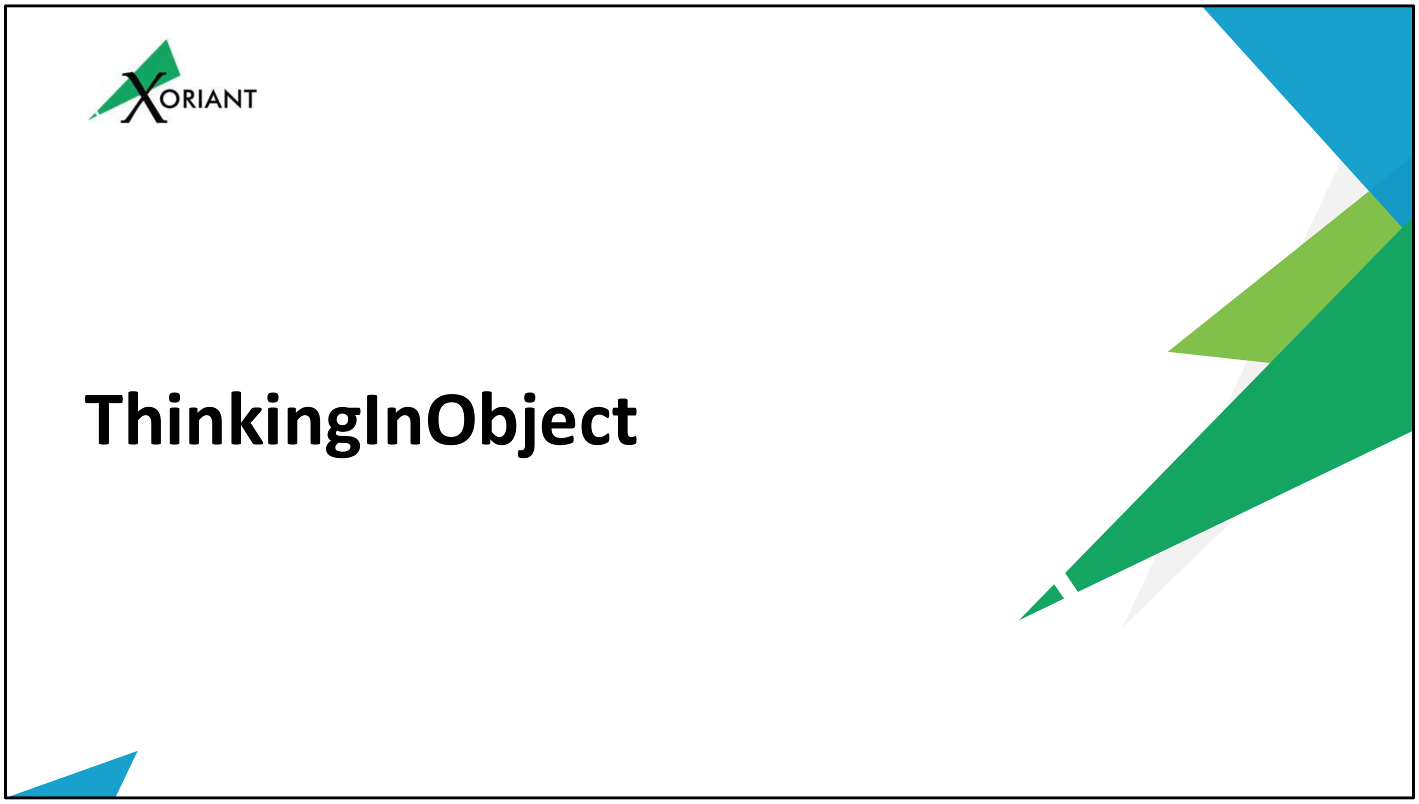
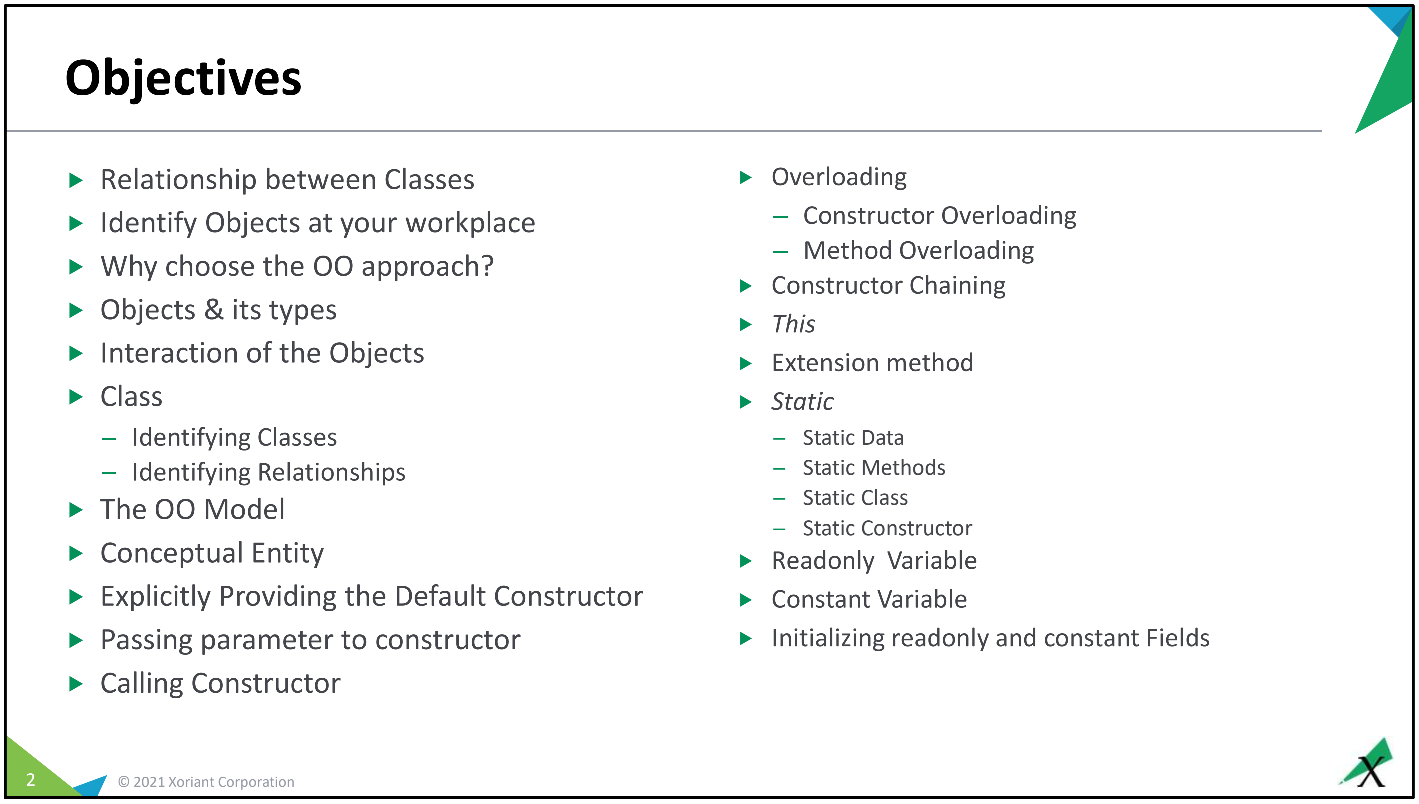


XORIANT

ThinkingInObject



Objectives

- Relationship between Classes
- Identify Objects at your workplace
- Why choose the OO approach?
- Objects & its types
- Interaction of the Objects
- Class
  - Identifying Classes
  - Identifying Relationships
- The OO Model
- Conceptual Entity
- Explicitly Providing the Default Constructor
- Passing parameter to constructor
- Calling Constructor
- Overloading
  - Constructor Overloading
  - Method Overloading
- Constructor Chaining
- *This*
- Extension method
- *Static*
  - Static Data
  - Static Methods
  - Static Class
  - Static Constructor
- Readonly Variable
- Constant Variable
- Initializing readonly and constant Fields

TIME FOR CASE STUDY

Case Study-3

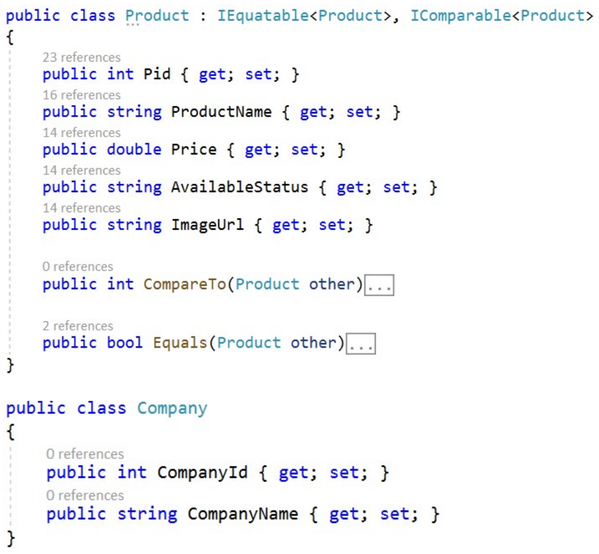
- As a portal **admin** of Shopon, I want to add more data associated to product like Company. Any company associated with Product will have Company details, Company contacts and other details.

Thought

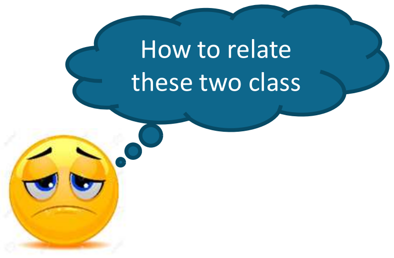
Thought

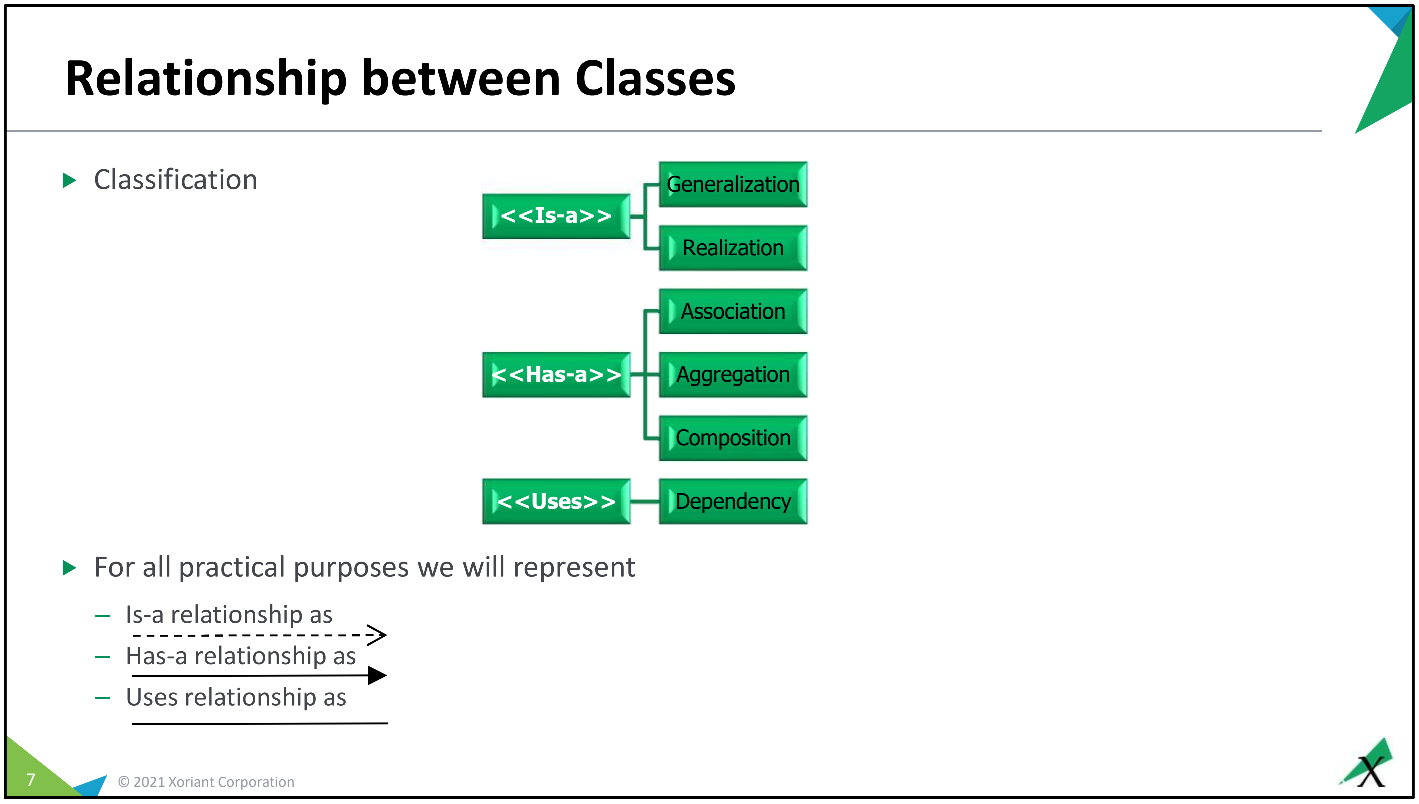
- To add more data related to Company, we can add new class named Company and associate it with Product.

Code



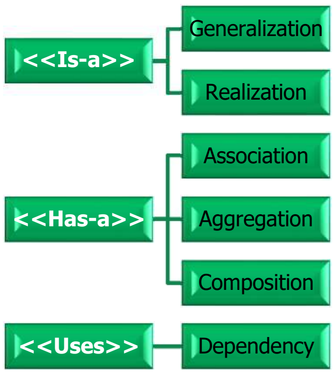
```
public class Product : IEquatable<Product>, IComparable<Product>
{
    23 references
    public int Pid { get; set; }
    16 references
    public string ProductName { get; set; }
    14 references
    public double Price { get; set; }
    14 references
    public string AvailableStatus { get; set; }
    14 references
    public string ImageUrl { get; set; }

    0 references
    public int CompareTo(Product other)...

    2 references
    public bool Equals(Product other)...
}

public class Company
{
    0 references
    public int CompanyId { get; set; }
    0 references
    public string CompanyName { get; set; }
}
```





Relationship between Classes

- Classification



- For all practical purposes we will represent
  - Is-a relationship as $\dashrightarrow$
  - Has-a relationship as $\longrightarrow$
  - Uses relationship as $\overline{\qquad\qquad}$

Identify Objects at your work place

Object

What makes laptop a LAPTOP ?

Objects are identified by their PROPERTIES and BEHAVIORS

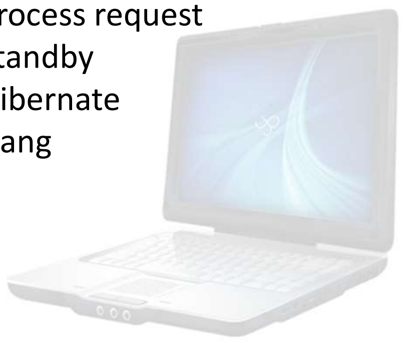
Object

PROPERTIES

1. Has LCD
2. Has Keyboard
3. Has USB Port
4. Has Mother board
5. Has Power Socket
6. Has Chip
7. Has RAM
8. Has ROM
9. Has Speakers
10. Has Camera

BEHAVIORS

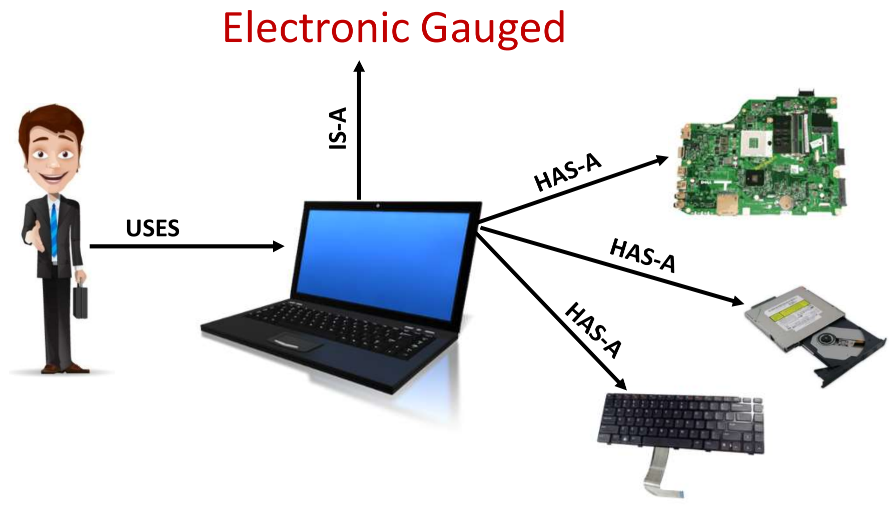
1. On
2. Off
3. Process request
4. Standby
5. Hibernate
6. Hang



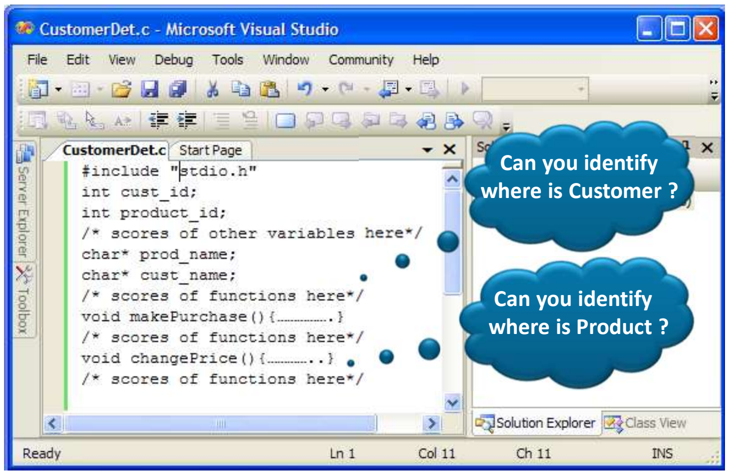
Class

Object

Interaction of the Objects

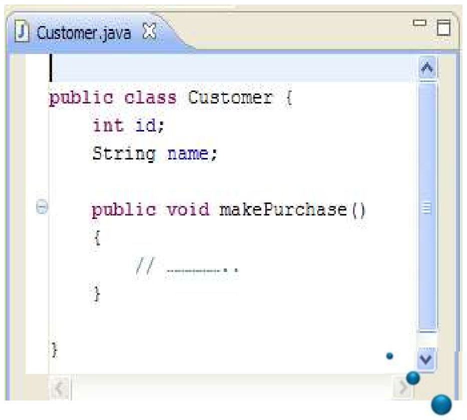


Why choose the OO approach?

