While folks are joining

- Get you laptops ready and login to www.crio.do
- Open Qmoney ME and start your workspace.
- Open Terminal and type
  - cd ~/workspace
- Clone the repo in ~/workspace directory
  - git clone git@gitlab.crio.do:bdt-sprint-codes/core-java-i/core-java-i-session-activities.git
- Open session-1 folder.
- Wait for Java Language Server Setup to complete.
- Setup Video for Reference

Crio Sprint: CORE-JAVA-1

Session 1

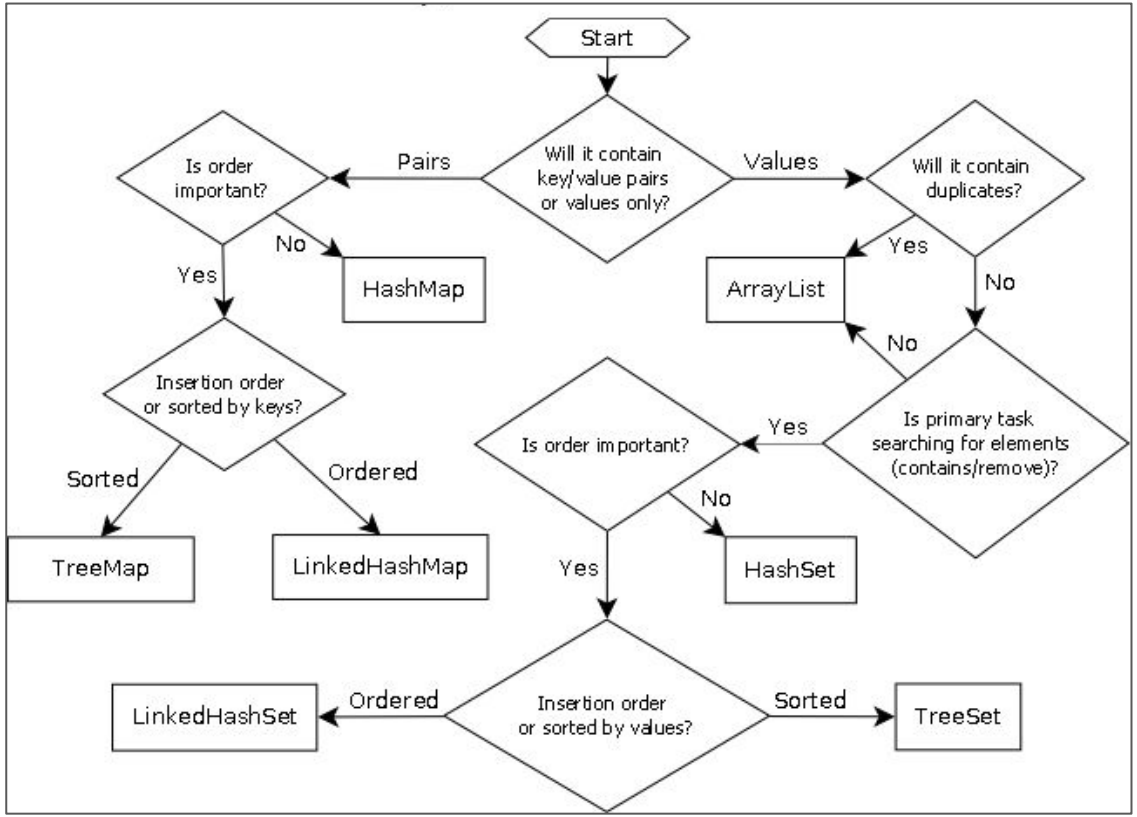
Contents of CORE-JAVA-1 Sprint

- ME: QMoney
- JSON
- Jackson
- HTTP & REST API Basics
- Comparator & Comparable
- Builder Design Pattern
- Java Streams
- Factory Design Pattern
- Gradle
- Exception Handling
- MultiThreading

Today’s Session Agenda

- Recap
  - Java Collection
- JSON
- Jackson
- ME: QMoney Intro
- ME: QMoney Module 1 Intro

