C# AND .NET FRAMEWORK

Presented By
Suruchi Dhall
Poornima Sundararaman

AGENDA

INTRODUCTION TO C#

- C# is a multi-paradigm programming language which is based on object-oriented and component-oriented programming disciplines.
- It provides a framework for free intermixing constructs from different paradigms.
- It uses the "best tool for the job" since no one paradigm solves all problems in the most efficient way.

BRIEF HISTORY

- C# was developed by Microsoft with Anders Hejlsberg as the principal designer and lead architect, within its .NET initiative and it made its appearance in 2000.
- Anders Hejlsberg and his team wanted to build a new programming language that would help to write class libraries in a .NET framework. He claimed that C# was much closer to C++ in its design.
- The name "C sharp" was inspired by musical notation where a sharp indicates that the written note should be made a semitone higher in pitch.
- Ever since its inception, C# has controversially been stated as an imitation of Java. However, through the course of time, both Java and C# have exhibited distinct features which support strong object-oriented design principles.

DESIGN GOALS

- C# was intended to be a simple, modern, object-oriented language.
- The language and implementation had to provide support for software engineering principles like strong type checking, array bounds checking and automatic garbage collection.
- The language was intended for development of software components suitable for deployment in distributed environments.
- Source code portability was important for programmers who were familiar with C and C++.
- Support for internalization to adapt the software to different languages.

WHY C# ? : FEATURES

- C# is the first "component-oriented" language in the C/C++ family.
- The big idea of C# is that everything is an object.
- C# is a programming language that directly reflects the underlying Common Language Infrastructure (CLI). Most of its intrinsic types correspond to value-types implemented by the CLI framework.
- Type-safety: C# is more type safe than C++. Type safety is the extent to which a programming language discourages or prevents type errors.
- C#, like C++, but unlike Java, supports operator overloading.
- Managed memory is automatically garbage collected. Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory that is no longer needed.
- C# provides properties as syntactic sugar for a common pattern in which a pair of methods, accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class.

WHY C# ? : FEATURES …cont.

- In addition to the try…catch construct to handle exceptions, C# has a try…finally construct to guarantee execution of the code in the finally block, whether an exception occurs or not.
- Unlike Java, C# does not have checked exceptions. This has been a conscious decision based on the issues of scalability and versionability.
- Multiple inheritance is not supported, although a class can implement any number of interfaces. This was a design decision to avoid complication and simplify architectural requirements throughout CLI.
- C# supports a strict Boolean data type, bool. Statements that take conditions, such as while and if, require an expression of a type that implements the true operator, such as the boolean type.

C# & OBJECT ORIENTED APPROACH

(I) Structs & Classes
(II) Interfaces
(III) Delegates
(IV) Switch Statement: Fall Through
(V) For Each: Control Flow
(VI) Virtual Methods
(VII) Boxing & Unboxing
(VIII) Common Type System
(IX) Generics
(X) Reflection

(I) STRUCTS & CLASSES IN C#

- In C# structs are very different from classes. Structs in C# are designed to encapsulate lightweight objects. They are value types (not reference types), so they're passed by value.

- They are sealed, which means they cannot be derived from or have any base class.

- Classes in C# are different from classes in C++ in the following ways:

i. There is no access modifier on the name of the base class and inheritance is always public.

ii. A class can only be derived from one base class. If no base class is explicitly specified, then the class will automatically be derived from System.Object.

iii. In C++, the only types of class members are variables, functions, constructors, destructors and operator overloads, C# also permits delegates, events and properties.

iv. The access modifiers public, private and protected have the same meaning as in C++ but there are two additional access modifiers available: (a) Internal (b) Protected internal

(II) INTERFACES

- C# does not support Multiple Inheritance
- However a class can implement number of interfaces
- It contains methods, properties, indexers, and events

```
interface DataBind
{
void Bind(IDataBinder bind);
}
Class EditBox: Control, DataBind
{
void DataBind.Bind(IDataBinder bind) {...}
}
```

(III) DELEGATES

- A delegate is similar to a function pointer in C/C#.
- Using a delegate allows a programmer to encapsulate a reference to a method inside a delegate object, which can then be passed to code.
- Declaring a delegate:

```
public delegate void BookDelegate(Book book);
```

