**Table of contents**

[1 Program Description 3](#_Toc122395488)

[2 Overview of program classes 8](#_Toc122395489)

[3 Class diagram 11](#_Toc122395490)

[4 Use case diagram 13](#_Toc122395491)

[App 14](#_Toc122395492)

# 1 Program description

Base of this application has been got from <https://gluconatekid.github.io/ProgrammingProjects/todolist.html>.

Translation has been done by Yandex translator automatically.

The WorkOrganizer app WorkOrganizer was created based on the TODO to-do list and is an extension of its functionality. The TODO Sheet app листis described below. For more information about the Todo list program , see the report on this program.

TODO list - anapplication similar to notes from your phone. It will help you achieve any result within the desired time frame. TO-DO list. TODO to-do list. You can't pass a session without this app. TODO to-do list includes:

- you can add notes;  
- can be edited;  
- you can sort it.

The app reminds the user of their scheduled tasks and consists of a table consisting of 6 columns:   
- Task – name of the task;  
- Day – day on which the task needs to be completed;  
- Time – time;  
- Deadline – day of the week, or the date on which the event will be held;  
- Mentor – the person that the task is associated  
with - Description – a more detailed description of the task.

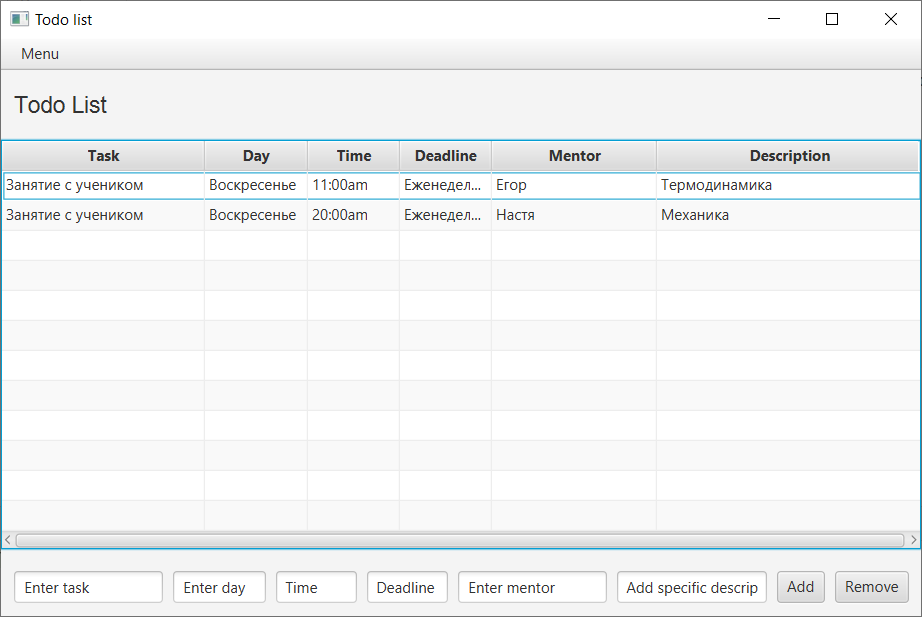


Figure 1 – The Todo list application window.

Notes from the Todo list стали задачами have become taskstasks). The table fields have been changed so that it is no longer a sheet of notes, but rather a list of tasks (see Figure 2). In WorkOrganizeraddition to adding and deleting tasks, WorkOrganizer also has a new functionality that automatically distributes tasks evenly by day of the week. You can do this by using the Organize tab in the top menu (see Figure 2).

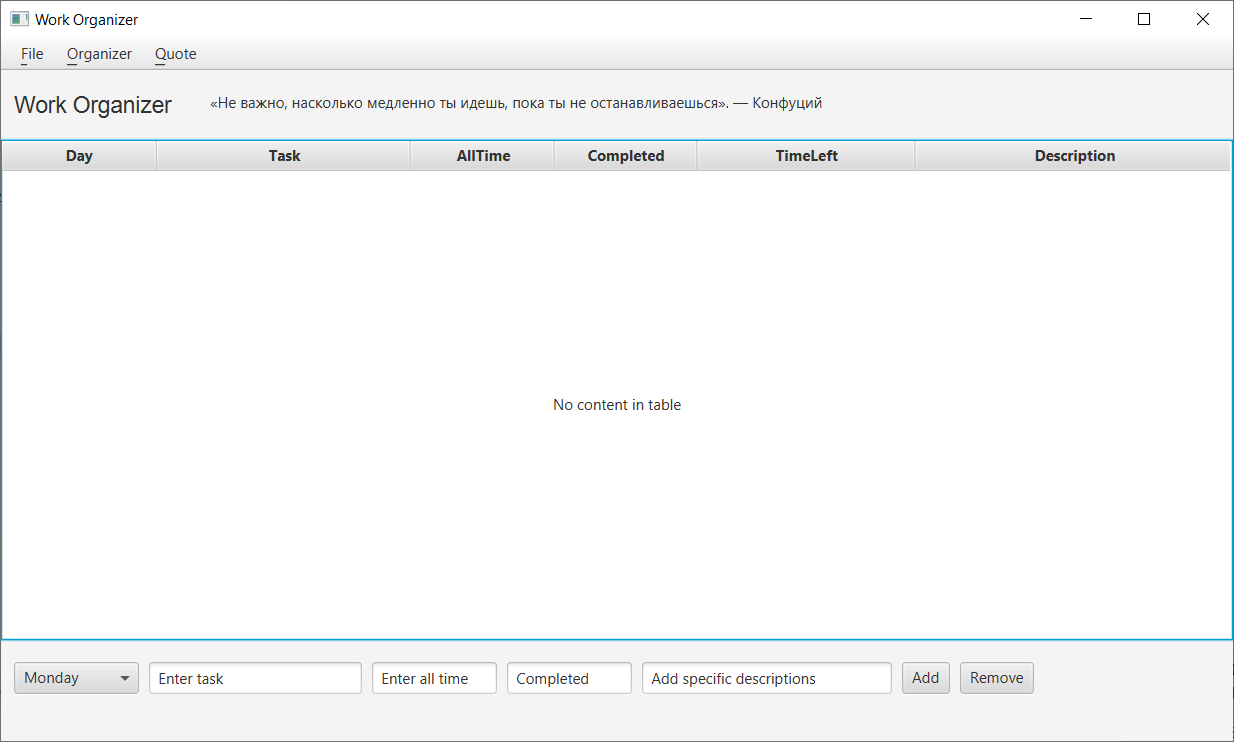


Figure 2 – Appearance of the WorkOrganizer app WorkOrganizer(when opened).

Table fields:

- Day – day of the week, takes strictly fixed values. When adding a value via the bottom panel, you can only select it from the drop-down list. When editing an issue, you can change the Day field, but only by the value from this list. In other words, there is protection against incorrect input.

- Task – directly the task. You can edit it.

- AllTime – all time required for the task. Based on this and the Completed field, the TimeLeft field is formed using a simple formula. But despite this, the AllTime field is editable. When editing, TimeLeft changes its value. This field is also protected from incorrect input – negative values, or non-numeric input.

- Completed – a field that the user changes as they work on the task. This field is protected from entering a number greater than 100 or less than 0, and from non-numeric input.

