Informatics Large Practical: JSON Parsing

Stephen Gilmore and Paul Jackson

School of Informatics

Friday, 9th October 2020

JSON

- JavaScript Object Notation (JSON) is a data interchange format, used to exchange data between programs.
- Objects are converted to strings for transmission or storage purposes, then converted back later.
  - Converting an object to a string is known as serialisation or marshalling.
  - Converting a string to an object is known as deserialisation or unmarshalling or parsing.
- JSON can be used for the same purposes as XML, but it is generally thought to be easier for humans to read than XML.

Example: JSON and Java objects of the Student class

JSON

```
{
    "name": "Susan Smith",
    "matric": "s2012345",
    "year": 1
}
```

jsonString

JAVA

```
public class Student {
    String name;
    String matric;
    int year;
}
```

Example: JSON and Java objects of the Student class

JSON

```
{
    "name": "Susan Smith",
    "matric": "s2012345",
    "year": 1
}
```

jsonString

JAVA

```
public class Student {
    String name;
    String matric;
    int year;
}
```

- We can even set the visibility of these fields to be **private** and add getter and setter methods. This will not have an impact on deserialisation.

Maven: Adding a dependency in pom.xml

XML

```xml
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.6</version>
</dependency>
```

- If your project doesn't already depend on Google's Gson parser for JSON then you will need to add it as a dependency.
- Now we can **import** com.google.gson.Gson into our code.

Deserialising a JSON record to a Java object using its class

JAVA

```
// Use the "fromJson(String, Class)" method
var student = new Gson().fromJson(jsonString, Student.class);
```

- We make a new Gson parser, and then call the fromJson method, passing in our JSON string and the Student class.
- Now we can access student.name, student.matric, and student.year.
- Things become a little more complicated if we want to parse a *list* of students.

Example: A list of JSON objects of the Student class

JSON

```
[ {
    "name": "Susan Smith",
    "matric": "s2012345",
    "year": 1
  },
  {
    "name": "Mary Black",
    "matric": "s1934567",
    "year": 2
  },
  {
    "name": "John White",
    "matric": "s1867890",
    "year": 3
  } ]
```

JAVA

```
public class Student {
    String name;
    String matric;
    int year;
}
```

$\longleftarrow$ $\boxed{\text{jsonListString}}$

Deserialising a JSON list to a Java object using its type

- We would like to deserialise this list into an object of type ArrayList⟨Student⟩ but that is an instantiation of the ArrayList class, not a class in its own right.
- We can't get a class associated with ArrayList⟨Student⟩ but we can get its Type using Java's Reflection API which is used to examine or modify methods, classes, or interfaces at runtime.
- We **import** two reflection classes, java.lang.reflect.Type and com.google.gson.reflect.TypeToken.

Deserialising a JSON list to a Java object using its type

JAVA

```
Type listType =
    new TypeToken<ArrayList<Student>>() {}.getType();
// Use the "fromJson(String, Type)" method
ArrayList<Student> studentList =
    new Gson().fromJson(jsonListString, listType);
```

- We create an anonymous inner class and then make an instance of it to be able to apply the getType method.
- **Note:** local variable type inference cannot infer the type of studentList due to the use of reflection.

Example: JSON and Java objects of the StudentDetails class

JSON

```
{
   "name": "Joe Green",
   "matric": "s2056789",
   "year": 1,

   "dateOfBirth": {
      "day": 31,
      "month": 8,
      "year": 2001
   }

}
```

jsonDetailsString

JAVA

```
public class StudentDetails {
    String name;
    String matric;
    int year;

    Date dateOfBirth;
    public static class Date {
        int day;
        int month;
        int year;
    }
}
```

Deserialising a JSON record to a Java object using its class

JAVA

```
// Use the "fromJson(String, Class)" method
var details =
    new Gson().fromJson(jsonDetailsString, StudentDetails.class);
```

- We make a new Gson parser as before, and then call the fromJson method, passing in our JSON string and the StudentDetails class.
- Now we can access details.name, details.matric, details.year, details.dateOfBirth.day, details.dateOfBirth.month and details.dateOfBirth.year.

Summary

- We have seen how to use the Gson parser to deserialise
  - simple JSON records,
  - lists of JSON records, and
  - complex JSON records.

Thank you for listening.