Polymorphism in Object-Oriented Programming

The Third Pillar

Polymorphism is a pivotal concept in object-oriented programming (OOP) that brings flexibility and adaptability to the code. At its core, polymorphism allows different objects or classes to respond to the same method or message in a variety of ways. This capability is rooted in the idea that objects can take on different forms, depending on their specific attributes and implementations.

In other words, polymorphism enables objects of different types to be treated as objects of a common base type. E.g., Bears & Lions are both Derived Classes of the Bass Class Animal. A more Detailed Example will be discussed below.

Inheritance The Foundation of Polymorphism

To understand polymorphism fully, we must first grasp the concept of inheritance. Inheritance allows us to create new classes that inherit features and attributes from a superclass or base class. This mechanism facilitates code reuse and the organization of related objects into hierarchical structures.

Now, let's delve into our **Pizza** analogy:

Imagine a superclass called "Pizza." This superclass defines fundamental properties and methods common to all pizzas, such as the crust type, size, and cooking time. From this "Pizza" class, we can create specific pizza types like Margherita, Pepperoni, and Veggie. These specialized pizza classes inherit the basic attributes and behaviours from the "Pizza" superclass.

Polymorphism and Pizza: Ordering with Flexibility

Here's where polymorphism comes into play. When you order a "Pizza," you don't specify the exact type (e.g., Margherita or Pepperoni) the actual pizza you receive is determined by the specific subclass you instantiate. In this way, polymorphism allows you to order a pizza without worrying about the details of its preparation.

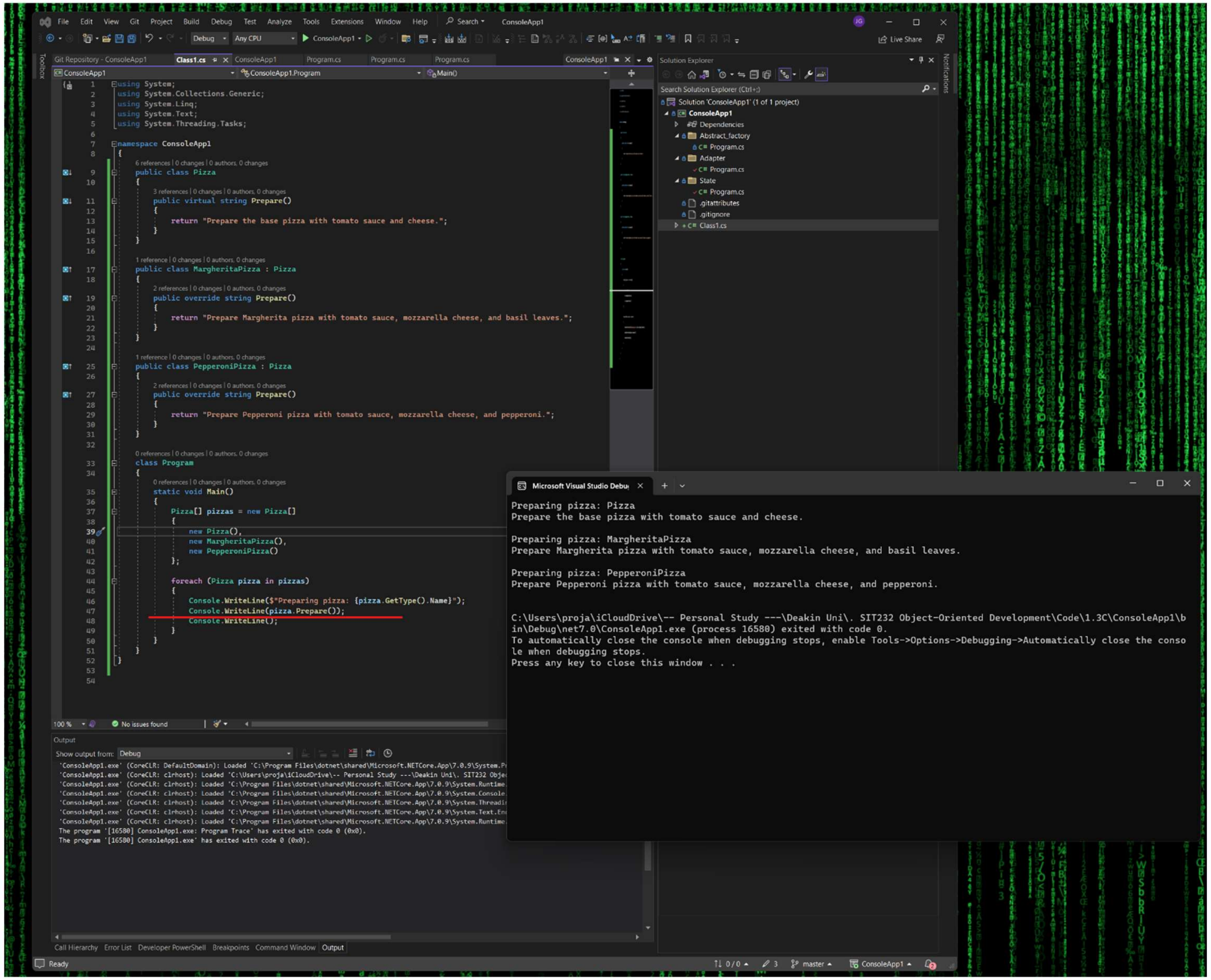
Now, let's consider the preparation process. When you order a "Pizza," the kitchen staff doesn't need to know which exact type of pizza you want; they just know it's a "Pizza." The kitchen staff will use the same "Prepare" method for all pizza orders. However, the outcome of this method varies depending on the specific pizza class being instantiated.