- TimeLeft – an immutable field whose value is calculated based on the previous two fields.

- Description – field for entering additional information about the issue (if necessary)

The functionality of the program is inherited from the parent program Todo list. In Work Organizer , you can add, delete, edit, and sort tasks (by day of the week, alphabetically, or by hour) (see Figure 3)..

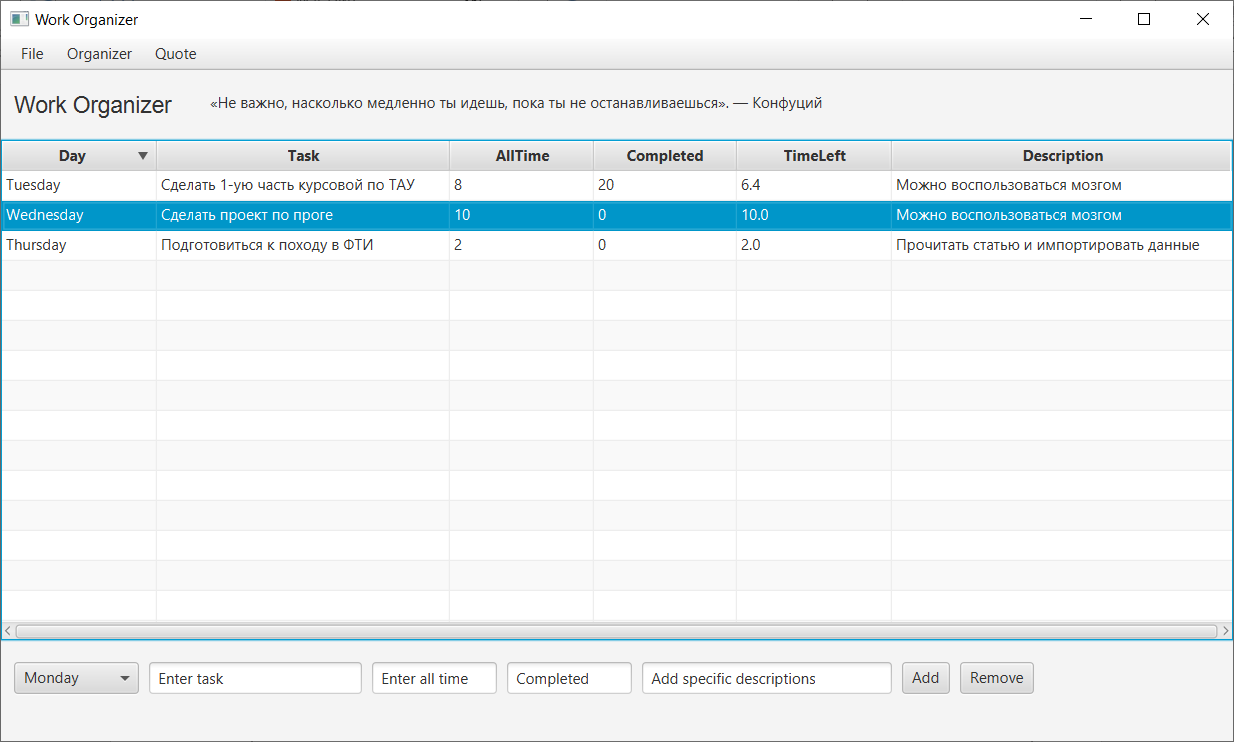


Figure 3 – A table with added tasks and tasks sorted by day of the week.

Added functionality for even distribution of tasks by day of the week. This is done as follows:

- click the Organizer button in the upper panel.

- select the Make week plan function.

- in the window that opens (see Figure 4), add tasks by clicking on the Add button.

- edit tasks by double-clicking on them.

- click on the Organize tasks button.

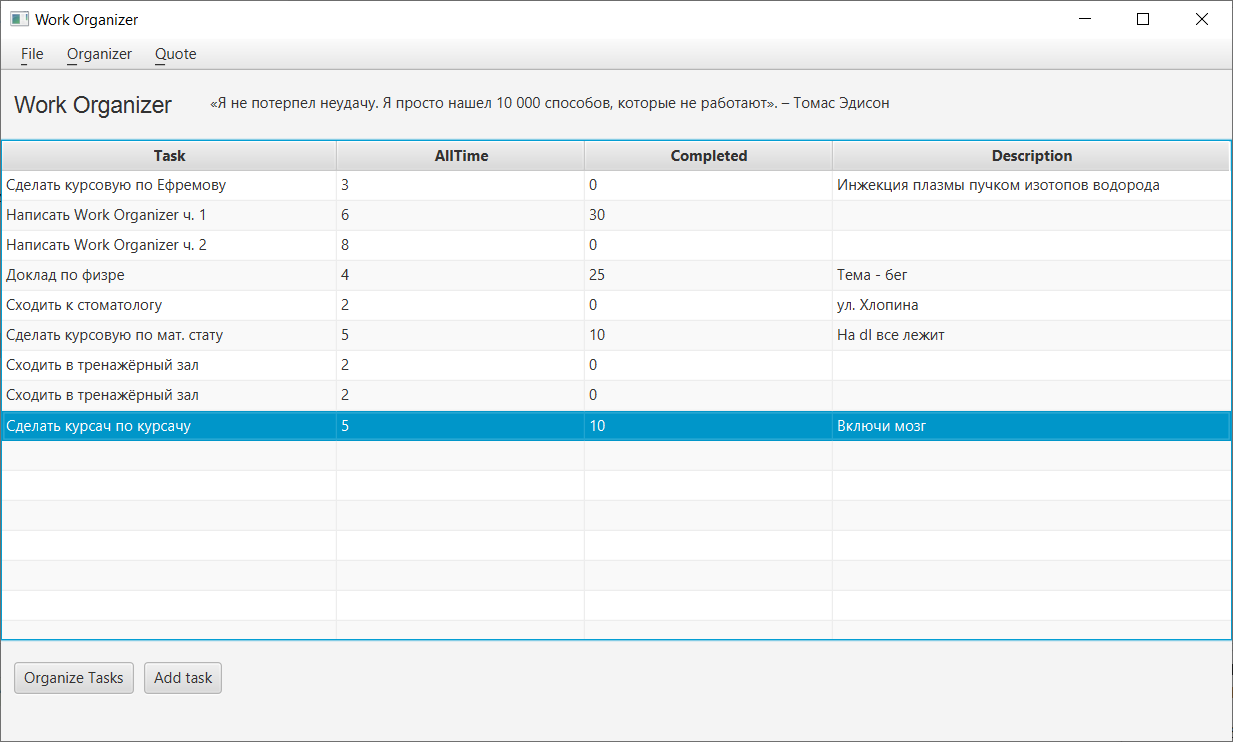


Figure 4 – The second scene with the addition of questions to distribute them evenly.

