Ballerina

Mastering Web Backend Fundamentals - 1

Mastering Data - Data Persistence and Visualization

Hello!

Hasitha Aravinda

hasitha@wso2.com | Associate Director / Architect | @BallerinaLang | WSO2

Maryam Ziyad

maryamm@wso2.com | Technical Lead| @BallerinaLang | WSO2

Malintha Ranasinghe

malinthar@wso2.com | Senior Software Engineer | @BallerinaLang | WSO2

About This Session

Mastering Web Backend Fundamentals

Mastering Data - Data Persistence and Visualization [Today]

- Learn the importance and the basics of data and data persistence
- Discover how to persist data in various data stores with ease
- Create interactive visualizations with Ballerina, while learning the basics

Concept to Cloud - Exploring Web Development [TBA]

- Learn efficient approaches to creating REST APIs
- Write REST APIs with Ballerina
- Deploy your web application to the cloud

Application development Hackathon [TBA]

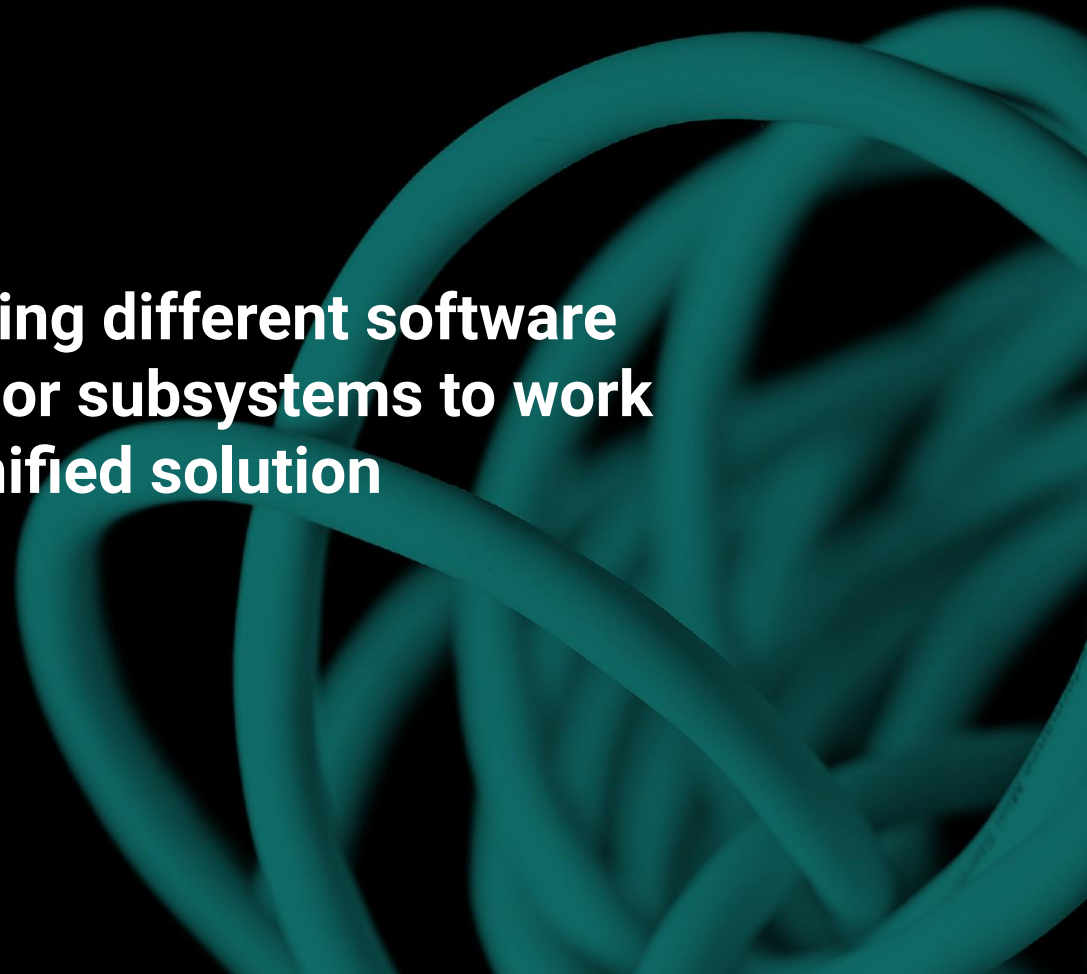
Mastering Data


"Today, every company is a software company."
Ballerina
Microsoft CEO - "Satya Nadella"

