Memory Management

1. Managing object lifetime from the CLR using the garbage collection mechanism
2. Accessing memory directly through pointers

Garbage Collection

new explicitly allocate heap memory.

Garbage collector (GC) automatically cleans up heap memory when object is not longer needed – calls Object.Finalize method automatically on a separate thread of execution.

```
using System;
public class SuperClass
{
  private string name;
  public SuperClass(string name) {  this.name = name;  }
  public override string ToString() {  return name;  }
}
class Program
{
  static void Process()
  {
    SuperClass s = new SuperClass("SuperClass");
    Console.WriteLine(s);
  }
  static void Main(string[] args)
  {
    Process();
    Console.WriteLine("End of Main");
  }
}
```

new allocates heap memory

s is no longer in scope

GC safely deallocates the object's memory because s no longer exits – calls automatically Object.Finalize

Results:
SupperClass
End of Main

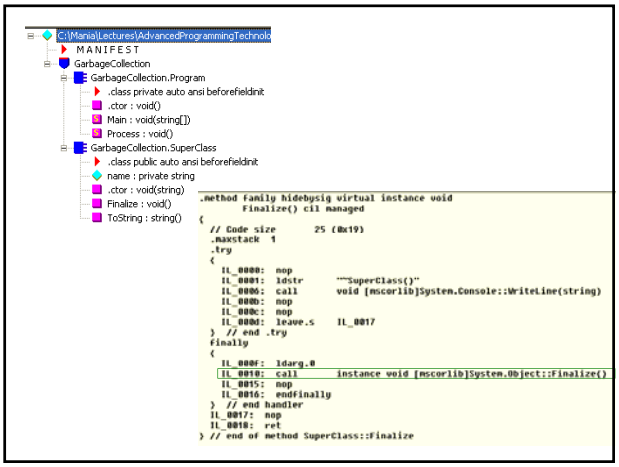
Overriding Finalize

- Method syntactically identical to a destructor

```
using System;
public class SuperClass
{
  …
  // Overriding Finalize
  ~SuperClass() {  Console.WriteLine("~SuperClass()");  }
}
```

Results:
SupperClass
End of Main
~SuperClass()

Finalize is called as the very last operation



Overriding Finalize in Derived Classes

- Objects are constructed "inside out" – the base class constructor is called first
- Objects are destroyed "outside in" – the derived class destructor completes its task before invoking the base class destructor

```
using System;

public class SuperClass
{
  private string name;
  public SuperClass(string name)
  {
    this.name = name;
    Console.WriteLine("SuperClass()");
  }
  public override string ToString() {  return name;  }
  ~SuperClass() {  Console.WriteLine("~SuperClass()");  }
}

public class DerivedClass : SuperClass
{
  public DerivedClass(string name) : base(name)
  {  Console.WriteLine("DerivedClass()");  }
  ~DerivedClass() {  Console.WriteLine("~DerivedClass()");
  }
}
```

```
class Program
{
  static void Process()
  {
    DerivedClass d = new DerivedClass("DerivedClass");
    Console.WriteLine(d);
  }
  static void Main(string[] args)
  {
    Process();
    Console.WriteLine("End of Main");
  }
}
```

Results:

```
SuperClass()
DerivedClass()
DerivedClass
End of Main
~DerivedClass()
~SuperClass()
```

Forcing Garbage Collection

- Method GC.Collect() forces garbage collection
- Method GC.WaitForPendingFinalizers() suspends the current thread until the queue of finalizers waiting to run on that thread is empty

```
class Program
{   static void Process()
    {   SuperClass s = new SuperClass("SuperClass");
        Console.WriteLine(s);
    }
    static void Main(string[] args)
    {   Process();
        Console.WriteLine("Do something");
        GC.Collect();
        GC.WaitForPendingFinalizers();
        Console.WriteLine("End of Main");
    }
}
```

Forcing collection with GC.Collect

Results:

```
SuperClass()
SuperClass
Do something
~SuperClass()
End of Main
```

```
class Program
{
  static void Process()
  {
    SuperClass s = new SuperClass("SuperClass");
    Console.WriteLine(s);
    s = null;
    GC.Collect();
    GC.WaitForPendingFinalizers();
  }
  static void Main(string[] args)
  {
    Process();
    Console.WriteLine("Do something");
    //GC.Collect();
    //GC.WaitForPendingFinalizers();
    Console.WriteLine("End of Main");
  }
}
```

Set the reference to null before forcing collection with GC.Collect

Results:

```
SuperClass()
SuperClass
~SuperClass()
Do something
End of Main
```

The Dispose Pattern

- Method Dispose() cleans recourses even references to the object are alive – explicit control of destructor like cleanup

  public void Dispose()

  - Suppress the GC's call to the Finalize method with GC.SuppressFinalize() method
- The developer has to call Dispose explicitly
- If the developer does not call Dispose the finalizer will be called – thus the finalizer has to call Dispose

```
public class SuperClass
{
  private string name;
  public SuperClass(string name)
  {
    this.name = name;
    Console.WriteLine("SuperClass()");
  }
  public override string ToString() {   return name;   }
  ~SuperClass()
  {
    Dispose();
    Console.WriteLine("~SuperClass()");
  }
  public void Dispose()
  {
    // Clean code is written here …
    Console.WriteLine("Dispose()");
    GC.SuppressFinalize(this);
  }
}
```

```
class Program
{
  static void Process()
  {
    SuperClass s = new SuperClass("SuperClass");
    Console.WriteLine(s);
    s.Dispose();
    s = null;
    GC.Collect();
    GC.WaitForPendingFinalizers();
  }
  static void Main(string[] args)
  {
    Process();
    Console.WriteLine("End of Main");
  }
}
```

Results:

```
SuperClass()
SuperClass
Dispose()
End of Main
```

The IDisposable Interface

```
public interface IDisposable
{
   void Dispose();
}
```

- Suppress the GC's call to the Finalize method with GC.SuppressFinalize() method
- The developer has to call Dispose explicitly
- If the developer does not call Dispose the finalizer will be called – thus the finalizer has to call Dispose

```
public class SuperClass : IDisposable
{  …  }
```