MC322 - Object Oriented Programming
Lesson 6.1
Objects and Portable Data: XML

Prof. Marcos M. Raimundo
Instituto de Computação - UNICAMP

Application Development and XML

- Portable languages are powerful tools, but they represent only half of the application development equation. Programs need to process data and convert it into information, which is critical for driving businesses. Information thus forms the other half of the portability equation.
- XML serves as a standard mechanism for defining and transporting data between potentially disparate systems. It enhances efficiency and security when used with object-oriented languages such as Java, VB, and C#.
- XML allows independent applications to share data efficiently and securely, providing a vital link between different systems.

Portable Data

Data Compatibility and E-Commerce

- The diversity of data storage formats poses a significant business challenge, especially when different companies using different systems want to interact.
- Example: Alpha Company uses an Oracle database for sales, while Beta Company uses SQL Server for purchasing. These systems are not directly compatible, which is a problem when these companies wish to conduct transactions over the Internet.
- Our solution involves creating an electronic purchase order system for Beta Company using SQL Server, designed to interact seamlessly with Alpha Company's Oracle-based sales system.
- This scenario underscores the broader necessity for companies to efficiently move information not only within their own systems but also across external interfaces like the Internet and local intranets to support varied electronic commerce needs.

XML Data Movement: Vertical and Horizontal Applications

- XML facilitates data movement in various ways, often described as vertical and horizontal.
- **Vertical Movement:** Data moves through multiple industry groups. For example, in finance, the Financial Products Markup Language (FpML) provides standard data definitions, helping to move information across industries. These are often termed as a vocabulary created using XML.
- **Horizontal Movement:** This type of application is industry-specific, such as those found in retail or transportation sectors. Sharing data efficiently is critical in all forms of electronic commerce.

XML Applications in Industries: RecipeML Example

- RecipeML (Recipe Markup Language) is an XML vocabulary specifically designed for industries related to food, such as hotels, restaurants, and publishers. It standardizes data exchange within these sectors.
- Using RecipeML allows for the standard and portable movement of data across different entities within the food industry.
- Other industries that utilize XML-based standards include legal, hospitality, accounting, retail, travel, finance, and education.
- **Concept of Portable Data:** While machine-level data may not be portable, XML helps achieve a higher-level of information portability across various programming environments like Java, VB, and C#. XML facilitates this essential information portability.

The Extensible Markup Language (XML)

XML: Extensible Markup Language

- XML (Extensible Markup Language) and HTML (Hypertext Markup Language) are both descendants of SGML (Standard Generalized Markup Language). SGML was first introduced in the 1970s and standardized in the 1980s.
- **HTML's Function:** Primarily developed to organize data via hyperlinks and present data in browsers, HTML focuses on data presentation rather than data definition and verification.
- **XML's Role:** In contrast to HTML, XML, defined in 1997 as a subset of SGML, is strict about its format and is designed to represent and verify data. It is not proprietary and is supported by the World Wide Web Consortium (W3C).
- XML's non-proprietary nature and its adoption by major IT industry leaders like Sun Microsystems, Microsoft, and IBM ensure its longevity and relevance in various fields including distributed computing and object persistence.

XML and Portable Information

- XML enhances "portable information" by working alongside object-oriented languages. It plays a crucial role in applications designed to interact with diverse data systems.
- **Case Study:** Alpha Company (a department store using an Oracle database) and Beta Company (a vacuum machine manufacturer using a SQL Server database) need to conduct transactions electronically over the Internet.
- Despite using different databases and potentially different data formats within these databases, XML enables these companies to share essential data effectively.
- **Key Objective:** The main goal is to facilitate business transactions such as sending and processing purchase orders without requiring a direct physical database connection.

Parsers

A parser is a program that reads a document and extracts specific information according to predefined rules. For instance, in a compiler, the parser checks each line of the program against grammar rules to correctly generate code. It ensures that commands like a print statement adhere to the correct syntax.

Connecting Systems via XML

- Alpha Company creates an XML specification to define the necessary information for transactions and stores this data in its SQL Server database.
- Object-oriented programming languages are employed to extract data from Alpha Company's database and generate an XML document adhering to agreed standards.
- This XML document is then transmitted over the Internet to Beta Company. Beta uses the same XML standard to parse the document and integrate the data into its Oracle database.
- **Data Flow Illustration:** Figure shows the data transfer process where data is first extracted by an application/parser from a SQL database, sent over a network, and then converted into an Oracle database format by another application/parser.

Sharing Data Between Two Companies

- We explore the practical application of our discussion through the collaboration between Alpha and Beta Companies. The goal is to create an XML document that facilitates a basic transaction between them.
- The document will include data specified in Table, which outlines the essential information to be transferred from one company to the other.

Object	Category	Field
supplier		
	name	
		<companyname>
	address	
		<street>
		<city>
		<state>
		<zip>
	product	
		type
		price
		count

Specification for Data to be Transferred.

