

Khan Aatif

**25**

**//**

**23**

**2022-2023**

Md taufik

**.**

**.**

**.**

Khan Saad

**2**

INDEX

|  |  |  |
| --- | --- | --- |
| SR NO. | TOPICS COVERED | PAGE NO. |
| 1. | ABSTRACT | 1 |
| 2. | INTRODUCTION | 2 |
| 3. | COMPONENTS USED program | 3 |
| 4. | PROGRAM | 4 |
| 5. | OUTPUT | 5 |
| 6. | USES | 6 |
| 7. | CONCLUSION | 7 |
| 8. | REFERENCE | 8 |

# ABSTRACT

Abstraction is a design level process and it is used to reduce the complexity at the designing stage of a project. Encapsulation is an implementation level process, and it is used to provide privacy and maintain control over the transparency of data at the implementation stage of a project.

Data Abstraction can be described as the technique of hiding internal details of a program and exposing the functionality only. Data Encapsulation can be described as the technique of binding up of data along with its correlate methods as a single unit. Implementation hiding is done using this technique.

Abstraction allows a programmer to design software better by thinking in general terms rather than specific terms while Polymorphism allows a programmer to defer choosing the code you want to execute at runtime.

1

# 

INTRODUCTION

Polymorphism literally means “having many forms”. In C++, both operators and functions can have many forms in that they can be adapted so that while the meaning of their operation is the same their implementation will allow for their use with different types of data.

Encapsulation is a way to restrict the direct access to some components of an object, so users cannot access state values for all of the variables of a particular object. Encapsulation can be used to hide both data members and data functions or methods associated with an instantiated class or object.

What is data binding? Data binding is the process that establishes a connection between the app UI and the data it displays. If the binding has the correct settings and the data provides the proper notifications, when the data changes its value, the elements that are bound to the data reflect changes automatically

2

# PROGRAM

**import java.io.\*;**

**class Animal {**

**private String name;**

**private int age;**

**public Animal(String name, int age) {**

**this.name = name;**

**this.age = age;**

**}**

**public String getName() {**

**return name;**

**}**

**public int getAge() {**

**return age;**

**}**

**public void eat() {**

**System.out.println("The animal is eating.");**

**}**

**}**

**class Dog extends Animal {**

**private String breed;**

**public Dog(String name, int age, String breed) {**

**super(name, age);**

**this.breed = breed;**

**}**

**public String getBreed() {**

**return breed;**

**}**

**public void bark() {**

**System.out.println("The dog is barking.");**

**}**

**@Override**

**public void eat() {**

**System.out.println("The dog is eating.");**

**}**

**}**

**public class Main {**

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

**Animal animal = new Animal("Animal", 5);**

**Dog dog = new Dog("Dog", 3, "Labrador");**

**// Polymorphism**

**Animal polyAnimal = new Dog("Poly Dog", 2, "Golden Retriever");**

**polyAnimal.eat();**

**// Encapsulation**

**System.out.println("Name: " + animal.getName() + ", Age: " + animal.getAge());**

**System.out.println("Name: " + dog.getName() + ", Age: " + dog.getAge() + ", Breed: " + dog.getBreed());**

**// Data binding**

**Animal dogAnimal = new Dog("Dog Animal", 1, "Poodle");**

**dogAnimal.eat();**

**// Inheritance**

**dog.bark();**

**}**

**}**

**OUTPUT**



# CONCLUSION

In this program, we have a base class `Animal` that contains private fields `name` and `age` and a method `eat()`. The `Dog` class extends `Animal` and adds a private field `breed` and a method `bark()`. The `eat()` method in `Dog` overrides the one in `Animal`.

In the `Main` class, we create an instance of `Animal` and `Dog` and demonstrate polymorphism by creating an `Animal` variable that references a `Dog` object. We also demonstrate encapsulation by using getter methods to access the private fields of `Animal` and `Dog`. Data binding is demonstrated by creating an `Animal` variable that references a `Dog` object and calling the `eat()` method. Finally, we demonstrate inheritance by calling the `bark()` method on a `Dog` object.

REFERENCE

1. [www.youtube.com](http://www.youtube.com)
2. [www.nerd.com](http://www.nerd.com)
3. www.grad.com