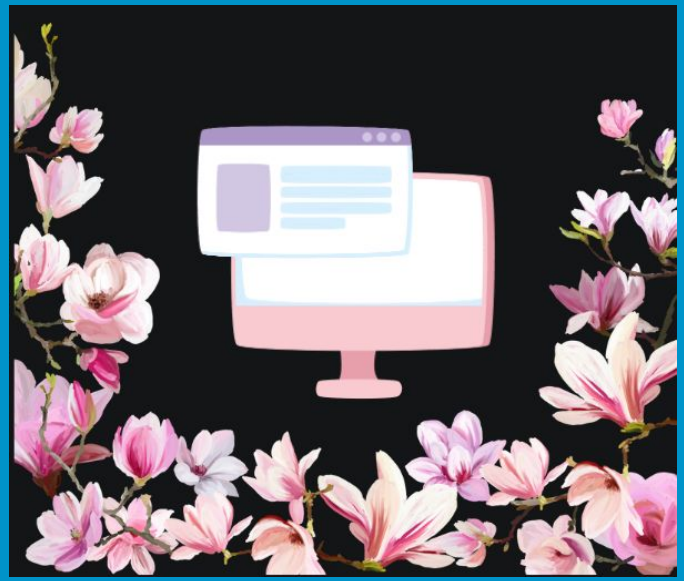
Abstract Classes



OOPs A PIE

Stands for:

A - abstraction

P - polymorphism

I - inheritance

E - encapsulation

Polymorphism

Stands for:
One thing having many forms

Abstract allows us to decide what parts we want to inherit.

Customize!

Partial Template

Hybrid Template

We can give a class the **abstract modifier keyword**. **This will provide a partial implementation of our base class for our child class to inherit from.**

modifier keyword → abstract

modifier keyword → abstract

```
4 references
abstract class Shape
{

}
```

Static Polymorphism

- Method overloading
- For example: The Console.WriteLine() method has 18 overloads.
- While we are writing our code or **at compile-time**, our methods are taking on multiple forms with the same name.

```
Console.WriteLine("string");
Console.WriteLine(7);
Console.WriteLine(true);
```

Dynamic Polymorphism

- Abstract classes
- We are telling our code that **at run-time** objects of a derived class may be treated as objects of a base class.

Dynamic Polymorphism

```
Shape shape1;

Console.WriteLine("Pick out a shape you would like: Circle, Square, or Triangle");
var userInput = Console.ReadLine();

if (userInput == "circle")
{
    shape1 = new Circle();
}
else if (userInput == "square")
{
    shape1 = new Square();
}
else
{
    shape1 = new Triangle();
}
```

Class marked with abstract keyword

```
4 references
abstract class Shape
{

}
```

3 Abstract Class Features

1. An abstract class cannot be instantiated.

```
Shape shape1 = new Shape();
```

Abstract Class Features

2. An abstract class may contain abstract methods and accessors.

```
4 references
abstract class Shape
{
    4 references
    public abstract int NumberOfSides { get; set; }
    1 reference
    public void GetArea()
    {
        Console.WriteLine("Calculates area");
    }
}
```

Abstract Class Features

3. A **non-abstract class** derived from an abstract class must include actual implementations of all inherited abstract methods and accessors.

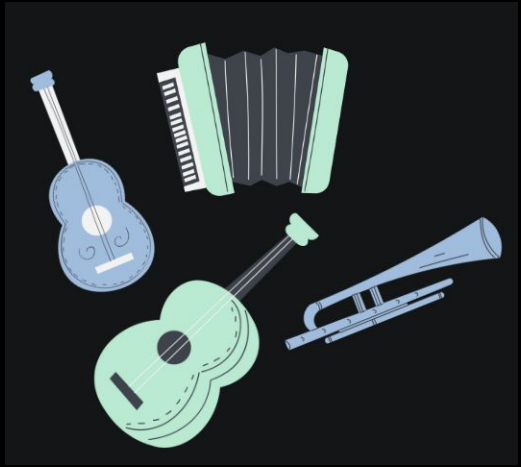
```
5 references
abstract class Shape
{
    4 references
    public abstract int NumberOfSides { get; set; }
    1 reference
    public abstract void GetArea();
}
```

Abstract Class Features

```
5 references
abstract class Shape
{
    4 references
    public abstract int NumberOfSides { get; set; }
    1 reference
    public abstract void GetArea();
}
```