After doing this algorithm, we get the following picture:

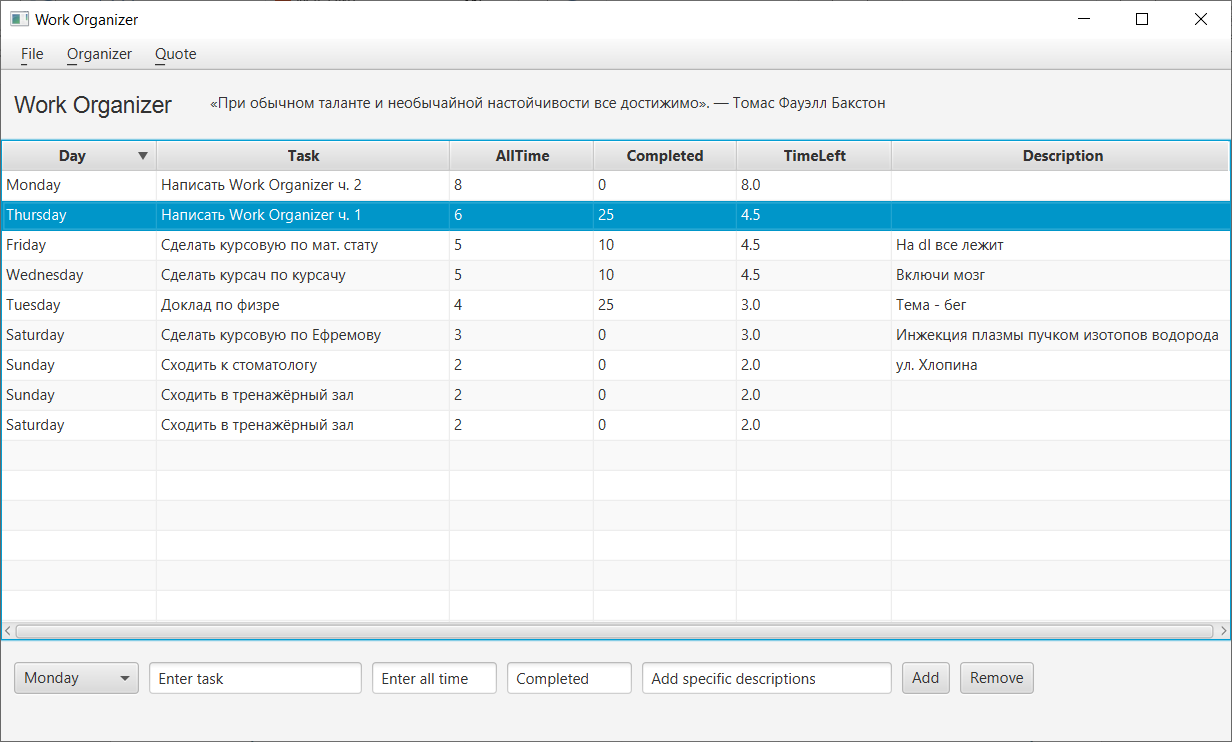


Figure 5-Tasks after distribution.

Also, as a feature, I added motivational and life quotes, which are displayed next to the program name, and are also updated when organizing tasks (when switching between windows), as well as when clicking on Quote and selecting New in the top panel.

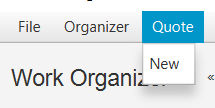
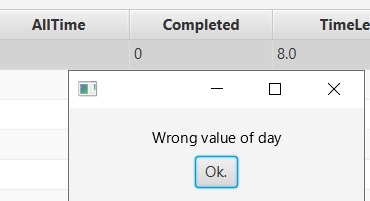
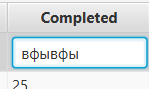


Figure 6 – Top menu of the program.

If incorrect input is entered at any point in the program, an error window will be called, and no input will be made.



Figures 7, 8 – Error output for incorrect input.

# 2 Overview of program classes

The app consists of 11 classes:

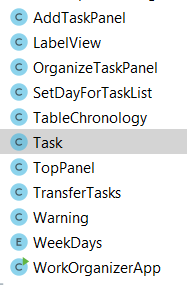


Figure 9-Program classes.

Let's review them.

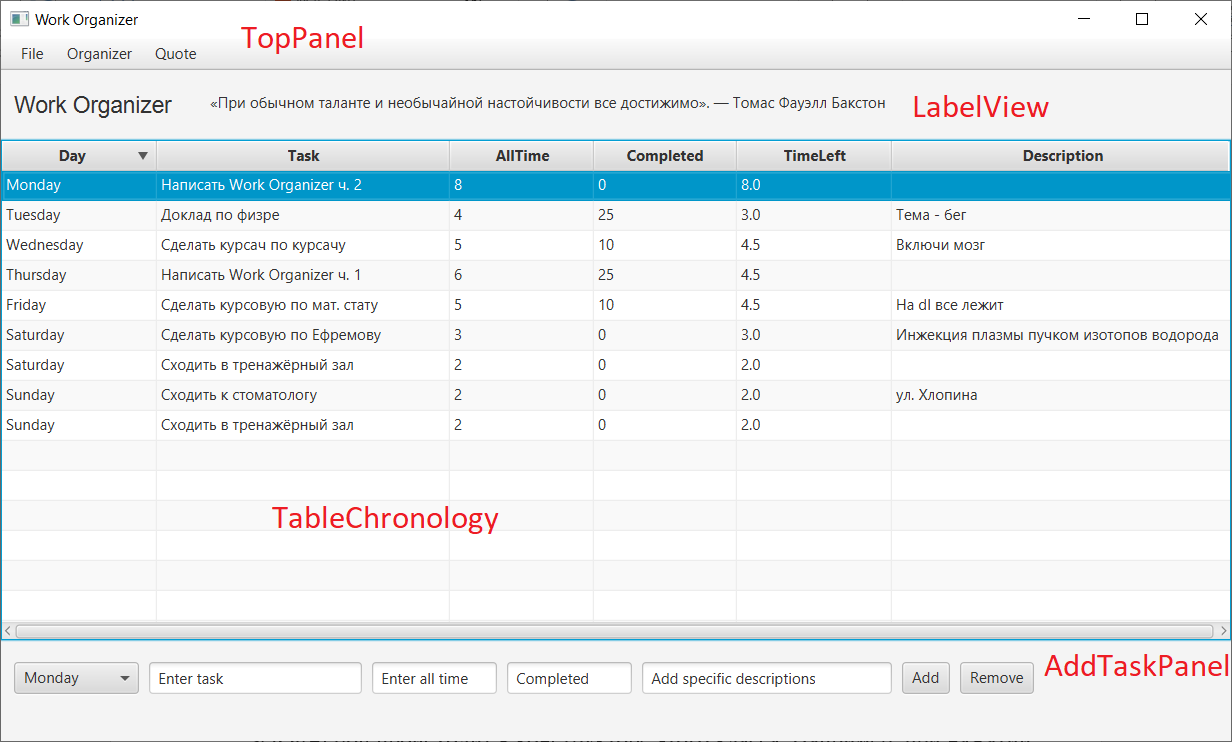


Figure 10-Classes in the window.

WorkOrganizerApp – the main class of the program, which contains the start() method, in which scenes are created (MainScene and organizeScene). For these scenes, component elements are created, such as: the top panel, title with a statement, tables, task addition panel, and also the main parameters of scenes are formed. In fact, all the components of the scene are contained in a VBox-a vertical box-a class that is necessary for special display of placement.

Let's go from top to bottom. The next class is topPanel. Contains вкладки the File, Organizer, and Quote tabs. Organizer and Quote have already been parsed, and the File tab lets you exit the app by clicking the exit button.

