

DAD 2023-24

Lab. 1 – Introduction to C# and gRPC

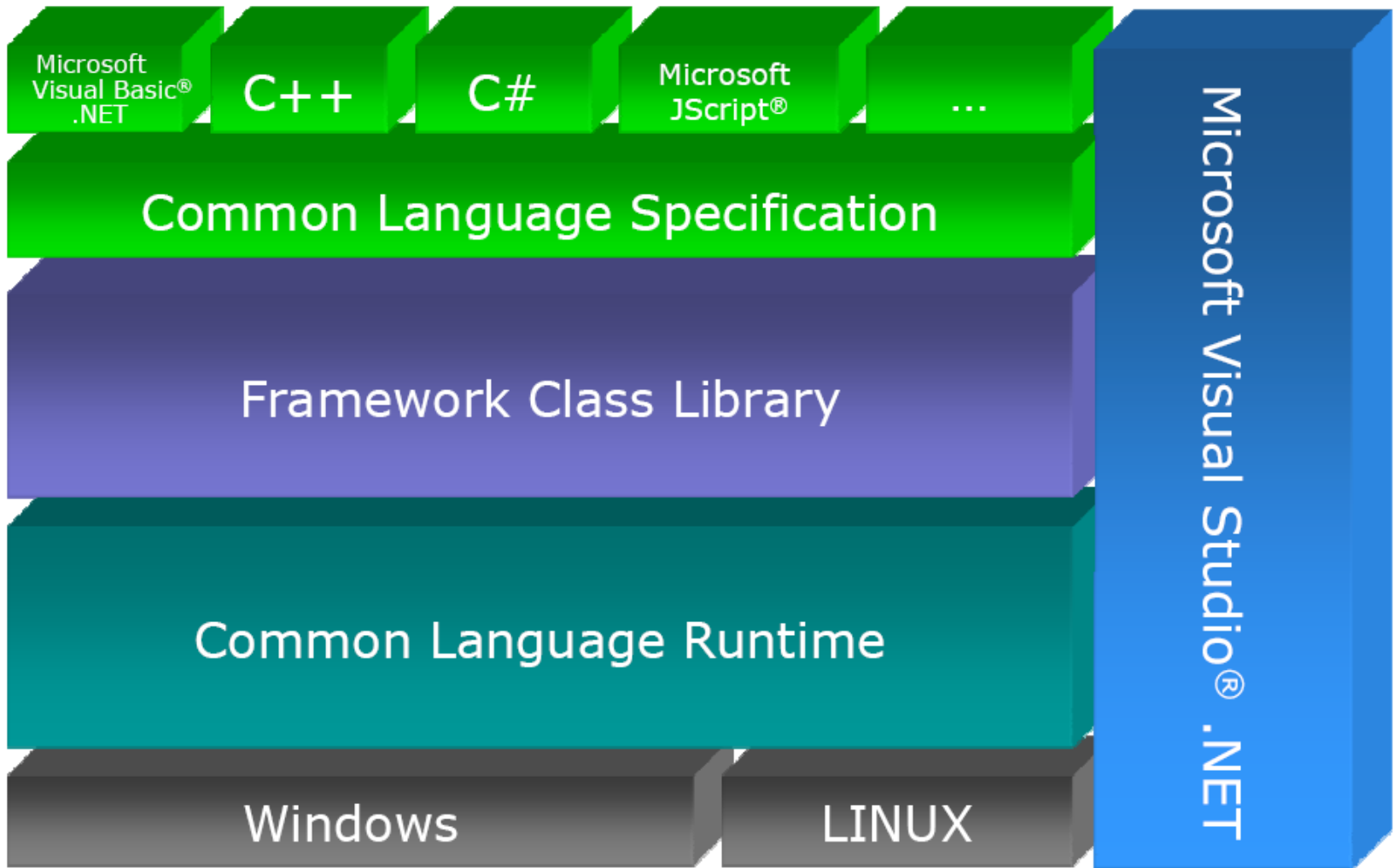
Summary

1. .NET Framework
2. C# Language
3. IDE: MS Visual Studio
4. Asynchronous Programming
5. gRPC