Validating the Document with the Document Type Definition (DTD)

XML Document Structure and Transaction

- We will create and send an XML document from Beta Company to Alpha Company. This document encapsulates a transaction including the company name, address, and product details.
- The document structure is nested: at the highest level is the supplier object, within which are nested elements for the company name, address, and product information.
- Additional layers of information are nested within the address and product details to provide comprehensive data about each element.
- To organize and standardize this transaction data, we will first define a DTD (Document Type Definition) that will govern the structure of all transactions in this example.

```
<!– DTD for supplier document –>
<!ELEMENT supplier ( name, address)>
<!ELEMENT name ( companyname)>
<!ELEMENT companyname ( #PCDATA)>
<!ELEMENT address ( street+, city, state, zip)>
<!ELEMENT street ( #PCDATA)>
<!ELEMENT city ( #PCDATA)>
<!ELEMENT state ( #PCDATA)>
<!ELEMENT zip ( #PCDATA)>
```

The Data Definition Document for Validation.

Understanding the DTD Structure

- The DTD (Document Type Definition) specifies the structure of an XML document using tags similar to HTML. The document begins with a comment to explain the DTD's purpose:

```
<!-- DTD for supplier document -->
```

- XML comments function like comments in other programming languages—they help document the code to make it easier to understand. It's important not to overcrowd the document with comments to maintain readability.
- The structure of the XML document is defined by elements declared in the DTD. For example, the supplier element is defined as follows:

```
<!ELEMENT supplier (name, address, product)>
```

This line declares that the "supplier" element contains child elements: "name", "address", and "product".

Detailed DTD Structure and Editing

- The "supplier" element as defined in the DTD contains three sub-elements: "name", "address", and "product". Each element must be present in the XML document for it to be valid:

```
<!ELEMENT supplier (name, address, product)>
```

- The "name" element itself includes a further sub-element called "companyname", which is defined to hold actual data (parsed character data or PCDATA):

```
<!ELEMENT name (companyname)>
<!ELEMENT companyname (#PCDATA)>
```

- This setup completes the hierarchy of the element tree in the XML document. The DTD file, typically named "supplier.dtd", can be created using any text editor, such as Notepad. There are also specialized tools and environments available for creating and managing DTDs.

Integrating DTD into an XML Document

- Now that the DTD (Document Type Definition) has been created, the next step is to construct an XML document that adheres to this DTD. The document must include all the elements defined in the "supplier" DTD:

```
<!-- Example of how XML conforms to DTD -->
```

- The actual data for the XML document, as outlined in Table, should be inserted only into the terminal (end) elements, not into aggregate elements like "address" and "name".
- To create this XML document, you can use a simple text editor like Notepad, similar to how the DTD was written. There are also specialized tools designed for creating XML documents, which might be explored later.

Object	Category	Field	Value
supplier	name	<companyname>	
			The Beta Company
	address		
		<street>	12000 Ontario St
		<city>	Cleveland
		<state>	OH
		<zip>	24388
	product		
		type	Vacuum Cleaner
		price	50.00
		count	20

Adding the Values to the Table.

Linking XML Document to DTD and Understanding Tag Structure

- The XML document is explicitly linked to its DTD definition to ensure it adheres to the specified structure:

```
<!DOCTYPE supplier SYSTEM "supplier.dtd">
```

  This line in the XML associates it with the "supplier.dtd" DTD file, establishing rules for its structure.
- In the XML document, tags are nested similarly to how classes might be nested in programming. The structure must mimic the hierarchy defined by the DTD.
- Tags like address act as "abstract" containers without direct data, while "concrete" tags such as street contain actual data:

```
<address>
    <street>12000 Ontario St</street>
</address>
```

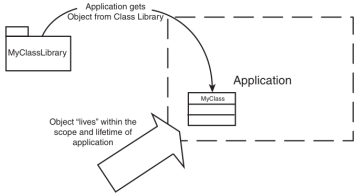
- Tools like XML Notepad enhance the development and inspection of XML documents, offering a user-friendly interface similar to that of Microsoft Notepad but optimized for XML editing.

Persistent Objects

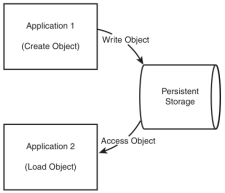
Persistent Objects Basics

- **Object Lifecycle:** Traditionally, an object such as an Employee with attributes (name, ss#, etc.) exists only within the duration of the application that instantiated it. When the application terminates, so does the object.
- **Persistence Concept:** To extend the life of an object beyond the application, it needs to be stored in a persistent storage medium. This action saves the state of the object, allowing it to be restored and used by other applications later.
- **Persistence Implementation:** For example, an Employee object is created in Application 1 and saved to a database. This makes it accessible to other applications:

  Application 2 can then instantiate this object and load its saved state, thereby achieving persistence.



