Serialization

By Pakita Shamoi

Introduction

- Many applications need to store or transfer data stored in objects.
- To make these tasks as simple as possible, the .NET Framework includes several serialization techniques.
- These techniques convert objects into binary, Simple Object Access Protocol (SOAP), or XML documents that can be easily stored, transferred, and retrieved.

SERIALIZING OBJECTS

- If you want to :
  - store the objects in a file
  - send an object to another process
  - transmit it across the network

  you do have to think about how the object is represented because you will need to convert it to a different format.

- This conversion is called ***serialization***

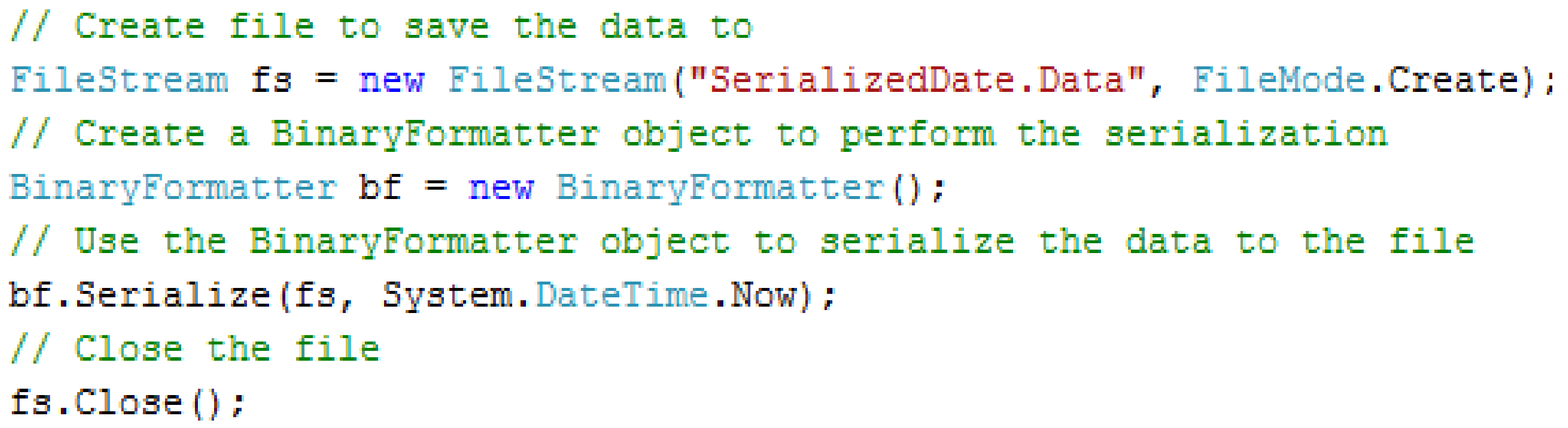
DEFINITIONS

- Serialization is the process of converting an object into a linear sequence of bytes that can be stored or transferred.
- Deserialization is the process of converting a previously serialized sequence of bytes into an object.

- Basically, if you want to store an object (or multiple objects) in a file for later retrieval, you store the output of serialization.
- The next time you want to read the objects, you call the deserialization methods, and your object is re-created

