**Polymorphism**

Polymorphism is the ability that allows objects of different types to be treated as objects of a common superclass or interface. It enables objects to be used in a flexible manner connecting a wide range of different types while maintaining a consistent interface. It is achieved through inheritance and the overriding method. When multiple classes are related through inheritance a subclass can inherit properties and behaviors from a superclass, polymorphism allows the subclasses to provide its own version, or implementation, of a method defined within the superclass. This means that objects and methods of the subclasses can be used interchangeably with objects and methods of the superclass. The idea behind polymorphism is determined by its runtime type rather than its declared type providing flexibility for code reuse and extensibility. There are two types of Polymorphism, the Static Polymorphism which is achieved through the overloading method, where multiple methods with the same name but different parameters are defined in a class, and Dynamic Polymorphism which is achieved through the overriding method, where a subclass provides its own implementation of a method defined within the superclass. Therefore, Polymorphism is essential in achieving code flexibility, modularity, and abstraction in object-oriented programming, allowing objects of different types to work together, and providing code reusability.

For example:

using System;

public abstract class Employee{

public Employee(string name) {

Name = name;

}

public string Name { get; set; }

public abstract decimal CalculateSalary();

}

public class SupportTI: Employee {

public SupportTI(string name):base(name) {

}

public override decimal CalculateSalary() { return 1000

} }

public class Developer : Employee {

public DeveloperI(string name):base(name) {

}

public override decimal CalculateSalary() {

return 3000;

}

Class Program{

Static void Main(string[] args){

List<Employee> employees = new List<Employee>();

Employees.add(new SupportTI(“John”));

Employees.add(new Developer(“Petter”));

foreach(Employee employee in employees){

float pay = employee.CalculateSalary();

Console.WriteLine($“{employee.name}-{pay}”);

}

}

}

The example above exemplifies the use of Polymorphism where the superclass Employees has a defined method named CalculateSalary() but it is overridden when the method is run within the sub-classes, allowing the objects of different types to be treated as of a common super class and providing flexibility and code reuse in handling different employee types.