Java Records

John M. Morrison

April 4, 2022

Contents

0 Introduction **1**

1 A Point Class **1**

2 Can I have other methods? **3**

3 Indecent Exposure **5**

4 The Compact Constructor **6**

0 Introduction

Java records, a new feature in Java 17 allows you to create "data classes" that avoid a lot of the usual boilerplate code such as getter, `toString()`, `hashCode()`, and `equals()` methods.

1 A Point Class

For example, consider the problem of producing a class for points in the plane with integer coördiantes. Using records, we proceed as follows.

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

public class Driver
{
    public static void main(String[] args)
    {
        Point p = new Point(3,4);
        System.out.println(p);
    }
}
```

Run this class and you will see this.

```
unix> java Driver
Point[x=3, y=4]
```

Notice we get a `toString()` method for free. You can, if you wish, override this method, but this string representation is quite serviceable. We now show getters at work.

```
public class Driver
{
    public static void main(String[] args)
    {
        Point p = new Point(3,4);
        System.out.println(p);
        System.out.println(p.x());
        System.out.println(p.y());
    }
}
```

```
unix> java Driver
Point[x=3, y=4]
3
4
```

Can malfeasing and misfeasing client programmers pervert our objects? Let's find out.

```
public class Driver
{
    public static void main(String[] args)
    {
        Point p = new Point(3,4);
        System.out.println(p);
        System.out.println(p.x());
        System.out.println(p.y());
        p.x = 5;
    }
}
```

Now we compile.

```
javac Driver.java
Driver.java:9: error: x has private access in Point
        p.x = 5;
         ^
1 error
```

We are rebuffed for our efforts. The state variables you create are `final`. If they of primitive or immutable type, then they are, in fact, constant. In these cases, records produce immutable objects.

Now for a little equality.

```
public class Driver
{
    public static void main(String[] args)
    {
        Point p = new Point(3,4);
        System.out.println(p);
        System.out.println(p.x());
        System.out.println(p.y());
        //p.x = 5;   Illegal
        Point q = new Point(3,4);
        System.out.println(p.equals(q));
    }
}
```

```
unix> java Driver
Point[x=3, y=4]
3
4
true
```

You get an `equals` method for free!

2 Can I have other methods?

Let us attempt to insert a second constructor.

```
public record Point(int x, int y)
{
    public Point()
    {
        this(0,0);
    }
}
```

Then we create an instance using it.

```
public class Driver
{
    public static void main(String[] args)
    {
        Point p = new Point(3,4);
        System.out.println(p);
        System.out.println(p.x());
        System.out.println(p.y());
        //p.x = 5;   Illegal
        Point q = new Point(3,4);
        System.out.println(p.equals(q));
        Point r = new Point();
        System.out.println(r);
    }
}
```

Now let's compile and attempt to run.

```
unix> java Driver
Point[x=3, y=4]
3
4
true
Point[x=0, y=0]
```

Et voila! it works! Can we have other methods? Let us test that hypothesis by doing just that.

```
public record Point(int x, int y)
{
    public Point()
    {
        this(0,0);
    }
    public double distanceTo(Point other)
    {
        return Math.hypot(x - other.x, y - other.y);
    }
}
```

```
public class Driver
{
    public static void main(String[] args)
    {
        Point p = new Point(3,4);
        System.out.println(p);
        System.out.println(p.x());
        System.out.println(p.y());
        //p.x = 5;   Illegal
        Point q = new Point(3,4);
        System.out.println(p.equals(q));
        Point r = new Point();
        System.out.println(r);
        System.out.println(p.distanceTo(r));
    }
}
```

Now compile and run.

```
unix> javac *.java
unix> java Driver
Point[x=3, y=4]
3
4
true
Point[x=0, y=0]
5.0
```

3 Indecent Exposure

Java records are immutable if all state is immutable or primitive. If you expose mutable state, you can be in for an ugly surprise. Consider this sad little record `UhOh.java`.

```
import java.util.List;
public record UhOh(String className, List<String> roster){}
```

We now take it for a test-drive in the class `Intruder.java`

```
import java.util.List;
import java.util.ArrayList;
public class Intruder
{
    public static void main(String[] args)
    {
        List<String> students = new ArrayList<>();
        students.add("key23a");
        students.add("doki22a");
        students.add("warner22p");
        UhOh mistake = new UhOh("CS4280", students);
        System.out.println(mistake);
        mistake.roster().add("obi22o");
        System.out.println(mistake);
    }
}
```

Run this code.

```
java Intruder
UhOh[className=CS4280, roster=[key23a, doki22a, warner22p]]
UhOh[className=CS4280, roster=[key23a, doki22a, warner22p, obi22o]]
```

Uh oh. Be so cautioned. Exposing mutable state in a record breaks immutability.

4 The Compact Constructor

The automatically built constructor that lies in every record is called the *canonical constructor*. You also have the option of implementing a compact constructor. This constructor runs *before* any other constructor. One use case is for throwing exceptions when illegal instances are being created. Here we create a simple example.

First we create a enum in the file `TransactionType.java`.

```
public enum TransactionType
{
    DEPOSIT,
    WITHDRAWL
}
```

Then here is our class for transactions.

```
public record Transaction(TransactionType type, int amount)
{
    public Transaction
    {
        if(amount < 0)
        {
            throw new IllegalArgumentException();
        }
    }
}
```

Notice the absence of parentheses after the header `public Transaction`.