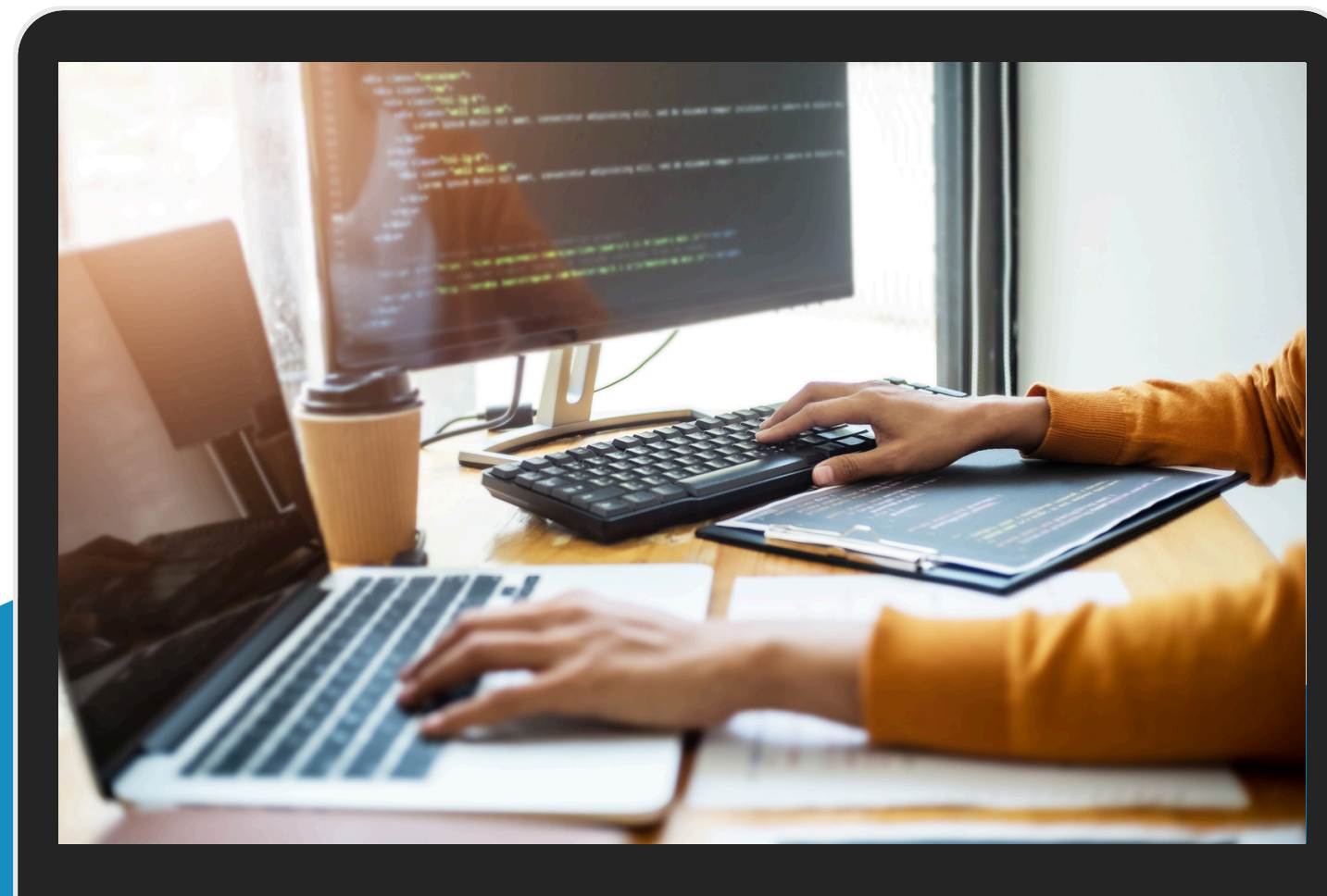


**INDUSTRY
CONNECT**
Gateway to Your IT Career

Software Developer Job Ready Programme




Agenda

Week 1



Day 2

- What is a Class
- Value types versus Reference types
- Class declaration
- Constructors and initialization



Types, classes and objects

What is a Class

A Class is like an object constructor, or a "blueprint" for creating objects.

In C#, classes are used to create custom types. The class defines the kinds of information and methods included in a custom type.

A type that is defined as a class is a reference type. At run time, when you declare a variable of a reference type, the variable contains the value null until you explicitly create an instance of the class by using the new operator

Value types versus Reference types

A data type is a value type if it holds a data value within its own memory space. It means the variables of these data types directly contain values.