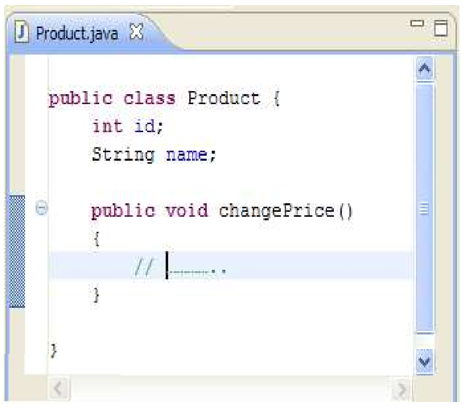

CustomerDet.c - Microsoft Visual Studio
File Edit View Debug Tools Window Community Help
CustomerDet.c Start Page
Can you identify where is Customer ?
Can you identify where is Product ?
Solution Explorer Class View
Ready Ln 1 Col 11 Ch 11 INS

```
#include "stdio.h"
int cust_id;
int product_id;
/* scores of other variables here*/
char* prod_name;
char* cust_name;
/* scores of functions here*/
void makePurchase(){..............}
/* scores of functions here*/
void changePrice(){...........}
/* scores of functions here*/
```

Why choose the OO approach ?


Customer.java

```
public class Customer {
    int id;
    String name;

    public void makePurchase()
    {
        // ……….
    }

}
```


Product.java

```
public class Product {
    int id;
    String name;

    public void changePrice()
    {
        // ……….
    }

}
```

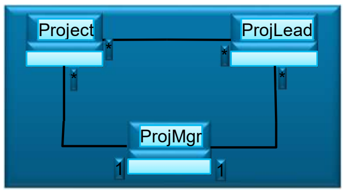
Objects & its types

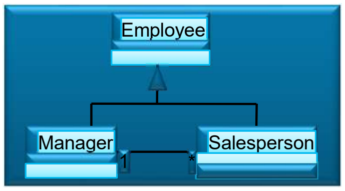
- We interact with ***objects*** everyday
  - A customer
  - An order
  - Your car
  - Your Mobile

- An object represents an entity – physical, conceptual or software
  - **Physical entity**
    - Employee, Customer, Supplier
  - **Conceptual entity**
    - Account, Policy, FeesCalculator
  - **Software entity**
    - List, Connection, etc.
- ***A programmer should make a good effort to capture the conceptual entities in addition to physical entities which are relatively straight forward to identify***

Why choose the OO approach?

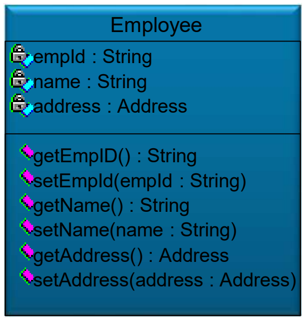
- The OO approach
  - Deals with classes as the building blocks
  - Allows Real World Modeling
  - The idea of OOP is to try to approach programming in a more natural way by grouping all the code that belongs to a particular object—such as an account or a customer — together
- Raise the level of abstraction
  - Applications can be implemented in the same terms in which they are described by users
- Easier to find nouns and construct a system centered around the nouns than actions in isolation
- Easier to visualize an encapsulated representation of data and responsibilities of entities present in the domain
- The modern methodologies recommend the object-oriented approach even for applications developed in C or Cobol





Class

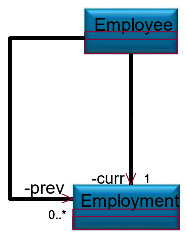
- User defined type
  - Encapsulates all the data and operations pertaining to an entity
  - Provides a Single representation to all the attributes defining the entity
  - Passing single representations is easier

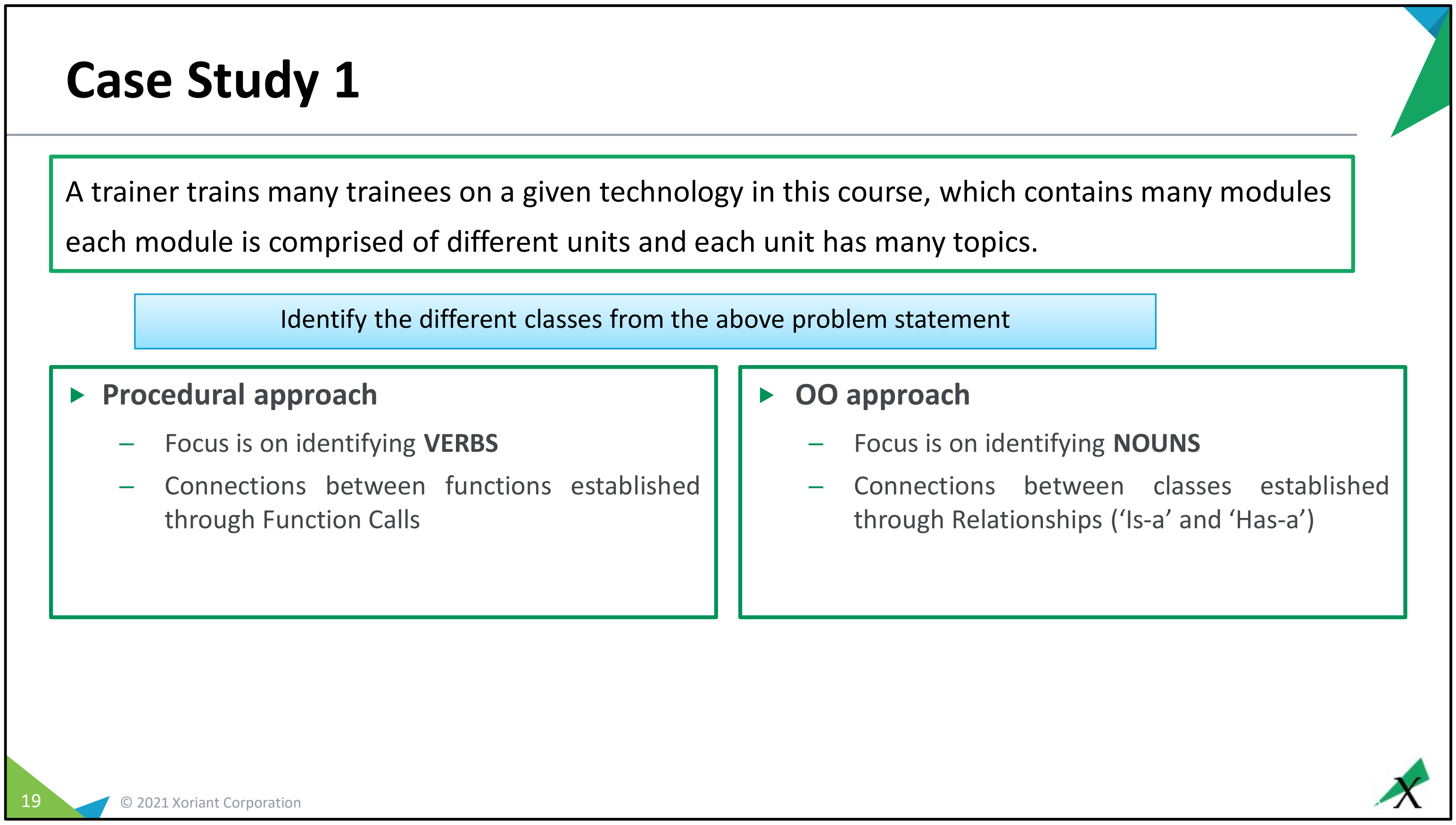


- Data types as collections
  - A struct in C encapsulates only data. Used as a data structure to store different types of data
  - An array is used to store different elements of the same type

Has-a Relationship

- The 'Has-a' relationships are qualified by
  - Multiplicity
    - The number of instances with which a class is associated
    - Can be 1, 0..1, *, 1..*, 0..*, 2..*, 5..10, etc.
    - Multiplicity is by default 1
  - Navigability
    - Can be unidirectional or bidirectional
    - Navigability is by default bi-directional
  - Role name
    - The name of the instance in the relationship
    - Multiple 'has-a' based on different roles are possible





Case Study 1

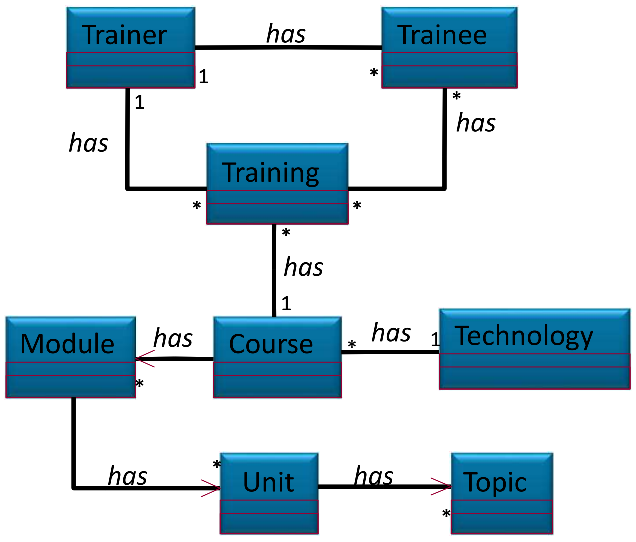
A trainer trains many trainees on a given technology in this course, which contains many modules each module is comprised of different units and each unit has many topics.

Identify the different classes from the above problem statement

- **Procedural approach**
  - Focus is on identifying **VERBS**
  - Connections between functions established through Function Calls
- **OO approach**
  - Focus is on identifying **NOUNS**
  - Connections between classes established through Relationships (‘Is-a’ and ‘Has-a’)

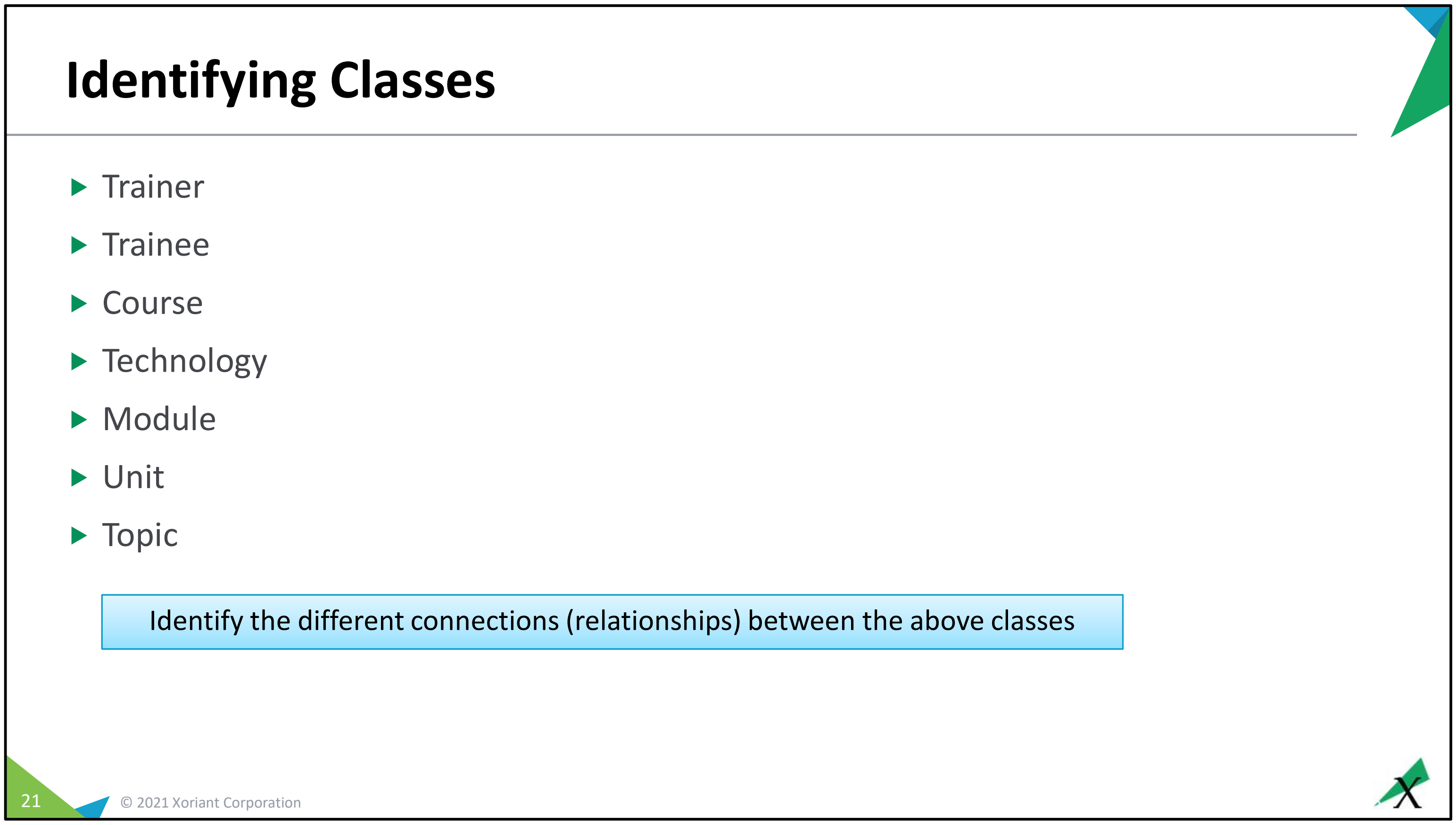
19

Solution



It is not necessary to always have a CONCEPTUAL ENTITY in a Design

Easier to model real-world problems through the OO approach than through the procedural approach



Identifying Classes

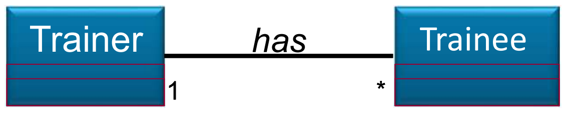
- Trainer
- Trainee
- Course
- Technology
- Module
- Unit
- Topic

Identify the different connections (relationships) between the above classes

21

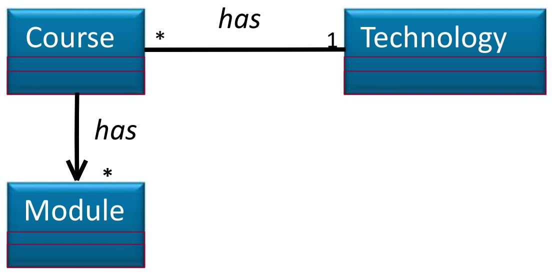
Identifying Relationships

- Trainer - Trainee
  - Trainer ‘HAS’ many Trainees
  - Every Trainee ‘HAS’ a Trainer



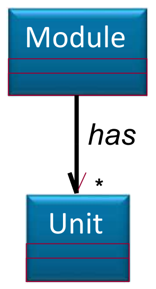
Identifying Relationships

- Course – Technology
- Course - Module
  - Course ‘HAS’ an associated Technology
  - A Technology has many courses
  - Course ‘HAS’ many Modules

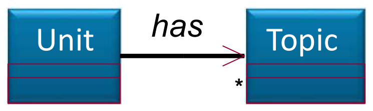


Identifying Relationships

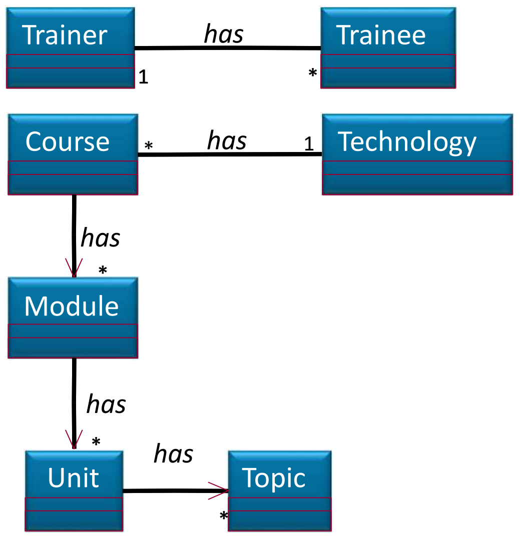
- Module – Unit
  - Module ‘HAS’ many Units



- Unit – Topic
  - Unit ‘HAS’ many Topics



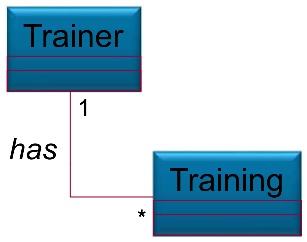
The OO Model



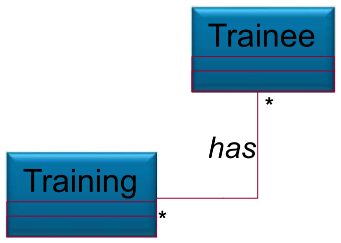
How do you relate the Trainer & Trainee to the Course?

