Fitness Tracker Presentation

By Rahshon Murff (Group 18)

Functional Specification

The Fitness Tracker application allows users to log workouts, track progress toward fitness goals, and manage their workout history.
The system provides a desktop user interface built in **Java Swing**, storing data locally using text or CSV files.

2.1 Functional Requirements

1. The system shall allow the user to add a new workout entry.
2. The system shall allow the user to edit an existing workout entry.
3. The system shall allow the user to delete a workout entry.
4. The system shall record workout attributes including: exercise type, duration, intensity, estimated calories burned
5. The system shall allow the user to view their workout history.
6. The system shall allow filtering of workout history by date or workout type.
7. The system shall allow the user to set a fitness goal (e.g., weekly calories burned).
8. The system shall track the user's progress toward the active fitness goal.
9. The system shall generate a progress summary (weekly or monthly).
10. The system shall export workout history to a CSV file.
11. The system shall save all user data to local storage.
12. The system shall load previously saved data on startup.

2.2 Technical Requirements

1. The system shall use **Java SE** with **Swing** for the graphical user interface.

2. The system shall store data locally using structured text or CSV files.

3. The system shall use a modular object-oriented class structure.

4. The system shall validate all user input and provide error handling.

Use Cases

UC-1: User Logs Workout

Primary Actor: User
Goal: Record details of a workout session.

Main Scenario

1. User selects **“Add Workout”** from the menu or main screen.
2. System displays the **Workout Entry Form**.
3. User enters the exercise type, duration, and intensity.
4. System calculates estimated calories burned.
5. User confirms the entry.
   System saves the workout.
6. System displays a confirmation message.

Walkthrough Use Cases

WT-1: User Logs Workout

Objects Involved

- **Actor:** User
- **View:** WorkoutEntryView (JPanel with text fields, combo box, “Save” button)
- **Controller:** WorkoutEntryController (implements ActionListener)
- **Model:** Workout, WorkoutLog

Walkthrough Steps

1. User clicks the **“Add Workout”** button in MainView.
2. MainView notifies WorkoutEntryController via `actionPerformed()`.
3. WorkoutEntryController instructs MainView to display WorkoutEntryView.
4. User enters the workout type, duration, and intensity into the UI fields.
5. User clicks **“Save”**.
6. WorkoutEntryController reads values from WorkoutEntryView using getters.
7. Controller creates a new **Workout** object with the input data.

8. Controller calls `WorkoutLog.addWorkout(workout)` to store the workout.
9. WorkoutLog adds the workout to its internal list.
10. Controller instructs MainView to refresh the workout list display.
11. System shows a confirmation message.

UC-2: User Views Workout History

Primary Actor: User
Goal: View previously recorded workouts.

Main Scenario

1. User selects **“View History”** from the menu.
2. System displays the list of workouts.
3. User applies a filter (date or type).

4. System refreshes the list based on the filter.
5. User selects a workout to view details.
6. System displays the selected workout information.

Walkthrough Use Cases

WT-2: User Views Workout History

Objects Involved

- **Actor:** User
- **View:** HistoryView (JPanel with table, filter fields)
- **Controller:** HistoryController (ActionListener, ListSelectionListener)
- **Model:** WorkoutLog

Walkthrough Steps

1. User clicks **“View History”** in MainView.
2. MainView notifies HistoryController via `actionPerformed()`.
3. HistoryController instructs MainView to display HistoryView.
4. HistoryView requests workout list from HistoryController.
5. HistoryController calls `WorkoutLog.getAllWorkouts()`.
6. Model returns list; HistoryController passes it to HistoryView.
7. HistoryView displays the list in a table.

8. User provides filter criteria (type or date).
9. User clicks **“Apply Filter”**.
10. HistoryController reads filter fields and requests filtered results.
11. Controller calls model methods like `WorkoutLog.filterByDate()` or `filterByType()`.
12. Controller sends updated list back to HistoryView.
13. HistoryView refreshes the table display.
14. User clicks an individual workout row.
15. HistoryController receives a selection event and instructs HistoryView to display details.

UC-3: User Sets Fitness Goal

Primary Actor: User
Goal: Create or update a fitness goal.

Main Scenario

1. User selects **“Set Goal”**.
2. System displays the goal setup form.
3. User enters target metrics (e.g., weekly calories).
4. User confirms the goal.
5. System saves the goal.
6. System displays confirmation.

Walkthrough Use Cases

WT-3: User Sets Fitness Goal

Objects Involved

- **Actor:** User
- **View:** GoalView (JPanel with numeric field + “Save Goal” button)

- **Controller:** GoalController (ActionListener)
- **Model:** Goal, GoalTracker

Walkthrough Steps

1. User clicks **“Set Goal”** from MainView.
2. MainView triggers `actionPerformed()` in GoalController.
3. GoalController displays GoalView.
4. User enters the target calories or duration goal.
5. User clicks **“Save Goal.”**
6. GoalController retrieves the input using GoalView getters.
7. Controller creates a new **Goal** object.
8. Controller calls `GoalTracker.setGoal(goal)`.
9. Model stores the new goal.
10. Controller refreshes MainView to show updated goal status.
11. System displays confirmation.

UC-4: User Edits Workout Entry

Primary Actor: User
Goal: Modify a previously saved workout.

Main Scenario

1. User selects a workout from the history list.
2. System displays workout details.
3. User selects **“Edit”**.
4. System displays the editable workout form.
5. User updates the fields.
6. User confirms changes.
7. System updates and saves the workout.
8. System displays confirmation.

Walkthrough Use Cases

WT-4: User Edits Workout Entry

Objects Involved

- **Actor:** User
- **View:** HistoryView, WorkoutEditView
- **Controller:** HistoryController, WorkoutEditController
- **Model:** Workout, WorkoutLog

Walkthrough Steps

1. User opens **HistoryView** via “View History.”
2. User selects a workout row from the table.
3. HistoryController receives selection event and shows WorkoutEditView.
4. WorkoutEditView displays the workout fields pre-filled.
5. User edits the workout information.
6. User clicks **“Save Changes.”**
7. WorkoutEditController reads the edited fields.
8. Controller updates the corresponding **Workout** object.
9. Controller calls `WorkoutLog.updateWorkout(updatedWorkout)`.
10. WorkoutLog updates its internal list.
11. Controller instructs HistoryView to refresh the table.
12. System displays confirmation.

UC-5: User Deletes Workout Entry

Primary Actor: User
Goal: Remove a workout from the history.

Main Scenario

1. User selects a workout from the history list.
2. System displays the workout details.
3. User selects **“Delete”**.
4. System asks for confirmation.
5. User confirms.
6. System deletes the workout.
7. System updates the history list.

Walkthrough Use Cases

WT-5: User Deletes Workout Entry

Objects Involved

- **Actor:** User
- **View:** HistoryView (table + delete button), ConfirmationDialog
- **Controller:** HistoryController
- **Model:** WorkoutLog

Walkthrough Steps

1. User opens **HistoryView**.
2. User selects a workout from the table.
3. User clicks **"Delete."**
4. HistoryController receives event via `actionPerformed()`.
5. System shows ConfirmationDialog.
6. User clicks **"Confirm."**
7. HistoryController calls `WorkoutLog.removeWorkout(workoutId)`.
8. Model deletes the entry.
9. Controller updates HistoryView by refreshing the workout list.
10. System displays deletion confirmation.

UC-6: User Exports Data

Primary Actor: User
Goal: Create a CSV export of workout history.

Main Scenario

1. User selects **"Export Data"**.
2. System prompts for file name/location.
3. User confirms.
4. System generates and saves a CSV file.
5. System displays success message.

Walkthrough Use Cases

WT-6: User Exports Workout Data

Objects Involved

- **Actor:** User
- **View:** ExportDialog (file chooser)
- **Controller:** ExportController
- **Model:** WorkoutLog, CSVExporter

Walkthrough Steps

1. User clicks **“Export Data.”**
2. ExportController receives event.
3. System opens ExportDialog.
4. User selects file name and location.
5. ExportController calls `WorkoutLog.getAllWorkouts()`.
6. Controller sends data to `CSVExporter.exportToCSV(workouts, filePath)`.
7. CSVExporter writes the file.
8. ExportController shows success message.

UC-7: User Views Progress Summary

Primary Actor: User
Goal: View weekly or monthly progress.

Main Scenario

1. User selects **“Progress Summary”**.
2. System calculates progress from stored workouts.
3. System displays the summary as a chart or table.

Walkthrough Use Cases

WT-7: User Views Progress Summary

Objects Involved

- **Actor:** User
- **View:** SummaryView (chart or table)
- **Controller:** SummaryController
- **Model:** WorkoutLog, GoalTracker

Walkthrough Steps

1. User clicks **“Progress Summary.”**
2. SummaryController receives the event.
3. Controller calls `WorkoutLog.getWorkoutsForRange(week/month)`.
4. Controller retrieves the active goal from `GoalTracker.getCurrentGoal()`.
5. Controller calculates progress metrics.
6. Controller passes data to SummaryView.
7. SummaryView displays charts/tables summarizing progress.

UC-8: User Saves or Loads Data

Primary Actor: User
Goal: Persist application data.

Main Scenario

1. System loads saved data on startup.
2. User interacts with the application.
3. User selects **“Save Data”** (optional).
4. System writes data to storage.
5. System confirms completion.

