* **QUESTIONS ON ENCAPSULATION & INHERITANCE**

**Q.1 What is Encapsulation in Java?**

Ans.:- **Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.The **Java Bean** class is the example of a fully encapsulated class.

**Q.2 Write a simple example of Encapsulation that has only one field with its setter and getter methods.**

Prog.:-

//A Java class which is a fully encapsulated class.

//It has a private data member and getter and setter methods.

**package** com.upgrad;

**public** **class** Student{

//private data member

**private** String name;

//getter method for name

**public** String getName(){

**return** name;

}

//setter method for name

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

**this**.name=name

}

}

//A Java class to test the encapsulated class.

**package** com.upgrad;

**class** Test{

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

//creating instance of the encapsulated class

Student s=**new** Student();

//setting value in the name member

s.setName("Madhusudan");

//getting value of the name member

System.out.println(s.getName());

}

}

**OUTPUT:-**

Madhusudan

**Q.3 Write an example of encapsulation that has only four fields with its setter and getter methods.**

//A Account class which is a fully encapsulated class.

//It has a private data member and getter and setter methods.

**class** Account {

//private data members

**private** **long** acc\_no;

**private** String name,email;

**private** **float** amount;

//public getter and setter methods

**public** **long** getAcc\_no() {

**return** acc\_no;

}

**public** **void** setAcc\_no(**long** acc\_no) {

**this**.acc\_no = acc\_no;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** **float** getAmount() {

**return** amount;

}

**public** **void** setAmount(**float** amount) {

**this**.amount = amount;

}

}

//A Java class to test the encapsulated class Account.

**public** **class** TestEncapsulation {

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

    //creating instance of Account class

    Account acc=**new** Account();

    //setting values through setter methods

    acc.setAcc\_no(9987177531L);

    acc.setName("Madhusudan Kamsali");

    acc.setEmail("madhusudankamsali@upgrad.com");

    acc.setAmount(500000f);

    //getting values through getter methods

    System.out.println(acc.getAcc\_no()+" "+acc.getName()+" "+acc.getEmail()+" "+acc.getAmount());

}

}

**OUTPUT:-**

**9987177531 Madhusudan Kamsali madhusudankamsali@upgrad.com 500000.0**

**Q.4 What is Inheritance in java?**

Ans.:- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

**Q.5 What are the term used in Inheritance?**

* **Class:**A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:**Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:**Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability**: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**Q.6 Write a Single Inheritance Example.**

**class** Animal{

**void** eat(){System.out.println("eating...");

}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");

}

}

**class** TestInheritance{

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

Dog d=**new** Dog();

d.bark();

d.eat();

}

}

**OUTPUT:-**

barking...

eating...

**Q.7 Write a Multiple Inheritance Example.**

**class** Animal{

**void** eat(){System.out.println("eating...");

}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");

}

}

**class** BabyDog **extends** Dog{

**void** weep(){System.out.println("weeping...");

}

}

**class** TestInheritance2{

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

BabyDog d=**new** BabyDog();

d.weep();

d.bark();

d.eat();

}

}

**OUTPUT:-**

weeping...

barking...

eating...

**Q.8 Write a Hierarchical Inheritance Example.**

**class** Animal{

**void** eat(){System.out.println("eating...");

}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");

}

}

**class** Cat **extends** Animal{

**void** meow(){System.out.println("meowing...");

}

}

**class** TestInheritance3{

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

Cat c=**new** Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}

}

**OUTPUT:-**

meowing...

eating...