PERSISTENT DATA

Sid Chi-Kin Chau

[Lecture 9]

Goals of This Lecture

- What is Persistent Data? And How?
  - Bespoke
  - Serialization
  - XML
  - JSON
- Compare Pros and Cons

What is Persistent Data?

- A critical task for applications is to save/retrieve data
  - Permanent data (storage of data from working memory)
    - It can be updated, but not as frequent as transient/volatile data
    - It is stored in database/SSD/harddisk/magnetic tape
- Why do we want permanent data?
  - Disadvantages of holding volatile data
    - To be used and reused (save and load), and fault tolerant
    - To be checked and validated for authentication
- How can we store data persistently?
  - The choice of the persistence method is part of the design of an application
    - Files (JSON, XML, images, …)
    - Databases

Uses of Data and Storage

Types	Use cases	Formats
Text files (unstructured data)	Word Processing	Raw text (ASCII, UTF-8) proprietary word processing formats .doc (generally unstructured)
Structured text files	Spreadsheet, sensor data, simple structured data	csv, tsv, bespoken, XML, JSON
Graphics	Images	png, jpeg (lossy), gif, bmp
Audio/Movie	Lecture recordings, music	mp3, mp4 (lossy)
Data compression	Large file storage	zip, tar, rar, …

Which is the Best Data Format

- Use case
  - What does your application do?
  - What kind of data you have?
  - Is there any restriction to meet?
    - Software licenses
    - Storage limitation
    - Rapid access to data
    - Rapid development

Aspects to Consider

- Programming Agility
  - Easy to develop (no overhead) and code
- Extensibility
  - Can data be easily extended? (e.g., add new fields, attributes, …)
  - Is it easy to add new fields in a CSV file?
  - Is it easy to add new attributes in a database?
- Portability
  - Will other applications access the data?
  - Will it run on other hardware?

Aspects to Consider

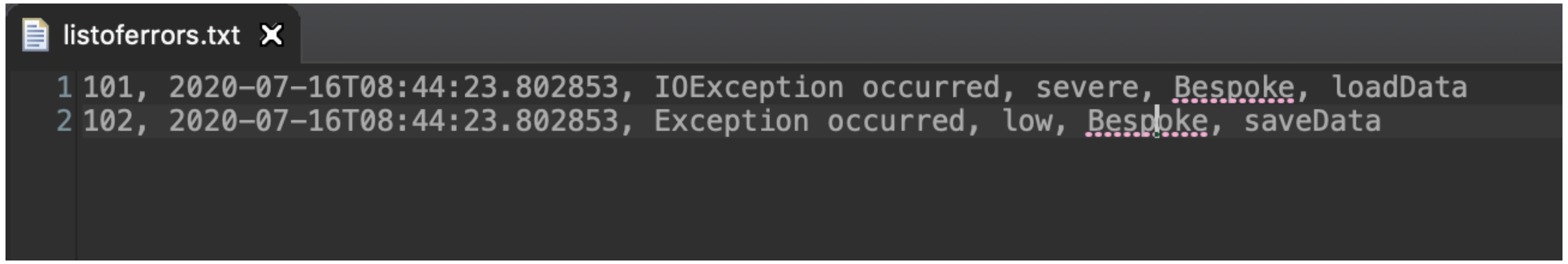
- Robustness
  - Bespoke vs XML vs JSON
    - Well-designed and structured format
    - Use of schema (how verify if your data is correctly formatted?)
    - Lack of schema and interoperability problems
- Size vs Completeness
  - Lossy vs Lossless
  - Audio/Image vs financial data/scientific data
- Internationalization
  - ASCII vs UTF-8
  - Who will use the data (audience)?

