AlmaMate

Value Types & Reference Types

Oct- 2017

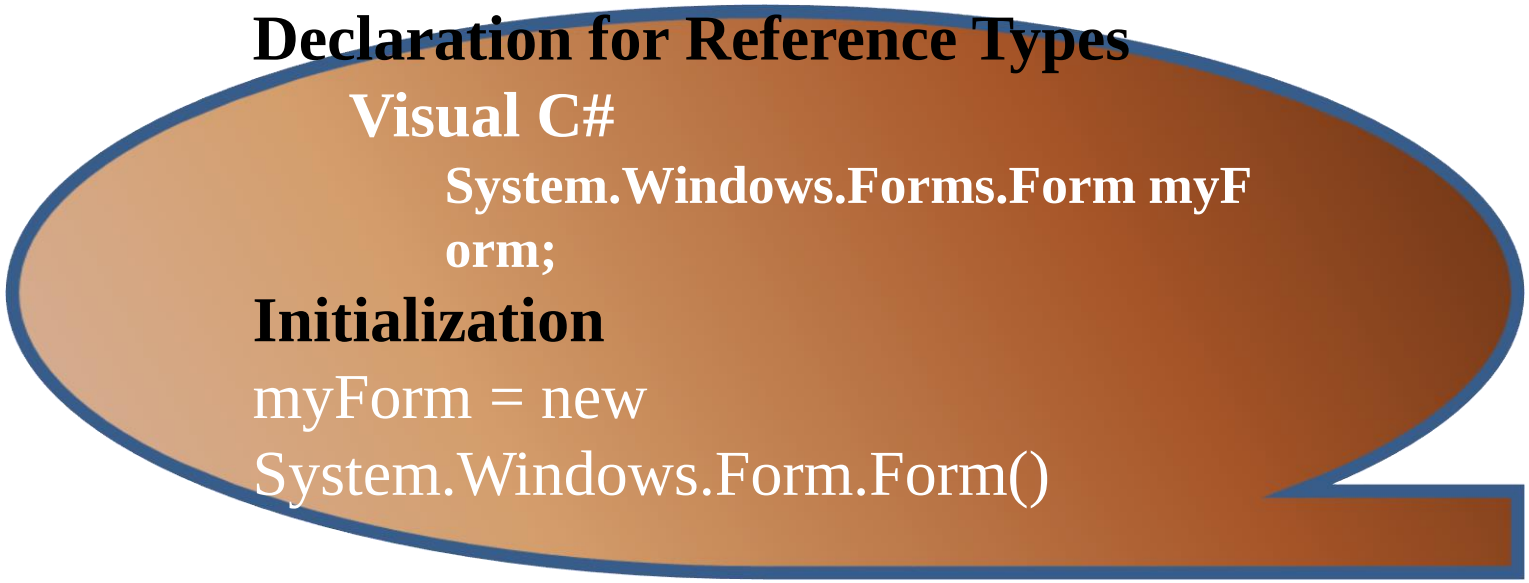
Objectives

- In this session, you will learn to:
  - Describe memory allocation
  - Use structures
  - Use enumerations

Describing Memory Allocation

AlmaMate

- The memory allocated to the variables is referred to in the following ways:
  - Value types: Contain data. Built-in data types, such as `int`, `char`, and `float` are value types.
  - Reference types: Contain address of the block of memory that holds the data. Data types, such as `string` and `class` are reference types.



