Java SE 17
Advanced Language Features

Records

Jesper de Jong
Software Architect

@jesperdj www.jesperdj.com

There is always something new to learn

Advanced features of Java

Become an advanced Java developer

Java SE 17
Advanced Language Features

Version Check

Version Check

This version was created by using:

- Java SE 17
- IntelliJ IDEA 2021.3

Version Check

This course is 100% applicable to:

- Java SE 17 or newer
- Any IDE or text editor that supports Java

Overview

- Records
- Sealed Classes and Interfaces
- Advanced Classes and Interfaces
- Advanced Generics
- Lambda Expressions and Method References
- Annotations
- Optional
- Try-with-resources and AutoCloseable

Examples and Exercises

Find the examples and exercises on the Pluralsight course page

Learn by working with the subject yourself

Immutable Data Objects

Immutable object

An object of which the state cannot be changed after the object is constructed.

Examples of Immutable Classes

String

Wrapper classes for primitive types (Integer, Long, ...)

BigInteger, BigDecimal

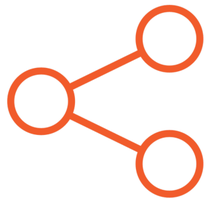
Date and time classes in the package java.time

Advantages of Immutability

Immutability makes programs less complicated

Immutable objects are thread-safe

Collections such as HashMap and HashSet expect immutability



Immutable objects can be safely be shared and reused

```java
// Inefficient code!
String str = "";
for (int i = 0; i < 100; i++) {
    str = str + "hello ";
}
```

```
new String: "hello "
new String: "hello hello "
new String: "hello hello hello "
new String: "hello hello hello hello "
new String: "hello hello hello hello hello "
...
```

Disadvantages of Immutability

Example: String concatenation in a loop

```
// Use StringBuilder instead
StringBuilder builder = new StringBuilder();
for (int i = 0; i < 100; i++) {
    builder.append("hello ");
}
String str = builder.toString();
```