Bespoke and Serialization

- Bespoke data files
  - Define your own persistent data format
    - Write your own data formatting and checking methods
    - Not often used in industry
    - Not robust and may incur extra bugs
- Serialization
  - Directly storing binary class data (and even whole executable class)
  - Serialization presents technical issues
    - Programming language dependent and platform dependent (big- or little-endian)
    - Loss of object references
    - Security issues
  - Deserialization: revert persistent data to a copy of class object

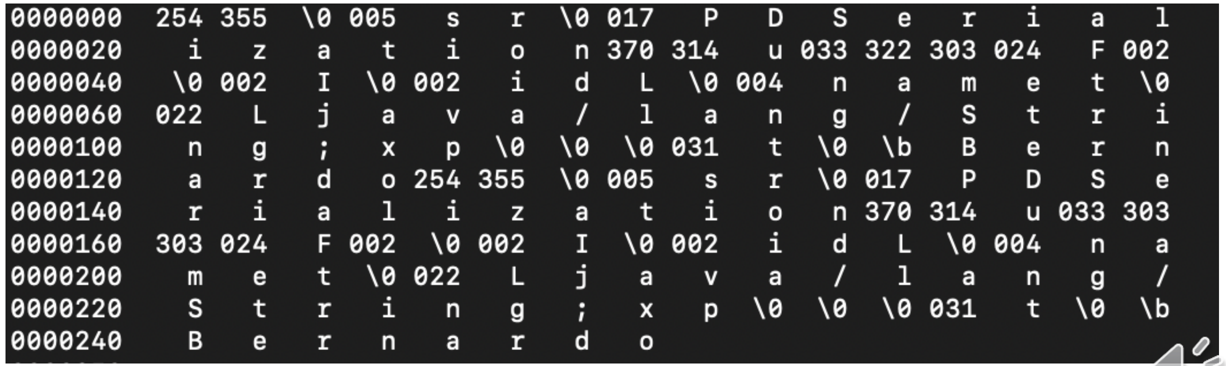
Bespoke and Serialization

- Bespoke
  - Implement a simple logging application
  - Save/load log errors to/from a text file



```
listoferrors.txt
101, 2020-07-16T08:44:23.802853, IOException occurred, severe, Bespoke, loadData
102, 2020-07-16T08:44:23.802853, Exception occurred, low, Bespoke, saveData
```

- Java Serialization
  - Implement a simple application
  - Terminal command: od -c data.ser



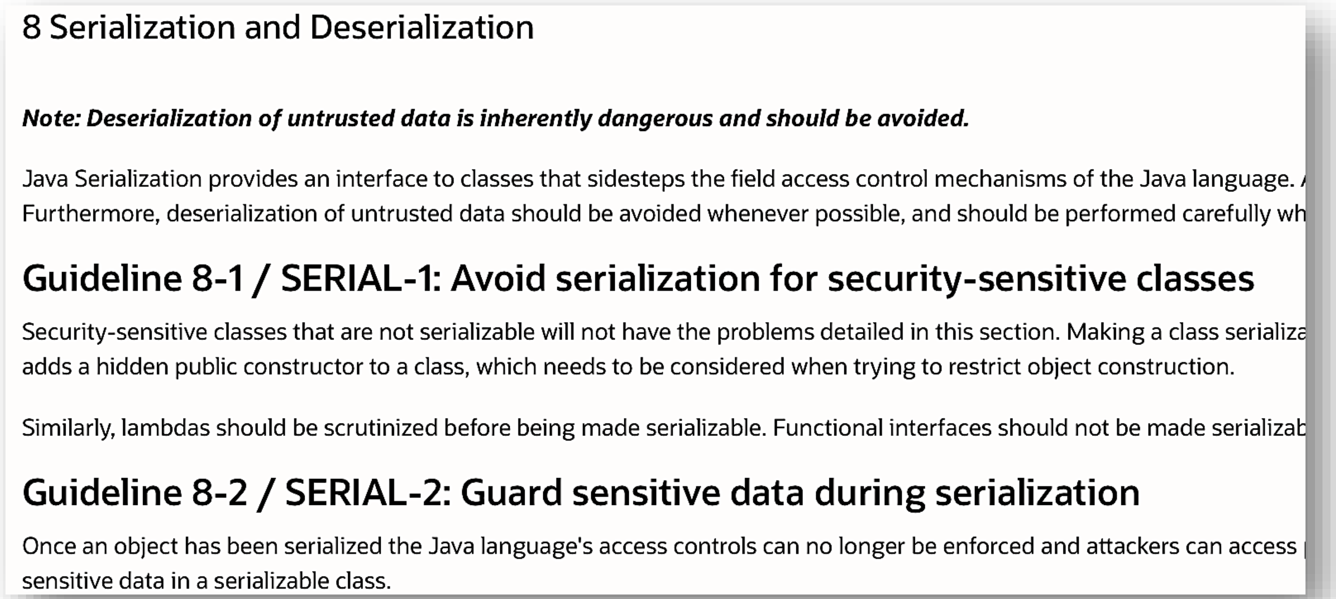
```
0000000 254 355  \0 005   s   r  \0 017   P   D   S   e   r   i   a   l
0000020   i   z   a   t   i   o   n 370 314   u 033 322 303 024   F 002
0000040  \0 002   I  \0 002   i   d   L  \0 004   n   a   m   e   t  \0
0000060 022   L   j   a   v   a   /   l   a   n   g   /   S   t   r   i
0000100   n   g   ;   x   p  \0  \0  \0 031   t  \0  \b   B   e   r   n
0000120   a   r   d   o 254 355  \0 005   s   r  \0 017   P   D   S   e
0000140   r   i   a   l   i   z   a   t   i   o   n 370 314   u 033 303
0000160 303 024   F 002  \0 002   I  \0 002   i   d   L  \0 004   n   a
0000200   m   e   t  \0 022   L   j   a   v   a   /   l   a   n   g   /
0000220   S   t   r   i   n   g   ;   x   p  \0  \0  \0 031   t  \0  \b
0000240   B   e   r   n   a   r   d   o
```

