

lab-task# 1



sTUDENT: ahsan faraz

Reg : FA22-BSE-073

SECTION: 5-B

**Polymorphism in Software Engineering**

**Introduction**

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows methods to do different things based on the object that it is acting upon. It enhances the flexibility and maintainability of code by enabling a single interface to represent different underlying forms (data types).

**Definition of Polymorphism**

Polymorphism means "many shapes" in Greek. In programming, it refers to the ability of a single function, operator, or method to behave differently based on the context, specifically the data type or class that it operates on. There are two main types of polymorphism:

1. **Compile-Time Polymorphism (Static Polymorphism)**: Resolved during compile time, primarily achieved through method overloading and operator overloading.
2. **Run-Time Polymorphism (Dynamic Polymorphism)**: Resolved during runtime, typically achieved through method overriding and interfaces.

**Importance of Polymorphism**

1. **Code Reusability**: Polymorphism promotes code reuse by allowing the same interface to be used for different data types.
2. **Flexibility**: Code can be written to work on the base class type, allowing it to operate on any derived class type.
3. **Maintainability**: Reduces code complexity, making it easier to update and maintain code without altering existing functionality.
4. **Extensibility**: New classes can be added with minimal changes to existing code, allowing for easier feature expansion.

**Types of Polymorphism**

**1. Compile-Time Polymorphism**

* **Method Overloading**: Occurs when multiple methods in the same class have the same name but different parameters (type or number).

**Example**:

java

Copy code

class MathOperations {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

**2. Run-Time Polymorphism**

* **Method Overriding**: Happens when a subclass provides a specific implementation of a method that is already defined in its superclass.

**Example**:

java

Copy code

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

void sound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

void sound() {

System.out.println("Cat meows");

}

}

**Conclusion**

Polymorphism is a powerful feature of object-oriented programming that enhances flexibility and maintainability in software design. By allowing objects to be treated as instances of their parent class, developers can write more generic and reusable code. Understanding and implementing polymorphism effectively is crucial for creating robust and scalable applications.

**References**

* Meyer, B. (1997). *Object-Oriented Software Construction*. Prentice Hall.
* Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.