Object Persistence

Java Persistence API (JPA)

COMP3607
Object Oriented Programming II

Week 11

Ways to store

There are many ways to make data persist in Java, including (to name a few):

- JDBC (Java Database Connectivity)
- Object Serialization
- File Input/Output
- JCA (Java EE Connector Architecture)
- Object databases
- XML databases.

JDBC

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database.

It provides methods to query and update data in a database, and is oriented towards relational databases.

A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.

Setting up a connection

When a Java application needs a database connection, one of the DriverManager.getConnection( ) methods is used to create a JDBC connection.

The URL used is dependent upon the particular database and JDBC driver.

```
try (Connection conn = DriverManager.getConnection(
     "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
     "myLogin",
     "myPassword")) {
     /* you use the connection here */
}
```

https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

Query Execution

Data is retrieved from the database using a database query mechanism.

The example below shows creating a statement and executing a query.

```
try (Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM MyTable")
) {
    while (rs.next()) {
        int numColumns = rs.getMetaData().getColumnCount();
        for (int i = 1; i <= numColumns; i++) {
            // Column numbers start at 1.
            // Also there are many methods on the result set to return
            //  the column as a particular type. Refer to the Sun documentation
            //  for the list of valid conversions.
            System.out.println( "COLUMN " + i + " = " + rs.getObject(i));
        }
    }
}
```

Serialization

Serialization (or serialisation) is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later (possibly in a different computer environment.

Uses of Serialization

- Transferring data through the wires (messaging).
- Storing data (in databases, on hard disk drives).
- Remote procedure calls, e.g., as in SOAP.
- Distributing objects, especially in component-based software engineering such as COM, CORBA, etc.
- Detecting changes in time-varying data.

Formats

Serializing the data structure in an architecture-independent format means preventing the problems of byte ordering, memory layout, or simply different ways of representing data structures in different programming languages.

- Python
- JSON (JavaScript Object Notation)
- XML (Extensible Markup Language)
- YAML (YAML Ain't Markup Language)

https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats

Example Formats

JSON

```json
{
  "first name": "John",
  "last name": "Smith",
  "age": 25,
  "address": {
    "street address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal code": "10021"
  },
  "phone numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ],
  "sex": {
    "type": "male"
  }
}
```

YAML

```yaml
first name: John
last name: Smith
age: 25
address:
  street address: 21 2nd Street
  city: New York
  state: NY
  postal code: '10021'
phone numbers:
  - type: home
    number: 212 555-1234
  - type: fax
    number: 646 555-4567
sex:
  type: male
```

XML

```xml
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" />
  <phoneNumbers>
    <phoneNumber type="home" number="212 555-1234"/>
    <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
  <sex type="male"/>
</person>
```

Example Formats

```yaml
---
a: 123                      # an integer
b: "123"                    # a string, disambiguated by quotes
c: 123.0                    # a float
d: !!float 123              # also a float via explicit data type prefixed by (!!)
e: !!str 123                # a string, disambiguated by explicit type
f: !!str Yes                # a string via explicit type
g: Yes                      # a boolean True (yaml1.1), string "Yes" (yaml1.2)
h: Yes we have No bananas   # a string, "Yes" and "No" disambiguated by context.
```

YAML

Jackson

Jackson is a very popular and efficient java based library to serialize or map java objects to JSON and vice versa.

Jackson

- Easy to use. - jackson API provides a high level facade to simplify commonly used use cases.
- No need to create mapping. - jackson API provides default mapping for most of the objects to be serialized.
- Performance. - jackson is fast, has a low memory footprint and is suitable for large object graphs or systems.
- Clean JSON. - jackson creates a clean and compact JSON results which is easy to read.
- No Dependency. - jackson library does not require any other library apart from jdk.
- Open Source - jackson library is open source and is free to use

Features

- Easy to use. - jackson API provides a high level facade to simplify commonly used use cases.
- No need to create mapping. - jackson API provides default mapping for most of the objects to be serialized.
- Performance. - jackson is fast, has a low memory footprint and is suitable for large object graphs or systems.
- Clean JSON. - jackson creates a clean and compact JSON results which is easy to read.
- No Dependency. - jackson library does not require any other library apart from jdk.
- Open Source - jackson library is open source and is free to use

Steps

1. Create ObjectMapper object.
2. DeSerialize JSON to Object.
3. Serialize Object to JSON.

Object Mapper

ObjectMapper is the main actor class of Jackson library. It provides functionality for reading and writing JSON, either to and from basic POJOs (Plain Old Java Objects), or to and from a general-purpose JSON Tree Model (JsonNode), as well as related functionality for performing conversions.

It is also highly customizable to work both with different styles of JSON content, and to support more advanced Object concepts such as polymorphism and Object identity. ObjectMapper also acts as a factory for more advanced ObjectReader and ObjectWriter classes.

Code Example

```
public class ExportAsJSON {

   public void writeJSON(Student student) throws JsonGenerationException,
                                                 JsonMappingException, IOException{
      ObjectMapper mapper = new ObjectMapper();
      mapper.writeValue(new File("student.json"), student);
   }


   public Student readJSON() throws JsonParseException, JsonMappingException, IOException{
      ObjectMapper mapper = new ObjectMapper();
      Student student = mapper.readValue(new File("student.json"), Student.class);
      return student;
   }


}
```

JAXB Marshalling

The JAXB Marshaller interface is responsible for governing the process of serializing Java content trees i.e. Java objects to XML data. This marshalling to XML can be done to variety of output targets:

- XML
- DOM
- SAX

Steps

1. Create POJO or bind the schema and generate the classes
2. Create the JAXBContext object
3. Create the Marshaller objects
4. Create the content tree by using set methods
5. Call the marshal method

JAXB Marshalling

```
*/
public class ExportAsXML {
    public void writeXML() throws Exception{
        JAXBContext contextObj = JAXBContext.newInstance(Student.class);

        Marshaller marshallerObj = contextObj.createMarshaller();
        marshallerObj.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

        Student s1=new Student();
        s1.setName("Sal");

        marshallerObj.marshal(s1, new FileOutputStream("src/files/student.xml"));
    }

    public void readXML() throws Exception{
        try {

        File file = new File("src/files/student.xml");
        JAXBContext jaxbContext = JAXBContext.newInstance(Student.class);

        Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
        Student s= (Student) jaxbUnmarshaller.unmarshal(file);

        System.out.println("From XML: " + s.getName()+" "+s.getAge());

      } catch (JAXBException e) {
        e.printStackTrace();
      }
    }

}
```

References

- https://en.wikibooks.org/wiki/Java_Persistence/What_is_Java_persistence%3F
- https://www.tutorialspoint.com/jackson/jackson_object_serialization.htm
- https://www.tutorialspoint.com/jackson/index.htm
- https://www.javatpoint.com/jaxb-marshalling-example
- https://howtodoinjava.com/jaxb/jaxb-annotations/