- Instantiating a delegate:

```
book.PaperbackBooks(new BookDelegate(Title));
```

- Calling a delegate:

```
processBook(b);
```

(IV)SWITCH STATEMENT: FALL THROUGH

- In C# a switch statement may not "fall through" to the next statement if it does any work. To accomplish this, you need to use an explicit goto statement:

```
switch (i)
{
case 4:CallFuncOne();
goto case 5;
case 5:
CallSomeFunc();
}
```

If the case statement does not work (has no code within it) then you can fall :

```
switch (i)
{
   case 4:// fall through
   case 5:
   CallSomeFunc();
}
```

(V)FOR EACH CONTROL FLOW

New control flow statement- foreach :

C# provides an additional flow control statement, for each. For each loops across all items in array or collection without requiring explicit specification of the indices.

Syntax:

```
Foreach(double someElement in MyArray)
{
    Console.WriteLine(someElement);
}
```

(VI)VIRTUAL METHODS

- In C# one can choose to override a virtual function from base class. Derived method can participate in polymorphism only if it uses the keyword override before it.
- In C++, if provided the same syntax method in derived class as base class virtual method, it will be automatically be overridden.
- In C# we have abstract methods and in C++ pure virtual methods. Both may not be exactly same, but are equivalent (as pure virtual can have function body)

- EXAMPLE:

```
class Base{
public virtual string VirtualMethod()
{ return "base virtual"; }
}
class Derived : Base{
public override string VirtualMethod()
{ return "Derived overriden"; }
}
```

(VII)BOXING AND UNBOXING

- *Boxing* is the operation of converting a value-type object into a value of a corresponding reference type.
- Boxing in C# is implicit.
- *Unboxing* is the operation of converting a value of a reference type (previously boxed) into a value of a value type.
- Unboxing in C# requires an explicit type cast. A boxed object of type T can only be unboxed to a T (or a nullable T).

- EXAMPLE :

```
int box_var = 42; // Value type.
object bar = box_var; // foo is boxed to bar.
int box_var2 = (int)bar; // Unboxed back to value type.
```

(VIII)COMMON TYPE SYSTEM

- C# has a *unified type system*. This unified type system is called Common Type System (CTS).

- A unified type system implies that all types, including primitives such as integers, are subclasses of the System.Object class. For example, every type inherits a ToString() method.

- CTS separates data types into two categories:
  1. Value types
  2. Reference types

(VIII)VALUE TYPE V/S REFERENCE TYPE

VALUE TYPE:

- Instances of value types do not have referential identity nor referential comparison semantics i.e. equality and inequality comparisons for value types compare the actual data values within the instances, unless the corresponding operators are overloaded.

- Value types are derived from System.ValueType, always have a default value, and can always be created and copied.

- They cannot derive from each other (but can implement interfaces) and cannot have an explicit default (parameterless) constructor.

(VIII) VALUE TYPE VS REFERENCE TYPE …cont.

REFERENCE TYPE:

- Reference types have the notion of referential identity - each instance of a reference type is inherently distinct from every other instance, even if the data within both instances is the same.

- It is not always possible to create an instance of a reference type, nor to copy an existing instance, or perform a value comparison on two existing instances.

- Specific reference types can provide services by exposing a public constructor or implementing a corresponding interface (such as ICloneable orIComparable). Examples: System.String, System.Array

(IX)GENERICS

- Generics use type parameters, which make it possible to design classes and methods that do not specify the type used until the class or method is instantiated.
- The main advantage is that one can use generic type parameters to create classes and methods that can be used without incurring the cost of runtime casts or boxing operations.
- EXAMPLE:

```
public class GenericList<T>
 {
 void Add(T input)  { }
}
class TestGenericList
{
private class ExampleClass { }
static void Main() {
//Declare a list of type int.
GenericList<int> list1 = new GenericList<int>();
//Declare a list of type string.
GenericList<string> list2 = new GenericList<string>();
}
}
```

(X) REFLECTION

- Reflection is useful in the following situations:
- When you need to access attributes in your program's metadata. See the topic Accessing Attributes With Reflection.
- For examining and instantiating types in an assembly.
- For building new types at runtime. Use classes in System.Reflection.Emit.
- For performing late binding, accessing methods on types created at run time.

