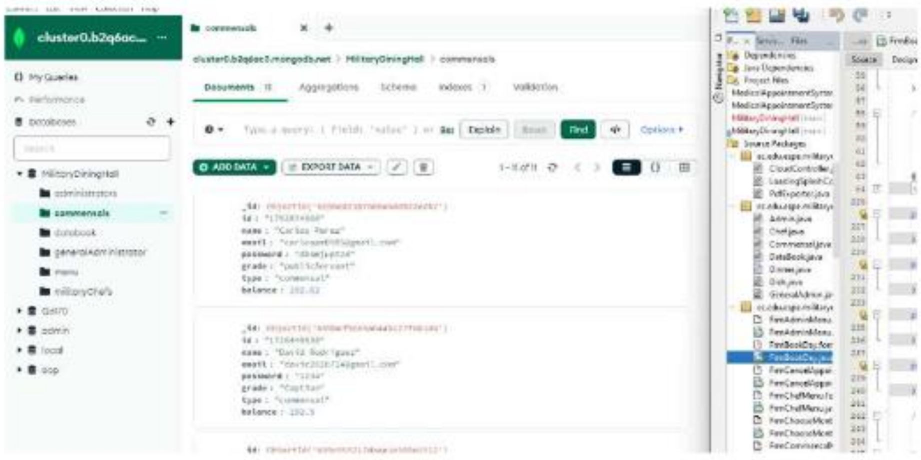
17. PEREZ CONDOR CARLOS ANDRES
18. RODRIGUEZ VILLAROEL DAVID JOSUE
19. SEGARRA DIAZ EDUARDO ANDRES
20. TRAVEZ CACHAGO ALEX ISMAEL

the name of the classes match that of the collections



```
_id: ObjectId('[illegible]')
id : "[illegible]"
name : "Carlos Perez"
email : "[illegible]@gmail.com"
password : "[illegible]"
grade : "publicServant"
type : "commensal"
balance : [illegible]

_id: ObjectId('[illegible]')
id : "[illegible]"
name : "David Rodriguez"
email : "[illegible]@gmail.com"
password : "1234"
grade : "Captian"
type : "commensal"
balance : [illegible]
```

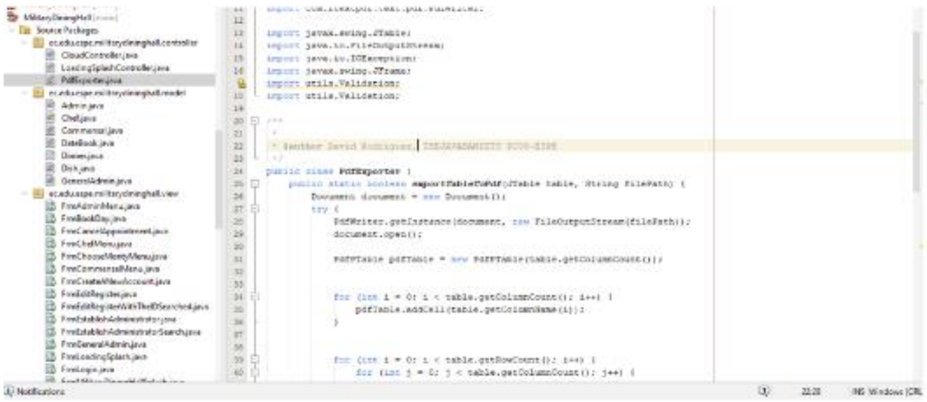
The class name was changed to an appropriate one.

```java
public class Table {

    public static void updateTableOfMenus(JTable tblTable) {
        List<Document> documents = CloudController.getMenuInformation();

        DefaultTableModel model = (DefaultTableModel) tblTable.getModel();
        model.setRowCount(0);

        for (Document doc : documents) {
            String date = doc.getString("date");
            String breakfast = doc.getString("breakfast");
            String lunch = doc.getString("lunch");
            String dinner = doc.getString("dinner");
            model.addRow(new Object[]{date, breakfast, lunch, dinner});
        }
    }
```

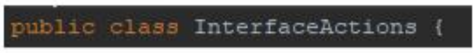
the method names were changed to appropriate ones

```java
public static boolean createANewAdministrator(Object object) {
    ConnectionString connectionString = new ConnectionString("mongodb+srv://segarra:segarra@cluster0.b2q6ac3.mo
    try (MongoClient mongoClient = MongoClients.create(connectionString)) {
        MongoDatabase database = mongoClient.getDatabase("MilitaryDiningHall");

        MongoCollection<Document> collection = database.getCollection("administrators");

        Gson gson = new Gson();
        String json = gson.toJson(object);
        Document studentDocument = Document.parse(json);

        collection.insertOne(studentDocument);

        return true;
    } catch (Exception e) {
        return false;
    }
}
```

