Session #2

REFACTORING AND SERIALIZATION

Agenda for today

- Recap on Week #1 and homework
- Revisiting some concepts and introducing a few new
- Demo
- Food
- Everybody codes

Master Class #2

After this session (and the accompanying homework) you should know:

- *Static vs Instance*
- *Lists, Interfaces and IEnumerables*
- *Better understanding of LINQ*
- *Referencing projects and managing dependencies in Visual Studio*
- *Using Nuget to find and include a package*
- *Serializing an object*

Enums

- A value type.
- The **enum** is used to declare a list of named integer constants

```
4 references
public enum Suits
{
    Spades,
    Hearts,
    Clubs,
    Diamonds
}
```

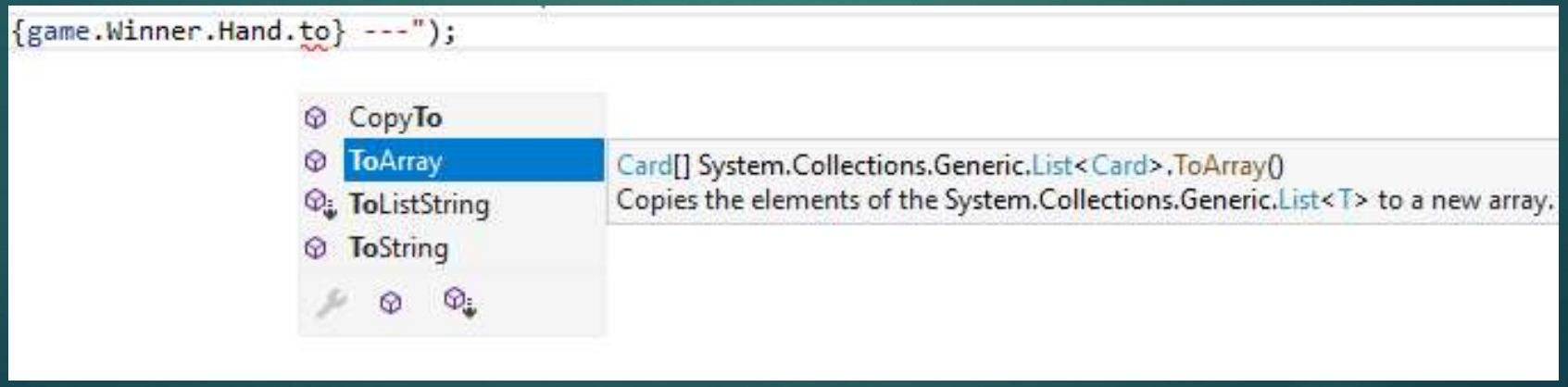
Static vs Instance

- Static methods can be called without an instantiated object: **var x=mytype.MyStaticMethod();**
- Static methods can only access static fields and properties.
- Static classes **cannot be instantiated,** its static members are accessed directly under the type name.
- Non-static methods can only be called on an instantiated object: **mytype obj=new mytype();** **var x=obj.MyMethod();**
- Non-static methods can access members and properties on the same object and use 'this'.
- Non-static classes can be instantiated (if they are not abstract). They can **also** contain static members though.

Extension Methods

- Static classes can hold static extension methods for non-static classes:

```
public static string ToListString(this IEnumerable<Card> cards)
{
    return string.Join(",", cards);
}
```



Generics

Generics allow you to define a class with placeholders for the type of its fields, methods, parameters, etc. Generics replace these placeholders with some specific type at compile time

```
class MyGenericClass<T>
{
    private T genericMemberVariable;

    public MyGenericClass(T value)
    {
        genericMemberVariable = value;
    }

    public T genericMethod(T genericParameter)
    {
        Console.WriteLine("Parameter type: {0}, value: {1}", typeof(T).ToString(),genericParameter);
        Console.WriteLine("Return type: {0}, value: {1}", typeof(T).ToString(), genericMemberVariable);

        return genericMemberVariable;
    }

    public T genericProperty { get; set; }
}
```

```
MyGenericClass<int> intGenericClass = new MyGenericClass<int>(10);

int val = intGenericClass.genericMethod(200);
```

https://www.tutorialsteacher.com/csharp/csharp-generics

Interfaces

- An interface contains definitions for a group of related functionalities that a non-abstract class or a struct must implement.
- An **interface** is like an abstract base class with only abstract members. ...
- An **interface** can't be instantiated directly. ...
- **Interfaces** can contain events, indexers, methods, and properties.
- **Interfaces** contain no implementation of methods (In **C#** 8.0, **Interfaces** can have default implementation for methods).

```
1 reference
public interface IGame
{
    5 references
    Player CurrentPlayer { get; }
    6 references
    int CurrentTurn { get; set; }
    9 references
    Deck Deck { get; set; }
    10 references
    List<Player> Players { get; set; }
    4 references
    GameState State { get; set; }
    14 references
    List<Card> Table { get; set; }
    5 references
    Player Winner { get; set; }

    3 references
    bool EvaluateIfGameOver(bool called);
    3 references
    bool NextTurn();
}
```

```
1 reference
public interface ISerializableGame
{
    2 references
    string SerializeGame();
}
```

```
13 references
public class Game : IGame, ISerializableGame
{
```

