What is class ?

​​A class in programming is a blueprint or a template for creating objects. Classes define a set of properties and methods (functions) that describe the behaviour and characteristics of objects created from that class. Objects created from a class are called instances of that class.

Classes provide a way to encapsulate data and behaviour into a single, reusable unit, and to create objects that have a specific type. This allows you to write code that is more modular and easier to maintain, as you can define the properties and methods of objects once, and then reuse that code in multiple places throughout your application.  For example, you could define a Person class to represent people, with properties such as name, age, and address, and methods such as getName() and getAge() to retrieve the values of these properties.

You could then create multiple instances of the Person class, each representing a different person, and use the properties and methods of these instances to work with the data for each person.  Classes are a fundamental concept in object-oriented programming (OOP), and are used in many programming languages, including Java, C#, Python, and TypeScript. They provide a way to structure and organise code, and to create objects that have a well-defined and reusable interface.

What is an abstract class ?

An abstract class is a class in object-oriented programming that cannot be instantiated on its own. An abstract class serves as a base class or a blueprint for other classes, providing common functionality that can be inherited and overridden by subclasses.  An abstract class can contain both abstract methods, which are methods without a body or implementation, and concrete methods, which have a body and can be called just like regular methods.

Abstract methods force subclasses to provide their own implementation, while concrete methods can be used as is or overridden by subclasses if needed.  Abstract classes are useful when you want to define a common base class that provides some default behavior, but you don't want to create instances of that class directly. Instead, you define subclasses that inherit from the abstract class and provide their own implementation for the abstract methods.

 This allows you to create objects that have a specific type and behaviour, while still maintaining the flexibility to extend or modify that behaviour as needed.  For example, you could define an abstract Shape class with an abstract method area() to calculate the area of a shape, and concrete methods such as getName() to return the name of the shape. You could then define subclasses such as Circle, Rectangle, and Triangle, each of which would inherit from the Shape class and provide its own implementation of the area() method.

What is the interface ?

An interface in object-oriented programming is a contract or a set of rules that defines the properties, methods, and events that an object should have. An interface specifies what an object should do, but not how it should do it. Interfaces provide a way to standardise the behaviour of objects, and to define a common set of properties, methods, and events that objects of a certain type should have.

For example, you could define an interface Shape with properties such as name and methods such as area() to calculate the area of a shape. You could then create classes such as Circle, Rectangle, and Triangle that implement the Shape interface, and provide their own implementation of the area() method. This allows you to create objects that have a specific type, such as Shape, and to ensure that these objects have the properties and methods specified by the interface.

Example of the difference between class and abstract class :

// Class example

class Shape {

  private String name;

  public Shape(String name) {

    this.name = name;

  }

  public String getName() {

    return this.name;

  }

  public double area() {

    return 0.0;

  }

}

class Circle extends Shape {

  private double radius;

  public Circle(double radius) {

    super("Circle");

    this.radius = radius;

  }

  @Override

  public double area() {

    return Math.PI \* this.radius \* this.radius;

  }

}

// Abstract class example

abstract class AbstractShape {

Example of the difference between interface and abstract class:

An interface is a blueprint for classes that defines a set of methods, properties, and events, but does not provide implementation.

An abstract class is a base class that can have methods with or without implementation and can also have properties. However, it cannot be instantiated on its own and must be subclassed in order to be used.

Example :

interface IShape {

    float CalculateArea();

}

abstract class Shape {

    public abstract float CalculateArea();

}

class Square : Shape {

    float side;

    public Square(float side) {

        this.side = side;

    }

    public override float CalculateArea() {

        return side \* side;

    }

}

In this example, IShape is an interface that defines the method CalculateArea, while Shape is an abstract class that provides a basic implementation of the same method. The Square class extends Shape and overrides the implementation of the CalculateArea method to provide its own implementation.