The next MenuView class is responsible for the top row, namely the Menu button. It consists of a MenuBar and several other classes. setOnAction is created and bound setOnAction in the constructor of this class. For example, when you click the Exit buttonExit, you exit the app.

Класс The LabelView class is used for app captions, as well as for displaying utterances. It is used to read from a file that stores all utterances.

The Table Chronology class is directly responsible for the main functionalityof the app. In fact, this class is a table and all the logic associated with it. TableChronology consists of a TableView, the mainclass that is managed by ourclass. TableView and contains columns and rows. Columns are created based on the name, the size occupied on the screen, and the handler responsible for the action when clicking on the field. The second field of the class-data -directly records from our table, which are presented in the form of a list of ObservableList Tasks, about the Task class, let's say further. Since the list is Observable, when deleting/changing/adding data, its display in the TableView will also change. TableView also has a feature to sort data by column in alphabetical order, and you can also use your own comparator to set sorting for the Day column.

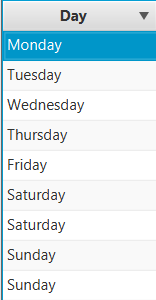
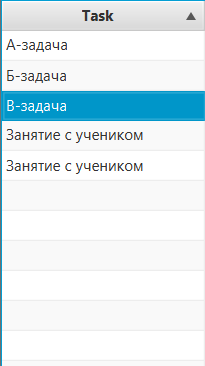


Figure 11-Sorting data by column in alphabetical order and in the order of days of the week.

Класс The Task class is an entity responsible for records in a table. It is nothing more than a simple data warehouse. It is worth noting that the fields are not just String, but SimpleStringProperty, which makes the text a property, which makes it easier to work with it.

AddTaskPanel – the lower part of the window, responsible for adding new entries. Has a link to tableChroonologyto change it. It consists of a horizontal box – Hbox, which stores the TextFieldsneeded to add the task to the mainlist. Also on the right are buttons for adding and deleting records.

OrganizeTaskPanel-an analog of AddTaskPanel for organizeScene.

SetDayForTaskList-essentially a drop-down list for selecting the day of the week.

Warning – an error that is used for incorrect input.

TransferData is essentially a static class. It specifies the logic for transferring data from organizeScene to MainScene, namely uniform distribution. This is done using the following algorithm: we add the longest task on the most unloaded day.

WeekDays-lists the days of the week. Made for more convenient working with days of the week as a custom data type.

You can make the following changes to classes that are responsible for logic:

- TableChronology, since it stores the Task, and it also contains the sorting logic and the logic for updating the values of the Time left field.

- Task, since this is a custom data type used in many places in the program.

- TransferTask, since it specifies the logic of the data transfer algorithm.

- WeekDays, since this is the data type for storing days of the week. It is also used in many places in the program and interacts with other classes.

- LabelView, because it reads data from a file, which is also the logic of the program.

# 3 Class diagram

To better understand the interaction between classes and their fields and methods, let's use the class diagram. See the next page.

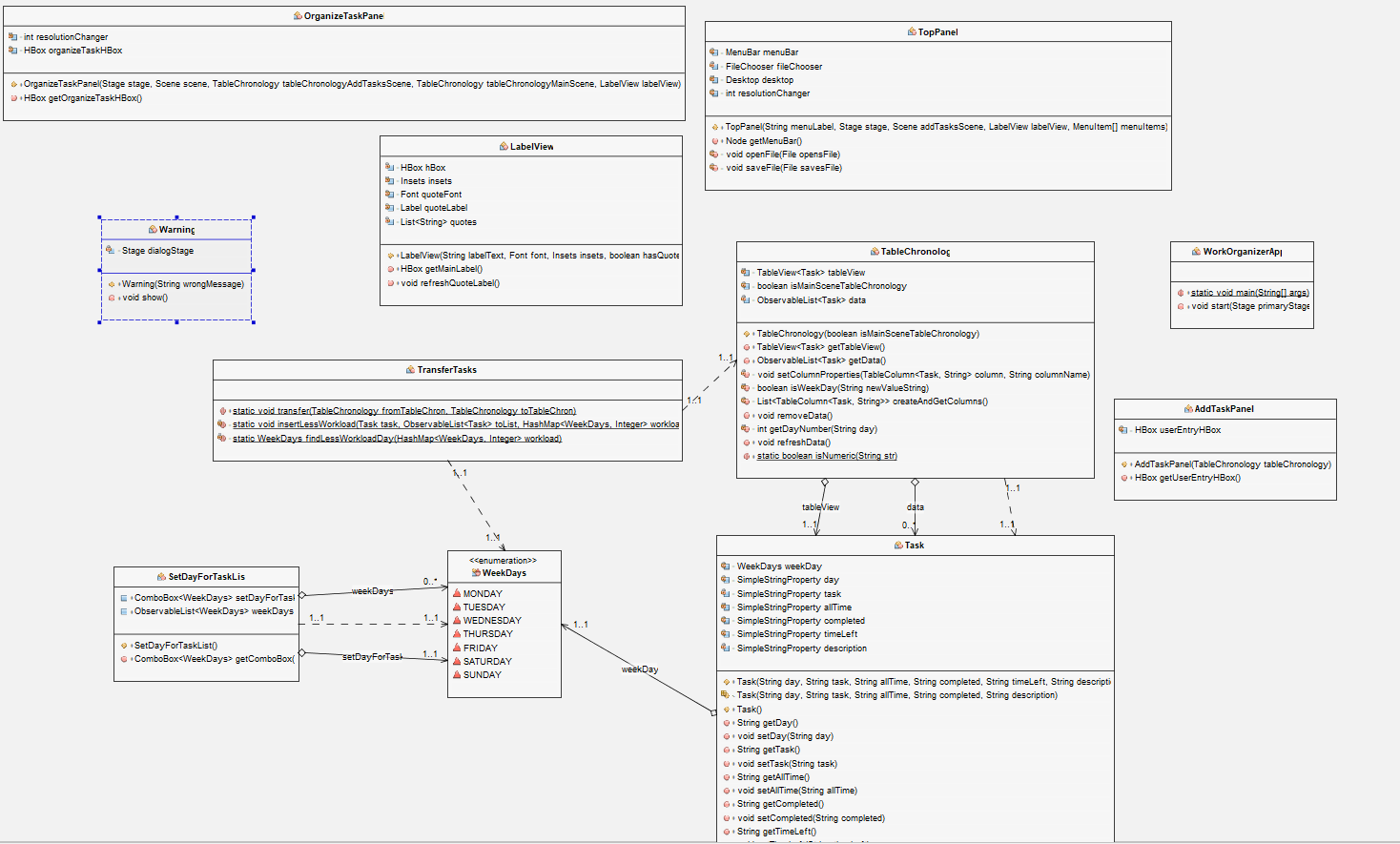


Figure 12-Class diagram.

# 4 Use case diagram

The functionality from the user's point of view can be easily and clearly depicted using a use case diagram:

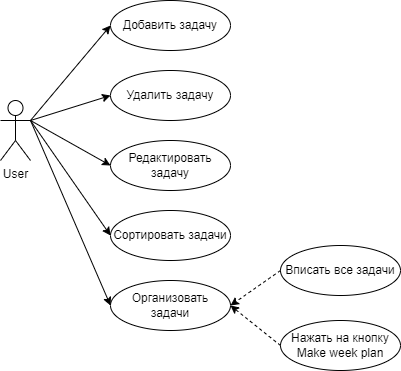


Figure 13-Use case diagram.