Object life cycle (Application 1).



Object life cycle with persistence

Using XML in the Serialization Process

XML Serialization Model

- **Limitations of Proprietary Serialization:** Proprietary methods, while efficient and compact, lack portability across different platforms. XML offers a standard solution that enhances portability.
- **Creating a Universal XML Model:** By modeling our serialization example in XML, we enable access from various programming environments, including C#, VB .NET, and Java. This model facilitates interoperability.
- **XML vs. Java Serialization:** The key distinction lies in the output; the XML model generates a document describing the Person class's attributes and properties. This method introduces some complexity but provides a structured and encapsulated representation of the class.

XML Integration in C# Person Class

- **XML Annotation in C#:** The Person class in C# uses XML annotations to define its attributes, ensuring the class is prepared for XML serialization. Here are some examples of these annotations:
- **Enhanced Encapsulation:** These XML tags do not just serialize data; they also tighten the class's encapsulation. By defining XML roots, attributes, and elements within the code, each attribute is closely tied to its representation in XML, enhancing both security and data integrity.
- **Standardized Access Through Properties:** Unlike traditional methods where getters and setters could vary wildly in name and implementation, in the XML model, these are standardized as properties of the attributes. This approach ensures uniform access and manipulation of class data, adhering to a consistent protocol.

```
[XmlRoot("person")]
public class Person
[XmlAttribute("name")]
public String Name
[XmlElement("age")]
public int Age
```

Detailed XML Attribute Implementation in C#

- **Defining XML Attributes:** The definition of an XML attribute in a C# class is more involved than a simple attribute declaration. Here is how the ‘Name‘ attribute is defined for XML serialization:

```
[XmlAttribute("name")]
public String Name
{
    get { return this.strName; }
    set { if (value == null) return; this.strName = value; }
}
```

Detailed XML Attribute Implementation in C#

- **Property Mechanics:** The code snippet above uses a property with custom getter and setter methods:
  - The *getter* ensures that the value of ‘strName‘ is returned whenever the ‘Name‘ property is accessed.
  - The *setter* provides a safeguard against null values being assigned to ‘strName‘, thus maintaining the integrity of the data.
- **Comparison to Simple Declaration:** This approach contrasts sharply with a straightforward attribute declaration like:

```
public String Name;
```

The XML attribute definition adds layers of functionality and safety checks that are not present in a simple field declaration.

XML Serialization in C#

- **XML Serialization Code:** The following C# method serializes an array of ‘Person‘ objects into an XML file using the ‘XmlSerializer‘:

```
public void Serialize()
{
    Person[] myPeople = new Person[3];
    myPeople[0] = new Person("John Q. Public", 32, 95);
    myPeople[1] = new Person("Jacob M. Smith", 35, 67);
    myPeople[2] = new Person("Joe L. Jones", 65, 77);
    XmlSerializer mySerializer = new XmlSerializer(typeof(Person[]));
    TextWriter myWriter = new StreamWriter("person.xml");
    mySerializer.Serialize(myWriter, myPeople);
    myWriter.Close();
}
```

XML Serialization in C#

- **Output XML Format:** Unlike Java serialization that uses a proprietary format, the XML output is standardized and easily interpretable across different platforms:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfPerson xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <Person name="John Q. Public">
        <age>32</age>
    </Person>
    <Person name="Jacob M. Smith">
        <age>35</age>
    </Person>
    <Person name="Joe L. Jones">
        <age>65</age>
    </Person>
</ArrayOfPerson>
```

XML Deserialization in C#

- **Deserialization Code:** The following method is used to read the XML file and restore the 'Person' objects:

```
public void DeSerialize()
{
    Person[] myRestoredPeople;
    XmlSerializer mySerializer = new XmlSerializer(typeof(Person[]));
    TextReader myReader = new StreamReader("person.xml");
    myRestoredPeople = (Person[])mySerializer.Deserialize(myReader);
    Console.WriteLine("My People restored:");
    foreach (Person listPerson in myRestoredPeople)
    {
        Console.WriteLine(listPerson.Name + " is " +
                          listPerson.Age + " years old.");
    }
    Console.WriteLine("Press any key to continue...");
    Console.ReadKey();
}
```

XML Deserialization in C#

- **Cross-Platform Accessibility:** One significant advantage of XML serialization is its platform and language neutrality. The XML file generated can be accessed and used by any programming language or platform that supports XML, such as Java, VB .NET, or C#.
- **Universal Implementation:** Despite the example of using a proprietary Java serialization in this chapter, the XML method allows for a universal approach that could be adopted in Java as well. This underscores XML's flexibility and widespread application possibilities.

MC322 - Object Oriented Programming
Lesson 6.1
Objects and Portable Data: XML

Prof. Marcos M. Raimundo
Instituto de Computação - UNICAMP