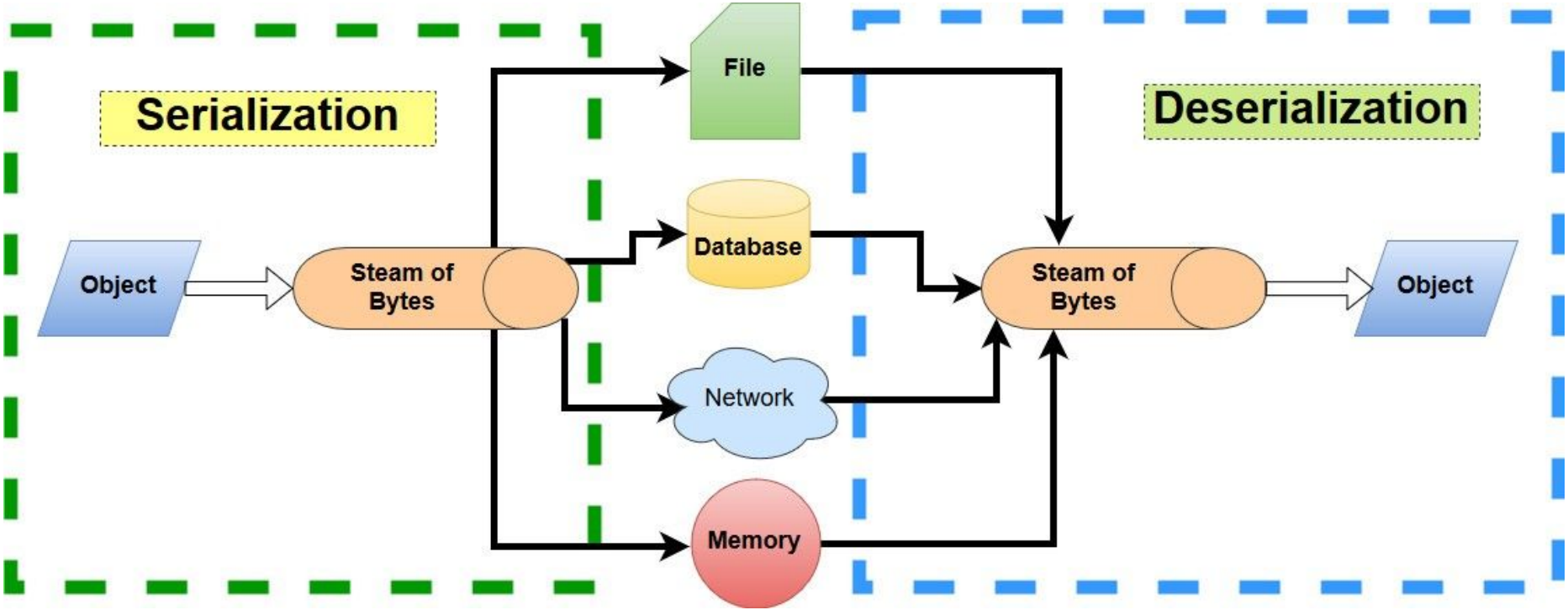
Recap - Java Collection



Thinking Caps

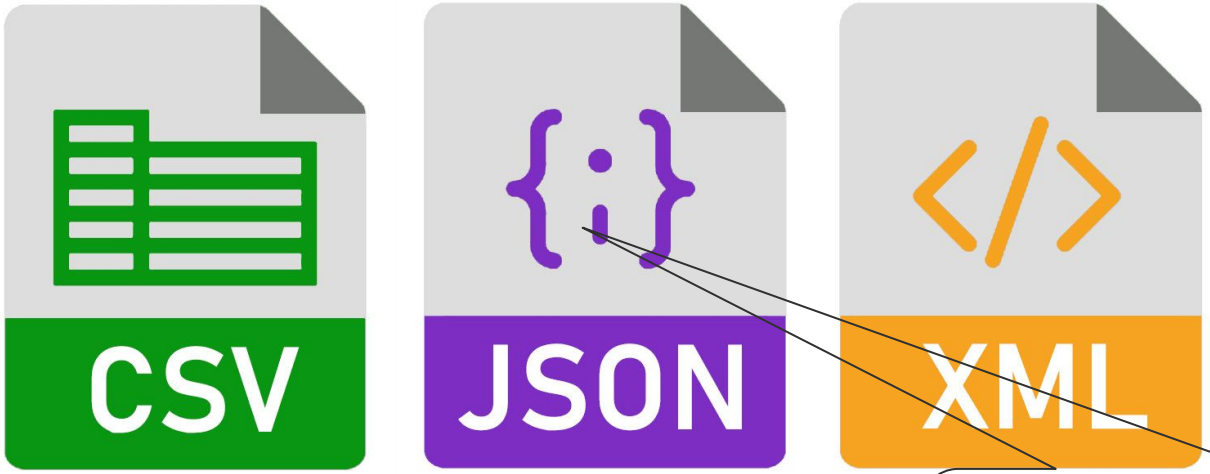
- What happens to data stored in Java Collections when program execution is completed?
- How can we persist the data even after completion of the program?
- Can persistent data be utilized by program written in any language? Why?
- In which format is data transferred across internet?