Terminology

Integration

The process of combining different software components, systems, or subsystems to work together as a single, unified solution

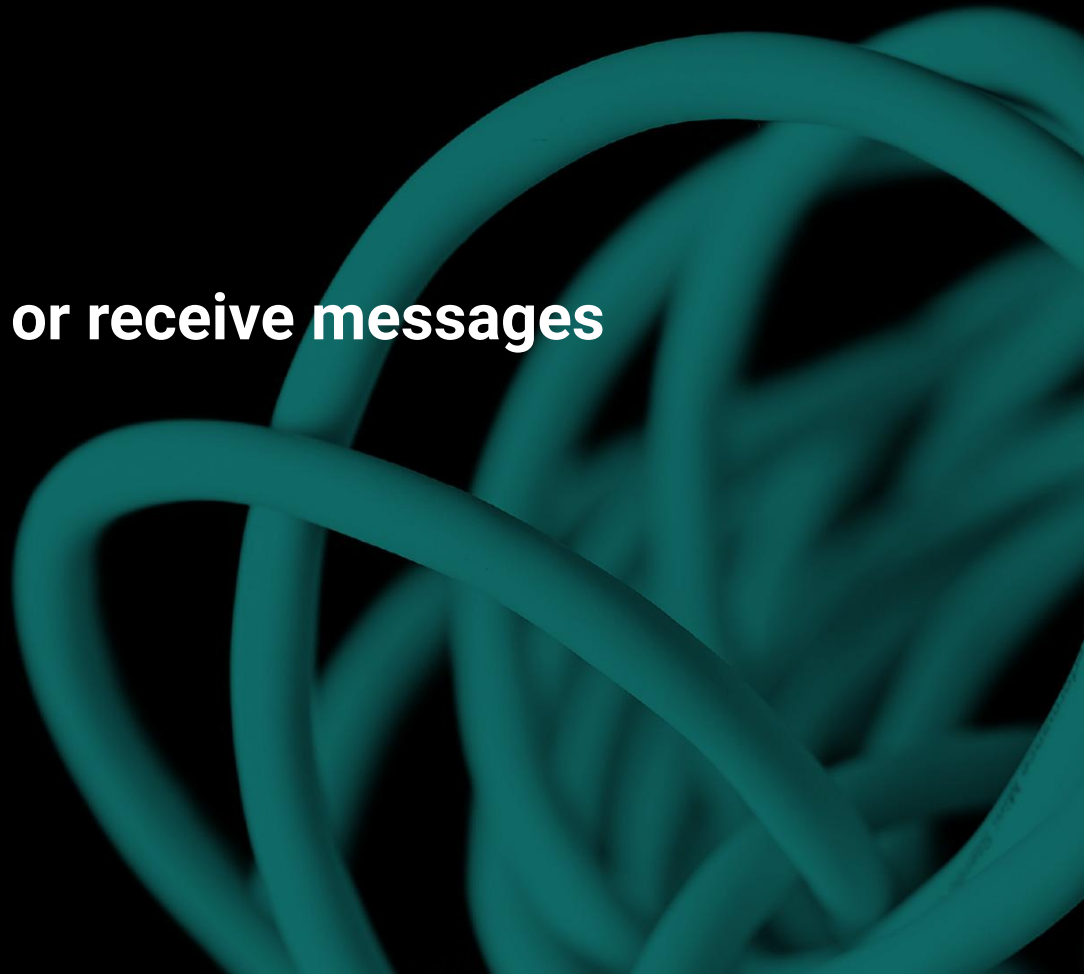


Integration - Examples

- Software Integration
- System Integration
- Data Integration
- API Integration
- Middleware Integration
- Continuous Integration
- Frontend and Backend Integration

Endpoints

An entity that can send or receive messages



Endpoints - Examples

- Databases
- Cloud Service Providers
  - AWS, Azure, Google, Huawei
- SaaS Products
  - Salesforce
  - SAP
- Files

➔ Inbound and Ingress are about things coming in.
➔ Outbound and Egress are about things going out.

