in metadata**

**• There are two ways in which we can pass the file to file stream -**

**o Direct path**

**o File object**

**File path**

**// direct path passing**

**FileInputStream fis = new FileInputStream("path")**

**// File object**

**File f = new File("path");**

**FileInputStream fis = new FileInputStream(f)**

**• The benifites of passing FileObject is we can use multiple functions in File like**

**o f.length();**

**o f.canRead();**

**o f.isFile();**

**o f.isDirectory();**

**o f.lastModified();**

**// File Reader**

**FlieReader fr = new FileReader("path");**

**int n;**

**while((n=fr.read()) != -1) // -1 is end of stream**

**{**

**System.out.println((char)n);**

**}**

**fr.close();**

**File Writer**

**• Byte Stream**

**// Byte Stream Program**

**public static void main(String [] args)**

**{**

**String s = "github.com/shreeshailaya";**

**FileOutputStream fos = null;**

**try**

**{**

**// for rewrite data**

**FOS = new FOS("path");**

**// for append data**

**FOS = new FOS("path",true);**

**byte [] arr = s.getBytes();**

**fos.write(arr); // entire array**

**fos.write(arr[0]); // one byte**

**fos..write(arr,0,7); // first 7 char**

**fos.write("\n".getBytes()); // new line**

**}**

**catch(FileNotFoundExpection ex)**

**{**

**ex.printStackTree();**

**}**

**finally**

**{**

**try catch**

**fos.close();**

**}**

**}**

**• Character Stream**

**// character stream program**

**public static void main(String [] args)**

**{**

**//for rewrite**

**FileWritter fw = new FileWritter("path");**

**// for append**

**FileWritter fw = new FileWritter("path",true);**

**Syso("Enter Data");**

**BufferReader br = new BufferReader(new inputStringReader(System.in));**

**String s = br.readLine();**

**fw.write(s);**

**fw.write("\n");**

**fw.write(s,0,10);**

**}**

**Reading file**

**class FileReadingDemo**

**{**

**p s v m(String [] args)**

**{**

**File f = new File("path");**

**sysout("Exist: "+f.exist());**

**if(f.exist() && f.isFile() && f.canRead())**

**{**

**FileReader fr = new FileReader(f);**

**int n;**

**while((n=fr.read() != -1))**

**{**

**sysout(n)**

**}**

**}**

**fr.close();**

**}**

**}**

**• Reading data by Char array**

**char [] ch = new Char[10];**

**int n;**

**while ((n=fr.read(arr)) != -1)**

**{**

**sysout(arr);**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**8/7**

**buffer Stream**

**• Buffer pulls data from file**

**• Buffer is temporary memory to file IO**

**• Loading the next data is done by buffer**

**• The purpose of buffer is memory IO and to avoid disk IO**

**• Disk IO is slower and time taking**

**• Same is applicable to write**

**• Program first writes data in buffer. When buffer is full, then it will flush the data to file**

**• Flushing is occured when buffer is full**

**• Also closing the stream will flush the data**

**• flush() is also used to flush the data to file**

**• Giving the size to buffer is optional. You can give with one of constructor**

**• We need to pass object according to the stream.**

**• Previousely we directly read/write data from hdd now here we read/write data with the help of buffer**

**//Syntax**

**bufferReader brr = new bufferReader(new FileReader("path"));**

**bufferWriter brr = new bufferWriter(new FileWriter("path"));**

**bufferOutputStream bos = new BOS()**

**int n;**

**while((n=br.read()) != -1)**

**{**

**sysout((char)n);**

**}**

**bufferWriter bw = new BW(new FileWriter("path"));**

**bw.write("Some String"); //Write on buffer**

**bw.newLine(); // new line method**

**bw.write("Some String");**

**bw.flush(); // remaining data flush to file**

**bw.close();**

**Object IO**

**Serialzation/ Deserialzation**

**• Saving/writing object state in some permanent media like file and DB.**

**• Retrive object state in some permanent media like file and DB.**

**• Streams used are -**

**o ObjectOutputStream --> write object/serialization**

**o ObjectInputStream --> read object/deserialization**

**• We are dealing with object here**

**• We are saving the state of the object in file/db**

**• state are the all instance data members**

**• static data members will be discarded here because they are realted with class**

**• No methods i.e behavior**

**• Only set of all instance data members are saved**

**• Some instance data members may be excluded, you can use keyword transient**

**• int empid, date doj, long noOfDays, String name are the state of object**

**• transient is not used to save data, syntax --> transient long noOfDays**

**• Object that is getting serialized should have serializable implemention**

**• Serialized interfaces are empty they do not contain any methods. They are called marker interface. It is only used for marking the object for serialization**