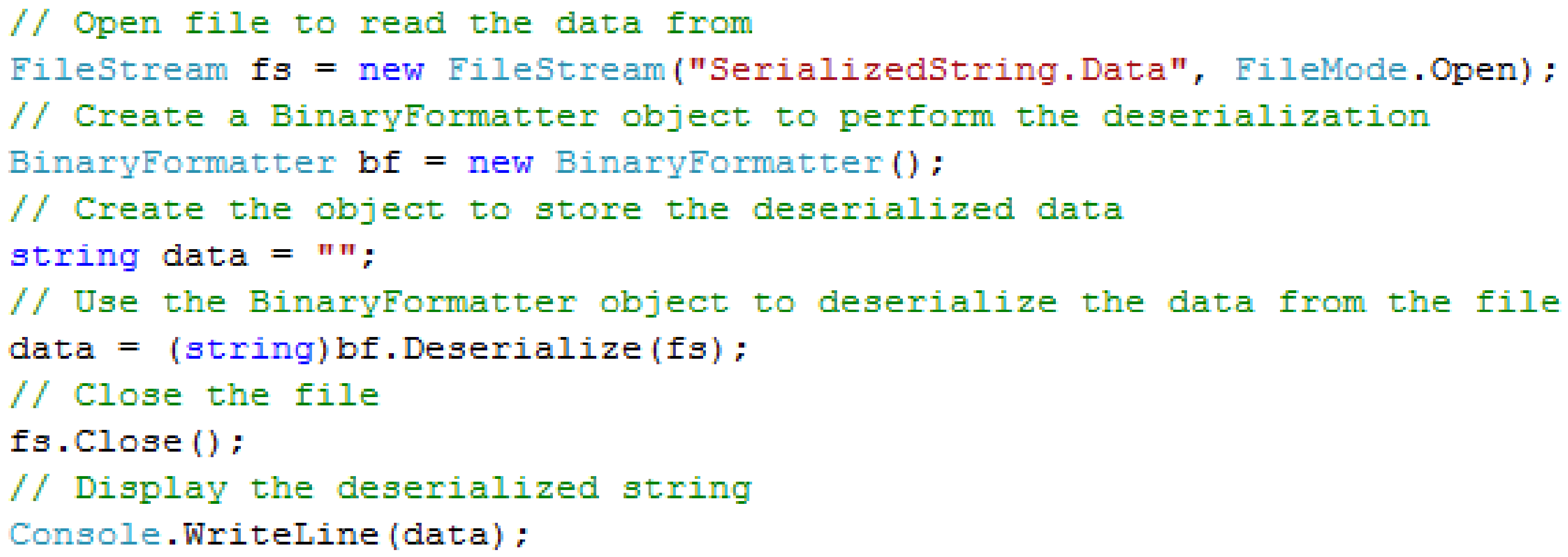
SERIALIZATION PROCEDURE

- Do not forget to import *System.IO, System.Runtime.Serialization, and System.Runtime.Serialization.Formatters.Binary*



```
// Create file to save the data to
FileStream fs = new FileStream("SerializedDate.Data", FileMode.Create);
// Create a BinaryFormatter object to perform the serialization
BinaryFormatter bf = new BinaryFormatter();
// Use the BinaryFormatter object to serialize the data to the file
bf.Serialize(fs, System.DateTime.Now);
// Close the file
fs.Close();
```

DESERIALIZATION PROCEDURE



```
// Open file to read the data from
FileStream fs = new FileStream("SerializedString.Data", FileMode.Open);
// Create a BinaryFormatter object to perform the deserialization
BinaryFormatter bf = new BinaryFormatter();
// Create the object to store the deserialized data
string data = "";
// Use the BinaryFormatter object to deserialize the data from the file
data = (string)bf.Deserialize(fs);
// Close the file
fs.Close();
// Display the deserialized string
Console.WriteLine(data);
```

Creating serializable classes

- You can serialize and deserialize custom classes by adding the Serializable attribute to the class.

```
[Serializable]
```

- You can disable serialization of some specific members of your class (such as temporary or calculated values) that might not need to be stored.

```
[NonSerialized]
```

- To enable your class to automatically initialize a nonserialized member, use the *IDeserializationCallback* interface, and then implement *IDeserializationCallback*.*OnDeserialization*.
- Each time your class is deserialized, the runtime will call this method after deserialization is complete.

