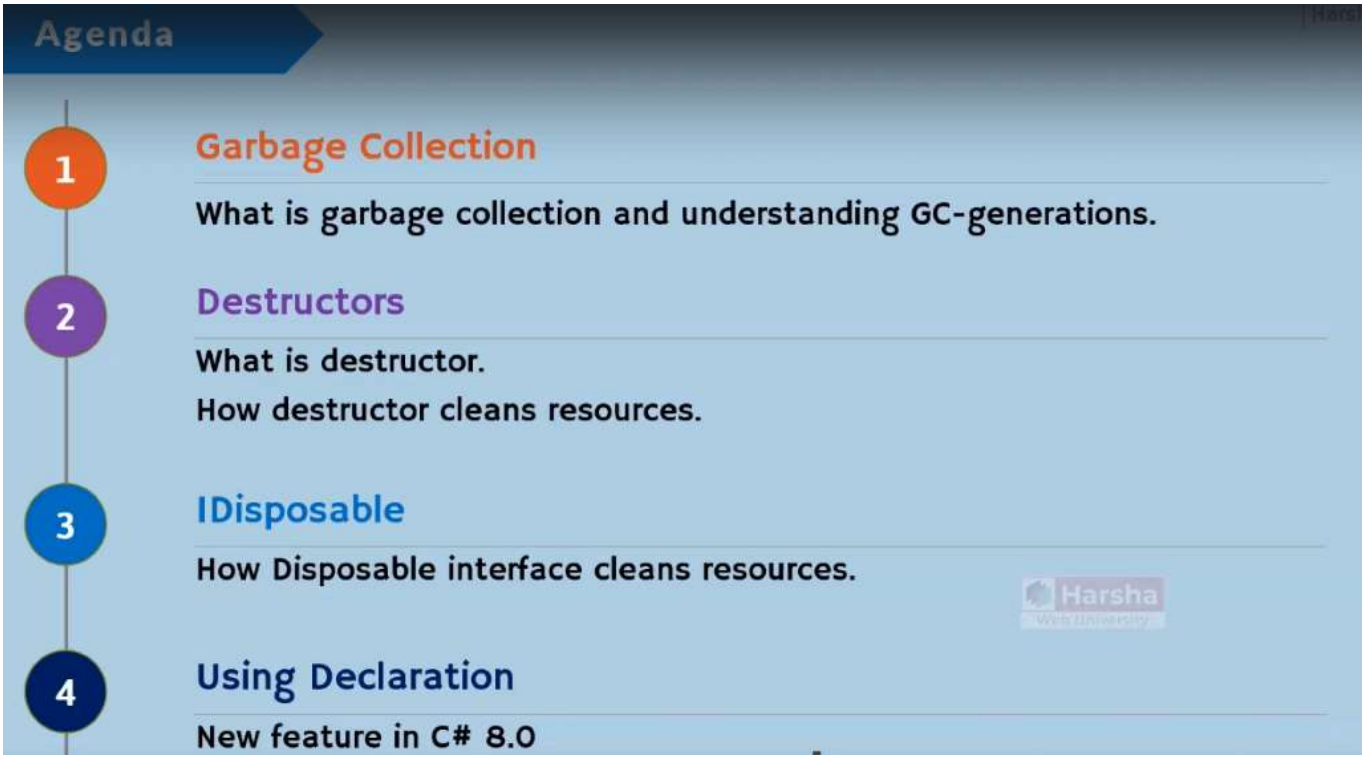
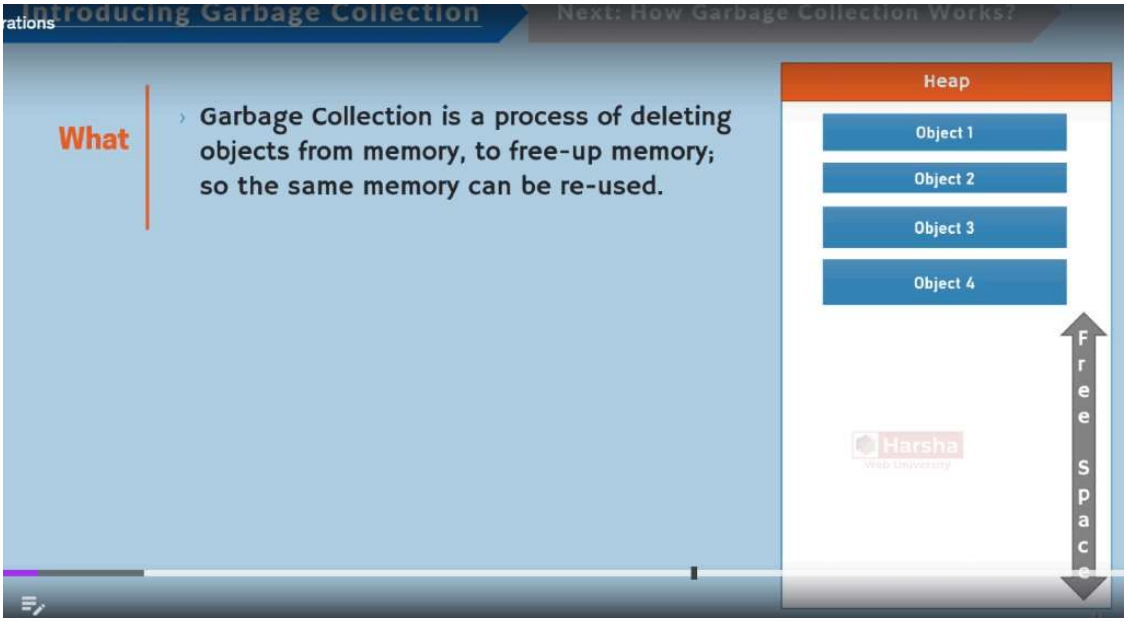
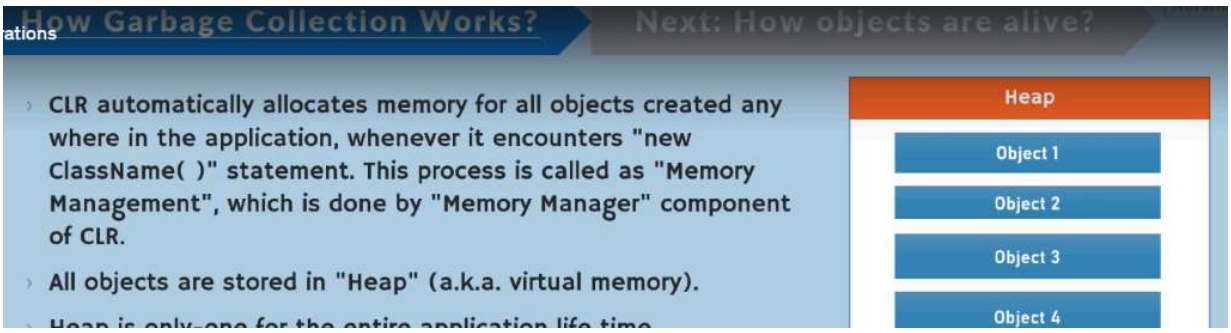
Section 19 : GC, Destructors, IDisposable