# Application

**Class The WorkOrganizerApp classWorkOrganizerApp:**

package com.example.workorganizer;  
  
import javafx.application.Application;  
import javafx.geometry.Insets;  
import javafx.scene.Scene;  
import javafx.scene.control.MenuItem;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Font;  
import javafx.stage.Stage;  
  
public class WorkOrganizerApp extends Application {  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) {  
 VBox allItemsMainScene = new VBox();  
 VBox allItemsAddTasksScene = new VBox();  
 Scene mainScene = new Scene(allItemsMainScene);  
 Scene organizeScene = new Scene(allItemsAddTasksScene);  
  
 allItemsMainScene.setSpacing(7);  
 allItemsAddTasksScene.setSpacing(7);  
  
 final LabelView labelMainScene = new LabelView("Work Organizer", new Font("Arial", 18),  
 new Insets(10, 10, 10, 10), true);  
 final LabelView labelAddTaskScene = new LabelView("Work Organizer", new Font("Arial", 18),  
 new Insets(10, 10, 10, 10), true);  
  
 TopPanel topPanelMainScene = new TopPanel("File", primaryStage, organizeScene, labelMainScene,  
 new MenuItem[] {new MenuItem("Open..."),  
 new MenuItem("Save As..."),  
 new MenuItem("Print..."),  
 new MenuItem("Exit")});  
  
 TopPanel topPanelAddTaskScene = new TopPanel("File", primaryStage, organizeScene, labelAddTaskScene,  
 new MenuItem[] {new MenuItem("Open..."),  
 new MenuItem("Save As..."),  
 new MenuItem("Print..."),  
 new MenuItem("Exit")});  
  
 TableChronology tableChronologyMainScene = new TableChronology(true);  
 TableChronology tableChronologyAddTasksScene = new TableChronology(false);  
  
 AddTaskPanel addTaskPanel = new AddTaskPanel(tableChronologyMainScene);  
  
 OrganizeTaskPanel organizeTaskPanel = new OrganizeTaskPanel(primaryStage, mainScene,  
 tableChronologyAddTasksScene, tableChronologyMainScene, labelMainScene);  
  
  
 allItemsMainScene.getChildren().addAll(topPanelMainScene.getMenuBar(), labelMainScene.getMainLabel(),  
 tableChronologyMainScene.getTableView(), addTaskPanel.getUserEntryHBox());  
  
 allItemsAddTasksScene.getChildren().addAll(topPanelAddTaskScene.getMenuBar(), labelAddTaskScene.getMainLabel(),  
 tableChronologyAddTasksScene.getTableView(), organizeTaskPanel.getOrganizeTaskHBox());  
  
 primaryStage.setWidth(1000);  
 primaryStage.setHeight(600);  
 primaryStage.setTitle("Work Organizer");  
 primaryStage.setScene(mainScene);  
 primaryStage.show();  
 }  
}

**Class topPanel classTopPanel:**

package com.example.workorganizer;  
  
import java.awt.Desktop;  
import java.io.File;  
import java.io.IOException;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
import javafx.application.Platform;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.print.PrinterJob;  
import javafx.scene.Node;  
import javafx.scene.Scene;  
import javafx.scene.control.Label;  
import javafx.scene.control.Menu;  
import javafx.scene.control.MenuBar;  
import javafx.scene.control.MenuItem;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
  
public class TopPanel {  
 private final MenuBar menuBar = new MenuBar();  
 private final FileChooser fileChooser = new FileChooser();  
 private final Desktop desktop = Desktop.*getDesktop*();  
 private int resolutionChanger = 1;  
  
 public TopPanel(String menuLabel, Stage stage, Scene addTasksScene,LabelView labelView, MenuItem[] menuItems) {  
 Label response = new Label("Menu");  
 Menu fileMenu = new Menu("\_" + menuLabel);  
 for (MenuItem menuItem : menuItems) {  
 fileMenu.getItems().add(menuItem);  
 }  
 menuBar.getMenus().add(fileMenu);  
  
 Menu Task = new Menu("\_" + "Organizer");  
 MenuItem addTask = new MenuItem("Make week plan");  
 Task.getItems().add(addTask);  
 menuBar.getMenus().add(Task);  
 addTask.setOnAction(ae -> {  
 labelView.refreshQuoteLabel();  
 stage.setScene(addTasksScene);  
 resolutionChanger \*= (-1);  
 stage.setWidth(stage.getWidth() + resolutionChanger);  
 });  
  
 Menu Quote = new Menu("\_" + "Quote");  
 MenuItem refresh = new MenuItem("New");  
 Quote.getItems().add(refresh);  
 refresh.setOnAction(ae -> labelView.refreshQuoteLabel());  
 menuBar.getMenus().add(Quote);  
  
 EventHandler<ActionEvent> MenuHandler =  
 ae -> {  
 String name = ((MenuItem) ae.getTarget()).getText();  
 if ("Exit".equals(name)) {  
 Platform.*exit*();  
 }  
 response.setText(name + " selected");  
 };  
  
 for (MenuItem menuItem : menuItems) {  
 menuItem.setOnAction(MenuHandler);  
 String name = menuItem.getText();  
 switch (name) {  
 case "Exit" -> menuItem.setOnAction(ae -> Platform.*exit*());  
 case "Open..." -> menuItem.setOnAction(  
 e -> {  
 File opensFile = fileChooser.showOpenDialog(stage);  
 if (opensFile != null) {  
 openFile(opensFile);  
 }  
 });  
 case "Save As..." -> menuItem.setOnAction(e -> {  
 File savesFile = fileChooser.showSaveDialog(stage);  
 if (savesFile != null) {  
 saveFile(savesFile);  
 }  
 });  
 case "Print..." -> menuItem.setOnAction(ae -> {  
 PrinterJob job = PrinterJob.*createPrinterJob*();  
 if (job != null) {  
 boolean showPrintDialog = job.showPrintDialog(stage.getOwner());  
 if (showPrintDialog) {  
 job.endJob();  
 }  
 }  
 });  
 }  
 }  
 }  
  
 public Node getMenuBar() {  
 return menuBar;  
 }  
  
 private void openFile(File opensFile) {  
 try {  
 desktop.open(opensFile);  
 } catch (IOException ex) {  
 Logger.*getLogger*(  
 WorkOrganizerApp.class.getName()).log(  
 Level.*SEVERE*, null, ex  
 );  
 }  
 }  
  
 private void saveFile(File savesFile) {  
 try {  
 desktop.open(savesFile);  
 } catch (IOException ex) {  
 Logger.*getLogger*(  
 WorkOrganizerApp.class.getName()).log(  
 Level.*SEVERE*, null, ex  
 );  
 }  
 }  
}

