**Polymorphism**

This the feature that allow us to perform single action in different ways like ability of a message to be displayed in many ways.

Two types of Polymorphism: -

1. Compile time Polymorphism
2. Run time Polymorphism
3. **Compile time Polymorphism**-

Compile time polymorphism is resolved during the compilation process. It allows different methods by using the same name and distinguished by the different parameter lists. Compiler resolves which method called during the compile time based on their parameters provided and ensures that correct method is executed without any run overhead. Compile time polymorphism does not involve any decision-making during runtime. Which method will be executed is determined entirely during the compilation process. This is also called static or early binding.

**Example:**

Method Overloading :

class MathOperations

{

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

int add(int a, int b, int c) {

return a + b + c;

}

}

Constructor overloading:

public Student(String name, int age){

this.name = name;

this.age = age;

}

public Student(String age){

this.age=age;

}

1. **Runtime Polymorphism:**

The ability of a method to execute different behaviors based on the object that calls even when the method is called a reference of the superclass type. It allows the specific method to be determined at runtime. Runtime Polymorphism is achieved through method overloading, where a subclass provides a specific implementation of a method that is already defined in its implementation of a method that is already defined in its superclass.

Here the method call is resolved at runtime rather than at compile time. The JVM determines which method to execute based actual object type.

The objects are allocated on the heap, memory management involves keeping track of object reference and ensuring that the correct methods are called based on these references. The JVM uses a method table to manage method refences which allows for efficient method resolution.

**Method Overriding: -** Whenever writing method in super and sub class in such a way that method name and parameter must be same called method overriding.

\*\*Both method overloading and overriding is based on parameter type not on return type.

example:-

class A{

void show(){

System.out.println(“Shapes”);

}

}

Class B extends A{

void show(){

System.out.println(“Circle”);

}

}