Walkthrough Use Cases

WT-8: System Saves and Loads Data

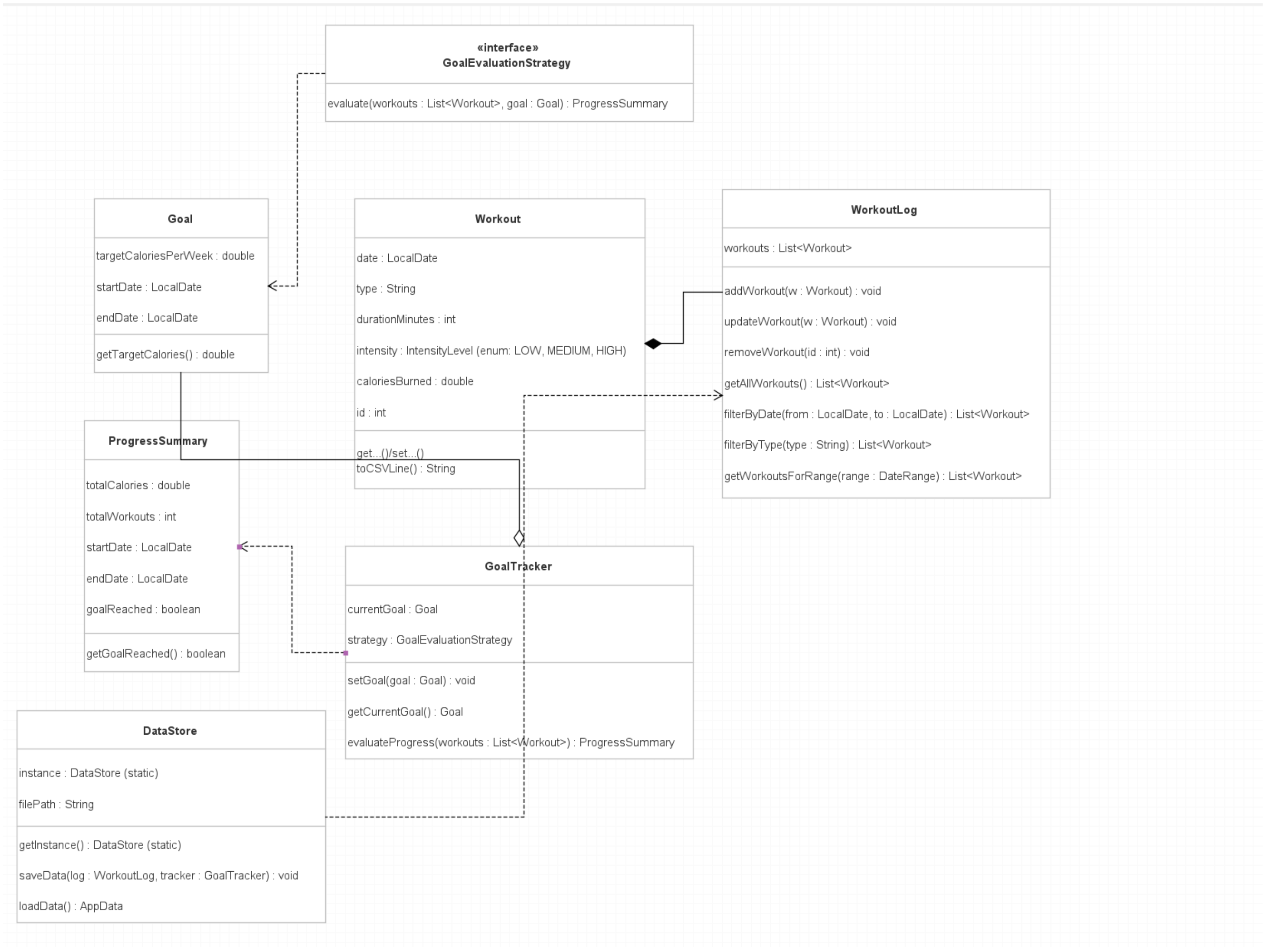
Objects Involved

- **Actor:** User
- **View:** MainView (optional “Save” button)
- **Controller:** AppController or FileController
- **Model:** WorkoutLog, GoalTracker, DataStore

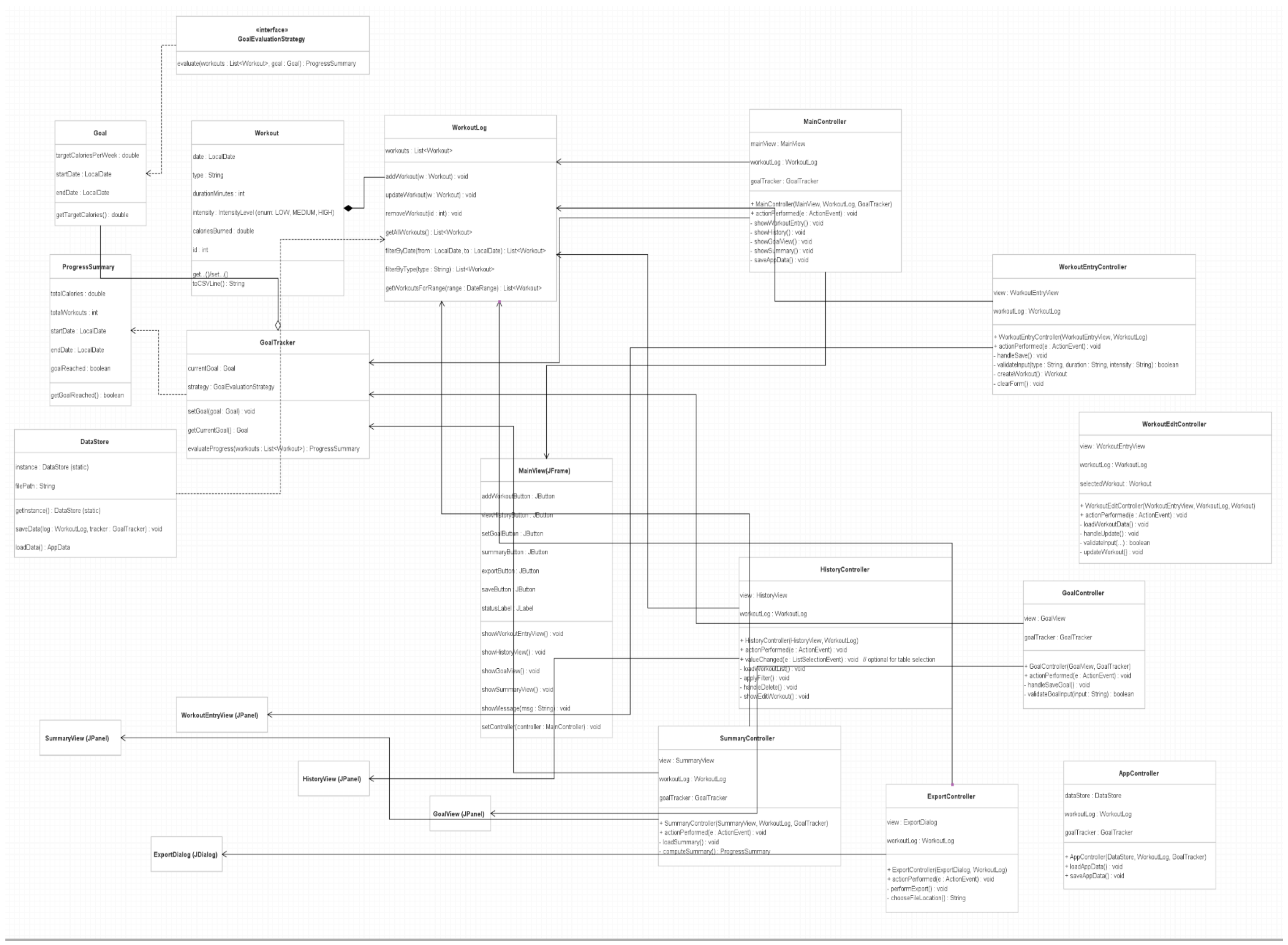
Walkthrough Steps

1. On application startup, AppController calls `DataStore.loadData()`.
2. DataStore loads workouts and goals from the save file.
3. Model objects populate internal lists.
4. MainView displays loaded data.
5. During use, User optionally clicks **“Save Data.”**
6. FileController receives the event.
7. FileController gathers data from WorkoutLog and GoalTracker.
8. FileController calls `DataStore.saveData(allData)`.
9. DataStore writes data to local storage.