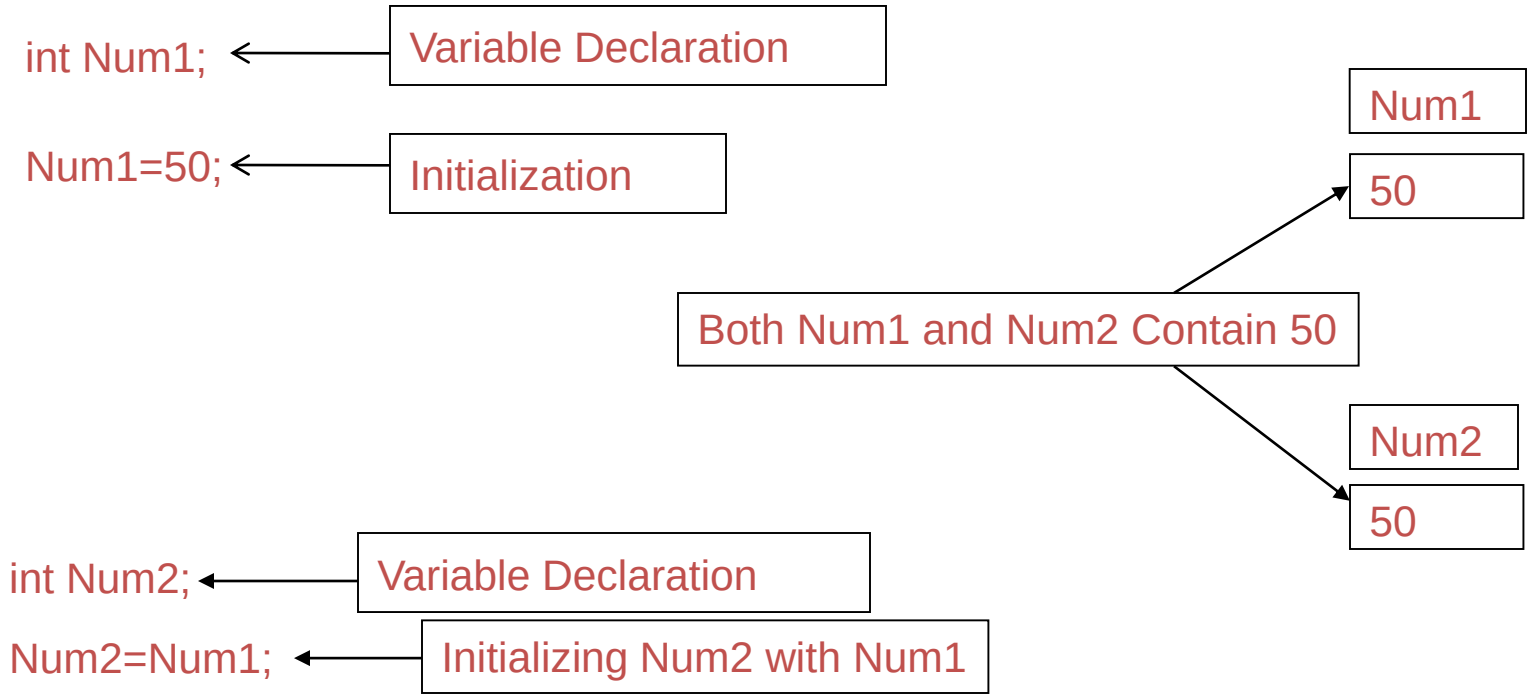
AlmaMate

- All variables are assigned default values
  - Zero for Numeric
  - False for Boolean
  - null for Reference

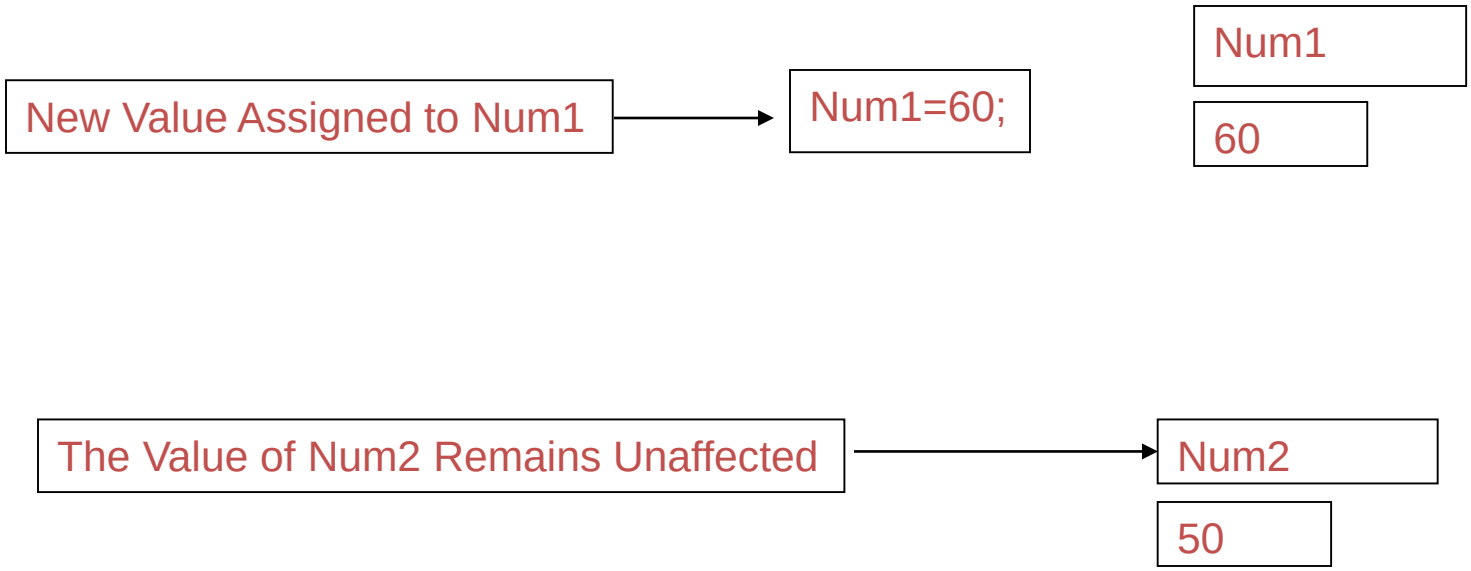
  But be careful, don’t rely on these initial values.