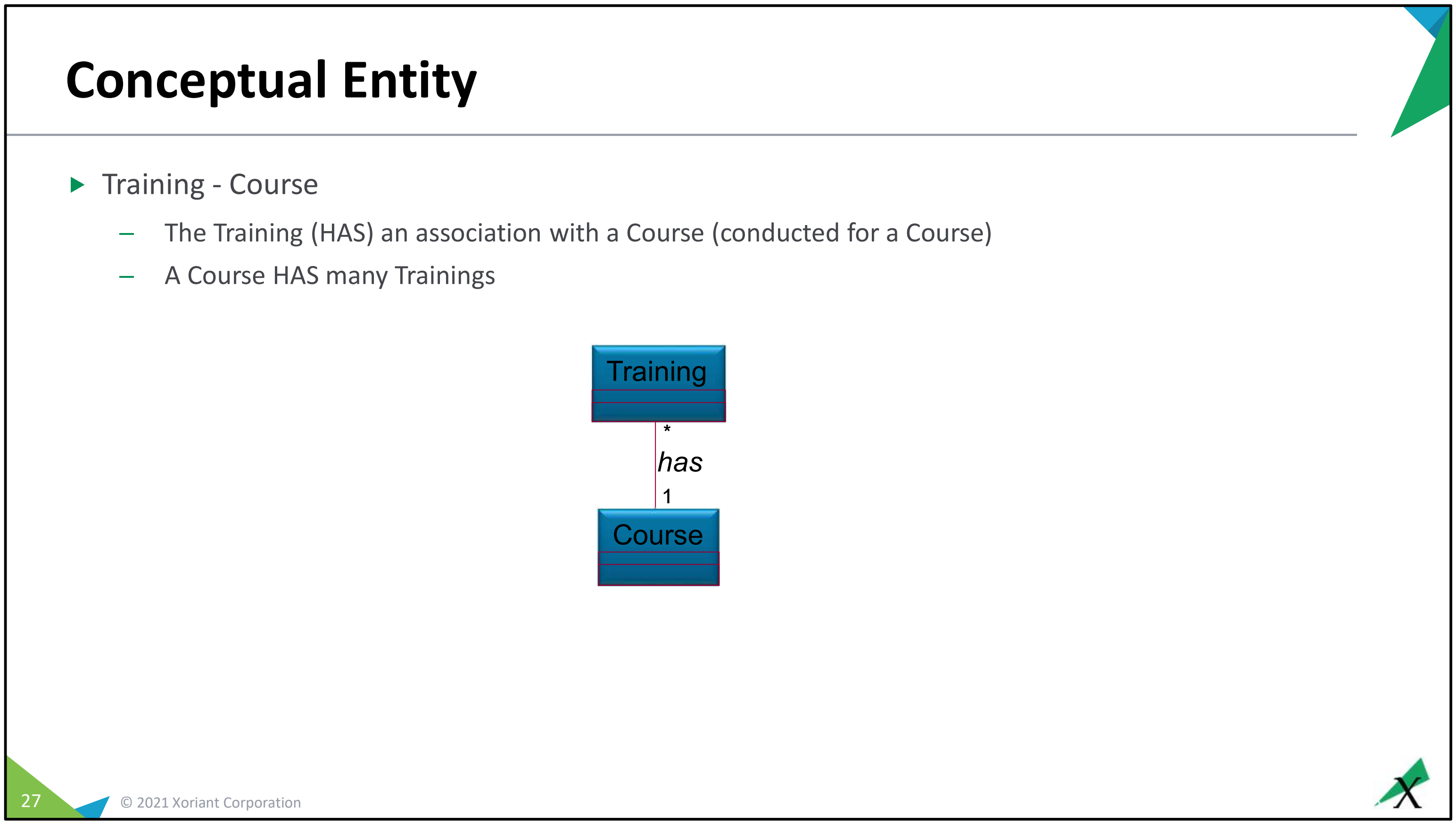
Conceptual Entity

- Trainer – Training
  - A Trainer (HAS) conducts many Trainings
  - A Training HAS a Trainer



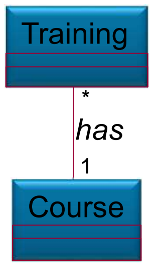
- Trainee – Training
  - A Trainee (HAS) attends many Trainings
  - A Training HAS a many Trainees





Conceptual Entity

- Training - Course
  - The Training (HAS) an association with a Course (conducted for a Course)
  - A Course HAS many Trainings



27

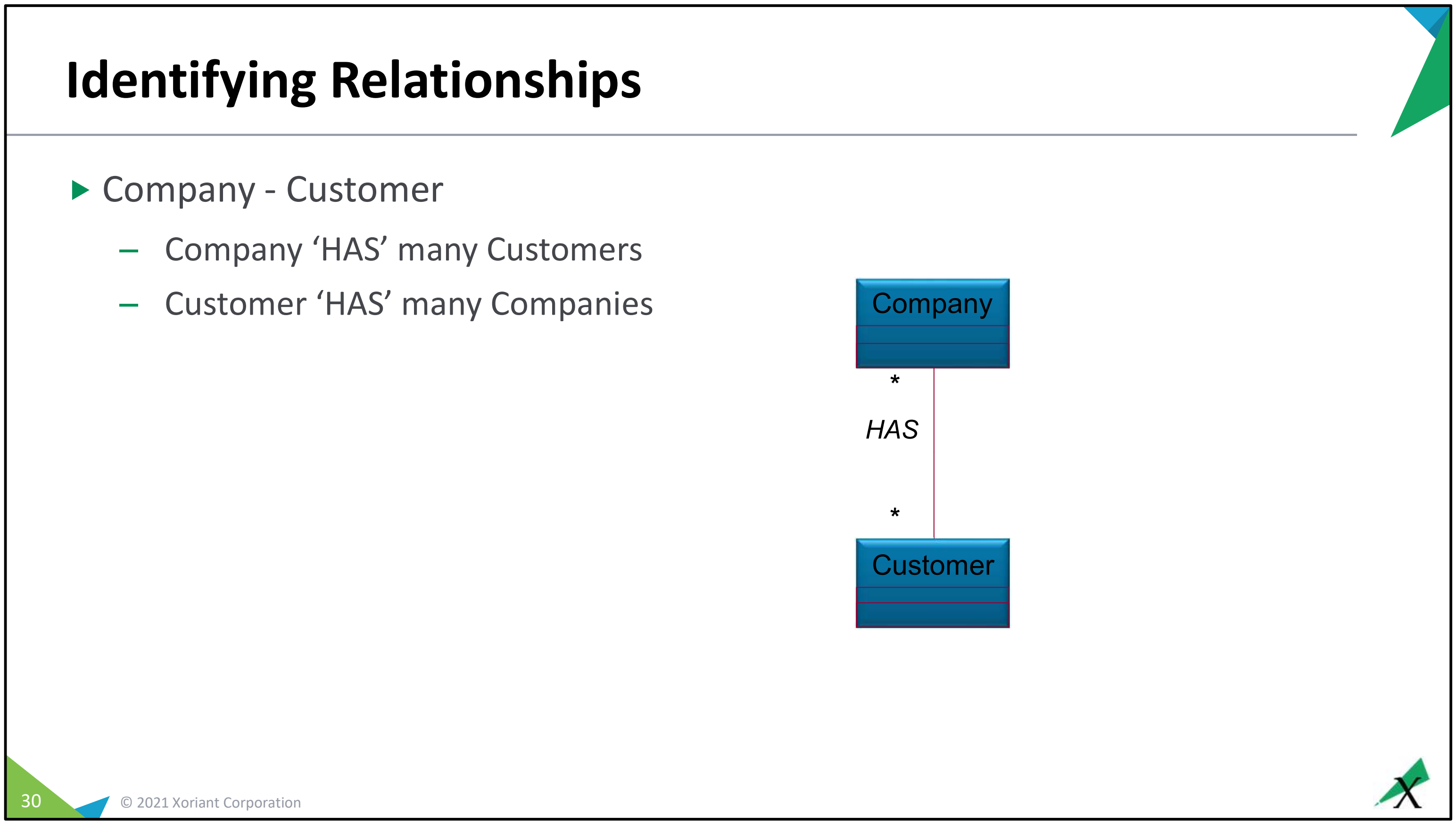
Case Study 2

A company sells different items to customers who have placed orders. An order can be placed for several items. However, a company gives special discounts to its registered customers.

- Identify the different classes from the above problem statement
- Identify the different connections (relationships) between the above classes

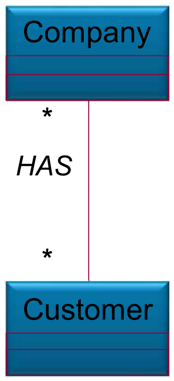
Identifying Classes

- Company
- Item
- Order
- Customer
- RegCustomer



Identifying Relationships

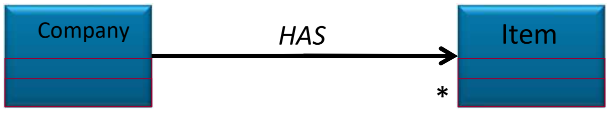
- Company - Customer
  - Company ‘HAS’ many Customers
  - Customer ‘HAS’ many Companies



30

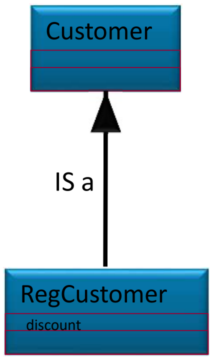
Identifying Relationships

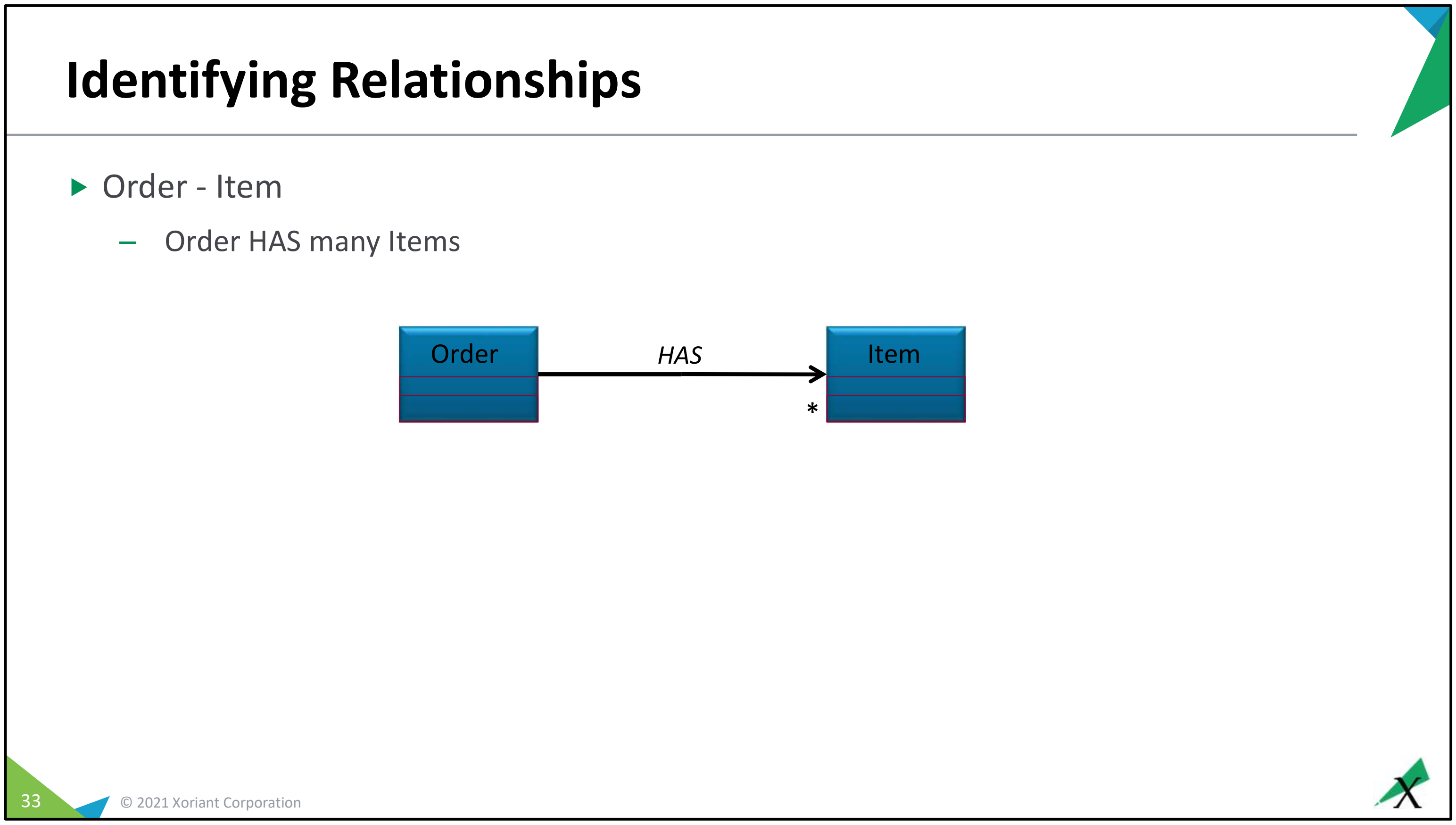
- Company - Item
  - Company HAS many Items



Identifying Relationships

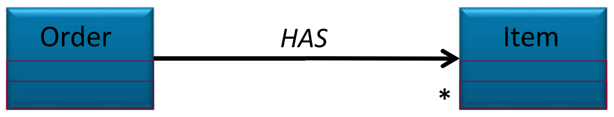
- Customer - RegCustomer
  - RegCustomer 'IS' a Customer

