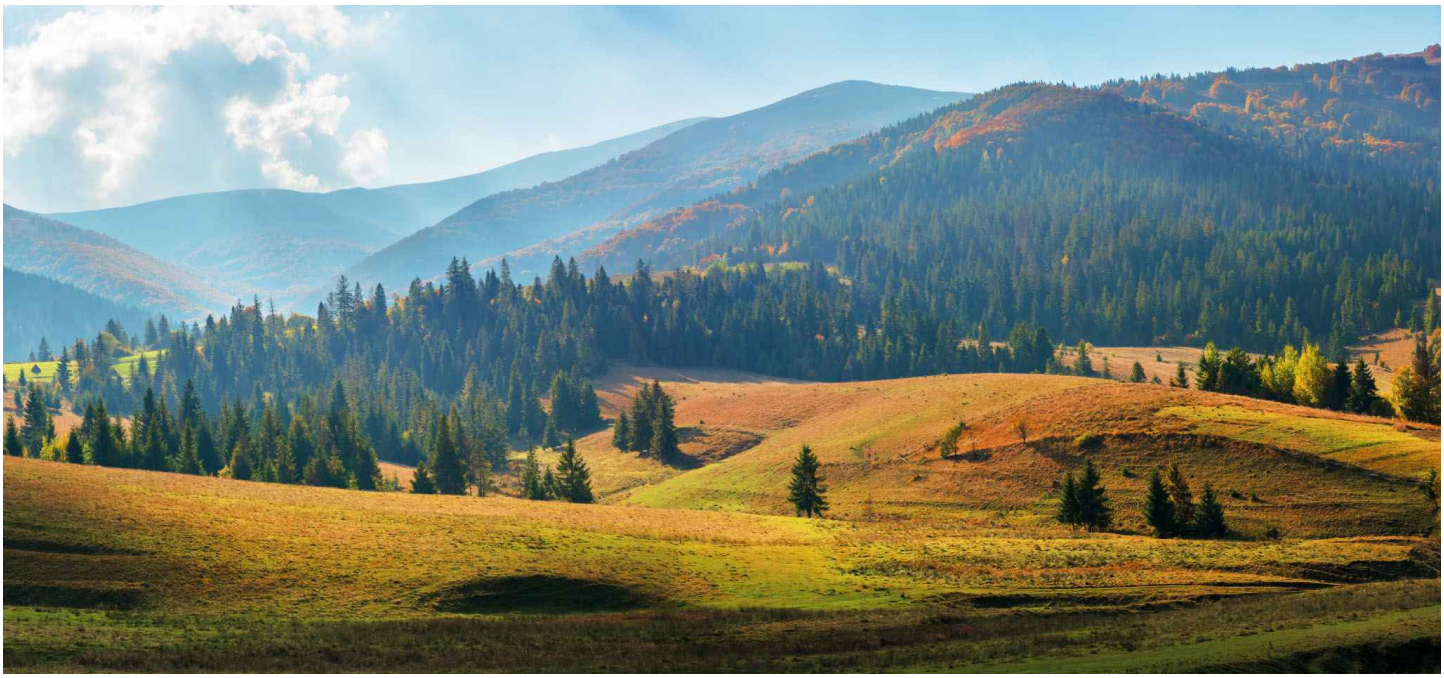

37-Java 16: Records

Author: Vincent Lau

Learning Objectives

> Understand the way to implement record in Java
>
> Understand the exact meaning of record when you created a record

Introduction

A `record` is a new type of class introduced in Java 14 that is specifically designed to model data and became a standard feature starting from Java 16. It provides a concise way to declare classes for holding immutable data, automatically generating useful methods like constructors, getters, equals, hashCode, and toString. Records can significantly reduce boilerplate code when dealing with data-centric classes. Here are some examples to help beginners understand records:

Purposes

Commonly, we write classes to simply hold data, such as database results, query results, or information from a service. In many cases, this data is immutable, since **immutability ensures the validity of the data without synchronization**.

While this accomplishes our goal, there are two problems with it:

1. There's a lot of boilerplate code
2. We obscure the purpose of our class: to represent a person with a `name` and `age`

Creating a Basic Record

- All-args constructor
- Getters

```java
public record Person(String name, int age) {}

public class RecordExample {
    public static void main(String[] args) {
        Person person = new Person("Alice", 30);
        System.out.println(person.name()); // Accessing the property using getter
        System.out.println(person.age());  // Accessing the property using getter
    }
}
```

equals() and hashCode()

```java
public record Point(int x, int y) {}

public class RecordEqualsExample {
    public static void main(String[] args) {
        Point p1 = new Point(1, 2);
        Point p2 = new Point(1, 2);

        System.out.println(p1.equals(p2)); // true, automatically generated equals method
        System.out.println(p1.hashCode() == p2.hashCode()); // true, automatically generated hashCode
    }
}
```

toString() Method

```java
public record Book(String title, String author) {}

public class RecordToStringExample {
    public static void main(String[] args) {
        Book book = new Book("Java Fundamentals", "John Doe");
        System.out.println(book); // Automatically generated toString method
    }
}
```

Add logic to All-args Constructor

```java
public record Color(int red, int green, int blue) {
    public Color {
        if (red < 0 || red > 255 || green < 0 || green > 255 || blue < 0 || blue > 255) {
            throw new IllegalArgumentException("Invalid color component");
        }
    }
}

public class RecordConstructorExample {
    public static void main(String[] args) {
        Color color = new Color(100, 150, 200); // Using the custom constructor
        System.out.println(color);
    }
}
```

Custom Instance Method

```java
public record Person(String name, int age) {

  public boolean isElderly(int age) {
    return this.age > 65;
  }
}

```

Custom Static Variable & Method

```
public record Person(String name, int age) {
  public static String UNKNOWN_ADDRESS = "Unknown";

  public static int add(int x, int y) {
    return x + y;
  }
}

```

What Record doesn't have

- Only one constructor, which is an all-args constructor. No other constructor.
- No setter (Immutable)

Summary

Records simplify the creation and management of data classes, making code more concise and readable. They are especially useful when dealing with data objects that require only immutability and basic behavior.