JSON parsing in Java using Jackson parser

Parsing JSON files in Java can be accomplished quite well using the Jackson Java JSON parser (http://jackson.codehaus.org). The main thing to keep in mind is that you need to define Java classes that match your data. Then, you can use the data binding available with the ObjectMapper class to read the JSON data directly into the Java class objects that you define.

As an example of how this is done, consider the following JSON file, "projects.json":

```
{
  "projects":
    [
      {
        "name":"Python",
        "sponsor":"Python Software Foundation",
        "url":"http://www.python.org",
        "description":
          [
            "Python is a general purpose programming language ",
            "that is used in a number of projects including ",
            "Zope, Plone, Django, Pyjamas and the Google Search ",
            "Engine. "
          ]
      },
      {
        "name":"Pyjamas",
        "sponsor":"Luke Kenneth Carson Leighton",
        "url":"http://pyjs.org",
        "description":
          [
            "Pyjamas is a Python port of the Google Web ",
            "Toolkit.  It is simpler to use than the ",
            "Google Web Toolkit, but does not have all the ",
            "latest features. "
          ]
      }
    ]
}
```

This JSON file consists of a key:value pair. The "projects" key has a value that is an array (or ArrayList). Each element in the array is an associative array with information on an Open Source project.

Modeling data with Java classes

The first step is to recognize what kind of Java classes we need to use to model this JSON data. In this case, it is easier to work your way from the inside and move out. If you look inside the information for an Open Source project, one of the objects that can be seen is the "description". We can use an ArrayList<String> for the description. The other keys point to Strings. Let's define a Project class that can store the "name", "sponsor", "url", and "description".

```
1 import java.util.ArrayList;
```

```
public class Project
{
   private String name;
   private String sponsor;
   private String url;
   private ArrayList<String> description;

   public ArrayList<String> getDescription()
   {
      return description;
   }
   public void setDescription(ArrayList<String> desc)
   {
      description = desc;
   }

   public void setName(String s)
   {
      name = s.trim();
   }
   public void setSponsor(String s)
   {
      sponsor = s.trim();
   }
   public void setUrl(String s)
   {
      url = s.trim();
   }
   public String getName()
   {
      return name;
   }
   public String getSponsor()
   {
      return sponsor;
   }
   public String getUrl()
   {
      return url;
   }
   public String toString()
   {
      String temp = "    {\n";
      temp += "      \"name\":\"" + name + "\",\n";
      temp += "      \"sponsor\":\"" + sponsor + "\",\n";
      temp += "      \"url\":\"" + url + "\",\n";
      temp += "      \"description\":\n";
      temp += "        [\n";
      for (int i = 0; i < description.size(); i++) {
         temp += "          \"" +
            description.get(i) + "\"";
         if (i < description.size() - 1) {
            temp += ",";
         }
         temp += "\n";
      }
      temp += "        ]\n";
      temp += "    }\n";
      return temp;
   }
```

```
62 }
```

Just as would be the case for a Java bean, no constructor is needed.

Lines 4-7 define the instance variables needed to store the properties of a project. Note that the description variable is an ArrayList<String>.

Lines 9-41 define simple accessor and mutator methods. These methods simply get and set the instance variables.

Lines 42-61 define the `toString()` method. This method is just used to create a String representation for a Project object that displays all these properties with a format that is indented to display the structure.

Moving outward, the outermost structure is a key:value pair. In other words, we have: `"projects":[ ]`

This is an associative array that can be modeled with a Java HashMap. So, the outermost class will be a class of type: `HashMap<String, ArrayList<Project>>`.

Here is the source code for "Projects.java". This is a kind of container class for Project Objects:

```
1  import java.util.HashMap;
2  import java.util.ArrayList;
3  public class Projects extends HashMap<String,ArrayList<Project>>
4  {
5  }
```

The Projects class definition is very simple. All you need to do is make this class inherit from `HashMap<String, ArrayList<Project>>`. No constructor or any other methods are required.

Using the Jackson JSON parser

If you are going to use the data binding processes of the Jackson JSON parser, the actual main program is very simple. Here is an example of a program that will process "projects.json", and write out the descriptions for each project.

```
1  import org.codehaus.jackson.map.ObjectMapper;
2  import java.io.*;
3  import java.util.ArrayList;
4  class ReadProjects
5  {
6     public static void main(String[] args) throws IOException
7     {
8        ObjectMapper mapper = new ObjectMapper();
9        Projects projs =
10          mapper.readValue(new File("projects.json"),Projects.class);
11       ArrayList<Project> projects = projs.get("projects");
12       for (Project p : projects) {
13          ArrayList<String> description = p.getDescription();
14          for (String s : description) {
15             System.out.println(s);
```

```
16              }
17          }
18      }
19  }
```

On line 1, we import the ObjectMapper class. This is the class that does all the work in parsing the JSON file and storing the parsed data inside the Java objects.

On line 6, we have the `main()` method declaration. Note that we throw IOExceptions so that we don't need to use `try-catch` blocks.

On line 8, we construct the ObjectMapper object. On line 9 we use that object's `readValue()` method to read the contents of "projects.json" into a Projects object. Line 11 uses the *projects* key to get the `ArrayList<Project>` and assigns this to the variable *projects*. Lines 12-17 define a simple nested for loop to print out the description portion of each project. This would produce the following output:

```
Python is a general purpose programming language
that is used in a number of projects including
Zope, Plone, Django, Pyjamas and the Google Search
Engine.
Pyjamas is a Python port of the Google Web
Toolkit.  It is simpler to use than the
Google Web Toolkit, but does not have all the
latest features.
```

Although this is a simple example, hopefully you can see that once we have read the JSON data into the Projects class object, we can do any manipulations we want with that data from within the Java program.