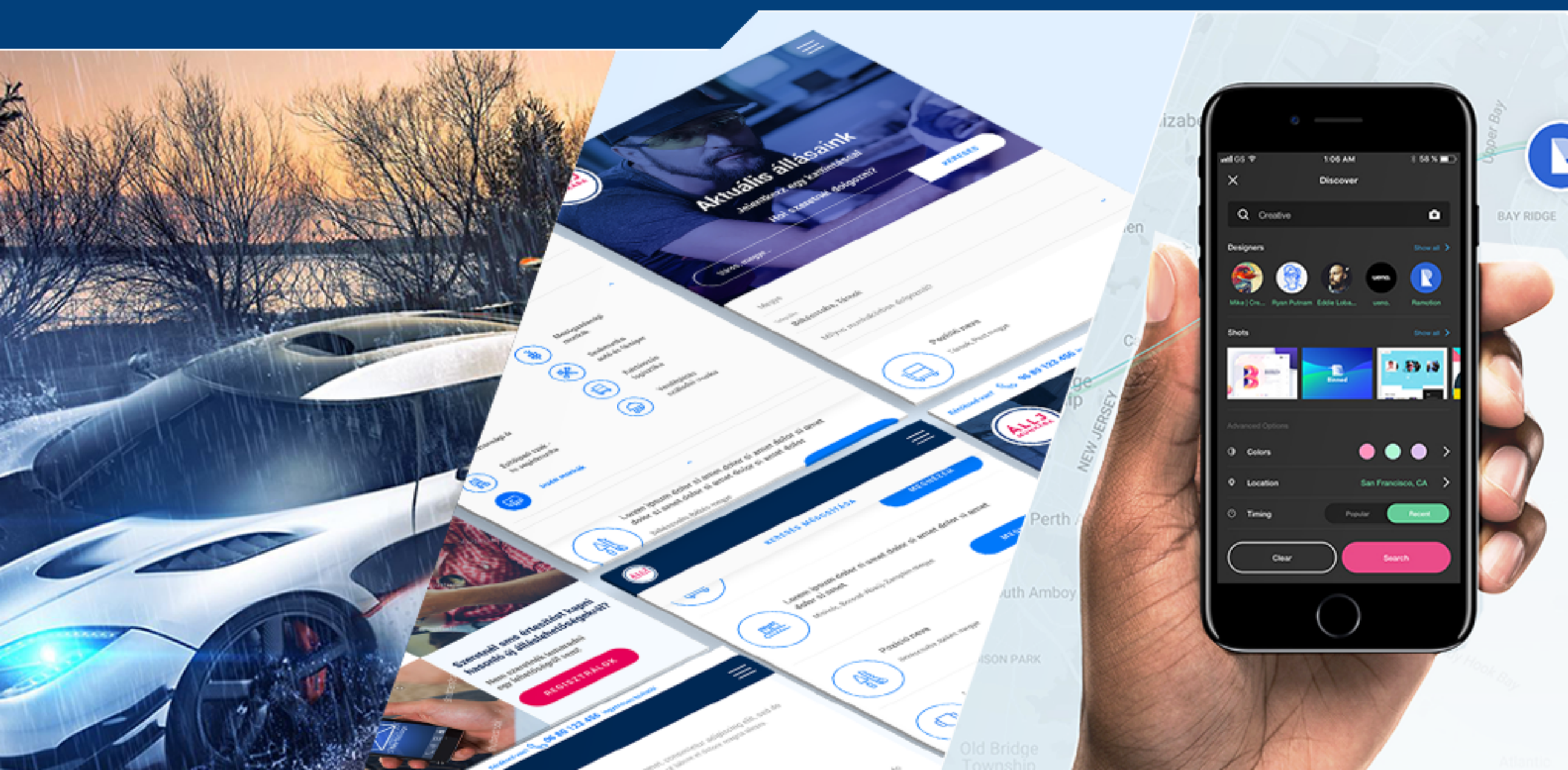
VT Academy
Thinking Ahead, Beyond Boundaries

Object Oriented Programming



Abstract Classe and Interface

Session 6

Objectives

VTC Academy
Thinking Ahead, Beyond Boundaries

- Abstract Classes
- Interfaces

Abstract Classes

- An abstract class can be referred to as incomplete base class
- It’s a class cannot be instantiated
- It can only be sub-classed
- An abstract class can implement methods that are similar for all the subclasses
- It can declare methods that are different for the different subclasses without implementing them
- Such methods are referred to as abstract methods

Abstract Classes Implementation

- An abstract class can be implemented in a way similar to implementing a normal base class.
- The subclass inheriting the abstract class has to override and implement the abstract methods.
- Subclass can implement the methods implemented in the abstract class with the same name and arguments.
- If the subclass fails to implement the abstract methods, the subclass cannot be instantiated as the C# compiler considers it as abstract.