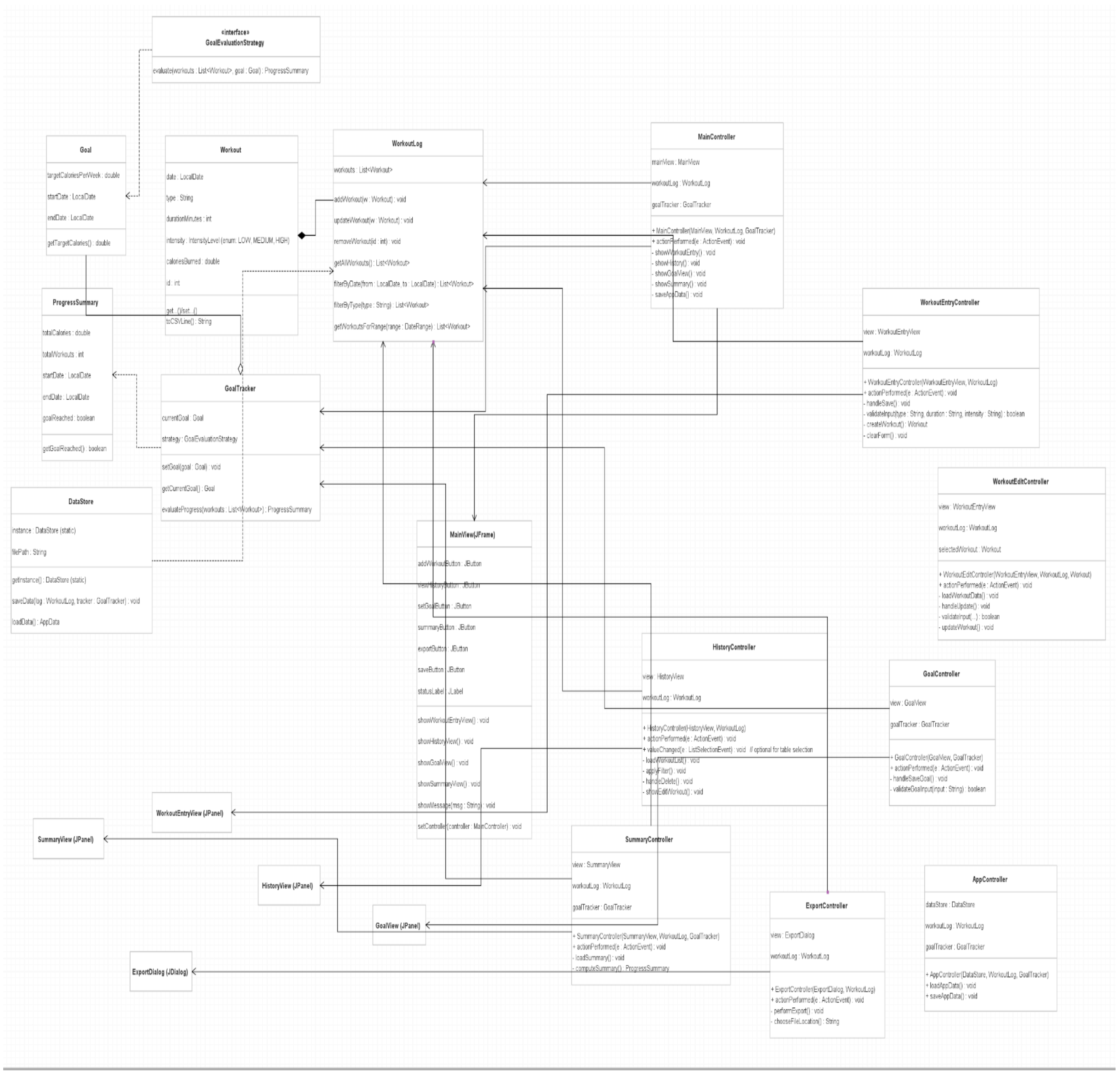
Model Classes


«interface»
GoalEvaluationStrategy
evaluate(workouts : List<Workout>, goal : Goal) : ProgressSummary
Goal
targetCaloriesPerWeek : double
startDate : LocalDate
endDate : LocalDate
getTargetCalories() : double
Workout
date : LocalDate
type : String
durationMinutes : int
intensity : IntensityLevel (enum: LOW, MEDIUM, HIGH)
caloriesBurned : double
id : int
get...()/set...()
toCSVLine() : String
WorkoutLog
workouts : List<Workout>
addWorkout(w : Workout) : void
updateWorkout(w : Workout) : void
removeWorkout(id : int) : void
getAllWorkouts() : List<Workout>
filterByDate(from : LocalDate, to : LocalDate) : List<Workout>
filterByType(type : String) : List<Workout>
getWorkoutsForRange(range : DateRange) : List<Workout>
ProgressSummary
totalCalories : double
totalWorkouts : int
startDate : LocalDate
endDate : LocalDate
goalReached : boolean
getGoalReached() : boolean
GoalTracker
currentGoal : Goal
strategy : GoalEvaluationStrategy
setGoal(goal : Goal) : void
getCurrentGoal() : Goal
evaluateProgress(workouts : List<Workout>) : ProgressSummary
DataStore
instance : DataStore (static)
filePath : String
getInstance() : DataStore (static)
saveData(log : WorkoutLog, tracker : GoalTracker) : void
loadData() : AppData

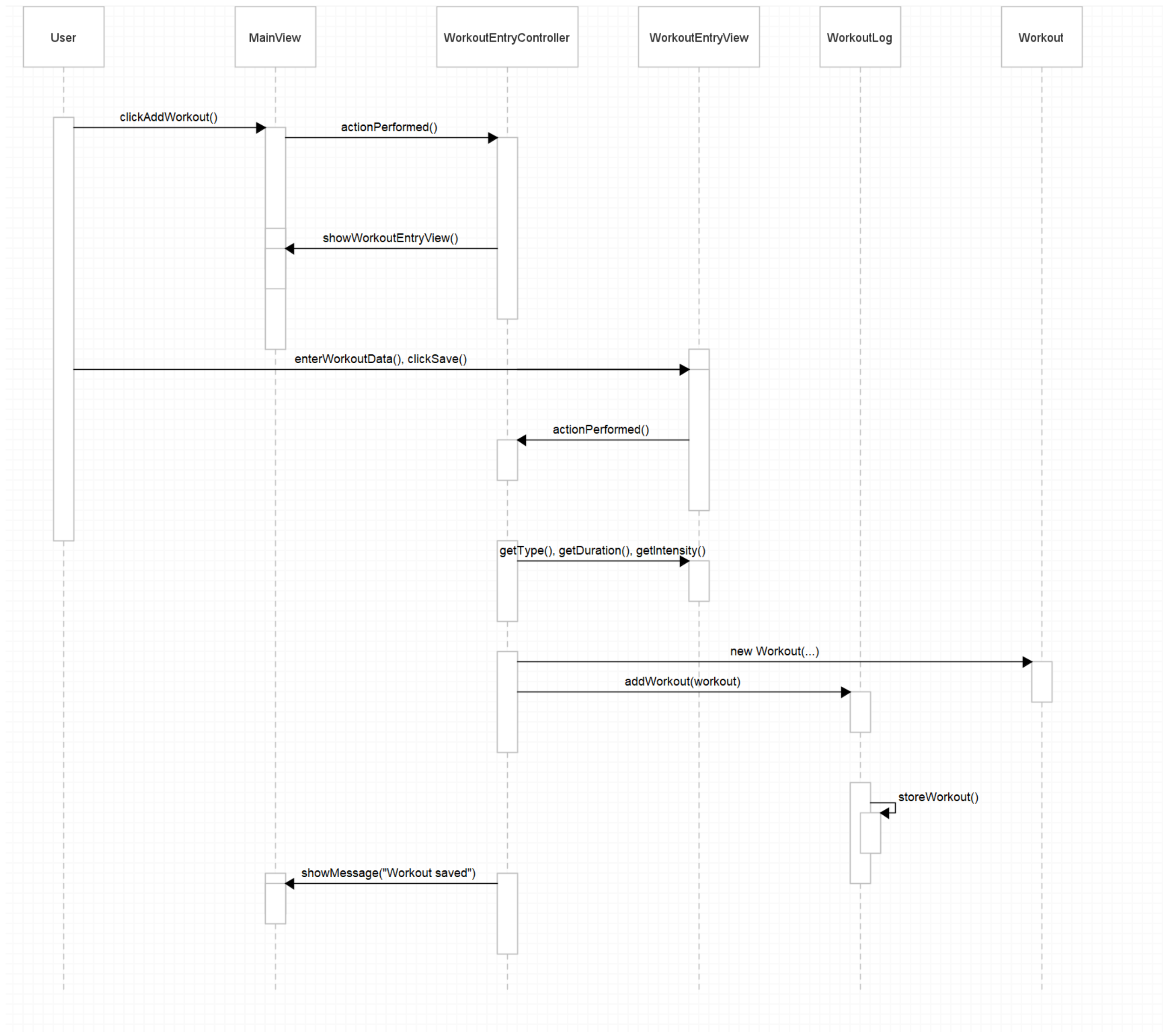
View Classes


«interface»
GoalEvaluationStrategy
evaluate(workouts : List<Workout>, goal : Goal) : ProgressSummary
Goal
targetCaloriesPerWeek : double
startDate : LocalDate
endDate : LocalDate
getTargetCalories() : double
Workout
date : LocalDate
type : String
durationMinutes : int
intensity : IntensityLevel (enum: LOW, MEDIUM, HIGH)
caloriesBurned : double
id : int
get...()/set...()
toCSVLine() : String
WorkoutLog
workouts : List<Workout>
addWorkout(w : Workout) : void
updateWorkout(w : Workout) : void
removeWorkout(id : int) : void
getAllWorkouts() : List<Workout>
filterByDate(from : LocalDate, to : LocalDate) : List<Workout>
filterByType(type : String) : List<Workout>
getWorkoutsForRange(range : DateRange) : List<Workout>
MainController
mainView : MainView
workoutLog : WorkoutLog
goalTracker : GoalTracker
+ MainController(MainView, WorkoutLog, GoalTracker)
+ actionPerformed(e : ActionEvent) : void
- showWorkoutEntry() : void
- showHistory() : void
- showGoalView() : void
- showSummary() : void
- saveAppData() : void
ProgressSummary
totalCalories : double
totalWorkouts : int
startDate : LocalDate
endDate : LocalDate
goalReached : boolean
getGoalReached() : boolean
GoalTracker
currentGoal : Goal
strategy : GoalEvaluationStrategy
setGoal(goal : Goal) : void
getCurrentGoal() : Goal
evaluateProgress(workouts : List<Workout>) : ProgressSummary
WorkoutEntryController
view : WorkoutEntryView
workoutLog : WorkoutLog
+ WorkoutEntryController(WorkoutEntryView, WorkoutLog)
+ actionPerformed(e : ActionEvent) : void
- handleSave() : void
- validateInput(type : String, duration : String, intensity : String) : boolean
- createWorkout() : Workout
- clearForm() : void
DataStore
instance : DataStore (static)
filePath : String
getInstance() : DataStore (static)
saveData(log : WorkoutLog, tracker : GoalTracker) : void
loadData() : AppData
WorkoutEditController
view : WorkoutEntryView
workoutLog : WorkoutLog
selectedWorkout : Workout
+ WorkoutEditController(WorkoutEntryView, WorkoutLog, Workout)
+ actionPerformed(e : ActionEvent) : void
- loadWorkoutData() : void
- handleUpdate() : void
- validateInput(...) : boolean
- updateWorkout() : void
MainView(JFrame)
addWorkoutButton : JButton
viewHistoryButton : JButton
setGoalButton : JButton
summaryButton : JButton
exportButton : JButton
saveButton : JButton
statusLabel : JLabel
showWorkoutEntryView() : void
showHistoryView() : void
showGoalView() : void
showSummaryView() : void
showMessage(msg : String) : void
setController(controller : MainController) : void
HistoryController
view : HistoryView
workoutLog : WorkoutLog
+ HistoryController(HistoryView, WorkoutLog)
+ actionPerformed(e : ActionEvent) : void
+ valueChanged(e : ListSelectionEvent) : void // optional for table selection
- loadWorkoutList() : void
- applyFilter() : void
- handleDelete() : void
- showEditWorkout() : void
GoalController
view : GoalView
goalTracker : GoalTracker
+ GoalController(GoalView, GoalTracker)
+ actionPerformed(e : ActionEvent) : void
- handleSaveGoal() : void
- validateGoalInput(input : String) : boolean
WorkoutEntryView (JPanel)
SummaryView (JPanel)
SummaryController
view : SummaryView
workoutLog : WorkoutLog
goalTracker : GoalTracker
+ SummaryController(SummaryView, WorkoutLog, GoalTracker)
+ actionPerformed(e : ActionEvent) : void
- loadSummary() : void
- computeSummary() : ProgressSummary
HistoryView (JPanel)
GoalView (JPanel)
ExportDialog (JDialog)
ExportController
view : ExportDialog
workoutLog : WorkoutLog
+ ExportController(ExportDialog, WorkoutLog)
+ actionPerformed(e : ActionEvent) : void
- performExport() : void
- chooseFileLocation() : String
AppController
dataStore : DataStore
workoutLog : WorkoutLog
goalTracker : GoalTracker
+ AppController(DataStore, WorkoutLog, GoalTracker)
+ loadAppData() : void
+ saveAppData() : void