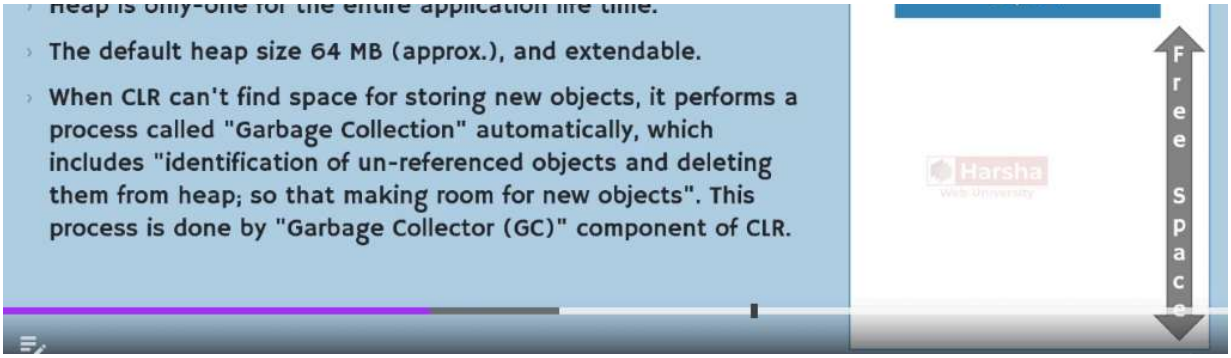
Monday, July 15, 2024 3:39 PM

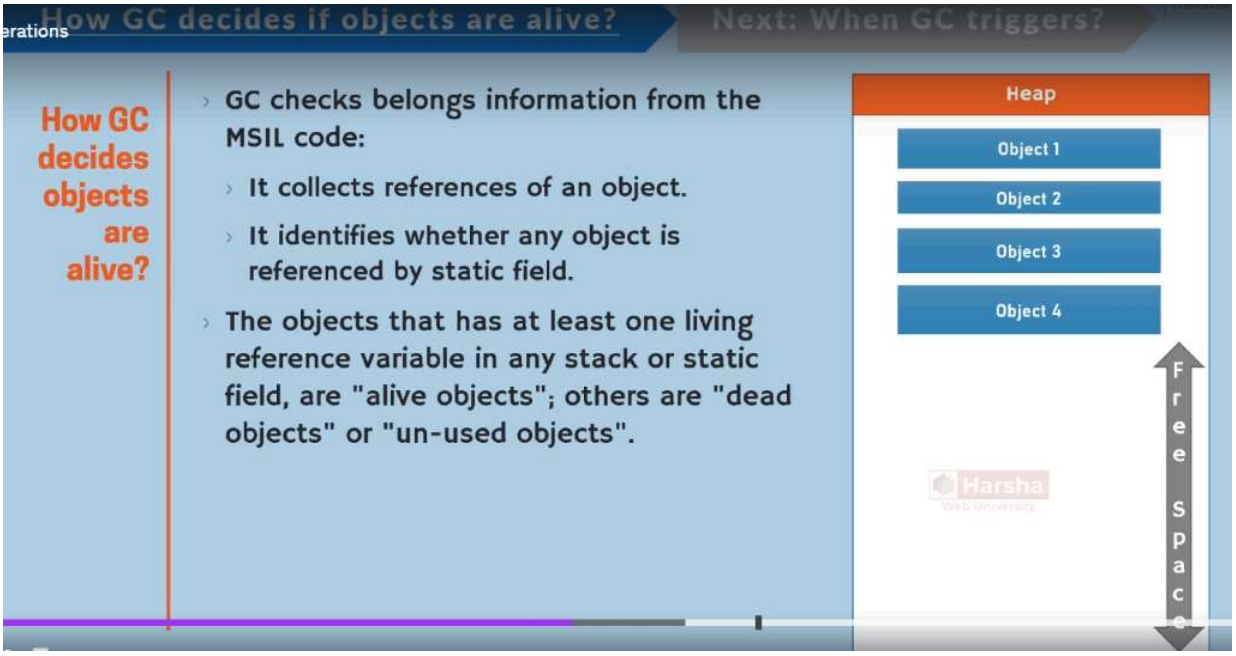


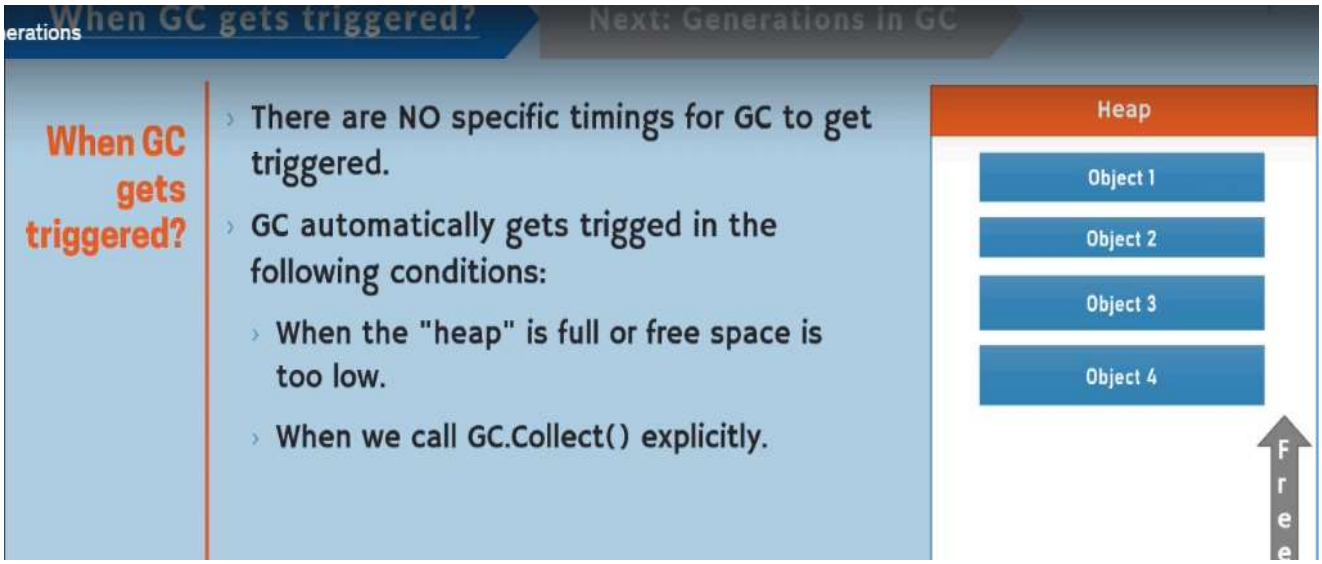
Garbage Collection is a process of deleting the unwanted objects from the memory after the usage.

