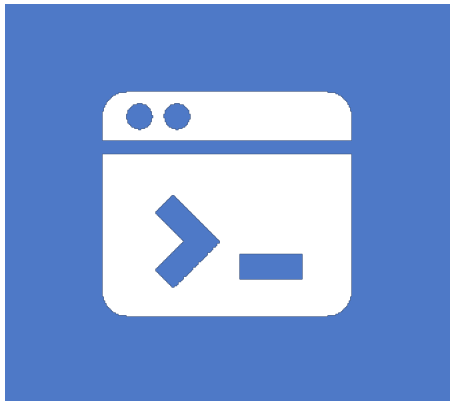
# Advanced Game Programming



Week 5

# Homework Review

Hard Problems

# Serialization

Dealing with State

# What is Serialization?

- **Serialization**: Process of transforming objects in memory into a sequence of bytes.
- **Deserialization:** Converting a sequence of bytes into an object in memory.
- Its purpose is storage and transmission of data, to be reconstructed later.

# Why Serialize Data?

- Save systems
- Communication over network
- Designing levels
- Data driven game design

- Considerations:
  - Compression
  - Versioning (forward and backward compatibility)
  - Human readable / editable
  - Cross-platform readability

# Formats - CSV

- Stands for “comma separated values” (TSV for tabs also exists)
- Pros:
  - Can edit in Excel or other spreadsheet application
  - Easy for humans to read and write.
- Cons:
  - Not compressed
  - Everything is a string

```
Los Angeles,34°03′N,118°15′W
New York City,40°42′46″N,74°00′21″W
Paris,48°51′24″N,2°21′03″E
```

# Formats - XML

- Stands for eXtensible Markup Language
- It’s a markup language, like HTML
  - Unlike HTML, *XML tags are not predefined*
- Designed to tag information
- Pros:
  - Human readable and writable
  - Easy to add more information in the future without breaking past data
  - Easy to write a parser
- Cons:
  - Everything is a string

# XML Example

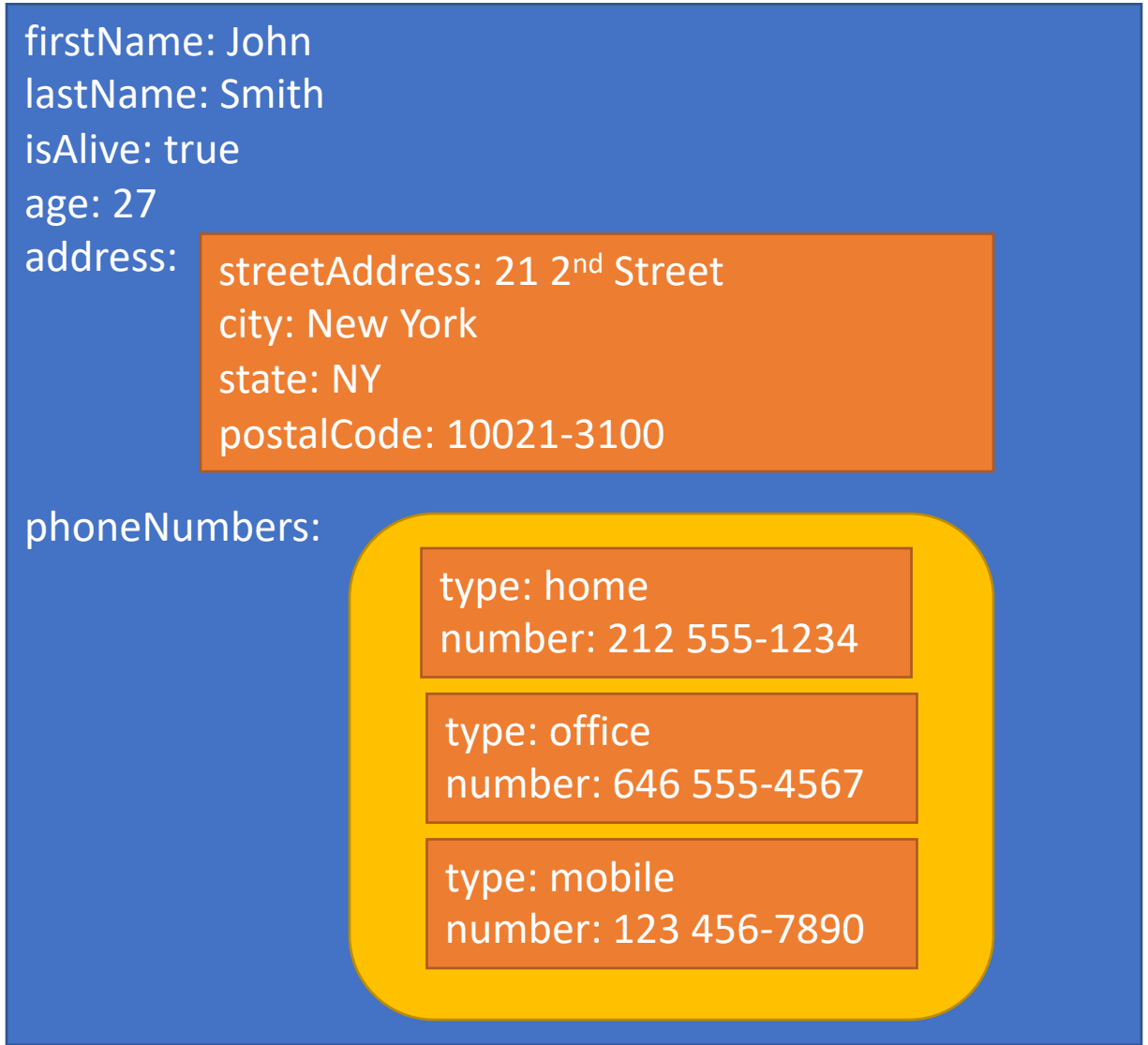
```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

# Formats - JSON

- Stands for “Javascript Object Notation”
- Contains two types:
  - Collection of name/value pairs
    - Names are strings (“name”)
    - Values are strings, numbers* or booleans
  - Ordered list or sequence
- Pros
  - Easy-ish for humans to read and write
  - Libraries exist for most languages
  - Good for working on the web (JavaScript automatically understands)
  - Data comes in structured, but no types
  - Widely adopted
- Cons
  - Not compressed

# JSON Example

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```



# Formats - YAML

- Stands for “YAML Ain't Markup Language” (seriously)
- Based on the PERL programming language
- Encodes strings, integers, and floats, lists, and maps (dictionaries)
- Pros:
  - Easy for humans to read and understand
  - Data comes in structured, and can declare types
- Cons:
  - Indentation is important (easy to make mistakes)
  - Not compressed

# YAML Example

