CPSC 501 – Assignment 1

Artem Golovin
30018900

Overview

The project that was refactored for this assignment is iKeirNez/assessment-loan-system. To preform refactoring on the project, the following fork was created awave1/assessment-loan-system. The fork contains two branches: `master` and `refactor`. The initial work is left untouched in `master` branch and the refactoring was done in `refactor` branch. To simplify the access, `refactor` branch is default (main) branch.

Original project did not include any README or instructions how to get up and running with the project, therefore I included README, describing the steps necessary to build, run, and test the code. To make it easier to manage dependencies and to build the project, I added support for gradle.

Refactoring structure

The refactorings that were made in the project, can be categorized into three categories: **minor**, **medium** and **major**. **Minor** refactorings were usually made together with **major** and **minor** refactorings, thus no commits were directly dedicated to **minor** refactorings.

Major changes

This section explains and displays the number of **major** changes that were made to the project during refactoring process. Each major change is associated with commit hash.

Using Visitor pattern to eliminate `instanceof` checking (2534ae1e)

Abstracting busy waiting for user input (bd538f0)

The application relies on input from keyboard. In classes, where the user input is required, the original code included use of `do { ... } while(...);` loops, to wait for valid user input. User input is required in `Property<T>` (and its subclasses), `Menu` and `Controller`. As a result, `do { ... } while(...);` loop has been moved to separate class called `ConsoleInput<T>`, in method `public Optional<T> waitForInput(InputWait<T> inputWait)`. `InputWait<T>` is an interface with only one method, passing it as a parameter allows us to pass anonymous lambda function (e.g. `arg -> {/* function body */}`) as a parameter, instead of implementing a method, that was declared in the interface. The following is the implementation of `waitForInput` method:

```java
public Optional<T> waitForInput(InputWait<T> inputWait) {
  Optional<T> fetchedObj;
  do {
    fetchedObj = inputWait.getInput(this.scanner);
  } while (!fetchedObj.isPresent());

  return fetchedObj;
}
```

When `waitForInput` is called, we have to supply instance of `InputWait`, for example:

```java
// Set the scanner
setScanner(scanner);
Optional<Option> option = waitForInput(scnr -> {
  /*
   * Do all the necessary things here, using Scanner scnr variable
   */

  // Return Optional object result
  return Optional.of(obj);
});
```

```java
//... Optional<Option> option can later be safely unwrapped and used
```

By abstracting the `do{...} while()` loop into a separate method, we got rid of duplicated code and made it more readable. Also other simple classes that need to wait for user input can now inherit this class and call `waitForInput`.

To do this refactoring, **Replace Method with Method object** technique was used. For example, with this technique applied, we got rid of `processLogin` method in `User` class, so the following code:

```java
private User processLogin() {
  User user;
  do {
    System.out.println("Please enter your username.");
    String username = scanner.next();

    System.out.println("Please enter your password.");
    String password = scanner.next();

    user = model.getUserRepository().getExact(username, password);

    if (user == null) { // invalid credentials
      System.out.println("Invalid credentials, please try again.");
    }
  } while (user == null);

  return user;
}
```

was replaced with:

```java
Optional<User> user = waitForInput(s -> {
  User usr;
  System.out.println("Please enter your username.");
  String username = scanner.next();

  if (user != null) {
    System.out.println("Please enter your password.");
    String password = scanner.next();
    usr = model.getUserRepository().getExact(username, password);
    if (usr == null) { // invalid credentials
      System.out.println("Invalid credentials, please try again.");
    }

    return Optional.of(usr);
});
```

The following files were changed and added as a result of this refactoring:

- src/main/java/com/keirnellyer/glencaldy/manipulation/property/type/Property.java
- src/main/java/com/keirnellyer/glencaldy/menu/Menu.java
- src/main/java/com/keirnellyer/glencaldy/runtime/Controller.java
- src/main/java/com/keirnellyer/glencaldy/runtime/Controller.java
- src/main/java/com/keirnellyer/glencaldy/util/ConsoleInput.java
- src/main/java/com/keirnellyer/glencaldy/util/InputWait.java

To test `ConsoleInput<T>`, I had to mock user input, using `ByteArrayInputStream`. There are four tests for this class, located in `ConsoleInputTest`. Two tests are used to test `ConsoleInput<User>` and one more to test and display functionality using built in object `ConsoleInput<String>`.

Adding `ConsoleInput<T>` allows us to add ability to use busy-waiting on any class, by utilizing Java Generics.

TODO (fcaddaa8, 8fb88595)

Medium changes

TODO (67287c0)

TODO (aef6c56)

TODO (b6b0c98, 8a898a0)

TODO (ad3147)