25/12/2023

* servlet dispatcher is responsible for doget, dopost in servlet.

PAGE No.
DATE / /

* Publisher - Subscribe Architecture :
  - used by technology named 'Kafka'.

Interview Q.
* Tell me How kafka works :
  - ex:- postman, netflix
  - explain with delegates
  - give example of the Banking event driven application

* Tell me : messaging architecture ?
* what is consumer and provider ?

* There are 23 Design Patterns :
  ex:- 1] Bridge Pattern
  2] Adaptor Pattern
  3] Singleton pattern -> only 1 obj. is created

- To develop software past we use design patterns :

* Rules of Design Pattern : [for Singleton Pattern] :
  1] constructor private and variable declared as static
  2] obj. is always created using static method
  3] if obj. is null create new obj if not null return reference of existing objection

Note:- MVC is everywhere :
  - Spring MVC
  - ASP.Net MVC
  - Django

VVIMP

Garbage Collector and IDisposable Interface

sir's

* Github → .net → Day 4 → DotnetTypesSolution

* certain collective data member of one class in one file and certain members in another file having same class, That's why we use partial classes.

* multiple .cs files → multiple partial classes

* GC.SuppressFinalize() → Garbage collector static functions
* GC.WaitForPendingFinalizers() → we can do efficient memory management by using these
* GC.Collect()

* In Java there is no destructor but we have to override the finalize() method but why?

→ Finalize() → declared in object class.
↳ Garbage collector always calls before object is getting removed from the heap

→ Garbage collector → works on Mark & Sweep strategy.
→ is a kind of daemon thread responsible for Automatic memory management
→ it find all the objects in the heap whose reference are lost or orphan orphan objects it marks that object this concept is called as (marking object). then it goes back and comes again and sweeps the object.

- Garbage collector works on mark and sweep algorithms. works in 2 iterations
- but before sweeping the objects it will call finalize() in java destructor in C# and then sweeps the object. this concept is called Indeterministic finalization.
- but we don't know when it will call coz when GC is gonna active we don't know.

- Role of GC is to keep heap clean from unwanted objects.
- It always keeps track on non-reference obj
- on 1st iteration it will mark the objects. and in 2nd iteration it will sweep the objects but before sweeping it becomes active and calls the finalize() or destructor. This concept is called as Indeterministic finalization

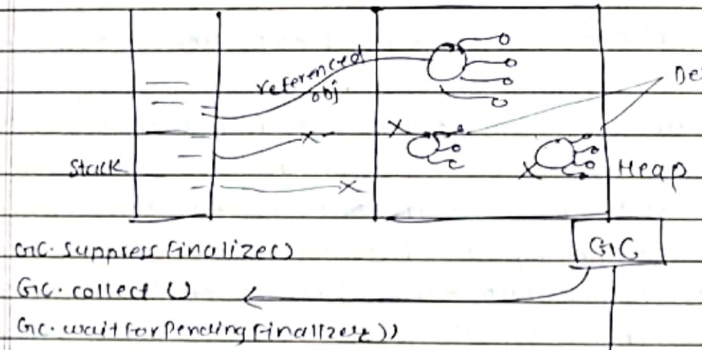
VVIMP

* IDisposable interface:
* Resource sharing problem will occur if we write our code of disclose/release in finalize();
So, for that in our code write IDisposable Interface and override Dispose().
So, what happens due to this is that we don't have to wait for GC to collect the obj. but we explicitly give it to dispose().
So, dispose() will call exactly after the curly {} brackets are closed and don't wait for GC to collect it.
This concept is called as Deterministic finalization
- Primary Thread will call the dispose function and collect the object.

Qs. Diff^n betn GC.SuppressFinalize and GC.WaitForPendingFinalizers?

Qs. How we reduce the work of Garbage collector

Qs. What is IDisposable Interface?



- Automatic Memory Mgmt.
- Mark & Sweep

- Disposable Interface is only in C#1 but not in Java.
- Dispose() → used Primary Thread for cleaning the objects.
- Destructor() → uses Garbage Collector for cleaning/sweeping the objects.
- Dispose() → to dispose shared resource easily for others to use.

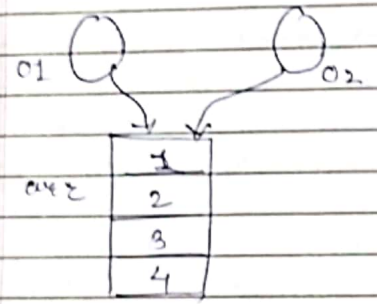
* Garbage Collector is a Singleton class.
  - only one object is there in the whole program
- GC.SuppressFinalize() → doesn't restrict/don't call the Finalize() of Garbage collector coz we've already cleaned/sweeped the object.

* ICloneable Interface:
  - override clone function/Method.

- If u want to achieve or make your obj. capable to deepcopy you have to implement an interface named ICloneable.
- To create replica of object
  → object O2 = O1
  - Here references are copied not objects it is called as Shallow Copy.
  But, to copy object:
  → object O2 = O1.clone()
  - Here object is copied and created new object this is called as Deep Copy.

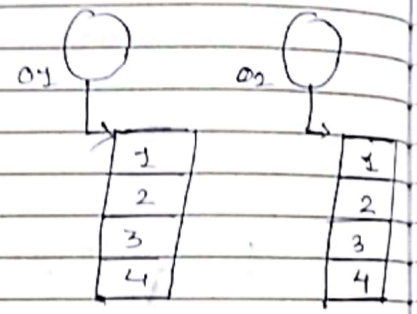
* Shallow Copy

Obj o2 = o1;



* Deep Copy.

obj o2 = o1.clone();



- we define interface coz we need to define functions, to increase capability of object. from clone we got this into about interface.

- Here class → is service provider which implements publisher
  interface → is publisher
  & consumers → consume it.

* Now, → .NET/Day 4/ DotnetTypessolution/

- Reproductive class is concrete class → means whole obj is created/produce object.

- Abstract class → non-reproductive objects class
  & we want polymorphism alongwith we don't want to create object

* Virtual:

- To create object as well as to achieve polymorphism so, the class must be concrete and method must be virtual
  - for virtual func() we need not to override it or we don't override it.
  - by making

- new virtual → not overriding it
  → new function creation with same name
  → is called as shadowing technique
  - I want to block polymorphism
  - I want to block overriding & to start new legacy of overriding for the upcoming classes

* overriding the basic behaviour of a fun^n in a derived class → This is Polymorphism.

* without inheritance Polymorphism is not possible.

notes

→ Property → has getter, setters → are called smart fields
↳ always hide data member individually

→ indexer: encapsulates collection inside it

Collection Framework:

•NFT / Day 4 / DotnetTypesSolution / CSKeywordsApp

INDEXERS:

→ are called smart Array

- In Day 6 → cmd → mrdir cssolution
  → cd cssolution
  → dotnet new console -o CSFeaturedApp
  → dotnet sln add
- CSFeaturedApp
  ↳ Books.cs
  ↳ Programs.cs
  ↳ using library;
  ↳ Books shelf = new Books();
  shelf.(all default methods from obj. class)

cssolution → dotnet run --project CSFeaturedApp\CSFeaturedApp.csproj

- to access the private data member in our case Books class → Array of strings, so:- i.e title array in our case.

(1) by using getters
- GetTitle()

(2) whenever u want your obj. to behave like an array for that in books class,
- public string this[int index]{
  get { return title[index]; }
  set { title[index] = value; }
}

Here, we're setting indexers

This is indexers syntax

notes

* Structure → value type → kept on stack
* By default every class / structure is inherited from object class except C++.

- If you want to encapsulate collection implement indexes with in a class.
- we can write only 1 indexer in a class but multiple properties are written in a class.
- Indexer hides collection inside property hides data members.

note:- 1] on right side of "=" Assig. operator getters are called
ex:- shelf[3];
string bookTitle = shelf[3];

2] on left side of "=" Assignment operator setters are called:
ex:- shelf[3] = "Sepian";

- In C# we can have protected as an access specifier. Protected helps to achieve datamembers to only child class.
- In .net we can define a variable of
  - class, abstract class, interface, enum, struct.
- to give meaningful values we can always use enum → which is a value type CTS variable.

[nullable type:-]

-> we can't print unassignable variable in C#

↳ like, int count;
Console.WL(count); // error.

-> int? count = null; // nullable type
↳ something is not pointing to

[OR]

Nullable<int> amount = null;

-> int marks = 0; // this is not null, this is integer type.
↳ something is pointing to

- To maintain integrity of the variable we have to declare it nullable type
- Always make a variable null and then assign it
- To avoid memory problems for your variable it is better to initialize your variable with null.
- To avoid dangling pointers (pointer which is not pointing to anything).

ex:- of वस्तु -> बोतल को रखो

Reflection: // In Java we uses GetClass()

- GetType() ← both are inherited from object class.
  - ↳ returns type 'T'
  - ↳ we can introspect (आत्मनिरीक्षण) the object.
  - ↳ by using reference of object we can do introspect.
  - ↳ whenever u want to read the metadata of the object u will go for Reflection type and to achieve it u can go like,

obj.GetType();

Reflection, Serialization (File I/O), Multithreading, networking are the pillars of every technology's structure.

This is real programming.

- public override void Draw() {
  Type t = this.GetType();
  Console.WL("Type = " + t.Name);
  Console.WL("(" + this.StartPoint + "), (" + endpoint . . . width . . . color);
  }
}

Github -> .NET/Day 3 / GraphicsSolution/OrderProcessing.

Explicit Interface Inheritance:

-> inside IOrderDetails (interface): 2 func()
- IcustomerDetails (''): 3 func()

-> In both of above ShowDetails() is common.

* CRM -> Customer Relationship Management in E-commerce

// Explicit Interface Inheritance

-> Void ICustomerDetails.ShowDetails() {
    Console.WriteLine();
}

-> Void IOrderDetails.ShowDetails() {
    Console.WriteLine();
}

* To read something from a console:
    -> Console.ReadLine();

* Now, CSFeaturesApp -> UtilityManager.cs
    -> namespace Utility
    -> using a keyword 'ref' for a variable, so here we r passing reference of a variable so in our swap fun() we r only swapping reference but not values.
    - Ex:- we haven't moved objects/chairs but we moved/swapped people/reference

* [Ref Keyword]

* [out Keyword]: No need to write multiple funs for radius as well as for circumference
    -> public void Calculate(int radius, float area, float circumference)
    {
        float area = (3.14) * radius * radius;
        float circumference = 2 * (3.14) * radius;
    }

public static void Calculate(int radius, out float area, out float circumference) {
    area = (3.14F) * r * r;
    circumference = 2 * (3.14F) * r;

- Here we can return multiple values as we want.

- To reduce uplifting we have to use 'Out' Keyword.

* [Params Keyword]: having a single behaviour for multiple calls.

public static void ShowNames(params string[] names) {
    foreach (string name in names) {
        Console.WriteLine(name);
    }
}

->in program.cs
    UtilityManager.ShowNames("Ravi");
    UtilityManager.ShowNames("A", "B", "C");

- every data must be of type string for above ex:- but for below ex it is type object.

- we can't done overloading here, but due to type of object we are changing only signature:
    -ex:
        UtilityManager.ShowNames(34, true, 56.5f, "A");
    -> we can do like this also.

'Sealed' in .NET is equivalent to 'final' in Java

* CVQ Library → for image processing used in C# to make face recognition app.

* Extension Method:
- You can extend a class with diff. behaviour.
- extension can be done without inheritance.

now, MathEngine.cs
↳ namespace Mathematics;
public sealed class MathEngine {
public int Addition (int op1, int op2) {
return op1 + op2;
}
public int subtraction (int op1, int op2) {
return op1 - op2;
}

in, Utility Manager.cs
↳ static int division();
↳ static int multiplication();

now, in Program.cs
↳ MathEngine engine = new MathEngine();
int result1 = engine.Addition(56, 56);
int result2 = Sub.
int result3 = Mult.
result4 = Div.
// it is not allowing now due to MathEngine class have this

and also we can't inherit it coz class MathEngine is sealed so, for that we have to adapth the multiplication and division class so,

① public static int division (this MathEngine e, int op1, int op2) {
return op1 / op2;
}

② public static int multiplication (this MathEngine e, int op1, int op2) {
return op1 * op2;
}

— x —

Two-D Arrays
→ used in Graphics
- given by int [,] mtrx = new int [2, 3]

Jagged Arrays
→ Array of Array
int [] [] mtrx = new int [2] [4]

Multidimensional Arrays [Jagged Arrays]
→ int [,] mtrx = new int [2, 3]

#