1. .NET Framework

Introduction
Architecture

.NET Framework Architecture

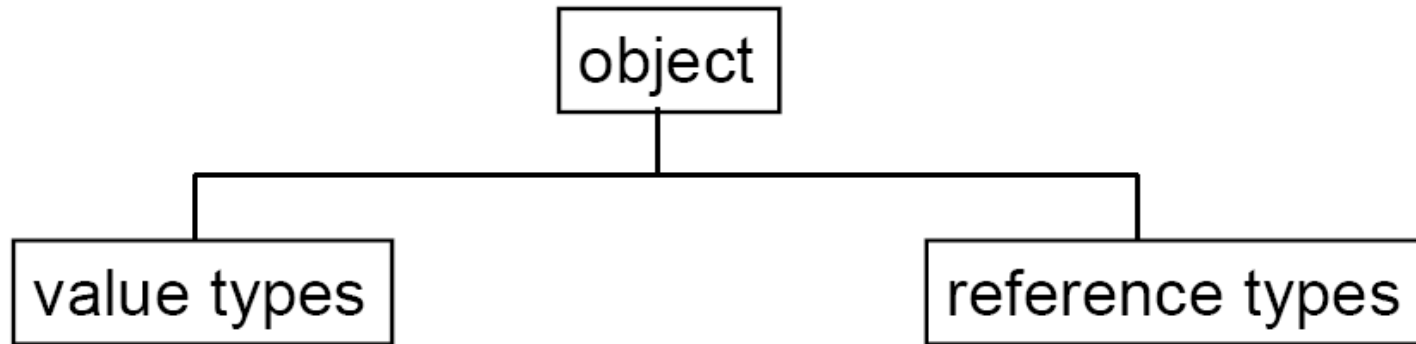


Framework Class Library

- System
- System.Collections
- System.Drawing
- System.IO
- System.Data
- System.Windows.Forms
- System.Web.UI
- System.Web.Services
- ...

Common Language Runtime

- Execution Environment
- Memory Management
- Garbage collection
- Common type system



- | | |
|---|--|
| <ul style="list-style-type: none">• Primitive types (int, double, tuples, etc..)• Stack allocated• Assignment copy values• Freed at the block's end• User-defined: struct, enum | <ul style="list-style-type: none">• Classes, arrays, ...• Allocated on heap• Assignments don't copy values• Garbage collected |
|---|--|

.NET: Main Advantages

- Virtual execution environment.
- Many libraries.
- APIs for web development.
- Language interoperability.
- New standard: C#

C#

Basic Syntax:

It's very similar to Java...

Hello World

```
using System;
```

```
public class HelloWorld {  
    public static void Main(string[]  
args) {  
        Console.WriteLine("Hello World!");  
    }  
}
```

A simple Class

```
public class Person {
    private string name;
    private int age;
    public Person(string name, int age) {
        this.name = name;
        this.age = age;
    }

    public void ShowInfo() {
        Console.WriteLine("{0} is {1} years old.", name, age);
    }
}
[...]
```

```
Person client = new Person("John", 25);
client.ShowInfo();
```

Execution Control

- if, for, do, while, switch, foreach...

- switch without fall-through (needs break, goto or return):

```
switch a {
case 2:
    x = 4;
    goto case 3
// explicit fall-through
case 3:
    ...
}
```

