

Section 19 : GC, Destructors, IDisposable

Monday, July 15, 2024 3:39 PM

Agenda

- 1 Garbage Collection**
What is garbage collection and understanding GC-generations.
- 2 Destructors**
What is destructor.
How destructor cleans resources.
- 3 IDisposable**
How Disposable interface cleans resources.
- 4 Using Declaration**
New feature in C# 8.0

Harsha Web University

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

Introducing Garbage Collection

What

- Garbage Collection is a process of deleting objects from memory, to free-up memory; so the same memory can be re-used.

Next: How Garbage Collection Works?

Heap

- Object 1
- Object 2
- Object 3
- Object 4

Free Space

Harsha Web University

How Garbage Collection Works?

- CLR automatically allocates memory for all objects created anywhere in the application, whenever it encounters "new ClassName()" statement. This process is called as "Memory Management", which is done by "Memory Manager" component of CLR.
- All objects are stored in "Heap" (a.k.a. virtual memory).
- Heap is only one for the entire application life time.

Next: How objects are alive?

Heap

- Object 1
- Object 2
- Object 3
- Object 4

- › Heap is only one for the entire application life time.
- › 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

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".



Free Space

When GC gets triggered?

Next: Generations in GC

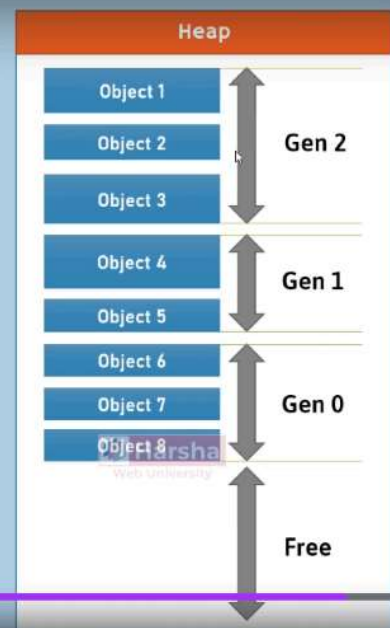
When GC gets triggered?

- › There are NO specific timings for GC to get triggered.
- › GC automatically gets triggered in the following conditions:
 - › When the "heap" is full or free space is too low.
 - › When we call GC.Collect() explicitly.

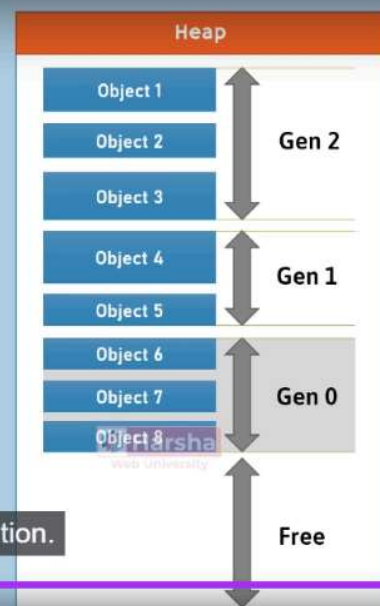


Generations

- › Heap contains three segments (called generations):
- › Generation 2 [Long-Lived Generation]
- › Generation 1 [Survival Generation]
- › Generation 0 [Short-Lived Generation]

**Generation 0**

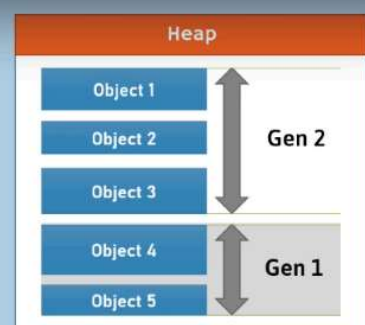
- › The "Generation 0" is the youngest generation and contains newly created short-lived objects and collected at first priority. The objects survive longer, are promoted to "Generation 1".

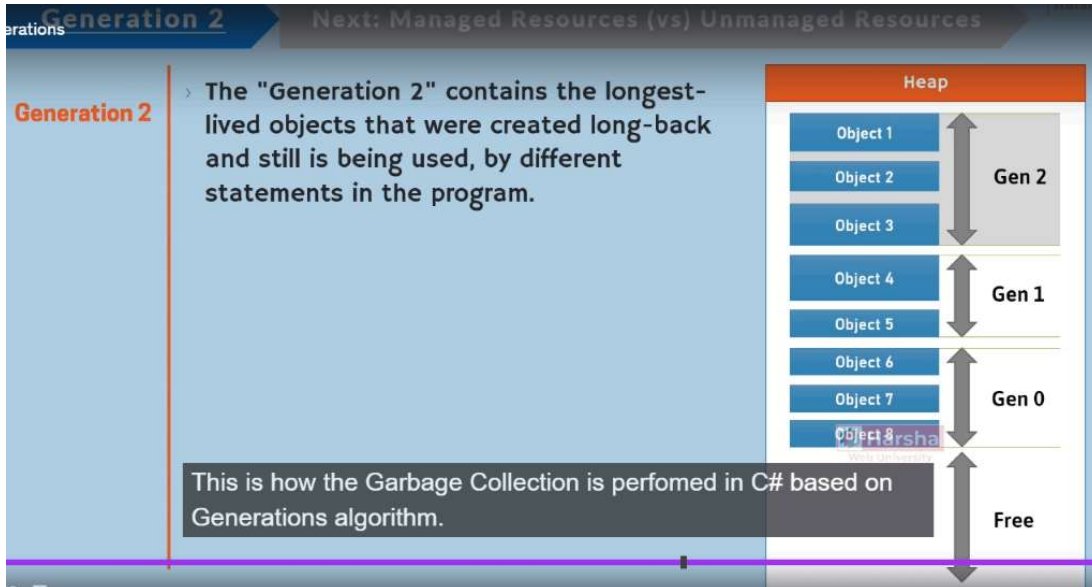
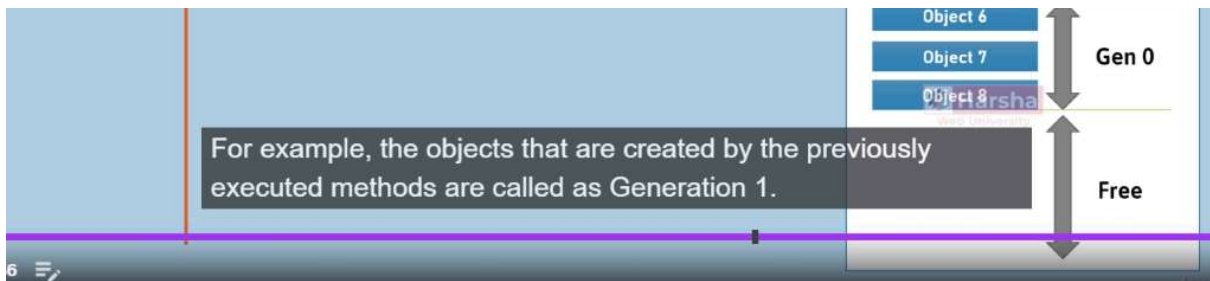


Generation 1 is the middle level generation.

Generation 1

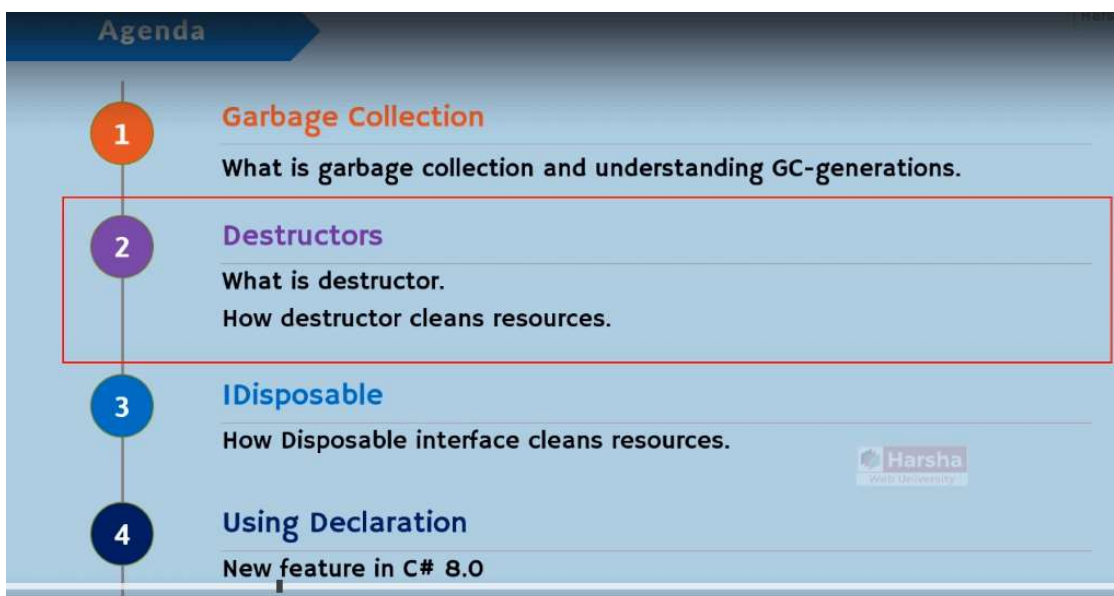
- › The "Generation 1" is buffer between "Generation 0" and "Generation 2".
- › The "Generation 1" mainly contains frequently-used and longer-lived objects.
- › Ex: The objects created in the previously-executed methods, but still accessible.





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.



It is strongly recommended close the file connections and database connections in the destructor, in order to avoid the memory leakages and performance issues.



- › Destructor doesn't de-allocate any memory; it just will be called by CLR (.net runtime engine) automatically, just before a moment of deleting the object of the class.



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.

Understanding Destructor

Next: Rules of Destructor

Advantage

- › We close database connections and file connections; so no memory wastage or leakage.



- › Destructor doesn't de-allocate any memory; it just will be called by CLR (.net runtime engine) automatically, just before a moment of deleting the object of the class.



Rules

- › Destructor's name should be same as class name, started with ~ (tilde) character.
- › A Destructor is unique to its class i.e. there cannot be more than one destructor in a class.
- › Destructor can't have parameters or return value.
- › Destructor is "public" by default, we can't change its access modifier.
- › Destructor doesn't support any other modifiers such as "virtual", "abstract", "override" etc.
- › Destructors can be defined only in classes; but not in structs, interfaces etc.
- › Destructors can't be overloaded or inherited.
- › Destructors are usually called at the end of program execution.

Agenda

1

Garbage Collection

What is garbage collection and understanding GC-generations.

2

Destructors

What is destructor.

How destructor cleans resources.

3

IDisposable

How Disposable interface cleans resources.

4

Using Declaration

New feature in C# 8.0

An alternative to Destructor.

Introducing IDisposable and Dispose

What

- › The "IDisposable" interface of "System" namespace, has a method called "Dispose", which is used to close un-managed resources that are created during the life-time of the object.

Implementing System.IDisposable interface

```
class ClassName : System.IDisposable
```


How

```
{
    public void Dispose()
    {
        //Close un-managed resources here
    }
}
```

Harsha
Web University

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.

Introducing IDisposable and Dispose

Next: Rules of Destructor

What

- › The "IDisposable" interface of "System" namespace, has a method called "Dispose", which is used to close un-managed resources that are created during the life-time of the object.

How

Implementing System.IDisposable interface

```
class ClassName : System.IDisposable
{
    public void Dispose()
    {
        //Close un-managed resources here
    }
}
```

Creating object with IDisposable

```
using (ClassName referenceVariable = new ClassName() )
{
    //your code here
}
```

Harsha
Web University

Understanding IDisposable and Dispose

Next: Using Declaration

- › The un-managed resources include file streams and database connections.



- › At the end of "using" statement, automatically "Dispose" method will be called.
- › Dispose is better than Destructor, because we need wait till 'end of application execution' to clear unmanaged resources; we clear them immediately after usage.

Agenda

1

Garbage Collection

What is garbage collection and understanding GC-generations.

2

Destructors

What is destructor.

How destructor cleans resources.

3

IDisposable

How Disposable interface cleans resources.

4

Using Declaration

New feature in C# 8.0

Using Declaration

What

- › You can prefix "using" keyword before the local variable declaration, in order to call "Dispose" method when that variable goes out of scope.

How

Creating object

```
public void Method()
{
    using ClassName referenceVariable = new ClassName();
    //do work here
} //Dispose will be called automatically here
```

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.