Serialization in Java

- Java Serialization
  - Class must implement Serializable
    - `public myClass implements Serializable`
  - Load serializable data by creating an `ObjectInputStream` object and casting the stream to the appropriate class type
  - Save serialized data by creating an `ObjectOutputStream` and writing the object to the stream
  - `ArrayLists` are serializable by default and are commonly used for serializing a data collection (many classes, such as `HashMaps`, are serializable (check documentation)

Serialization in Java

- Deserialization of untrusted data is inherently dangerous and not recommended
  - https://www.oracle.com/java/technologies/javase/seccodeguide.html



8 Serialization and Deserialization

Note: Deserialization of untrusted data is inherently dangerous and should be avoided.

Java Serialization provides an interface to classes that sidesteps the field access control mechanisms of the Java language. /
Furthermore, deserialization of untrusted data should be avoided whenever possible, and should be performed carefully wh

Guideline 8-1 / SERIAL-1: Avoid serialization for security-sensitive classes

Security-sensitive classes that are not serializable will not have the problems detailed in this section. Making a class serializa
adds a hidden public constructor to a class, which needs to be considered when trying to restrict object construction.

Similarly, lambdas should be scrutinized before being made serializable. Functional interfaces should not be made serializab

Guideline 8-2 / SERIAL-2: Guard sensitive data during serialization

Once an object has been serialized the Java language's access controls can no longer be enforced and attackers can access
sensitive data in a serializable class.

XML

- XML (eXtensible Markup Language)
  - Open standards for general data formatting specifications
  - Cross platforms, cross programming languages
  - Wide industry support (W3C)
  - A plenty of tools and programming libraries
  - Long history of deployment
- Example
  - HTML
  - .docx (Word document) is represented using XML

XML

- XML Structure / Tree
  - XML is case sensitive! <Root> ≠ <root>

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<root>
  <child attributes="0">
    <subchild>…</subchild>
    <subchild>…</subchild>
     …
  </child>
  <child>
    <subchild>…</subchild>
    <subchild>…</subchild>
  </child>
…
</root>
```

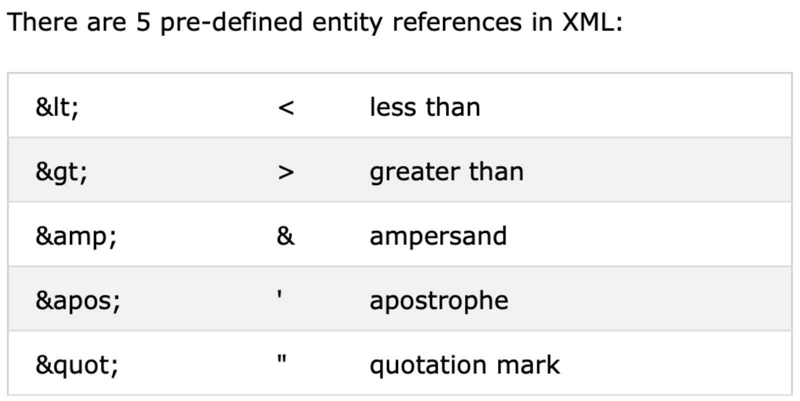
XML Example

- XML example:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<People>
  <Person id=“1”>
    <FirstName> Homer </FirstName>
    <LastName> Simpson </LastName>
  </Person>
  <Person id=“2”>
    <FirstName> Johnny </FirstName>
    <LastName> Goodman </LastName>
  </Person>
</People>
```