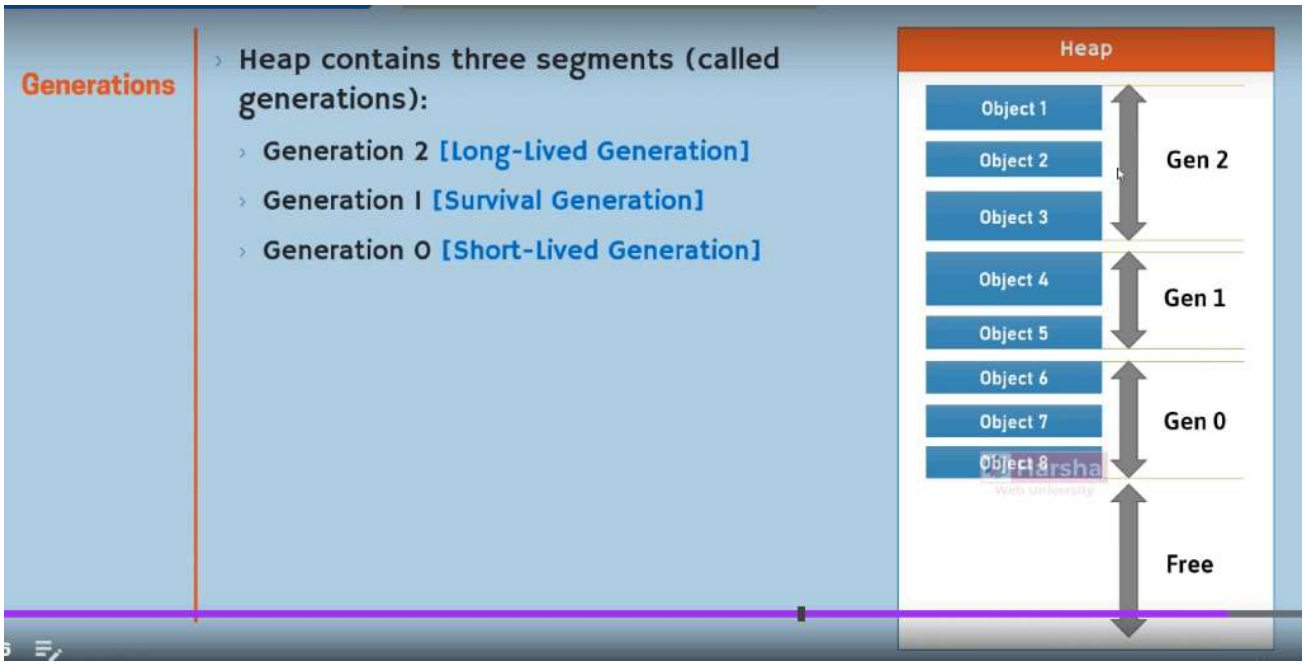


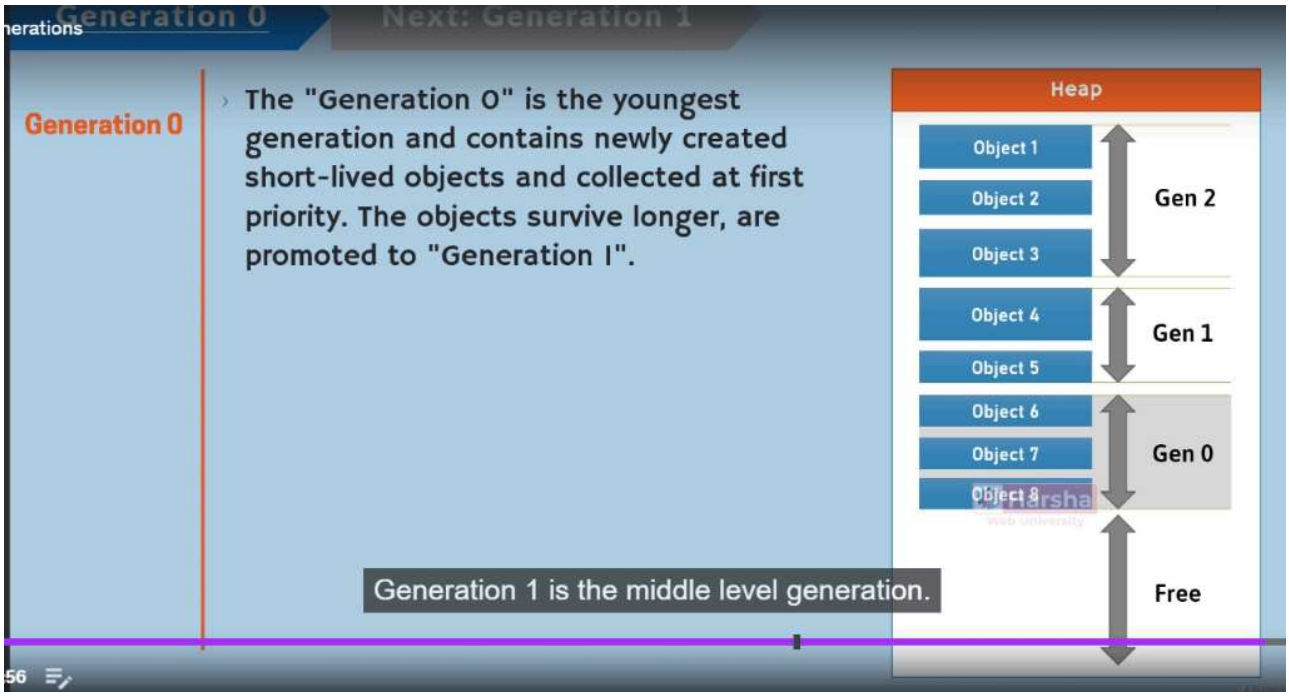


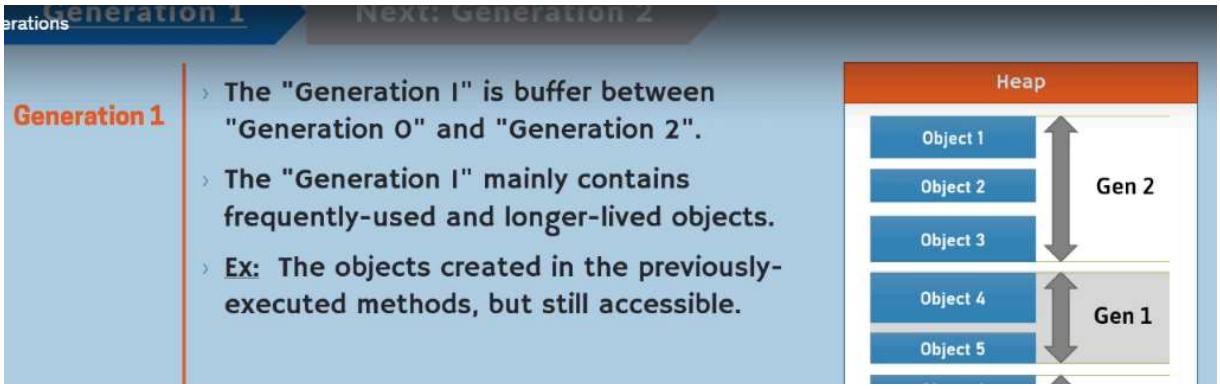

The default heap size 64 MB (approx.), and extendable.
When CLR can't find space for storing new objects, it performs a process called "Garbage Collection" automatically, which includes "identification of un-referenced objects and deleting them from heap; so that making room for new objects". This process is done by "Garbage Collector (GC)" component of CLR.
Free Space
Harsha

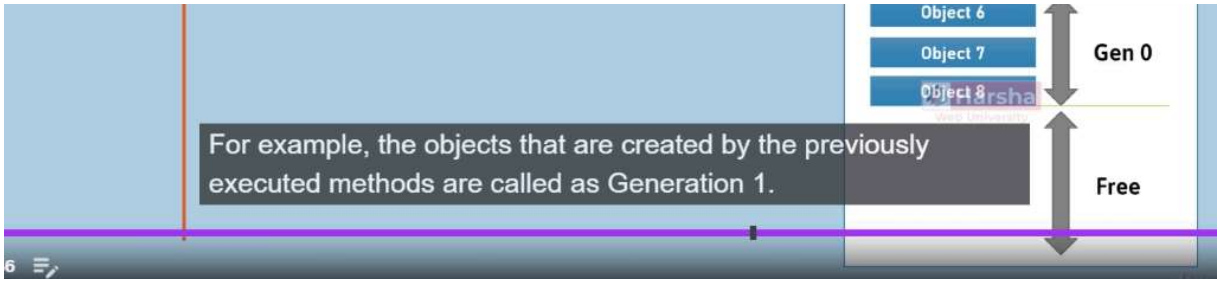

How GC decides if objects are alive?
Next: When GC triggers?
How GC decides objects are alive?
GC checks belongs information from the MSIL code:
It collects references of an object.
It identifies whether any object is referenced by static field.
The objects that has at least one living reference variable in any stack or static field, are "alive objects"; others are "dead objects" or "un-used objects".
Heap
Object 1
Object 2
Object 3
Object 4
Free Space
Harsha

