Enumerated Types

C# Programming
Enumerated Types

Enumerated Types

- A way to "design your own data"
- Rather than deciding what kinds of things that we want to store, for an enumerated type we can decide what values our variable can store
- These are actually held by C# as numbers, but it enumerates the possible values that we invent – hence the name

Cricket and enumerated types

- We might want to store how a batsman was taken out of a game of cricket:
  - Bowled out
  - Run out
  - Caught
  - Not out
  - Leg before wicket
- We could use an integer, but it makes much more sense to use an enum:

UNIVERSITY OF Hull

Example enum type

```
enum OutReason {
   bowledOut,
   runOut,
   caught,
   notOut,
   legBeforeWicket
}
```

- This sets out all the values that the type may have (these are all the reasons you can be out in cricket)

Creating a enum variable

- We can now use our new type to create a variable that holds the reason why the player was out:

```
OutReason reason;
reason = OutReason.bowledOut;
```

- The variable `reason` is of type `OutReason`
- We can only set this variable to the values that we have allowed in the type

Literal values of type OutReason

```
if (reason == OutReason.bowledOut)
{
    Console.WriteLine ( "Bowled out" );
}
```

- Literal values of the enum are given by using the type name, followed by the particular value

UNIVERSITY OF Hull

enums as numeric values

- You can perform assignment with enumerated types
- But you cannot perform arithmetic as it would be meaningless
- If you are feeling brave you can use casting to get at the numbers which represent the type values
  - But only do this if you know what you are doing....

Cricket and enumerated types

- We might want to store how our batsman was taken out of the game:
  - Bowled out
  - Run out
  - Caught
  - Not out
  - Leg before wicket
- We could use an integer, but it makes much more sense to use an enum:

Enums in structures

- Enumerated types are very useful in structures:

```
struct Player {
    public string Name;
    public int Score;
    public OutReason Reason;
}
```

- We now store the reason why the player was out

Enum input/output

- Unfortunately we have to do extra work when we read in or print out our enum values
- Our code will have to get the information from the user and set the appropriate value
- It will also have to decide what to print

Making life easier – the switch

- Fortunately C# has a construction which can help
- It is the switch construction
- It lets us write code which picks a particular action based on a value
- We can use a switch on our enumerated type

Output with switch

```
switch (reason) {
 case OutReason.caught:
   Console.WriteLine ("Caught");
   break;
 case OutReason.notOut :
   Console.WriteLine ("Not out");
   break;
}
```

- This switch prints a message which describes the contents of `reason`

Input with switch

```
switch (reasonString) {
 case "caught":
   reason = OutReason.caught;
   break;
 case "not out" :
   reason = OutReason.notOut;
   break;
 default :
   Console.WriteLine ("Error");
   break;
}
```

Summary

- You can use the enum feature to add new types to your programs
- Once you have the new type you can declare values of that type
- A value of an enum type can only occupy one of the values given