XML

- XML parser error!
  - Use &lt; instead of “<”
  - https://www.w3schools.com/xml/xml_syntax.asp



There are 5 pre-defined entity references in XML:

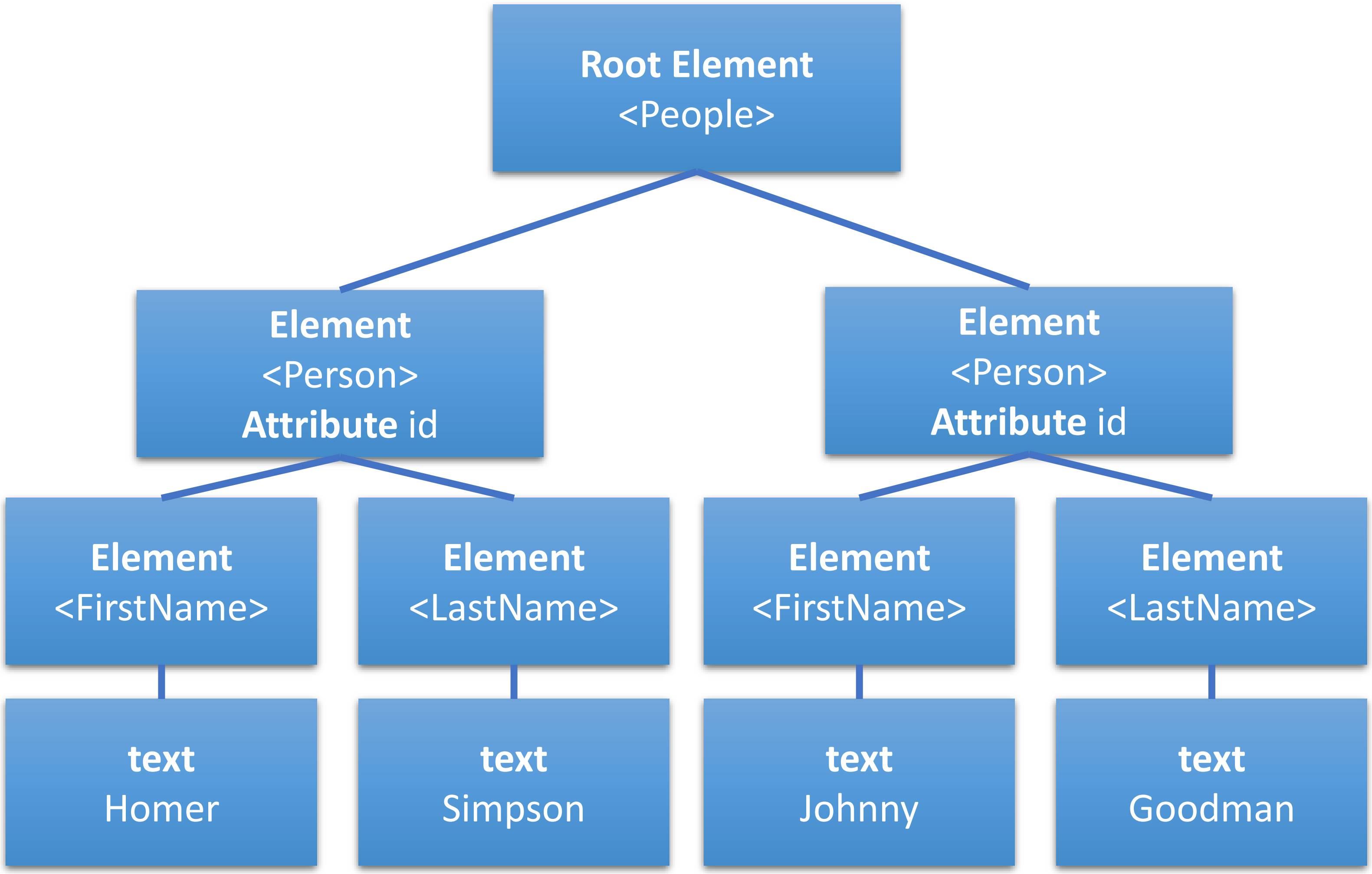
&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<root>
  <child attributes=“0”>
    <subchild> 10 < x < 100 </subchild>
    <subchild>…</subchild>
     …
  </child>
…
</root>
```

Two Options for XML in Java

- Two approaches:
  - SAX
    - Simple API for XML
    - SAX treats XML as stream and allows extraction of data as stream is read - preferable for very large documents (gigabyte)
  - DOM
    - Document Object Model (structured around XML standard)
    - Java DOM reads in entire XML tree and generate the node object
- SAX is faster and more efficient than DOM
- DOM has more structures than SAX

XML DOM



Root Element
<People>
Element
<Person>
Attribute id
Element
<Person>
Attribute id
Element
<FirstName>
Element
<LastName>
Element
<FirstName>
Element
<LastName>
text
Homer
text
Simpson
text
Johnny
text
Goodman

XML DOM

- DOM requires a number of steps to save data to file:
  - Create a `DocumentBuilder` (uses `DocumentBuilderFactory`)
  - Document created from a `DocumentBuilder` object
  - Create and append elements
  - Transform the XML to a Result (output file)
- Similar series of steps for loading XML/DOM:
  - `DocumentBuilderFactory`
  - Document Builder
  - Document
  - Class data structures

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import org.w3c.dom.*;
…
…
```

