Scenario 2.1 (Builder Design Pattern)

Definition

One of the key benefits of utilizing the builder design pattern is that it allows you to change the internal representation of the product, encapsulates the code for building and representation, and gives you control over the construction phases.

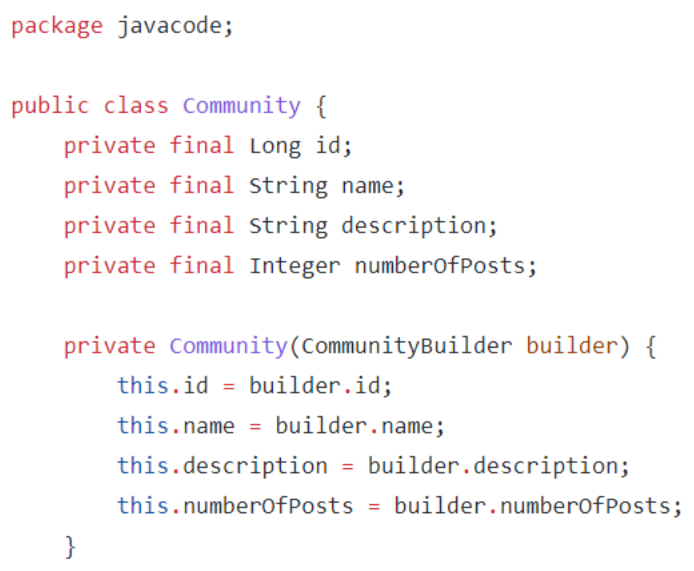
The construction of a complex object should be separated from its representation in order to achieve multiple representations within the same process.

How the builder design pattern is needed

Let's talk about the scenario in our application. Let's imagine the major entity in our application is Community. We will not wish to change the state of a Community object once it has been fully created, both in theory and in practice. We need to develop an immutable Community Class because we will be creating numerous instances of Community.

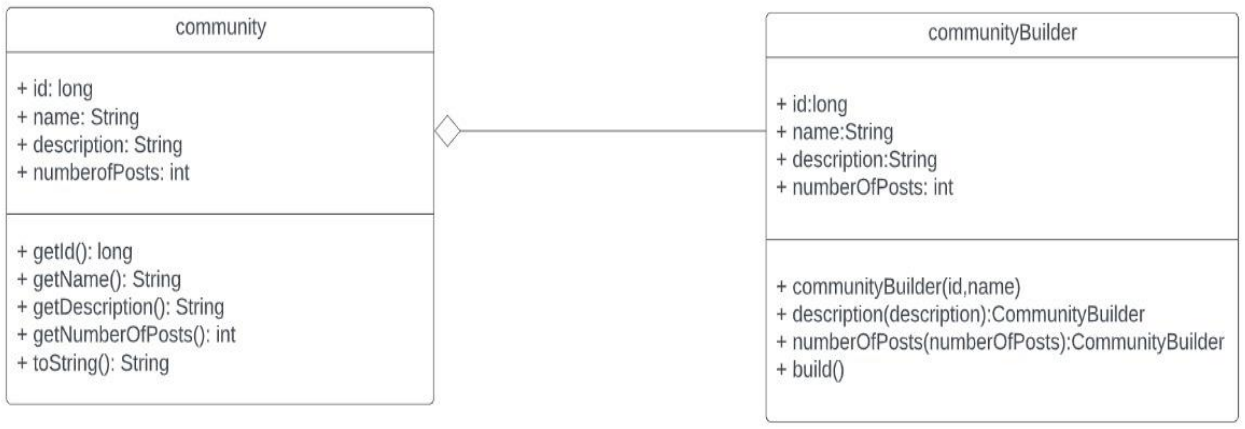
The Community Class has several properties to be constructed, but not all of them must be passed when the Community is established; some of the fields are optional. To satisfy variation Community formation situations with non-mandatory fields, we need to build many Communities with distinct scenarios and variant attributes. In most cases, if we wish to create an immutable Community class, we must supply all information to the constructor as inputs. However, if one or more properties are introduced later, we will require extra constructors because we do not want to change the entire present implementation.

The builder approach allows you to consume more attributes while maintaining the Community class's immutability.

