SAVING AND READING DATA

Serialization, Deserialization, and Reflection

```
public class Shape
{
    public Color StrokeColor;
    public float StrokeThickness;
    public float X;
    public float Y;
    public float Width;
    public float Height;
    public Color FillColor;
    public bool Filled;
    public ShapeType Type;
}
```

Some Classes are Just Data

- In general classes should be immutable
- But sometimes it is just data
- Consider the shape class
- We may want to edit it
- We may want to save / load it to disk
- Yes we can make it immutable
- Too much work, too little ROI

Every time you change a field if immutable

- You have to construct a new object
- And copy all of the previous fields over
- This is a lot of superfluous code
- The advantages of immutability are insufficient

AN ASIDE: MODERN C# HAS A SOLUTION

They are called records and with expressions

```
public record ShapeRecord(
    Color StrokeColor,
    float StrokeThickness,
    float X,
    float Y,
    float Width,
    float Height,
    Color FillColor,
    bool Filled,
    ShapeType Type
);
```

RECORD EXAMPLE

```
public class InheritanceExample
{
    public record Point(int X, int Y);
    public record NamedPoint(string Name, int X, int Y) : Point(X, Y);

    public static void Main()
    {
        Point p1 = new NamedPoint("A", 0, 0);
        Point p2 = p1 with { X = 5, Y = 3 };
        Console.WriteLine(p2 is NamedPoint);  // output: True
        Console.WriteLine(p2);  // output: NamedPoint { X = 5, Y = 3, Name = A }
    }
}
```

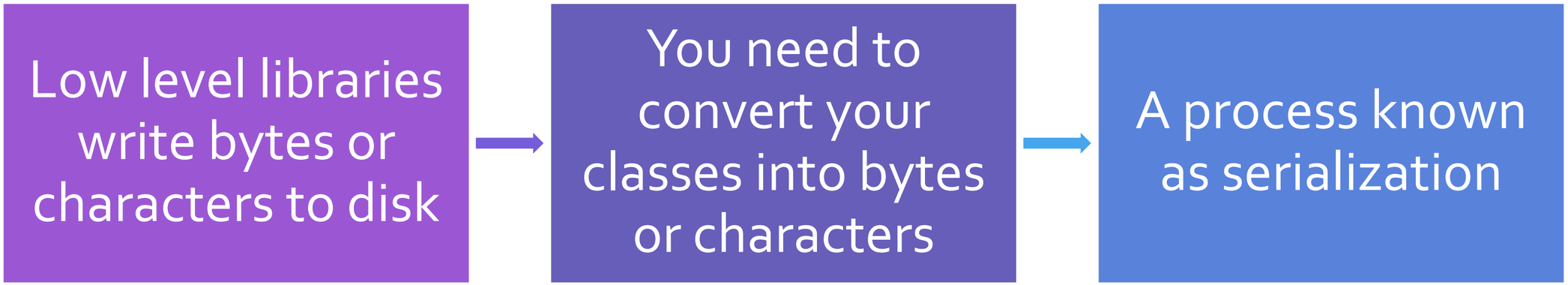
Either way, we still a have a problem

We still need to solve the problem of notification

When data models change the application must react

We will talk about this in a separate lecture

How do you save data to disk?



Serialization / Deserialization

Serialization is the process of converting objects into bytes (or characters) so that they can save to disk or transmitted over a network.

Deserialization is the process of reconstituting byte or character streams back into the original objects

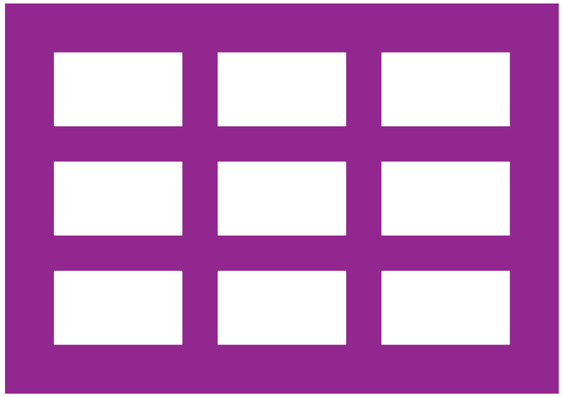
How Serializers work

BY QUERYING OR SETTING FIELDS AT RUNTIME

THIS IS CALLED REFLECTION (AKA INTROSPECTION)

Reflection

- Reflection allow objects to query information about objects
- For example: what are the names and types of the fields?
- What values does each field hold?
- In C# we can query this kind of information about any object
- We can also change the values of fields at run-time
- We can even invoke methods on an object at run-time



Example Reflection Code

```
[Test]
public void TestReflection()
{
    var t = MyShape.GetType();
    foreach (var fi in t.GetFields())
    {
        var name= fi.Name;
        var value = fi.GetValue(MyShape);
        Console.WriteLine($"Field {name} has value {value}");
    }
}
```

Output of Reflection Demo

```
Field StrokeColor has value Color [SaddleBrown]
Field StrokeThickness has value 2
Field X has value 5
Field Y has value 10
Field Width has value 12
Field Height has value 13
Field FillColor has value Color [Blue]
Field Filled has value True
Field Type has value Ellipse
```

How do you read data from disk?

You need to convert from a stream of characters or bytes

Doing this in a robust manner is very hard

How do you put the values in the right place?

The solution

Use a standardized format like XML or JSON

These are formats designed specifically for this purpose

Also, they can be read as plain text!

Serializer Libraries Exist

- XmlSerializer
- JsonSerializer
- BinarySerializer

```
<?xml version="1.0" encoding="utf-16"?>
<Shape xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xmlns:xsd="http://
  www.w3.org/2001/XMLSchema">
  <StrokeColor />
  <StrokeThickness>2</StrokeThickness>
  <X>5</X>
  <Y>10</Y>
  <Width>12</Width>
  <Height>13</Height>
  <FillColor />
  <Filled>true</Filled>
  <Type>Ellipse</Type>
</Shape>
```