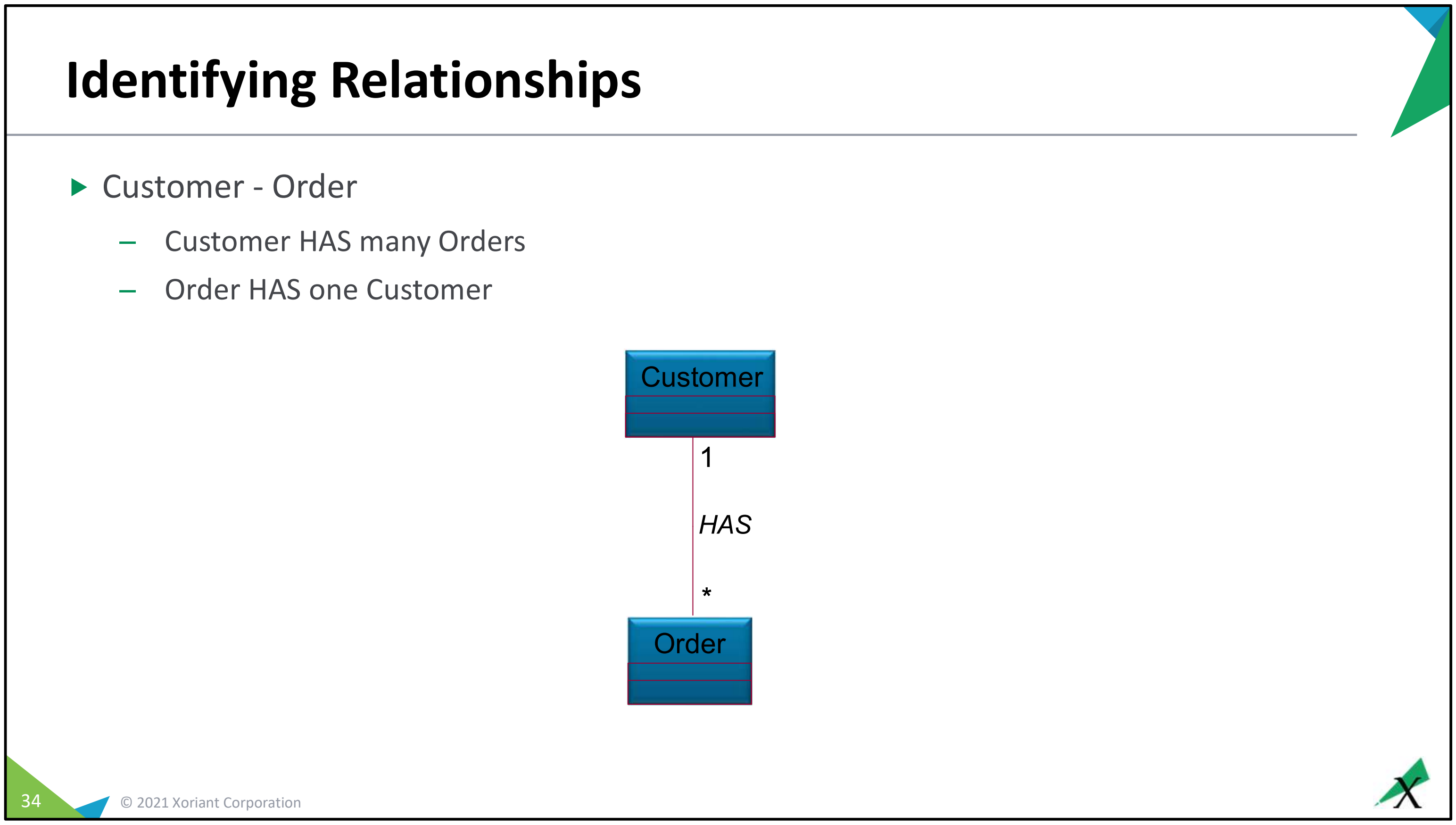




Identifying Relationships

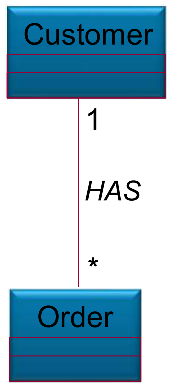
- Order - Item
  - Order HAS many Items





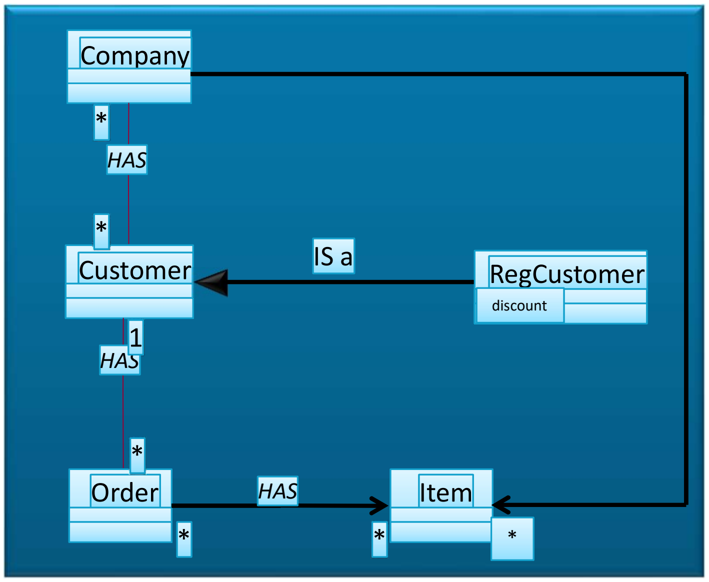
Identifying Relationships

- Customer - Order
  - Customer HAS many Orders
  - Order HAS one Customer



34

The OO Model



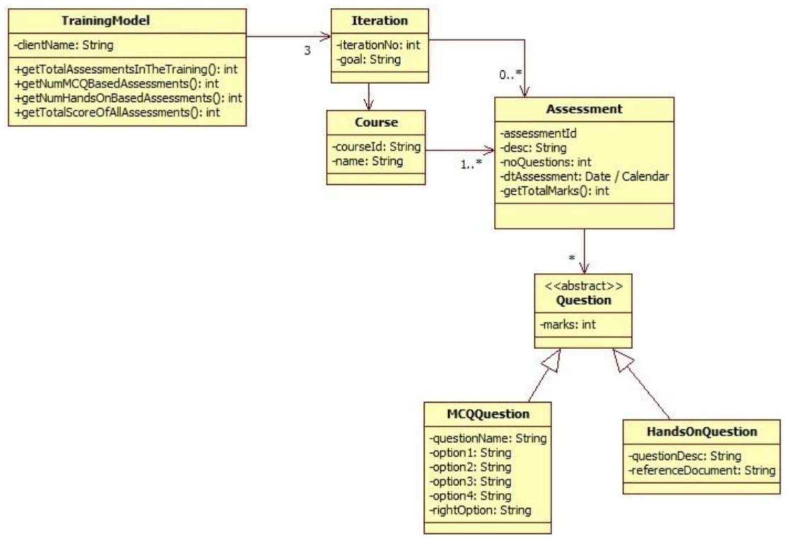
- A Customer can place many orders implies that RegCustomer can also place many Orders.
- A Company has many Customers implies that a Company also has many RegCustomers

Case Study 3

In the Xoriant Assure Assessment Framework, every course can have assessments. An Iteration has many courses and can also have additional assessments. The training model has 4 Iterations. An assessment can be of multiple-choice type, hands-on exercise or project.

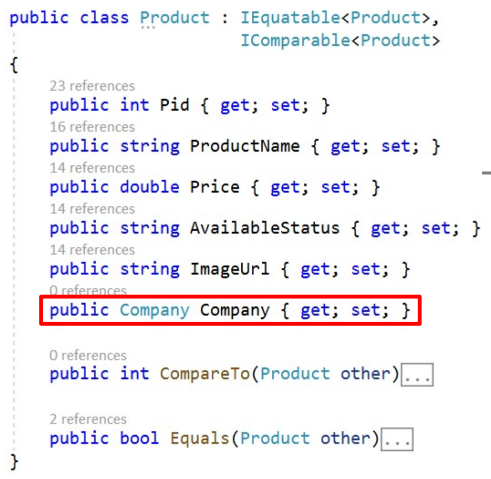
- Identify the different classes from the above problem statement
- Identify the different connections (relationships) between the above classes

The OO Model

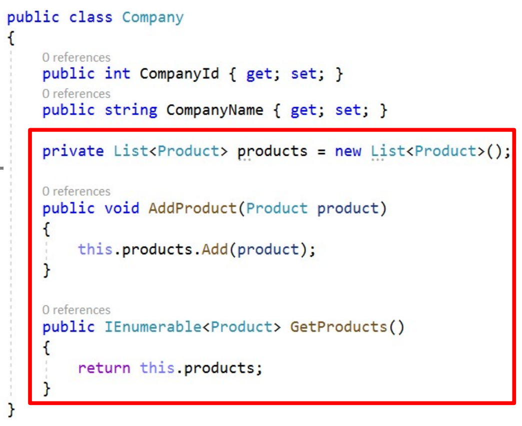


TrainingModel
-clientName: String
+getTotalAssessmentsInTheTraining(): int
+getNumMCQBasedAssessments(): int
+getNumHandsOnBasedAssessments(): int
+getTotalScoreOfAllAssessments(): int
3
Iteration
-iterationNo: int
-goal: String
0..*
Course
-courseId: String
-name: String
1..*
Assessment
-assessmentId
-desc: String
-noQuestions: int
-dtAssessment: Date / Calendar
-getTotalMarks(): int
*
<<abstract>>
Question
-marks: int
MCQQuestion
-questionName: String
-option1: String
-option2: String
-option3: String
-option4: String
-rightOption: String
HandsOnQuestion
-questionDesc: String
-referenceDocument: String

Solution - Shopon



```
public class Product : IEquatable<Product>,
                       IComparable<Product>
{
    23 references
    public int Pid { get; set; }
    16 references
    public string ProductName { get; set; }
    14 references
    public double Price { get; set; }
    14 references
    public string AvailableStatus { get; set; }
    14 references
    public string ImageUrl { get; set; }
    0 references
    public Company Company { get; set; }

    0 references
    public int CompareTo(Product other)...

    2 references
    public bool Equals(Product other)...
}
```



```
public class Company
{
    0 references
    public int CompanyId { get; set; }
    0 references
    public string CompanyName { get; set; }

    private List<Product> products = new List<Product>();

    0 references
    public void AddProduct(Product product)
    {
        this.products.Add(product);
    }

    0 references
    public IEnumerable<Product> GetProducts()
    {
        return this.products;
    }
}
```

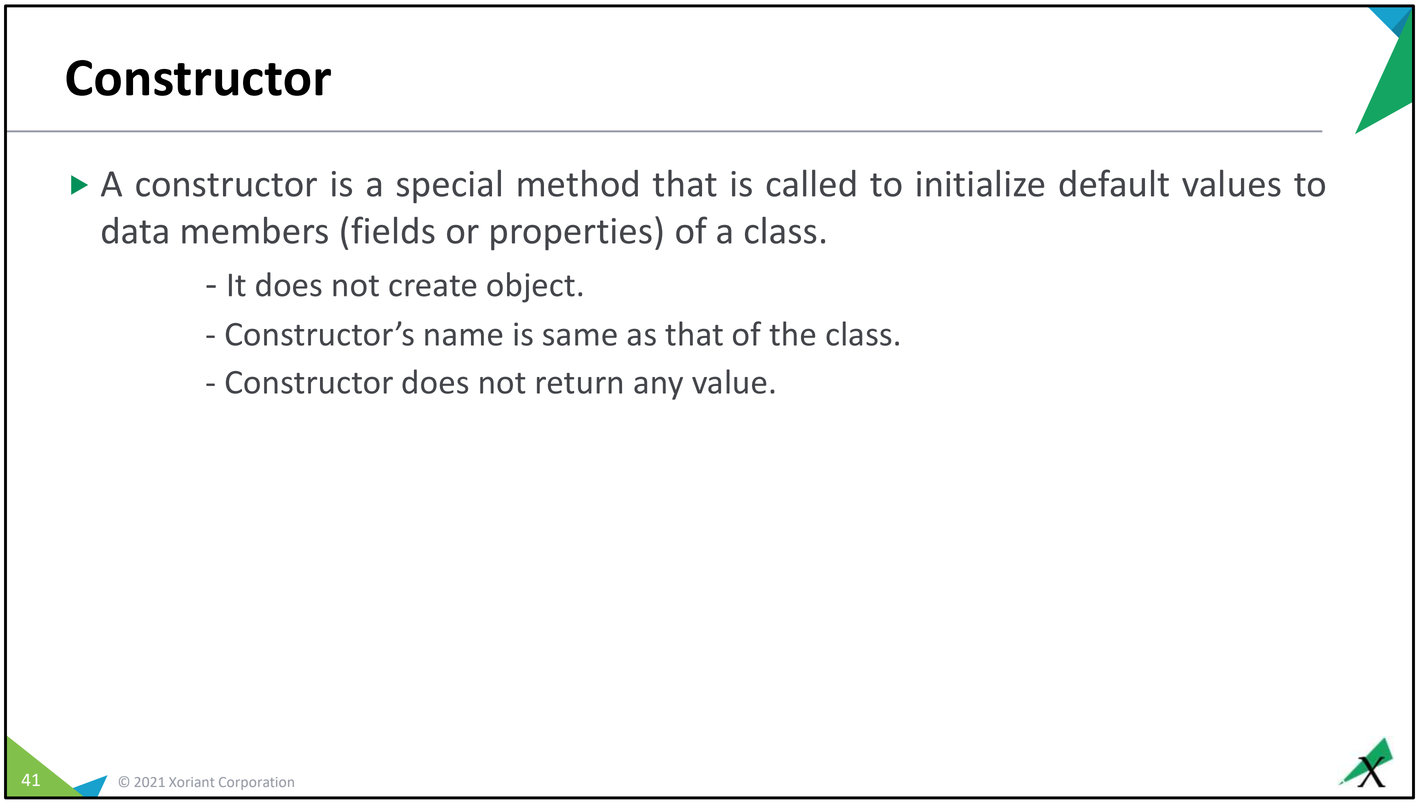
Case Study - Continued

- As a portal **admin** of Shopon, I can create product without even adding any column data. I want the system not to create product without Pid, ProductName and Price.

Thought

Thought

- While creating Product, we must pass the values so that it is getting stored. This can be achieved by using Constructor.


Constructor

- A constructor is a special method that is called to initialize default values to data members (fields or properties) of a class.
  - It does not create object.
  - Constructor’s name is same as that of the class.
  - Constructor does not return any value.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-constructors

Default Constructor

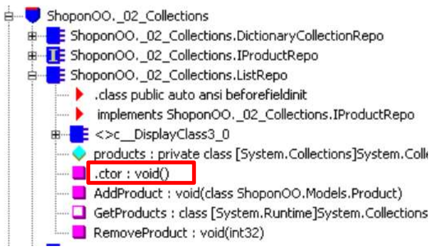
- Features of a default constructor
  - Public accessibility
  - Same name as the class
  - No return type—not even **void**
  - Expects no arguments
  - Initializes all fields to **zero**, **false** or **null**

NOTE: run **ILDASM** command passing the assembly name to see the disassembled code of the file using **Developer Command Prompt for VS**

- **Constructor Example:**

```
public class ListRepo : IProductRepo
{
    private List<Product> products = new List<Product>();

    10 references
    public void AddProduct(Product product)...

    7 references
    public IEnumerable<Product> GetProducts()...

    6 references
    public void RemoveProduct(int id)...
}
```

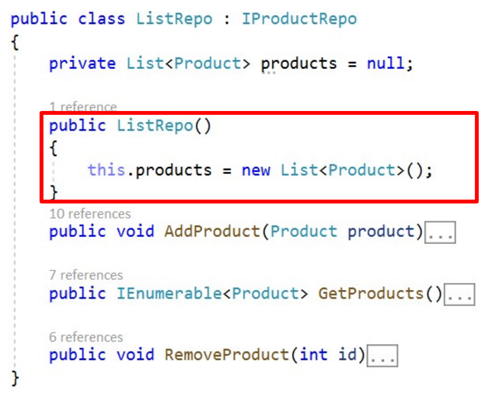
Disassembly of ShoponOO.Models.dll



```
ShoponOO._02_Collections
  ShoponOO._02_Collections.DictionaryCollectionRepo
  ShoponOO._02_Collections.IProductRepo
  ShoponOO._02_Collections.ListRepo
    .class public auto ansi beforefieldinit
    implements ShoponOO._02_Collections.IProductRepo
    <>c__DisplayClass3_0
    products : private class [System.Collections]System.Coll
    .ctor : void()
    AddProduct : void(class ShoponOO.Models.Product)
    GetProducts : class [System.Runtime]System.Collections
    RemoveProduct : void(int32)
```

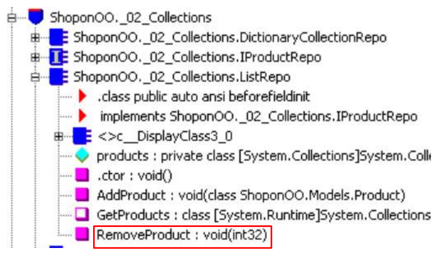
Explicitly Providing the Default Constructor

- Default constructor will be supplied by compiler
- But, the default constructor might be inappropriate
  - If so, do not use it; write your own!
  - If we are providing default constructor, then compiler will not supply the default constructor

