The lecture 7

Classes VS Structs

- The majority of types in a framework should be classes, but if instances of the type are small and commonly short-lived or are commonly embedded in other objects define a struct.

Class

- ✓ Declared with class keyword
- ✓ Supports inheritance
- ✓ User-defined constructors can be implemented
- ✓ Data members can be initialized in the class definition
- ✓ Reference type (Heap)

Struct

- ✓ Declared with struct keyword
- ✓ Doesn't Support inheritance
- ✓ User-defined constructors can't be implemented
- ✓ Data members can't be initialized in the struct definition
- ✓ Value type (Stack)

Struct

```
class Program
  {
      static void Main(string[] args)
      {
          Employee newEmployee = new
Employee();
          newEmployee.employeeName =
"James";
          newEmployee.employeeJob =
"Programmer";
          newEmployee.Salary = 5000;
          Console.WriteLine($"Employee
name is {newEmployee.employeeName}
and his job is {newEmployee.employeeJob}
and starting salary is
{newEmployee.Salary}");
          newEmployee.SayHi();
      }
  }
```

```
struct Employee
  {
      public string employeeName;
      public string employeeJob;
      private decimal salary;

      public decimal Salary
      {
          get { return salary; }
          set { salary = value; }
      }

      public void SayHi()
      {
          Console.WriteLine("Hi from the
method in struct");
      }
  }
```

Enumerations

```
class Program
  {
     static void Main(string[] args)
     {
        string weekDayName =
WeekDay.Monday.ToString();
        WeekDay day =
WeekDay.Sunday;

        Console.WriteLine((int)day);
     }

  }
```

```
enum WeekDay
  {
     Monday,
     Tuesday,
     Wednesday,
     Thursday,
     Friday = 40,
     Saturday = 50,
     Sunday = 60
  }
```

Interfaces

- **An interface contains definitions for a group of related functionalities that a class or a struct can implement.**
- **Think of it as contract that all the classes inheriting the interface should follow. The interface defines the '*what*' part of the contract and the deriving classes define the '*how*' part of the contract.**

Interfaces

Interface

Abstract Class

Similarities

- Can't be instantiated directly
- Must implement all its members
- Can contain events, methods, and properties.

Differences

- Can't have method implementations
- Allow multiple inheritance
- Can't have access modifiers, everything is public
- Can't contain variables

- Can have method implementations
- Doesn't allow multiple inheritance
- Can contain access modifiers
- Can contain variables

Interfaces

```
class Program
  {
      static void Main(string[] args)
      {
          Dogs dog = new Dogs();

dog.Attack();

dog.SayHi();
          dog.Run();

      }
  }
```

```
public interface IAnimals
  {
      void Run();
  }

  public interface IDogCommands : IAnimals
  {
      void Stay();
      void Sit();
      void Attack();

      string DogName
      {
          set;
          get;
      }
  }
```

```
interface Trainer
  {

  }

  class Animals
  {
      string AnimalName;

      public void SayHi()
      {

Console.WriteLine("Hi from the animals class");
      }
  }
```

```
class Dogs : Animals, IDogCommands, Trainer
  {
      private string DogBreed;

      public void Stay()
      {
          Console.WriteLine("Dog is staying");
      }

      public void Sit()
      {
          Console.WriteLine("Dog is sitting");
      }
      public void Attack()
      {
          Console.WriteLine("Dog is attacking");
      }
      public void Run()
      {
          Console.WriteLine("Animal is running");
      }

      public string DogName { get; set; }
  }
```