- EXAMPLE:

```
// Using GetType to obtain type information:
int i = 42;
System.Type type = i.GetType();
System.Console.WriteLine(type);
```

ADVANTAGES OF C#

- It allows design time and run time attributes to be included.
- It allows integrated documentation using XML.
- No header files, IDL etc. are required.
- It can be embedded into web pages.
- Garbage collection ensures no memory leakage and stray pointers.
- Due to exceptions, error handling is well-planned and not done as an afterthought.
- Allows provision for interoperability.

APPLICATIONS OF C#

- The three main types of application that can be written in C# are:

1. Winforms - Windows like Forms.
2. Console - Command line Input and Output.
3. Web Sites : Web sites need IIS (Microsoft's web server) and ASP.NET.

INTRODUCTION TO .NET FRAMEWORK

- .NET framework is a software framework primarily for Microsoft Windows. It includes a large library & provides language interoperability across several programming languages.
- Programs written for the .NET Framework execute in a software environment, as opposed to a hardware one for most other programs. Common examples of such programs include Visual Studio, Team Explorer UI, Sharp Develop .
- Programmers combine their own source code with the .NET Framework and other libraries. The .NET Framework is intended to be used by most new applications created for the Windows platform.

HISTORY OF .NET

- .NET was developed by Microsoft in the mid 1990s, originally under the name of 'Next Generation Windows Services'.

- .NET 1.1 was the first version to be included as a part of the Windows OS. It provided built-in support for ASP .NET, ODBC & Oracle databases. It provided a higher level of trust by allowing the user to enable Code Access Security in ASP. NET.

- Currently, Windows 8 supports version 4.5 of .NET which supports provision for 'Metro Style Apps'

DESIGN FEATURES

- Interoperability: .NET Framework provides means to access functionality implemented in newer and older programs that execute outside the .NET environment. Access to COM components is provided in the System.Runtime.InteropServices and System.EnterpriseServices namespaces of the framework.

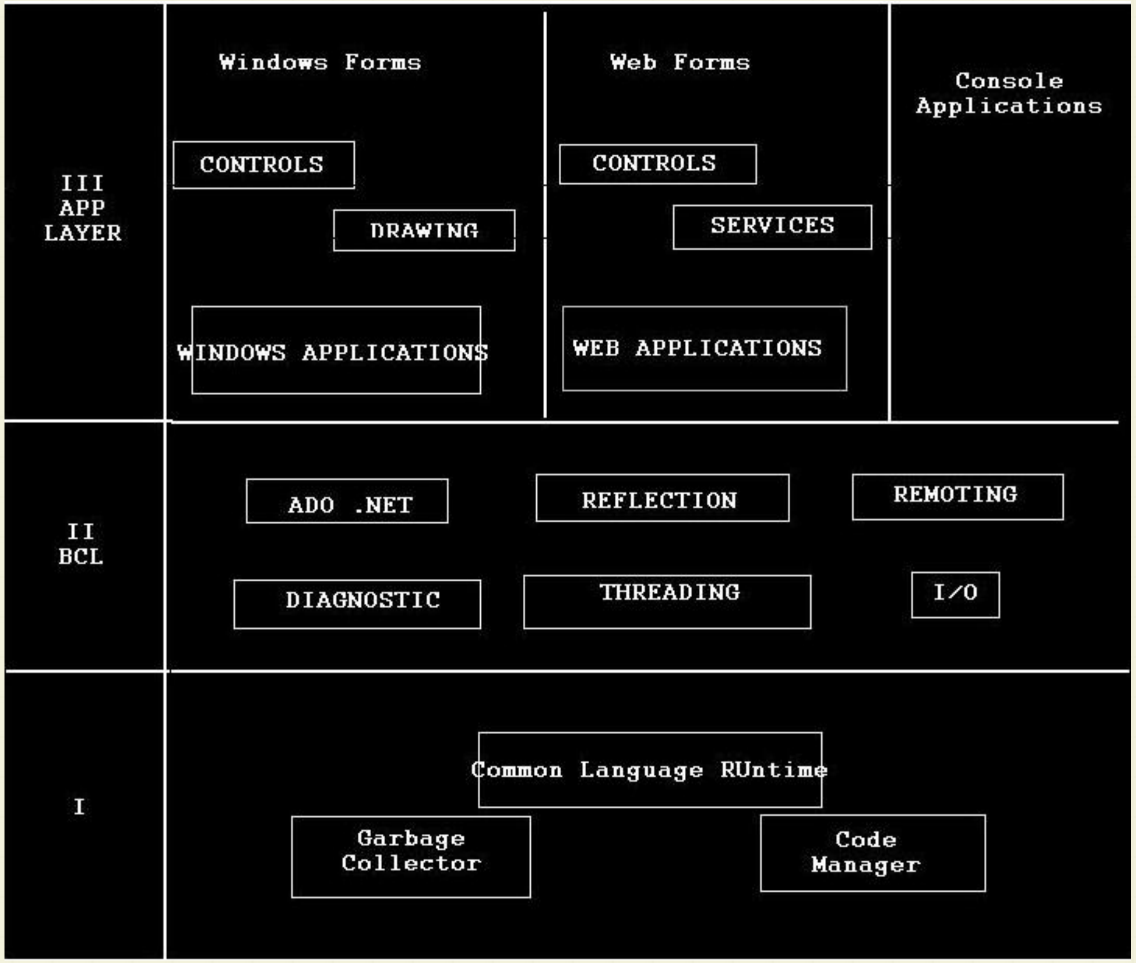
- Common Language Runtime engine: CLR serves as the execution engine of the .NET Framework. All .NET programs execute under the supervision of the CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.

- Language Independence: .NET Framework introduces Common Type System which define all possible data types & programming constructs supported by CLR & ruled for their interaction as per CLI specification. This allows the exchange of types & object instances between libraries & their applications written using any conforming .NET language.

DESIGN FEATURES …cont.

- BCL: It is a library of functionality which is available to all languages using the Framework. It consists of classes, interfaces or reusable types that integrate with CLR.
- Portability: The framework allows platform-agnostic & cross-platform implementations for other OS'. The availability of specifications for the CLI, & C# make it possible for third parties to create compatible implementations of the framework & it's languages on other platforms.

.NET FRAMEWORK ARCHITECTURE



.NET ARCHITECTURE DESIGN

CLR:

- The software environment in which programs for .NET framework run is known as the 'Common Language Runtime'. It is Microsoft's implementation of a 'Common Language Infrastructure'. Its purpose is to provide a language-neutral platform for application development & execution.

- The CLI is responsible for exception handling, garbage collection, security & interoperability.

- The CIL code is housed in CLI assemblies. The assembly consists of many files, one of which must contain metadata for assembly.

VM:

- An application VM provides services such as security, memory management & exception handling.

- The security mechanism supports 2 main features.

- Code Access Security: It is based on proof that is related to a specific assembly. It uses the same to determine permissions granted to the code.

- Validation & Verification: Validation determines whether the code satisfies specified requirements and Verification determines whether the conditions imposed are satisfied or not.

.NET ARCHITECTURE DESIGN …cont.