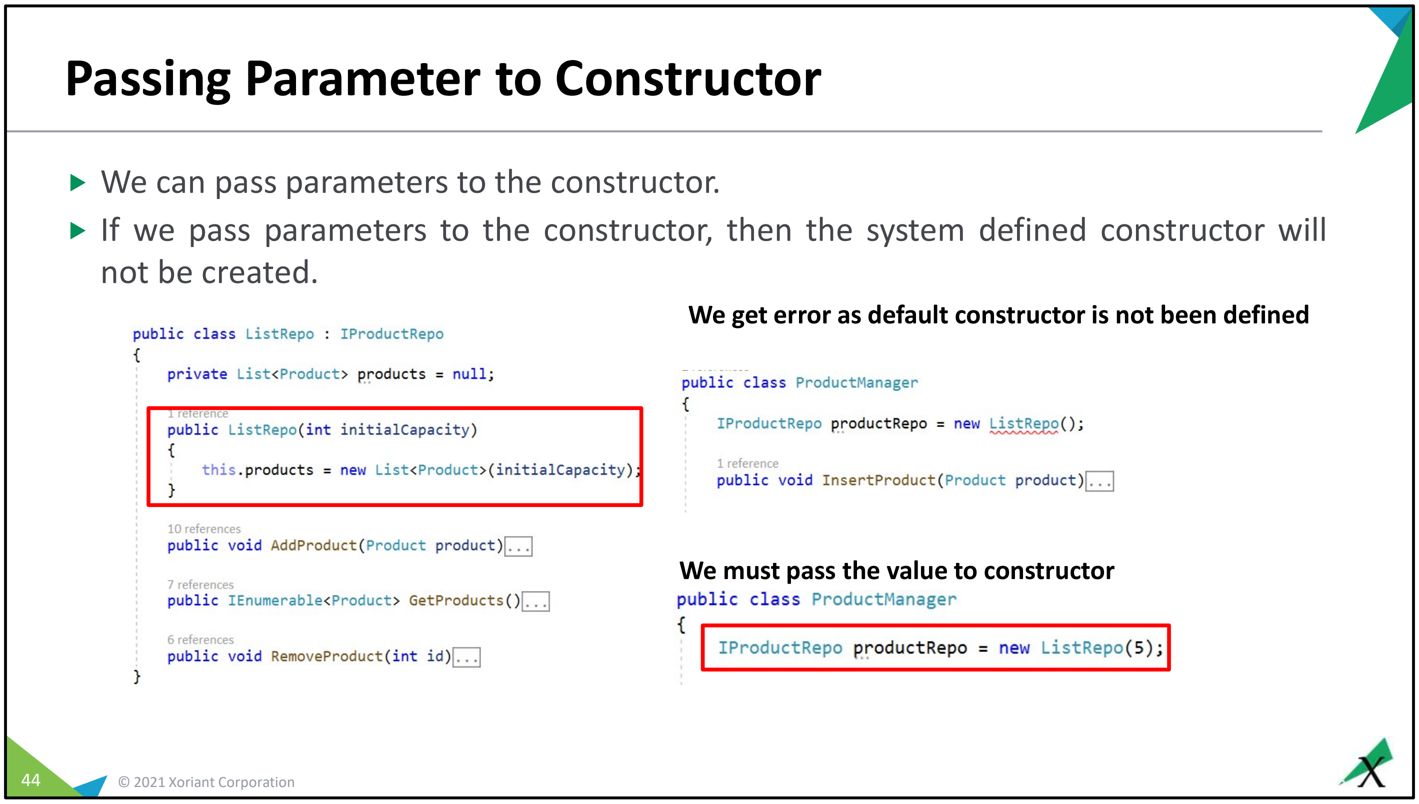


```
public class ListRepo : IProductRepo
{
    private List<Product> products = null;

    1 reference
    public ListRepo()
    {
        this.products = new List<Product>();
    }
    10 references
    public void AddProduct(Product product)...

    7 references
    public IEnumerable<Product> GetProducts()...

    6 references
    public void RemoveProduct(int id)...
}
```

Disassembly of ShoponOO.Models.dll



```
ShoponOO._02_Collections
    ShoponOO._02_Collections.DictionaryCollectionRepo
    ShoponOO._02_Collections.IProductRepo
    ShoponOO._02_Collections.ListRepo
        .class public auto ansi beforefieldinit
        implements ShoponOO._02_Collections.IProductRepo
        <>c__DisplayClass3_0
        products : private class [System.Collections]System.Coll
        .ctor : void()
        AddProduct : void(class ShoponOO.Models.Product)
        GetProducts : class [System.Runtime]System.Collections
        RemoveProduct : void(int32)
```



Passing Parameter to Constructor

- We can pass parameters to the constructor.
- If we pass parameters to the constructor, then the system defined constructor will not be created.

```
public class ListRepo : IProductRepo
{
    private List<Product> products = null;

    1 reference
    public ListRepo(int initialCapacity)
    {
        this.products = new List<Product>(initialCapacity);
    }

    10 references
    public void AddProduct(Product product)...

    7 references
    public IEnumerable<Product> GetProducts()...

    6 references
    public void RemoveProduct(int id)...
}
```

We get error as default constructor is not been defined

```
public class ProductManager
{
    IProductRepo productRepo = new ListRepo();

    1 reference
    public void InsertProduct(Product product)...
```

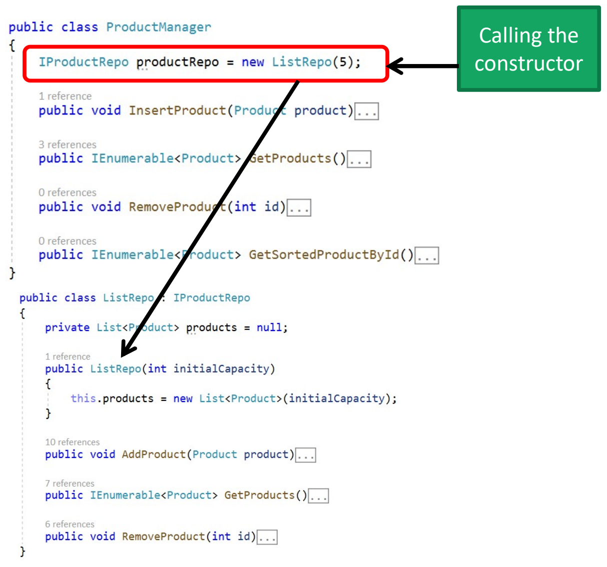
We must pass the value to constructor

```
public class ProductManager
{
    IProductRepo productRepo = new ListRepo(5);
```

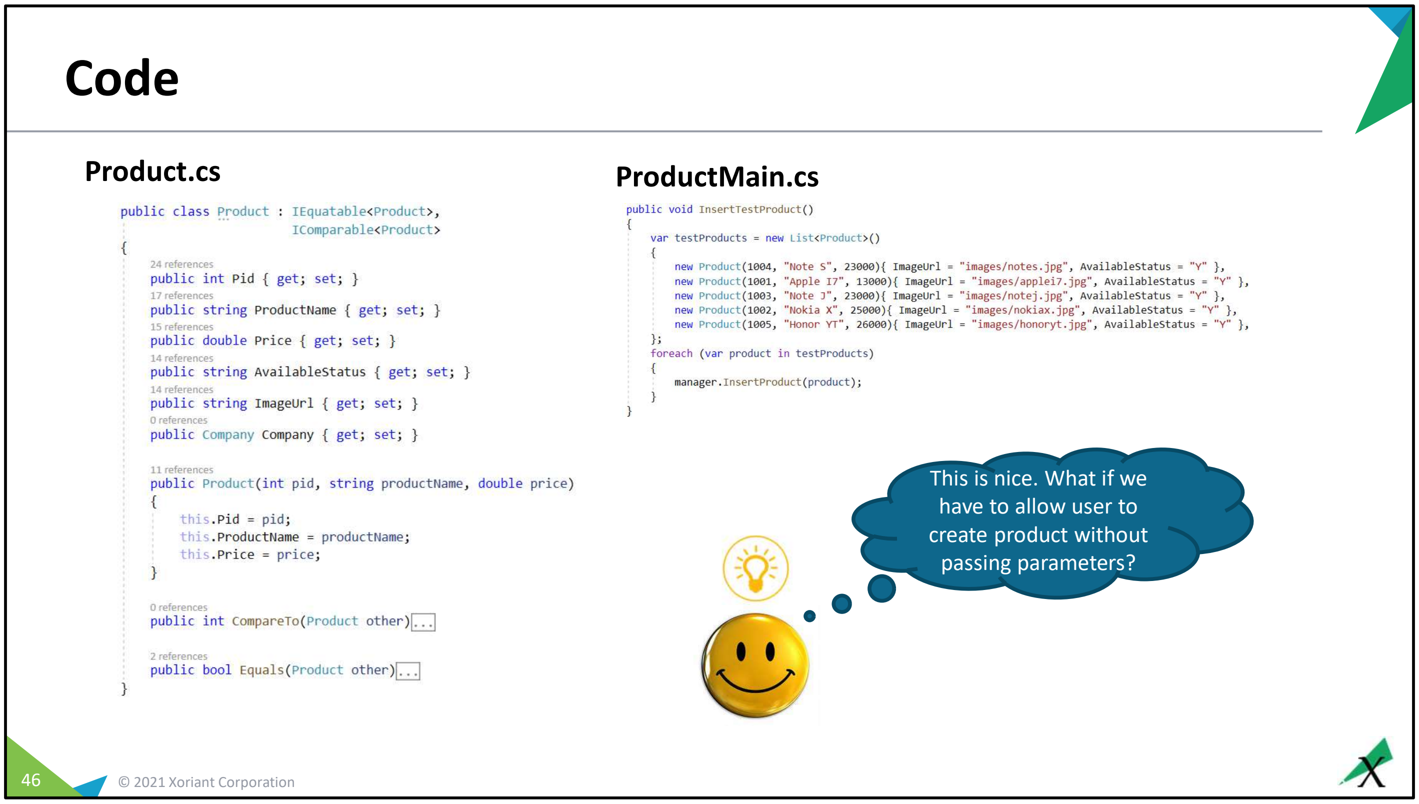
As we have not defined zero parameter constructor, compiler will throw error. To overcome this, we have to define zero parameter constructor.

Calling Constructor

- Initialize the object’s fields with default values by using a constructor
  - Use the name of the class followed by parentheses followed by ‘new’ keyword
  - As default value, zero is assigned to numerical fields (int, double, long etc.), null is assigned to reference types (string or any class variables) and false is assigned to boolean types



```
public class ProductManager
{
    IProductRepo productRepo = new ListRepo(5);

    1 reference
    public void InsertProduct(Product product)[...]

    3 references
    public IEnumerable<Product> GetProducts()[...]

    0 references
    public void RemoveProduct(int id)[...]

    0 references
    public IEnumerable<Product> GetSortedProductById()[...]
}
  public class ListRepo : IProductRepo
  {
      private List<Product> products = null;

      1 reference
      public ListRepo(int initialCapacity)
      {
          this.products = new List<Product>(initialCapacity);
      }

      10 references
      public void AddProduct(Product product)[...]

      7 references
      public IEnumerable<Product> GetProducts()[...]

      6 references
      public void RemoveProduct(int id)[...]
  }
```



Code

Product.cs

```
public class Product : IEquatable<Product>,
                       IComparable<Product>
{
    24 references
    public int Pid { get; set; }
    17 references
    public string ProductName { get; set; }
    15 references
    public double Price { get; set; }
    14 references
    public string AvailableStatus { get; set; }
    14 references
    public string ImageUrl { get; set; }
    0 references
    public Company Company { get; set; }

    11 references
    public Product(int pid, string productName, double price)
    {
        this.Pid = pid;
        this.ProductName = productName;
        this.Price = price;
    }

    0 references
    public int CompareTo(Product other)...

    2 references
    public bool Equals(Product other)...
}
```

ProductMain.cs

```
public void InsertTestProduct()
{
    var testProducts = new List<Product>()
    {
        new Product(1004, "Note S", 23000){ ImageUrl = "images/notes.jpg", AvailableStatus = "Y" },
        new Product(1001, "Apple I7", 13000){ ImageUrl = "images/applei7.jpg", AvailableStatus = "Y" },
        new Product(1003, "Note J", 23000){ ImageUrl = "images/notej.jpg", AvailableStatus = "Y" },
        new Product(1002, "Nokia X", 25000){ ImageUrl = "images/nokiax.jpg", AvailableStatus = "Y" },
        new Product(1005, "Honor YT", 26000){ ImageUrl = "images/honoryt.jpg", AvailableStatus = "Y" },
    };
    foreach (var product in testProducts)
    {
        manager.InsertProduct(product);
    }
}
```

This is nice. What if we have to allow user to create product without passing parameters?

46

© 2021 Xoriant Corporation

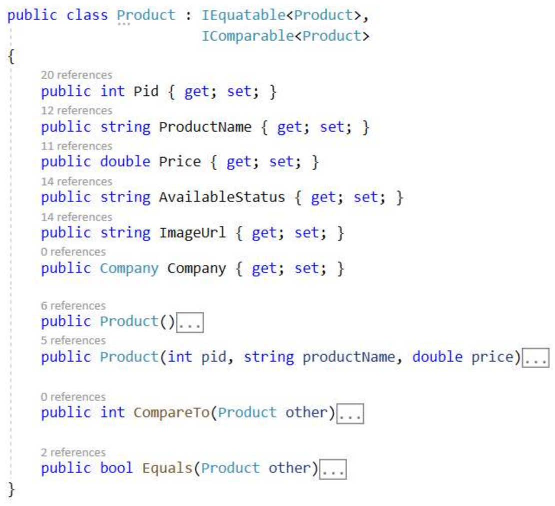
Allow the interns to think. We can create zero parameter constructor.

Constructor Overloading

- As we have to allow user to create Product without passing the parameter, we can **overloading** constructor.

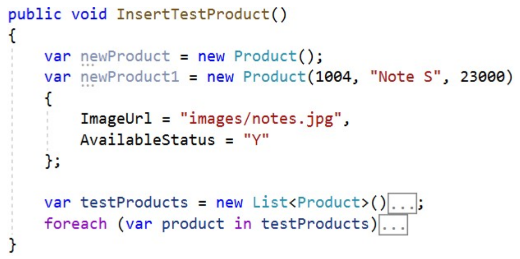
Constructor Overloading - Example

Product.cs

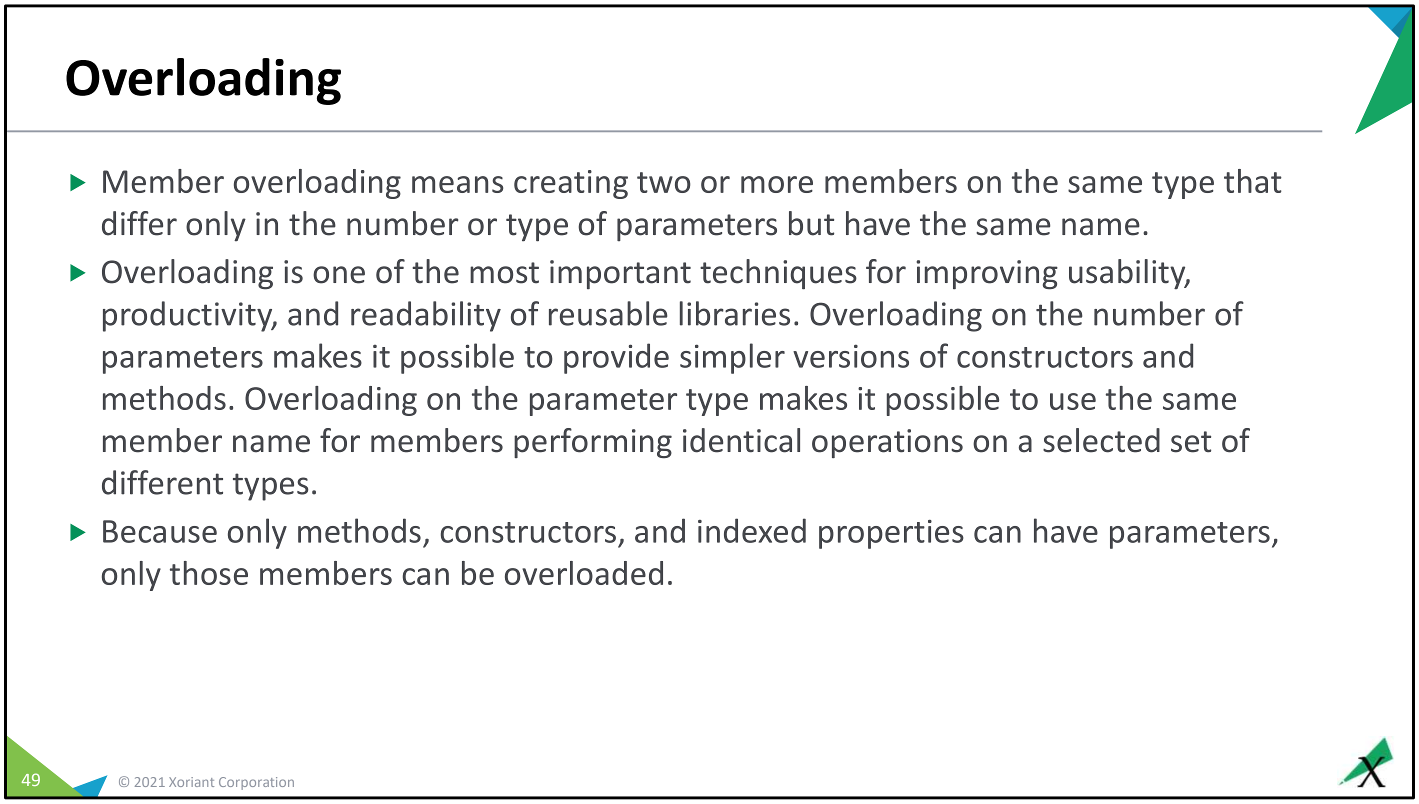


```
public class Product : IEquatable<Product>,
                       IComparable<Product>
{
    20 references
    public int Pid { get; set; }
    12 references
    public string ProductName { get; set; }
    11 references
    public double Price { get; set; }
    14 references
    public string AvailableStatus { get; set; }
    14 references
    public string ImageUrl { get; set; }
    0 references
    public Company Company { get; set; }

    6 references
    public Product()...
    5 references
    public Product(int pid, string productName, double price)...

    0 references
    public int CompareTo(Product other)...

    2 references
    public bool Equals(Product other)...
}
```

ProductMain.cs



```
public void InsertTestProduct()
{
    var newProduct = new Product();
    var newProduct1 = new Product(1004, "Note S", 23000)
    {
        ImageUrl = "images/notes.jpg",
        AvailableStatus = "Y"
    };

    var testProducts = new List<Product>()...;
    foreach (var product in testProducts)...
}
```



