JAVA RECORDS

Efthimios Alepis

CONTENTS

SUMMARY

- Enhance the Java programming language with records, which are classes that act as transparent carriers for immutable data.
- Records can be thought of as nominal tuples.

PRIMARY GOALS

- Devise an object-oriented construct that expresses a simple aggregation of values.
- Help developers to focus on modeling immutable data rather than extensible behavior.
- Automatically implement data-driven methods such as equals and accessors.
- Preserve long-standing Java principles such as nominal typing and migration compatibility.

DESCRIPTION

- A record class declaration consists of a name; optional type parameters (generic record declarations are supported); a header, which lists the "components" of the record; and a body.
- A record class declares the following members automatically:
    - For each component in the header, the following two members:
        - A private final field with the same name and declared type as the record component. This field is sometimes referred to as a component field.
        - A public accessor method with the same name and type of the component; in the Rectangle record class example, these methods are Rectangle::length() and Rectangle::width().
    - A canonical constructor whose signature is the same as the header. This constructor assigns each argument from the new expression that instantiates the record class to the corresponding component field.
    - Implementations of the equals and hashCode methods, which specify that two record classes are equal if they are of the same type and contain equal component values.
    - An implementation of the toString method that includes the string representation of all the record class's components, with their names.
    - As record classes are just special kinds of classes, you create a record object (an instance of a record class) with the new keyword

REMARKS

- You can explicitly declare any of the members derived from the header, such as the public accessor methods that correspond to the record class's components
- If you implement your own accessor methods, then ensure that they have the same characteristics as implicitly derived accessors
- If you implement your own versions of the equals, hashCode, and toString methods, then ensure that they have the same characteristics and behavior as those in the java.lang.Record class, which is the common superclass of all record classes
- You can declare static fields, static initializers, and static methods in a record class, and they behave as they would in a normal class
- You cannot declare instance variables (non-static fields) or instance initializers in a record class
- You can declare instance methods in a record class, independent of whether you implement your own accessor methods. You can also declare nested classes and interfaces in a record class, including nested record classes (which are implicitly static)

REMARKS CONTINUED

- You can create a generic record class

```
record Triangle<C extends Coordinate> (C top, C left, C right) { }
```

- You can declare a record class that implements one or more interfaces

```
record Customer(...) implements Billable { }
```

- You can annotate a record class and its individual components

```
record Rectangle(
    @GreaterThanZero double length,
    @GreaterThanZero double width) { }
```