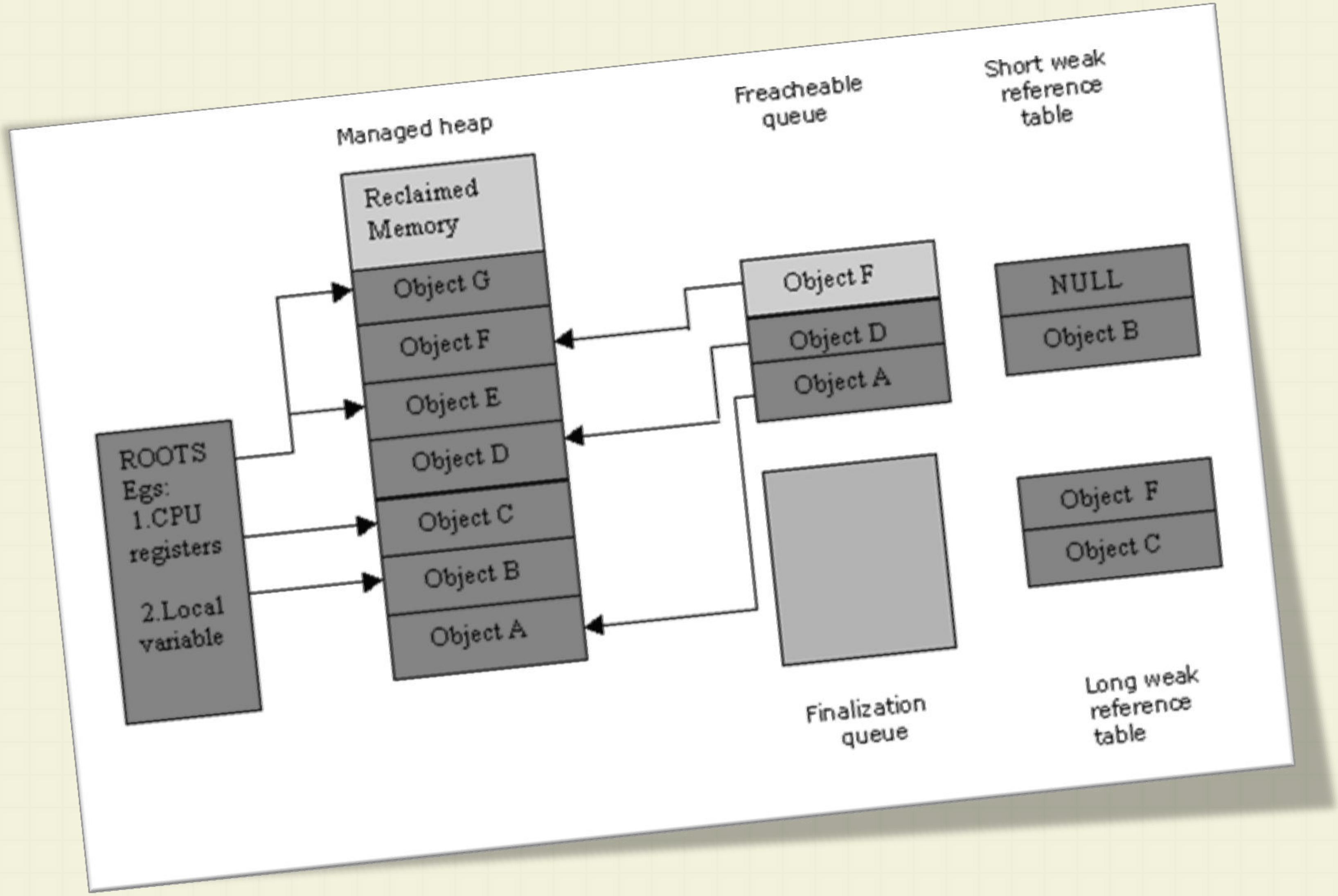
Class Library:

- The class library & CLR essentially constitute the .NET Framework. The Framework's base class library provides UI, data access, database connectivity, algorithms, network communications & web application development.
- In spite of the varied functionality, the BCL includes a small subset of the entire class library & is the core set of classes that serve as the basic API of the CLR.
- The Framework Class Library is a superset of BCL & refers to the entire class library which includes libraries for ADO.NET, ASP.NET, Windows Forms, etc.
- It is much larger in scope compared to C++ & comparable to libraries in Java.

.NET AND OBJECT-ORIENTED APPROACH

- Memory management in .NET Framework is a crucial aspect.
- Memory is allocated to instantiations of .NET objects from the managed heap, a pool of memory managed by the CLR. As long as there exists a reference to an object, either a direct reference or via a graph, the object is considered to be in use.
- When there is no reference to an object, and it cannot be reached or used, it becomes garbage, eligible for collection
- NET Framework includes a garbage_collector which runs periodically, on a separate thread from the application's thread, that enumerates all the unusable objects and reclaims the memory allocated to them.

GARBAGE COLLECTION



GARBAGE COLLECTION…cont.