Overloading

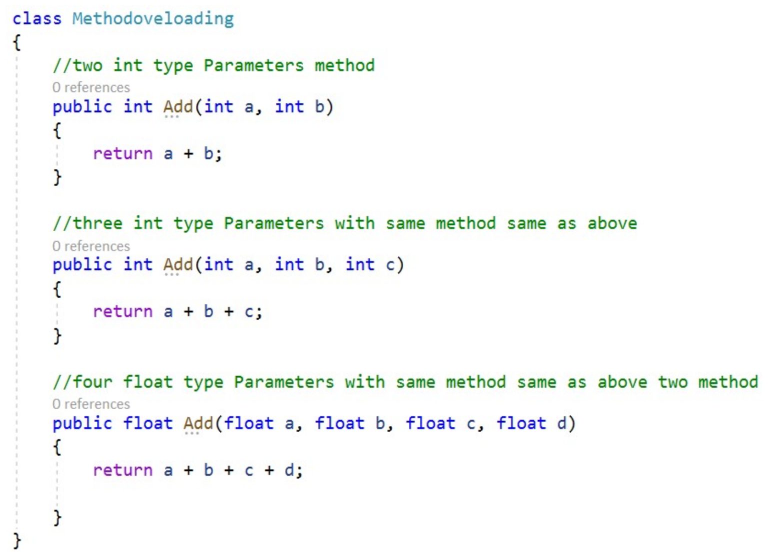
- Member overloading means creating two or more members on the same type that differ only in the number or type of parameters but have the same name.
- Overloading is one of the most important techniques for improving usability, productivity, and readability of reusable libraries. Overloading on the number of parameters makes it possible to provide simpler versions of constructors and methods. Overloading on the parameter type makes it possible to use the same member name for members performing identical operations on a selected set of different types.
- Because only methods, constructors, and indexed properties can have parameters, only those members can be overloaded.

https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/member-overloading

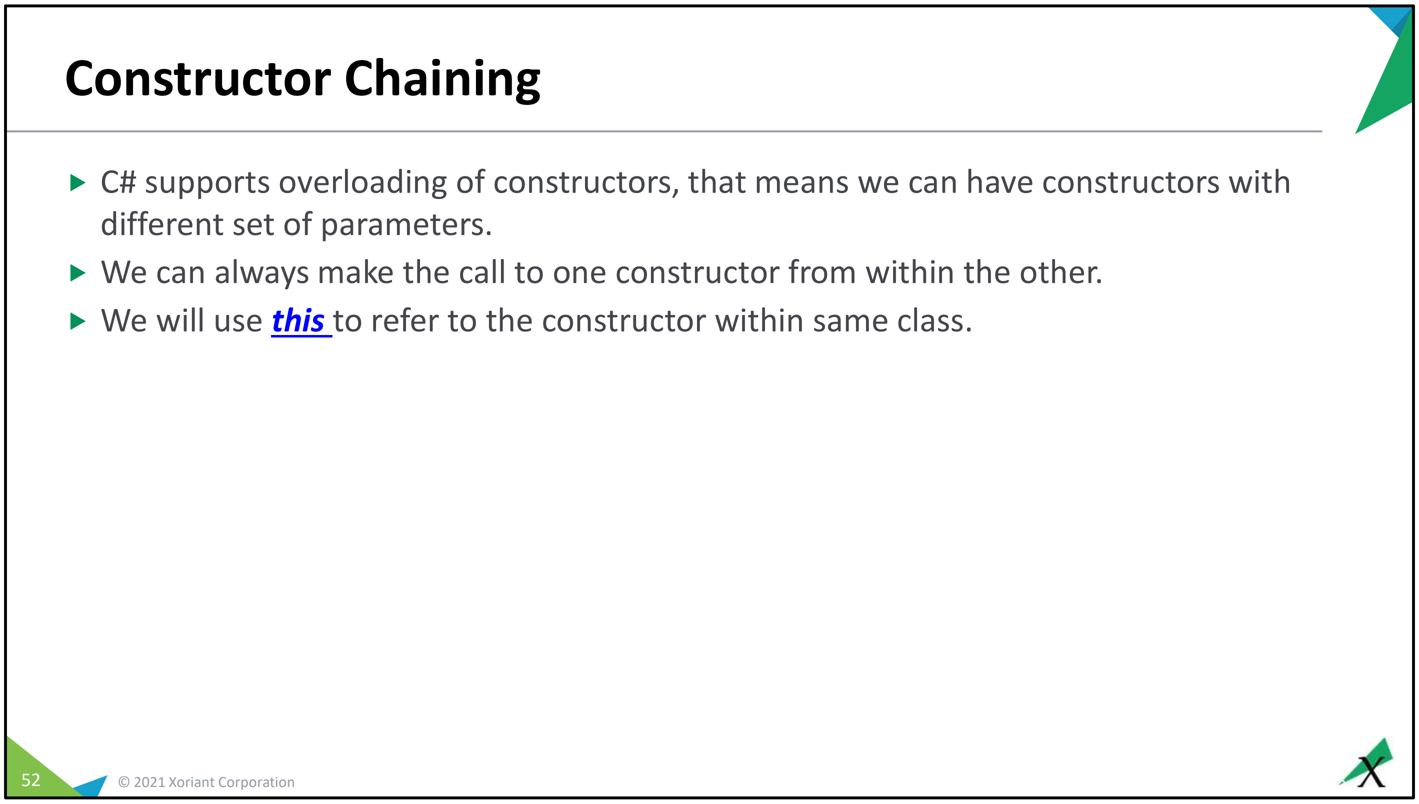
Method Overloading

- Method overloading allows programmers to use multiple methods with the same name. The methods are differentiated with their number and type of method arguments. Method overloading is an example of the polymorphism feature of an object oriented programming language.

Method Overloading - Example



```
class Methodoveloading
{
    //two int type Parameters method
    0 references
    public int Add(int a, int b)
    {
        return a + b;
    }

    //three int type Parameters with same method same as above
    0 references
    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }

    //four float type Parameters with same method same as above two method
    0 references
    public float Add(float a, float b, float c, float d)
    {
        return a + b + c + d;

    }
}
```



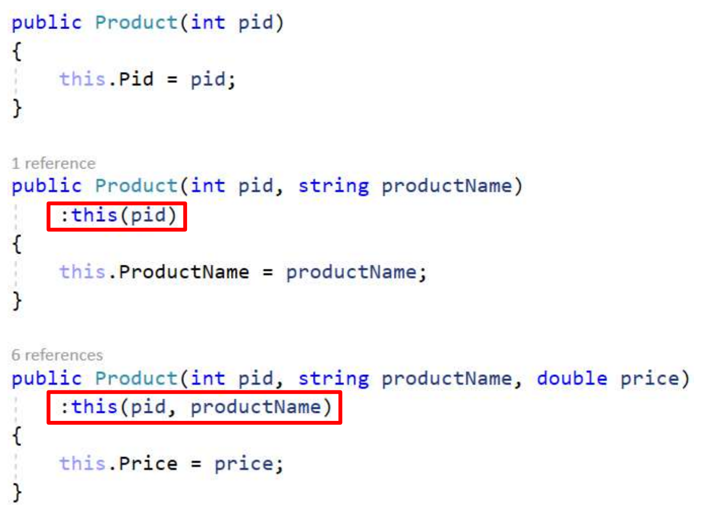
Constructor Chaining

- C# supports overloading of constructors, that means we can have constructors with different set of parameters.
- We can always make the call to one constructor from within the other.
- We will use ***this*** to refer to the constructor within same class.

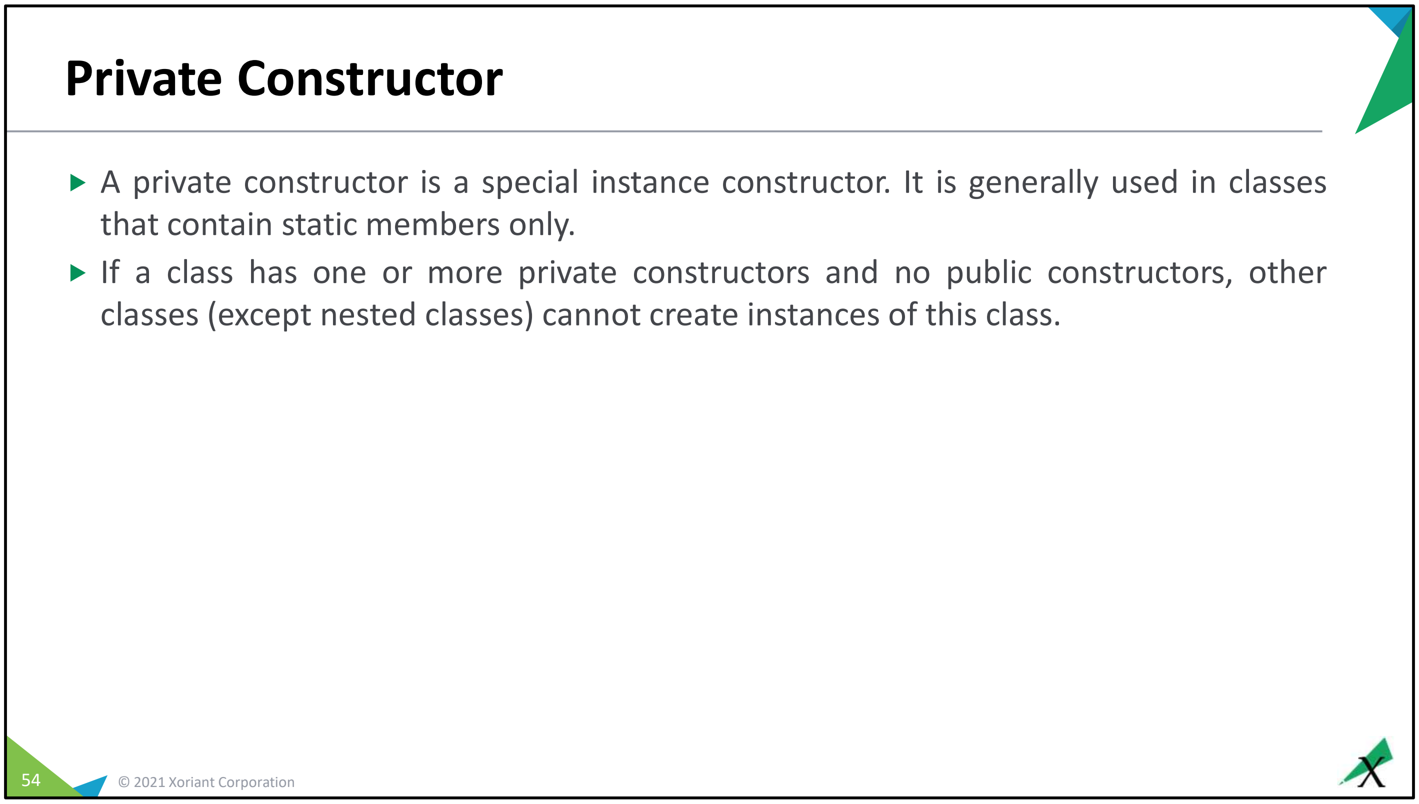
52

https://www.c-sharpcorner.com/UploadFile/825933/constructor-chaining-in-C-Sharp/

Constructor Chaining - Example



```
public Product(int pid)
{
    this.Pid = pid;
}

1 reference
public Product(int pid, string productName)
    :this(pid)
{
    this.ProductName = productName;
}

6 references
public Product(int pid, string productName, double price)
    :this(pid, productName)
{
    this.Price = price;
}
```



Private Constructor

- A private constructor is a special instance constructor. It is generally used in classes that contain static members only.
- If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances of this class.

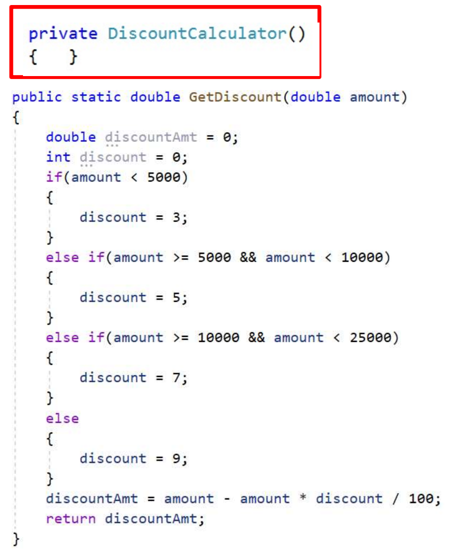
54

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/private-constructors

Private constructors are used to prevent creating instances of a class when there are no instance fields or methods, such as the Math class, or when a method is called to obtain an instance of a class. If all the methods in the class are static, consider making the complete class static.

Private Constructor - Example

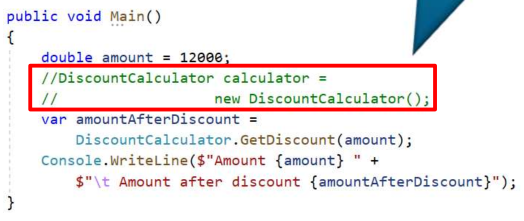
- DiscountCalculator.cs



```
private DiscountCalculator()
{   }
```

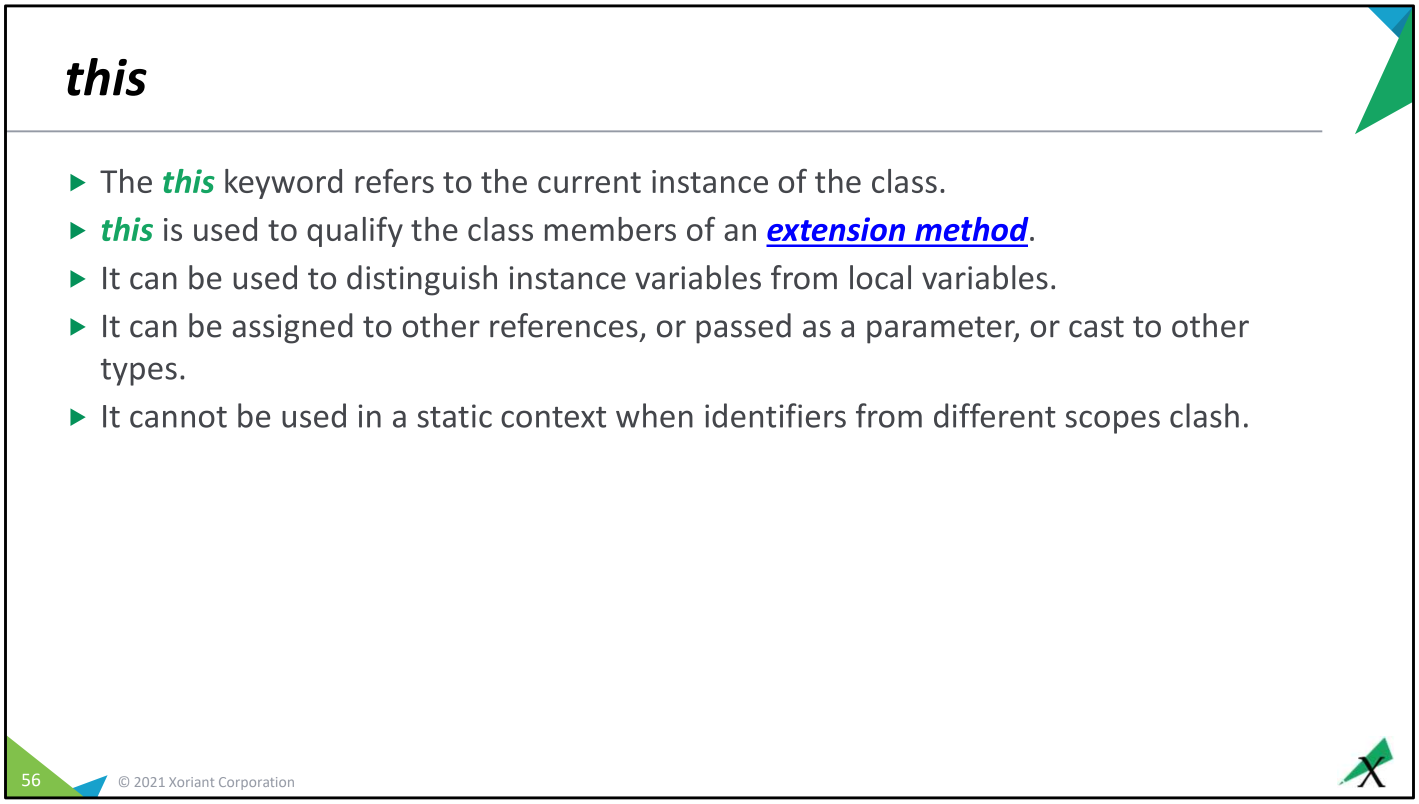
```
public static double GetDiscount(double amount)
{
    double discountAmt = 0;
    int discount = 0;
    if(amount < 5000)
    {
        discount = 3;
    }
    else if(amount >= 5000 && amount < 10000)
    {
        discount = 5;
    }
    else if(amount >= 10000 && amount < 25000)
    {
        discount = 7;
    }
    else
    {
        discount = 9;
    }
    discountAmt = amount - amount * discount / 100;
    return discountAmt;
}
```

DiscountCalculatorMain.cs



```
public void Main()
{
    double amount = 12000;
    //DiscountCalculator calculator =
    //                    new DiscountCalculator();
    var amountAfterDiscount =
        DiscountCalculator.GetDiscount(amount);
    Console.WriteLine($"Amount {amount} " +
        $"\t Amount after discount {amountAfterDiscount}");
}
```

Will give error as the constructor is private



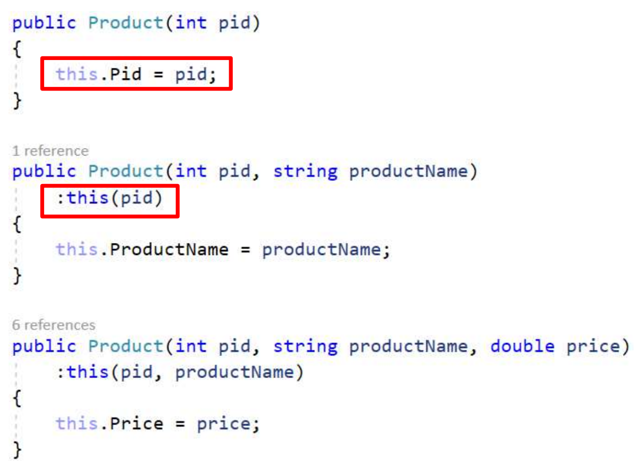
this

- The ***this*** keyword refers to the current instance of the class.
- ***this*** is used to qualify the class members of an ***extension method***.
- It can be used to distinguish instance variables from local variables.
- It can be assigned to other references, or passed as a parameter, or cast to other types.
- It cannot be used in a static context when identifiers from different scopes clash.