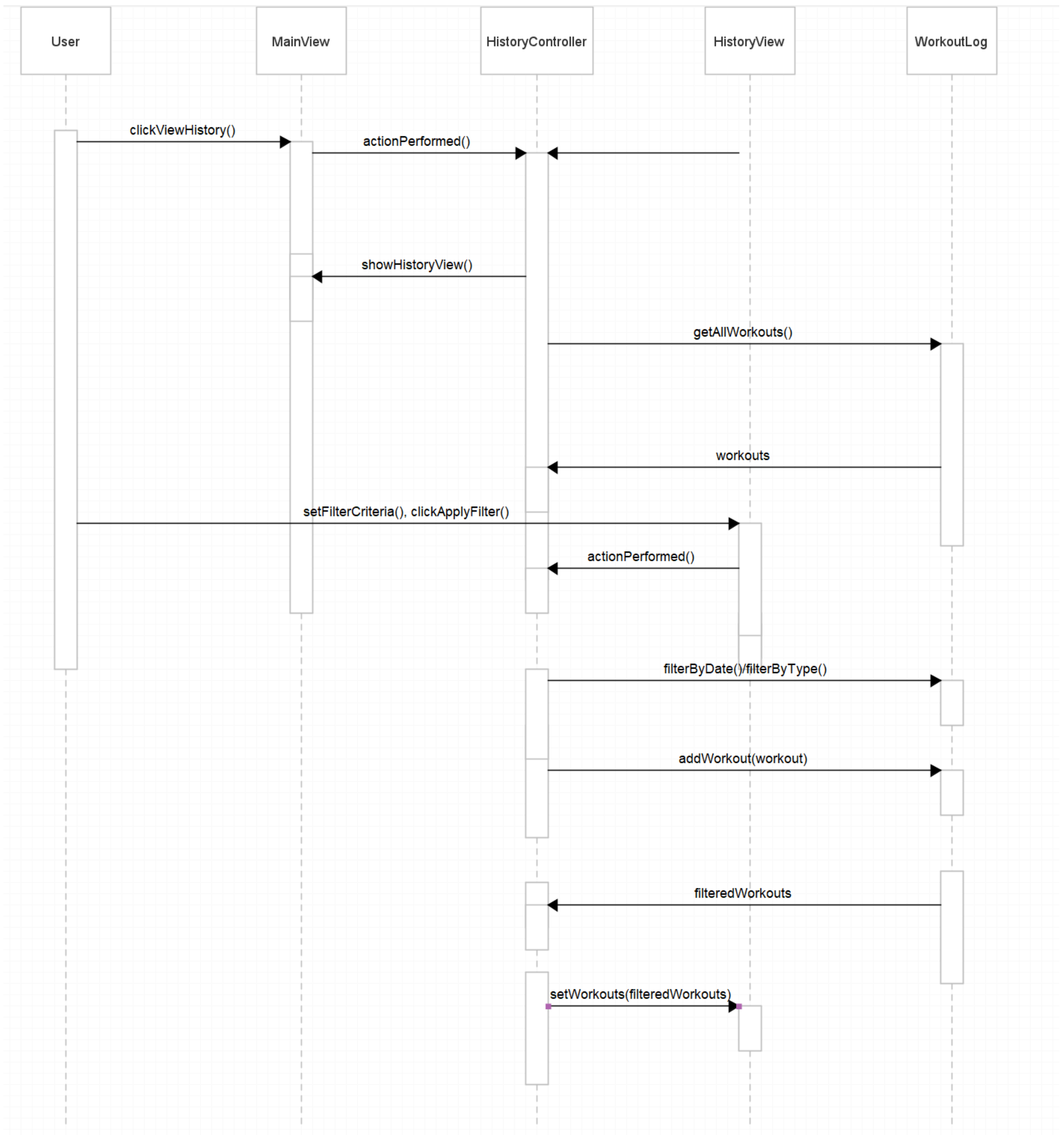
Controller Classes


«interface»
GoalEvaluationStrategy
evaluate(workouts : List<Workout>, goal : Goal) : ProgressSummary
Goal
targetCaloriesPerWeek : double
startDate : LocalDate
endDate : LocalDate
getTargetCalories() : double
Workout
date : LocalDate
type : String
durationMinutes : int
intensity : IntensityLevel (enum: LOW, MEDIUM, HIGH)
caloriesBurned : double
id : int
get...()/set...()
toCSVLine() : String
WorkoutLog
workouts : List<Workout>
addWorkout(w : Workout) : void
updateWorkout(w : Workout) : void
removeWorkout(id : int) : void
getAllWorkouts() : List<Workout>
filterByDate(from : LocalDate, to : LocalDate) : List<Workout>
filterByType(type : String) : List<Workout>
getWorkoutsForRange(range : DateRange) : List<Workout>
MainController
mainView : MainView
workoutLog : WorkoutLog
goalTracker : GoalTracker
+ MainController(MainView, WorkoutLog, GoalTracker)
+ actionPerformed(e : ActionEvent) : void
- showWorkoutEntry() : void
- showHistory() : void
- showGoalView() : void
- showSummary() : void
- saveAppData() : void
ProgressSummary
totalCalories : double
totalWorkouts : int
startDate : LocalDate
endDate : LocalDate
goalReached : boolean
getGoalReached() : boolean
GoalTracker
currentGoal : Goal
strategy : GoalEvaluationStrategy
setGoal(goal : Goal) : void
getCurrentGoal() : Goal
evaluateProgress(workouts : List<Workout>) : ProgressSummary
WorkoutEntryController
view : WorkoutEntryView
workoutLog : WorkoutLog
+ WorkoutEntryController(WorkoutEntryView, WorkoutLog)
+ actionPerformed(e : ActionEvent) : void
- handleSave() : void
- validateInput(type : String, duration : String, intensity : String) : boolean
- createWorkout() : Workout
- clearForm() : void
DataStore
instance : DataStore (static)
filePath : String
getInstance() : DataStore (static)
saveData(log : WorkoutLog, tracker : GoalTracker) : void
loadData() : AppData
WorkoutEditController
view : WorkoutEntryView
workoutLog : WorkoutLog
selectedWorkout : Workout
+ WorkoutEditController(WorkoutEntryView, WorkoutLog, Workout)
+ actionPerformed(e : ActionEvent) : void
- loadWorkoutData() : void
- handleUpdate() : void
- validateInput(...) : boolean
- updateWorkout() : void
MainView(JFrame)
addWorkoutButton : JButton
viewHistoryButton : JButton
setGoalButton : JButton
summaryButton : JButton
exportButton : JButton
saveButton : JButton
statusLabel : JLabel
showWorkoutEntryView() : void
showHistoryView() : void
showGoalView() : void
showSummaryView() : void
showMessage(msg : String) : void
setController(controller : MainController) : void
HistoryController
view : HistoryView
workoutLog : WorkoutLog
+ HistoryController(HistoryView, WorkoutLog)
+ actionPerformed(e : ActionEvent) : void
+ valueChanged(e : ListSelectionEvent) : void // optional for table selection
- loadWorkoutList() : void
- applyFilter() : void
- handleDelete() : void
- showEditWorkout() : void
GoalController
view : GoalView
goalTracker : GoalTracker
+ GoalController(GoalView, GoalTracker)
+ actionPerformed(e : ActionEvent) : void
- handleSaveGoal() : void
- validateGoalInput(input : String) : boolean
WorkoutEntryView (JPanel)
SummaryView (JPanel)
SummaryController
view : SummaryView
workoutLog : WorkoutLog
goalTracker : GoalTracker
+ SummaryController(SummaryView, WorkoutLog, GoalTracker)
+ actionPerformed(e : ActionEvent) : void
- loadSummary() : void
- computeSummary() : ProgressSummary
HistoryView (JPanel)
GoalView (JPanel)
ExportDialog (JDialog)
ExportController
view : ExportDialog
workoutLog : WorkoutLog
+ ExportController(ExportDialog, WorkoutLog)
+ actionPerformed(e : ActionEvent) : void
- performExport() : void
- chooseFileLocation() : String
AppController
dataStore : DataStore
workoutLog : WorkoutLog
goalTracker : GoalTracker
+ AppController(DataStore, WorkoutLog, GoalTracker)
+ loadAppData() : void
+ saveAppData() : void

