GRAND CIRCUS

C# .NET BOOTCAMP

OBJECT ORIENTED PROGRAMMING

OBJECT ORIENTED PROGRAMMING

Explain some of the foundational concepts that we've covered already. How do they interact?

OBJECT ORIENTED CONCEPTS

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

OBJECT ORIENTED CONCEPTS

ABSTRACTION

- Hide the details and only show functionality.
- Example: When you do a phone call, you don't need to know the internal process of how the wires carry your voice.

OBJECT ORIENTED CONCEPTS

ENCAPSULATION

- Wrapping related functionality and data together as one unit.
- Example: Using classes to define an employee.

OBJECT ORIENTED CONCEPTS

INHERITANCE

- A class acquiring properties and behaviors of parent class.
- Provides code reusability, and used to achieve the forth concept, Polymorphism

OBJECT ORIENTED CONCEPTS

POLYMORPHISM

- Same task can be done in different ways depending on the object's type.
- Polymorphism is also the ability of an object to take on many forms. Example: An Employee is a Person, A Dog is an Animal

OBJECT ORIENTED CONCEPTS

POLYMORPHISM

- Same task can be done in different ways depending on the object's type.
- Polymorphism is also the ability of an object to take on many forms. Example: An Employee is a Person, A Dog is an Animal

CLASS VS. OBJECTS

- A class is a template or a blue print that describes the supported behavior and state of all objects of its type.
- An object is an instance of a class.

INSTANCE VARIABLES

An instance variable may be a value data type, an object created from a class such as the String class, or an object created from a user-defined class such as the Product class.

Created when the object is created, in contrast to static variables.

INSTANCE VARIABLES

To prevent other classes from accessing instance variable, use the private keyword to declare the as private.

We can declare the instance variable for a class anywhere outside the constructors and methods of the class.

SYNTAX

```
rate|protected Type variableName;
```

EXAMPLES

```
)le price;
: quantity;
:ing code;
)duct product;
```

EXAMPLE IN CLASS

```
s Product

 code instance variables here
ing code;
ing description;
ble price;
ructors and methods of the class …
ible to code instance variables here
 test;
```

CONSTRUCTORS

WHAT ARE CONSTRUCTORS?

- A constructor is a method that is used to initialize the state of an object.
- A constructor must use the same name and capitalization as the name of the class.

CONSTRUCTORS

WHAT ARE CONSTRUCTORS?

- In the absence of a constructor C# will create a default constructor that initializes all numeric types to 0, all boolean types to false, and all objects to null.
- The parameter list of the constructor forms the signature of the constructor. Each constructor must have a unique signature.

SYNTAX

```
sName([parameterList])

ements of the constructor
```

EXAMPLES

```
luct(String code, String description, double price)

)de;
.on = description;
)rice;
```

THE THIS KEYWORD

Can be used to refer to instance methods and data defined inside the class.

Since C# implicitly uses the this keyword for instance variables and methods, we don't need to explicitly code it unless a parameter has the same name as an instance variable.

EXAMPLE

```
l setX(int x)
```

RECAP

What you should know at this point:

- What is OOP, and what are the concepts of OOP
- What are classes and objects
- Difference between classes and objects
- How to create classes and objects
- Know how to define instance fields
- Define and use constructors
- Using this keyword

INHERITANCE AND POLYMORPHISM

WHAT IS INHERITANCE?

Inheritance allows us to create new classes based on existing ones.

A new class (subclass, derived class, child class) inherits the fields, constructors, and methods of the class it is based on (superclass, base class, parent class).

EXTENDING CLASSES

A subclass can extend the superclass by adding new fields, constructors, and methods. It can also override a method from the superclass with its own version of the method.

Classes in C# can derive from only a single direct base class (Single Inheritance)

USING INHERITANCE

Create generic superclasses that implement common elements of related subclasses.

Create classes that inherit from classes that are defined by the .NET API.

ACCESS MODIFIERS

ACCESS MODIFIERS SPECIFY THE ACCESSIBILITY OF THE MEMBERS DECLARED BY A CLASS.

- Public: You can access from everywhere.
- Protected: Access is limited to within the class and any class that inherits from the class
- Private: Access is only limited to within the class

ACCESS MODIFIERS

- Internal: Access is limited exclusively to classes defined within the current project assembly
- protected internal: Access is limited to the current assembly and types derived from the containing class.

CREATING A SUBCLASS

SYNTAX

```
)lic class SubclassName:SuperclassName
```

CREATING A SUBCLASS

We can override methods coming from the parent class if they are defined as virtual.

The override keyword gives the programmer a chance to change the behaviour of methods passed from the parent class.

We can use the "base" keyword to call a constructor or method of the parent class.

CREATING A SUBCLASS

EXAMPLE ON OVERRIDE

```
ıss Person
  {  protected string Name;
      public virtual void PrintName()
      {
      Console.WriteLine(Name);
      }
  }
```

CREATING A SUBCLASS

EXAMPLE ON OVERRIDE

```
Employee : Person
  {   private string Title;
      public override void PrintName()
          {
              Console.WriteLine(Title +" " + Name);
          }
  }
```

POLYMORPHISM

WHAT IS POLYMORPHISM?

Polymorphism is a feature of inheritance that allows us to treat object of different subclasses that are derived from the same superclass as if they had the type of the superclass.

POLYMORPHISM

EXAMPLE ON POLYMORPHISM

Using the Person and the Employee class

```
rson p = new Employee();
```

SEALED CLASSES

We can prevent a class from being inherited by using the sealed keyword.

```
ıled class MyClass

this class cannot be extended!
```

RECAP

What you should know at this point:

- What is Inheritance.
- Importance of Inheritance.
- How Inheritance works.
- What is polymorphism.
- Why we need polymorphism.
- Using the sealed keyword.

INTERFACES AND ABSTRACT CLASSES

ABSTRACT CLASSES

- An abstract class can be inherited by other classes, but not used to create an object.
- An abstract method is also created using the abstract keyword. Abstract methods have no body. They cannot have private access.
- An abstract class may not contain abstract methods, but any class that does contain abstract methods must be declared as abstract.

ABSTRACT CLASSES

Example

```
.ass Shape

:ed string Name;
 virtual double PrintName()


.teLine(Name);

 abstract double GetArea();
```

INTERFACES

An interface is a special type of coding element that provides many of the advantages of multiple inheritance. An interface defines a set of public methods that can be implemented by a class.

A class that implements an interface must provide an implementation for each method defined by the interface.

INTERFACES VS ABSTRACT CLASSES

Advantages of an abstract class

- Can use instance as well as static variables and constants
- Can define regular methods that contain code (concrete), and abstract methods
- Can define fields

INTERFACES VS ABSTRACT CLASSES

Advantages of an Interface

- A class can directly implement multiple interfaces
- Any object created from a class that implements an interface can be used wherever the interface is accepted

INTERFACES

Example

```
:Shape

etArea();
```

HOW TO WORK WITH INTERFACES

- An interface can inherit one or more interfaces
- An interface can't inherit a class
- A class that implements an interface must implement all the methods declared by the interface as well as all the methods declared by any inherited interfaces unless the class is defined as abstract.

//

RECAP

What you should know at this point:

- What are abstract classes and why we use them.
- How to code and use abstract classes.
- What are interfaces and why we use them.
- How to code and work with interfaces.
- Difference between abstract classes and interfaces.