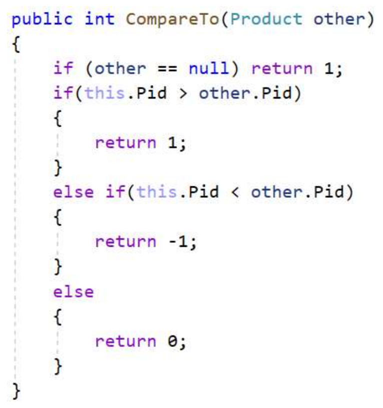
56

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/this

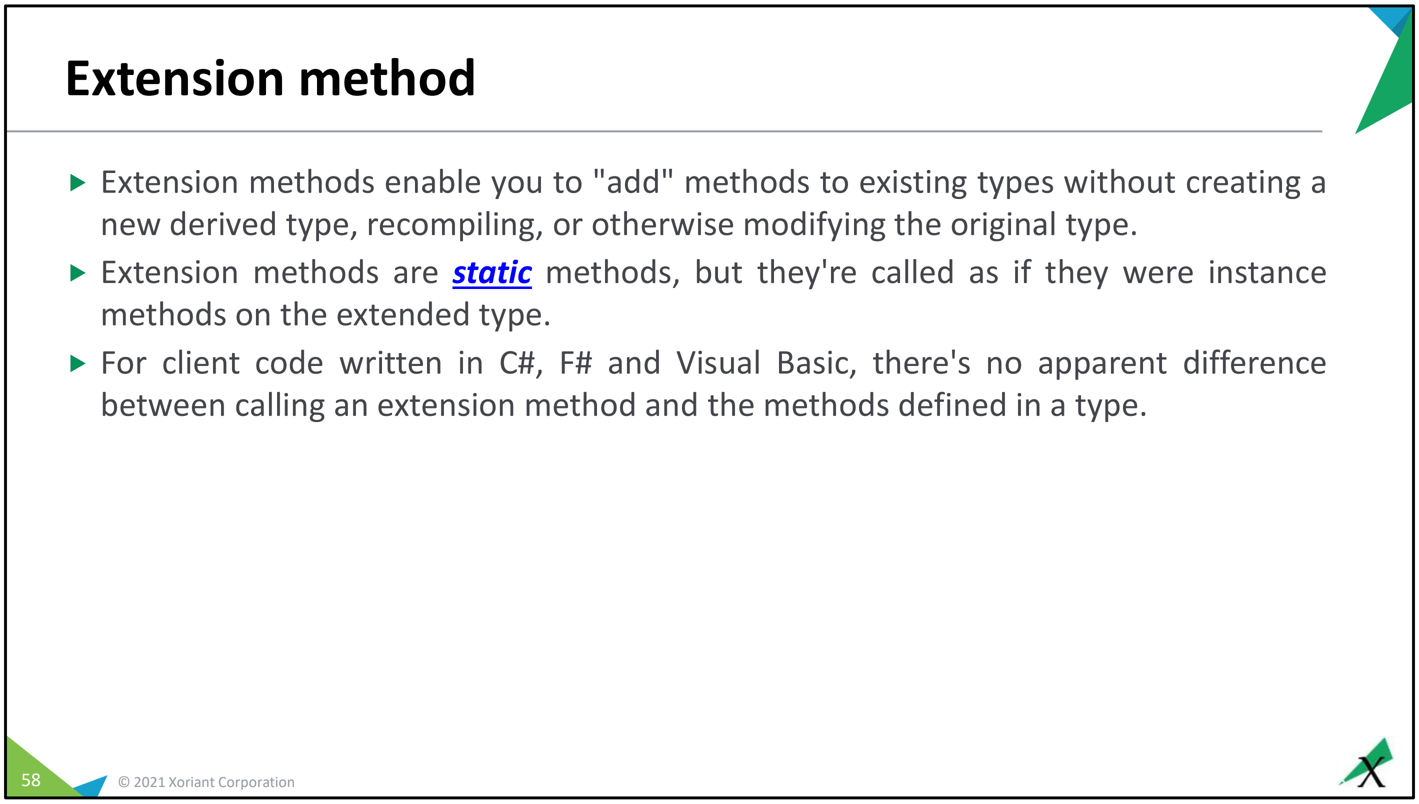
this - example



```
public Product(int pid)
{
    this.Pid = pid;
}

1 reference
public Product(int pid, string productName)
    :this(pid)
{
    this.ProductName = productName;
}

6 references
public Product(int pid, string productName, double price)
    :this(pid, productName)
{
    this.Price = price;
}
```



```
public int CompareTo(Product other)
{
    if (other == null) return 1;
    if(this.Pid > other.Pid)
    {
        return 1;
    }
    else if(this.Pid < other.Pid)
    {
        return -1;
    }
    else
    {
        return 0;
    }
}
```



Extension method

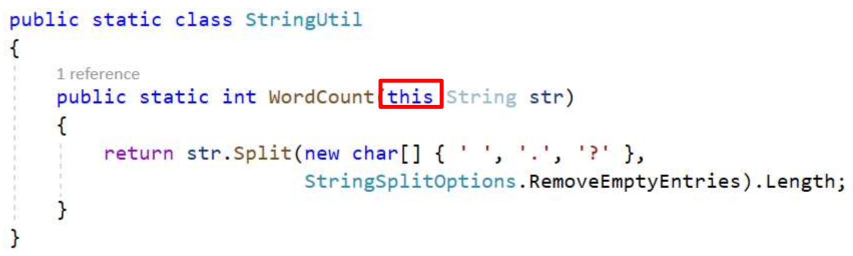
- Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.
- Extension methods are ***static*** methods, but they're called as if they were instance methods on the extended type.
- For client code written in C#, F# and Visual Basic, there's no apparent difference between calling an extension method and the methods defined in a type.

58

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

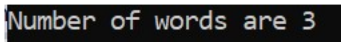
Extension method - Example

StringUtil.cs



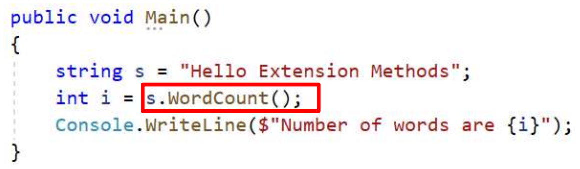
```
public static class StringUtil
{
    1 reference
    public static int WordCount(this String str)
    {
        return str.Split(new char[] { ' ', '.', '?' },
                         StringSplitOptions.RemoveEmptyEntries).Length;
    }
}
```

Output



```
Number of words are 3
```

Main.cs



```
public void Main()
{
    string s = "Hello Extension Methods";
    int i = s.WordCount();
    Console.WriteLine($"Number of words are {i}");
}
```

static

- The ***static*** modifier is a keyword.
- Use the static modifier to declare a static member, which belongs to the type itself rather than to a specific object.
- The static modifier can be used to declare static classes. In classes, interfaces, and structs, we may add the static modifier to fields, methods, properties, operators, events, and constructors.
- The static modifier can't be used with indexers or finalizers.

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static

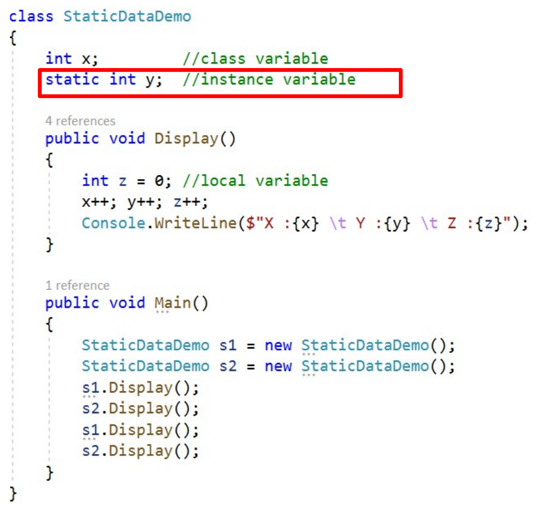
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members

Static Data

- 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 has been created.
- The static member is always accessed by the class name, not the instance name. Only one copy of a static member exists, regardless of how many instances of the class are created. Static methods and properties cannot access non-static fields and events in their containing type, and they cannot access an instance variable of any object unless it's explicitly passed in a method parameter.

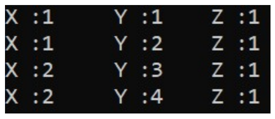
Static Data – Example

Static Methods



```
class StaticDataDemo
{
    int x;          //class variable
    static int y;   //instance variable

    4 references
    public void Display()
    {
        int z = 0; //local variable
        x++; y++; z++;
        Console.WriteLine($"X :{x} \t Y :{y} \t Z :{z}");
    }

    1 reference
    public void Main()
    {
        StaticDataDemo s1 = new StaticDataDemo();
        StaticDataDemo s2 = new StaticDataDemo();
        s1.Display();
        s2.Display();
        s1.Display();
        s2.Display();
    }
}
```

Output



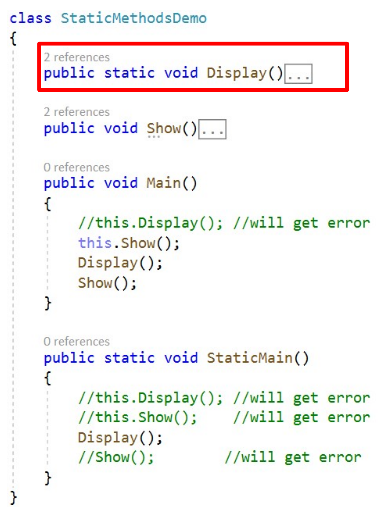
```
X :1        Y :1       Z :1
X :1        Y :2       Z :1
X :2        Y :3       Z :1
X :2        Y :4       Z :1
```

Static Methods

- Static methods can be overloaded but not overridden, because they belong to the class, and not to any instance of the class.

Static Methods - Example

StaticMethodDemo.cs

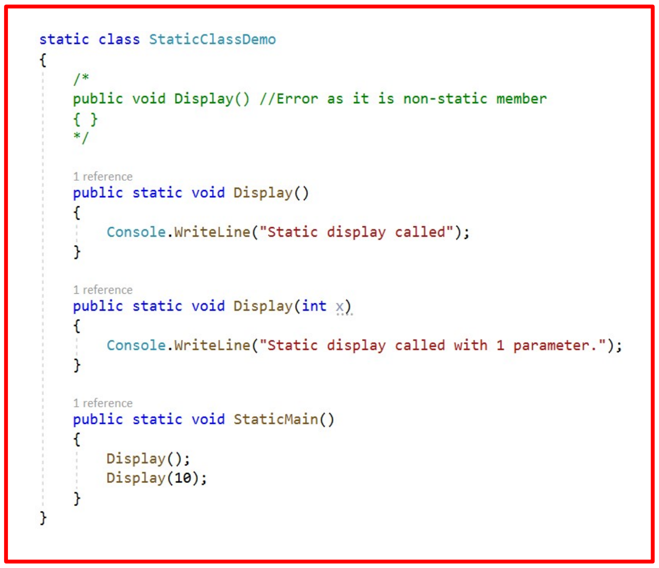


```
class StaticMethodsDemo
{
    2 references
    public static void Display()[...]

    2 references
    public void Show()[...]

    0 references
    public void Main()
    {
        //this.Display(); //will get error
        this.Show();
        Display();
        Show();
    }

    0 references
    public static void StaticMain()
    {
        //this.Display(); //will get error
        //this.Show();    //will get error
        Display();
        //Show();        //will get error
    }
}
```

Static Class

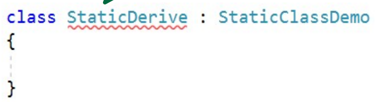
- A **static** class is basically the same as a non-static class, but there is one difference: a static class cannot be instantiated.
- There is no instance variable, you access the members of a static class by using the class name itself.
- Static classes are sealed and therefore cannot be inherited. They cannot inherit from any class except **Object**. Static classes cannot contain an instance constructor. However, they can contain a static constructor.

Static Class - Example

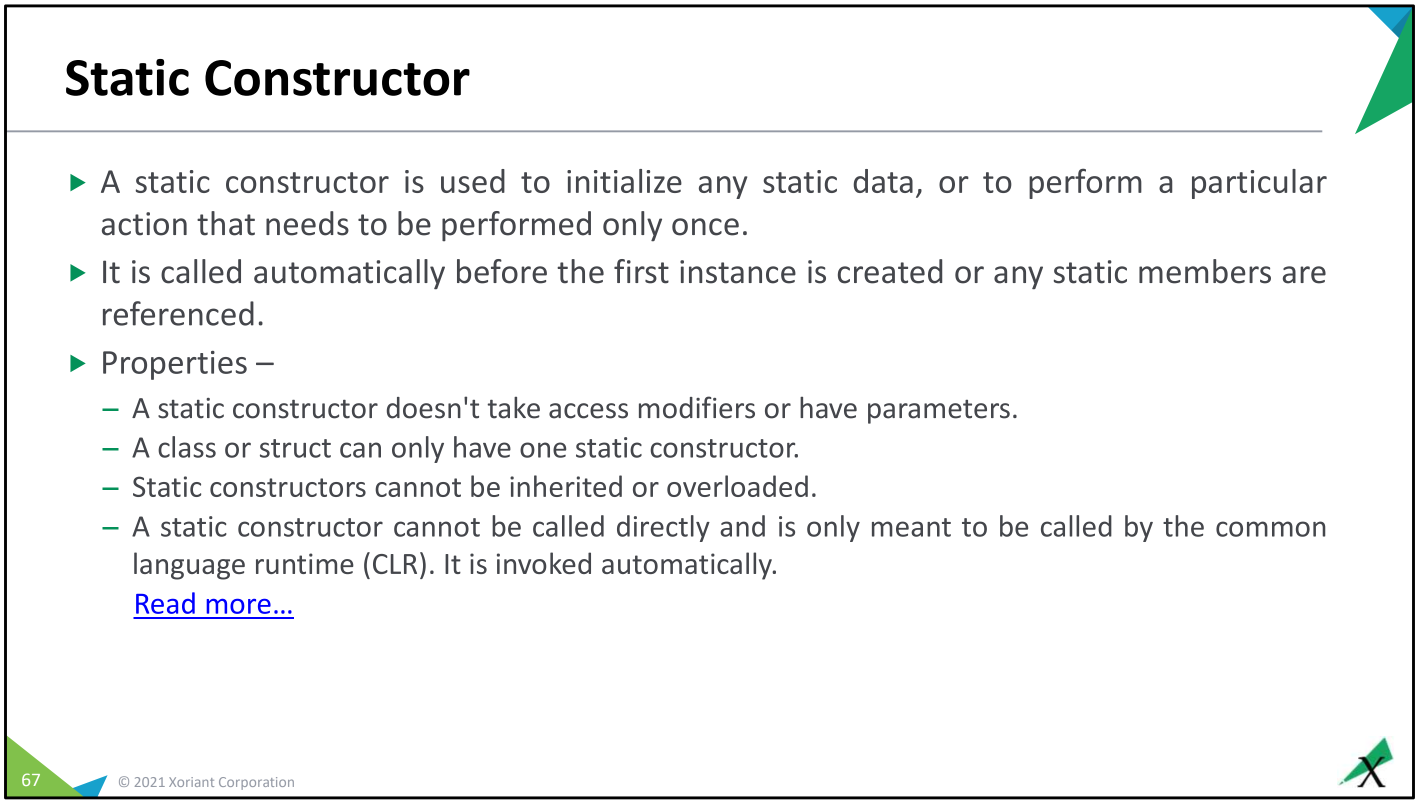


```
static class StaticClassDemo
{
    /*
    public void Display() //Error as it is non-static member
    { }
    */

    1 reference
    public static void Display()
    {
        Console.WriteLine("Static display called");
    }

    1 reference
    public static void Display(int x)
    {
        Console.WriteLine("Static display called with 1 parameter.");
    }

    1 reference
    public static void StaticMain()
    {
        Display();
        Display(10);
    }
}
```

Get error as we cannot derive from static class.



```
class StaticDerive : StaticClassDemo
{

}
```



Static Constructor

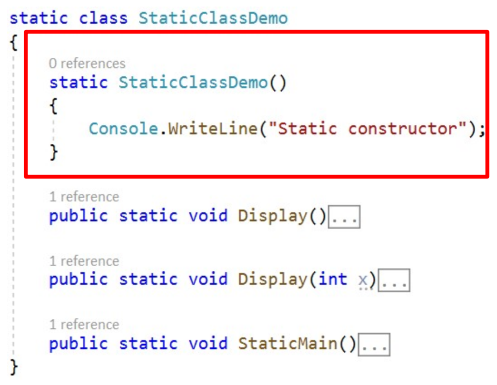
- A static constructor is used to initialize any static data, or to perform a particular action that needs to be performed only once.
- It is called automatically before the first instance is created or any static members are referenced.
- Properties –
  - A static constructor doesn't take access modifiers or have parameters.
  - A class or struct can only have one static constructor.
  - Static constructors cannot be inherited or overloaded.
  - A static constructor cannot be called directly and is only meant to be called by the common language runtime (CLR). It is invoked automatically.