Protocols

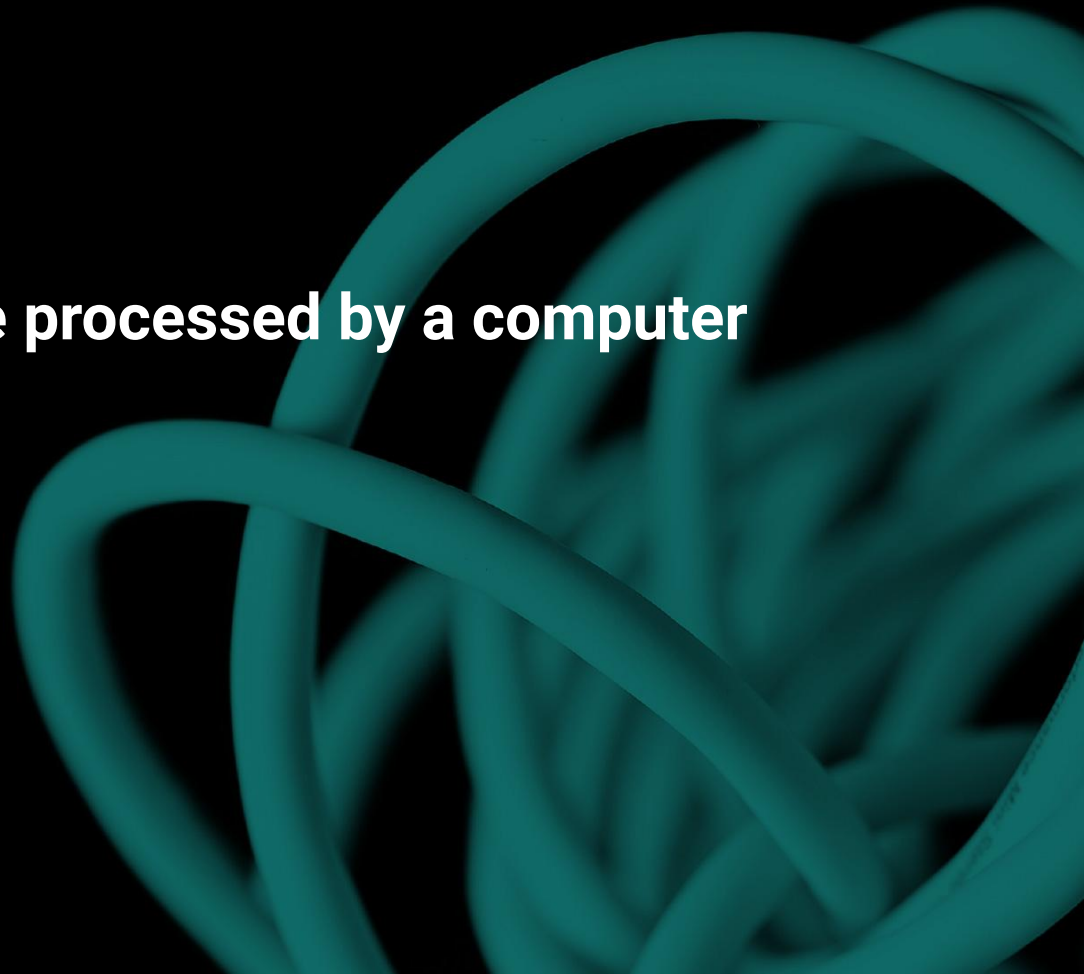
A set of rules or standards used to allow devices to communicate.

Protocols

- Transport Layer Protocols: TCP/UDP
- Web Services and APIs:
  - HTTP, gRPC, GraphQL, WebSocket, SOAP
- Data Exchange Protocols (EDI)
  - FTP, AS2, OFTP2
- Messaging Protocols
  - AMQP, MQTT, STOMP
- Database
  - Proprietary Protocols, APIs - ODBC, JDBC
- File Sharing
  - SMB, NFS
- Email
  - SMTP, IMAP, POP3

Data

Information that can be processed by a computer system.