- What are the different video formats supported by Youtube video?
- How does Youtube convert data packets into streaming video?
    - Using a Decoder
- What is the process of encoding / decoding called as?
    - Serialization and Deserialization

(De) Serialization



Data File Format

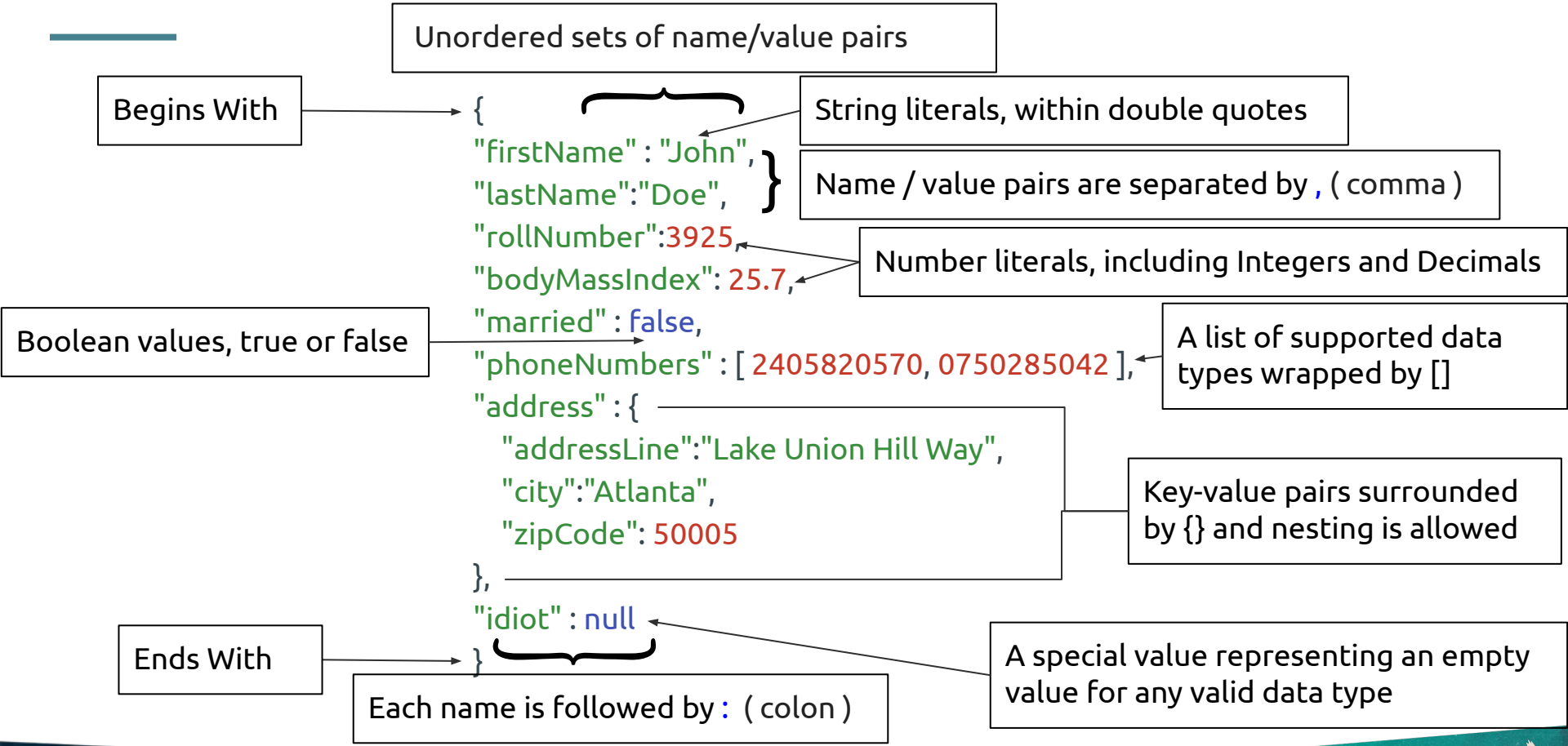
- Can you list down famous data formats used to store / exchange data?