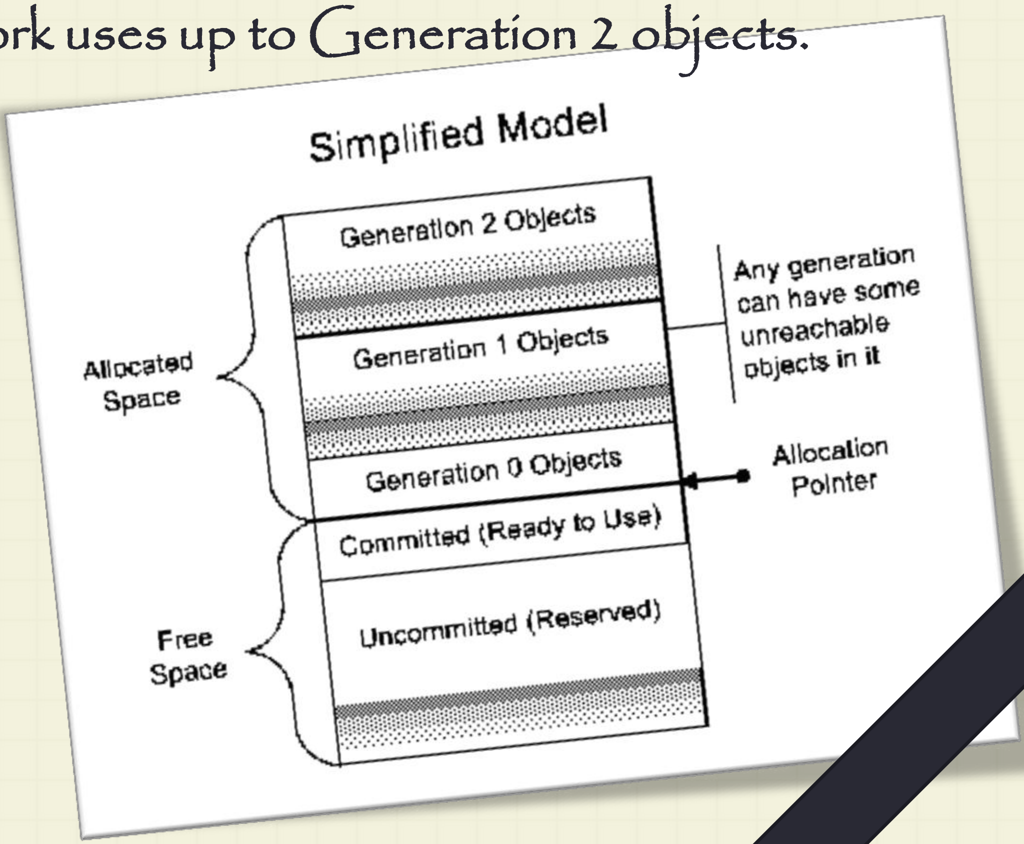
- Each .NET application has a set of roots, which are pointers to objects on the managed heap (*managed objects*). These may be references to static objects, objects defined as local variables or method parameters currently in scope, objects referred to by CPU registers

- When the GC runs, it pauses the application, and for each object referred to in the root, it recursively collects all the objects reachable from the root objects and marks them as reachable.

- It uses CLI metadata and reflection to discover the objects encapsulated by an object, and then recursively walk them. It then enumerates all the objects on the heap (which were initially allocated contiguously) using reflection. All objects not marked as reachable are garbage.

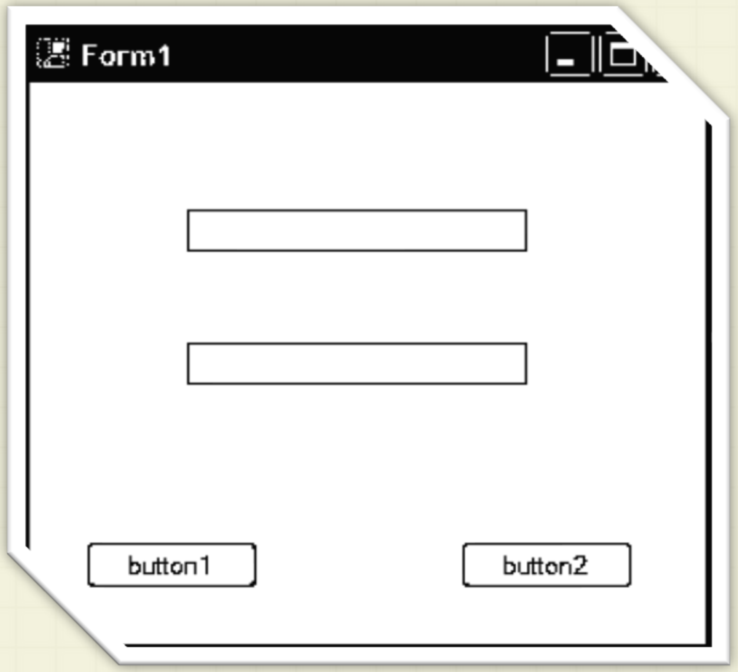
GARBAGE COLLECTION…cont.