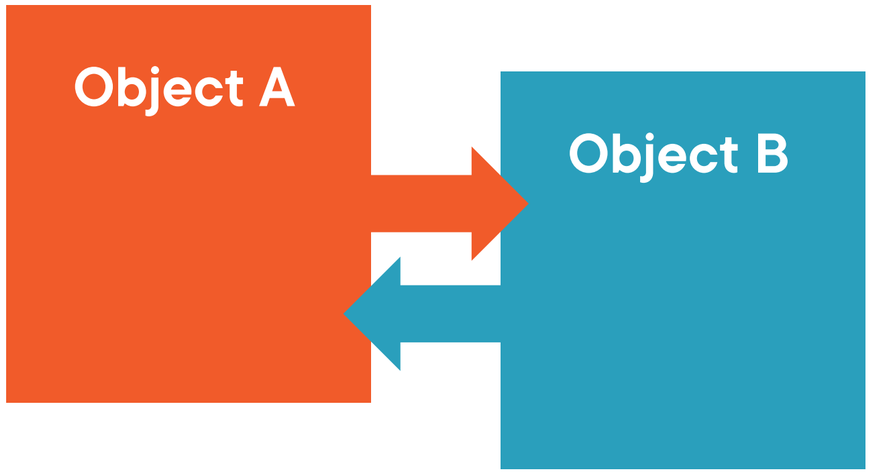
Disadvantages of Immutability

Example: String concatenation in a loop

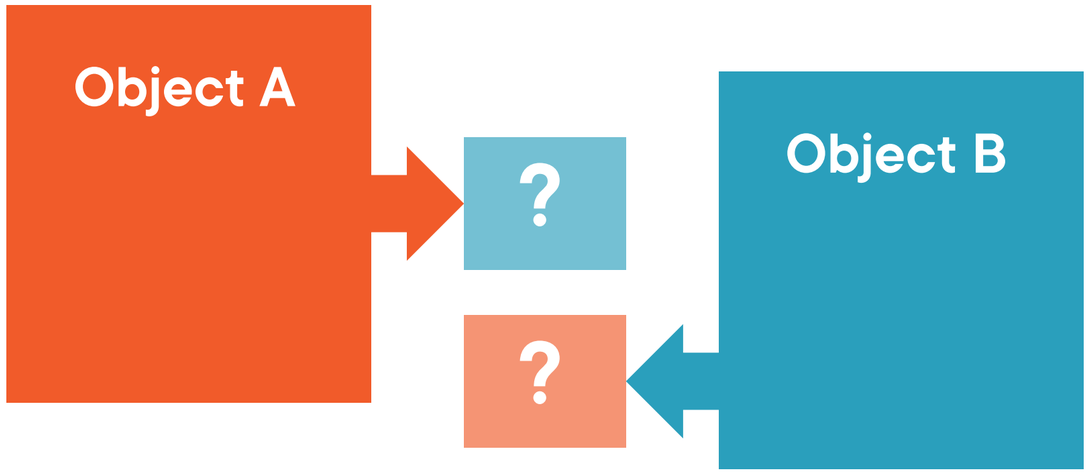
Disadvantages of Immutability

Circular references

Mutable objects



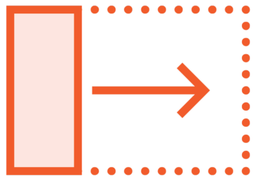
Immutable objects



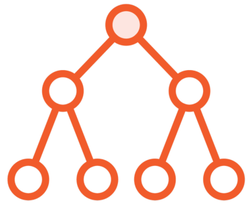
The Class Hierarchy of Records

The Class Hierarchy of Records

Records are implicitly final



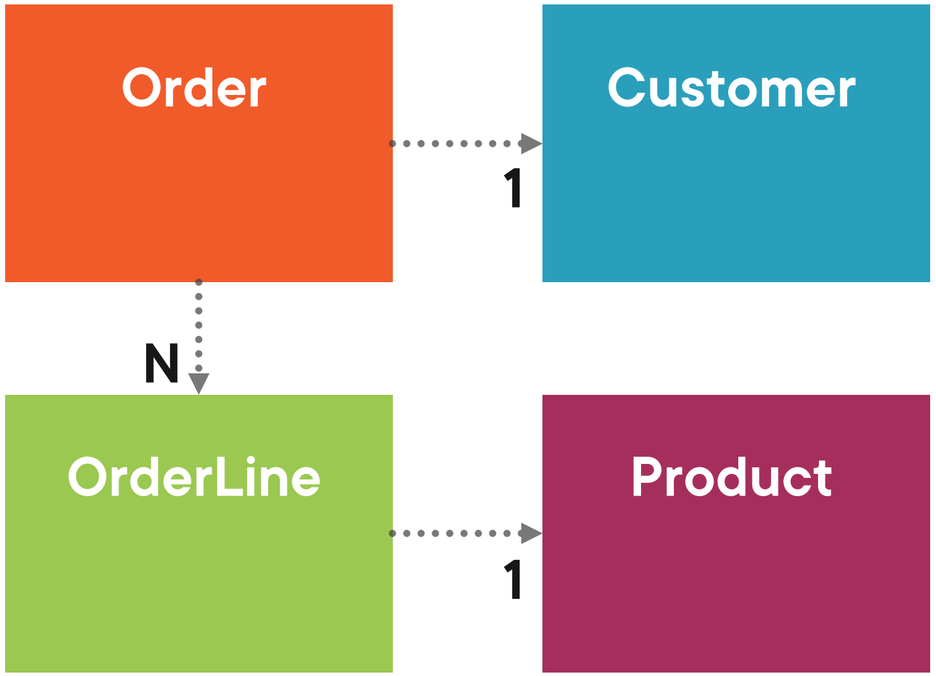
Records cannot extend classes, but can implement interfaces