- Less common - many more like Avro, Parquet, ORC ,etc.

- Most common format to transfer data on the internet. Human readable.
- Microservices - mostly use JSON to exchange information.

JSON

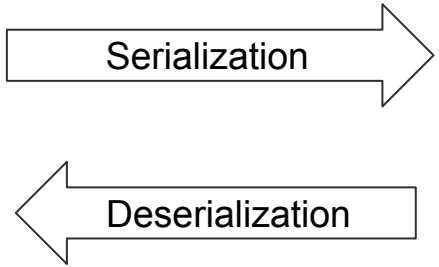

Unordered sets of name/value pairs
Begins With
{
"firstName" : "John",
"lastName":"Doe",
"rollNumber":3925,
"bodyMassIndex": 25.7,
"married" : false,
"phoneNumbers" : [2405820570, 0750285042],
"address" : {
"addressLine":"Lake Union Hill Way",
"city":"Atlanta",
"zipCode": 50005
},
"idiot" : null
}
String literals, within double quotes
Name / value pairs are separated by , (comma)
Number literals, including Integers and Decimals
Boolean values, true or false
A list of supported data types wrapped by []
Key-value pairs surrounded by {} and nesting is allowed
Ends With
Each name is followed by : (colon)
A special value representing an empty value for any valid data type

JSON (De)Serialization

Java stores information as Objects. How to (De) Serialize Java POJO to JSON vice-versa?

```
ObjectMapper om = new ObjectMapper();
Member m = new Member("J Jonah Jameson",29,"Omni Man");
String s = om.writeValue(outputFile,m);
```

```
public class Member {
   public String name;
   public Integer age;
   public String secretIdentity;
   ...
}
```

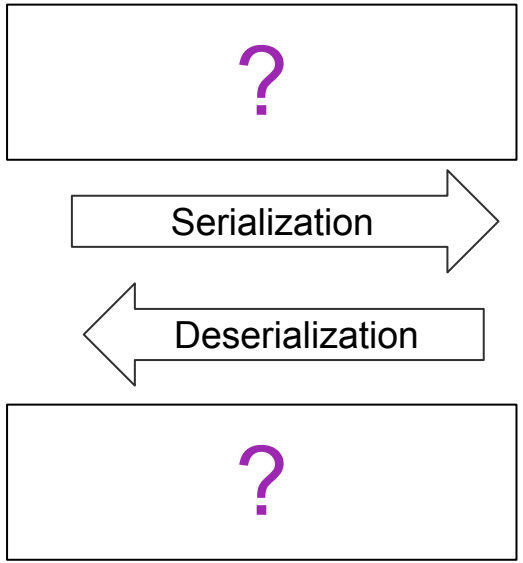



```
{
   "name": "J Jonah Jameson",
   "age": 60,
   "secretIdentity": "Omni Man"
}
```

```
String inputFile = "{\"name\": \"J Jonah Jameson\",\"age\": 60,\"secretIdentity\": \"Omni Man\"}";
ObjectMapper om = new ObjectMapper();
Member m = om.readValue(inputFile, Member.class);
```