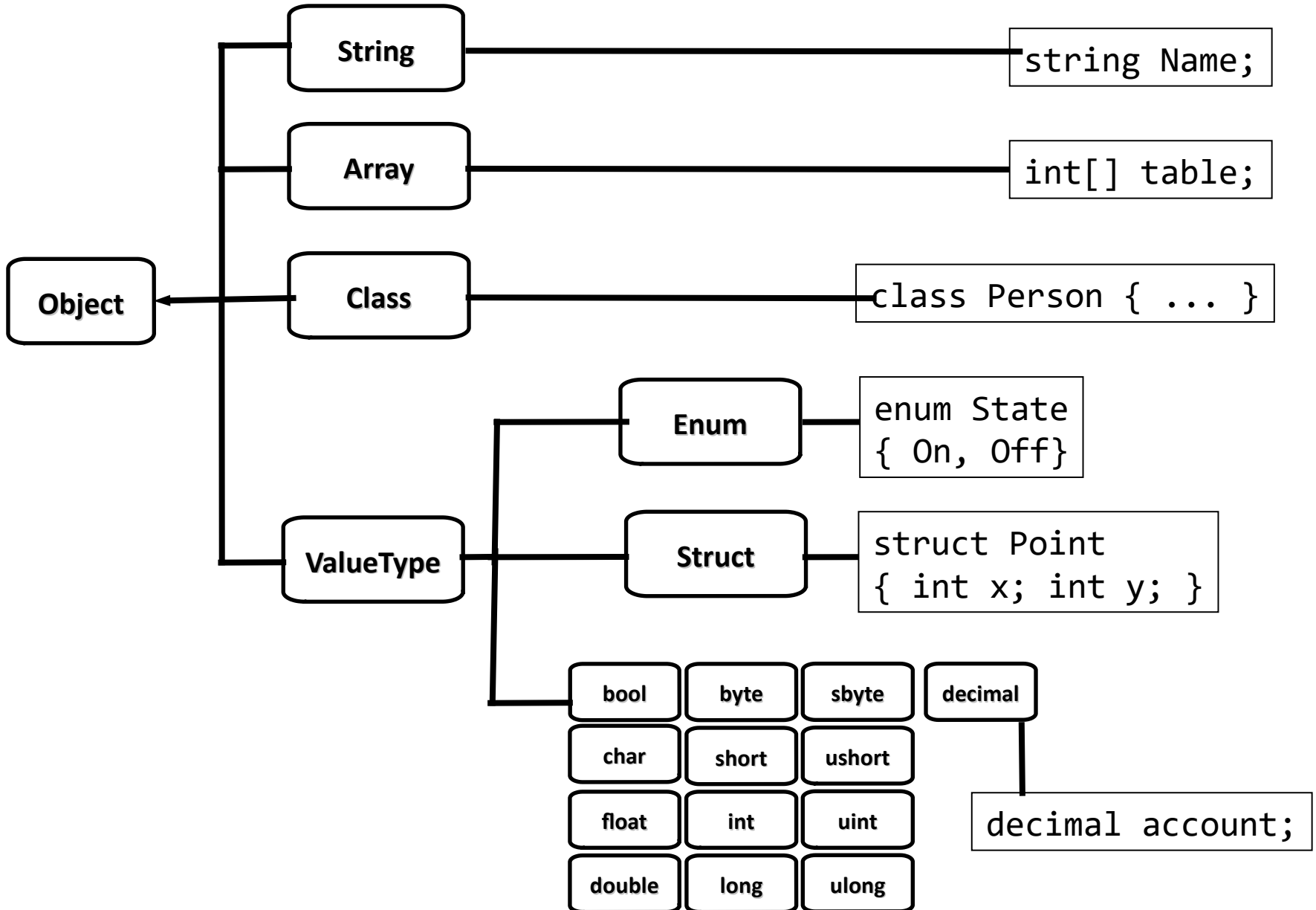
- Switch with pattern matching:

```
switch (shape) {
    case Square s:
        return s.Side * s.Side;
    case Circle c:
        return c.Radius * c.Radius *
            Math.PI;
}
```

Classes

- Name hierarchy: namespaces
- Simple class inheritance.
- Multiple interface inheritance.
- Class members:
 - Fields, methods, properties, indexers, events,..
 - Access levels: `public`, `protected`, `internal`, `private`
 - Members can be `static` or `instance`.
 - `abstract` members also possible.

C#: Type System



C#: Inheritance

```
public class Person
{
    private string name;

    public Person(string name) {
        this.name = name;
    }

    public virtual void ShowInfo()
    {
        Console.WriteLine("Name:{0}",
            name);
    }
}
```

```
public class Employee : Person
{
    private string company;

    public Employee(string name,
                    int company)
        : base(name)
    {
        this.company = company;
    }

    public override void ShowInfo() {
        base.ShowInfo();
        Console.WriteLine("Company: {0}",
            company);
    }
}
```

- By default, methods are not virtual!

C#: Lambda Expressions

A lambda is an anonymous function of the format:

```
(input-parameters) => expression
```

or

```
(input-parameters) => { <sequence-of-statements> }
```

```
// Example 1
```

```
Func<int, int, bool> testForEquality = (x, y) => x == y;  
Console.WriteLine(testForEquality(4,4));
```

```
//Example 2
```

```
int[] numbers = { 2, 3, 4, 5 };  
var squaredNumbers = numbers.Select(x => x * x);
```

Asynchronous Programming

+

Intro to G-RPC

Async Programming

Asynchronous Programming

- Asynchronous Programming allows abstract concurrent activities without Thread management.
- Async. Prog. uses the abstraction of **Tasks**.
- Tasks can be waited on until the asynchronous activity (e.g. I/O) is done.
- Tasks can be started explicitly on different threads.

