SIT232: Object Oriented Development

Abstraction: simplifying complexities

What is Abstraction?

Abstraction is a fundamental principle in OOP that focuses on representing essential features while hiding unnecessary details. It involves creating abstract classes and interfaces that defines a common set of behaviors or properties shared among related objects. It allows us to create generalized models that represents real-world entities, making the code more flexible, reusable and maintainable.

Benefits:

- **Simplifies complex system:** Abstraction helps break down complex systems into manageable, modular components, making them easier to understand and work with. For example:

```
public abstract class Vehicle
{
 protected string _make;
 protected string _model;
 protected int _year;

 public Vehicle(string make, string model, int year)
 {
     _make = make;
     _model = model;
     _year = year;
 }

 public abstract void Start();

 public abstract void Stop();
 public void DisplayDetails()
 {
     Console.WriteLine("Vehicle Details:");
     Console.WriteLine("Make: " + _make);
     Console.WriteLine("Model: " + _model);
     Console.WriteLine("Year: " + _year);
 }
}
```

In the above code the Vehicle abstract class provides a common interface for all types of vehicles. It abstracts the common properties like _make, _model, and _year, simplifying the representation of vehicles in the system. This abstraction allows us to handle different vehicle types using a unified interface, reducing the complexity of managing each vehicle type separately.

- **Focuses on essential characteristics:** By abstracting away implementation details, abstraction allows us to focus on the essential characteristics and functionalities of an object. For example: In the above code, he Vehicle abstract class defines the essential characteristics shared by all vehicles, such as the make, model, and year. By abstracting Start() and Stop() operations, the code focuses on the vital actions that are common to all vehicles, irrespective of their specific types.

- **Encourages code reusability:** Abstract classes and interfaces provide a blueprint for related objects, enabling code reusability and reducing redundant implementation.
- **Promotes loose coupling:** Abstraction promotes loose coupling between components, as objects interact based on their common characteristics rather than specific implementations.

Objectives:

- **Simplification:** Abstraction aims to simplify complex systems by focusing on the essential aspects and hiding unnecessary details. It allows developers to work with higher-level concepts and models, reducing the cognitive load and making the code easier to understand and maintain.
- **Encapsulation:** Abstraction works hand in hand with encapsulation, another key principle of OOP. It allows for encapsulating the internal state and implementation details within a class, while providing a simplified, abstracted interface to interact with the class. This helps in maintaining data integrity, enforcing access restrictions, and promoting code organization and maintainability.
- **Polymorphism**: Abstraction enables polymorphism, a powerful feature of OOP. Polymorphism allows objects of different concrete classes that implement the same abstract interface or inherit from the same base class to be treated interchangeably. This provides flexibility in designing and working with diverse objects through a unified interface, promoting code flexibility and maintainability. Like in the give code

```
public abstract class Vehicle
{
    // ...

    public abstract void Start();
    public abstract void Stop();
}

public class Car : Vehicle
{
    // ...

    public override void Start()
    {
        Console.WriteLine("Car engine started.");
    }

    public override void Stop()
    {
        Console.WriteLine("Car engine stopped.");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Vehicle myCar = new Car("Toyota", "Camry", 2020);

        myCar.Start(); // calling polymorphic method
        myCar.Stop();  // “”
    }
}
```

The derived class overrides these abstract methods with its own implementation that is appropriate for the specific vehicle type. This allows for polymorphic behavior, where the same method name (Start() or Stop()) can exhibit different behaviors based on the actual type of the object.

Requirements for this:

You need to learn about the basics of C# and understanding of polymorphism and encapsulation is required
Link for my encapsulation notes:
https://github.com/17012004/Encapsulation

Summarize what you learned:

Abstraction is a fundamental concept in programming that involves simplifying complex systems by focusing on essential features while hiding unnecessary details. It allows us to create models or representations of real-world entities or concepts in a simplified and generalized manner. Abstraction involves identifying and capturing the essential characteristics, properties, and behaviors of an object or system, while omitting or abstracting away the implementation details that are not relevant to the higher-level perspective. It provides a high-level view or interface that allows users to interact with the system without needing to understand the underlying complexities. Abstraction promotes modularity, reusability, and maintainability by separating the concerns and allowing components to be developed and updated independently.

Link for my code for understanding of Abstraction:
https://github.com/17012004/Abstraction/blob/main/Program.cs

Please give your feedback and any query on:
mailto:vansh4856.be22@chitkara.edu.in

and add CC/BCC as:
mailto:harsimran.kaur@chitkara.edu.in