Data Formats

- XML
- JSON
- YAML
- TOML
- Binary
- Plain Text
- CSV
- …

Data

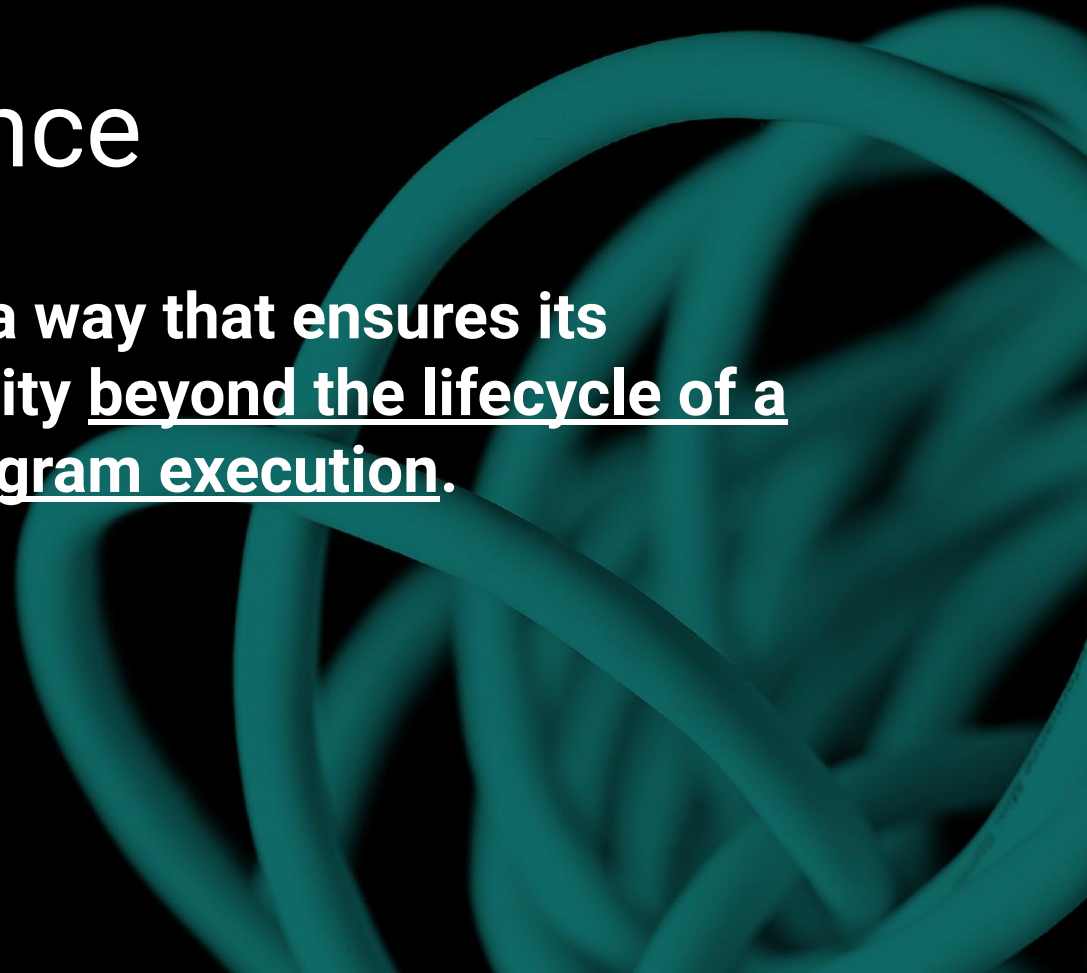
- **Binary**: Data represented in a non-human-readable format
- **Textual**: Data represented in a human-readable format

- **Structured** - Data is stored in tables of a SQL Database, Dates & time
- **Unstructured** - Plain text doc, audio, video
- **Semi-Structured** - XML and JSON

- **Time-Series** - Weather data, Stock Prices
- **Streaming**: (Bounded/Unbounded) - IoT streams, Multiplayer games data
- **Multimedia**: Images, audio, and video files
- …

Data Persistence

The storage of data in a way that ensures its continuity and availability beyond the lifecycle of a specific process or program execution.



Why is Data Persistence Important?

- Continuity of Operations
- Historical Analysis
- Legal and Regulatory Compliance
- Enhancing User Experience

Methods of Data Persistence

- Databases
  - MySQL, PostgreSQL, MongoDB, …
- Files
  - CSV, XML, JSON
- Data Warehouses
  - Amazon Redshift, Google BigQuery, Snowflake.
- Cloud Storages
  - AWS S3, Google Cloud Storage, Azure Blob
- Cloud Service and Software
  - Google Sheets
- In-memory
  - Caches, Redis
- Blockchain

Challenges in Data Persistence

- Data Integrity
- Scalability
- Security
- Performance
- Cost

Transactions

A transaction is a sequence of one or more operations (like reading, inserting, updating, or deleting data) that is executed as a single unit.