Activity #1 - Parse Stock Data

```
public class Trade {
  public String symbol;
  public int quantity;
  public String purchaseDate;
}
```

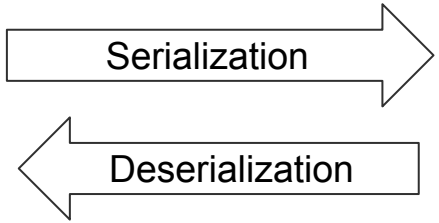



```
[
  {
    "symbol": "AAPL",
    "purchaseDate": "2019-01-02"
    "quantity": 100
  },
  {
   "quantity": 10,
   "purchaseDate": "2019-01-02",
    "symbol": "MSFT"
  }
]
```

- Complete the (De)Serialization Logic in parseJSONJacksomattically method.
- How does Jackson know which variable to map a JSON key to?
- What happens if change the name of symbol to symbl in POJO? Edit and Run the Program.
- What happens if change data type of quantity to String in POJO? Edit and Run the program.
- What will Jackson if there are duplicate keys in JSON? Try it out!

Activity #2 - Annotations to the Rescue

```
public class Trade {
  public String symbol;
  public int quantity;
  public String purchaseDate;
}
```



Keys don't match ?

```
[
  {
    "1. symbol": "AAPL",
    "2. quantity": 100,
    "3. purchaseDate": "2019-01-02",
  },
  {
    "1. symbol": "MSFT",
    "2. quantity": 10,
    "3. purchaseDate": "2019-01-02",
  }
]
```

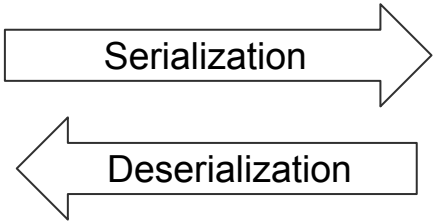
- Copy (De)Serialization logic from activity 1 to this activity.
- Run the program. What do you see?
- Suddenly keys have changed. What would you do? Do we have better solution?
  - Use @JsonProperty Annotation
- Annotate other required fields as well. Execute the program.

```
public class Trade {

  @JsonProperty("1. Symbol")
  public String symbol;
```

Activity #2.1 - Annotations to the Rescue

```
public class Trade {
  public String symbol;
  public int quantity;
  public String purchaseDate;
}
```




```
[
  {
    "1. symbol": "AAPL",
    "2. quantity": 100,
    "3. purchaseDate": "2019-01-02",
    "4. weather": "Rainy"
  },
  {
    "1. symbol": "MSFT",
    "2. quantity": 10,
    "3. purchaseDate": "2019-01-02",
    "4. weather": "Sunny"

  }
]
```

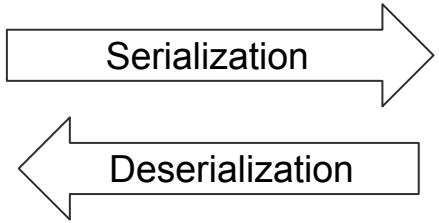
Unwanted fields ?

- Copy (De)Serialization logic from activity 1 to this activity.
- Copy changes made in Trade.java from activity 2 to this activity.
- Run the program. What do you see?
- Unwanted field was added in JSON. What would you do? Do we have better solution?
  - Use @JsonIgnoreProperties Annotation
  - Annotate the Trade POJO

Activity #3 - Private Data

```
public class Trade {
  private String symbol;
  private int quantity;
  private String purchaseDate;
}
```

Fields are private ?

