New Assignment (#7)

Combines:

• ItemizedOverlay
• Location

Due: 10/26/2011
Web Service Communication

To obtain information from the City of Winston-Salem about *open work orders*, we will request information from their exposed *web-service*.
Web Service Communication

To obtain information from the City of Winston-Salem about open work orders, we will request information from their exposed web-service.
XML Parsing: Key Ideas

• Tags:
  Start: <tag>
  End: </tag>

• Content:
  <tag>content</tag>

• Nesting:
  <tag1>
    <tag2>content</tag2>
  </tag1>

  Close most recently opened tag!

• Attributes:
  <tag label="value">

• Auto-closing tag:
  <tag label="value"/>

  Equivalent to no-content
  <tag label="value"></tag>
XML Parsing: Constrained

• For City of Winston-Salem app, XML returned will be constrained:
  – No auto-closing tags
  – No attribute value pairs
  – Properly nested
  – Assume all entities completely described (same tags provided for each entity)
  – No escape characters (&lt; for <)
  – OK to have empty content
Work Orders XML

An example of actual work order data we will receive

Query: first two work orders within ½ mile of WFU Welcome Center
A Simpler XML Example

<meals>
  <meal>
    <id>1</id>
    <main>Hamburger</main>
    <side>French fries</side>
    <drink>Coke</drink>
    <price>3.99</price>
  </meal>
  <meal>
    <id>2</id>
    <main>Nuggets</main>
    <side>Onion rings</side>
    <drink>Sweet tea</drink>
    <price>4.39</price>
  </meal>
</meals>
XML ↔ Objects

• We want to be able to map the XML back to objects/classes
• What would be an appropriate class/object representation for a meal?

```xml
<meal>
  <id>2</id>
  <main>Nuggets</main>
  <side>Onion rings</side>
  <drink>Sweet tea</drink>
  <price>4.39</price>
</meal>
```

```java
class Meal {
  int id;
  String main;
  String side;
  String drink;
  double price;
}
```
The `<meals>` tags basically support the notion of a List of Meal objects:

```
ArrayList<Meal> orders;
```
Parser

• A parser, technically, is a piece of software that:
  – Analyzes the tokens of a given textual document
  – Ensures the tokens are in a structure that could have been generated from a grammar

• A compiler employs a parser to ensure that programs you write follow the appropriate Java grammar rules

• We will use parsing in an informal sense:
  – Breaking up a text document into tokens and working with those tokens
Handcrafting an XML Parser

• Assume know what we are expecting:
  – Parser is aware what a meals list and a meal are composed of in XML
  – Parser is aware of a meals list arriving in String format
  – Parser is not aware of number of items specified in list
  – Parser is not aware of order of description of meals
Meals Parser: General Approach

- Create empty list of Meal objects
- Ignore `<meals>`</meals> tags
- For each `<meal>...`</meal> combination
  - Create a blank Meal object
  - Find each tag/content group
    - Discover tag
    - Discover content
    - Set value in Meal object based on tag and content
  - Add completed Meal to list
Meal Class Definition

```java
public class Meal {
    int id;
    String main;
    String side;
    String drink;
    double price;

    public Meal(int idArg, String mainArg, String sideArg, String drinkArg, double priceArg) {
        id = idArg;
        main = mainArg;
        side = sideArg;
        drink = drinkArg;
        price = priceArg;
    }

    public Meal() {
        id = -1;
        main = "";
        side = "";
        drink = "";
        price = -1.00;
    }

    public int getId() { return id; }
    public void setId(int idArg) { id = idArg; }
    public String getMain() { return main; }
    public void setMain(String mainArg) { main = mainArg; }
    public String getSide() { return side; }
    public void setSide(String sideArg) { side = sideArg; }
    public String getDrink() { return drink; }
    public void setDrink(String drinkArg) { drink = drinkArg; }
    public double getPrice() { return price; }
    public void setPrice(double priceArg) { price = priceArg; }

    public String toString() {
        return "Meal #" + id + ": Main=" + main + " Side=" + side + " Drink=" + drink + " Price=" + price;
    }
}
```
StringTokenizer

• StringTokenizer is a class in Java that supports breaking a string apart

• Four key methods:
  – Constructor: `StringTokenizer(String stringToBreak, String symbolsToBreakOn)`
    • Takes the string you want to break apart into tokens, and the symbol(s) that mark where to break the string
  – `int countTokens()`
    • Returns the number of tokens that resulted from breaking the string apart
  – `String nextToken()`
    • Returns (consumes) the next token after the last one you have consumed
  – `boolean hasMoreTokens()`
    • Returns true if there are more tokens left in the string that have not been consumed
StringTokenizer Example

```java
import java.io.*;
import java.util.*;

public class STExample
{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String typedInLine = scanner.nextLine();

        StringTokenizer tokenizer = new StringTokenizer(typedInLine, " ");
        System.out.println("Number of tokens in string when breaking on space: " + tokenizer.countTokens());

        while (tokenizer.hasMoreTokens()) {
            System.out.println(tokenizer.nextToken());
        }

        scanner.close();
    }
}
```
StringTokenizer

- When parsing XML, if we use “<>” as our symbols to break on, we can get a list of separated tags and content.

```java
import java.io.*;
import java.util.*;

public class XMLSTExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String typedInLine = scanner.nextLine();

        StringTokenizer tokenizer = new StringTokenizer(typedInLine, "<>");
        System.out.println("Number of tokens in string when breaking on <>: " + tokenizer.countTokens());

        while (tokenizer.hasMoreTokens()) {
            System.out.println(tokenizer.nextToken());
        }
    }
}
```

```
meat
    <meal><main>burger</main><side>fries</side><drink>water</drink></meal>
    <meal><main>water</main><side></side><drink></drink></meal>
    <meal><main>id</main><side>1</side><drink></drink></meal>
    <meal><main>price</main><side>4.50</side><drink></drink></meal>
```

```
Number of tokens in string when breaking on <>: 17
```
Handcrafted XML Parser

• See example in Eclipse/on website