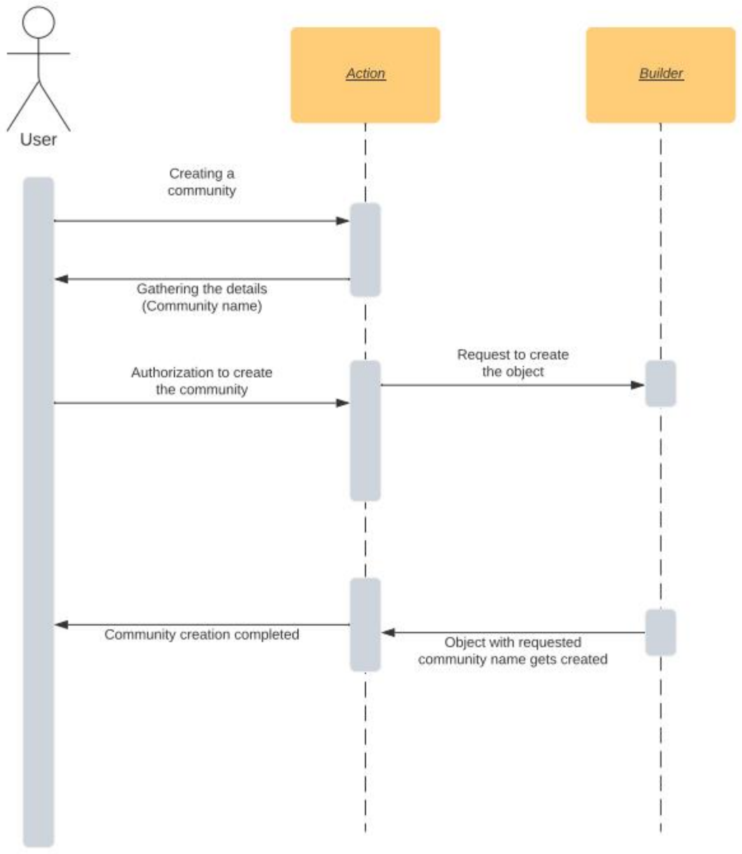


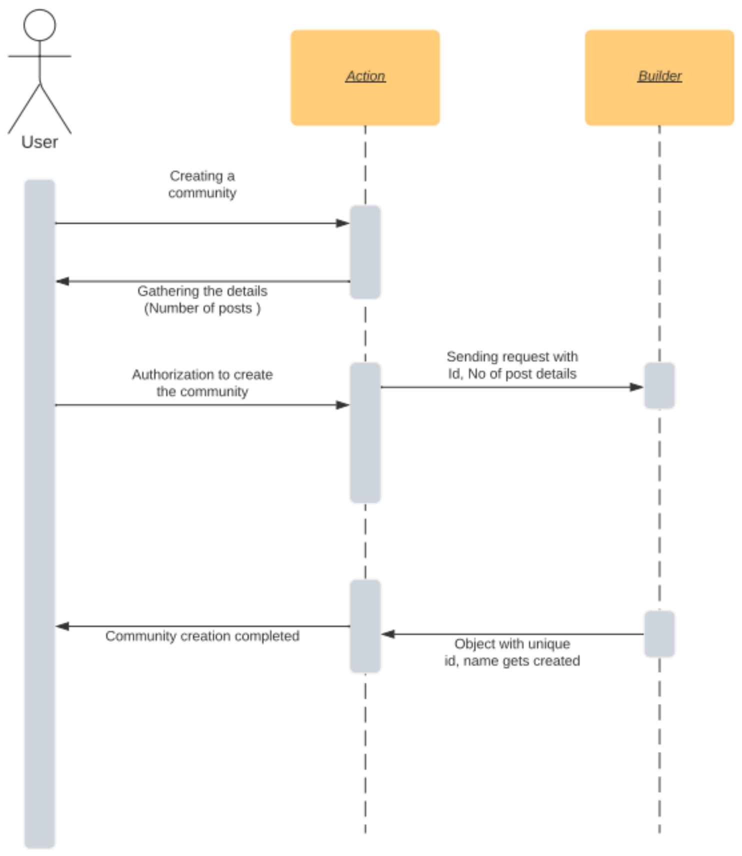
```java
package javacode;

public class Community {
    private final Long id;
    private final String name;
    private final String description;
    private final Integer numberOfPosts;

    private Community(CommunityBuilder builder) {
        this.id = builder.id;
        this.name = builder.name;
        this.description = builder.description;
        this.numberOfPosts = builder.numberOfPosts;
    }
```

Class Diagram for Builder Design Pattern



Sequence Diagram for Builder


User
Action
Builder
Creating a community
Gathering the details (Community name)
Authorization to create the community
Request to create the object
Community creation completed
Object with requested community name gets created


User
Action
Builder
Creating a community
Gathering the details (Number of posts)
Sending request with Id, No of post details
Authorization to create the community
Community creation completed
Object with unique id, name gets created

Builder Pattern Implementation

Below solution delineates an additional class CommunityBuilder which helps us in building desired Community instance with all mandatory attributes and a combination of optional attributes, without losing the immutability.

Because the Community object produced below lacks a setter method, its state cannot be altered once it has been constructed. This ensures that the requisite immutability is achieved.

```
public static class CommunityBuilder {
    private final Long id;
    private final String name;
    private String description;
    private Integer numberOfPosts;

    public CommunityBuilder(Long id, String name) {
        this.id = id;
        this.name = name;
    }
    public CommunityBuilder description(String description){
        this.description=description;
        return this;
    }
    public CommunityBuilder numberOfPosts(Integer numberOfPosts){
        this.numberOfPosts=numberOfPosts;
        return this;
    }
    public Community build() {
        Community community =  new Community(this);
        return community;
    }
}
```