HW N.-21

Name: Eduardo Andres Segarra Diaz

Group: The java bandits

NRC: 14549

Pitfalls

Pitfall #1: Oddball Solution

Code:

```
30      public static String obtainIdFromJSON(String json) {
31          JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
32          return jsonObject.get("id").getAsString();
33      }
34
35      public static String obtainNameFromJSON(String json) {
36          JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
37          return jsonObject.get("name").getAsString();
38      }
39
40      public static String obtainEmailFromJSON(String json) {
            JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
42          return jsonObject.get("email").getAsString();
43      }
44
45      public static String obtainPasswordFromJSON(String json) {
46          JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
47          return jsonObject.get("password").getAsString();
48      }
49
50      public static String obtainGradeFromJSON(String json) {
51          JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
52          return jsonObject.get("grade").getAsString();
53      }
54
55      public static String obtainTypeFromJSON(String json) {
56          JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
57          return jsonObject.get("type").getAsString();
58      }
59
60      public static Float obtainBalanceFromJSON(String json) {
61          JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
62          return Float.valueOf(jsonObject.get("balance").getAsString());
63      }
```

Problem description:

In the utils package, within the DataCollection class, the methods obtainIdFromJSON, obtainNameFromJSON, obtainEmailFromJSON, obtainPasswordFromJSON, obtainGradeFromJSON, obtainTypeFromJSON, and obtainBalanceFromJSON are all repeating the same logic to obtain and convert data from JSON.

To address this issue, a generic method should be created to manage the functionality for all these methods.

Pitfall #2: Duplicate code

Code:

```
public static void navigateToUserMenuFromLogin(JFrame parentFrame, String loginResponse) {
    String id = DataCollection.obtainIdFromJSON(loginResponse);
    String name = DataCollection.obtainNameFromJSON(loginResponse);
    String type = DataCollection.obtainTypeFromJSON(loginResponse);
    double accountBalance = DataCollection.obtainBalanceFromJSON(loginResponse);

    switch (type) {
        case "commensal" -> {
            FrmCommensalMenu frmCommensalMenu = new FrmCommensalMenu(name, id, accountBalance, type);
            parentFrame.setVisible(false);
            frmCommensalMenu.setVisible(true);
        }
        case "administrators" -> {
            FrmAdminMenu frmAdminMenu = new FrmAdminMenu(name, accountBalance, type, id);
            parentFrame.setVisible(false);
            frmAdminMenu.setVisible(true);
        }
        case "militaryChef" -> {
            FrmChefMenu frmChefMenu = new FrmChefMenu(name);
            parentFrame.setVisible(false);
            frmChefMenu.setVisible(true);
        }
        case "generalAdministrator" -> {
            FrmGeneralAdmin frmGeneralAdmin = new FrmGeneralAdmin(name, id, accountBalance, type);
            parentFrame.setVisible(false);
            frmGeneralAdmin.setVisible(true);
        }
    }
}
```

```
public static void navigateToUserMenu(JFrame parentFrame, String name, String id, double accountBalance, String type) {
    switch (type) {
        case "commensal" -> {
            FrmCommensalMenu frmCommensalMenu = new FrmCommensalMenu(name, id, accountBalance, type);
            parentFrame.setVisible(false);
            frmCommensalMenu.setVisible(true);
        }
        case "administrators" -> {
            FrmAdminMenu frmAdminMenu = new FrmAdminMenu(name, accountBalance, type, id);
            parentFrame.setVisible(false);
            frmAdminMenu.setVisible(true);
        }
        case "militaryChef" -> {
            FrmChefMenu frmChefMenu = new FrmChefMenu(name);
            parentFrame.setVisible(false);
            frmChefMenu.setVisible(true);
        }
        case "generalAdministrator" -> {
            FrmGeneralAdmin frmGeneralAdmin = new FrmGeneralAdmin(name, id, accountBalance, type);
            parentFrame.setVisible(false);
            frmGeneralAdmin.setVisible(true);
        }
    }
}
```

Problem description:

In the utils package, within the InterfacesActions class, the methods navigateToUserMenu and navigateToUserMenuFromLogin have similar functionality and this can be solved with a generic method.

Pitfall #3: Duplicate code

Code:

```
public static void settingLabelsInvisible(JLabel lblBreakfast, JLabel lblAvailableBreakfast, JLabel lblLunch,
        JLabel lblAvailableLunch, JLabel lblSnack, JLabel lblAvailableSnack) {
    lblBreakfast.setVisible(false);
    lblAvailableBreakfast.setVisible(false);
    lblLunch.setVisible(false);
    lblAvailableLunch.setVisible(false);
    lblSnack.setVisible(false);
    lblAvailableSnack.setVisible(false);
}

public static void settingLabelsVisible(JLabel lblBreakfast, JLabel lblAvailableBreakfast, JLabel lblLunch,
        JLabel lblAvailableLunch, JLabel lblSnack, JLabel lblAvailableSnack) {
    lblBreakfast.setVisible(true);
    lblAvailableBreakfast.setVisible(true);
    lblLunch.setVisible(true);
    lblAvailableLunch.setVisible(true);
    lblSnack.setVisible(true);
    lblAvailableSnack.setVisible(true);
}
```

Problem description:

In the utils package, within the LabelsActions class, the methods settingLabelsInvisible and settingLabelsVisible have similar functionality and this can be solved with a generic method.