Pros and Cons of XML

Pros	Cons
• Robust, extendable • More human readable • Platform independent • Programming language independent • XML supports Unicode (international encoding) • Easy format verification • Can represent many data structures (trees, lists…) • Native support in Java	• XML syntax is verbose and redundant • XML file sizes are usually big because of above • Does not support Array

JSON

- JavaScript Object Notation (JSON)
  - Like XML, is also an open standard for data format that is widely used
  - Originally designed for sending data between web client and server, but also very useful for data storage
- Built around attribute-value pairs
- Produces smaller and more readable documents than XML
- JSON example:

```
[{"age":11,"name":"Bart"},
 {"age":40,"name":"Homer"}]
```

```
{“attribute-name”:{JSON object}}
{“attribute-name”:“string”}
{“attribute-name”:[array]}
{“attribute-name”:1} (number)
{“attribute-name”:true} (boolean)
{“attribute-name”:null}
```

Pros and Cons of JSON

Pros	Cons
• More lightweight • Human readable • Straightforward to implement • Support array and null • Can easily distinguish boolean, number, and string type • Data is available as JSON objects	• Lacking language specific features of XML (e.g., XML attributes..) • No native support in Java • No display capabilities (no markup language)

Database

- Database management systems (DBMS) are commonly used for storage of large volumes of data
  - Fast and efficient large data retrieval and processing
  - Parallel and distributed data retrieval and processing
- Relational databases
  - Linking tables through unique identifiers to avoid problems of duplicating data entries
  - Standardized data retrieval and processing commands (e.g., SQL)

PostgreSQL ORACLE® DATABASE Microsoft® SQL Server™ MySQL™ MariaDB

Database Example

- Represent a person in a bespoke/csv file:

```
id, FullName, HomePhone, MobilePhone, WorkPhone
1, Alice, 555-555, 123-321, ?
2, Bob, ? ,123-222, ?
```

- Relational Database (RDB)
  - SQL (Structure Query Language) designed for data query and manipulation

Person

id	FullName
1	Alice
2	Bob
...	...

ContactPhone

id	PhoneNumber
1	555-555
2	123-222
1	123-321

Reference

- IBM developer works 5 things you need to know about serialization
  - https://developer.ibm.com/technologies/java/articles/j-5things1/
- Oracle serialization FAQ
  - https://www.oracle.com/technetwork/java/javase/tech/serializationfaq-jsp-136699.html
- W3C XML standards pages
  - https://www.w3.org/standards/
- JSON
  - https://www.json.org/
  - https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf