SHARATHKUMAR G S

Abhishek sir conducted test on 27-02-2023

1)Explain About the below given concepts

* Inheritance
* Encapsulation
* Polymorphism
* Abstraction

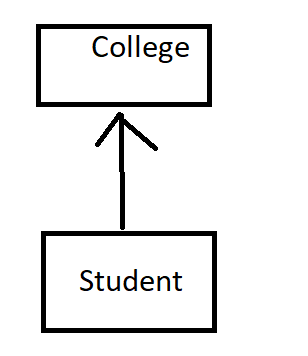
Give a example for these concepts

Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **encapsulation ,Abstract**, **polymorphism**, etc.

A)Inheritance :-

* one object acquires all the properties and behaviors of a parent object, it is known as inheritance
* It provides code reusability.
* It types:single inheritance,multilevel inheritance ,heirachial inheritance,multiple inheritance,hybrid inheritance

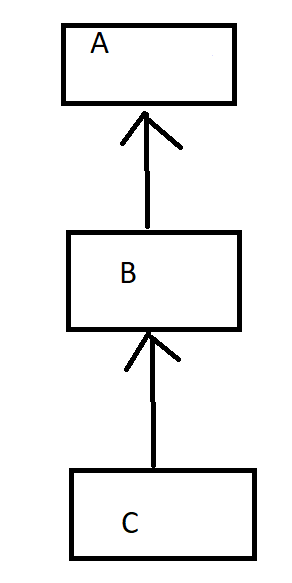
1) Programm for **Single inheritance;**



//create a  class for College

* Class College{
* String name =”xworkz”;
* }
* //create a class for Student AND extends to parent class College
* Class Student { s
* String name=”sharath”;
* Public static void main(String[] args)
* {    Student s =new Student();
* System.out.println(s.name );
* System.out.println(s.name);
* }
* }

2)Multilevel inheritance



When there is a parent multiple parent and child class in sequential order is called multilevel inheritance

Class A

{

void eat()

{

System.out.println (“eating”);

}

Class B extends A

{

Void bark()

{

System.out.println(“bark”);

}

}

Class C extends B

{

void sleep()

{

System.out.println(“sleeping”);

}

}

Class MutilevelExample

{

Public static void main(String[] args)

{

C c= new C();

System.out.println(c.sleep());

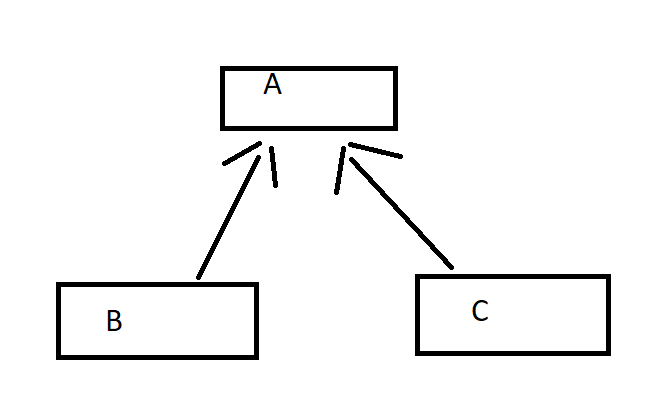
System.out.println(c.bark());

System.out.println(c.eat());

}

}

3)hierarchial inheritance



When two or more classes inheriting the same class is called hierarchial inheritance

**package** com.xworkz.inheritence;

**public** **class** A {

**void** eat()

{

System.***out***.println("EATING");

}

}

**package** com.xworkz.inheritence;

**public** **class** B **extends** A {

**void** walking()

{

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

}

}

**package** com.xworkz.inheritence;

**public** **class** C **extends** A {

**void** sleep()

{

System.***out***.println("sleeping");

}

}

**package** com.xworkz.inheritence;

**public** **class** Runner {

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

C c = **new** C();

c.eat();

c.sleep();

}

}

Output:

Eating

Sleeping for c object

if

B b = new B()

b.eat();

b.walk();

output is :eating

walking

4)multiple inheritance

A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

**package** com.xworkz.inheritence;

**public** **class** A {

**void** eat()

{

System.***out***.println("EATING");

}

}

**package** com.xworkz.inheritence;

**public** **class** B {

**void** walking()

{

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

}

}

**package** com.xworkz.inheritence;

**public** **class** C **extends** A,B{

C c= new C();

c.walking();

}

}

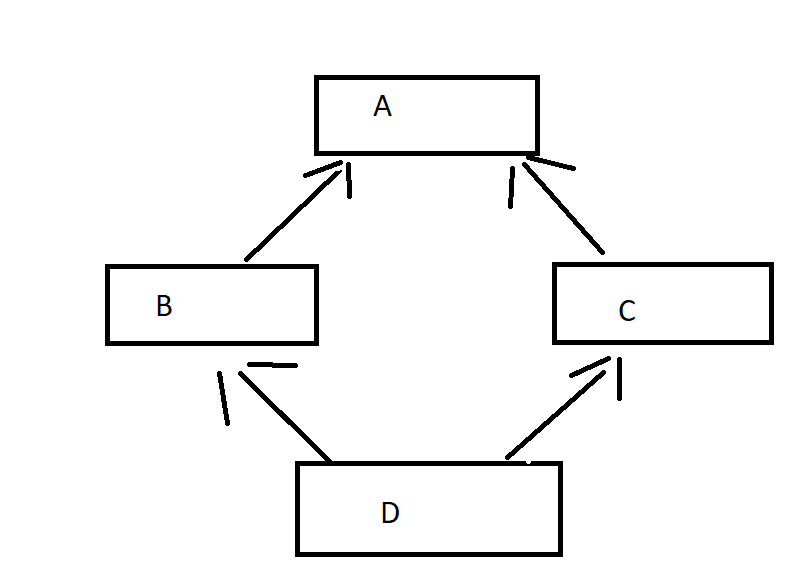
Output:

Compile time error because ambiguity problem occurs

So we achieve this through interface

5)Hybrid inheritance:

Combination of two or more types of inheritance is called hybrid inheritance



**class** A{

}

**class** B **extends** A {

}

**class** C **extends** A {

}

**public** **class** D **extends** B {

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

{

A a = **new** A();

B b = **new** B();

C c = **new** C();

System.***out***.println(a **instanceof** A);

System.***out***.println(b **instanceof** B);

System.***out***.println(c **instanceof** A);

}

}

Output:

True

True

True

* Encapsulation

Process of wrapping code and data together into a single unit is called Encapsulation

Example is capsule or in java way example is **Java Bean**

**package** com.xworkz.inheritence;

**public** **class** Encapsulationsss {

**private** String name;

**private** **int** rollNumber;

**private** **int** age;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getRollNumber() {

**return** rollNumber;

}

**public** **void** setRollNumber(**int** rollNumber) {

**this**.rollNumber = rollNumber;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

}

**package** com.xworkz.inheritence;

**public** **class** Runner {

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

Encapsulationsss ec = **new** Encapsulationsss();

ec.setName("sharath");

ec.setRollNumber(1333);

ec.setAge(25);

System.***out***.println(ec.getName()+"\n"+ec.getRollNumber()+"\n"+ec.getAge());

}

3)Polymorphism

Poly=means many

Morphism =forms

Definition:one task is performed in different ways*, it is known as polymorphism*

There are two types of polymorphism

1)compile time polymorphism/overloading

2)runtime polymorphism/overriding

**1)compile time polymorphism/overloading**

If a Class has multiple methods having same name but different in parameters, it is known as

**compile time** /**method Overloading**.

**Its advantage is :** Method overloading is increasing the readability of the program.

There are

1)Method without return type and without parameters();

2)Method with return type and without parameters();

3)Method with out return type and with parameters();

4)Method with return type and with parameters(0;

1)public void m()

{

System.out.println(“hello”);

}

2)int m(){

return 10;

}

3)void(int a)

{

System.out.println(a);

}

4)int m(int a)

{

System.out.println(“hello”);

Return a;

}

2)method overridding:

Method overriding means same method by changing the implementation is called method overloading

Rules

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class
3. There must be an is-a relationship

Class A{

Void add()

{

System.out.println(“hello adding 2 numbers”);

}

}

Class c extends A{

Void add()

{

System.out.println(“hello adding 3 numbers”);

}

}

Class runner

{

Public static void main(String[] args)

{

B b= new B();

b.add();

}

}

Output is always the override method implementation only answer is :-hello adding 3 numbers

4)Abstratction

Definition:the name itself definition hiding the implementation and showing the functionality is called abstraction

**public** **abstract** **class** Runner {

**abstract** **int** interest();

}

**class** Union **extends** Runner{

**int** interest(){**return** 5;}

}

**class** Corp **extends** Runner{

**int** interest(){**return** 3;}

}

**class** TestBank{

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

Runner b=**new** Union();

System.***out***.println("Rate of Interest is: "+b.interest()+" %");

Runner b1=**new** Corp();