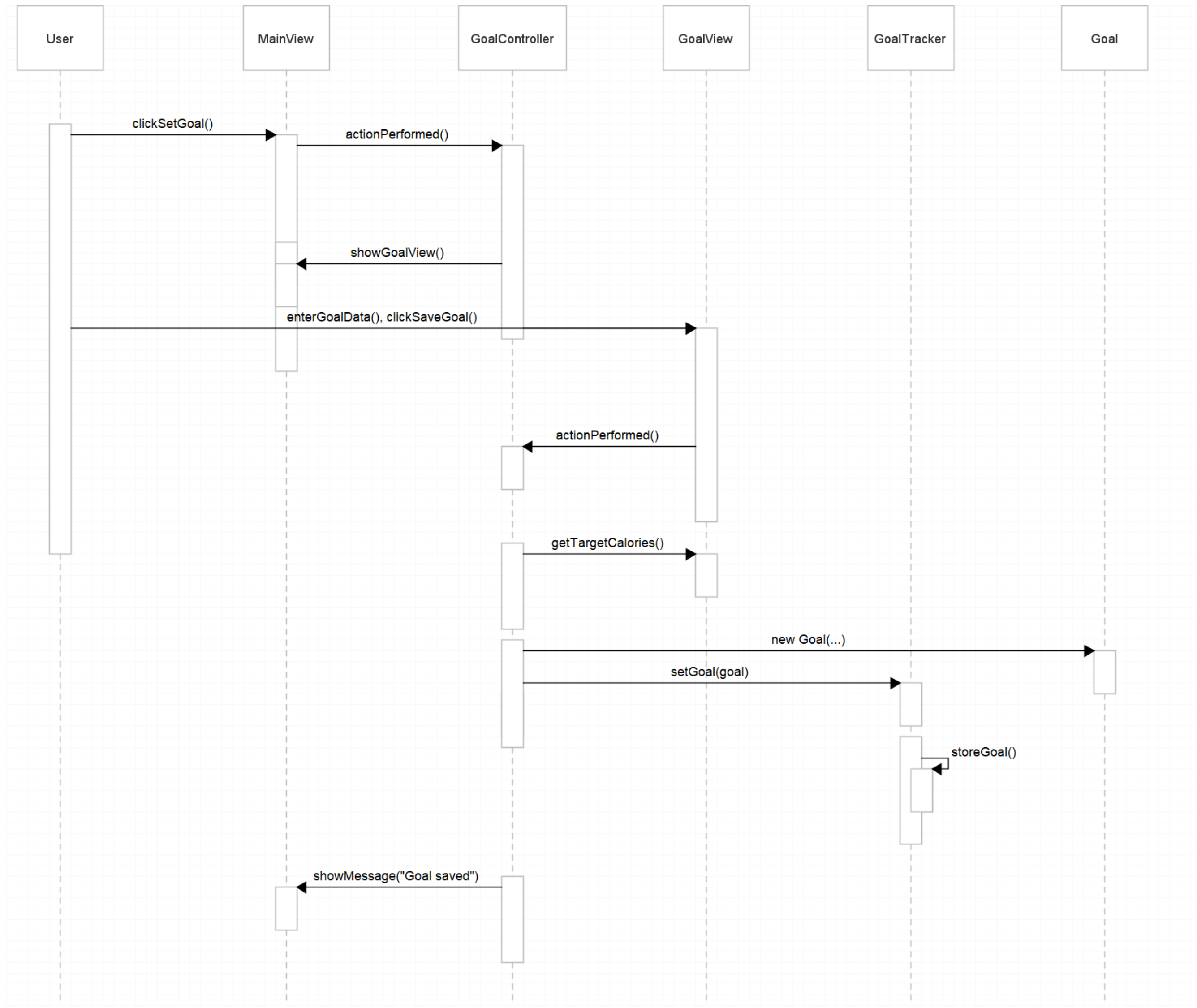
SD-1: User Logs Workout



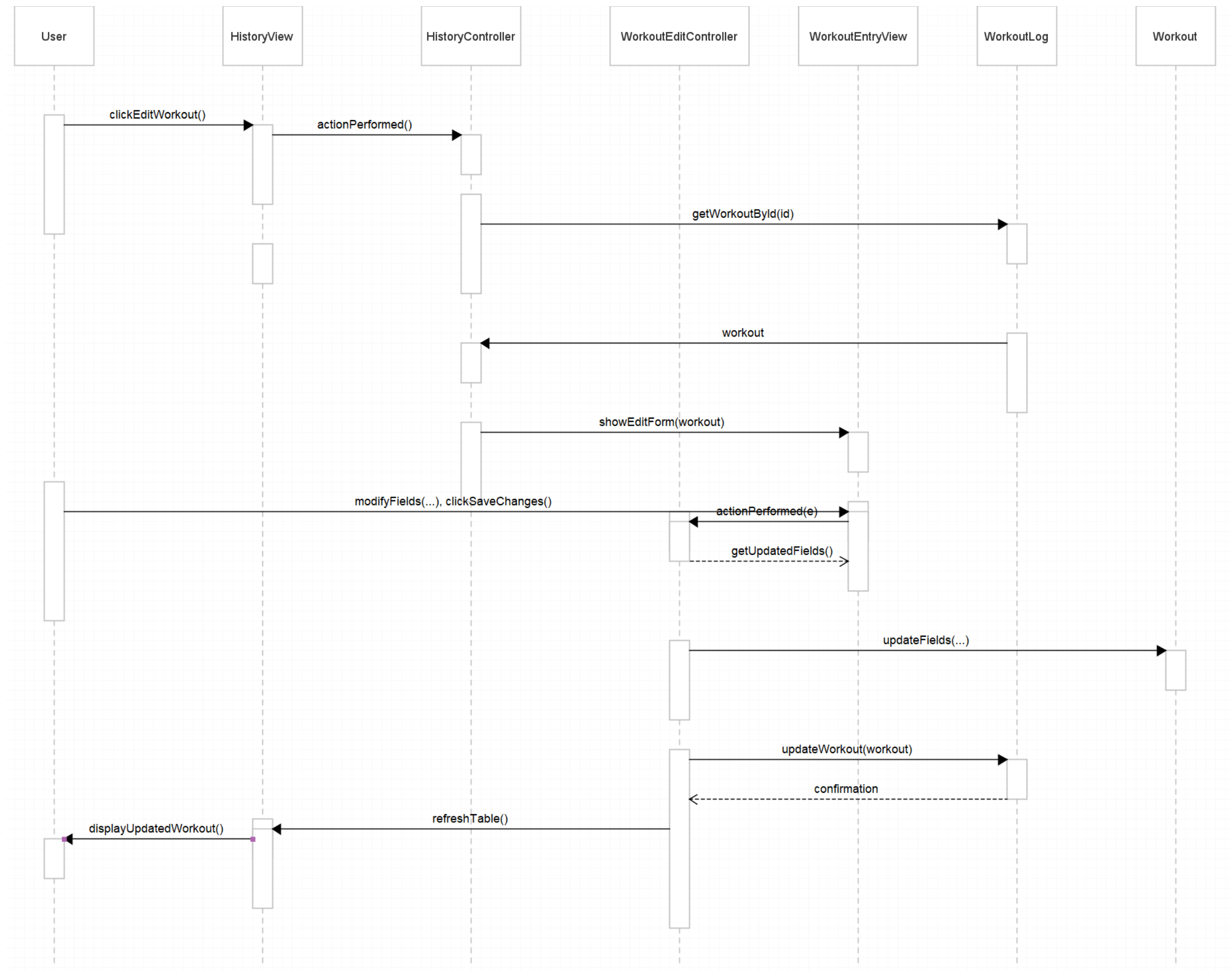
SD-2: User Views Workout History



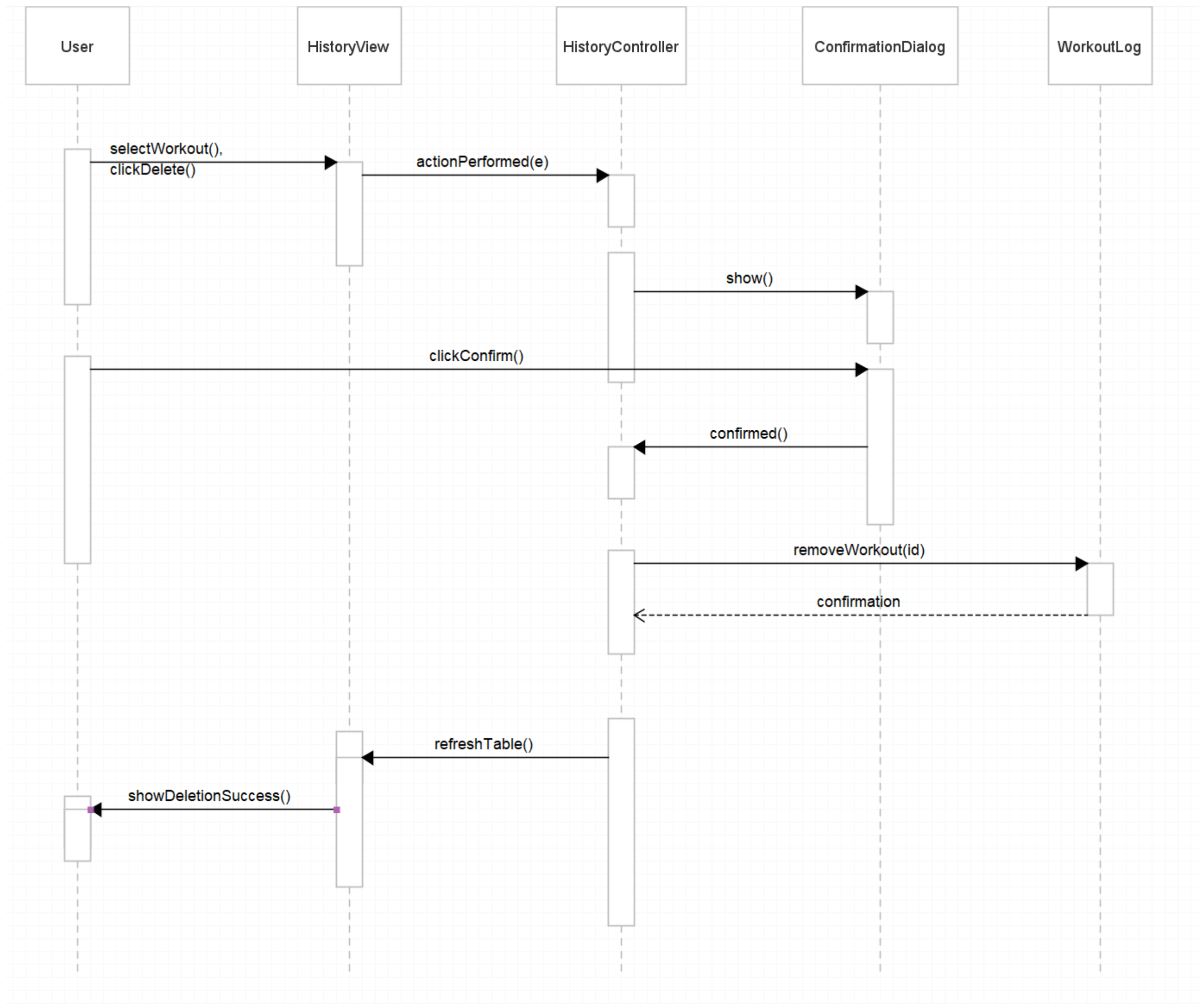
SD-3: User Sets Fitness Goal



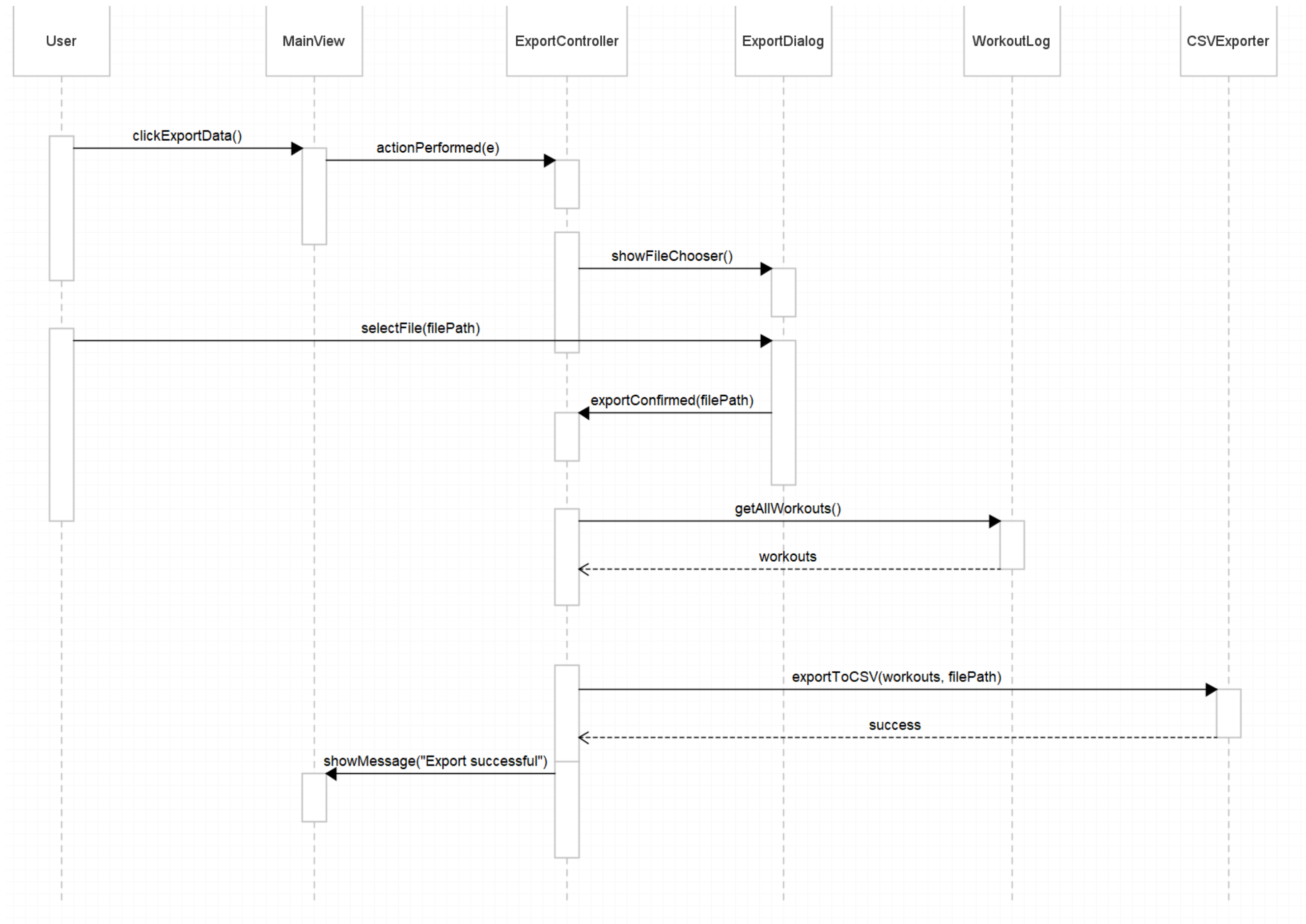
SD-4: User Edits Workout Entry



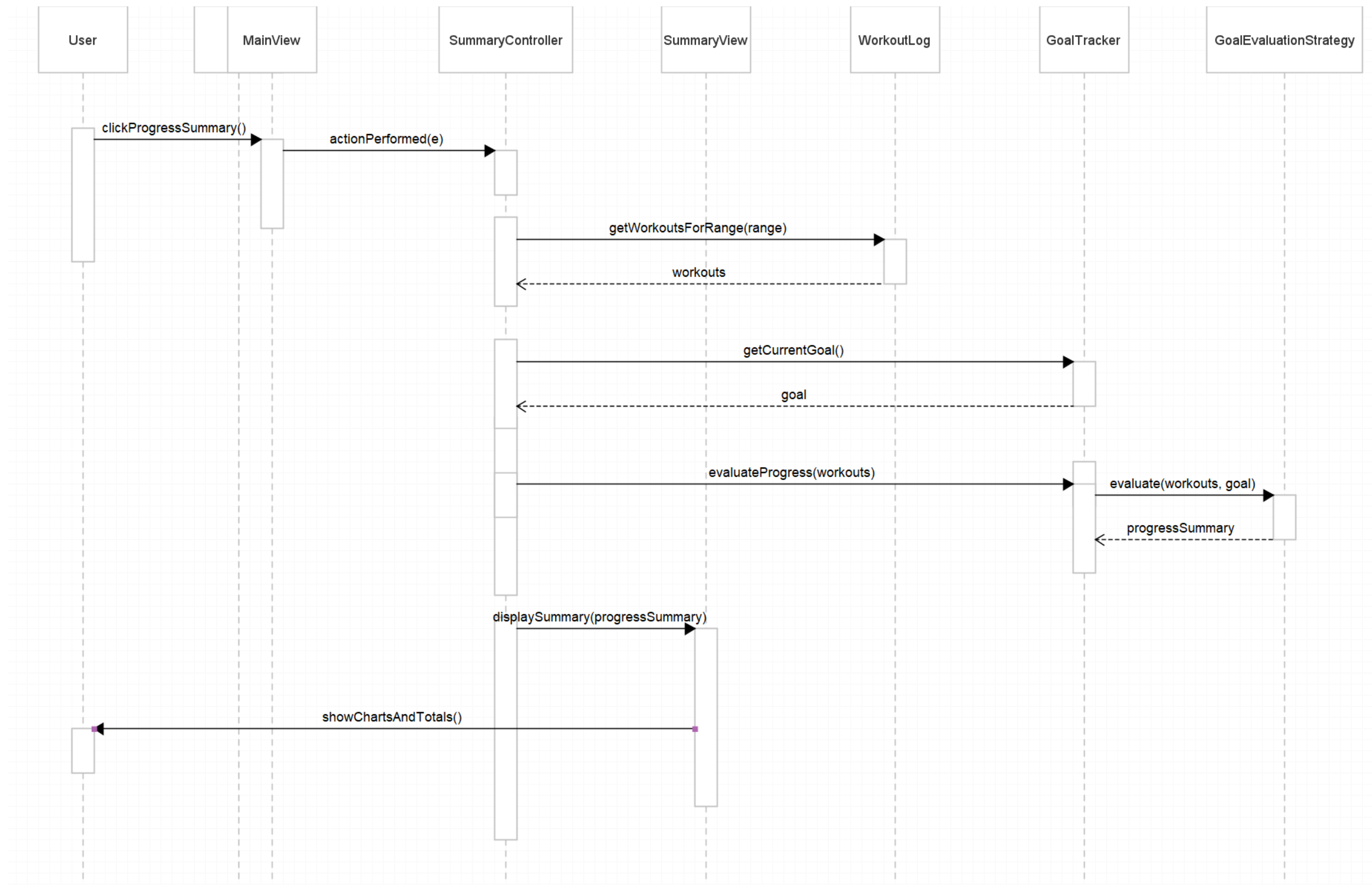
SD-5: User Deletes Workout Entry



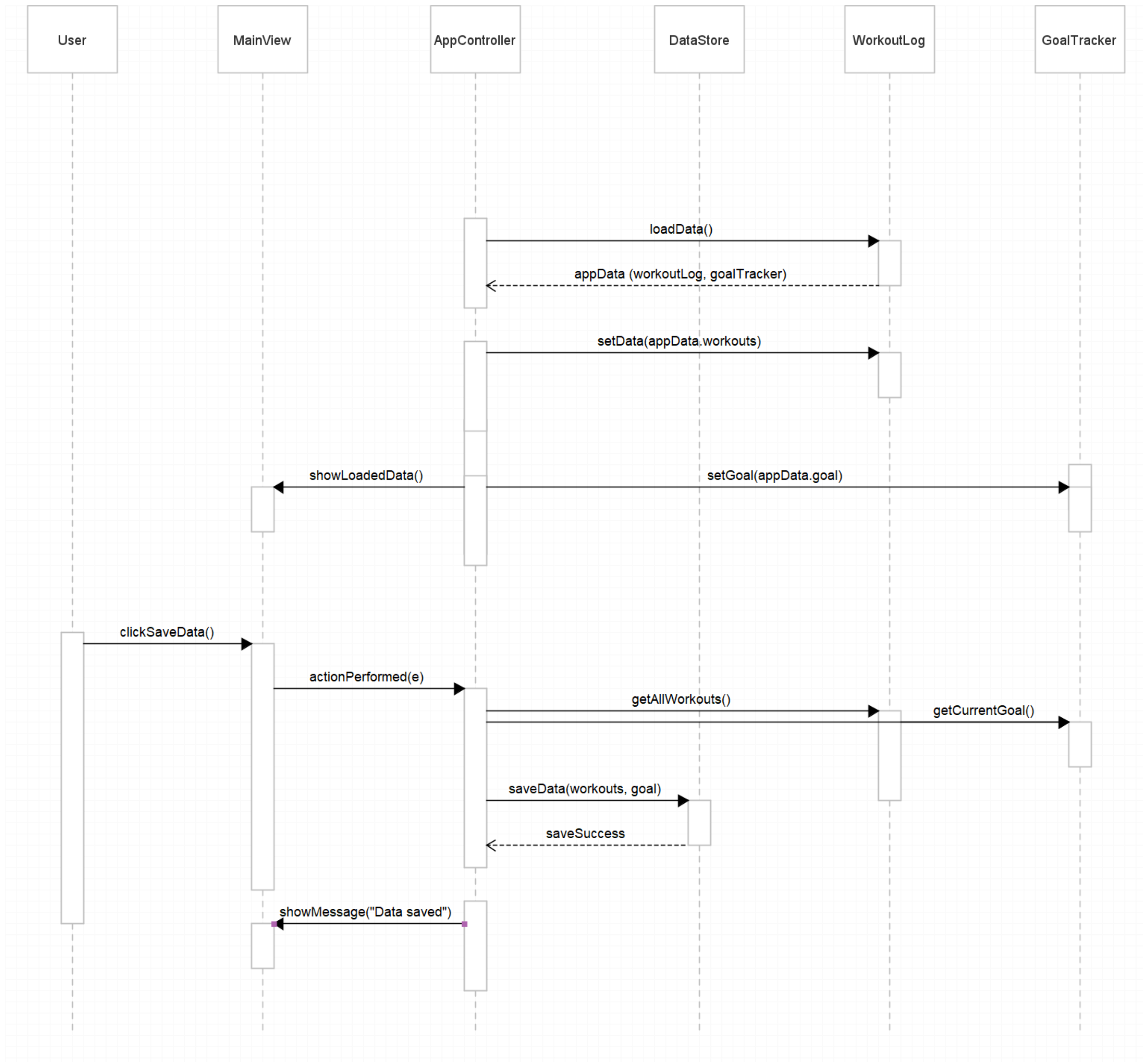
SD-6: User Exports Data



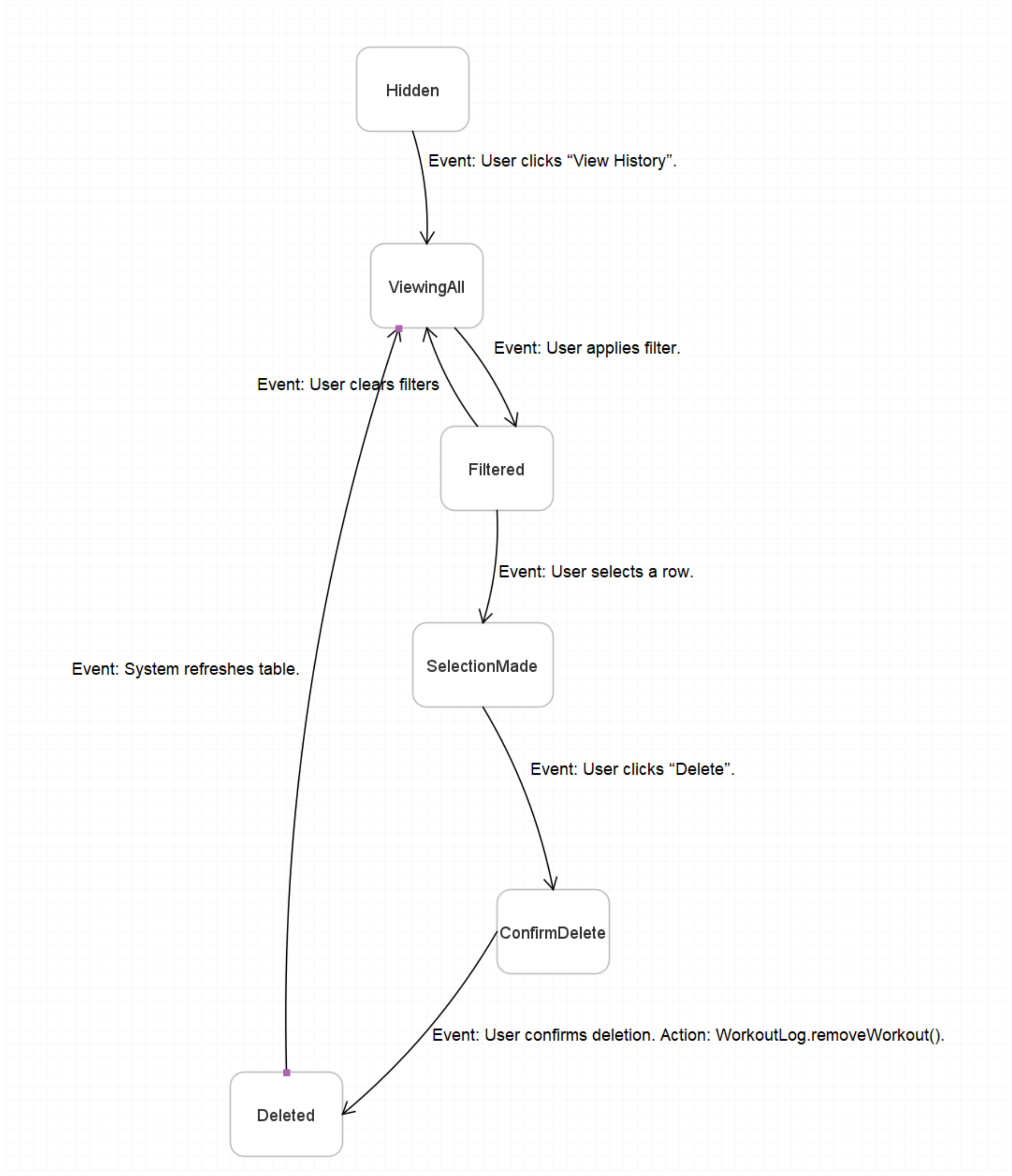
SD-7: User Views Progress Summary



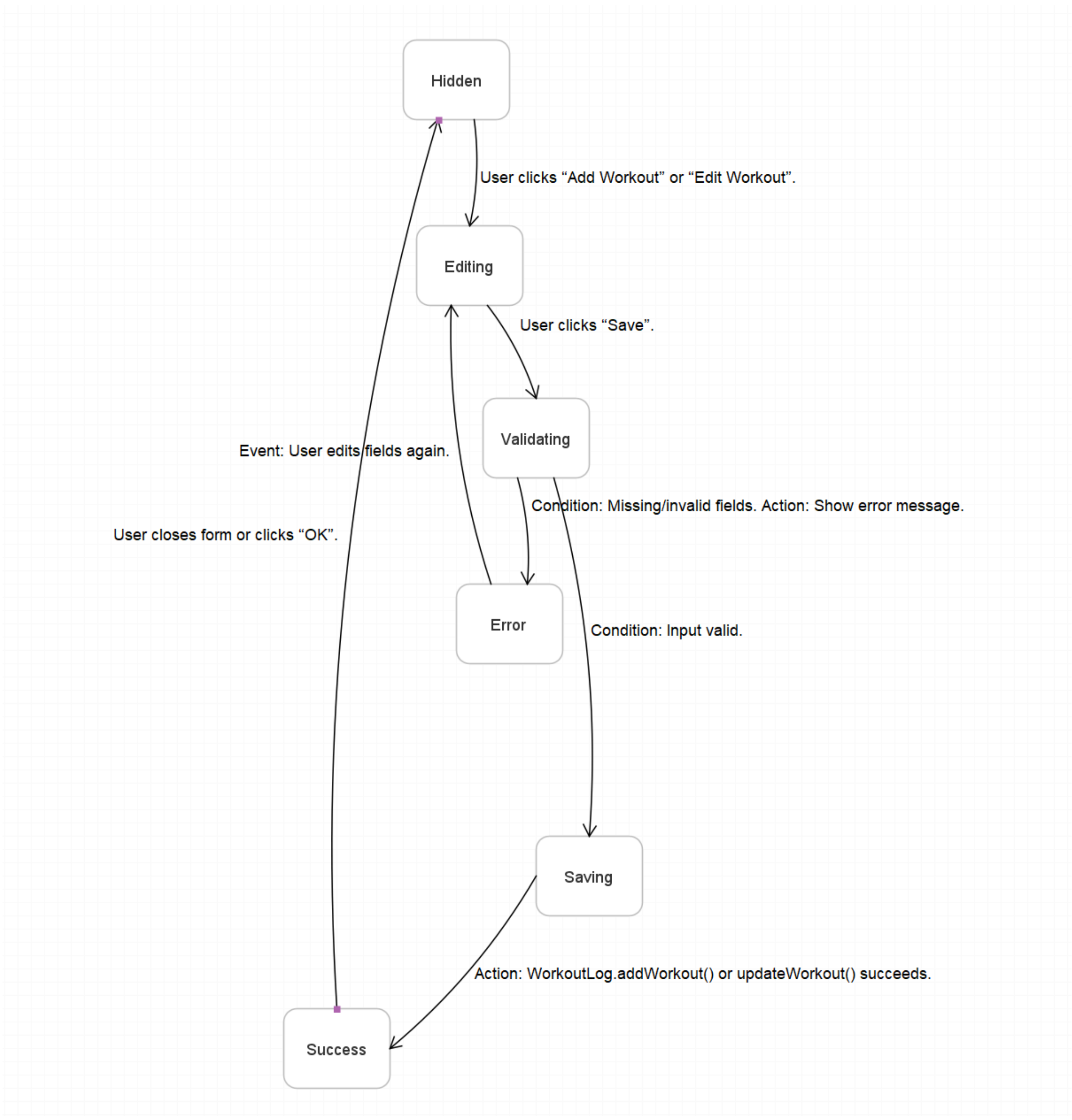
SD-8: Save/Load Data



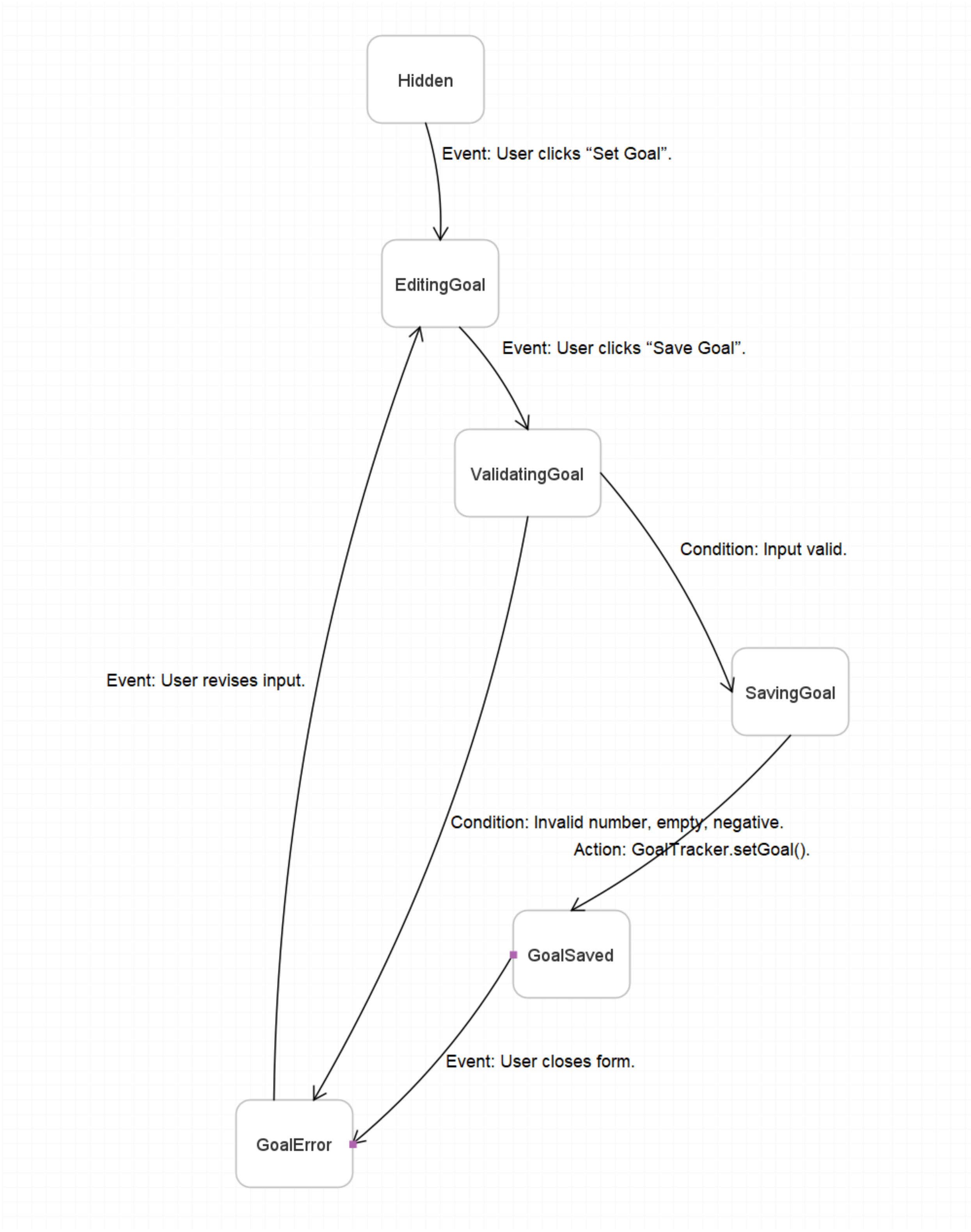
HistoryView State Diagram


Hidden
Event: User clicks “View History”.
ViewingAll
Event: User applies filter.
Event: User clears filters
Filtered
Event: User selects a row.
Event: System refreshes table.
SelectionMade
Event: User clicks “Delete”.
ConfirmDelete
Event: User confirms deletion. Action: WorkoutLog.removeWorkout().