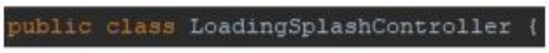
unnecessary comments were deleted



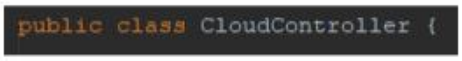
```java
import com.itextpdf.text.pdf.PdfWriter;

import javax.swing.JTable;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.swing.JFrame;
import utils.Validation;
import utils.Validation;

/**
 *
 * @author David Rodriguez, [illegible]
 */
public class PdfExporter {
    public static boolean exportTableToPdf(JTable table, String filePath) {
        Document document = new Document();
        try {
            PdfWriter.getInstance(document, new FileOutputStream(filePath));
            document.open();

            PdfPTable pdfTable = new PdfPTable(table.getColumnCount());


            for (int i = 0; i < table.getColumnCount(); i++) {
                pdfTable.addCell(table.getColumnName(i));
            }


            for (int i = 0; i < table.getRowCount(); i++) {
                for (int j = 0; j < table.getColumnCount(); j++) {
```



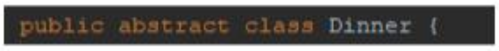
```java
public class InterfaceActions {
```



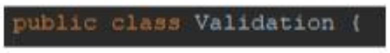
```java
public class LoadingSplashController {
```



```java
public class CloudController {
```

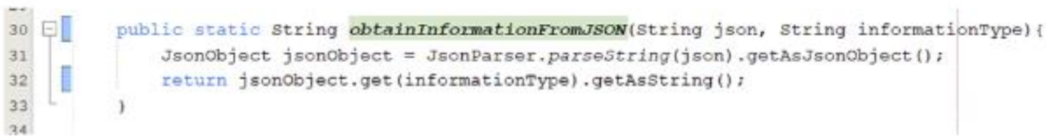


```java
public abstract class Dinner {
```



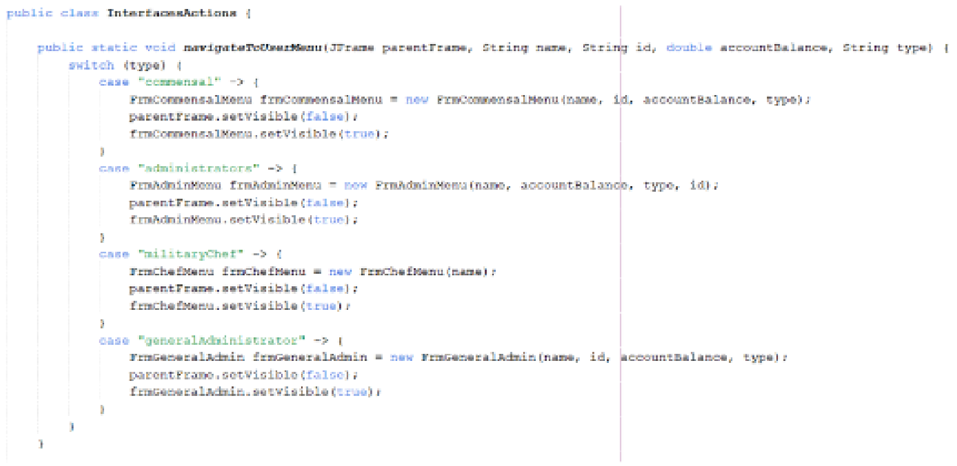
```java
public class Validation {
```

Pitfall: Oddball Solution



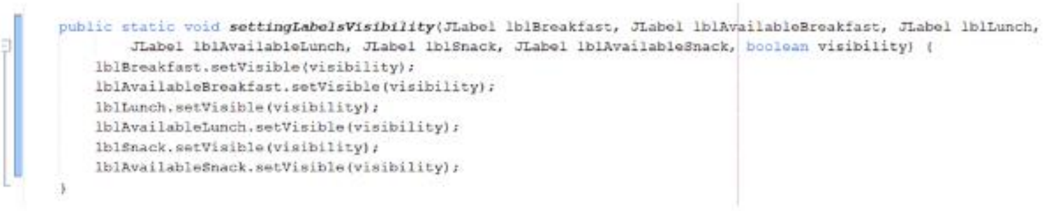
```java
    public static String obtainInformationFromJSON(String json, String informationType){
        JsonObject jsonObject = JsonParser.parseString(json).getAsJsonObject();
        return jsonObject.get(informationType).getAsString();
    }
```