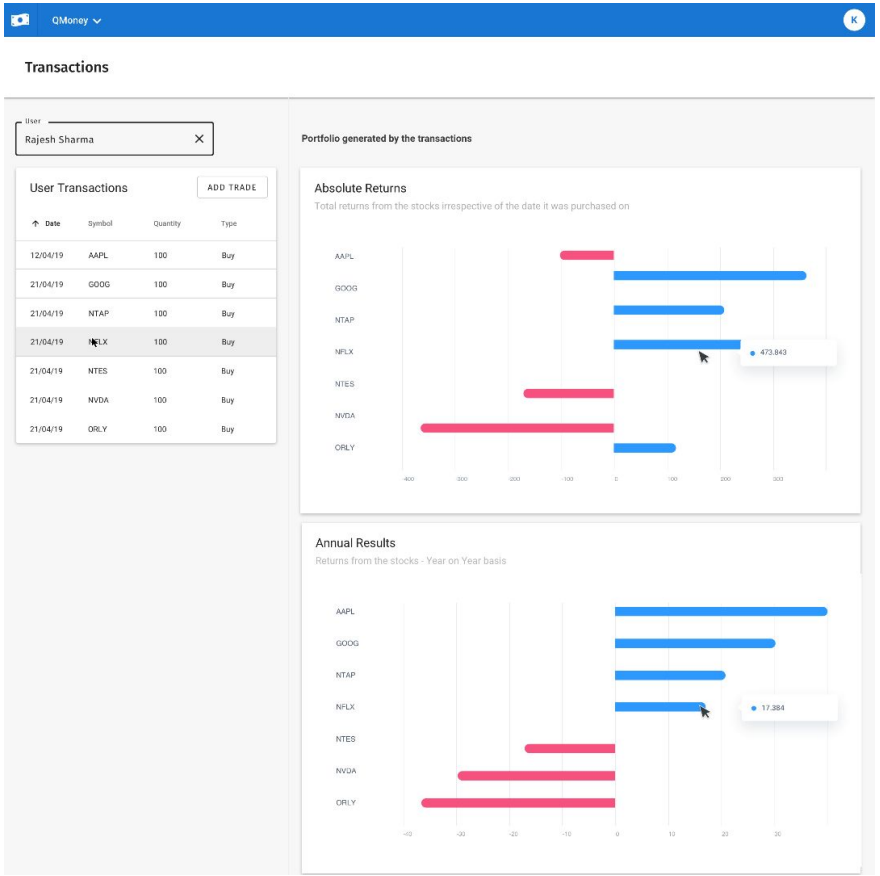



```
[
  {
    "symbol": "AAPL",
    "purchaseDate": "2019-01-02"
    "quantity": 100
  },
  {
   "quantity": 10,
   "purchaseDate": "2019-01-02",
    "symbol": "MSFT"
  }
]
```