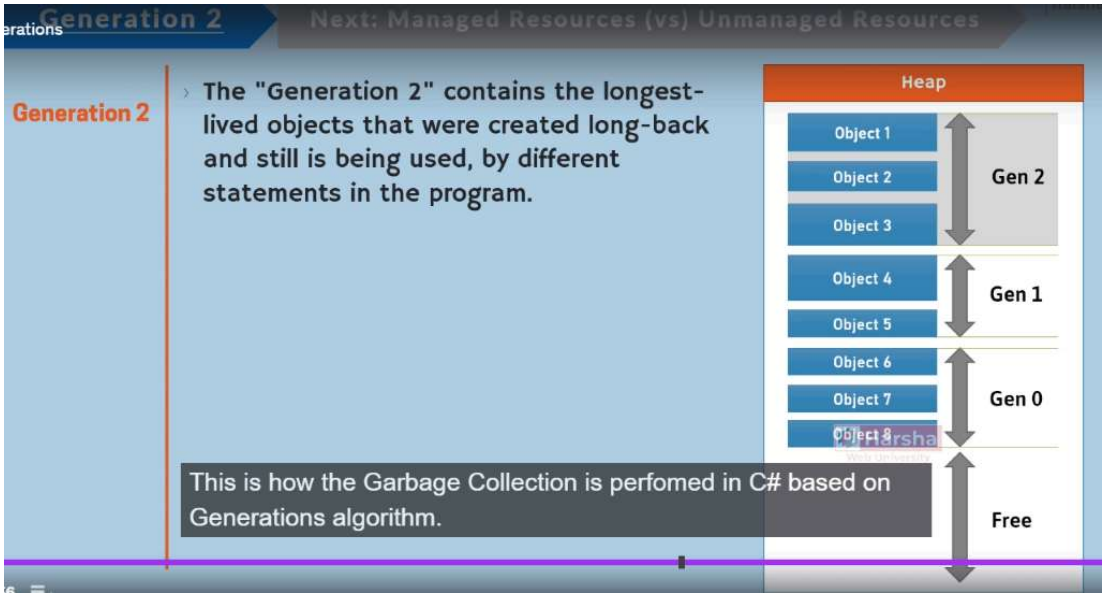

When GC gets triggered?
Next: Generations in GC
When GC gets triggered?
There are NO specific timings for GC to get triggered.
GC automatically gets trigged in the following conditions:
When the "heap" is full or free space is too low.
When we call GC.Collect() explicitly.
Heap
Object 1
Object 2
Object 3
Object 4


Generations
Heap contains three segments (called generations):
Generation 2 [Long-Lived Generation]
Generation 1 [Survival Generation]
Generation 0 [Short-Lived Generation]
Heap
Object 1
Object 2
Object 3
Object 4
Object 5
Object 6
Object 7
Object 8
Gen 2
Gen 1
Gen 0
Free


Generation 0
Next: Generation 1
Generation 0
The "Generation 0" is the youngest generation and contains newly created short-lived objects and collected at first priority. The objects survive longer, are promoted to "Generation 1".
Heap
Object 1
Object 2
Object 3
Object 4
Object 5
Object 6
Object 7
Object 8
Gen 2
Gen 1
Gen 0
Free
Generation 1 is the middle level generation.

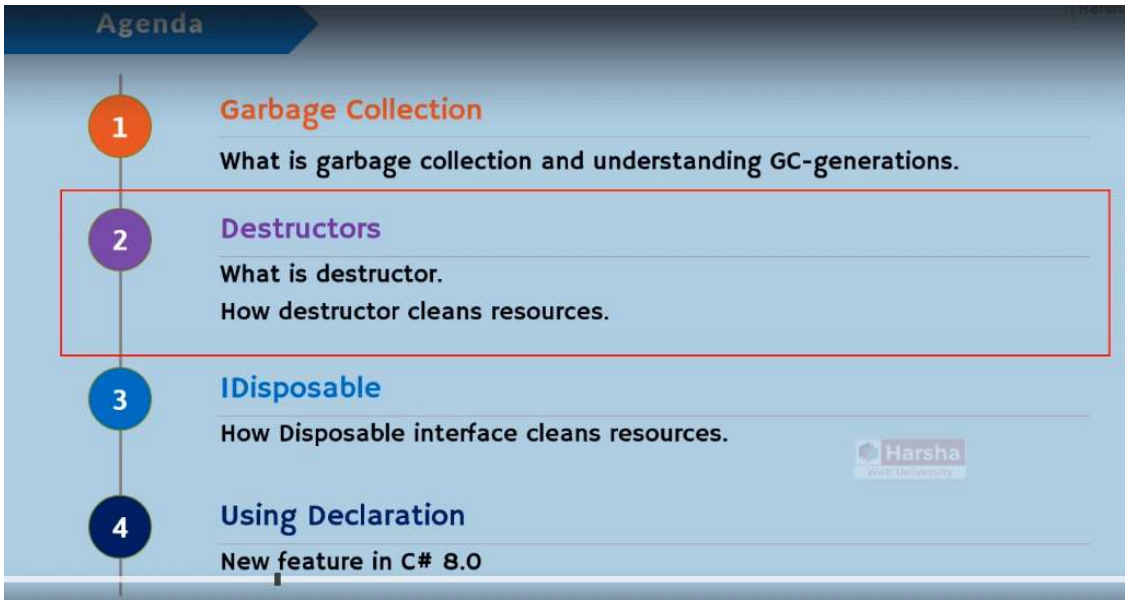

Generation 1
Next: Generation 2
Generation 1
The "Generation 1" is buffer between "Generation 0" and "Generation 2".
The "Generation 1" mainly contains frequently-used and longer-lived objects.
Ex: The objects created in the previously-executed methods, but still accessible.
Heap
Object 1
Object 2
Object 3
Object 4
Object 5
Gen 2
Gen 1

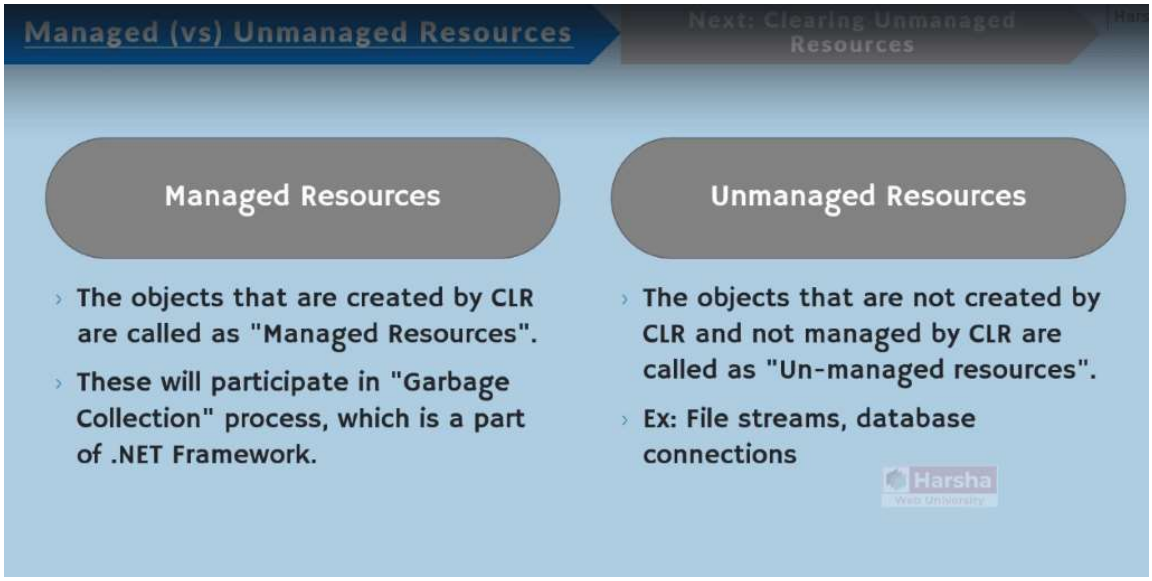




Generation 2 objects are subject to delete at any time next time when the Garbage Collection is performed.

This is how the Garbage Collection is performed in C# based on Generations algorithm. This is also called as 'Mark and Compact algorithm'.