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored

Class declaration

Classes are declared by using the class keyword followed by a unique identifier, as shown in the following example:

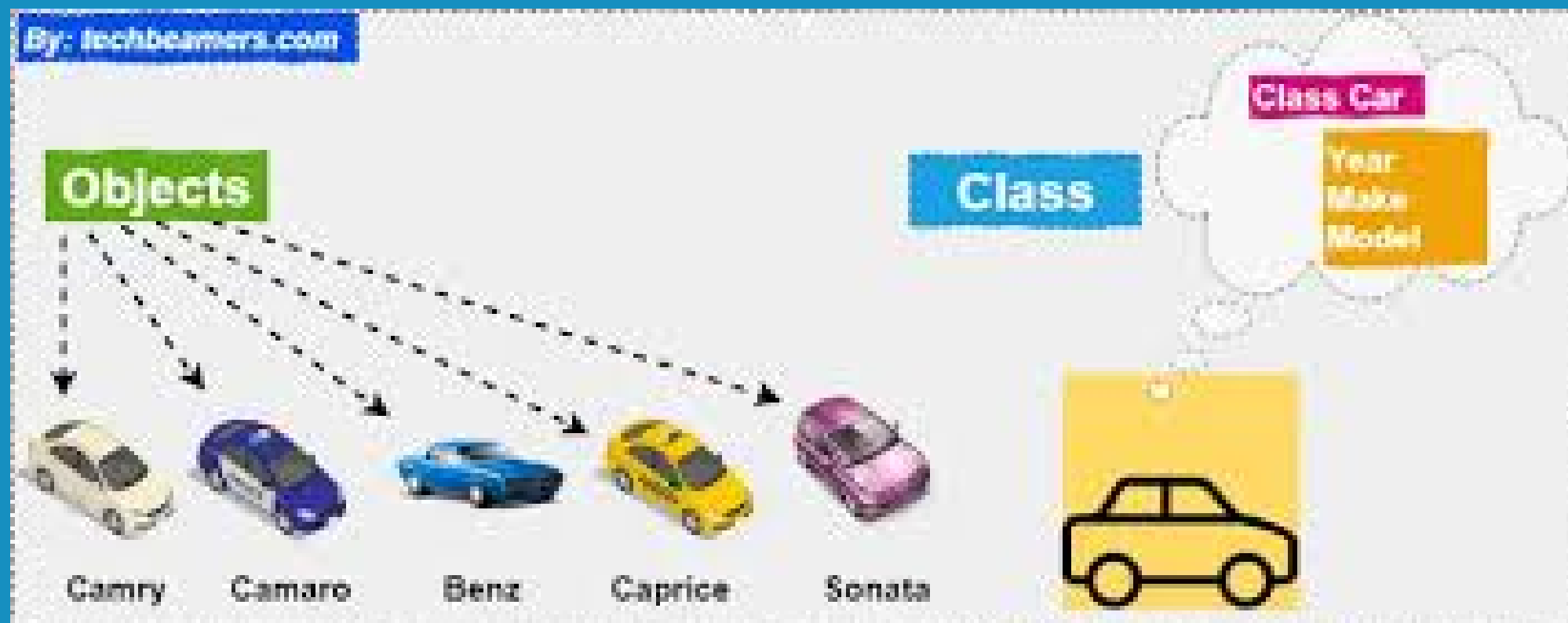
```
public class Customer { // Fields, properties, methods and events go here... }
```

NB remember our access modifiers. We don't want everything to be declared as public so use the most appropriate modifier

Classes and Objects

Although they're sometimes used interchangeably, a class and an object are different things. A class defines a type of object, but it isn't an object itself.

An object is a concrete entity based on a class, and is sometimes referred to as an instance of a class.



Classes and Objects contd

Objects can be created by using the new keyword followed by the name of the class, like this:

```
Customer object1 = new Customer();
```

You can create an object reference without creating an object at all

```
Customer object1
```

The above code will create a null object

Static Classes and Static Class Members

A static class is basically the same as a non-static class, but there's one difference: a static class can't be instantiated.

```
public static class TemperatureConverter
{
    public static double CelsiusToFahrenheit(string temperatureCelsius){}
}
```

A non-static class can contain static methods, fields, properties, or events. The static member is callable on a class even when no instance of the class exists.

Constructors and initialization

Objects can be created by using the new keyword followed by the name of the class, like this:

```
Customer object1 = new Customer();
```

You can create an object reference without creating an object at all

```
Customer object1
```

The above code will create a null object

Constructors and initialization

When you create an instance of a type, you want to ensure that its fields and properties are initialized to useful values.

Every .NET type has a default value. Typically, that value is 0 for number types, and null for all reference types. You can rely on that default value when it's reasonable in your app.

```
public class Container
{
    // Initialize capacity field to a default value of 10:
    private int _capacity = 10;
}
```

Constructors and initialization

You can require callers to provide an initial value by defining a constructor that's responsible for setting that initial value:

```
public class Container { private int _capacity; public Container(int capacity)  
    => _capacity = capacity; }
```

```
public class Container { private int _capacity; public Container(int capacity) => _capacity  
    = capacity; }
```

```
public class Container(int capacity) { private int _capacity = capacity; }
```

Constructors and initialization

You can also use the required modifier on a property and allow callers to use an object initializer to set the initial value of the property:

```
public class Person
{
    public required string LastName
    { get; set; }
    { get; set; }
}
```

```
public class Person { public required string LastName { get; set; } public
    required string FirstName { get; set; } }
```

```
var p1 = new Person(); // Error! Required properties not set
var p2 = new Person() { FirstName = "Grace", LastName = "Hopper" };
```