C# Notes: Delegates, Lambda Expressions, and Event Handling

1. Delegates

A delegate is a reference type used to encapsulate a method with a specific signature. It allows methods to be passed as parameters. Delegates are used for implementing events and callback methods.

There are three types of delegates in C#:

1. Single-cast delegate
2. Multi-cast delegate
3. Generic delegate (Func, Action, Predicate)

Basic Syntax:

delegate returnType DelegateName(parameterList);

Example 1: Single-cast Delegate

```
public delegate void GreetDelegate(string name);

public class Greet
{
    public static void SayHello(string name)
    {
        Console.WriteLine("Hello, " + name);
    }
}

class Program
{
    static void Main()
    {
        GreetDelegate del = new GreetDelegate(Greet.SayHello);
        del("Alice");
    }
}
```

Example 2: Multi-cast Delegate

```
public delegate void NotifyDelegate();

public class Notifications
{
    public static void EmailNotification()
    {
        Console.WriteLine("Email sent.");
    }

    public static void SMSNotification()
    {
        Console.WriteLine("SMS sent.");
    }
}

class Program
{
    static void Main()
    {
        NotifyDelegate notify = Notifications.EmailNotification;
        notify += Notifications.SMSNotification;

        notify();  // Both methods will be called
    }
}
```

2. Lambda Expressions

Lambda expressions are anonymous functions that can contain expressions or statements. They are especially useful in LINQ queries and anonymous method definitions.

Syntax:

(parameters) => expression

(parameters) => { statements }

Example 1: Simple Lambda

```
Func<int, int, int> multiply = (x, y) => x * y;
```

```
Console.WriteLine(multiply(4, 5));  // Output: 20
```

Example 2: Statement Lambda

```
Action<string> greet = name => {
    string message = "Welcome, " + name;
    Console.WriteLine(message);
};
greet("Bob");
```

Example 3: Using Lambda with List

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
var evenNumbers = numbers.Where(n => n % 2 == 0);

foreach (var num in evenNumbers)
    Console.WriteLine(num);
```

3. Event Handling

Events provide a way for a class to notify other classes or objects when something happens.

Events are declared using delegates and are commonly used in GUI and real-time systems.

Steps for event handling:

1. Define a delegate.
2. Declare an event using that delegate.
3. Create event handler methods.
4. Subscribe and trigger the event.

Example:

```
public class Button
{
    public delegate void ClickHandler();
    public event ClickHandler OnClick;
```

```
    public void Click()
    {
        Console.WriteLine("Button clicked.");
        OnClick?.Invoke();
    }
}

public class UI
{
    public void HandleClick()
    {
        Console.WriteLine("Button click handled in UI.");
    }
}

class Program
{
    static void Main()
    {
        Button button = new Button();
        UI ui = new UI();

        button.OnClick += ui.HandleClick;
        button.Click();
    }
}
```