Session #4

COMPLETING THE WEB GAME

Agenda for today

- Recap on Week #3 and homework
- Revisiting some concepts and introducing a few new
- Demo
- Food
- Everybody codes

Master Class #4

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

- *Exceptions and Exception handling*
- *IoC and Dependency Injection*
- *Controllers, actions and views*
- *Understand the game logic*

Exceptions

- When a run-time error happens, an **exception** is **thrown.**
- Exceptions are **objects** as well. Defined in **classes**.
- **Inherits** from **System.Exception**
- You can throw your own using **throw new Exception();**
- Exceptions can be caught and **handled** in **try-catch** blocks
- Make sure to wrap must-execute code in **finally** blocks

```
0 references
static void Main(string[] args)
{
    if (args.Length == 0)
        throw new ApplicationException("You need to provide some arguments");

    try
    {
        int x = 100 / int.Parse(args[0]);
    }
    catch (FormatException)
    {
        Console.WriteLine("First parameter must be a number");
    } catch(DivideByZeroException)
    {
        Console.WriteLine("First parameter cannot be zero");
    } catch(Exception e)
    {
        //Some other exception. Rethrow
        throw e;
    }
    finally
    {
        //Do stuff that should always be done - even in exception scenario.
    }
}
```

Inversion of Control - IoC

Inversion of Control (IoC) means to create instances of dependencies first and latter instance of a class (optionally injecting them through constructor), instead of creating an instance of the class first and then the class instance creating instances of dependencies.

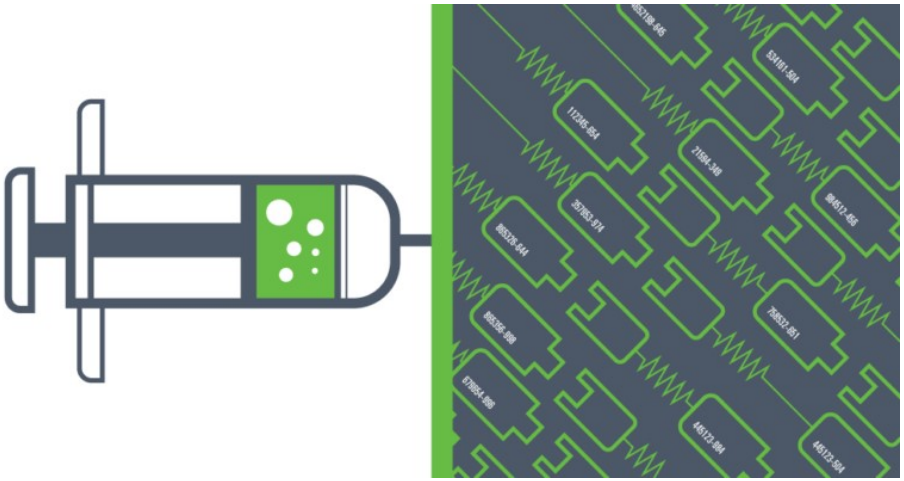
- Hard to replace other class
- Hard to create unit tests
- Less flexible

```
1 reference
public class MyClass
{
    1 reference
    public MyOtherClass OtherClass { get; set; }

    0 references
    public MyClass()
    {
        OtherClass = new MyOtherClass();
    }
}
```

Dependency Injection

- An **IoC** design pattern
- Inject dependencies into classes
- Uses a central service registration engine
- Often uses Interfaces as base types to inject
- Provided in popular opensource projects as StructureMap and Castle.Core
- Comes out-of-the-box in ASP.NET Core



Example - GameService

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddTransient<IGameService, GameService>();
}
```

```
private readonly IGameService _gameService;

0 references
public GameController(IGameService gs)
{
    _gameService = gs;
}
```

Game logic / steps

1. Extend GameController with MakeAMove action method
    1. Move is either Knock, DrawFromDeck or DrawFromTable
2. Extend GameController with action method to drop card
3. Change main game view (Index) to support actions
4. Add GameOver view

Code time

Homework after session #4

- Prepare GameService for Dependency Injection
- Complete single-player game
- Add player action handling
- GameOver action and view