When we are working with code, we are creating new objects. There are many ways to create objects; the most used is the class constructor. But, do we initialize our object properly in the best possible way?

Let's consider that we own a company that produces headphones. We want to model our domain and processes in code. In most cases, our headphones are black and cost $150, so we just write a class and provide a default constructor, as demonstrated below:

1

public class Headphones {

2

​

3

private final Color color;

4

private final BigDecimal priceInDollars;

5

​

6

public Headphones() {

7

this.color = Color.BLACK;

8

this.priceInDollars = BigDecimal.valueOf(150.0);

9

}

10

​

11

//other methods

12

​

13

}

Great success! Our company is making plenty of money from our $150 black headphones. After some time, we decide that it is time to create headphones in different colors. We can still create only black headphones and send them to a painter, but it is not efficient to cover black headphones with another color. Another reason that we don't want to do that is that it will cost additional money.

We are mainly creating black headphones, but, for this special case, we will create a new constructor. We have decided to set the price to a default value, but the color will be provided as shown below:

1

public Headphones(final Color color) {

2

this.color = color;

3

this.priceInDollars = BigDecimal.valueOf(150.0);

4

}

Again, after some time, we decide that different color headphones can have a different price. And, again, we are creating a new constructor. Our class will now look like this:

1

public class Headphones {

2

​

3

private final Color color;

4

private final BigDecimal priceInDollars;

5

​

6

public Headphones() {

7

this.color = Color.BLACK;

8

this.priceInDollars = BigDecimal.valueOf(150.0);

9

}

10

​

11

public Headphones(final Color color) {

12

this.color = color;

13

this.priceInDollars = BigDecimal.valueOf(150.0);

14

}

15

​

16

public Headphones(final Color color, final BigDecimal priceInDollars) {

17

this.color = color;

18

this.priceInDollars = priceInDollars;

19

}

20

​

21

//other methods

22

​

23

}

As we can see, this leads us to code duplication. Each constructor has to set all fields by themselves, and if we want to change the default value, we have to track all constructors. What can we do with that? The first solution that came to mind is that we can inline our fields with default values:

1

public class Headphones {

2

​

3

private Color color = Color.BLACK;

4

private BigDecimal priceInDollars = BigDecimal.valueOf(150.0);

5

​

6

public Headphones() {

7

}

8

​

9

public Headphones(final Color color) {

10

this.color = color;

11

}

12

​

13

public Headphones(final Color color, final BigDecimal priceInDollars) {

14

this.color = color;

15

this.priceInDollars = priceInDollars;

16

}

17

​

18

//other methods

19

​

20

}

Now, our constructors are only setting the fields that they want to change. We can also do it in another way and set default values using the initializer:

1

public class Headphones {

2

​

3

private Color color;

4

private BigDecimal priceInDollars;

5

​

6

{

7

this.color = Color.BLACK;

8

this.priceInDollars = BigDecimal.valueOf(150.0);

9

}

10

​

11

//other

12

​

13

}

These three ways of setting object fields in a moment of construction can, of course, be mixed. You can set one field by a constructor, the second field by inlining a value, and the third field set in an initializer, but what are the benefits of that?

Well, I don't see any.

What I notice here is that there are plenty of problems:

* We don't have one single point of creation — if we want to provide a behavior, which all our constructors will share, we will copy and paste it from one to another.
* We are making our class more complicated than it should be. When we are checking the class constructor, but we don't want to jump between too many places to check how an object is created.
* After transformation, our fields cannot be final. We are setting default values regardless of whether we change them or not, so we have to make the possibility of changing it
* Unnecessary field initialization is creating an unused object. This is not really a problem until you are creating millions of objects

But, the big question is: what can we do to get rid of these problems?

The answer to that is a concept called **constructor chaining**.

Constructor chaining means that we are writing our constructors in a way that has them calling other constructors inside a class, finally reaching the single top constructor. This top constructor is called the primary constructor, and all others are called secondary.

The primary constructor will be our single point of creation and initialization, so we can put all common behavior into it. To keep consistency, we have only one primary constructor; ideally, this constructor will set all our class fields. Also, it is important to remember that the primary constructor doesn't have to be public, if we don't want to expose primary constructor outside our class or package.

Constructor chaining is a solution to all the problems I mentioned previously with constructors in Java. With that, you don't need to worry about code duplication. As a result, it will make your class less complicated and more maintainable. All constructors are in the end calling the primary constructor, which is a single point of creation that can share common code.

You only have to check constructors to see the initialization of the code — you don't need to search the whole class. Along with that, your final fields can, in fact, be final. You don't create unnecessary objects.

I recommend that you write the primary constructor as the last constructor in a class and all secondaries before it, as I did in the following Headphones class:

1

public class Headphones {

2

​

3

private final Color color;

4

private final BigDecimal priceInDollars;

5

​

6

public Headphones() {

7

this(Color.BLACK);

8

}

9

​

10

public Headphones(final Color color) {

11

this(color, BigDecimal.valueOf(150.0));

12

}

13

​

14

public Headphones(final Color color, final BigDecimal priceInDollars) {

15

this.color = color;

16

this.priceInDollars = priceInDollars;

17

}

18

​

19

//other methods

20

​

21

}

One important point to keep in mind is to never use initializer blocks and never initialize fields in the inline — always chain your constructors.

Let me explain. When you are creating a new class, always think about how to design it properly. Exposing many constructors will provide different ways of creation, which leads to an increase in the usability of your code.

Designing a class means not only providing a nice API, which will be easy to use, but it also means keeping all code inside maintainable and readable for developers. This will work well with that class in the future. Using constructor chaining with primary and secondary constructors will provide the perfect solution to this problem. Happy chaining!