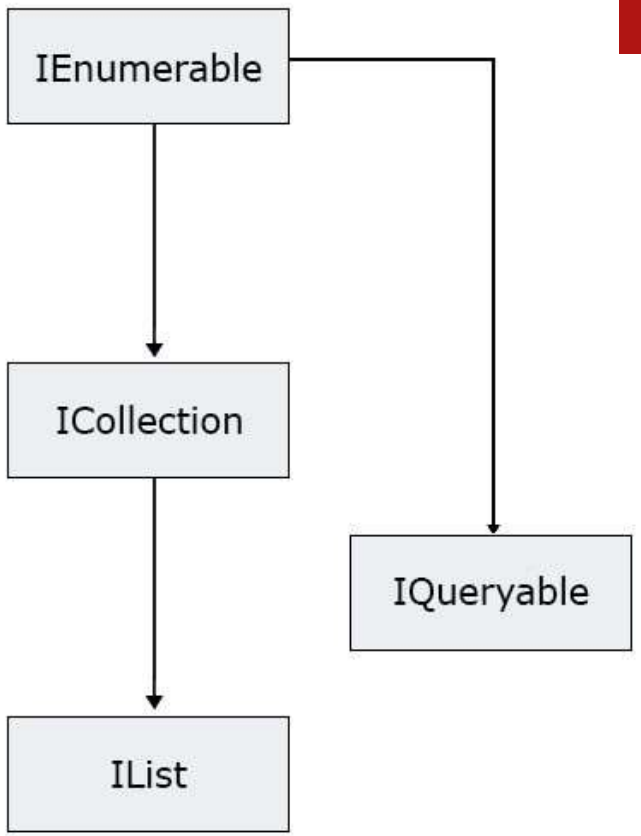
```
//Game implementation
IGame game = new Game();
```

IEnumerable

- IEnumerable: GetNext()
- ICollection:
  + Add/Delete/count
- IList:
  + Insert/RemoveAt/…

- Arrays are IEnumerables

https://medium.com/developers-arena/ienumerable-vs-icollection-vs-ilist-vs-iqueryable-in-c-2101351453db



INHERITANCE

LINQ

LINQ is a libary used to execute queries directly in C# syntax against many types of data. It is implemented as a set of **extension methods** on the IEnumerable<T> interface.

Classic vs LINQ query

Get the names of the students on the list who is over 18 classic approach:

```
List<string> AdultStudents=new List<string>();

foreach(var s in Students){

    if(s.Age>18)
    AdultStudents.Add(s.Name);

}
```

Get the names of students on the list who is over 18 LINQ approach:

```
List<string> AdultStudents=
    Students

        .Where(s => s.Age>18)

        .Select(s => s.Name)

        .ToList();
```

Console Example

```
0 references
static void Main(string[] args)
{
    //Load wordlist
    Words=File.ReadAllLines(@"C:\Users\AllanThraen\OneDrive - CodeArt ApS\Data\Ordbog\Opdatering af RO 2019 til aftagere\RO2012 fuldformer 2019.txt")
        .Select(l => l.Split(';').Skip(1).First().ToLower())
        .ToList();

    Console.WriteLine("Available letters: ");
    var basechars = Console.ReadLine().ToLower().ToArray();


    //Which words consists only of these chars?
    var lst=Words.Where(w => w.Distinct().All(c => basechars.Contains(c) && w.Count(cc => cc==c)==basechars.Count(cc => cc==c) ))
        .OrderByDescending(w => w.Length)
        .ToList();

    foreach(var l in lst)
    {
        Console.WriteLine(l);
    }
}
```