**• Only Serialized objects can be searized else it will create NotSerializableException**

**• association - has a**

**• interface - is a**

**• In is a relation do it at the level of super class**

**• In has a relationship do seperately for all component**

**• If implementation of serialization is done for parent class, then child class is automatically affected**

**• For serialization, we use objectOutputStream supported with FileOutput**

**• byte stream are compatible with byte stream**

**• character stream is only compactable with char stream**

**• For reading object from file we need OobjectinputStream with FileInputStream**

**• osi.readObject() is the method which reads the object or returns object**

**• Then we have to do down typecast to see the data**

**public static void main(String[] args) throws FileNotFoundException, IOException {**

**// TODO Auto-generated method stub**

**Employee [] emp = new Employee[3];**

**emp[0]=new Employee(1, "shree");**

**emp[1]=new Employee(2, "shail");**

**emp[2]=new Employee(3, "Vitkar");**

**ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("/home/shree/Desktop/shree"));**

**for(Employee e : emp)**

**{**

**oos.writeObject(e);**

**}**

**System.out.println("Serialization Done");**

**}**

**// Deserialization**

**public static void main(String[] args) throws FileNotFoundException, IOException, ClassNotFoundException {**

**// TODO Auto-generated method stub**

**ObjectInputStream ois = new ObjectInputStream(new FileInputStream("/home/shree/Desktop/shree"));**

**Employee [] emp = new Employee[3];**

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

**{**

**//Object o = ois.readObject();**

**emp[i]=(Employee)ois.readObject();**

**}**

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

**{**

**System.out.println("Employee Name "+emp[i].getName()+"\n Employee ID "+emp[i].getEmpid());**

**}**

**}**

**Collection Framework**

**• It gives ready-made classes which have implemented a particular data structure**

**• All collection classes will store collection of objects not primitive data types**

**• Collections cannot be of int value but can be of Integer object**

**• Collection framework is re-designed after version 1.2/**

**• It is a class which acts as a data structure**

**• Classes are based on set of interfaces**

**• BAse interface is Collection**

**• There are few methods which are required in all collections like -**

**o add();**

**o remove();**

**o size();**

**o iterator();**

**• Collection is further extended to =**

**o List**

**o Set**

**o Queue**

**• Set is further divided into =**

**o SortedSet();**

**• List can consist of duplicates**

**• Set is a set of unique data**

**• We are going to see classes in interface (list set and queue)**

**List Interface**

**• Duplicate elements are allowed in list**

**• abstractList is a class**

**• Two important list classes are**

**o ArrayList**

**o LinkedList**

**• ArrayList is most used collection**

**• ArrayList can dynamically grow and shrink and workded as array**

**• When there is a requirment to search data, we use ArrayList**

**• When there is a requirment to add/delete data, then we use LinkedList**

**List l = new ArrayList();**

**l.add("bakul");**

**l.add(new Person());**

**String s = (String)l.get(2); // It will genetrate class castclassexception**

**// we cannoct convert Person object as string**

**// care has to be taken at the time of adding the element**

**List <String> l = new ArrayList<>();**

**// because of <String> i can only pass/add string so there will not need of type cast**

**• is generics which will add type safety to the collection**

**• Because of this we can only add string, no need of typecast**

**//without using generic**

**class Demo**

**{**

**//List l = new ArrayList(); //This will generate error**

**List <String> l = new ArrayList<>();**

**l.add(new String("Welcome"));**

**//l.add(new Date());**

**//l.add(new Integer(45));**

**//String s = (String)l.get(1);**

**// String s = l.get(0);**

**Sysout(l); //call toString print all elements**

**l.add(2,"xyz"); //addded to index 2**

**l.remove(3); //remove element**

**l.remove("xyz"); // remove by value**

**// advanced loop**

**for( String s : 1)**

**sysout(s.toUpperCase());**

**}**

**9/7(Collections)**

**Utility