The GC used by .NET Framework is actually '*generational*'. Objects are assigned a *generation*; newly created objects belong to '*Generation 0*'. The objects that survive a garbage collection are tagged as '*Generation 1*', and the Generation 1 objects that survive another collection are '*Generation 2*' objects. The .NET Framework uses up to Generation 2 objects.



GUI APPLICATIONS USING C# & .NET

- Windows form is used to create applications with a user interface.
- The following are the steps to create a Windows Form Application:
  - i. Open a new Project and choose the Windows Form Application
  - ii. Start dragging components from the Toolbox to the Form. The form will look something like this:

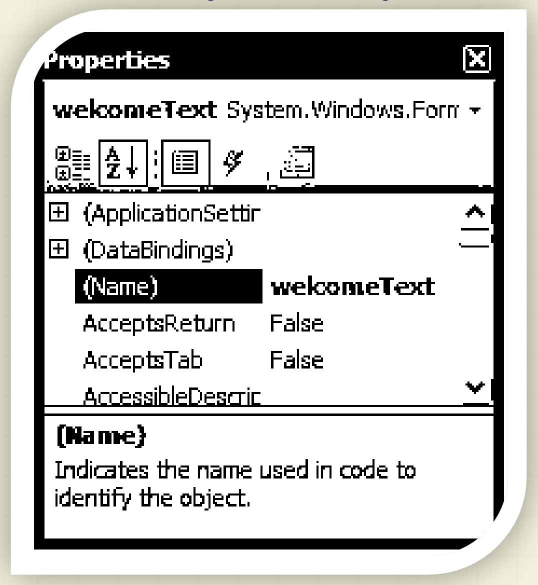


GUI APPLICATIONS USING C# & .NET …cont.

iii. As components are added to the Form, Visual Studio assigns default names to each one. It is via these names that any C# code will interact with the user interface of the application.

iv. The name of these components can be changed in the *Properties* panel

v. In addition to changing the names of components it is also possible to change a myriad array of different properties via the properties panel.



(ApplicationSettir
(DataBindings)