**Class LabelView class:**

package com.example.workorganizer;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
import javafx.geometry.Insets;  
import javafx.scene.control.Label;  
import javafx.scene.layout.HBox;  
import javafx.scene.text.Font;  
  
public class LabelView {  
 private final HBox hBox;  
 private final Insets insets;  
 private final Font quoteFont;  
 private final Label quoteLabel;  
 private final List<String> quotes = new ArrayList<>();  
  
 public LabelView(String labelText, Font font, Insets insets, boolean hasQuote){  
 this.insets = insets;  
 this.quoteFont = new Font(font.getSize()/1.5);  
  
 if (hasQuote) {  
 try {  
 FileReader fileReader = new FileReader("media//quotes.txt");  
 BufferedReader bufferedReader = new BufferedReader(fileReader);  
 String line = bufferedReader.readLine();  
 while (line != null) {  
 if (!line.equals("")){  
 quotes.add(line);  
 }  
 line = bufferedReader.readLine();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 Label mainLabel = new Label(labelText);  
 mainLabel.setFont(font);  
 mainLabel.setPadding(insets);  
  
 Font quoteFont = new Font(font.getSize()/1.5);  
 Random rnd = new Random();  
 quoteLabel = new Label(quotes.get(rnd.nextInt(quotes.size())));  
 quoteLabel.setFont(quoteFont);  
 quoteLabel.setPadding(insets);  
  
 hBox = new HBox();  
 hBox.getChildren().addAll(mainLabel, quoteLabel);  
 hBox.setSpacing(10);  
 }  
  
 public HBox getMainLabel() {  
 return hBox;  
 }  
  
 public void refreshQuoteLabel() {  
 Random rnd = new Random();  
 quoteLabel.setText(quotes.get(rnd.nextInt(quotes.size())));  
 quoteLabel.setFont(quoteFont);  
 quoteLabel.setPadding(insets);  
 }  
}

**Class TableChronology class**

package com.example.workorganizer;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Comparator;  
import java.util.List;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.control.cell.TextFieldTableCell;  
  
public class TableChronology {  
 private final TableView<Task> tableView = new TableView<>();  
 private final boolean isMainSceneTableChronology;  
 private ObservableList<Task> data = FXCollections.*observableArrayList*();  
  
 public TableChronology(boolean isMainSceneTableChronology){  
 this.isMainSceneTableChronology = isMainSceneTableChronology;  
 List<TableColumn<Task, String>> columns = createAndGetColumns();  
 tableView.setEditable(true);  
  
 tableView.setItems(data);  
 for (TableColumn<Task, String> column : columns) {  
 tableView.getColumns().add(column);  
 }  
 tableView.setColumnResizePolicy(TableView.*CONSTRAINED\_RESIZE\_POLICY*);  
 }  
  
 public TableView<Task> getTableView() {  
 return tableView;  
 }  
  
 public ObservableList<Task> getData() {  
 return data;  
 }  
  
 private void setColumnProperties(TableColumn<Task, String> column, String columnName){  
 column.setCellValueFactory(  
 new PropertyValueFactory<>(column.getText()));  
 column.setCellFactory(TextFieldTableCell.*forTableColumn*());  
 switch (columnName) {  
 case "Day" -> column.setOnEditCommit(  
 t -> {  
 String newValueString = t.getNewValue();  
 if (isWeekDay(newValueString)) {  
 t.getTableView().getItems().get(t.getTablePosition().getRow()).setDay(newValueString);  
 } else {  
 Warning warning = new Warning("Wrong value of day");  
 warning.show();  
 }  
 tableView.refresh();  
 }  
 );  
 case "Task" -> column.setOnEditCommit(  
 t -> t.getTableView().getItems().get(t.getTablePosition().getRow()).setTask(t.getNewValue())  
 );  
 case "AllTime" -> column.setOnEditCommit(  
 t -> {  
 String newValueString = t.getNewValue();  
 if (*isNumeric*(newValueString)) {  
 double newValue = Double.*parseDouble*(t.getNewValue());  
 if (newValue < 0) {  
 Warning warning = new Warning("Wrong value of all time");  
 warning.show();  
 tableView.refresh();  
 } else {  
 t.getTableView().getItems().get(t.getTablePosition().getRow()).setAllTime(newValueString);  
 refreshData();  
 }  
 } else {  
 Warning warning = new Warning("Wrong value of day");  
 warning.show();  
 tableView.refresh();  
 }  
 }  
 );  
 case "Completed" -> column.setOnEditCommit(  
 t -> {  
 String newValueString = t.getNewValue();  
 if (*isNumeric*(newValueString)) {  
 double newValue = Double.*parseDouble*(t.getNewValue());  
 if (newValue < 0 || newValue > 100) {  
 Warning warning = new Warning("Wrong value of completed");  
 warning.show();  
 tableView.refresh();  
 } else {  
 t.getTableView().getItems().get(t.getTablePosition().getRow()).setCompleted(newValueString);  
 refreshData();  
 }  
 } else {  
 Warning warning = new Warning("Wrong value of day");  
 warning.show();  
 tableView.refresh();  
 }  
 }  
 );  
 case "TimeLeft" -> column.setOnEditCommit(  
 t -> t.getTableView().getItems().get(t.getTablePosition().getRow()).setTimeLeft(t.getNewValue())  
 );  
 case "Description" -> column.setOnEditCommit(  
 t -> t.getTableView().getItems().get(t.getTablePosition().getRow()).setDescription(t.getNewValue())  
 );  
 }  
 }  
  
 private boolean isWeekDay(String newValueString) {  
 for (WeekDays weekDay: WeekDays.*getValues*()) {  
 if (newValueString.equals(weekDay.toString())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private List<TableColumn<Task, String>> createAndGetColumns() {  
 TableColumn<Task, String> dayCol = new TableColumn<>("Day");  
 TableColumn<Task, String> taskCol = new TableColumn<>("Task");  
 TableColumn<Task, String> allTimeCol = new TableColumn<>("AllTime");  
 TableColumn<Task, String> completedCol = new TableColumn<>("Completed");  
 TableColumn<Task, String> timeLeftCol = new TableColumn<>("TimeLeft");  
 TableColumn<Task, String> descriptionCol = new TableColumn<>("Description");  
  
 dayCol.setMinWidth(70);  
 taskCol.setMinWidth(150);  
 allTimeCol.setMinWidth(60);  
 completedCol.setMinWidth(60);  
 timeLeftCol.setMinWidth(120);  
 descriptionCol.setMinWidth(200);  
  
 setColumnProperties(dayCol, "Day");  
 setColumnProperties(taskCol, "Task");  
 setColumnProperties(allTimeCol, "AllTime");  
 setColumnProperties(completedCol, "Completed");  
 setColumnProperties(timeLeftCol, "TimeLeft");  
 setColumnProperties(descriptionCol, "Description");  
  