Deleted

WorkoutEntryView State Diagram

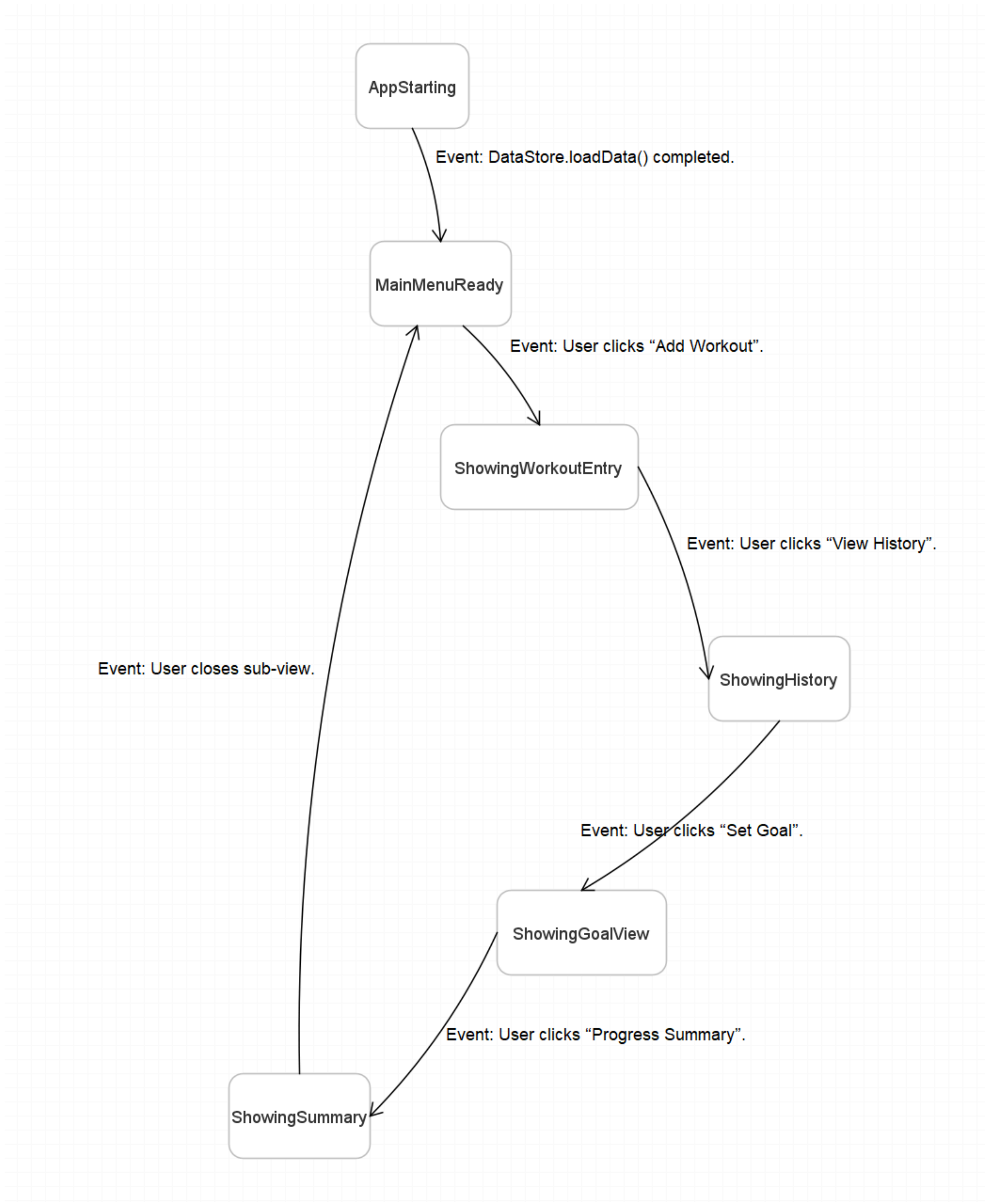

Hidden
User clicks “Add Workout” or “Edit Workout”.
Editing
User clicks “Save”.
Validating
Event: User edits fields again.
Condition: Missing/invalid fields. Action: Show error message.
User closes form or clicks “OK”.
Error
Condition: Input valid.
Saving
Action: WorkoutLog.addWorkout() or updateWorkout() succeeds.
Success

GoalView State Diagram



Hidden
Event: User clicks “Set Goal”.
EditingGoal
Event: User clicks “Save Goal”.
ValidatingGoal
Condition: Input valid.
SavingGoal
Event: User revises input.
Condition: Invalid number, empty, negative.
Action: GoalTracker.setGoal().
GoalSaved
Event: User closes form.
GoalError

MainView State Diagram


AppStarting
Event: DataStore.loadData() completed.
MainMenuReady
Event: User clicks “Add Workout”.
ShowingWorkoutEntry
Event: User clicks “View History”.
