Twitter Thread by Vikas Rajput

Vikas Rajput
@vikasrajputin

Java: Why do we use getter and setter methods?

a thread...

In Java classes, we normally create the getter and setter methods to read and update class level fields respectively.

Let's find out why do we following this practice.

Consider a class "Account", having fields like accountName and accountBalance - to show the name and balance of the account.

As a common practice, both the variables are private and define the public getter and setter method to read and write their values.

Eg:

Account.java

```java
public class Account {
    private String accountName;
    private int accountBalance;

    public String getAccountName() {
        return accountName;
    }
    public void setAccountName(String accountName) {
        this.accountName = accountName;
    }
    public int getAccountBalance() {
        return accountBalance;
    }
    public void setAccountBalance(int accountBalance) {
        this.accountBalance = accountBalance;
    }
}
```

Using the above example, let's see different use-cases where having getter and setter methods can be game-changing.

1. Validation:

The public getter and setter method act as a single door to access the private fields.

Before updating the value we can run any validation in the setter method and accordingly allow field modification.

Eg:

2. Security

Similar to Validation, we can also put any security-related code to secure our data inside the getter and setter.

For eg. Check if a user has access to the field based on our complex security logic and then allow the user to either read or update the value.

3. ReadOnly or WriteOnly Permission

To allow only write permission, we can keep setter methods.

Similarly, to allow only read permission to fields, we can remove the setter method and only keep the getter method as shown below:

Account.java

```java
public class Account {
    private String accountName;
    private int accountBalance;

    public String getAccountName() {
        return accountName;
    }
    public int getAccountBalance() {
        return accountBalance;
    }

    // no setters method
    // so this pojo will act as only read only class
}
```

4. Immutability:

To create an immutable class, we can remove the setter and put-getter methods.

In getter methods, we can return a new copy instead of returning the original object to protect it from getting modified.

AccountFinal.java

```java
public final class AccountFinal {

    private final String accountName;
    private final int accountBalance;
    private final Address address;

    // constructor with all arguments

    //getter method for accountName and accountBalance

    // modify getter method to return new copy of Address object
    // to achieve immutability
    public Address getAddress() {
        return new Address(address.getCity(),address.getPin()) ;
    }
}

/*
Note: For simplicity, we've not created a full-fledged immutable class
and few parts of the class are also hidden
*/
```

Conclusion:

In the above scenarios, we've only achieved encapsulations at diff levels & that's the main reason for using getter/setter in java.

To see the above examples in more detail and run them you can access below git repo:
https://t.co/vHZsSmtJqS