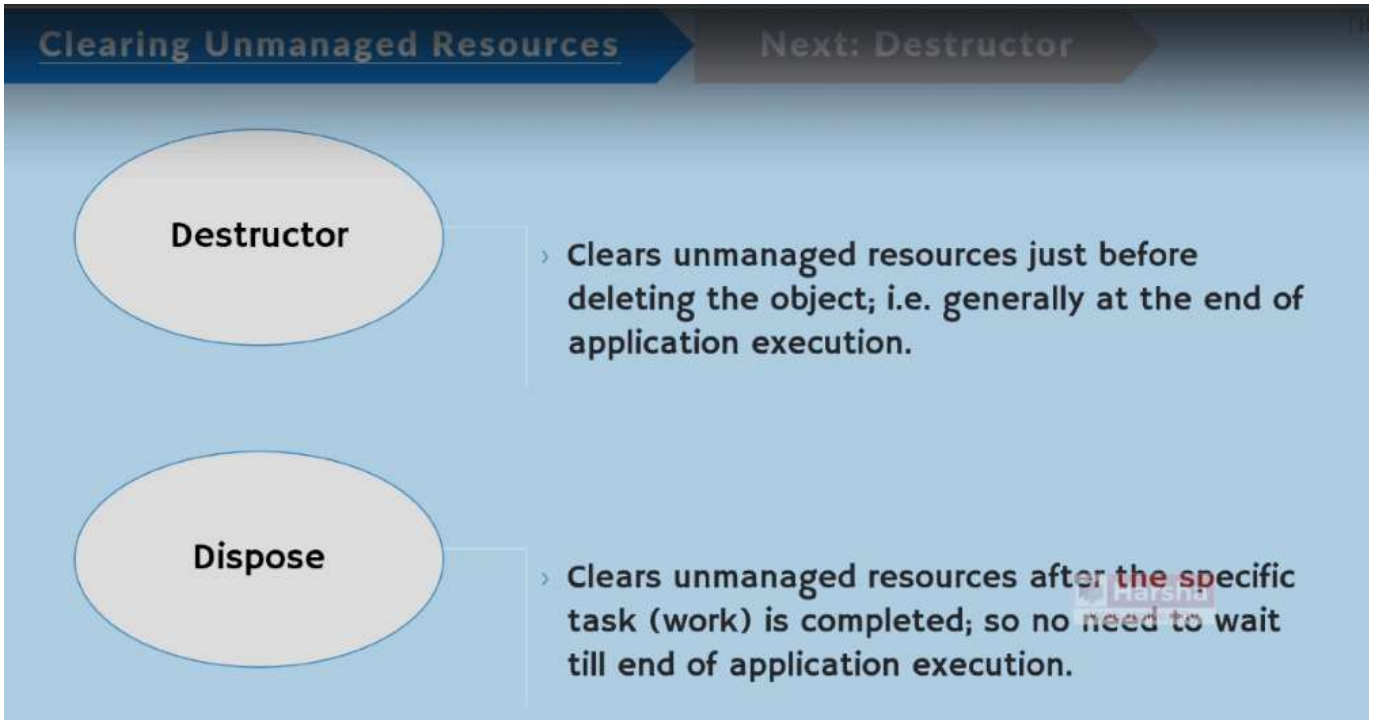


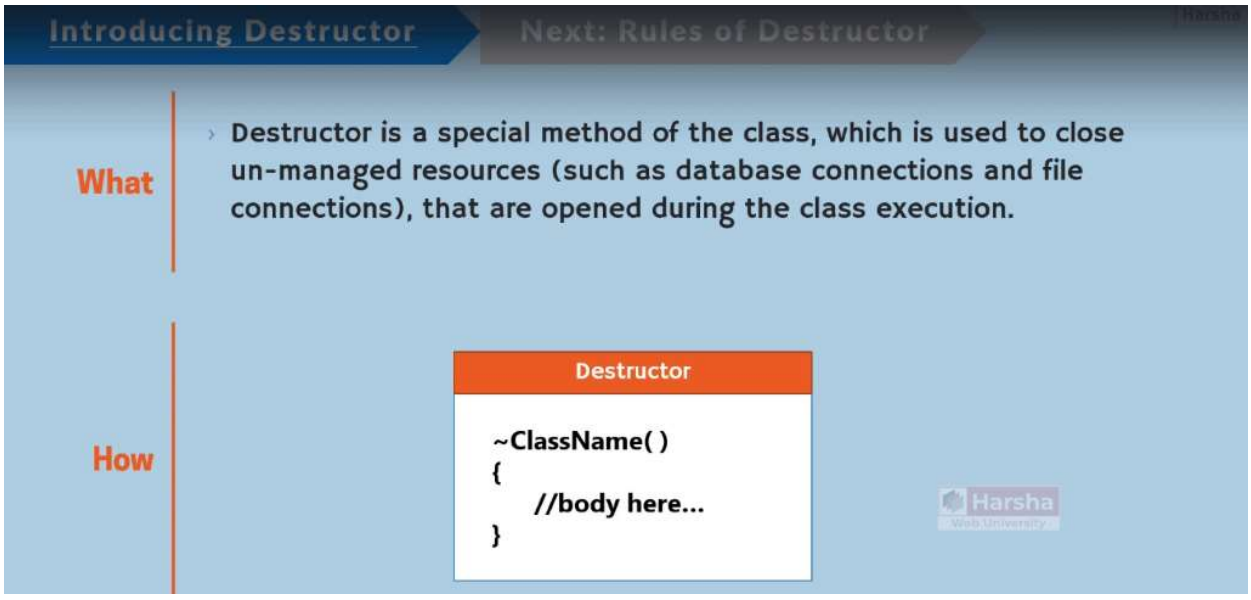
The database connection object is an example of Unmanaged resources – because the database is something which is external to the program. The program is trying to connect to the database memory with the help of some connection protocols that run based on the operating system. That is why the database connection is outside the boundary of the CLR. So In that case, the CLR can not allocate or deallocate the memory of the connection objects.

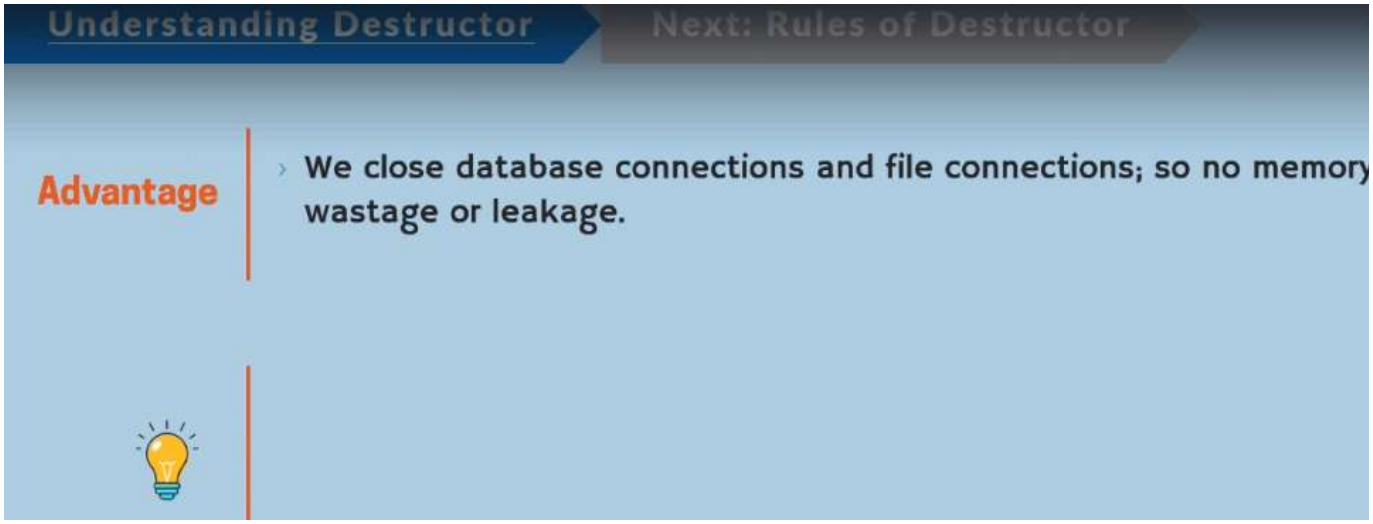
Now the question is who will clear the un-managed resources?

