Introduction to Objects and Abstract Data Types in Java

Objectives

- Understand and be able to write basic Extensible Markup Language (XML)
- Understand how to parse XML in Java
- Construct objects using data parsed from XML
- Read user input in Java and use it to direct the control flow of a program

XML and DOM

- **XML:** Extensible Markup Language is a flexible language that allows us to store data in a hierarchical fashion.
- **DOM:** The Document Object Model is, in part, a the hierarchical structure of objects that is defined by an XML document.

Basics of XML

- We can create pairs of tags to encapsulate data:
  - <name>Eluvetie</name>
  - <genre>Folk Metal</genre>
- We can group pairs of tags together with other tags:

```
<band>
    <name> Eluvetie </name>
    <genre> Folk Metal </genre>
</band>
```

Basics of XML

And we can keep growing these collections

```
<band>
    <name> Eluvetie </name>
    <genre> Folk Metal </genre>
    <songs>
        <songname> A Rose for Epona </songname>
        <songname> Omnos </songname>
        <songname> Lvgvs </songname>
    </songs
</band>
```

Basics of XML

And we can keep growing these collections

```
<bands>
    <band>
        <name> Eluvetie </name>
        <genre> Folk Metal </genre>
        <songs>
            <songname> A Rose for Epona </songname>
            <songname> Omnos </songname>
            <songname> Lvgvs </songname>
        </songs
    </band>
    <band>
        <name> Slice the Cake </name>
        <genre> Prog. Metal </genre>
        <songs>
            <songname> Westward Bound - Part 1</songname>
            <songname> Westward Bound - Part 2</songname>
        </songs
    </band>
</bands>
```

The lay of the land

1. Create an instance of the DocumentBuilderFactory object.
2. Use the DocumentBuilderFactor to create a DocumentBuilder. We will be using this object to parse our XML.
3. Open the file by creating a new File object.
4. Use our document builder to parse the XML in the File object.

DocumentBuilderFactory

```
DocumentBuilderFactors dbf = DocumentBuilderFactory.newInstances()
```

- A factory is a class that is responsible for making instances of other classes.
- Factories are a common design pattern in Object Oriented Programming.
- DocumentBuilderFactories produces DocumentBuilder

DocumentBuilder

```
DocumentBuilderFactors dbf = DocumentBuilderFactory.newInstances()

DocumentBuilder db = null;
try{
    db = dbf.newDocumentBuilder();
} catch(ParserConfigurationException e){
    System.out.println("Failed to configure document builder object");
}
```

- We use our dbf to instantiate a new DocumentBuilder via the newDocumentBuilder() method.
- Calling this function can produce an ParserConfigurationException which Java requires us to catch.

File

```
DocumentBuilderFactors dbf = DocumentBuilderFactory.newInstances()

DocumentBuilder db = null;
try{
    db = dbf.newDocumentBuilder();
} catch(ParserConfigurationException e){
    System.out.println("Failed to configure document builder object");
}

File f = new File(filePath)
```

- Now that we have our XML parser which will build our DOM we need to open the file.
- We provide the `File` class's constructor the path to our XML file.

Document

```
DocumentBuilderFactors dbf = DocumentBuilderFactory.newInstances()

DocumentBuilder db = null;
try{
    db = dbf.newDocumentBuilder();
} catch(ParserConfigurationException e){
    System.out.println("Failed to configure document builder object");
}

File f = new File(filePath)

Document doc = null;
try{
    doc = db.parse(f);
} catch(SAXException | IOEXception e){
    System.out.println("Failed to parse document.")
}
```

- Finally, we create our Document (i.e., the DOM) by giving the document builder our opened file.
- There are two exceptions that can occur. Either the XML may not be valid or the file may not be readable. Java requires that we catch both of these.

Nodes

```
<bands>
  <band>
    <name> Eluvetie </name>
    <genre> Folk Metal </genre>
  </band>
  <band>
    <name> Slice the Cake
          </name>
    <genre> Prog. Metal
          </genre>
  </band>
</bands>
```

- The document produced ends up being a collection of Nodes
  - A "Node" is all the XML between two tags.
- We can get a list of nodes (NodeList) via:
  - NodeList nodes = doc.getElementsByTagName("tagName").
- We can get an individual node from that node list via:
  - Node node = nodes.item(i).
- We can get the text content from the node via:
  - String nodeText = node.getTextContent().