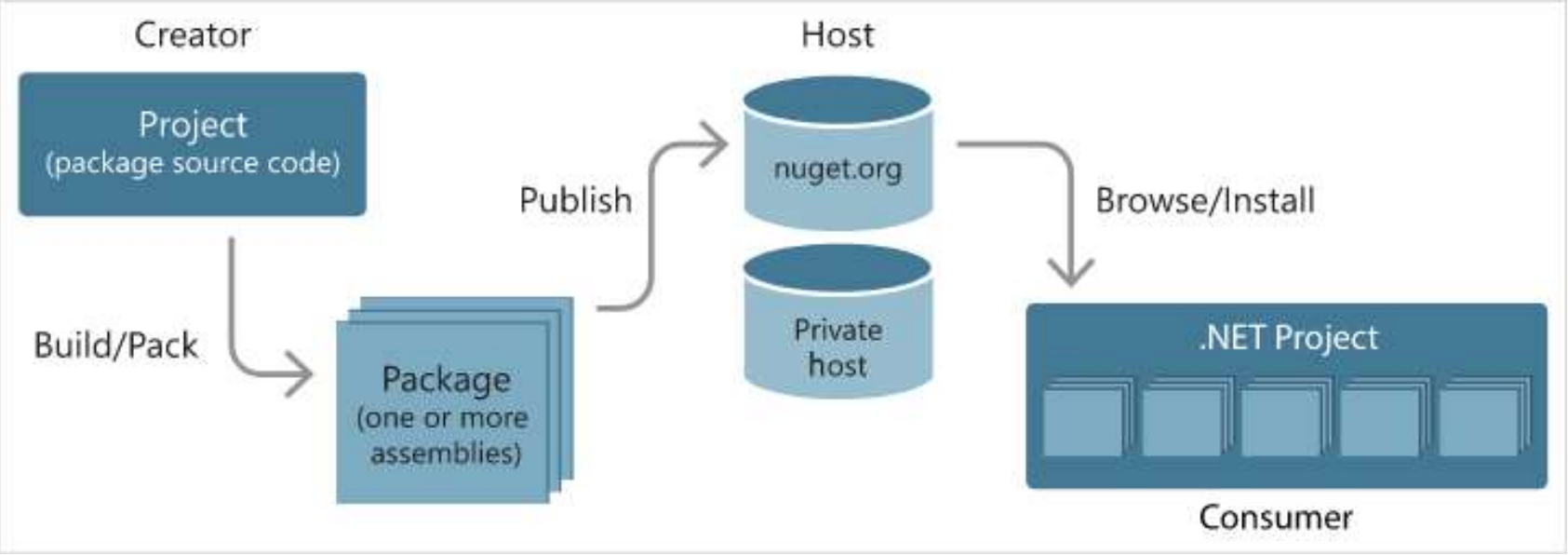
Code reuse

Class Libraries and Linking

A class library is a library of code

It typically compiles to a .dll file (dynamic linked library)

You can link it, by referencing it, in other projects

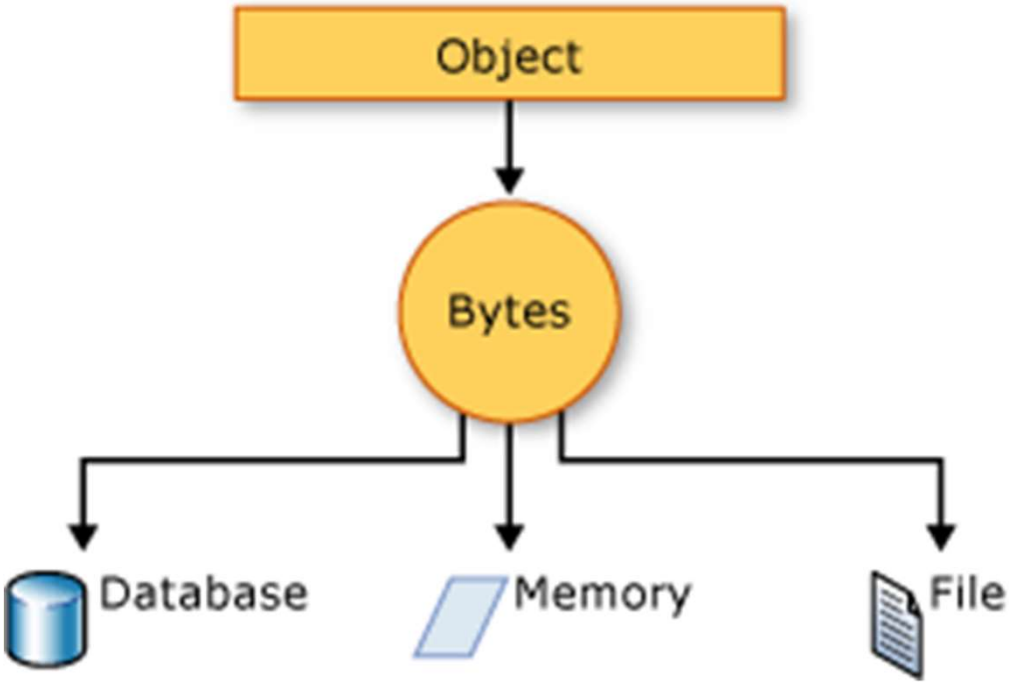


Nuget packages

https://docs.microsoft.com/en-us/nuget/what-is-nuget

Serialization

- To serialize is to turn an object into bytes so it can be transported or stored.
- Json is often used, so is xml or csv




Code time

Homework after session #2

- Reorganize and refactor code into class library
- Use Newtonsoft.Json nuget package
- Serialize / deserialize games
- Try it out