The POJO vs. The Record

In the last video, we talked about the Plain Old Java Object, and we showed you how it comes with a lot of what we call boilerplate code.

It's code that's repetitive and follows certain rules.

Once created, this code is rarely looked at, or modified.

In fact, there are tools that'll just regenerate all of this code, if your underlying data, or domain model changes.

The Record type

The record was introduced in JDK 14, and became officially part of Java in JDK 16.

It's purpose is to replace the boilerplate code of the POJO, but to be more restrictive.

Java calls them "plain data carriers".

The Record type

The record is a special class that contains data, that's not meant to be altered.

In other words, it seeks to achieve immutability, for the data in its members.

It contains only the most fundamental methods, such as constructors and accessors.

Best of all, you the developer, don't have to write or generate any of this code.

Implicit or Generated Code that Java provides

What does Java tell us about what is implicitly created, when we declare a record as we did in this code?

```
public record LPAStudent(String id, String name, String dateOfBirth, String classList) {}
```

First, it's important to understand that the part that's in parentheses, is called the record header

The record header consists of record components, a comma delimited list of components.

Implicit or Generated Code that Java provides

```
public record LPAStudent(String id, String name, String dateOfBirth, String classList) {}
```

For each component in the header, Java generates:

- A field with the same name and declared type as the record component.
- The field is declared private and final.
- The field is sometimes referred to as a component field.

Implicit or Generated Code that Java provides

```
public record LPAStudent(String id, String name, String dateOfBirth, String classList) {}
```

Java generates a toString method that prints out each attribute in a formatted String.

In addition to creating a private final field for each component, Java generates a public accessor method for each component.

This method has the same name and type of the component, but it doesn't have any kind of special prefix, no get, or is, for example.

The accessor method, for id in this example, is simply id().

Why have an immutable record?

Why is the record built to be immutable?

There are more use cases for immutable data transfer objects, and keeping them well encapsulated.

You want to protect the data from unintended mutations.

POJO vs. Record

If you want to modify data on your class, you won't be using the record.

You can use the code generation options for the POJO, as we showed in the earlier video.

But if you're reading a whole lot of records, from a database or file source, and simply passing this data around, then the record is a big improvement.

Java's new type, the record

We've only touched on some of the features of the record, to give you an introduction.

When we do talk more about the final keyword, and immutability of data, especially as it may be affected by concurrent threads, we'll be revisiting this type.

We'll also be showing it to you in action, when we get to the Database and I/O sections of this course.