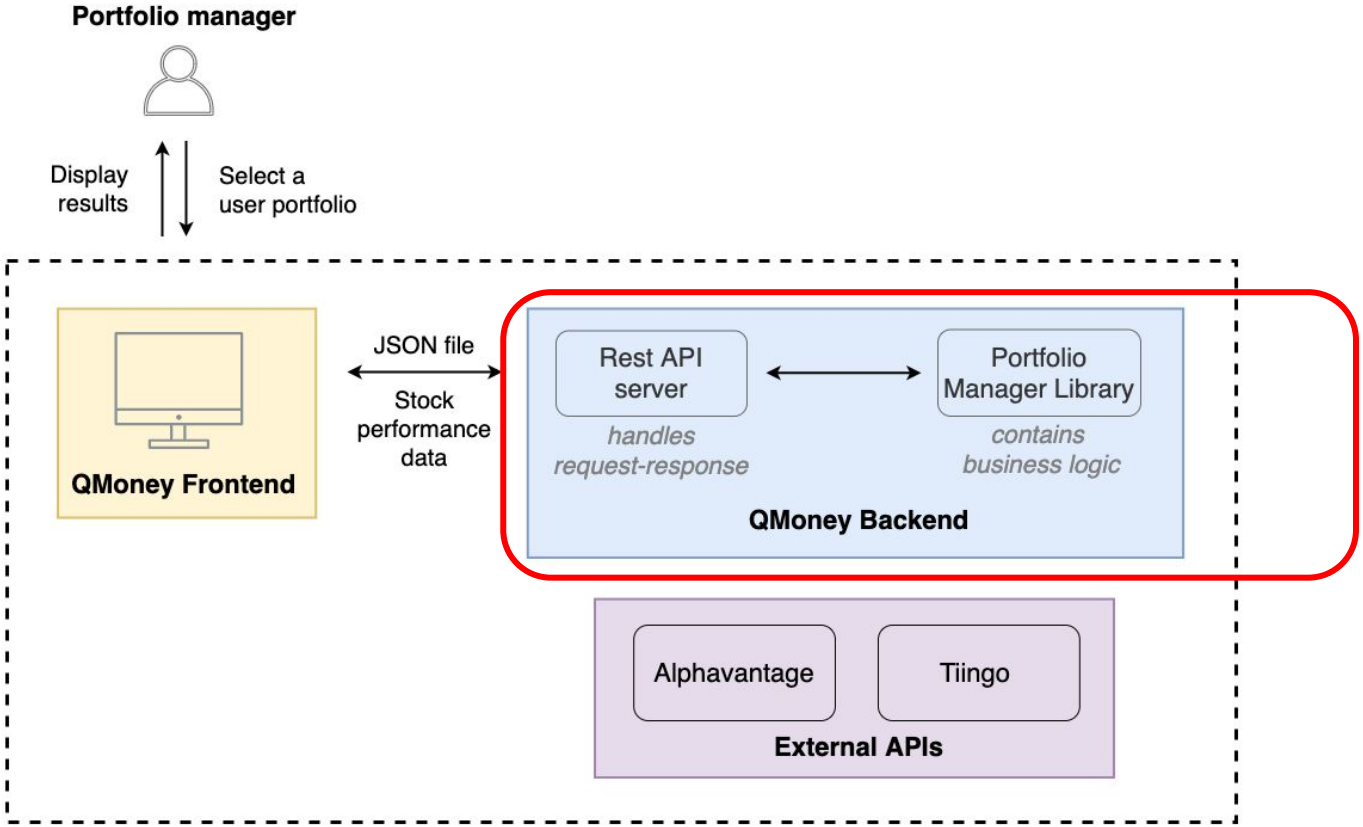
- Copy (De)Serialization logic from activity 1 to this activity.
- Run the program. What do you see? Any solution?
  - Add Getters
- What happens if change getSymbol to getSYmbol?
- Is there any annotation to make non-public field serializable without Getters? Google it.

Introduction to QMoney

- Stock Analyzer Tool for Portfolio Managers
- Annualized Returns and Absolute Returns



QMoney Architecture



Concepts Covered in QMoney

- JSON
- Jackson
- Consuming REST API
- Gradle
- Factory Pattern
- Exception Handling
- Concurrency

QMoney Product Features to be implemented

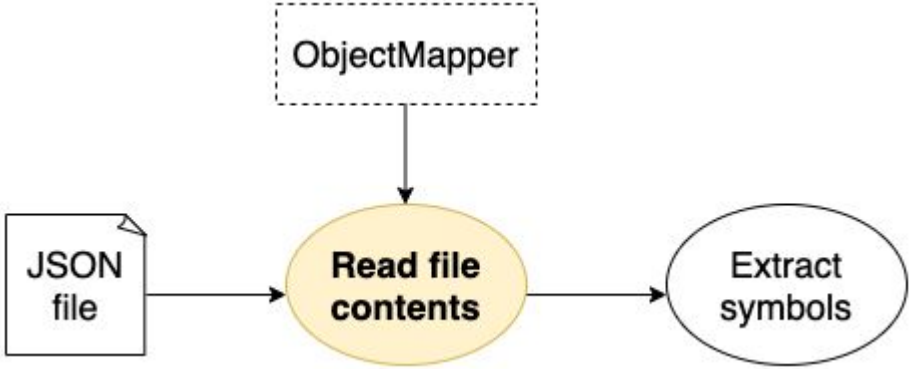
1. Read user portfolio file
2. Get stock quotes from a third-party provider
3. Implement logic to perform calculations
4. Create a portfolio management library
5. Publish the library
6. Add another service provider
7. Handle user issues
8. Enhance performance of the app

} Optional

Module 1 Overview - JSON Parsing

- User Stock Portfolio is in the form of JSON Data format.
- In the longer term, this JSON Data will be retrieved through REST API but for this module, data will be retrieved from a temporary JSON file.
- Learn how to parse this data and perform operations on it.
- This will be required to calculate Annualized Returns of Portfolio in Upcoming Modules.

