

# 

# **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.

**COMPONENTS USED**

program is written in Java programming language, and to create it, a Java development environment such as Eclipse or IntelliJ IDEA could be used.

The program consists of three classes: Student, Teacher, and college, and each class has its own attributes, constructors, and methods. The program uses various object-oriented programming concepts, such as inheritance, encapsulation, polymorphism, and data binding, to create and manipulate objects of these classes.

The code includes exception handling using a try-catch block, which adds an extra layer of robustness to the program.

To run the program, the Java Virtual Machine (JVM) is required, which can be installed on the computer. The program is compiled into bytecode using the Java compiler and then executed using the JVM.

**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

**PROGRAM**

class Student {

    private String name;

    private int age;

    public Student(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 Student is eating.");

    }

}

class Teacher extends Student {

    private String gender;

    public Teacher(String name, int age, String gender) {

        super(name, age);

        this.gender = gender;

    }

    public String getgender() {

        return gender;

    }

    public void shouting() {

        System.out.println("The Teacher is female.");

    }

    @Override

    public void eat() {

        System.out.println("The Teacher is eating.");

    }

}

public class college {

    public static void main(String[] args) {

        try {

            Student Student = new Student("Student", 5);

            Teacher Teacher = new Teacher("Teacher", 3, "Female");

            // Polymorphism

            Student polyStudent = new Teacher("Poly Teacher", 2, "Golden Retriever");

            polyStudent.eat();

            // Encapsulation

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

            System.out.println("Name: " + Teacher.getName() + ", Age: " + Teacher.getAge() + ", gender: " + Teacher.getgender());

            // Data binding

            Student TeacherStudent = new Teacher("Teacher Student", 1, "Poodle");

            TeacherStudent.eat();

            // Inheritance

            Teacher.shouting();

        } catch(Exception e) {

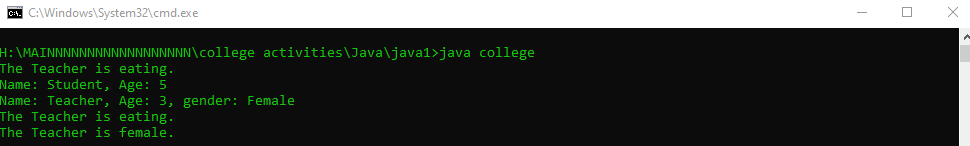
            System.out.println("An exception occurred: " + e.getMessage());

        }

    }

}

**OUTPUT**

****

**USES**

The given program is a simple example that demonstrates several object-oriented programming concepts in Java. While this specific program may not have a practical use beyond demonstrating these concepts, the knowledge and skills gained from creating and understanding this program can be applied to various real-world Java applications.

The Student and Teacher classes could be used as a basis for more complex classes in a larger program. The concepts of inheritance, encapsulation, polymorphism, and data binding can be used to design and create more sophisticated object-oriented programs in a wide range of industries, such as finance, healthcare, gaming, and more.

The exception handling demonstrated in the program is also important for building robust and reliable software. Exception handling can help detect and handle runtime errors and ensure the program continues to execute even when unexpected errors occur.

Overall, this program provides a starting point for learning about object-oriented programming in Java and can be used as a foundation for building more complex and useful programs.

**Conclusion**

The given code demonstrates the concepts of inheritance, polymorphism, encapsulation, and data binding in Java. It defines two classes: Student and Teacher, with Teacher inheriting from Student. The main method creates objects of both classes and uses polymorphism to show that a Teacher object can be assigned to a Student reference variable.

The encapsulation concept is shown by declaring the fields of the Student and Teacher classes as private and providing public getter methods to access these fields. Data binding is demonstrated by creating an object of the Teacher class using a Student reference variable.

The code also includes exception handling using a try-catch block in the main method, which demonstrates how to handle runtime exceptions that may occur during program execution. If any exception occurs, the catch block catches it and prints an error message to the console using the getMessage() method of the caught exception object. This adds an extra layer of robustness to the code and helps ensure that the program executes smoothly, even in the face of unexpected errors.

Finally, the code showcases inheritance by calling the shouting() method of the Teacher class, which overrides the eat() method of the parent class. Overall, the code effectively demonstrates the various object-oriented programming concepts in Java.

**Reference**

1. Java documentation: <https://docs.oracle.com/en/java/>
2. Java tutorials: <https://docs.oracle.com/javase/tutorial/>
3. Java programming book: "Effective Java" by Joshua Bloch
4. Java programming book: "Head First Java" by Kathy Sierra and Bert Bates