Example

```
[Serializable]
class ShoppingCartItem : IDeserializationCallback
{
    public int productId;
    public decimal price;
    public int quantity;
    [NonSerialized]
    public decimal total;
    public ShoppingCartItem(int _productID, decimal _price, int _quantity)
    {
        productId = _productID;
        price = _price;
        quantity = _quantity;
        total = price * quantity;
    }
    void IDeserializationCallback.OnDeserialization(Object sender)
    {
        // After deserialization, calculate the total
        total = price * quantity;
    }
}
```

Providing version compatibility

- You might have version compatibility issues if you ever attempt to deserialize an object that has been serialized by an earlier version of your application.
- Specifically, if you add a member to a custom class and attempt to deserialize an object that lacks that member, the runtime will throw an exception.
- Apply the `[OptionalField]` attribute to newly added members that might cause compatibility problems.

```
[OptionalField] public bool taxable;
```

- During deserialization, if the member was not serialized, the runtime will leave the member's value as *null* rather than throwing an exception.

Choosing serialization formats

- **BinaryFormatter** **(**System.Runtime.Serialization.Formatters.Binary Namespace), this formatter is the most efficient way to serialize objects that will be read by only .NET Framework–based applications.

- **SoapFormatter** **(**System.Runtime.Serialization.Formatters.Soap Namespace), this XML-based formatter is the most reliable way to serialize objects that will be transmitted across a network or read by non–.NET Framework applications.

XML SERIALIZATION

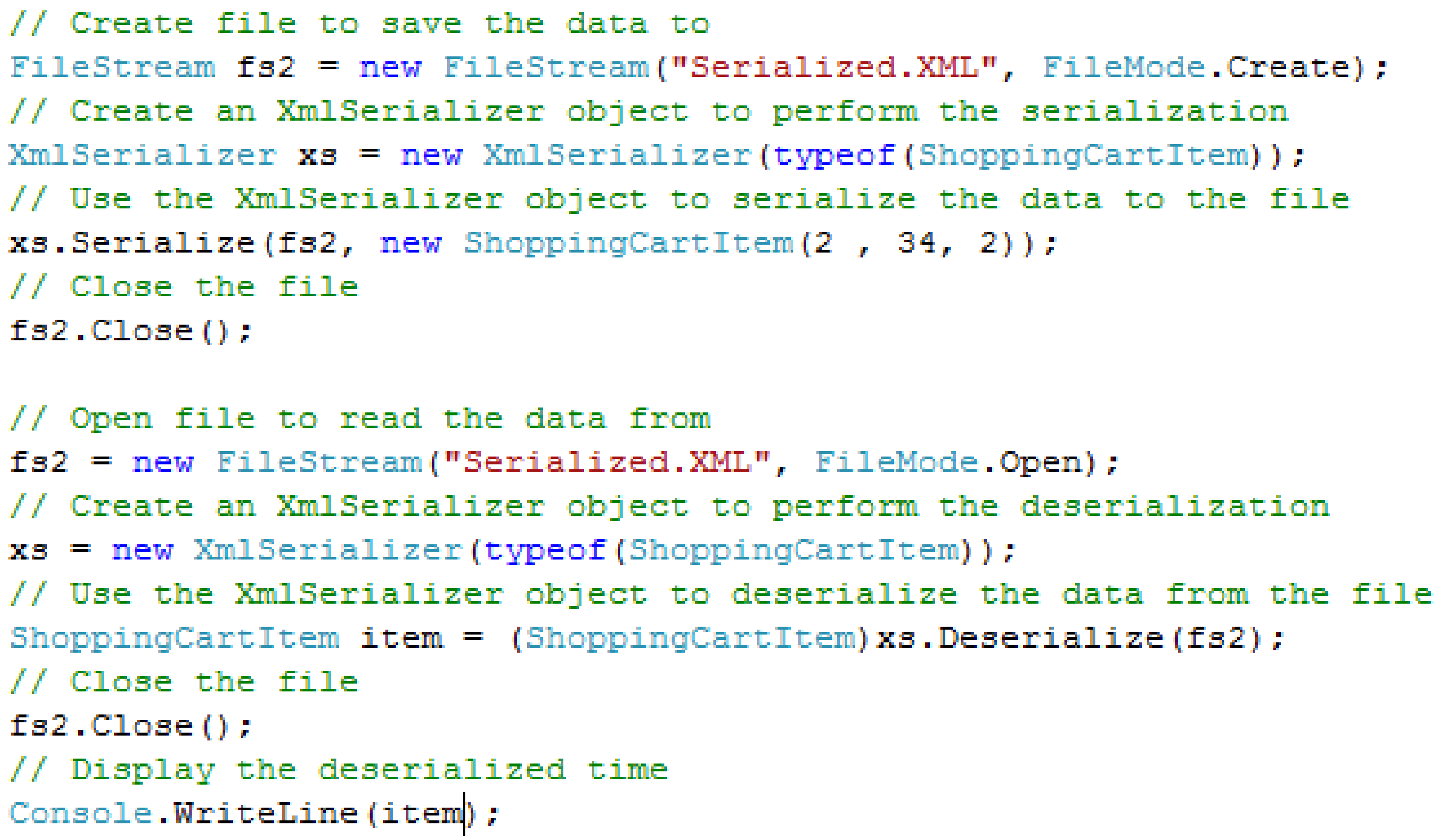
- With XML serialization, you can write almost any object to a text file for later retrieval with only a few lines of code.
- Similarly, you can use XML serialization to transmit objects between computers through Web services—even if the remote computer is not using the .NET Framework.
- Use XML serialization when you need to exchange an object with an application that might not be based on the .NET Framework, and you do not need to serialize any private members
- Advantages include:
  - Interoperability (XML is a standard)
  - It can be viewed and edited