Either all operations within the transaction are completed successfully, or none of them.

- Atomicity
- Consistency
- Isolation
- Durability

Why Transactions are Essential for Data Persistence

- Protecting Data Integrity
- Handling System Failures
- Simplifying Complex Operations
- Supporting Concurrent Operations
- Ensuring Business Logic

Example
Data Persistence in Databases

Methods of Interaction with a database

- Direct Database Interaction
  - Native SQL (Structured Query Language) Queries
  - Create, Read, Update, Delete (CRUD)
  - Pros:
    - More control over query optimization.
    - Often provides better performance for complex queries.
  - Cons:
    - Can lead to SQL injection vulnerabilities if not handled properly.
    - Tightly couples application logic with database-specific SQL dialects.
    - Harder to migrate to another database system.

Methods of Interaction with a database

- ORM (Object-Relational Mapping) Based Interaction
  - Entities and Persistence Layers
  - Examples: Hibernate / OpenJPA (Java), Entity Framework (C#), Sequelize (JavaScript), Bal Persist (Ballerina)
  - Pros:
    - Provides a more intuitive and object/data-oriented way to interact with the database.
    - Reduces the risk of SQL injection, as the ORM often handles query creation.
    - Can be database agnostic, allowing easier migrations between different database systems.
  - Cons:
    - Sometimes there is a performance overhead.
    - For very complex queries, ORM might not be as efficient as hand-tuned SQL.

Points to Consider

- Specific requirements of the project.
- The expertise of the development team.
- The trade-offs between direct control and abstraction.
    - Direct database interaction is often favored for **high-performance** applications
    - ORM-based interaction, on the other hand, is popular for its productivity benefits and for projects where **database independence** is a key consideration.

Ballerina Persist Tool

Bal Persist offers

- Simplified Database Interaction
- Developer Productivity
- Data Mapping
- Database independence
- Consistency and Integrity

Bal Persist - Highlights

- Define entities using record syntax
- An Entity must contain at least one identifier field.
- Relationship between two entities.
  - 1-1
  - 0-n
  - n-n
- Visualize data model as an ER diagram
- Data model validation and Code Actions support

Bal Persist - Design

See entities as REST resources

/<Resource>/<key>

Basic REST operations

- GET (Default Operation) - Retrieve entity data
- POST - Create/Submit an entity
- PUT - Update an existing entity
- DELETE - Remove an entity

Demo

Let's build a simple library system

Entities

- Books
- Authors
- Members
- Borrowings

Relationships

- An Author can have zero or more Books.
- A Book has an Author.
- A Member can have zero or more Borrowings.
- A Book can be borrowed by zero or one Member at a time.

Why Ballerina?

Challenges

- Robustness Principle - "Be conservative in what you send, be liberal in what you accept"
- Working With Data
- Type Safety
  - Convert to application-specific data types
  - A type-safe way to manipulate data
- Transactions


The best way to manipulate data
is to represent data as data
Source
https://www.manning.com/books/data-oriented-programming
Ballerina

Plain Data

- Pure data, independent of processing that might be applied to the data
- Messages exchanged by network protocols are represented by plain data
- Can be directly serialized to and from JSON in a simple, natural way