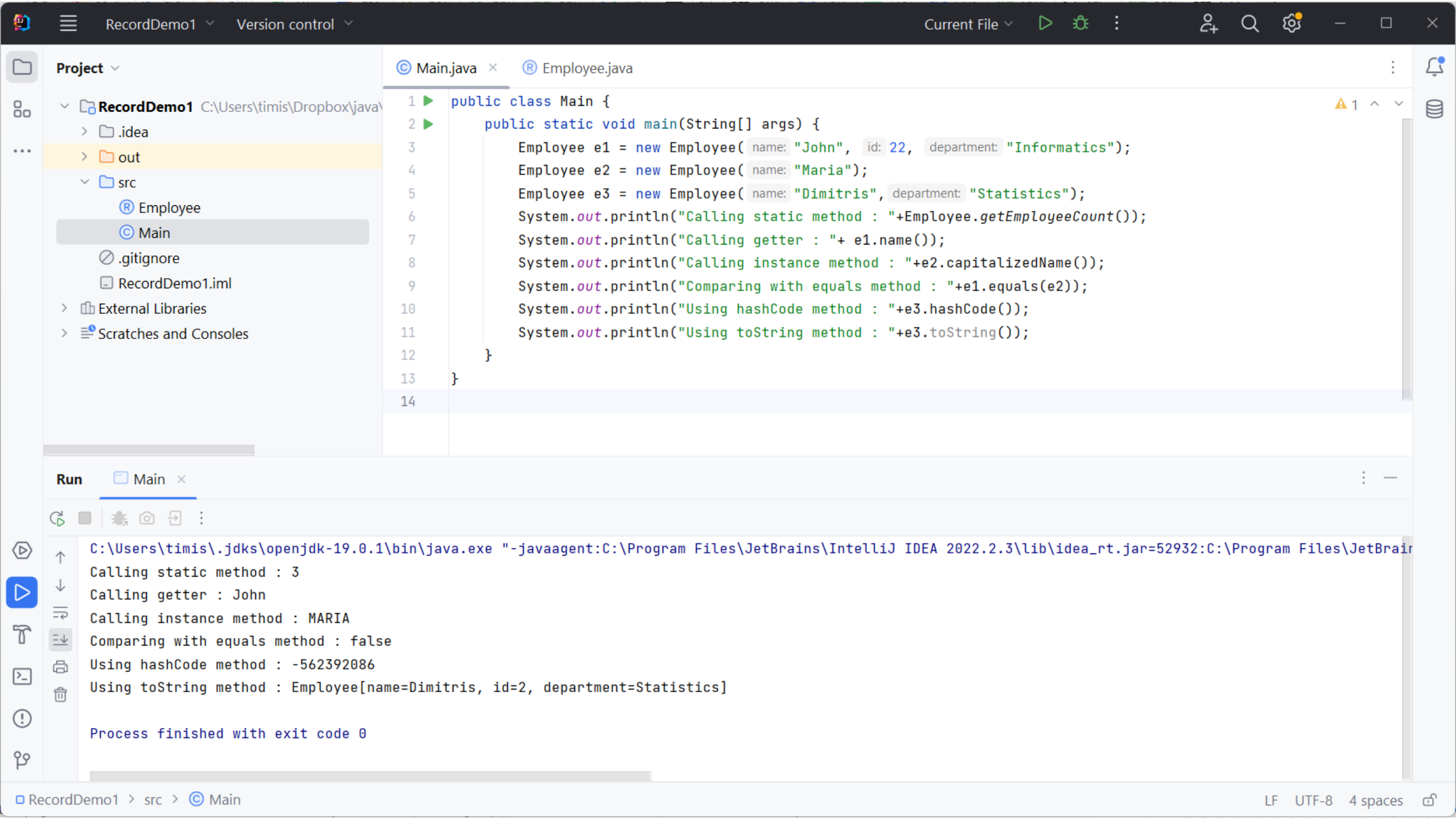
RECORD VS CLASS


```
record Rectangle(double length, double width) { }
```

```
public final class Rectangle {
    private final double length;
    private final double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    double length() { return this.length; }
    double width()  { return this.width; }

    // Implementation of equals() and hashCode(), which specify
    // that two record objects are equal if they
    // are of the same type and contain equal field values.
    public boolean equals...
    public int hashCode...

    // An implementation of toString() that returns a string
    // representation of all the record class's fields,
    // including their names.
    public String toString() {...}
}
```

EXAMPLE

```
public record Employee(String name, int id,String department) {
    3 usages
    static int eCount = 1;
    no usages
    public Employee(String name) {
        this(name,eCount++,  department: "Accounting");
    }
    no usages
    public Employee(String name, String department) {
        this(name,eCount++,department);
    }
    no usages
    String capitalizedName() {
        return this.name.toUpperCase();
    }
    no usages
    static int getEmployeeCount() {
        return eCount;
    }
}
```


RecordDemo1
Version control
Current File
Project
RecordDemo1 C:\Users\timis\Dropbox\java\
.idea
out
src
Employee
Main
.gitignore
RecordDemo1.iml
External Libraries
Scratches and Consoles
Main.java
Employee.java
1 public class Main {
2 public static void main(String[] args) {
3 Employee e1 = new Employee(name: "John", id: 22, department: "Informatics");
4 Employee e2 = new Employee(name: "Maria");
5 Employee e3 = new Employee(name: "Dimitris", department: "Statistics");
6 System.out.println("Calling static method : "+Employee.getEmployeeCount());
7 System.out.println("Calling getter : "+ e1.name());
8 System.out.println("Calling instance method : "+e2.capitalizedName());
9 System.out.println("Comparing with equals method : "+e1.equals(e2));
10 System.out.println("Using hashCode method : "+e3.hashCode());
11 System.out.println("Using toString method : "+e3.toString());
12 }
13 }
14
Run
Main
C:\Users\timis\.jdks\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2.3\lib\idea_rt.jar=52932:C:\Program Files\JetBrain
Calling static method : 3
Calling getter : John
Calling instance method : MARIA
Comparing with equals method : false
Using hashCode method : -562392086
Using toString method : Employee[name=Dimitris, id=2, department=Statistics]
Process finished with exit code 0
RecordDemo1 > src > Main
LF
UTF-8
4 spaces

CONCLUSION

Using records with their compiler-generated methods, we can reduce boilerplate code and improve the reliability of our immutable classes.

FURTHER READING

https://openjdk.org/jeps/395

https://docs.oracle.com/en/java/javase/20/language/records.html