Pitfall: Duplicate code



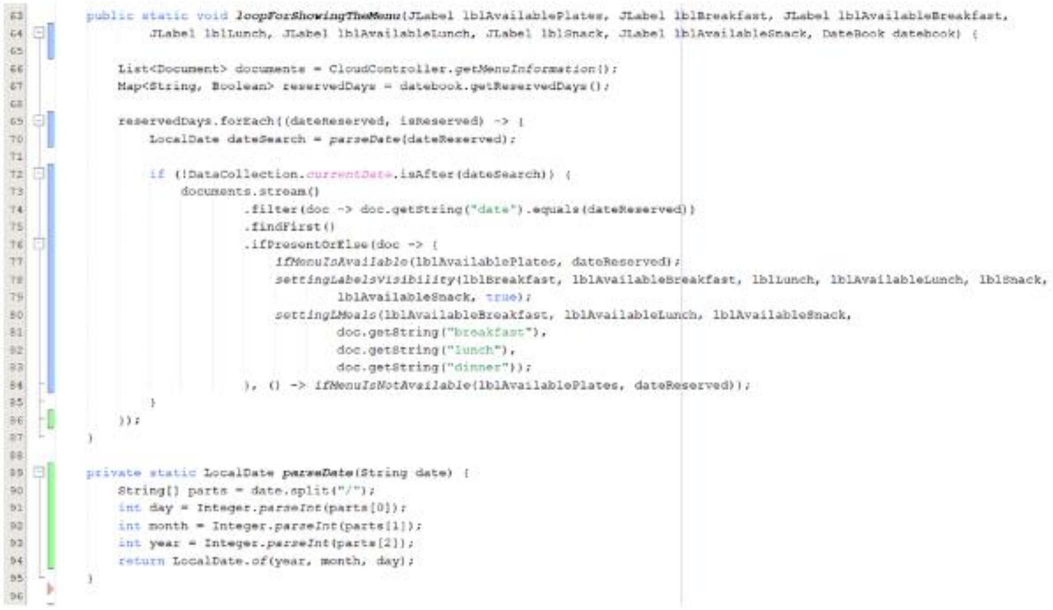
```java
public class InterfacesActions {

    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);
            }
        }
    }
```

Pitfall: Duplicate code

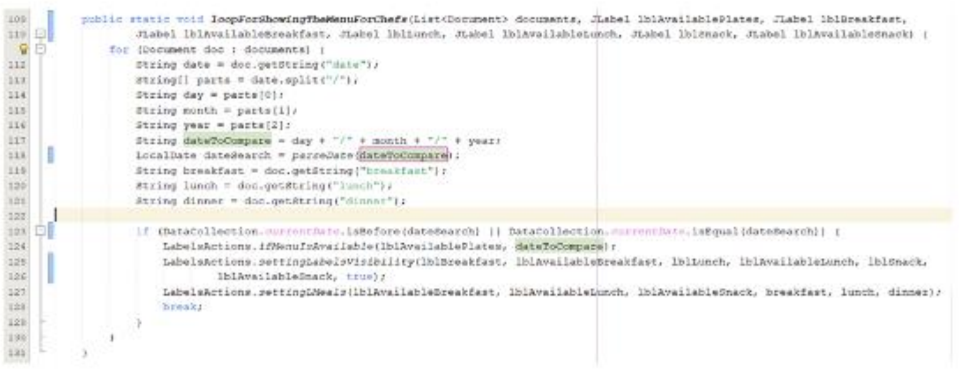


```java
public static void settingLabelsVisibility(JLabel lblBreakfast, JLabel lblAvailableBreakfast, JLabel lblLunch,
        JLabel lblAvailableLunch, JLabel lblSnack, JLabel lblAvailableSnack, boolean visibility) {
    lblBreakfast.setVisible(visibility);
    lblAvailableBreakfast.setVisible(visibility);
    lblLunch.setVisible(visibility);
    lblAvailableLunch.setVisible(visibility);
    lblSnack.setVisible(visibility);
    lblAvailableSnack.setVisible(visibility);
}
```

