**Enumeration:**

* An enumeration is a set of named integer constants. The keyword enum declares an enumerated type.
* **The general form for an enumeration is:**

**enum** name { enumeration list };

* Here, the type name of the enumeration is specified by name.
* The enumeration list is a comma-separated list of identifiers.
* **Here is an example**:

It defines an enumeration called Apple that enumerates various types of apples:

**enum** Apple { Jonathan, GoldenDel, RedDel, Winesap, Cortland, McIntosh };

* A key point to understand about an enumeration is that each of the symbols stands for an integer value.
* However, no implicit conversions are defined between an **enum** type and

the built-in integer types, so an explicit cast must be used.

* Also, a cast is required when converting between two enumeration types.
* Since enumerations represent integer values, you can use an enumeration to control a **switch** statement or as the control variable in a **for** loop,
* **for example:**
* Each enumeration symbol is given a value one greater than the symbol that precedes it.
* By default, the value of the first enumeration symbol is 0. Therefore, in the **Apple** enumeration, **Jonathan** is 0, **GoldenDel** is 1, **RedDel** is 2, and so on.
* The members of an enumeration are accessed through their type name via the dot operator.
* **For example:**

Console.WriteLine(Apple.RedDel + " has the value " +(int)Apple.RedDel);

**Displays:**

RedDel has the value 2

**Initialize an Enumeration:**

* You can specify the value of one or more of the symbols by using an initializer.
* Do this by following the symbol with an equal sign and an integer value.
* Symbols that appear after initializers are assigned values greater than the previous initialization value.
* **For example**, the following code assigns the value of 10 to **RedDel**:

**enum** Apple{Jonathan, GoldenDel, RedDel = 10, Winesap, Cortland, McIntosh};

Now the values of these symbols are:

|  |  |
| --- | --- |
| Jonathan | 0 |
| GoldenDel | 1 |
| RedDel | 10 |
| Winesap | 11 |
| Cortland | 12 |
| McIntosh | 13 |

**Use Enumerations:**

* Enumerations are very useful when your program requires one or more specialized symbols.
* For example, imagine that you are writing a program that controls a conveyor belt in a factory.
* You might create a method called Conveyor( ) that accepts the following commands as parameters: start, stop, forward, and reverse. Instead of passing Conveyor( ) integers, such as 1 for start, 2 for stop, and so on.
* Here is an example of this approach:

// Simulate a conveyor belt.

**using System;**

**class ConveyorControl**

{

// Enumerate the conveyor commands.

public **enum** Action{ Start, Stop, Forward, Reverse };

public void **Conveyor**(Action com)

{

**switch**(com)

{

**case** Action.Start:

Console.WriteLine("Starting conveyor.");

**break;**

**case** Action.Stop:

Console.WriteLine("Stopping conveyor.");

**break;**

**case** Action.Forward:

Console.WriteLine("Moving forward.");

**break**;

**case** Action.Reverse:

Console.WriteLine("Moving backward.");

**break;**

}

}

}

**class** ConveyorDemo

{

**public static void Main()**

{

**ConveyorControl** c = **new** **ConveyorControl();**

c.Conveyor(ConveyorControl.Action.Start);

c.Conveyor(ConveyorControl.Action.Forward);

c.Conveyor(ConveyorControl.Action.Reverse);

c.Conveyor(ConveyorControl.Action.Stop);

}

}

**The output from the program is shown here:** Starting conveyor.

Moving forward.

Moving backward.

Stopping conveyor.