Data and Object Oriented Programming

- Encapsulation and behavior-centric approach

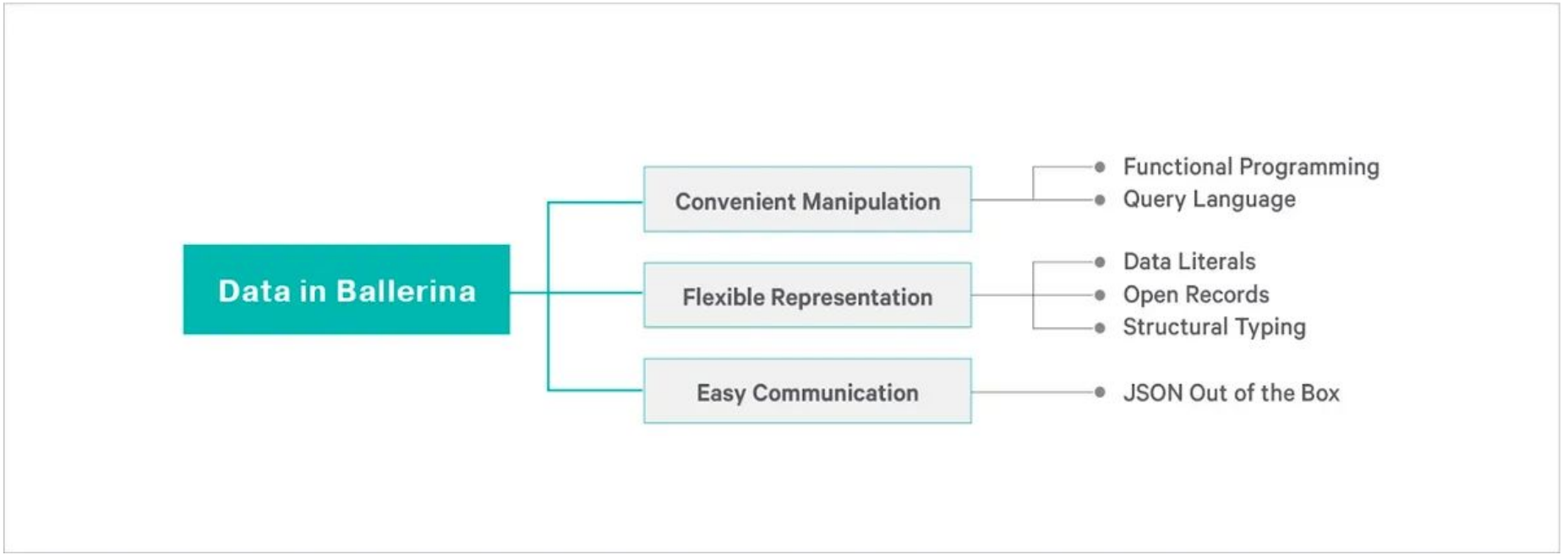
- Great for apps with complicated logic with several boundaries
  - Defining and defending boundaries
  - Ensures data integrity and restricts direct access.
  - Ideal for monoliths, allows multiple teams to collaborate

- But, does OOP work for transferring data?
  - Serialization and deserialization can be costly.
  - Not always efficient for batch operations or data streaming.

Data Oriented Programming

- Focus on efficient manipulation, representation, and storage of data
- Model data as (immutable) data
  - Separate code from Data
- Great for handling network interactions

Handling Data in Ballerina



Source https://www.infoq.com/articles/ballerina-data-oriented-language/

JSON

```json
{
    "firstName": "John",
    "lastName": "Doe",
    "age": 17,
    "books": [
        {
            "title": "The Volleyball Handbook",
            "author": {
                "firstName": "Robert",
                "lastName": "Miller"
            }
        },
        {
            "title": "Clean Code",
            "author": {
                "firstName": "Robert",
                "lastName": "Martin"
            }
        }
    ]
}
```

Java records

```java
record Author(String name, String address) {
}
record Book(String title, Author author) {
}
record Member(String lastName, String firstName, int age, Book[] books) {
}

Run | Debug
public static void main(String[] args) {
    Member kelly = new Member(
            lastName:"Kelly",
            firstName:"Kapowski",
            age:17,
            new Book[]{
                    new Book(
                            title:"The Volleyball Handbook",
                            new Author(name:"Bob", address:"Miller")
                    )}
    );
    System.out.println(kelly);
}
```

Ballerina

Ballerina records

```
type Author record {
    string firstName;
    string lastName;
};

type Book record {
    string title;
    Author author;
};

type Member record {
    string firstName;
    string lastName;
    int age;
    Book[] books;
};
```

```
Member kelly = {
    firstName: "Kelly",
    lastName: "Kapowski",
    age: 17,
    books: [
        {
            title: "The Volleyball Handbook",
            author: {
                firstName: "Bob",
                lastName: "Miller"
            }
        }
    ]
};
```