Pitfall: Long Methods

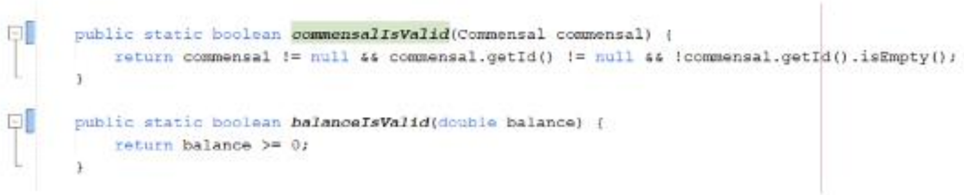


```java
public static void loopForShowingTheMenu(JLabel lblAvailablePlates, JLabel lblBreakfast, JLabel lblAvailableBreakfast,
        JLabel lblLunch, JLabel lblAvailableLunch, JLabel lblSnack, JLabel lblAvailableSnack, DateBook datebook) {

    List<Document> documents = CloudController.getMenuInformation();
    Map<String, Boolean> reservedDays = datebook.getReservedDays();

    reservedDays.forEach((dateReserved, isReserved) -> {
        LocalDate dateSearch = parseDate(dateReserved);

        if (!DataCollection.currentDate.isAfter(dateSearch)) {
            documents.stream()
                    .filter(doc -> doc.getString("date").equals(dateReserved))
                    .findFirst()
                    .ifPresentOrElse(doc -> {
                        ifMenuIsAvailable(lblAvailablePlates, dateReserved);
                        settingLabelsVisibility(lblBreakfast, lblAvailableBreakfast, lblLunch, lblAvailableLunch, lblSnack,
                                lblAvailableSnack, true);
                        settingLMeals(lblAvailableBreakfast, lblAvailableLunch, lblAvailableSnack,
                                doc.getString("breakfast"),
                                doc.getString("lunch"),
                                doc.getString("dinner"));
                    }, () -> ifMenuIsNotAvailable(lblAvailablePlates, dateReserved));
        }
    });
}

private static LocalDate parseDate(String date) {
    String[] parts = date.split("/");
    int day = Integer.parseInt(parts[0]);
    int month = Integer.parseInt(parts[1]);
    int year = Integer.parseInt(parts[2]);
    return LocalDate.of(year, month, day);
}
```

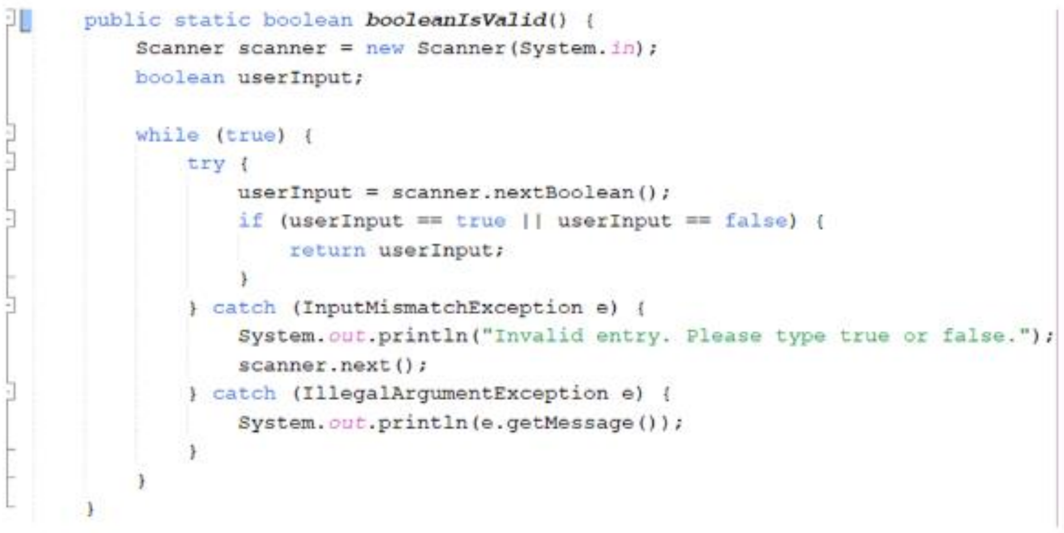


```java
public static void loopForShowingTheMenuForChefs(List<Document> documents, JLabel lblAvailablePlates, JLabel lblBreakfast,
        JLabel lblAvailableBreakfast, JLabel lblLunch, JLabel lblAvailableLunch, JLabel lblSnack, JLabel lblAvailableSnack) {
    for (Document doc : documents) {
        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 = parseDate(dateToCompare);
        String breakfast = doc.getString("breakfast");
        String lunch = doc.getString("lunch");
        String dinner = doc.getString("dinner");

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

Pitfall: Inconsistent Names



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

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



```java
public static boolean booleanIsValid() {
    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());
        }
    }
}
```

Pitfall: Dead Code

The method is no longer in the code