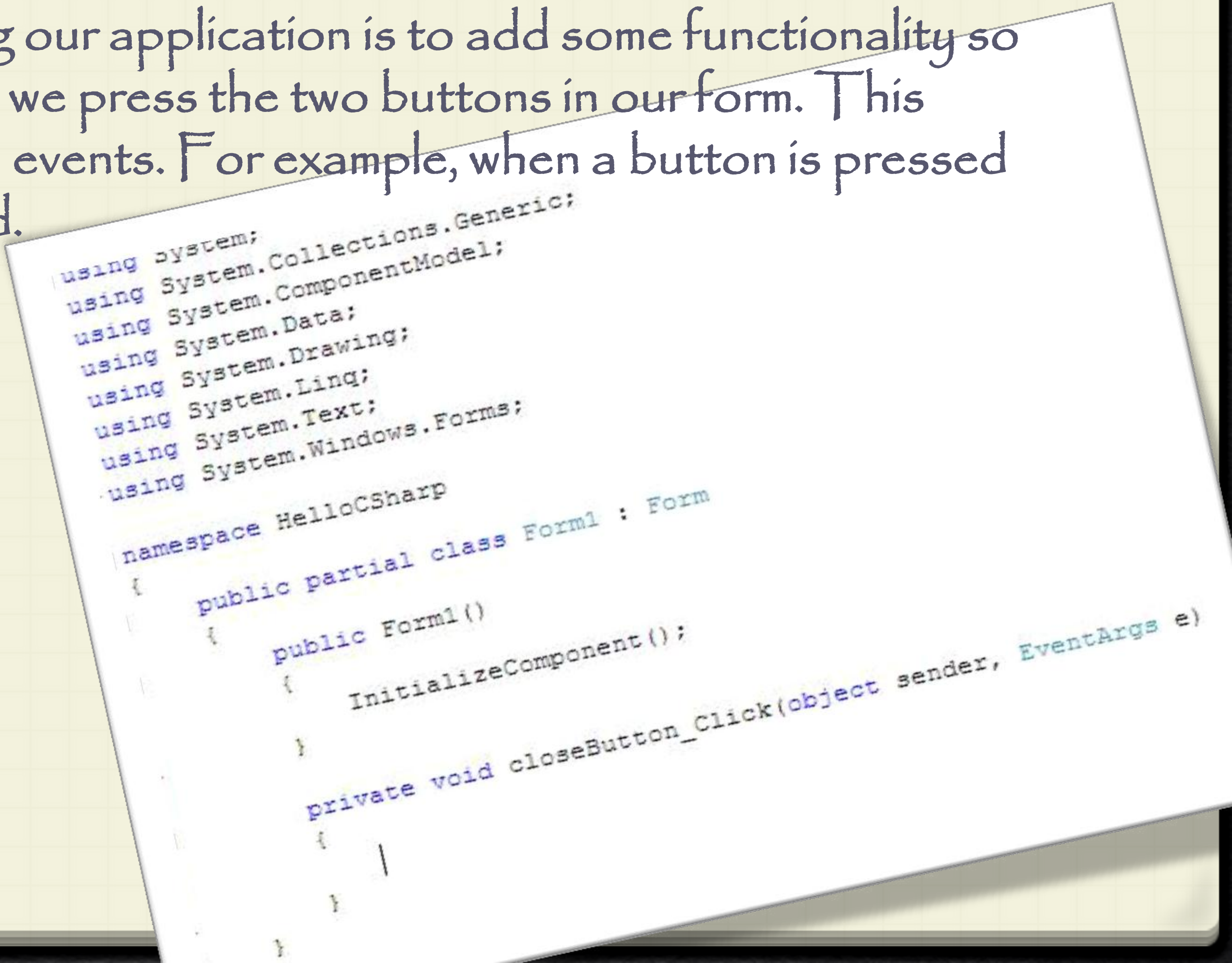
(Name)	welcomeText
AcceptsReturn	False
AcceptsTab	False
AccessibleDescric	

(Name)
Indicates the name used in code to identify the object.

GUI APPLICATIONS USING C# & .NET …cont.

vi. Adding behavior to a Visual Studio C# application:

The next task in creating our application is to add some functionality so that things happen when we press the two buttons in our form. This behavior is controlled via events. For example, when a button is pressed a *Click* event is triggered.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace HelloCSharp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void closeButton_Click(object sender, EventArgs e)
        {
            |
        }
    }
```

GUI APPLICATIONS USING C# & .NET …cont.

- Codes can be added for on Button Click events. Some examples are:

```
(I) private void closeButton_Click(object sender, EventArgs e)
        {
                Close();
        }

(II) private void helloButton_Click(object sender, EventArgs e)
        {
                welcomeText.Text = "Hello " + nameText.Text;
        }
```

ADVANTAGES OF .NET

- Platform independent
- Supports multiple programming languages
- Easy to deploy
- Supports various security features such as cryptography, application domain, verification process etc.

WRAPPING UP

- Introduced basic concepts of C# programming.
- Discussed similarities and differences between C# & other programming languages (C,C++,Java).
- Discussed Object-Oriented behaviour of C#.
- Introduced concepts related .NET framework.
- Explained .NET architecture.
- Shed light upon Object-Oriented approach in .NET framework.
- Advantages & Applications of C# and .NET.

REFERENCES

- www.microsoft.com
- www.techotopia.com
- www.c-sharpcorner.com
- www.codeproject.com
- www.wikipedia.org