Classes**

**• It is a collection of objects**

**• You can add any object to collection**

**• Base of collection is lied on set interface**

**• Collections is an interface that has some methods add,remove..etc**

**• Collections is extended to List, Set and Queue**

**• In list, duplicates are allowed**

**Iterator method for ittration data from collection**

**\*\*\*\* // Collection upto Reversion**

**Collection freame work**

**• Types of collections classes**

**o List**

** al**

** ll**

**o set**

** hs**

** ts**

**o queue**

** pq**

**• adding, removing, Itreating**

**• common algos Collections (class with static methods)**

**• Map key value map --> set**

**• lagecy classes**

**o stack**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**10/7**

**Iterating**

**• We can iterate data from three ways from collections =**

**o Advanced for loop**

**o using iterator**

**o List Iterator(only for list)**

**• Advanced for loop**

**set<String> str = new ArrayList<>();**

**for(String s : str)**

**{**

**syso(s);**

**}**

**• using iterator**

**• Iterator has two methods as follows -**

**o has()**

**o hasNext()**

**List<String> str = new ArrayList<>();**

**Iterator i = str.iterator();**

**while(hasNext())**

**{**

**syso((String)next());**

**}**

**• ListIterator**

**• ListIterator has three methods as follows -**

**o next()**

**o hasNext()**

**o hasPrevious()**

**List<String> str = new ArrayList<>();**

**ListIterator<String> li = str.listIterator();**

**while(li.hasNext())**

**li.next();**

**while(li.hasPrevious())**

**sysout(li.previous());**

**HashSet**

**• Hashing is the storing a data in a particular way**

**13/7**

**Multithreading**

**• What is a thread? Compare thread process?**

**• Why multithread application?**

**• Thread life cycle**

**• Creating a simple thread**

**• There are two different ways to create a thread**

**o extends Thread (Class)**

**o implement runnable (Interface)**

**• Thread**

**• Specific to the task**

**o run()**

**• control the task**

**o start()**

**o join()**

**o Sleep()**

**• In runnable it has only one method that is run();**

**Create a thread**

**//syntax**

**class MyThread extends Thread**

**{**

**p v run()**

**{**

**//specify the task**

**}**

**}**

**class ThreadApp**

**{**

**p s v m();**

**{**

**Thread t = new MyThread(); //newly created state**

**t.start(); // before call run you have to call start()**

**// when you call start() then it will automically call run() from MyThread class**

**}**

**}**

**// Implement Runnable**

**class MyTask implements Runnable**

**{**

**pv run()**

**{**

**//specify task**

**}**

**}**

**class ThreadApp**

**{**

**main()**

**{**

**Runnabble r = new MyTask();**

**Thread t = new Thread(r); //newly created state**

**t.start(); // call run()**

**// we are indirect calling the start by thread**

**// Runnable is always depend on thread**

**}**

**}**

**• Runnable always depends on thread**

**class MyThread extends Thread**

**{**

**public void run() //task to be done by the thread**

**{**

**for(int i =1;i<=5;i++)**

**{**

**sysout("child Thread"+i);**

**try**

**{**

**Thread.sleep(500); //sleep is static method**

**// we cannot handle throws exception here**

**}**

**catch (InterruptedException e)**

**{**

**e.printStackTree();**

**}**

**}**

**}**

**}**

**main()**

**{**

**Thread t = new MyThread();**

**t.start**

**for(int i=1;i<=5;i++)**

**{**

**}**

**}**

**// Runnable**

**main()**

**{**

**// represent task**

**Runnable r = new MyTask();**

**Thread t = new Thread(r);**

**}**

**Threasd Synchronization**

**• Because of Multiple thread we achive better performance**

**• When multiple thread share common resources (object), it may lead to data inconsistancy**

**• Thread Sync is needed when multiple thread share common resources (Object)**

**• Because every thread gives some time, it not able to complete its task**

**• 1st thread will lock the object**

**• Other processes will go to wait because the object is locked**

**• When 1st thread completes the task, it will unlock**

**• Best example is ATM**

**• Here is the delay of time but we are achieving data consistency**

**• Synchronised keyword is used to lock the object**

**• You can use sync keyword on a block of code or method**