The common superclass for records is java.lang.Record

Practical Use Cases for Records

Domain Objects



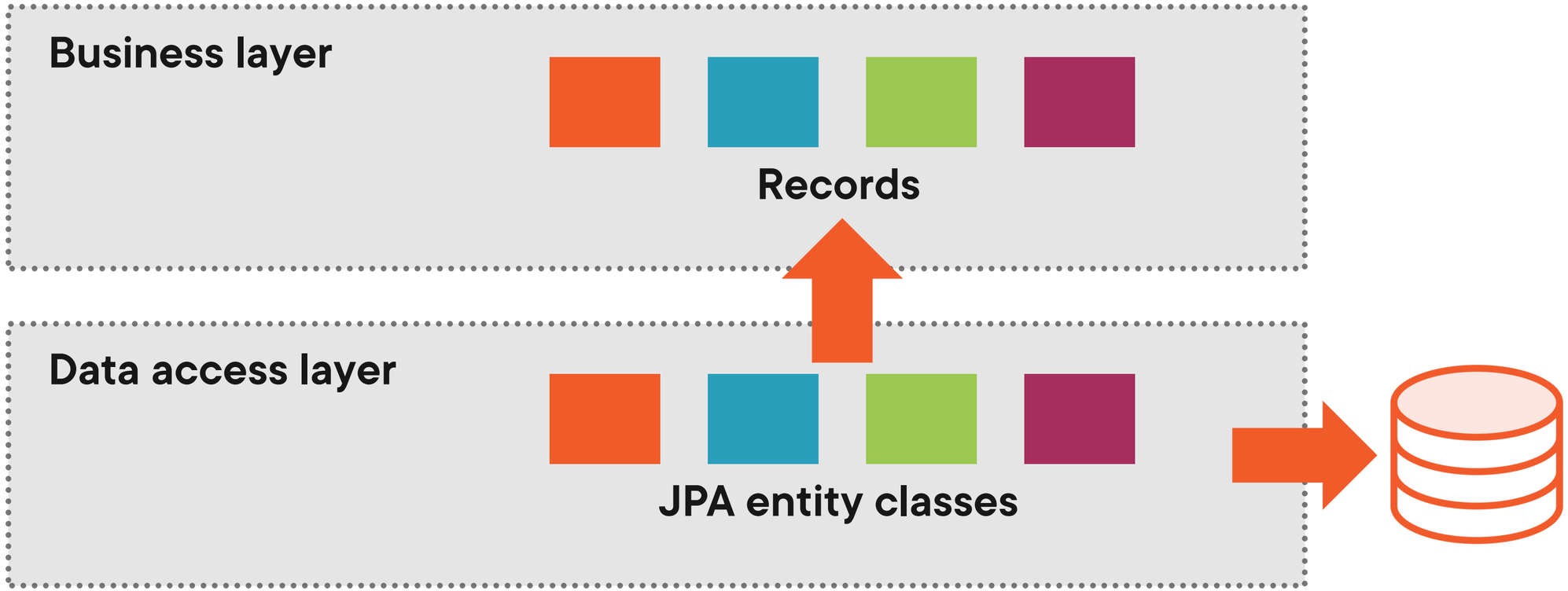
Domain model

Database

JPA (Java Persistence API)