Worksheet: XML Parsing

Off to the worksheet to see how this works in practice!

Why Bother With XML?

```
<bands>
  <band>
    <name> Eluvetie </name>
    <genre> Folk Metal </genre>
  </band>
  <band>
    <name> Slice the Cake </name>
    <genre> Prog. Metal </genre>
  </band>
</bands>
```

- We often want to separate the program from the data it is using.
  - Imagine we wanted to make a music player app.
  - App and songs the app plays are separate.
- XML provides a structured and (relatively) easy to parse method of doing this.
- There are other methods of storing data:
  - Comma Separated Values (csv) files
  - Databases (e.g., SQL, SQLite, MongoDB)

We can read data. Now what?

Class 1: A class to store data/methods

```
class Bands{
    String name;
    String genre;
    Bands(String name, String genre){
        this.name = name;
        this.genre = genre;
    }
}
```

Class 2: A class to read data from the file

```
class ParseBands{
    public static void main(String[] args){
        // Parse all bands into NodeList called bands
        List<Bands> bandList = new ArrayList<>();
        for(int i = 0; i < bands.getLength(); i++){
            Element elem = (Element) bands.item(i);

            String name = elem.getElementsByTagName("name").item(0).getTextContent();
            String genre = elem.getElementsByTagName("genre").item(0).getTextContent();

            bandList.add(new Bands(name, genre));
        }
    }
}
```

Worksheet: XML Parsing and Objects

Off to the worksheet to see how this works in practice!

Getting Keyboard Input from the User

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter value: ");
```

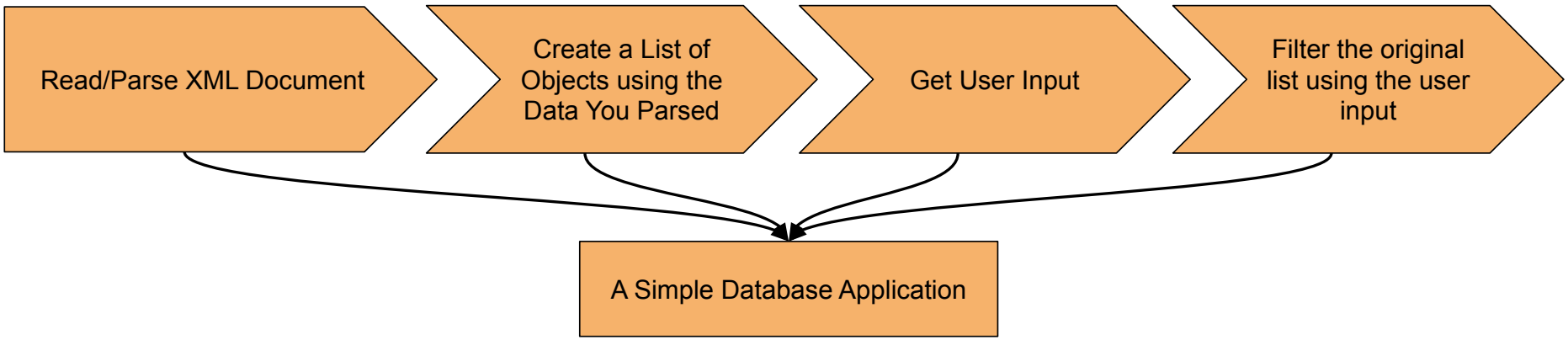
- System.in is the standard input stream for your computer (a.k.a., keyboard).
  - Others include System.err and System.out.
- Scanner "watches" that stream and waits for: 1) data and 2) a newline in order to determine that input has ended.
- Depending on what we want we can use the following functions to get a value from the Scanner:
  - Integer val = sc.nextInt()
  - Float val = sc.nextFloat()
  - String val = sc.nextLine()
  - Double val = sc.nextDouble()

Extending our Filtering Pattern

```
class Animal implements Comparable<Animal>{
    //...
    @Override
    public int compareTo(Animal other){
        return name.compareTo(other.name);
    }
}
```

- If we are going to build and potentially sort information we need some method of comparing two classes
- Our class must implement the Comparable interface
- We must also override the compareTo method of that interface.
  - Generally involves calling the compareTo method of some attribute of the class.

XML Parsing + Making Objects + Filtering + User Input



Off to the worksheet to practice this!