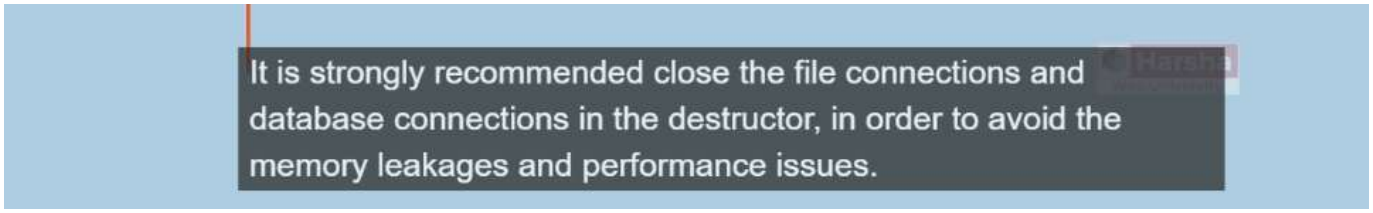
For example in your application, you are connecting to two database and two files. So who will delete that two connections ? If the database connections and file connections are not closed – the databases and files still under open – it may lead to memory leakages or performance issues.

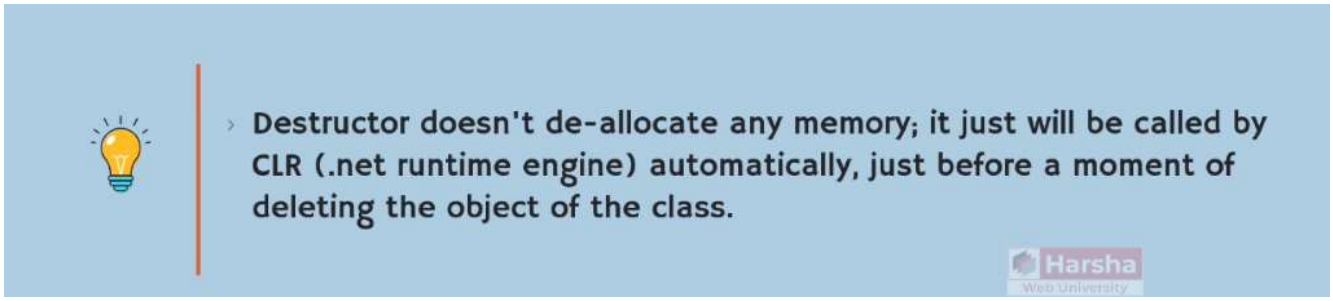
C# provides two ways to clear the un-managed resources. Destructor and Dispose method.


Clearing Unmanaged Resources
Next: Destructor
Destructor
Clears unmanaged resources just before deleting the object; i.e. generally at the end of application execution.
Dispose
Clears unmanaged resources after the specific task (work) is completed; so no need to wait till end of application execution.


Introducing Destructor
Next: Rules of Destructor
What
Destructor is a special method of the class, which is used to close un-managed resources (such as database connections and file connections), that are opened during the class execution.
How
Destructor
~ClassName( )
{
//body here...
}

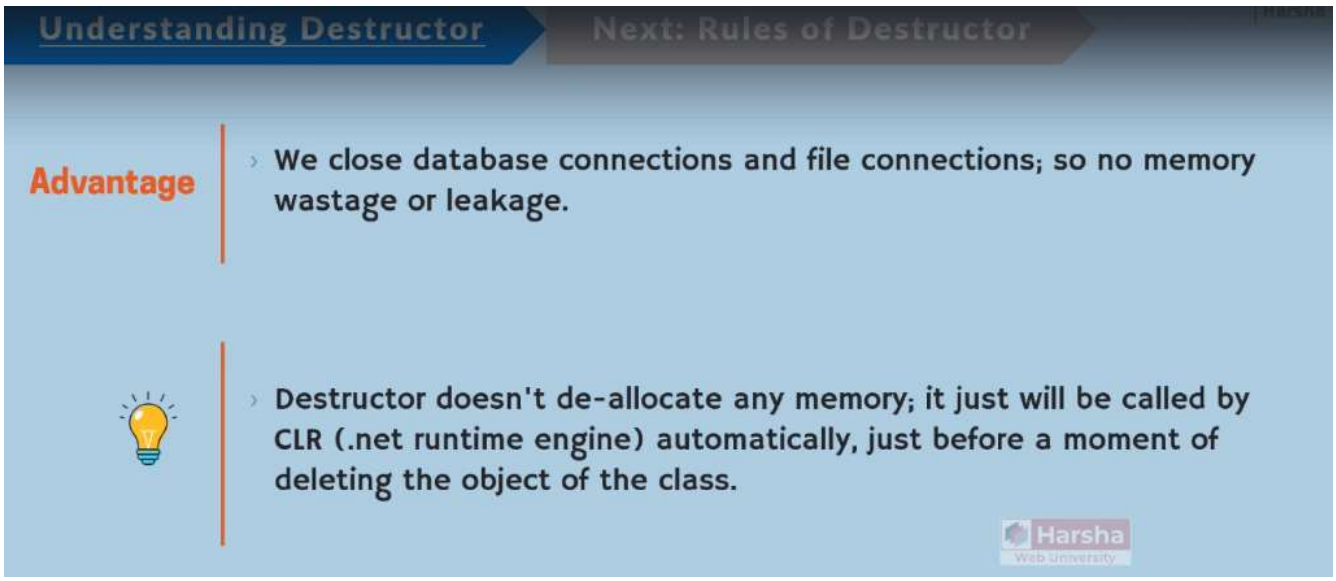

Understanding Destructor
Next: Rules of Destructor
Advantage
We close database connections and file connections; so no memory wastage or leakage.

