Implementing a Concert Ticket System using Data Oriented Programming

Data Oriented Programming (DOP) is a programming paradigm that separates how data is represented from how it is processed. DOP is based on the following four principles, that when used correctly, result in the creation of less complex and more reliable code:

1. **Model the Data and Only the Data:** Use read-only Java records for data and sealed types for choices. Limit behaviour using stateless methods.
2. **Make the Data Immutable:** Use read-only data types like records and enums.
3. **Validate Data at the Boundary:** Reject invalid data inside record constructors or in receiving code.
4. **Make Illegal States Unrepresentable:** Create a record or read-only data structure for every possible state.

These four principles can be realised in DOP by modelling data using **Algebraic Data Types (ADTs)** and generic data structures. An ADT is a **Product Type** (a record) or a **Sum Type** (a sealed type hierarchy or an enumeration) and should be used as follows:

- **Data:** represent data using records or immutable collections.
- **Choices:** use sealed types and enums to represent the different types of state.
- **Processing:** use stateless (static) methods and pattern matching (instanceof and switch expressions) to process the data.

DOP is not a replacement for OOP and should be used together with SOLID principles to create less complex and more flexible software.

Concert Ticket System

Entities, Attributes and Relationships

- **Attendee**
  - **ID:** Unique identifier for the attendee.
  - **Name:** Full name of the attendee.
  - **Email:** The email address of the attendee.
  - **Tickets:** List of tickets purchased by the attendee.

- **Artist**
  - **ID:** Unique identifier for the artist.
  - **Name:** Full name of the artist.
  - **Genre:** The music genre, from the set of {BLUES, COUNTRY, FOLK, HIP HOP, JAZZ, METAL, POP, PUNK, ROCK, SOUL}.

- **Concert**
  - **ID:** Unique identifier of the concert.
  - **Artist:** The performing artist.
  - **Date:** The date and time of the concert.
  - **Venue:** The venue where the concert will take place.

- **Ticket**
  - **ID:** Unique identifier for the ticket.
  - **Concert:** The associated concert.
  - **Attendee:** The attendee named on the ticket.
  - **Price:** The price of the ticket.
  - **Advisory:** Recommendations for person attending the concert.

- **Venue**
  - **ID:** Unique identifier of the venue.
  - **Name:** The full name of the venue.
  - **Type:** The type of venue, from the set of {Amphitheatre, Arena, Theatre, Conference Centre, Club, Opera House, Not Specified}.
  - **Country:** The country in which the venue is located.
  - **Capacity:** The capacity of the venue.

- **Country**
  - **Name:** The full name of the country.
  - **Temperature:** The average temperature in the country in degrees Celsius.
  - **Precipitation:** The average rainfall in the country in mm.
  - **EU Member:** is the country an EU member state?

Relationships

- An attendee can purchase multiple tickets.
- A concert take place in a venue on a specific date and at a specific time.
- A ticket is associated with one attendee and one concert.

Rules

- A ticket may be given for free, but a ticket price cannot be negative.
- The concert date must be in the future.
- A venue capacity must be at least 100.
- Tickets for concerts in non-EU countries should contain an advisory notice to bring a passport if travelling.
- Tickets for concerts in countries with an average temperature >13° should contain an advisory notice to bring light clothing with them when travelling.
- Tickets for concerts in countries with an average rainfall >1000mm should contain an advisory notice to bring waterproof clothing with them when travelling.
- Tickets for concerts in venues with tiered seating or a capacity greater than 10,000 should contain an advisory notice warning of dangers of vertigo and enochlophobia.

Operations

The system should provide operations that Allow attendees to:

- ***purchase*** a ticket for a concert.
- ***search*** for concerts using a variety of different search criteria.
- check for any ***advisory information*** about travelling to a concert.

When you have implemented the requirements above, create a ***Runner*** class with the following functionality in ***main()*** to test the application:

```
var artist = new Artist("Queen", Artist.Genre.ROCK);
var venue = new Venue("Croke Park", VenueType.STADIUM, Country.IRELAND, 80_000);
var concert = new Concert(artist, LocalDateTime.of(2025, 12, 8, 20, 30), venue);
var attendee = new Attendee("John Doe", "John.Doe@atu.ie", List.of());

var ticket = TicketSystem.purchase(new TicketOperation.Purchase(attendee, concert, 79.99));
System.out.println(ticket);

List<Concert> res = TicketSystem.search(new TicketOperation.Search(List.of(concert),
        (con -> con.date().isBefore(LocalDateTime.of(2025, Month.DECEMBER, 24, 22, 00)))));
System.out.println(res);


//Records are immutable, so we have to re-construct a new record to update
var tmp = new ArrayList<Ticket>();
tmp.addAll(attendee.tickets());
tmp.add(ticket);
attendee = new Attendee(attendee.name(), attendee.email(), tmp);
```

Exercises - Upload the Completed Lab to the VLE

- Refactor the *Optional<String>* values returned by methods in *TicketSystem* to instances of a ***generic immutable return type*** like the sealed interface *Miscellaneous* in the lecture notes.
- In your README, comment on use of the extensive use of ***immutability*** in DOP. What are its benefits and drawbacks?