Records cannot be used for JPA entities

Domain Objects – Layered Architecture



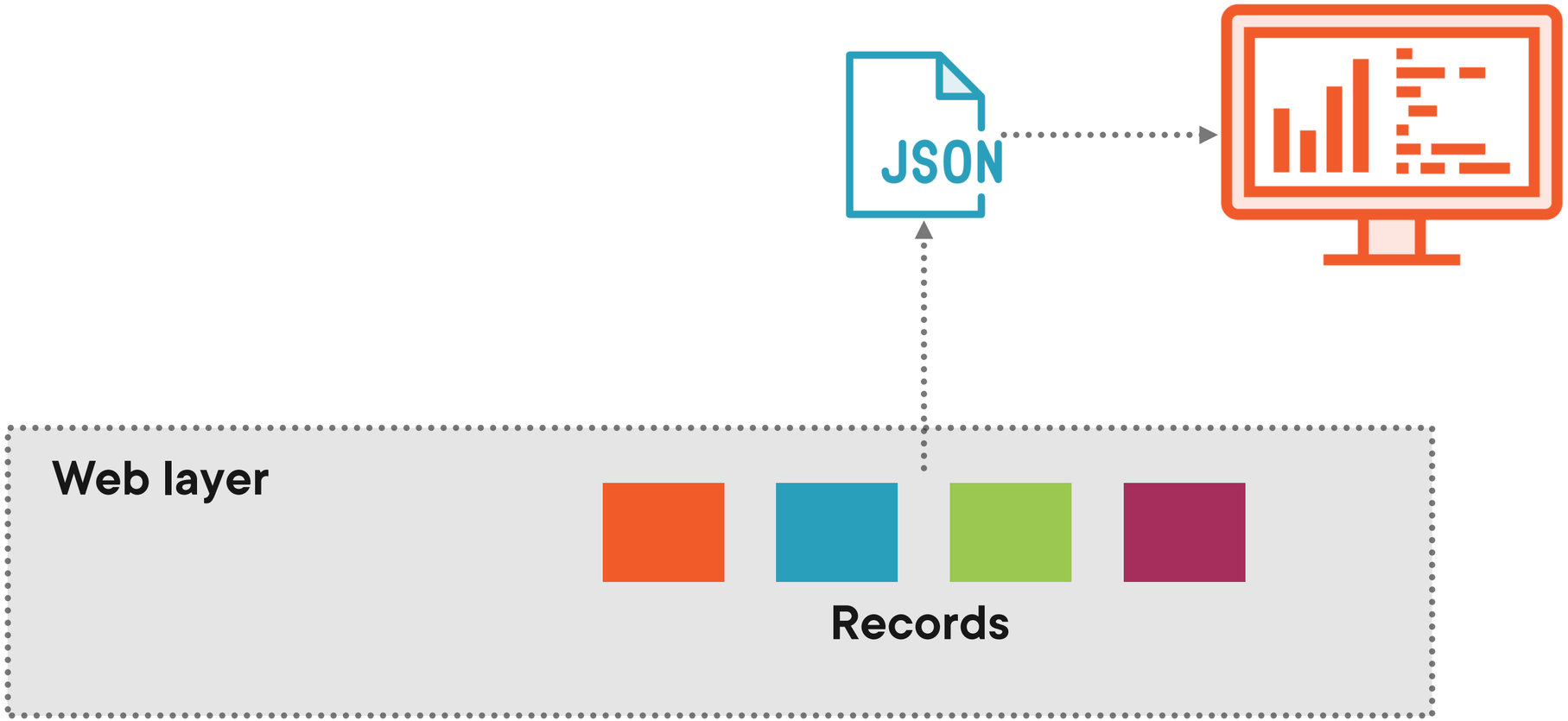
Value Objects

Examples

- String
- Integer, Long, ...
- BigInteger
  BigDecimal
- java.time

A value object does not have an identity

Data Transfer Objects



Non-Use Cases for Records

JavaBeans

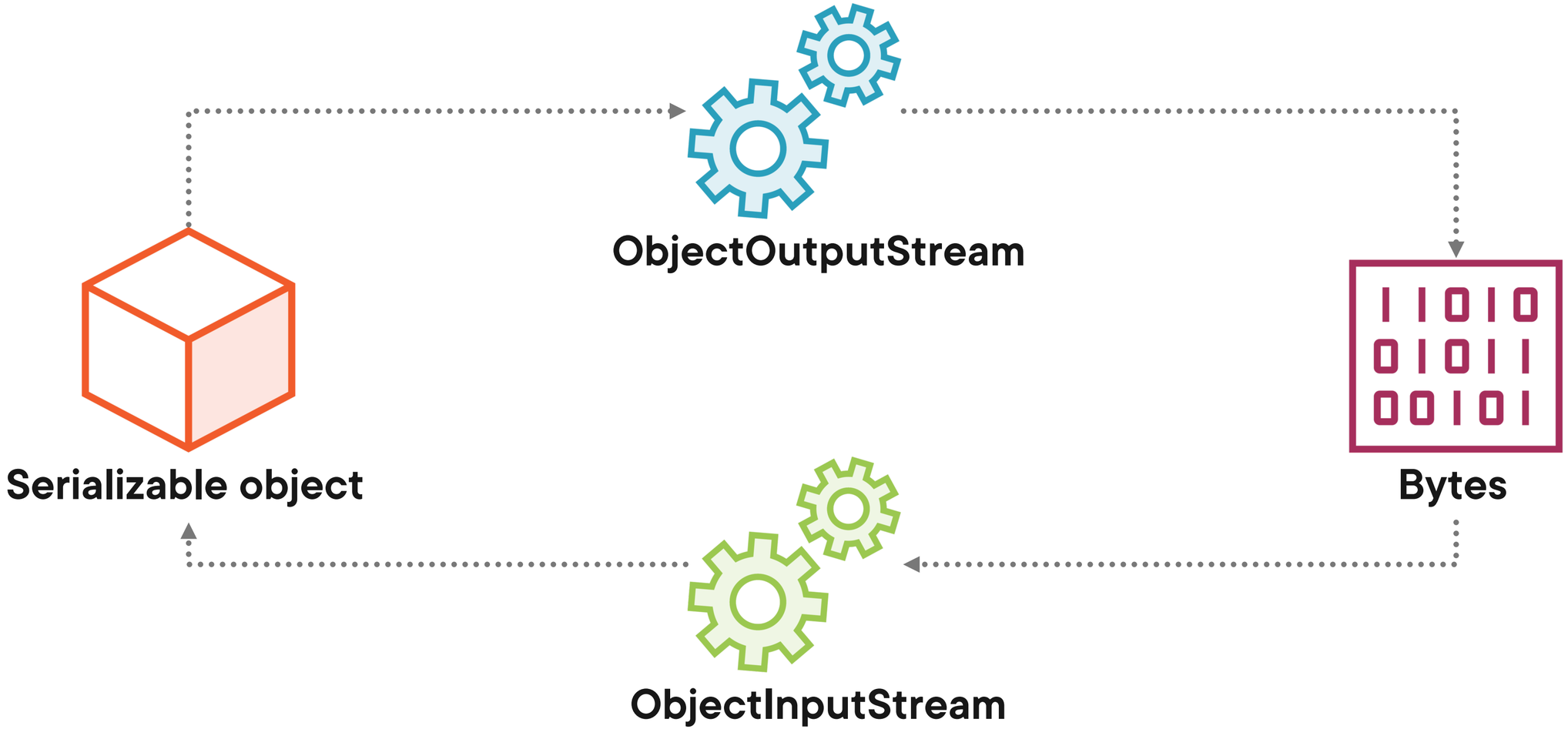
Records are not a direct replacement for JavaBeans

Singletons

The canonical constructor cannot be made private

Customizing Serialization of Records

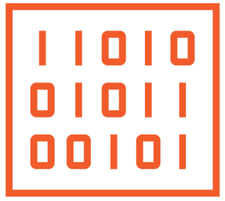
Serialization



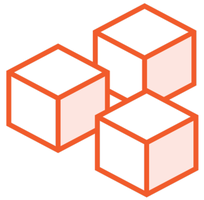
Records and Serialization

When a record is deserialized, its canonical constructor is called

writeObject() and readObject() do *not* work for records



Interface Externalizable does *not* work for records



writeReplace() and readResolve() *do* work for records

Summary

Immutability

Records

- Components of a record
- Accessor methods
- Canonical constructor
- equals, hashCode, toString
- Compact constructor
- Additional constructors
- java.lang.Record

Summary

Use cases

- Domain objects
- Value objects
- Data transfer objects
- Not for: JPA entities
- No direct replacement for JavaBeans
- Not for: Singletons

Builder pattern

“Wither” methods

Up Next: Sealed Classes and Interfaces