 Comparator<String> dayComparator = (d1, d2) -> getDayNumber(d2) - getDayNumber(d1);  
  
 dayCol.setComparator(dayComparator);  
 if (isMainSceneTableChronology){  
 timeLeftCol.setEditable(false);  
 return new ArrayList<>(Arrays.*asList*(dayCol, taskCol, allTimeCol,  
 completedCol, timeLeftCol, descriptionCol));  
 }  
  
 return new ArrayList<>(Arrays.*asList*(taskCol, allTimeCol,  
 completedCol, descriptionCol));  
 }  
  
 public void removeData() {  
 data = FXCollections.*observableArrayList*();  
 tableView.setItems(data);  
 }  
  
 private int getDayNumber(String day) {  
 int dayNumber;  
 switch (day) {  
 case "Monday" -> dayNumber = 0;  
 case "Tuesday" -> dayNumber = 1;  
 case "Wednesday" -> dayNumber = 2;  
 case "Thursday" -> dayNumber = 3;  
 case "Friday" -> dayNumber = 4;  
 case "Saturday" -> dayNumber = 5;  
 case "Sunday" -> dayNumber = 6;  
 default -> dayNumber = -1;  
 }  
 return dayNumber;  
 }  
  
 public void refreshData() {  
 for (Task task :data) {  
 double completed = Double.*parseDouble*(task.getCompleted());  
 double allTime = Double.*parseDouble*(task.getAllTime());  
 double timeLeft = allTime \* (1 - completed / 100);  
 task.setTimeLeft(String.*valueOf*(timeLeft));  
 }  
 tableView.refresh();  
 }  
  
 public static boolean isNumeric(String str) {  
 try {  
 Double.*parseDouble*(str);  
 return true;  
 } catch(NumberFormatException e){  
 return false;  
 }  
 }  
}

**Class The Task class:**

package com.example.workorganizer;  
  
import javafx.beans.property.SimpleStringProperty;  
  
public class Task {  
  
 private WeekDays weekDay;  
 private final SimpleStringProperty day;  
 private final SimpleStringProperty task;  
 private final SimpleStringProperty allTime;  
 private final SimpleStringProperty completed;  
 private final SimpleStringProperty timeLeft;  
 private final SimpleStringProperty description;  
  
 public Task(String day, String task, String allTime, String completed, String timeLeft, String description) {  
 this.day = new SimpleStringProperty(day);  
 this.task = new SimpleStringProperty(task);  
 this.allTime = new SimpleStringProperty(allTime);  
 this.completed = new SimpleStringProperty(completed);  
 this.timeLeft = new SimpleStringProperty(timeLeft);  
 this.description = new SimpleStringProperty(description);  
 }  
  
 Task(String day, String task, String allTime, String completed, String description) {  
 this.day = new SimpleStringProperty(day);  
 this.task = new SimpleStringProperty(task);  
 this.allTime = new SimpleStringProperty(allTime);  
 this.completed = new SimpleStringProperty(completed);  
 this.timeLeft = new SimpleStringProperty("-1");  
 this.description = new SimpleStringProperty(description);  
 }  
  
  
 public Task() {  
 this.day = new SimpleStringProperty("Enter day");  
 this.task = new SimpleStringProperty("Enter task");  
 this.allTime = new SimpleStringProperty("0");  
 this.completed = new SimpleStringProperty("0");  
 this.timeLeft = new SimpleStringProperty("Time left");  
 this.description = new SimpleStringProperty("Specific description");  
 }  
  
 public String getDay() {  
 return day.get();  
 }  
  
 public void setDay(String day) {  
 this.day.set(day);  
 }  
  
 public String getTask() {  
 return task.get();  
 }  
  
 public void setTask(String task) {  
 this.task.set(task);  
 }  
  
 public String getAllTime() {  
 return allTime.get();  
 }  
  
 public void setAllTime(String allTime) {  
 this.allTime.set(allTime);  
 }  
  
 public String getCompleted() {  
 return completed.get();  
 }  
  
 public void setCompleted(String completed) {  
 this.completed.set(completed);  
 }  
  
 public String getTimeLeft() {  
 return timeLeft.get();  
 }  
  
 public void setTimeLeft(String timeLeft) {  
 this.timeLeft.set(timeLeft);  
 }  
  
 public String getDescription() {  
 return description.get();  
 }  
  
 public void setDescription(String deadline) {  
 this.description.set(deadline);  
 }  
  
}

**AddTaskPanel class:**

package com.example.workorganizer;  
  
import javafx.geometry.Insets;  
import javafx.scene.control.Button;  
import javafx.scene.control.ComboBox;  
import javafx.scene.control.TextField;  
import javafx.scene.layout.HBox;  
  
public class AddTaskPanel {  
 private final HBox userEntryHBox = new HBox();  
  
 public AddTaskPanel(TableChronology tableChronology) {  
 userEntryHBox.setSpacing(8);  
 userEntryHBox.setPadding(new Insets(10, 10, 10, 10));  
  
 SetDayForTaskList addDay = new SetDayForTaskList();  
 ComboBox<WeekDays> addDayComboBox = addDay.getComboBox();  
  
 TextField addTask = new TextField();  
 TextField addAllTime = new TextField();  
 TextField addCompleted = new TextField();  
 TextField addDescription = new TextField();  
  
 addDayComboBox.setValue(WeekDays.*MONDAY*);  
 addTask.setText("Enter task");  
 addAllTime.setText("Enter all time");  
 addCompleted.setText("Completed");  
 addDescription.setText("Add specific descriptions");  
  
 addTask.setPrefWidth(170);  
 addDayComboBox.setPrefWidth(100);  
 addAllTime.setPrefWidth(100);  
 addCompleted.setPrefWidth(100);  
 addDescription.setPrefWidth(200);  
  
 Button addButton = new Button("Add");  
 addButton.setOnAction(e -> {  
 if (!TableChronology.*isNumeric*(addAllTime.getText())){  
 Warning wrongValueCompleted = new Warning("Wrong value of all time");  
 wrongValueCompleted.show();  
 return;  
 }  
  
 double addAllTimeDouble = Double.*parseDouble*(addAllTime.getText());  
 if (addAllTimeDouble < 0) {  
 Warning wrongValueCompleted = new Warning("Wrong value of all time");  
 wrongValueCompleted.show();  
 return;  
 }  
  
 if (!TableChronology.*isNumeric*(addCompleted.getText())){  
 Warning wrongValueCompleted = new Warning("Wrong value of completed");  
 wrongValueCompleted.show();  
 return;  
 }  
  
 double addCompletedDouble = Double.*parseDouble*(addCompleted.getText());  
 if (addCompletedDouble < 0 || addCompletedDouble > 100) {  
 Warning wrongValueCompleted = new Warning("Wrong value of completed");  
 wrongValueCompleted.show();  
 return;  
 }  
  
 tableChronology.getData().add(new Task(  
 addDayComboBox.getValue().toString(),  
 addTask.getText(),  
 addAllTime.getText(),  
 addCompleted.getText(),  
 addDescription.getText()));  
 addDayComboBox.setValue(WeekDays.*MONDAY*);  
 addTask.setText("Enter task");  
 addAllTime.setText("Enter all time");  
 addCompleted.setText("Completed");  
 addDescription.setText("Add specific descriptions");  
 tableChronology.refreshData();  
 });  
  
