**REFLECTION**

It is used for obtaining information of attributes at runtime. In other words we can say that Reflection is use to read the metadata of an assembly and exact information of a given type. The ability of a managed code to read its own metadata for the purpose of finding assemblies, modules and type information at runtime. In other words, reflection provides objects that encapsulate assemblies, modules and types. A program reflects on itself by extracting metadata from its assembly.

Example :

using System. Reflection;

public class ReflectionExample1

{

public static void Main()

{

try

{

int number = 10;

Type type = number.GetType();

Console.WriteLine(type);

} catch(Exception ex)

{

Console.WriteLine(ex);

}

}

}

Output:

System.Int32

**Indexers**

Indexers allows our class to be used just like arrays. Or we can say we can index the object using [] brackets just like the arrays.

It is a special type of properties which add functionality to a class to be indexed.

We can use any access modifiers which indexer, and similarly any valid c# return type.

Example:

using System;

namespace IndexerDemo

{

class People

{

string[] people = { "Kunal", "Rahul", "Ashish" };

public string this[int index]

{

get

{

return people[index];

}

set

{

people[index] = value;

}

}

}

class Program

{

static void Main(string[] args)

{

try

{

People obj = new People();

Console.WriteLine(obj[0]);

Console.WriteLine(obj[1]);

Console.WriteLine(obj[2]);

Console.ReadKey();

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

}

}

}

Output:

Kunal

Rahul

Ashish

**Delegates**

In simple words , it is pointer to a function.

Example:

using System;

namespace DelegateExample1

{

class Program

{

public delegate void MethodPointer();

static void MyMethod()

{

Console.WriteLine("This is invoke through pointer");

}

static void Main(string[] args)

{

try

{

MethodPointer obj = new MethodPointer(MyMethod);

obj.Invoke();

Console.ReadKey();

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

}

}

}

Output:

This method is invoke through pointer

This is not a efficient way to called a method. We can directly call a method. So the question here is , What is the benefit of calling method via pointer ?

Delegates are especially used for implementing events and the call-back methods.

Example:

using System;

namespace DelegateCallBack

{

public class CallDelegate

{

public delegate void CallBack(int i);

public void longRunningMethod(CallBack obj)

{

for(int i=0; i<10000;i++)

{

//Does something

obj(i);

}

}

}

class Program

{

static void CallbackMethod(int i)

{

Console.WriteLine(i);

}

static void Main(string[] args)

{

CallDelegate obj = new CallDelegate();

obj.longRunningMethod(CallbackMethod);

}

}

}

**Events**

Events enable a [class](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or object to notify other classes or objects when something of interest occurs. The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.

The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the publisher class. Some other class that accepts this event is called the subscriber class. Events use the publisher-subscriber model.

**Question and Answers**.

1. **What is delegate in C#?**

A delegate in C# is an object that holds the reference to a method. It is like function pointer in C++.

### 2)What is Reflection?

Reflection allows us to get metadata and assemblies of an object at runtime.

3) **What is a point of a pointer when we can invoke the actual function/method?**  
  
Many times rather than pointing to actual methods , we would like to point to abstraction. For example rather than pointing to “Add(int,int)” , “Substract(int,int)” , we create an abstract delegate pointer “Operation(int,int)” and point to any of them above.

4)**What are the uses of delegates ?**  
  
There are two basic uses of delegates :-

Callbacks / events. Method and function abstraction.

5**) What are events ?**

Events are encapsulation over delegates.

6) **Why callback methods use delegates?**

This ability to refer to a method as a parameter makes delegates ideal for defining callback methods. For example, a sort algorithm could be passed a reference to the method that compares two objects. Separating the comparison code allows for the algorithm to be written in a more general way.

7) **What is Reflection?**  
**The process of obtaining information about assemblies and the types defined within them, and creating, invoking, and accessing type instances at run time.**

**8) What is late binding?**  
Binding is the process of locating the declaration that corresponds to a uniquely specified type. When this process occurs at run time rather than at compile time, it is called late binding.

9) **How do you get to Type objects from an assembly that is already loaded?**

Use Type.GetType to get the Type objects from an assembly that is already loadedUse Type..::.GetType to get the Type objects from an assembly that is already loaded.

**References**

[www.youtube.com](http://www.youtube.com)

[www.javatpoint.com](http://www.javatpoint.com)

[www.tutorialspoint.com](http://www.tutorialspoint.com)

[www.akadia.com](http://www.akadia.com)

[www.codeproject.com](http://www.codeproject.com)