EXPERIMENT NO. 4

1. What is inheritance?

Ans:

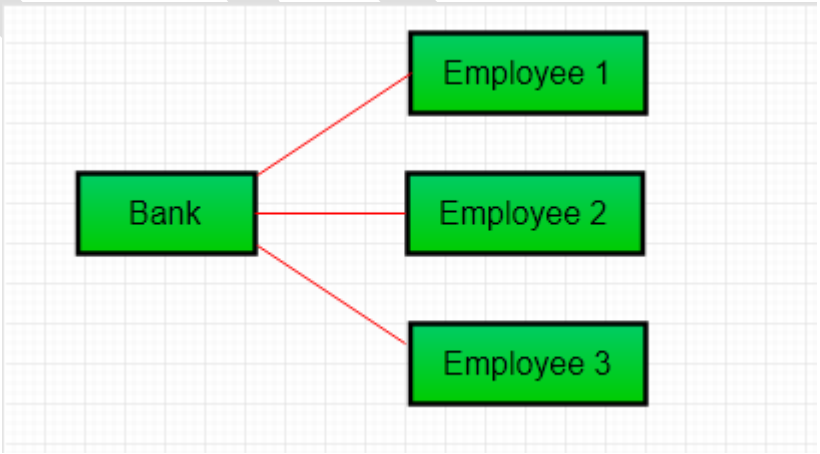
Inheritance is one of the core concept of object-oriented programming. It is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods. In Java, each class can only be derived from one other class. That class is called a superclass, or parent class. The derived class is called subclass, or child class.

2.How you differentiate association, aggregation and inheritance with example in java.

Ans: Aggregation:

- It represents **Has-A** relationship.
- It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation,**both the entries can survive individually** which means ending one entity will not effect the other entity

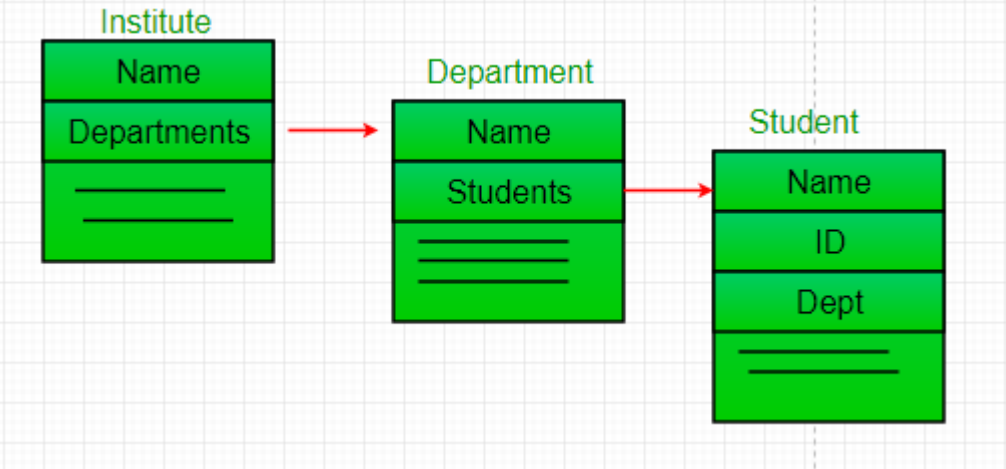
Aggregation



- It represents **part-of** relationship.

•In composition, both the entities are dependent on each other.
•When there is a composition between two entities, the composed object **cannot exist** without the other entity.

Composition



3 Give an example of 'this' and 'super' statement ?

Ans:

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
class JBT {
public static void main(String[] args) {
JBT obj = new JBT();
obj.methodTwo();
```

```
}
void methodOne(){
System.out.println("Inside Method ONE");
}
void methodTwo(){
System.out.println("Inside Method TWO");
this.methodOne();// same as calling methodOne()
}
}
```

4.Why Object class is by default super class for any class in java

Ans:

Java.lang.Object class is the super base class of all Java classes. Every other Java classes descends from Object. If a class is declared without extending another class then it will implicitly extend Object class. This is taken care of by the JVM.

5.What are the different method of Object class ?

Ans:

1. **toString()**: toString() provides String representation of an Object and used to convert an object to String.
2. **hashCode()**: For every object, JVM generates a unique number which is hashcode. It returns distinct integers for distinct objects.
3. **equals(Object obj)**: It gives a generic way to compare objects for equality.
4. **getClass()**: Returns the class object of "this" object and used to get actual runtime class of the object.
5. **Finalize()** method : This method is use to dispose system resources, perform clean-up activities and minimize memory leaks.
6. **clone()**: It returns a new object that is exactly the same as this object.
7. **Wait()**, N**otify(),NotifyAll()-related to concurrency.**

6.What is upcasting? What are the benefits of upcasting ?

Ans:

Upcasting: Java permits an object of a sub class can be referred by its super class. It is done during Runtime.

Benefit of Upcasting:-

- Gives us great advantage like polymorphism or grouping different objects.

- You can access the methods in the super class through the object of sub-class

7.What is use of @Override annotation. ?

Ans:

@Override annotation is used when we override a method in sub class. Using @Override annotation while overriding a method is considered as a best practice for coding in java because of the following two advantages:

1) If programmer makes any mistake such as wrong method name, wrong parameter types while overriding, you would get a compile time error. As by using this annotation you instruct compiler that you are overriding this method. If you don't use the annotation then the sub class method would behave as a new method (not the overriding method) in sub class.

2) It improves the readability of the code. So if you change the signature of overridden method then all the sub classes that overrides the particular method would throw a compilation error, which would eventually help you to change the signature in the sub classes. If you have lots of classes in your application then this annotation would really help you to identify the classes that require changes when you change the signature of a method.

8. What is final field, final method, final class in JAVA?

Ans:

Final Field-final variables must be used only for the values that we want to remain constant throughout the execution of program. When a variable is declared with *final* keyword, its value can't be modified, essentially, a constant.

Final Method-When a method is declared with *final* keyword, it is called a final method. A final method cannot be overridden. The Object class does this—a number of its methods are final.

Final Class-Final Class is the class which cannot be Inherit to another class.

9.What is the use of Static Field and Static Variable?

Ans:

Static Fields

A field of a class that is static can be accessed without an instance of the class, they are good means of storing information.To make a static field, you simply place the *static* keyword before the declaration of a class's member variable, or field:

Static Methods

A static method of a class can be called without an instance of that class. As a result, static methods don't have any access to instance variables, or instance fields, because instance variables store information about an instance of a class. Likewise, static methods do not have access to the *this* keyword in Java because t*his* refers to the current instance of a class, which is not present in the scope of a static method. Further, instance methods of a class cannot be called from static methods of that class because there is no instance of the class present in a static method. However, if an instance of the class is passed to the static method, that instance is free as usual to perform its instance methods in the scope of the static method.

We use static methods for functionality that is universal to a class. Static methods do not care about instances of the class or their state. Finally, instances of a class can call the public static methods of that class from anywhere in the program, but an instance of a class can only call its private static methods, like any other private method, from within the class.

10. **What is dynamic method dispatch ?**

Ans:

Runtime Polymorphism or Dynamic method dispatch

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.When an overridden method is called by a reference, java determines which version the type of object which it referred determines which version of overridden method will be called.

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.