Experiment No. 8
Implement a program multiple inheritance with interface
Date of Performance:
Date of Submission:

Aim: Implement a program on multiple inheritance with interface.

Objective: Implement multiple inheritance in a program to perform addition, multiplication and transpose operations on a matrix. Create an interface to hold prototypes of these methods and create a class input to read input. Inherit a new class from this interface and class. In main class create object of this child class and invoke required methods.

Theory:

- In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces.
- An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.
- However, Java supports multiple interface inheritance where an interface extends more than one super interfaces.
- A class implements an interface, but one interface extends another interface. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.
- The following is the syntax used to extend multiple interfaces in Java:

```
access_specifier interface subinterfaceName extends superinterface1, superinterface2, …… {
```

```
// Body }
```

Code:

```
class MultipleInheritance{  public
static void main(String args[]){  BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
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...");}
}
```

Conclusion:

Interfaces are incredibly useful for achieving abstraction and implementing polymorphism. They allow us to define a contract or set of methods that a class must implement, without specifying how those methods are implemented. This promotes loose coupling and flexibility in our code, as we can define multiple classes that implement the same interface and provide different implementations for the methods. By programming to interfaces, we can write code that is more modular, extensible, and maintainable. To implement an interface in Java, a class simply needs to use the "implements" keyword followed by the interface name. The class must then provide implementations for all the methods defined in the interface. This allows us to achieve the benefits of interfaces, such as code reusability and the ability to work with objects of different classes through a common interface.