Purpose

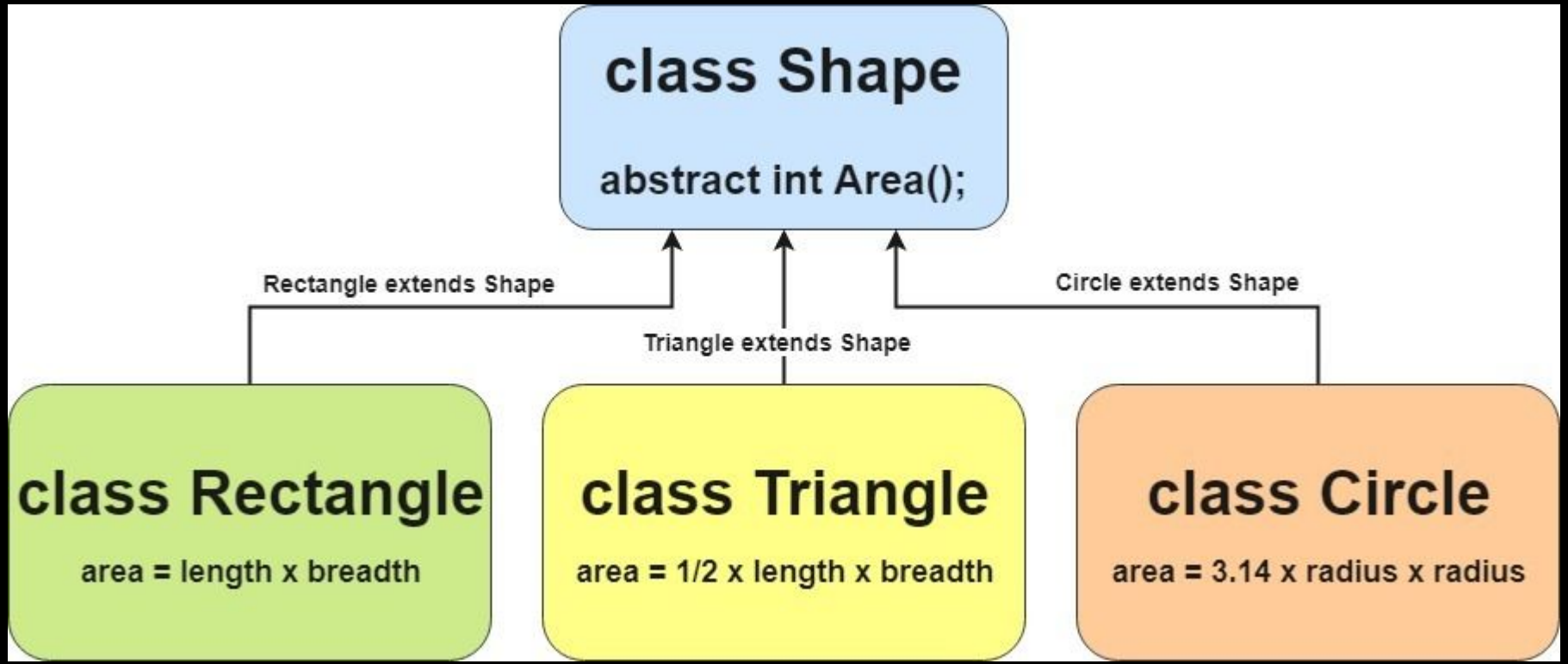
- The purpose of an abstract class is to define some common behavior that can be inherited by multiple subclasses, without implementing the entire class.



Abstract Classes - Partial Template

- An abstract class serves as a base class for other classes.
- Classes derived from an abstract class must provide an implementation for all its abstract methods.

Abstract Class



Abstract Classes

- **Can have abstract or virtual members**

Abstract Classes

- **Abstract member: You MUST implement member in derived classes**
- **Virtual member: OPTIONAL to override**

Virtual keyword

- Method provides a default implementation
- *Option* to override the default implementation

Abstract Classes - 3 Keywords

1. **Abstract**
2. **Virtual**
3. **Override**

Override keyword

- Used in the derived class
- Used to provide implementation for abstract OR virtual member declared in the base class

Abstract Class Example

```
//abstract base class
4 references
internal abstract class Customer
{

    3 references
    public abstract void PrintCustomerName(); //stubbed out method



}
```

Inheriting from Abstract Class

```
1 reference
internal class BronzeTierCustomer : Customer
{
    1 reference
    public override void PrintCustomerName()
    {
        Console.WriteLine("Print Bronze Tier Customer's name... ");
    }
}
```

Inheriting from Abstract Class

```
1 reference
internal class GoldTierCustomer : Customer
{
    // must have override keyword to implement the abstract properties/methods

    1 reference
    public override void PrintCustomerName()
    {
        Console.WriteLine("Print Gold Tier Customer's name... ");
    }
}
```

Inheriting from Abstract Class

```
1 reference
internal class SilverTierCustomer : Customer
{
    1 reference
    public override void PrintCustomerName()
    {
        Console.WriteLine("Print Silver Tier Customer's name... ");
    }

}
```

Virtual - Optional

```
//abstract base class
4 references
internal abstract class Customer
{

    //virtual keyword will allow you to override IF YOU WANT TO (optional)
    0 references
    public virtual void GreetCustomer()
    {
        Console.WriteLine("Hello, Guest");
    }



}
```

Virtual - Optional

```
1 reference
internal class GoldTierCustomer : Customer
{
    1 reference
    public override void GreetCustomer()
    {
        Console.WriteLine("Hello, Gold Guest");
    }
}
```

Virtual - Optional

```
1 reference
internal class SilverTierCustomer : Customer
{
    1 reference
    public override void GreetCustomer()
    {
        Console.WriteLine("Hello, Silver Guest");
    }

}
```

Virtual - Optional

```
1 reference
internal class BronzeTierCustomer : Customer
{
    //will use GreetCustomer Method from the Abstract Customer Class
}
```

The Bronze Customer got the default message

```
//abstract base class
4 references
internal abstract class Customer
{

    //virtual keyword will allow you to override IF YOU WANT TO (optional)
    0 references
    public virtual void GreetCustomer()
    {
        Console.WriteLine("Hello, Guest");
    }



}
```