Immutable Object

Michal Moravik, SD20w2

Name: Immutable object

Category: ...

Intent: Creating an object that cannot be changed later on.

Motivation: Since they cannot be changed later, it is safe to have an immutable object in your program. This way we can avoid situations like for example concurrency problem. By having immutable objects, we can ensure that all threads looking at the same object see the same state of the object.

Steps:

1. Make your class final
2. Make all properties private and final
3. Do not add setters (redundant)

Implementation:

```java
public final class ImmutableClassAuthor {
    final private String[] publishedBooks;
    final private int age;

    public ImmutableClassAuthor(String[] publishedBooks, int age) {
        this.publishedBooks = publishedBooks;
        this.age = age;
    }

    public String[] getPublishedBooks() {
        return publishedBooks;
    }

    public int getAge() {
        return age;
    }
}
```

Consequences:

Pros:

- Protects the object from being changed, therefore, protects the program from possible errors
- Thread-safe

Cons:

- If you want to have an object with different values, you need to remove the object or add a new one

Known uses:

- Should be added every time when we do not expect any changes in the object.

Related patterns:

- ...