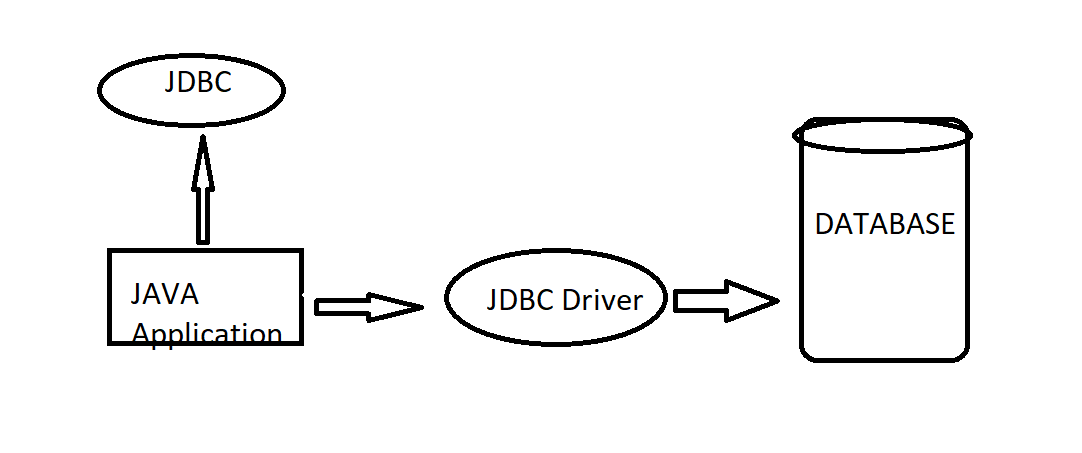
System.***out***.println("Rate of Interest is: "+b1.interest()+" %");

}}

2)Explain in summary about JDBC?

JDBC-JAVA DATABASE CONNECTIVITY

* JDBC is a Java API to connect and execute the query with the database



* Register the Driver class
* Create connection
* Create statement
* Execute queries
* Close statement
* Close connection

1)register the driver class

Syntax:

Class.forName(com.mysql.cj.jdbc.Driver)

The  method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

2)Create connection:

Connection connection = DriverManager.getConnection(URL,USERNAME,PASSWORD);

Create a connection with the database. Establishing by DriverManager.getConnection(by passing three paramteres that is URL,USERNAME,PASSWORD);

* 3) Create statement

PreparedStatement stmt=con.createPreparedStatement();

It will create

* 4) Execute queries

Query.executeUpdate();

Query.executeQuery();

Query.getResultList();

Query.getSingleResult();

4)close statement();

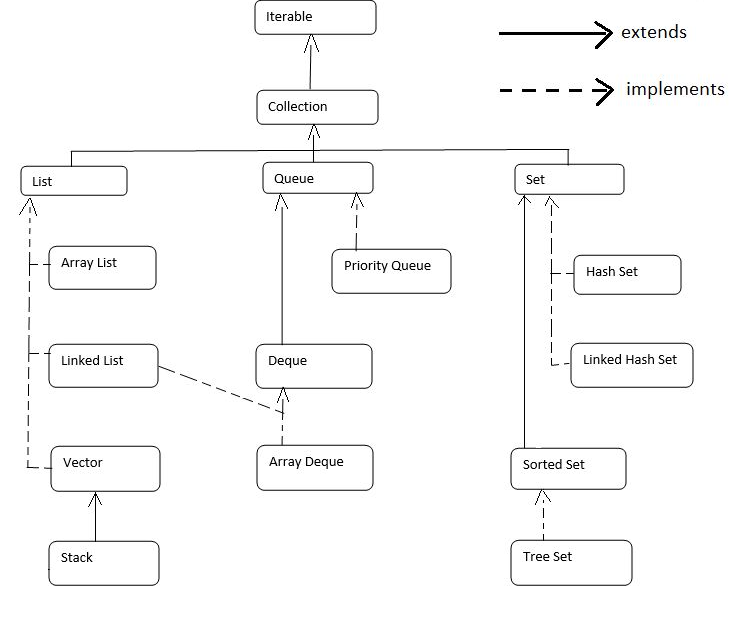
Stmt.close();

5)close connection();

Connection.close();

3)Explain the hierarchy of Collection using Diagram

Write briefly about LinkedList,HashMap,TreeSet



Linked List:

DataStructure:doubly linked list

Linked List will allow

Insertion order preserved

Duplicate allows

Null value

Is non synchronized

Manipulation is fast because no shifting

Hashmap

* Java Hashmap contains values based on the key.
* Java Hashmap contains only unique keys.
* Java Hashmap may have one null key and multiple null values.
* Java Hashmap is non synchronized.
* Java Hashmap maintains no order.
* The initial default capacity of Java Hashmap class is 16 with a load factor of 0.75.

TreeSet

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quiet fast.
* Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.
* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quite fast.
* Java TreeSet class doesn't allow null elements.
* Java TreeSet class is non-synchronized.
* Java TreeSet class maintains ascending order.
* The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

4)Difference between hibernate and JPa

|  |  |
| --- | --- |
| Jpa | hibernate |
| .javax.persistence | Org.herbernate |
| It is not an implementation | Is the implementation of jpa |
| Standard api that permits to perform database operation | It is used in mappting java datatypes with sql |
| It uses java persistence query language | It uses hibernate query language |
| It uses Entitymanagerfactory | Sessionfactory |
| To make action use Entityamanager | To make action Session |