Algebraic Types / Union Types - Java

```java
sealed interface JsonValue permits JsonString, JsonNumber, JsonNull, JsonBoolean, JsonArray, JsonObject {}

record JsonString(String s) implements JsonValue { }
record JsonNumber(double d) implements JsonValue { }
record JsonNull() implements JsonValue { }
record JsonBoolean(boolean b) implements JsonValue { }
record JsonArray(List<JsonValue> values) implements JsonValue { }
record JsonObject(Map<String, JsonValue> pairs) implements JsonValue { }

Run | Debug
public static void main(String[] args) {
    JsonValue j = new JsonObject(Map.of(
            k1:"name", new JsonString(s:"John Doe"),
            k2:"age", new JsonNumber(d:43),
            k3:"city", new JsonString(s:"New York")
    ));
    if (j instanceof JsonObject(var pairs)
            && pairs.get(key:"name") instanceof JsonString(String name)
            && pairs.get(key:"age") instanceof JsonNumber(double age)
            && pairs.get(key:"city") instanceof JsonString(String city)) {
        // use name, age, city
        JsonString nameJson = (JsonString) pairs.get(key:"name");
        JsonNumber ageJson = (JsonNumber) pairs.get(key:"age");
        JsonString cityJson = (JsonString) pairs.get(key:"city");
    }
}
```

Algebraic Types / Union Types - Ballerina

```
// json is a built-in type in Balleirna
// It is the union of () | boolean | int | float | decimal | string | json[] | map<json>

type Person record {|
    string name;
    int age;
    string city;
|};

Run | Debug | Visualize
public function main() {
    json j = {name:"John Doe", age:43, city:"New York"};

    Person|error p = j.fromJsonWithType();
    if p is Person {
        string name = p.name;
        int age = p.age;
        string city = p.city;
        io:println(string `Name: ${name}, Age: ${age}, City: ${city}`);
    } else {
        io:println("j is not a Person");
    }
}
```

Query expressions in Ballerina

```
// Prints the top 10 countries having the highest case-fatality ratio.
Run | Debug | Visualize
public function main() returns error? {
    Country[] countries = getCountries();

    json summary =
        from var {country, continent, population, cases, deaths} in countries
            where population >= 100000 && deaths >= 100
            let decimal caseFatalityRatio = <decimal>deaths / <decimal>cases * 100
            order by caseFatalityRatio descending
            limit 10
            select {country, continent, population, caseFatalityRatio};
    io:println(summary);
}
```

Consuming a service in Ballerina

```
type Country record {
    string country;
    int population;
    string continent;
    int cases;
    int deaths;
};

Run | Debug | Visualize
public function main() returns error? {
    http:Client diseaseEp = check new ("https://disease.sh/v3");
    Country[] countries = check diseaseEp->/covid19/countries;
    io:println(countries);
}
```

Data validation at the boundary

```
type Country record {
    string country;
    @constraint:Int {
        minValue: 100000
    }
    int population;
    string continent;
    int cases;
    @constraint:Int {
        minValue: 100
    }
    int deaths;
};

Run | Debug | Visualize
public function main() returns error? {
    http:Client diseaseEp = check new ("https://disease.sh/v3");
    Country[] countries = check diseaseEp->/covid19/countries;
    io:println(countries);
}
```


Abstractions allow **developers** to work with **higher-level concepts** rather than **getting bogged down in** the **nitty-gritty** of how those concepts are realized.
Ballerina

Learn Ballerina

1. https://ballerina.io/learn

2. WSO2 Self-Paced Training

   https://lms.wso2.com/collections/ballerina

Programing Challenge

First 10 eligible submissions get free vouchers for Ballerina Certification and Ballerina branded swag.

Steps

1. Star https://github.com/ballerina-platform/ballerina-lang
2. Extend the library system by adding one more entity.
3. Put your solution into a public GitHub repo.
4. Submit your solution in https://forms.gle/JtsR3z2Gk9ARVtPXA

Programing Challenge

- Extend the library system solution by adding the following entity.
  - Reviews
    - ➢ Many times, after reading a book, members want to leave feedback or a review.
    - ➢ A review has a rating (0-10) and a comment (string).
    - ➢ A member can leave zero or more reviews.
    - ➢ A book can have zero or more Reviews.
- The program should
  - Create 10 different books with authors.
  - Create 5 members and each with at least 1 borrowings.
  - Create 5 reviews.
  - Find and Print list of books that did borrowed by members.

Join with Ballerina Community

Discord : https://discord.gg/ballerinalang

SO https://stackoverflow.com/questions/tagged/ballerina

Twitter https://twitter.com/ballerinalang

GitHub : https://github.com/ballerina-platform

Q & A

THANK YOU