Name	**Shubhan Singh**
UID no.	**2022300118**
Experiment No.	**7**

PROBLEM STATEMENT :	*Mammal class that has following attributes* *a) Category //animal or bird* *b) No. of legs* *Dog class inherits from mammal class and attributes are* *a) Name* *b) Age* *Method calculateLifeExpectancy returns the years of the dog based on the breed.* *Method popularBreed will return the most popular breed* *Labrador class inherits from Dog and has attributes* *a) Colour* *b) Weight* *Method speak returns the sound of the dog* *public String speak() { return "WOOF"; }* *Method calculateAvgWeight returns the average weight of all Labrador.* *German Shepherd class inherits from Dog and has attributes* *a) Height* *b) Lifespan* *Also has method that returns the sound of the dog* *public String speak() { return "GRUNT"; }* *Method calculateAvgHeight returns the average height of all German Shepherd.*
THEORY:	**@Override annotation in Java:** The **@Override** annotation in Java is used to indicate that a method in a subclass is intended to override a method with the same signature in its superclass. This annotation provides a compile-time check to ensure that the method is indeed overriding a method in the superclass, preventing accidental mistakes or typos in method names or parameters. Here are some key points to understand about the **@Override** annotation in Java: 1. Usage: The **@Override** annotation is placed immediately before the

method declaration in the subclass that is intended to override a method in the superclass. It is an optional annotation but recommended to use for clarity and to catch any potential errors during compilation.

2. Purpose: The primary purpose of **@Override** is to provide a compile-time check, ensuring that the annotated method is a valid override of a method in the superclass. If the annotated method doesn't match a method in the superclass, a compilation error will occur.
3. Verification: When a method is annotated with **@Override**, the Java compiler checks if the annotated method matches a method in the superclass based on the method name, return type, and parameter types. If the compiler detects a mismatch, it will raise a compilation error.
4. Benefits: The **@Override** annotation helps to improve code clarity and maintainability. By using it, you explicitly indicate your intention to override a method, making the code more readable for other developers. It also serves as a safeguard against errors, ensuring that you are correctly overriding methods from the superclass.
5. Best practices: It is good practice to always use the **@Override** annotation when you intend to override a method. It is particularly important to use it when dealing with methods from interfaces or abstract classes to ensure that the overridden methods are correctly implemented.

Super keyword in Java:

The **super** keyword in Java is used to refer to the superclass (parent class) of a subclass. It provides a way to access and call the superclass's members, such as fields, constructors, and methods, from within the subclass. The **super** keyword is primarily used to differentiate between the members of the subclass and those of the superclass with the same name.

Here are some key points to understand about the **super** keyword in Java:

1. Accessing superclass members: The **super** keyword allows you to access superclass members that are hidden or overridden in the subclass. You can use **super** to access superclass fields, invoke superclass constructors, and call superclass methods.
2. Invoking superclass constructors: By using **super** in a constructor of a subclass, you can invoke the constructor of the superclass. This is particularly useful when the superclass has its own initialization

	logic that needs to be executed before the subclass can be properly initialized. 3. Differentiating members: If a subclass overrides a method or hides a field from the superclass, you can use the **super** keyword to refer to the superclass's version of the method or field. This allows you to access or invoke the superclass implementation from within the subclass. 4. Method overriding: When a subclass overrides a method from the superclass, the **super** keyword can be used to invoke the overridden method in the superclass. This is useful when you want to extend the behavior of the superclass's method in the subclass while still retaining the original functionality. 5. Superclass constructor invocation: When invoking a superclass constructor using **super**, it must be the first statement in the constructor body of the subclass. This ensures that the superclass initialization is performed before the subclass's own initialization.
PROGRAM:	``` import java.util.Scanner; class Mammals{ boolean is_bird; public int getNo_of_legs() { return no_of_legs; } public boolean isIs_bird() { return is_bird; } public void setNo_of_legs(int no_of_legs) { this.no_of_legs = no_of_legs; } public void setIs_bird(boolean is_bird) { this.is_bird = is_bird; } int no_of_legs; } class Dog extends Mammals{ public Dog(String name, int age){ is_bird=false; no_of_legs=4; this.name=name; this.age=age; } int age; ```

```
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
class Labrador extends Dog{
    public Labrador(String name, int age, String colour,
float weight){
        super(name, age);
        this.colour=colour;
        this.weight=weight;
        sumlabage+=age;
        sumWt+=weight;
        lab_count++;
    }
    String colour;
    float weight;
    public static int sumlabage=0;
    public String getColour() {
        return colour;
    }

    public float getWeight() {
        return weight;
    }

    public void setColour(String colour) {
        this.colour = colour;
    }

    public void setWeight(float weight) {
        this.weight = weight;
    }

    public static float sumWt=0;
    static int lab_count=0;

    public static int getLab_count() {
        return lab_count;
    }
    public static float avglabLifeExpectancy(){
        return (float) sumlabage /lab_count;
    }
```

```
    public String speak() { return "WOOF"; }
    public static float calculateAvgWeight(){
        return sumWt/lab_count;
    }
}
class germanSheperd extends Dog{
    public germanSheperd(String name, int age, float height,
int lifespan){
        super(name, age);
        this.height=height;
        this.lifespan=lifespan;
        sumHt+=height;
        sumshepage+=age;
        shep_count++;
    }
    float height;
    int lifespan;
    public static float sumHt=0;
    public static int shep_count=0;
    public static int sumshepage=0;
    public static float avgshepLifeExpectancy(){
        return (float) sumshepage /shep_count;
    }

    public static int getShep_count() {
        return shep_count;
    }

    public String speak() { return "GRUNT"; }
    public static float calculateAvgHeight(){
        return sumHt/shep_count;
    }
}

public class Tester_mammals {
    public static float avgLifeExpectancy(String breed){
        if(breed.equals("Labrador")){
            return Labrador.avglabLifeExpectancy();
        }
        else{
            return germanSheperd.avgshepLifeExpectancy();
        }
    }
    public static String popularBreed(){
        if(Labrador.lab_count>germanSheperd.shep_count){
            return "Labrador";
        }
        else if
(Labrador.lab_count==germanSheperd.shep_count){
            return "Both breeds are equally popular";
        }
        else{
            return "German Sheperd";
        }
    }
```

```
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter total number of labradors
and german sheperds:");
        int nolab=sc.nextInt();
        int noshep=sc.nextInt();
        Labrador[] Lab_arr=new Labrador[nolab];
        germanSheperd[] shep_arr=new germanSheperd[noshep];
        String tempname,tempcolor;
        int tempage,templifespan;
        float tempweight,tempheight;

        for(int i=0;i<nolab;i++){
            sc.nextLine();
            System.out.println("Enter name, age, color and
weight(in Kg) of Labrador "+(1+i));
            tempname=sc.nextLine();
            tempage=sc.nextInt();
            sc.nextLine();
            tempcolor=sc.nextLine();
            tempweight=sc.nextFloat();
            Lab_arr[i]=new
Labrador(tempname,tempage,tempcolor,tempweight);
        }
        for(int i=0;i<noshep;i++){
            sc.nextLine();
            System.out.println("Enter name, age, height(in
cm) and lifespan of german sheperd "+(i+1));
            tempname=sc.nextLine();
            tempage=sc.nextInt();
            sc.nextLine();
            tempheight=sc.nextFloat();
            templifespan=sc.nextInt();
            shep_arr[i]=new
germanSheperd(tempname,tempage,tempheight,templifespan);
        }
        System.out.println("The most popular breed is:
"+popularBreed());
        System.out.println("The life expectancy of labradors
and german sheperds respctively is: ");
        System.out.println(avgLifeExpectancy("Labrador")+"
and "+avgLifeExpectancy("German Sheperd"));
        System.out.println("The average weight of Labradors
if: "+Labrador.calculateAvgWeight());
        System.out.println("The average height of german
sheperds is: "+germanSheperd.calculateAvgHeight());

    }
}
```

(Result displayed on next page)
RESULT:

```
Enter total number of labradors and german sheperds:
2
2
Enter name, age, color and weight(in Kg) of Labrador 1
Lab#1
12
brown
14.5
Enter name, age, color and weight(in Kg) of Labrador 2
Lab#2
14
grey
16.9
Enter name, age, height(in cm) and lifespan of german sheperd 1
GS#1
9
56
22
Enter name, age, height(in cm) and lifespan of german sheperd 2
GS#2
16
67
25
The most popular breed is: Both breeds are equally popular
The life expectancy of labradors and german sheperds respctively is:
13.0 and 12.5
The average weight of Labradors if: 15.7
The average height of german sheperds is: 61.5
```