 Button removeButton = new Button("Remove");  
 removeButton.setOnAction(e -> {  
 Task selectedItem = tableChronology.getTableView().getSelectionModel().getSelectedItem();  
 tableChronology.getData().remove(selectedItem);  
 });  
  
 userEntryHBox.getChildren().addAll(addDayComboBox, addTask, addAllTime, addCompleted,  
 addDescription, addButton, removeButton);  
 }  
  
 public HBox getUserEntryHBox() {  
 return userEntryHBox;  
 }  
}

}

**OrganizeTastPanel class :**

package com.example.workorganizer;  
  
import javafx.geometry.Insets;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.HBox;  
import javafx.stage.Stage;  
  
public class OrganizeTaskPanel {  
 private int resolutionChanger = 1;  
 private final HBox organizeTaskHBox;  
 public OrganizeTaskPanel (Stage stage, Scene scene, TableChronology tableChronologyAddTasksScene,  
 TableChronology tableChronologyMainScene, LabelView labelView) {  
 Button organizeTaskButton = new Button("Organize Tasks");  
 organizeTaskButton.setOnAction(actionEvent -> {  
 labelView.refreshQuoteLabel();  
 stage.setScene(scene);  
 resolutionChanger \*= (-1);  
 stage.setWidth(stage.getWidth() + resolutionChanger);  
 TransferTasks.*transfer*(tableChronologyAddTasksScene, tableChronologyMainScene);  
 tableChronologyMainScene.refreshData();  
 tableChronologyAddTasksScene.removeData();  
 });  
  
 Button addButton = new Button("Add task");  
 addButton.setOnAction(e -> tableChronologyAddTasksScene.getData().add(new Task()));  
  
 this.organizeTaskHBox = new HBox(organizeTaskButton, addButton);  
 organizeTaskHBox.setSpacing(8);  
 organizeTaskHBox.setPadding(new Insets(10, 10, 10, 10));  
 }  
  
 public HBox getOrganizeTaskHBox() {  
 return organizeTaskHBox;  
 }  
}

**SetDayForTaskList class :**

package com.example.workorganizer;  
  
import java.util.Arrays;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.scene.control.ComboBox;  
  
public class SetDayForTaskList {  
 public ComboBox<WeekDays> setDayForTask;  
 public ObservableList<WeekDays> weekDays;  
  
 public SetDayForTaskList() {  
 weekDays = FXCollections.*observableArrayList*();  
 weekDays.addAll(Arrays.*asList*(WeekDays.*values*()));  
 setDayForTask = new ComboBox<>(weekDays);  
 }  
  
 public ComboBox<WeekDays> getComboBox() {  
 return setDayForTask;  
 }  
}

**Класс TransferTasks class**

package com.example.workorganizer;  
  
import java.util.Comparator;  
import java.util.HashMap;  
import java.util.Map;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
  
public class TransferTasks {  
 public static void transfer(TableChronology fromTableChron, TableChronology toTableChron) {  
 ObservableList<Task> fromList = fromTableChron.getData();  
 ObservableList<Task> toList = FXCollections.*observableArrayList*();  
 HashMap<WeekDays, Integer> workload = new HashMap<>();  
 for (WeekDays weekDay : WeekDays.*getValues*()){  
 workload.put(weekDay, 0);  
 }  
  
 Comparator<Task> comparatorOfTime = (task1, task2) -> {  
 int workload1 = Integer.*parseInt*(task1.getAllTime());  
 int workload2 = Integer.*parseInt*(task2.getAllTime());  
 return workload2 - workload1;  
 };  
  
 fromList.sort(comparatorOfTime);  
  
 for (Task task :fromList) {  
 *insertLessWorkload*(task, toList, workload);  
 }  
  
 toTableChron.getData().setAll(toList);  
 }  
  
 private static void insertLessWorkload(Task task, ObservableList<Task> toList, HashMap<WeekDays, Integer> workload) {  
 WeekDays lessWorkloadDayNumber = *findLessWorkloadDay*(workload);  
 task.setDay(lessWorkloadDayNumber.toString());  
 toList.add(task);  
 Integer lessWorkloadDayWorkload = workload.get(lessWorkloadDayNumber);  
 int taskWorkload = Integer.*parseInt*(task.getAllTime());  
 if (lessWorkloadDayWorkload == null) {  
 lessWorkloadDayWorkload = taskWorkload;  
 } else {  
 lessWorkloadDayWorkload += taskWorkload;  
 }  
 workload.put(lessWorkloadDayNumber, lessWorkloadDayWorkload);  
 }  
  
 private static WeekDays findLessWorkloadDay(HashMap<WeekDays, Integer> workload) {  
 WeekDays lessWorkloadDay = null;  
 Integer lessWorkloadDayWorkload = 0;  
 for (Map.Entry<WeekDays, Integer> day: workload.entrySet()) {  
 if (lessWorkloadDay == null) {  
 lessWorkloadDay = day.getKey();  
 lessWorkloadDayWorkload = day.getValue();  
 continue;  
 }  
  
 if (lessWorkloadDayWorkload > day.getValue()) {  
 lessWorkloadDay = day.getKey();  
 lessWorkloadDayWorkload = day.getValue();  
 }  
 }  
 return lessWorkloadDay;  
 }  
}

**Класс Warning Class**

package com.example.workorganizer;  
  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.VBox;  
import javafx.scene.text.Text;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
  
public class Warning {  
 private final Stage dialogStage;  
 public Warning(String wrongMessage) {  
 dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
  
 Button okButton = new Button("Ok.");  
 okButton.setOnAction(actionEvent -> dialogStage.close());  
 VBox vbox = new VBox(new Text(wrongMessage), okButton);  
 vbox.setAlignment(Pos.*CENTER*);  
 vbox.setSpacing(7);  
 vbox.setPadding(new Insets(15));  
  
 dialogStage.setMinWidth(250);  
 dialogStage.setMinWidth(250);  
 dialogStage.setScene(new Scene(vbox));  
 }  
  
 public void show() {  
 dialogStage.show();  
 }  
}

**Класс WeekDays class**

package com.example.workorganizer;  
  
public enum WeekDays {  
 *MONDAY*,  
 *TUESDAY*,  
 *WEDNESDAY*,  
 *THURSDAY*,  
 *FRIDAY*,  
 *SATURDAY*,  
 *SUNDAY*;  
  
 public static WeekDays getWeekDay(int number) {  
 WeekDays weekDay = null;  
 switch (number){  
 case 0 -> weekDay = *MONDAY*;  
 case 1 -> weekDay = *TUESDAY*;  
 case 2 -> weekDay = *WEDNESDAY*;  
 case 3 -> weekDay = *THURSDAY*;  
 case 4 -> weekDay = *FRIDAY*;  
 case 5 -> weekDay = *SATURDAY*;  
 case 6 -> weekDay = *SUNDAY*;  
 }  
 return weekDay;  
 }  
  
 @Override  
 public String toString() {  
 switch (this) {  
 case *MONDAY* -> {  
 return "Monday";  
 }  
 case *TUESDAY* -> {  
 return "Tuesday";  
 }  
 case *WEDNESDAY* -> {  
 return "Wednesday";  
 }  
 case *THURSDAY* -> {  
 return "Thursday";  
 }  
 case *FRIDAY* -> {  
 return "Friday";  
 }  
 case *SATURDAY* -> {  
 return "Saturday";  
 }  
 case *SUNDAY* -> {  
 return "Sunday";  
 }  
 }  
 return null;  
 }  
  
 public static WeekDays[] getValues(){  
 WeekDays[] weekDays = new WeekDays[7];  
 weekDays[0] = *MONDAY*;  
 weekDays[1] = *THURSDAY*;  
 weekDays[2] = *WEDNESDAY*;  
 weekDays[3] = *TUESDAY*;  
 weekDays[4] = *FRIDAY*;  
 weekDays[5] = *SATURDAY*;  
 weekDays[6] = *SUNDAY*;  
 return weekDays;  
 }  
}