Pitfall #4: Long Methods

Code:

```
public static void loopForShowingTheMenu(JLabel lblAvailablePlates, JLabel lblBreakfast, JLabel lblAvailableBreakfast,
        JLabel lblLunch, JLabel lblAvailableLunch, JLabel lblSnack, JLabel lblAvailableSnack, DateBook datebook,
        LocalDate today) {
    List<Document> documents = CloudController.getMenuInformation();
    Map<String, Boolean> reservedDays = datebook.getReservedDays();

    for (Document doc : documents) {
        String date = doc.getString("date");
        String breakfast = doc.getString("breakfast");
        String lunch = doc.getString("lunch");
        String dinner = doc.getString("dinner");

        for (Map.Entry<String, Boolean> entry : reservedDays.entrySet()) {
            String dateReserved = entry.getKey();

            String[] parts = dateReserved.split("/");
            String day = parts[0];
            String month = parts[1];
            String year = parts[2];
            LocalDate dateSearch = LocalDate.of(Integer.parseInt(year), Integer.parseInt(month), Integer.parseInt(day));
            String dateToCompare = day + "/" + month + "/" + year;

            if (today.isBefore(dateSearch) || today.isEqual(dateSearch)) {
                if (dateToCompare.contentEquals(date)) {
                    ifMenuIsAvailable(lblAvailablePlates, dateToCompare);
                    settingLabelsVisible(lblBreakfast, lblAvailableBreakfast, lblLunch, lblAvailableLunch, lblSnack,
                            lblAvailableSnack);
                    settingLMeals(lblAvailableBreakfast, lblAvailableLunch, lblAvailableSnack, breakfast, lunch, dinner);
                    break;
                } else if (!dateToCompare.contentEquals(date)) {
                    ifMenuIsNotAvailable(lblAvailablePlates, dateToCompare);
                    break;
                }
            }

        }
```

```java
public static void loopForShowingTheMenuForChefs(List<Document> documents, LocalDate today, JLabel lblAvailablePlates,
        JLabel lblBreakfast, JLabel lblAvailableBreakfast, JLabel lblLunch, JLabel lblAvailableLunch, JLabel lblSnack,
        JLabel lblAvailableSnack) {
    for (Document doc : documents) {
        String date = doc.getString("date");
        String[] parts = date.split("/");
        String day = parts[0];
        String month = parts[1];
        String year = parts[2];
        String dateToCompare = day + "/" + month + "/" + year;
        LocalDate dateSearch = LocalDate.of(Integer.parseInt(year), Integer.parseInt(month), Integer.parseInt(day));
        String breakfast = doc.getString("breakfast");
        String lunch = doc.getString("lunch");
        String dinner = doc.getString("dinner");

        if (today.isBefore(dateSearch) || today.isEqual(dateSearch)) {
            LabelsActions.ifMenuIsAvailable(lblAvailablePlates, dateToCompare);
            LabelsActions.settingLabelsVisible(lblBreakfast, lblAvailableBreakfast, lblLunch, lblAvailableLunch, lblSnack,
                    lblAvailableSnack);
            LabelsActions.settingLMeals(lblAvailableBreakfast, lblAvailableLunch, lblAvailableSnack, breakfast, lunch, dinner);
            break;
        }
    }
}
```

Problem description:

In the utils package, within the LabelsActions class, the methods loopForShowingTheMenuForChefs and loopForShowingTheMenu have the nesting logic is quite complex and difficult to follow, with several levels of loops and conditions. The solution for this problem is to divide the code in small and manageable methods.

Pitfall #5: Inconsistent Names

Code:

```java
public static boolean validCommensal(Commensal commensal) {
    return commensal != null && commensal.getId() != null && !commensal.getId().isEmpty();
}

public static boolean validBalance(double balance) {
    return balance >= 0;
}
```

```java
public static boolean validBoolean() {
    Scanner scanner = new Scanner(System.in);
    boolean userInput;

    while (true) {
        try {
            userInput = scanner.nextBoolean();
            if (userInput == true || userInput == false) {
                return userInput;
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid entry. Please type true or false.");
            scanner.next();
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Problem description:

In the utils package, within the Validation class, the methods validCommensal, validBalance and validBoolean do not describe well the functionality of these methods.

Pitfall #6: Dead Code

Code:

```
public static boolean validBoolean() {
    Scanner scanner = new Scanner(System.in);
    boolean userInput;

    while (true) {
        try {
            userInput = scanner.nextBoolean();
            if (userInput == true || userInput == false) {
                return userInput;
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid entry. Please type true or false.");
            scanner.next();
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Problem description:

In the utils package, within the Validation class, the method validBoolean is never used in the code. The solution is to either implement this method somewhere in the code or remove it.