Describing Memory Allocation (Contd.)

- Value Type:

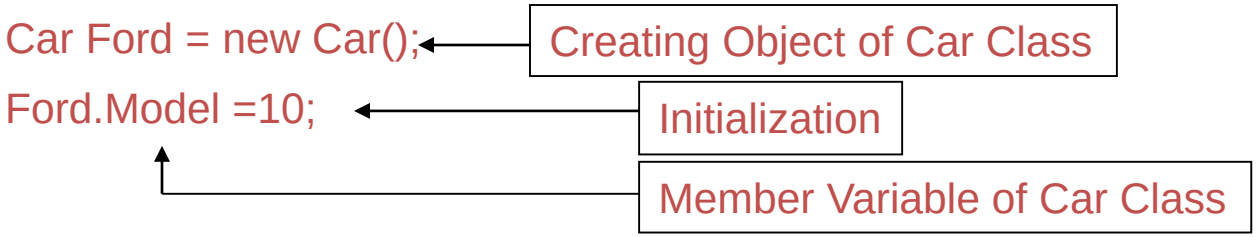


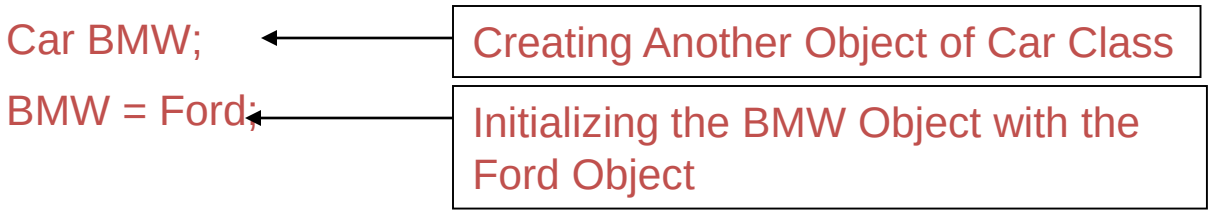
Describing Memory Allocation (Contd.)