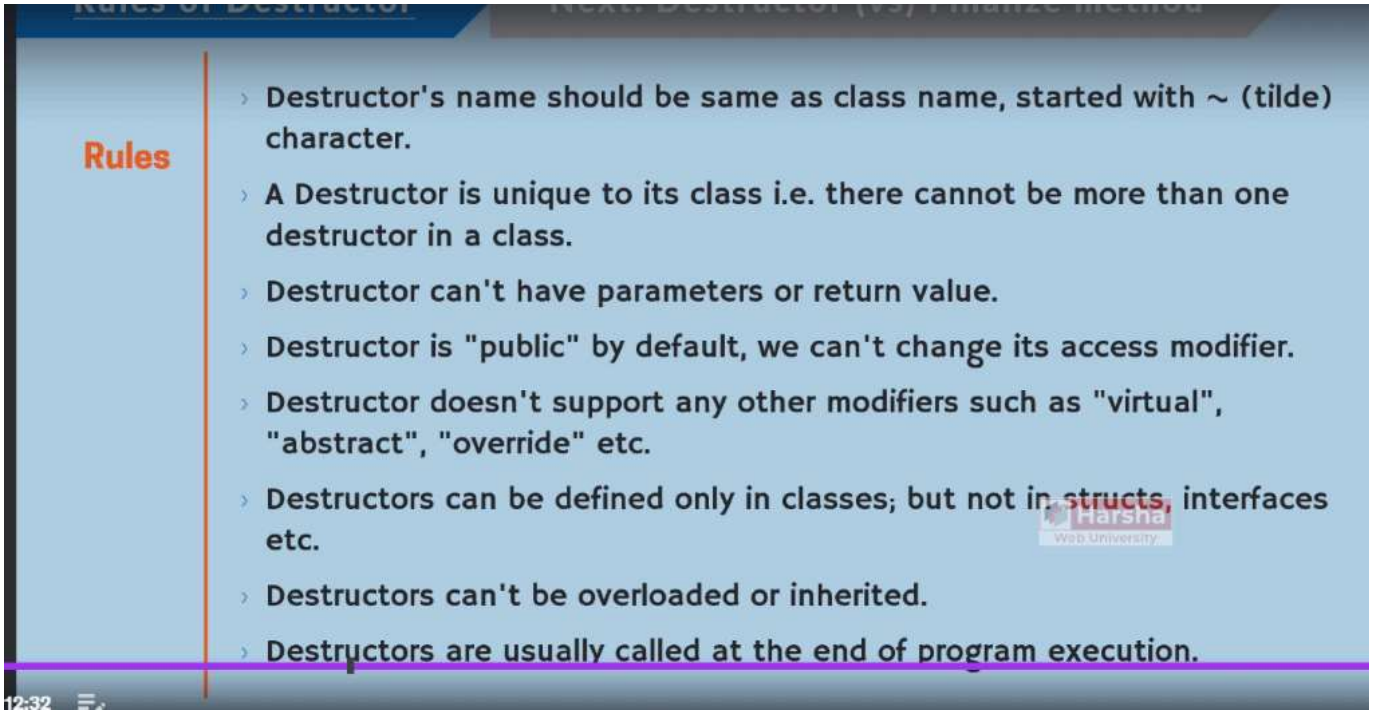




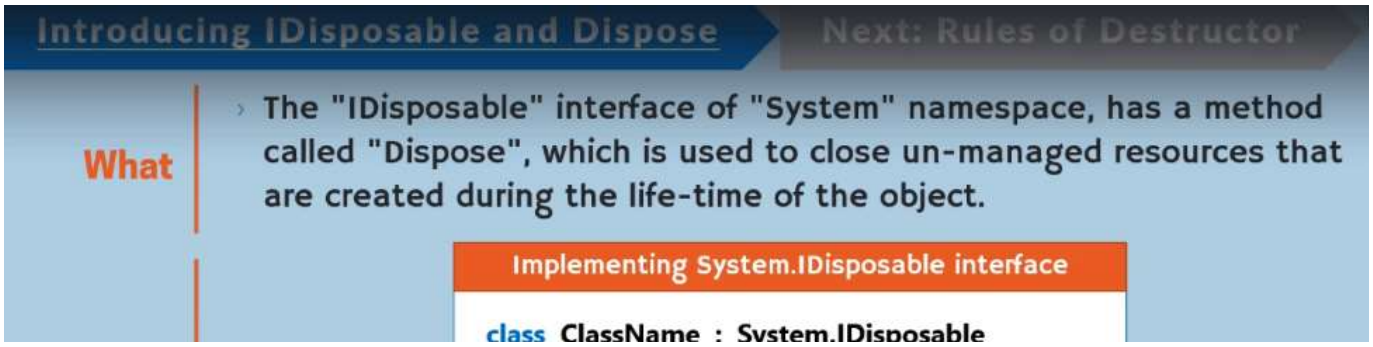
Constructor does not allocate and destructor does not deallocate memory. The purpose of constructor is that to initialize values of the fields and also to open the file connections and database connections if required.

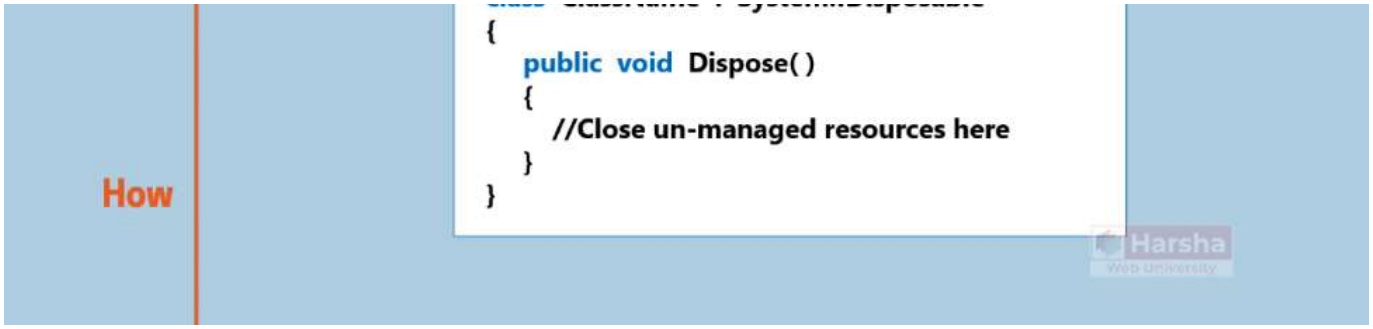
The only purpose of destructor is that – to close the un-managed resources, that means closing the file connections and database connections.





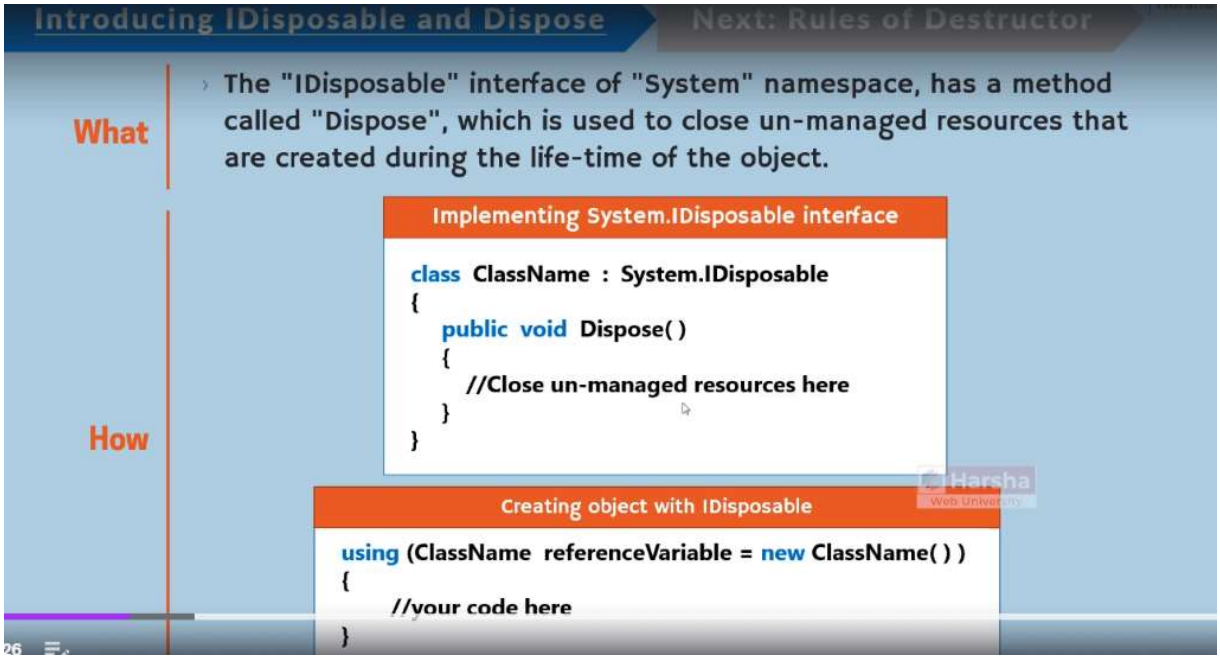
An alternative to Destructor.