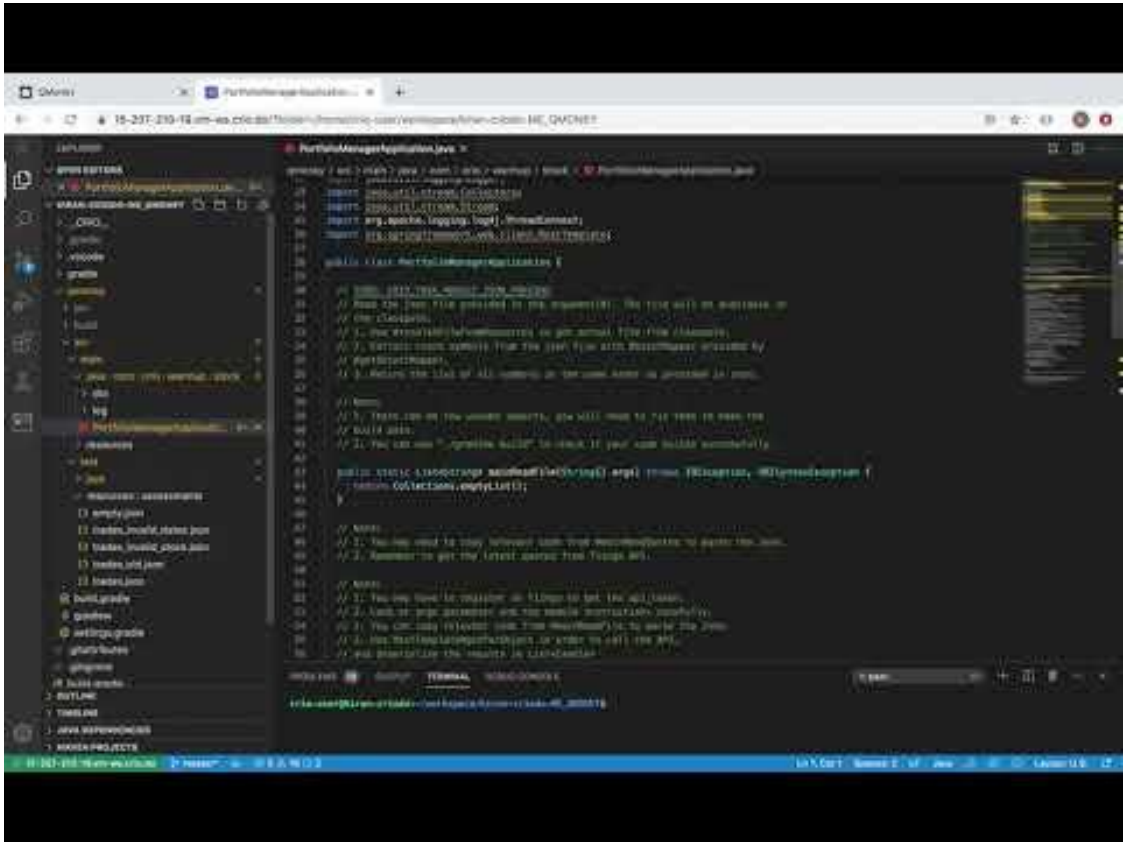
Module 1 Intro - JSON Parsing



Files to be modified

PortfolioManager.java

QMoney Module Intro Video (Available on Crio.Do)



Take home exercises for the session

- Byte: Jackson (Crio.Do)
- Byte: Jackson Advanced (Crio.Do)
- ME: QMoney - Module 1

Feedback

Thank you for joining in today.

We'd love to hear your thoughts and feedback - Feedback for CORE-JAVA-1 Session

Further Reading

- Do+JSON+with+Jackson+by+Baeldung.pdf

References

- Byte: Jackson (Crio.Do)

Thank you