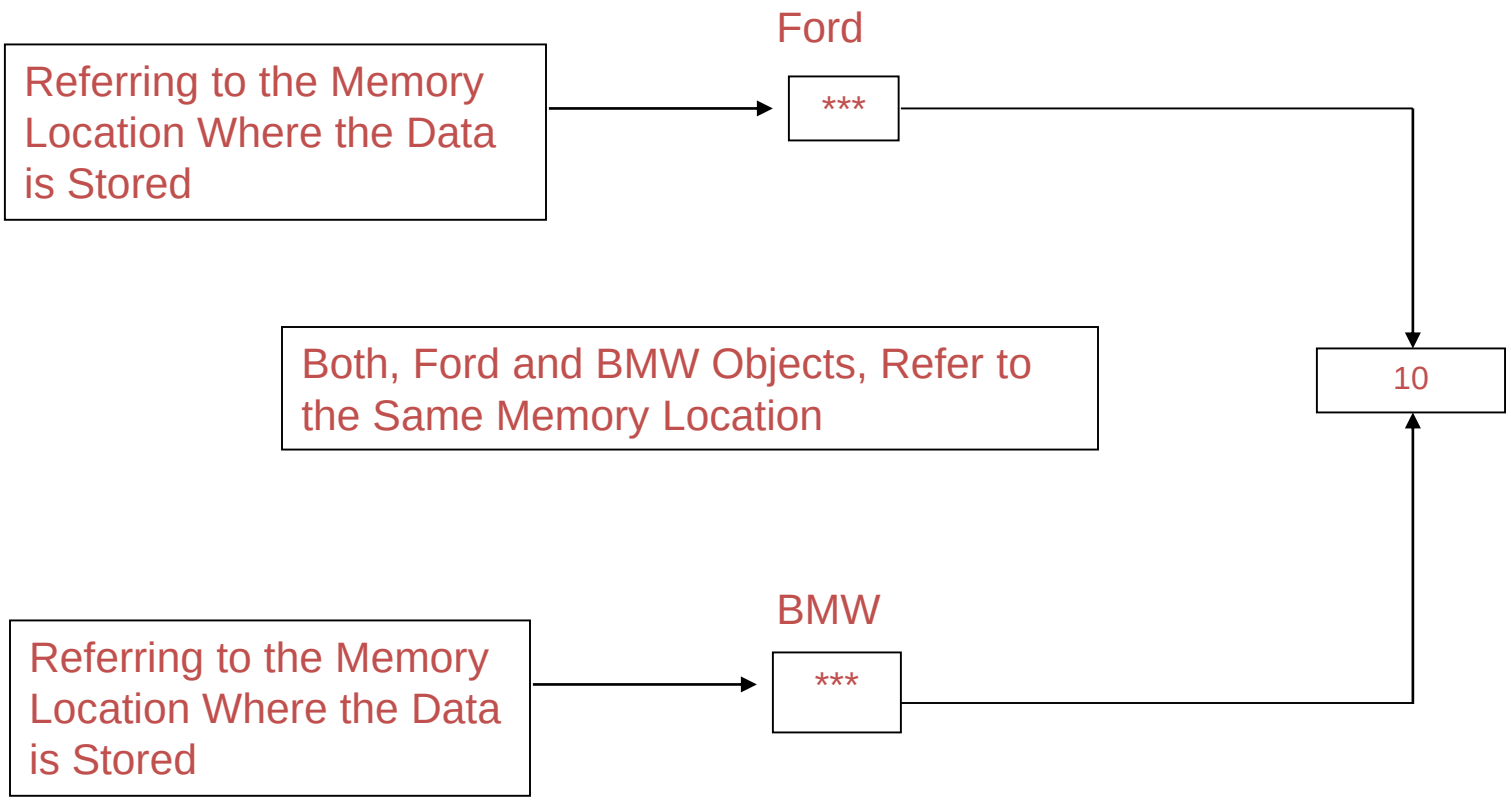
Describing Memory Allocation (Contd.)

- Reference Type:





Describing Memory Allocation (Contd.)



AlmaMate

Memory is of two forms: stack & heap

- Stack is local memory – used for value types.
- Heap is general memory – used for reference types.
- A reference type is allocated in the heap, but it’s reference variable is allocated on the stack.

Using Structures

AlmaMate

- A structure is a value type data type.
- When you want a single variable to hold related data of various data types, you can create a structure.
- To create a structure you need to use the `struct` keyword.
- **Benefits**
  - Not allocated in heap, so GC have less pressure.
  - Effective use of memory
- The following code shows how you can create a structure:

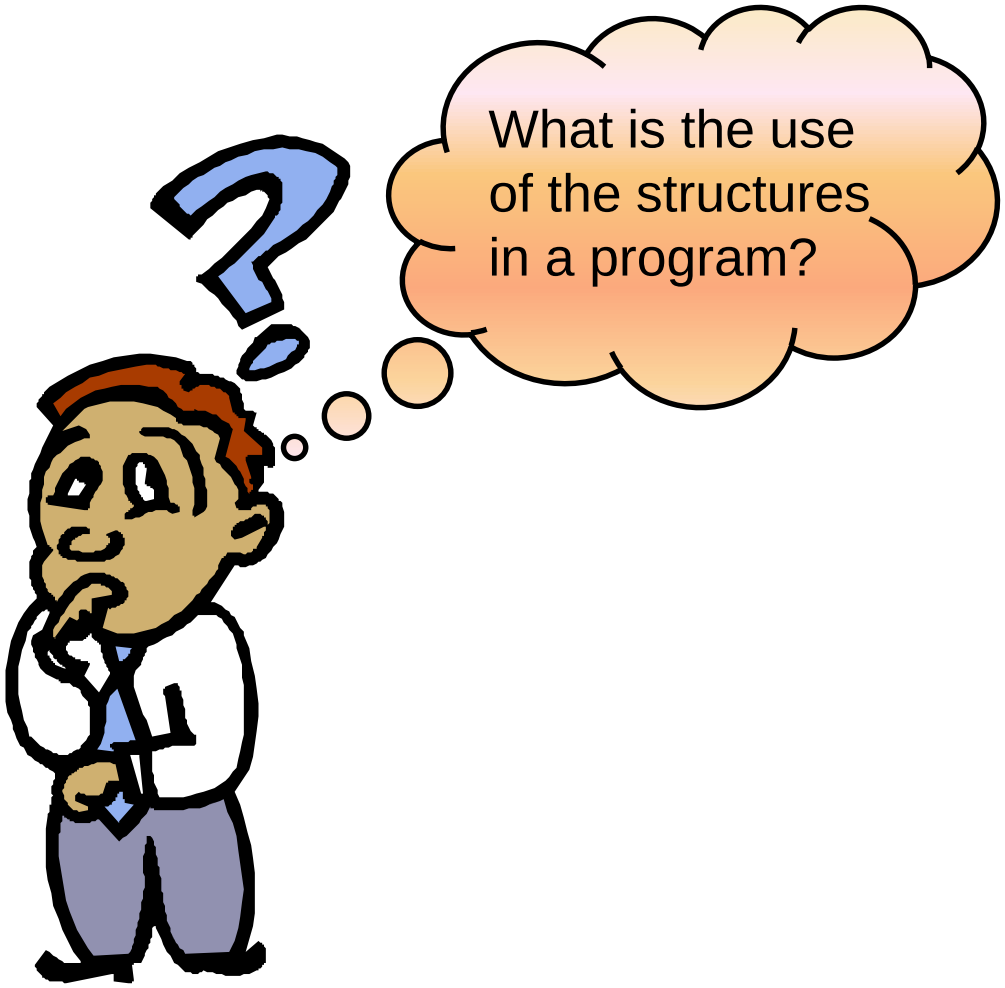
```
struct Bill_Details
{
   public string bill_No;      // Bill Number
   string ord_Dt;                // Order Date
   string custName;             // Customer name
   public string product;    // Product Name
   public double cost;         // Cost of the product
   public double due_Amt;  // Total amount due
}
```

Using Structures (Contd.)

AlmaMate

- Structures contain data members, which are defined within the declaration of the structure.
- Structure members are private by default in C#.
- Structures differ from classes and the differences are:
  - Structures are value types and get stored in a stack.
  - Structures do not support inheritance.
  - Structures cannot have default constructor.

Using Structures (Contd.)



Using Structures (Contd.)

- The following code uses the `Bill_Details` structure:

```
using System;
namespace Bills
{
            public struct Bill_Details
            {
             public string bill_No;
             public string ord_Dt;
             public string custName;
             public string product;
             public double cost;
             public double advance_Amt;
             public double due_Amt;
}
```

Using Structures (Contd.)

AlmaMate

```
class TestStructure
        {
public static void Main(string[] args)
{
   Bill_Details billObj = new Bill_Details();
    billObj.bill_No = "A101";
    billObj.ord_Dt = "10/10/06";
    billObj.custName = "Joe";
    billObj.product = "Petrol";
    billObj.cost = 100;
    billObj.advance_Amt = 50;
    billObj.due_Amt = 50;
   Console.Clear();
```

Using Structures (Contd.)

```
        Console.WriteLine("Bill Number is {0}“,
      billObj.bill_No);
          Console.WriteLine("Order Date is {0}“,
      billObj.ord_Dt);
          Console.WriteLine("Customer Name is      {0}“,
billObj.custName);
          Console.WriteLine("Product is {0}“,
      billObj.product);
          Console.WriteLine("Cost is {0}“,
      billObj.cost);
        Console.WriteLine("Advance Amount is {0}",
      billObj.advance_Amt);
          Console.WriteLine("Due Amount is {0}“,
      billObj.due_Amt);
    }
 } }
```

Using Enumerations

AlmaMate

- Enumeration is a value type data type.
- An Enumeration is an user-defined integer type,which is used for attaching names to Numbers .
- Enumerators:
  - Contain own values and cannot inherit or pass inheritance.
  - Allows you to assign symbolic names to integral constants.
  - Can be created by using the enum keyword.
  - Strongly typed
  - No implicit conversions to/from int
  - Can specify underlying base type
  - Byte, short, int, long

Declaring an Enumeration

- The following code is an example of declaring an enumeration named Days:

```
enum Days {Mon, Tue, Wed, Thu, Fri, Sat, Sun };
```

Implementing Enumerations

AlmaMate

- After declaring the enumeration, you can use the enumeration type in the same manner as any other data types.
- The following example shows the implementation of enumerations:

```
using System;
namespace EnumDays
{class EnumTest
{enum Days {Mon, Tue, Wed, Thu, Fri, Sat, Sun };
 static void Main(string[] args)
   {  int First_Day = (int)Days.Mon;
        int Last_Day  = (int)Days.Sun;
     Console.WriteLine("Mon = {0}", First_Day);
     Console.WriteLine("Sun = {0}", Last_Day);
 } } }
```

Complete Example:

AlmaMate

```
enum Employees:byte
{
  ok = 50,cancel = 100
}
```

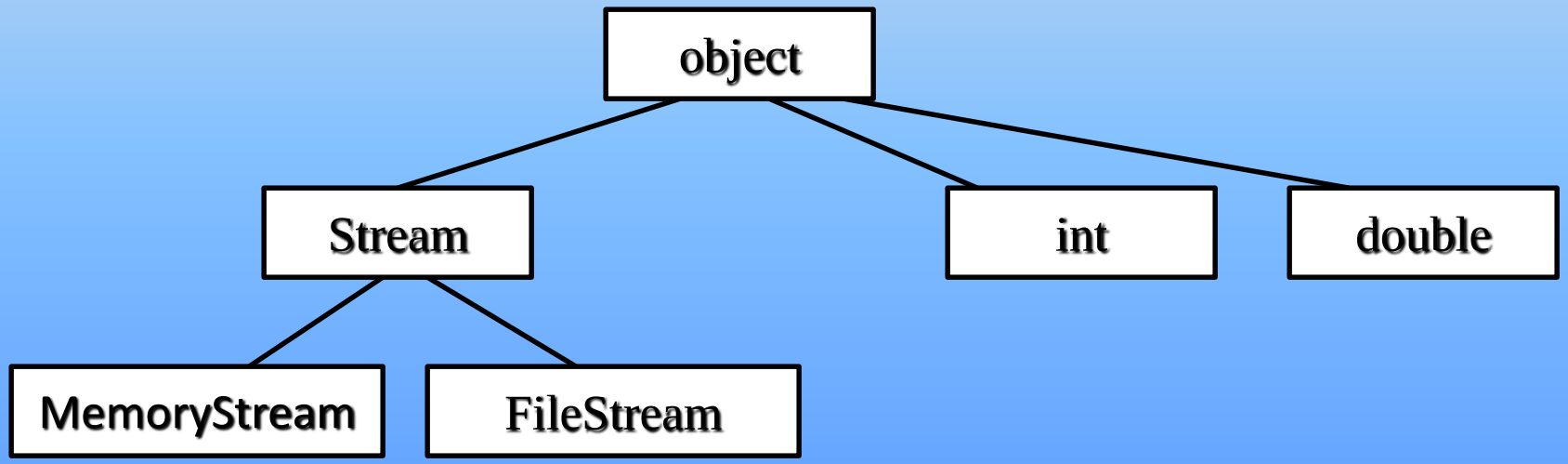
```
struct Emp
{
  public Employees em;
  public string id;
}
```

```
class Emptest
{
  public static void Main()
  {
    Emp e;
    e.em = Employees.cancel;
    e.id = "002";
    Console.WriteLine(e.em);
    Console.WriteLine(e.id);
  }
}
```

Unified Type System

AlmaMate

- **Everything is an object**
  - All types ultimately inherit from object
  - Any piece of data can be stored, transported, and manipulated with no extra work



Boxing / UnBoxing

AlmaMate

- .NET provides a unified type system.
- All types including value types derive from the type object.
- It is possible to call object methods on any value, even values of primitive types such as int.

Boxing - Converting a value type to reference type is called Boxing.
Unboxing - Is the opposite operation and is an explicit operation.

Examples

```
using System;
class Example
{
static void Main() {
Console.WriteLine(5.ToString());

}
}
```

```
class Example
{
        static void Main() {
                int  a= 2;
                object obj = a;// boxing

                int b= (int) obj;// unboxing

        }
}
```

Summary

- In this session, you learned that:
  - Memory allocated to variables is of two types, value type and reference type.
  - Variables of value types directly contain their data in a variable.
  - Reference type variables contain only a reference to a memory location containing data.
  - Structures can be used to hold related data of various data types in single variable.
  - Enumerator allows you to assign symbolic names to integral constants.