1)Find out the difference between Destructor, finalize and Dispose

🡪

**Destructor**

Destructor implicitly calls the Finalize method, they are technically the same. Dispose is available with objects that implement the IDisposable interface.They are special methods that contains clean up code for the object. You can’t call them explicitly in your code as they are called implicitly by GC (Garbage Collector). In C# they have same name as the class name preceded by the “~” sign. Like-

class MyClass  
{  
public MyClass ()  
{  
}  
~MyClass ()  
{  
}  
}

**Dispose**  
These are just like any other methods in the class and can be called explicitly but they have a special purpose of cleaning up the object. In the dispose method we write clean up code for the object. It is important that we freed up all the unmanaged resources in the dispose method like database connection, files etc.

The class implementing dispose method should implement IDisposable which is inherited by interface and it contains GC.SuppressFinalize method for the object it is disposing if the class has destructor because it has already done the work to clean up the object, then it is not necessary for the garbage collector to call the object’s Finalize method.  
· Dispose() is called by the user

· Same purpose as finalize, to free unmanaged resources. However, implement this when you are writing a custom class that will be used by other users.

· Overriding Dispose () provides a way for the user code to free the unmanaged objects in your custom class.

· Dispose method can be invoked only by the classes that IDisposable interface.

**Finalize**  
Finalize () is called by Garbage Collector implicitly to free unmanaged resources. The garbage collector calls this method at some point after there are no longer valid references to the object. There are some resources like windows handles, database connections which cannot be collected by the garbage collector. Therefore the programmer needs to call Dispose() method of IDisposable interface.  
· Implement it when you have unmanaged resources in your code, and want to make sure that these resources are freed when the Garbage collection happens.

· Finalizers should release unmanaged resources only.

· Finalizers should always be protected, not public or private so that the method cannot be called from the application’s code directly and at the same time, it can make a call to the base.Finalize method  
class MyClass : IDisposable  
{  
private bool IsDisposed = false;  
public void Dispose()  
{  
Dispose(true);  
GC.SupressFinalize(this);  
}  
protected void Dispose(bool Diposing)  
{  
if (!IsDisposed)  
{  
if (Disposing)  
{  
//Clean Up managed resources  
}  
//Clean up unmanaged resources  
}  
IsDisposed = true;  
}  
~MyClass()  
{  
Dispose(false);  
}  
}

2) Difference between Multithreading and parallel programming

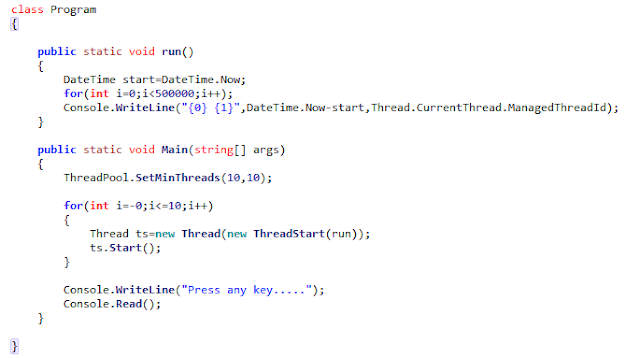
🡪

Multithreading –

Like we've clearly stated earlier, A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. It enables you to write in a way where multiple activities can proceed concurrently in the same program.

First Let we see about Multi Threading**,** Concepts,

[](http://2.bp.blogspot.com/-z3UAnMsOxIs/UdrxzTuhpSI/AAAAAAAAAHc/dPzuiYTZZns/s1600/multi_threading_program.png)

**Parallel Programming –**

In very simple terms, it is the use of multiple resources, in this case, processors, to solve a problem. This type of programming takes a problem, breaks it down into a series of smaller steps, delivers instructions, and processors execute the solutions at the same time. It is also a form of programming that offers the same results as concurrent programming but in less time and with more efficiency. Many computers, such as laptops and personal desktops, use this programming in their hardware to ensure that tasks are quickly completed in the background

Now Let we see parallel Programming and there advancements.  
class Program  
    {  
        **public** static void **Main**(string[] args)  
        {  
            Action Measure= (body)=>  
            {  
                DateTime start=DateTime.Now;  
                **body**();  
                Console.**WriteLine**("{0} {1}",DateTime.Now-start,Thread.CurrentThread.ManagedThreadId);  
            };  
             
            Action Cal=()=>{**for**(**int** i=0;i<500000;i++);};  
            ThreadPool.**SetMinThreads**(10,10);  
             
            Parallel.**For**(0,10,\_=>{**Measure**(Cal);});  
                                     
            Console.**WriteLine**("Press any key.....");  
            Console.**Read**();  
        }  
         
    }