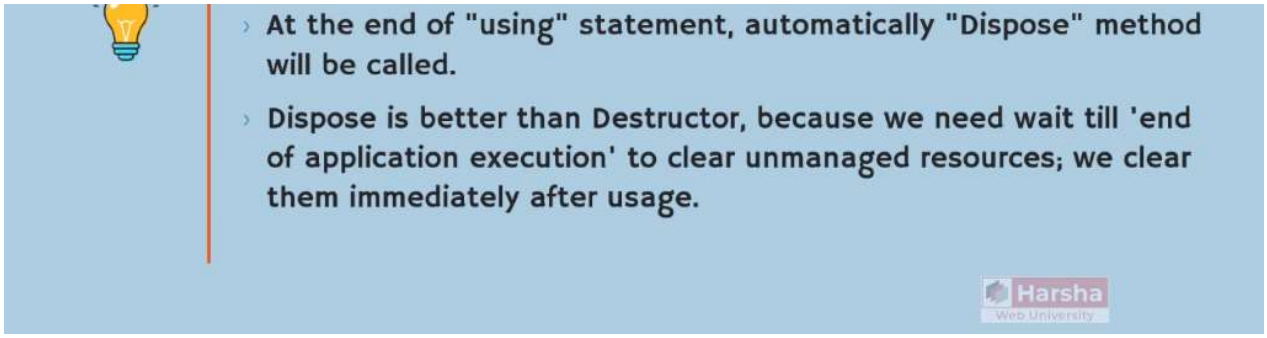


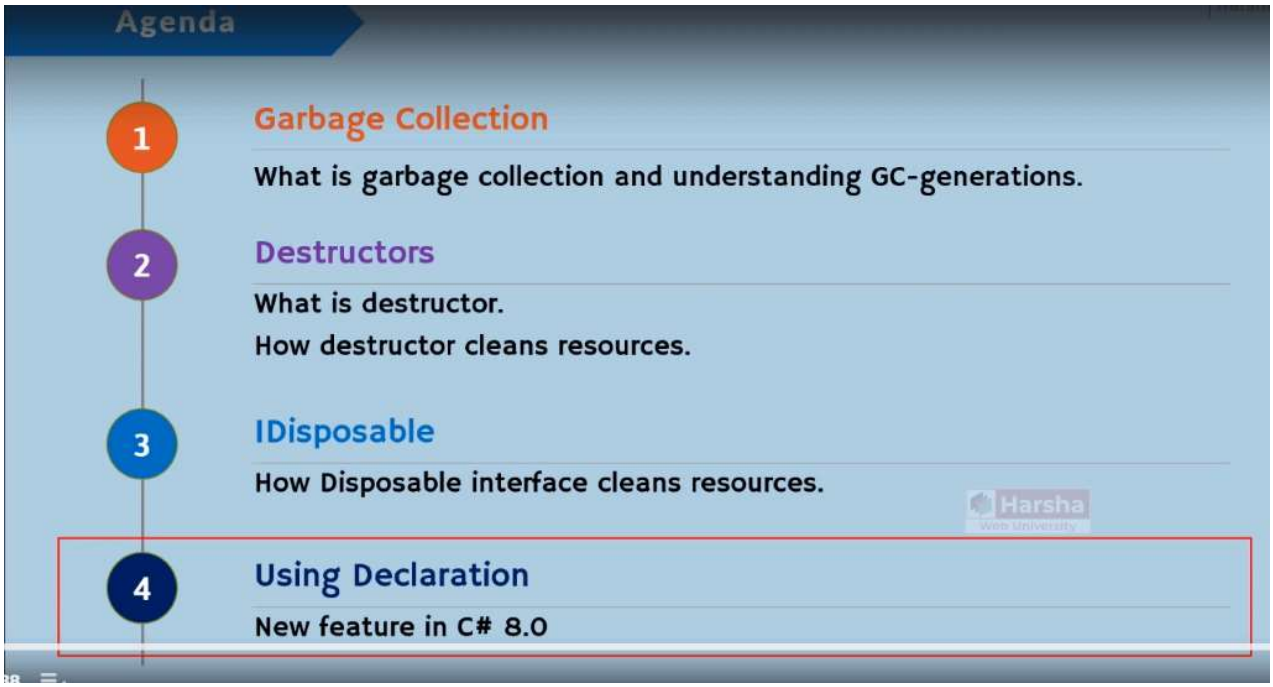


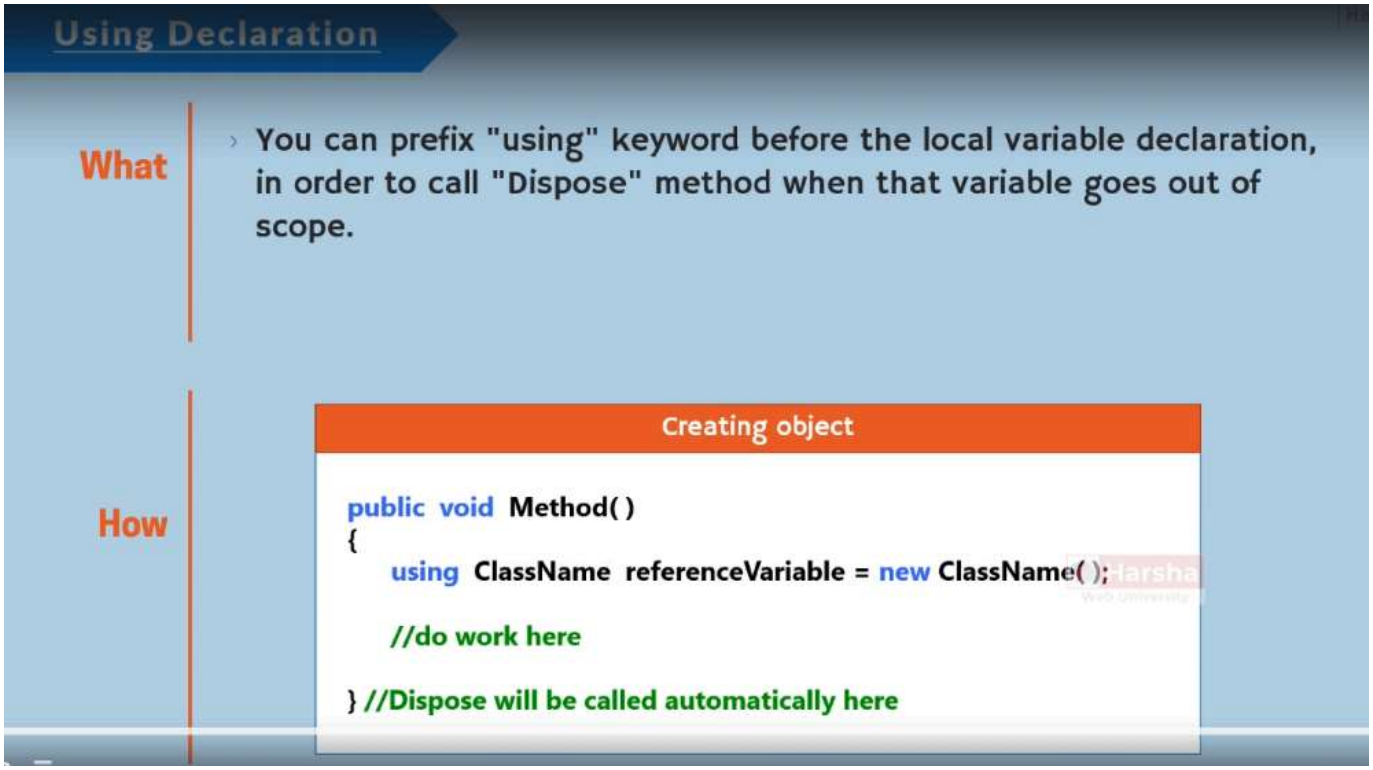
Dispose method has to be called by a developer explicitly. But Destructor is called automatically.

Example: Imagine a long running application that runs more than few hours. Suppose at the beginning of the application – you have created an object. In case if you are using the destructor – you need to wait for clearing the un-managed resources of the object till end of the application. But in case of dispose, you need not wait for end of the application.









So the 'csproj' file is the XML file that contains the project contains the project related settings - including the C# version and list of files of the current working project.