Let me break that down.

If you order a Margherita pizza, the "Prepare" method for Margherita pizzas will be executed, which includes adding tomato sauce, mozzarella cheese, and basil leaves to the pizza. If you order a Pepperoni pizza, the "Prepare" method for Pepperoni pizzas will be invoked, which involves adding tomato sauce, mozzarella cheese, and slices of pepperoni.

This demonstrates the essence of polymorphism in action. It allows us to call the **same method**, "Prepare," on objects of different pizza types, and each object responds in its unique way based on its class-specific implementation.

In conclusion, polymorphism in object-oriented programming, illustrated through our pizza analogy, is a powerful concept that enhances code reusability, flexibility, and organization. It enables us to work with objects at a higher level of abstraction, invoking methods without concerning ourselves with the specific type of object we're dealing with, making our code more adaptable and efficient. Just as you can order a "Pizza" without specifying the exact type, polymorphism allows software developers to design systems that gracefully handle diverse objects within a common framework, promoting cleaner and more maintainable code



```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    public class Pizza
    {
        public virtual string Prepare()
        {
            return "Prepare the base pizza with tomato sauce and cheese.";
        }
    }

    public class MargheritaPizza : Pizza
    {
        public override string Prepare()
        {
            return "Prepare Margherita pizza with tomato sauce, mozzarella cheese, and basil leaves.";
        }
    }

    public class PepperoniPizza : Pizza
    {
        public override string Prepare()
        {
            return "Prepare Pepperoni pizza with tomato sauce, mozzarella cheese, and pepperoni.";
        }
    }

    class Program
    {
        static void Main()
        {
            Pizza[] pizzas = new Pizza[]
            {
                new Pizza(),
                new MargheritaPizza(),
                new PepperoniPizza()
            };

            foreach (Pizza pizza in pizzas)
            {
                Console.WriteLine($"Preparing pizza: {pizza.GetType().Name}");
                Console.WriteLine(pizza.Prepare());
                Console.WriteLine();
            }
        }
    }
}
```

Microsoft Visual Studio Debu

```
Preparing pizza: Pizza
Prepare the base pizza with tomato sauce and cheese.

Preparing pizza: MargheritaPizza
Prepare Margherita pizza with tomato sauce, mozzarella cheese, and basil leaves.

Preparing pizza: PepperoniPizza
Prepare Pepperoni pizza with tomato sauce, mozzarella cheese, and pepperoni.


C:\Users\proja\iCloudDrive\-- Personal Study ---\Deakin Uni\. SIT232 Object-Oriented Development\Code\1.3C\ConsoleApp1\b
in\Debug\net7.0\ConsoleApp1.exe (process 16580) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
'ConsoleApp1.exe' (CoreCLR: DefaultDomain): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\7.0.9\System.Pr
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Users\proja\iCloudDrive\-- Personal Study ---\Deakin Uni\. SIT232 Objec
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\7.0.9\System.Runtime.
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\7.0.9\System.Console.
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\7.0.9\System.Threadi
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\7.0.9\System.Text.En
'ConsoleApp1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\7.0.9\System.Runtime.
The program '[16580] ConsoleApp1.exe: Program Trace' has exited with code 0 (0x0).
The program '[16580] ConsoleApp1.exe' has exited with code 0 (0x0).
```

1. We define a base class Pizza with a virtual Prepare method that returns a generic description for preparing a base pizza.

2. We create two derived classes, MargheritaPizza and PepperoniPizza, which inherit from the base class Pizza. Each of these derived classes overrides the Prepare method to provide its own unique description of pizza preparation.

3. In the Main method, we simulate an order with an array of pizzas (order). This order includes one generic pizza, one Margherita pizza, and one Pepperoni pizza.

4. We loop through the order and call the Prepare method on each pizza. Depending on the pizza type (polymorphism!!) the corresponding overridden Prepare method is called, providing a specific description of how that pizza is prepared.