```yaml
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     133.7
      quantity:  1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
    state:  KS

ship-to:  *id001

specialDelivery:  >
    Follow the Yellow Brick
    Road to the Emerald City.
    Pay no attention to the
    man behind the curtain.
...
```

# Formats - Binary

- You can choose to write or transmit individual bytes
- Pros:
  - Most compressed
- Cons:
  - Not human readable
  - Must create something to interpret and write out on both sides

# Other Formats

- Flatbuffers
- SOAP
- BSON
- MessagePack
- protobuf

# Solution 1 – [Serializable] Class

- Pros:
  - Quick and Easy to code
  - C# handles for you
- Cons:
  - If you change anything, your game breaks
  - No control over how data is stored
  - May not be compressed
  - May not be easy to use with other tools / languages
  - If it’s a monobehaviour that’s serializable, it has major impacts on the editor behaviour

# [Serializable] Class Example

*Github Repo for Unity Project*

*Scene “Class Serialization Example”*
*Script ”ClassSerializationExample.cs”*

# Finally, it’s time for generic types!

- What is a generic?
  - A generic allows you to use a type as a parameter for a *class* or *method*
- Syntax

```
public class GenericClass<T> {
        public void Function(T input) { }
}

private T GenericMethod<T> { }

private void Method<T> where T : BaseClass { }
```

- Purpose
  - Reusability
  - Performance
  - Type safety

# With Generic Types

```
Jack Schlesinger *
public void WriteObject<T>(string fileName, T toWrite) where T : ISerializable {
    IFormatter formatter = new BinaryFormatter();
    Stream stream = new FileStream( path: fileName + ".bin", FileMode.Create, FileAccess.Write, FileShare.None);
    formatter.Serialize(stream, toWrite);
    stream.Close();
}
```

```
new *
public T ReadData<T>(string fileName) where T : ISerializable
{
    IFormatter formatter = new BinaryFormatter();
    Stream stream = new FileStream( path: fileName + ".bin", FileMode.Open, FileAccess.Read, FileShare.Read);
    var toReturn = (T) formatter.Deserialize(stream);
    stream.Close();

    return toReturn;
}
```

# Solution 2 – Writing To String

- Pros:
  - Can make more compressed
  - Control how and what data is stored
  - Versioning control
- Cons:
  - Need to manage what is written out
  - Less quick to create

# String Class Example

*Github Repo for Unity Project*

*Scene “Class Serialization Example”*
*Script ”ClassSerializationExample.cs”*

# Finally, it’s time for interfaces!

- What is an interface?
  - An interface is a type definition, that represents a contract between the object and its user. It can contain method and property declarations, but cannot be directly instantiated as an object, nor can its data members be defined.
  - Although a class cannot inherit from multiple classes, a class can implement multiple interfaces.
  - Access specifiers (i.e. **private, internal**, etc.) cannot be provided for interface members, as all members are public by default.
  - There are no static methods.
- Syntax

```
public interface IInterfaceName {
        void Method();
        void variable { get; set; }
}
```

- Purpose
  - Guarantee that multiple classes implement certain functions

# Important Notes

- *Always close your file streams.*
- Often, it’s easiest to choose data types that are human readable until it becomes a block

# Considerations Of Save System

- Forwards Compatibility: If you add more saved information in the future, should old save files still work?
  - Need to include versioning if so.
- Human readable: Should you be able to read the information saved?
- Cross-platform: Do you need to interact with this data in another application?  Do you want to put it in a database, or on the web?
- Frequency of saves: Do you want to save every move?  Every load?  Every frame? What makes sense for your game?
- Data to save: Do you save every entity's location?  Just the location of nearby enemies?