Abstract Classes Example


```
// abstract class
abstract class ShapesClass
{
    abstract public int Area();
}
class Square : ShapesClass
{
    int side = 0;

    public Square(int n)
    {
        side = n;
    }
    // Area method is required to avoid a compile-time error
    public override int Area()
    {
        return side * side;
    }

    static void Main()
    {
        Square sq = new Square(12);
        Console.WriteLine("Area of the square = {0}", sq.Area());
    }
}
```

Abstract Classes Features

- An abstract class cannot be instantiated
- An abstract class may contain abstract methods and accessors
- It is not possible to modify an abstract class with the sealed modifier because the two modifers have opposite meanings. The **sealed** modifier prevents a class from being inherited and the **abstract** modifier requires a class to be inherited.
- A non-abstract class derived from an abstract class must include actual implementations of all inherited abstract methods and accessors

Interfaces

- Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members.
- An interface is a pure abstract base class
- It contains only abstract methods and no method implementation
- A class that implements a particular interface must implement the members listed by that interface

Declaring Interfaces

- Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default.
- Following is an example of an interface declaration:

```
public interface ITransactions
{
    // interface members
    void showTransaction();
    double getAmount();
}
```

Interfaces Inheritance Example

VTC Academy
Thinking Ahead, Beyond Boundaries

```
public interface ITransactions
{
   // interface members
   void showTransaction();
   double getAmount();
}

public class Transaction : ITransactions
{
   private string tCode;
   private string date;
   private double amount;
   public Transaction()
   {
      tCode = " ";
      date = " ";
      amount = 0.0;
   }
   public Transaction(string c, string d, double a)
   {
      tCode = c;
      date = d;
      amount = a;
   }
   public double GetAmount()
   {
      return amount;
   }
   public void showTransaction()
   {
      Console.WriteLine("Transaction: {0}", tCode);
      Console.WriteLine("Date: {0}", date);
      Console.WriteLine("Amount: {0}", getAmount());
   }
}
```

Multiple Inheritance

- C# allows multiple interface implementation using interfaces
- Example:

```
interface IControl
{
    void Paint();
}
interface ISurface
{
    void Draw();
}
class SampleClass : IControl, ISurface
{
    public void Paint()
    {
        Console.WriteLine("Paint method in SampleClass");
    }
    public void Draw()
    {
        Console.WriteLine("Draw method in SampleClass");
    }
}
```

Explicit Interface Implementation

- If a class implements two interfaces that contain a member with the same signature, then implementing that member on the class will cause both interfaces to use that member as their implementation.

- Example:

```
// Declare the English units interface:
interface IEnglishDimensions
{
    float Length();
    float Width();
}

// Declare the metric units interface:
interface IMetricDimensions
{
    float Length();
    float Width();
}
```

Multiple Inheritance

```
class Box : IEnglishDimensions, IMetricDimensions {
    float lengthInches;
    float widthInches;
    public Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }
    // Explicitly implement the members of interface IEnglishDimensions
    float IEnglishDimensions.Length()
    {
        return lengthInches;
    }
    float IEnglishDimensions.Width()
    {
        return widthInches;
    }
    // Explicitly implement the members of interface IMetricDimensions
    float IMetricDimensions.Length()
    {
        return lengthInches * 2.54f;
    }
    float IMetricDimensions.Width()
    {
        return widthInches * 2.54f;
    }
}
```

Interfaces Inheritance

- New interface can be created by combining together other interfaces
- The syntax for this is similar to that used for inheritance, except that more than one interface can be merged to form a single interface
- Example:

```
interface IParentInterface1
{
    void ParentInterface1Method();
}
interface IParentInterface2
{
    void ParentInterface2Method();
}

interface IMyInterface : IParentInterface1, IParentInterface1
{
    void MethodToImplement();
}
```

Abstract Classes vs Interfaces

VTC Academy
Thinking Ahead, Beyond Boundaries

Feature	**Interface**	**Abstract class**
Multiple inheritance	A class may inherit several interfaces.	A class may inherit only one abstract class.
Default implementation	An interface cannot provide any code, just the signature.	An abstract class can provide complete, default code and/or just the details that have to be overridden.
Access Modfiers	An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Core VS Peripheral	Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface.	An abstract class defines the core identity of a class and there it is used for objects of the same type.
Speed	Requires more time to find the actual method in the corresponding classes.	Fast
Fields and Constants	No fields can be defined in interfaces	An abstract class can have fields and constrants defined

Summary

- An abstract class can be referred to as incomplete base class
- An abstract class may contain abstract methods and accessors
- Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members.
- One class can inherit more than one interface