Task & Task<TResult>

- A Task represent an asynchronous operation.
- They can be waited on.
- Task<TResult> return a TResult.

```
// Create a task and supply a user delegate by using a  
lambda expression.
```

```
Task taskA = new Task( () => Console.WriteLine("Hello from  
taskA.")).Start();
```

```
// Start the task.
```

```
taskA.Start();
```

```
taskA.Wait();
```

async

- Allows running code asynchronously on a runtime managed thread pool.
- Async methods can contain await-ed operations.
- Async methods return a Task or Task<TResult>
- See:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>

await

- Can be called on anything implementing the `GetAwaiter` method.
- Blocks the current thread until an asynchronous result is returned.
- For example:

```
Task<int> downloading = DownloadWebpageAsync();  
// do something else and then get the results  
int bytesLoaded = await downloading;
```

G-RPC in C#

G-RPC Proto Buffers

- Specify the protocol between client and server: the service interfaces in a language agnostic syntax (see <https://developers.google.com/protocol-buffers/docs/proto3>)

```
syntax = "proto3";
```

```
service ChatServerService {  
    rpc Register (ChatClientRegisterRequest) returns  
    (ChatClientRegisterReply);  
}
```

```
message ChatClientRegisterRequest {  
    string nick = 1;  
    string url = 2;  
}
```

```
message ChatClientRegisterReply {  
    bool ok = 1;  
}
```

Server (1)

- Implements Services described in Protobuf:

```
// ChatServerService is the namespace defined in the protobuf
// ChatServerServiceBase is the generated base implementation of the service
public class ServerService : ChatServerService.ChatServerServiceBase {
    // example of Server data structure
    Dictionary<string, string> clientMap = new Dictionary<string, string>();

    public ServerService() {
    }

    public override Task<ChatClientRegisterReply>
        Register( ChatClientRegisterRequest request, ServerCallContext
        context) { return Task.FromResult(Reg(request));
    }
}
```


Server (2)

```
public ChatClientRegisterReply Reg(ChatClientRegisterRequest request) {
    lock (this) {
        clientMap.Add(request.Nick, request.Url);
    }
    return new ChatClientRegisterReply
    {
        Ok = true
    };
}
```

Server (3)

- Responds to client requests.
- Grpc.Core.Server is multithreaded!

```
static void Main(string[] args) { int Port = 50051;
Server server = new Server
{
    Services = { ChatServerService.BindService(new
                ServerService()) },
    Ports = { new ServerPort("localhost", Port,
                ServerCredentials.Insecure) }
};

    server.Start(); Console.ReadKey();
        server.ShutdownAsync().Wait();
    }
```

Client

- Can do calls to a server.

- Steps:

- **Disable HTTPS (optional):**

```
AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2UnencryptedSupport", true);
```

- **Create Channel:**

```
GrpcChannel channel =  
GrpcChannel.ForAddress("http://localhost:50051");
```

- **Create Client:**

```
var client = new  
ChatServerService.ChatServerServiceClient(channel);
```

- **Do calls:**

```
client.Register(registerRequest);
```

Server Development

- Create Visual Studio Project
- Add code package (Tools->NuGet Package Manager):
 - Grpc.Core, which contains the .NET G-RPC Core.
 - Google.Protobuf, which contains protobuf message APIs for C#.
 - Grpc.Tools, which contains C# tooling support for protobuf files.
- Add proto folder and protobuf file
- Add protobuf to project by adding following line to the G-RPC ItemGroup in the project file (csproj file):

```
<Protobuf Include="protos\ChatServices.proto"  
  GrpcServices="Server" />
```
- Implement services
- Add server start code. Done! ;-)

Client Development

- Create Visual Studio Project
- Add code packages (Tools->NuGet Package Manager):
 - Grpc.Net.Client, which contains the .NET Core client.
 - Google.Protobuf, which contains protobuf message APIs for C#.
 - Grpc.Tools, which contains C# tooling support for protobuf files.
- Add protos folder and copy of server protobuf file
- Define client namespace in protobuf file:

```
option csharp_namespace = "ChatClient";
```
- Add protobuf to project by adding following line to the G-RPC ItemGroup in the project file (csproj file):

```
<Protobuf Include="Protos\ChatServices.proto"  
GrpcServices="Client" />
```
- Add client code: create Channel, Client and server calls. Done! ;-)