STEPS

- Serialization
  - Create a stream, TextWriter, or XmlWriter object to hold the serialized output.
  - Create an XmlSerializer object (in the System.Xml.Serialization namespace) by passing it the type of object you plan to serialize.
  - Call the XmlSerializer.Serialize method to serialize the object and output the results to the stream.

- Deserialization
  - Create a stream, TextReader, or XmlReader object to read the serialized input.
  - Create an XmlSerializer object (in the System.Xml.Serialization namespace) by passing it the type of object you plan to deserialize.
  - Call the XmlSerializer.Deserialize method to deserialize the object, and cast it to the correct type.

IMPLEMENTATION



```
// Create file to save the data to
FileStream fs2 = new FileStream("Serialized.XML", FileMode.Create);
// Create an XmlSerializer object to perform the serialization
XmlSerializer xs = new XmlSerializer(typeof(ShoppingCartItem));
// Use the XmlSerializer object to serialize the data to the file
xs.Serialize(fs2, new ShoppingCartItem(2 , 34, 2));
// Close the file
fs2.Close();

// Open file to read the data from
fs2 = new FileStream("Serialized.XML", FileMode.Open);
// Create an XmlSerializer object to perform the deserialization
xs = new XmlSerializer(typeof(ShoppingCartItem));
// Use the XmlSerializer object to deserialize the data from the file
ShoppingCartItem item = (ShoppingCartItem)xs.Deserialize(fs2);
// Close the file
fs2.Close();
// Display the deserialized time
Console.WriteLine(item);
```

Requirements for a class

- Specify the class as public.
- Specify all members that must be serialized as public.
- Create a parameterless (no-arg) constructor.

- Unlike classes processed with standard serialization, classes do not have to have the ***Serializable*** attribute to be processed with XML serialization.
- If there are private or protected members, they will be skipped during serialization.

Controlling XML serialization

- *XMLAttribute*
- *XMLElement*
- *XMLIgnore*
- *XMLRoot*
- …

- You could use these to make a serialized class conform to specific XML requirements

EXAMPLE

- Change the ShoppingCartItem element name to CartItem.
- Make *productId* an attribute of *CartItem* rather than a separate element.
- Do not include the total in the serialized document.

```
[XmlRoot ("CartItem")]
public class ShoppingCartItem
{
[XmlAttribute] public Int32 productId;
public decimal price;
public Int32 quantity;
[XmlIgnore] public decimal total;
public ShoppingCartItem()
{
}
}
```