  Read more…

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-constructors

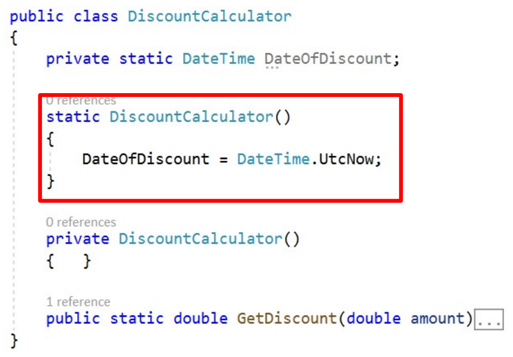
Static Constructor - Example

StaticClassDemo.cs



```
static class StaticClassDemo
{
    0 references
    static StaticClassDemo()
    {
        Console.WriteLine("Static constructor");
    }

    1 reference
    public static void Display()...

    1 reference
    public static void Display(int x)...

    1 reference
    public static void StaticMain()...
}
```

DiscountCalculator.cs



```
public class DiscountCalculator
{
    private static DateTime DateOfDiscount;

    0 references
    static DiscountCalculator()
    {
        DateOfDiscount = DateTime.UtcNow;
    }

    0 references
    private DiscountCalculator()
    {   }

    1 reference
    public static double GetDiscount(double amount)...
}
```

Case Study

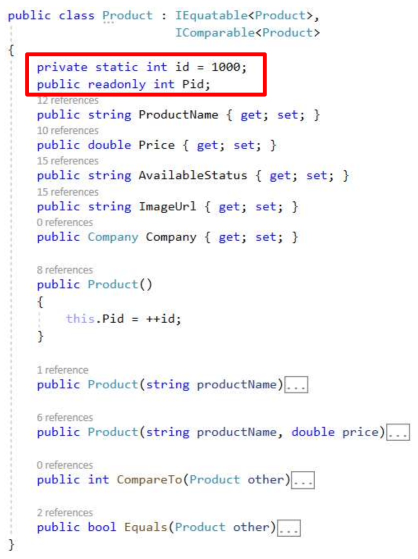
- As a portal **admin** of Shopon, I have to remember the ID’s. I want system to *auto generate ID’s*. I should not be *allowed to modify* this ID.

Thought

Thought

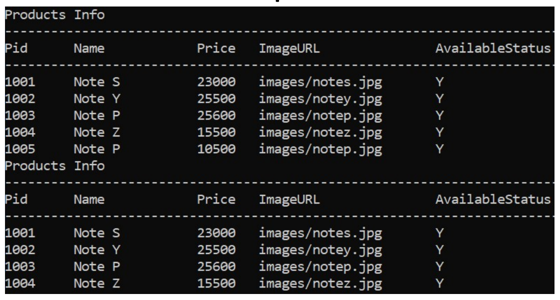
- ID’s are used in the system and user don’t have to remember it. These keys are ***surrogate key*** which has no business meaning.
- We can generate these keys using static data.
- We can make the data as **readonly** so that user is not allowed to modify the data.

Code



```
public class Product : IEquatable<Product>,
                       IComparable<Product>
{
    private static int id = 1000;
    public readonly int Pid;
    12 references
    public string ProductName { get; set; }
    10 references
    public double Price { get; set; }
    15 references
    public string AvailableStatus { get; set; }
    15 references
    public string ImageUrl { get; set; }
    0 references
    public Company Company { get; set; }

    8 references
    public Product()
    {
        this.Pid = ++id;
    }

    1 reference
    public Product(string productName)...

    6 references
    public Product(string productName, double price)...

    0 references
    public int CompareTo(Product other)...

    2 references
    public bool Equals(Product other)...
}
```

Output

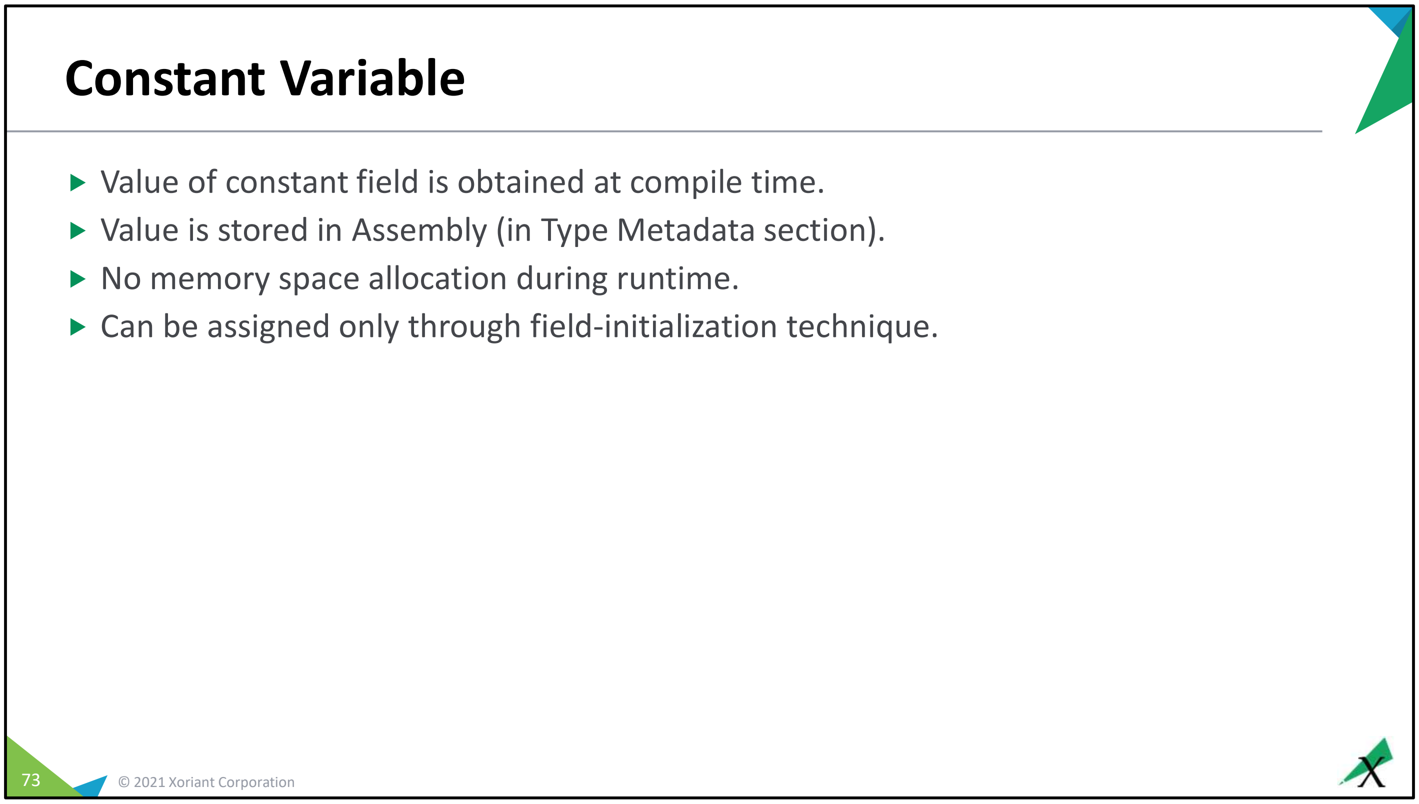


```
Products Info
---------------------------------------------------------------------------
Pid       Name             Price    ImageURL               AvailableStatus
---------------------------------------------------------------------------
1001      Note S           23000    images/notes.jpg       Y
1002      Note Y           25500    images/notey.jpg       Y
1003      Note P           25600    images/notep.jpg       Y
1004      Note Z           15500    images/notez.jpg       Y
1005      Note P           10500    images/notep.jpg       Y
Products Info
---------------------------------------------------------------------------
Pid       Name             Price    ImageURL               AvailableStatus
---------------------------------------------------------------------------
1001      Note S           23000    images/notes.jpg       Y
1002      Note Y           25500    images/notey.jpg       Y
1003      Note P           25600    images/notep.jpg       Y
1004      Note Z           15500    images/notez.jpg       Y
```

Readonly Variable

- Value of read-only field is obtained at run time.
- Value is not stored in assembly.
- Memory space is allocated during run time.
- Can be assigned through constructor and field-initialization technique.

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly



Constant Variable

- Value of constant field is obtained at compile time.
- Value is stored in Assembly (in Type Metadata section).
- No memory space allocation during runtime.
- Can be assigned only through field-initialization technique.

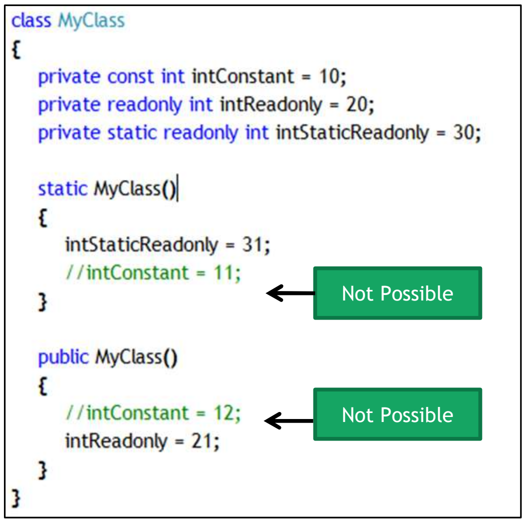
73

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constants

Initializing readonly and constant Fields

- Readonly fields must be initialized
  - Explicitly at their declaration in a variable initializer and inside an instance constructor
  - Static read only variable can be assigned explicitly inside static constructor and at their declaration in a variable initializer

- Constant variable must be initialized
  - Explicitly at their declaration in a variable initializer
  - Can't be explicitly initialized anywhere else



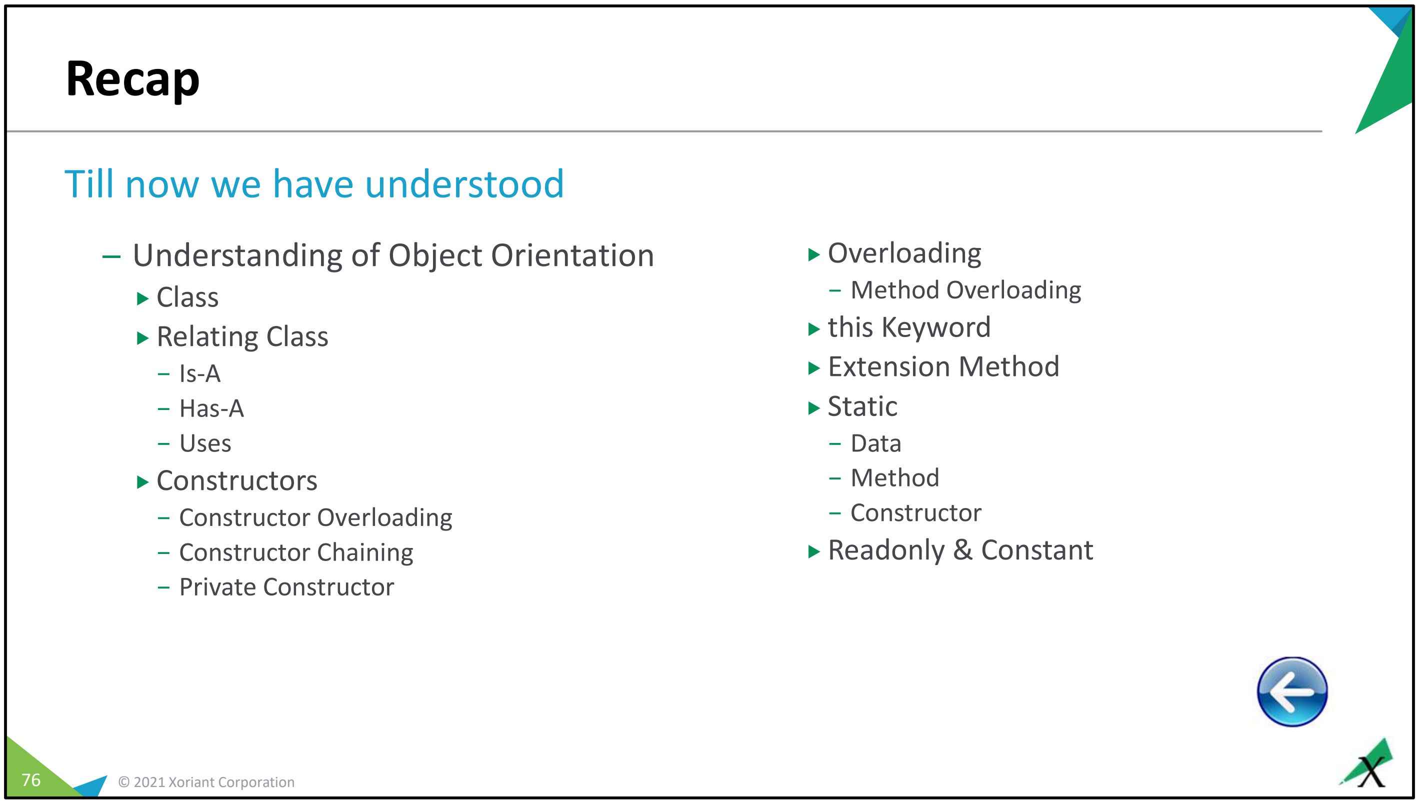
```
class MyClass
{
    private const int intConstant = 10;
    private readonly int intReadonly = 20;
    private static readonly int intStaticReadonly = 30;

    static MyClass()
    {
        intStaticReadonly = 31;
        //intConstant = 11;
    }

    public MyClass()
    {
        //intConstant = 12;
        intReadonly = 21;
    }
}
```

Next Step

Recap

Useful links

Thank you



Recap

Till now we have understood

- Understanding of Object Orientation
  - Class
  - Relating Class
    - Is-A
    - Has-A
    - Uses
  - Constructors
    - Constructor Overloading
    - Constructor Chaining
    - Private Constructor
- Overloading
  - Method Overloading
- this Keyword
- Extension Method
- Static
  - Data
  - Method
  - Constructor
- Readonly & Constant

Useful Links

- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-constructors
- https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/member-overloading
- https://www.c-sharpcorner.com/UploadFile/825933/constructor-chaining-in-C-Sharp/
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/private-constructors
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/this
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-constructors
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constants

US – CORPORATE HEADQUARTERS

1248 Reamwood Avenue,
Sunnyvale, CA 94089
Phone: (408) 743 4400

343 Thornall St 720
Edison, NJ 08837
Phone: (732) 395 6900

UK

57 Rathbone Place,
4th Floor, Holden House,
London, W1T 1JU , UK

89 Worship Street Shoreditch,
London EC2A 2BF, UK
Phone: (44) 2079 938 955

INDIA

Mumbai
4th Floor, Nomura
Powai , Mumbai 400 076
Phone: +91 (22) 3051 1000

Pune
5th floor, Amar Paradigm Baner Road
Baner, Pune 411 045
Phone: +91 (20) 6604 6000

Bangalore
4th Floor, Kabra Excelsior,
80 Feet Main Road, Koramangala 1st Block,
Bengaluru (Bangalore) 560034
Phone: +91 (80) 4666 1666

www.xoriant.com