ShowingHistory
Event: User clicks “Set Goal”.
ShowingGoalView
Event: User clicks “Progress Summary”.
ShowingSummary
Event: User closes sub-view.

Files

Filename: AppController.java

```
package cop4331.controller;

import cop4331.model.*;

public class AppController {

    public static void saveData(WorkoutLog log, GoalTracker tracker) {

        DataStore.getInstance().saveData(log, tracker);

    }

}
```

Filename: ExportController.java

```
package cop4331.controller;

import cop4331.view.ExportDialog;

import cop4331.model.WorkoutLog;

import cop4331.util.CSVExporter;

import javax.swing.*;

import java.awt.event.*;

public class ExportController implements ActionListener {
```

```
private ExportDialog dialog;

private WorkoutLog log;

public ExportController(ExportDialog dialog, WorkoutLog log) {

    this.dialog = dialog;

    this.log = log;

    actionPerformed(null); // auto-run

}

@Override

public void actionPerformed(ActionEvent e) {

    String path = dialog.chooseFileLocation();

    if (path == null) return;

    try {

        CSVExporter.exportToCSV(log.getAllWorkouts(), path);

        JOptionPane.showMessageDialog(null, "Exported!");

    } catch (Exception ex) {

        JOptionPane.showMessageDialog(null, "Export failed.");
        }

    }

}
```

Filename: GoalController.java

```
package cop4331.controller;

import cop4331.view.*;

import cop4331.model.*;

import javax.swing.*;

import java.awt.event.*;

public class GoalController implements ActionListener {

    private GoalView view;

    private GoalTracker tracker;

    private JFrame frame;

    private MainView mainView;

    public GoalController(GoalView view, GoalTracker tracker, JFrame frame, MainView
mainView) {
    this.view = view;

    this.tracker = tracker;

    this.frame = frame;

    this.mainView = mainView;

    view.saveGoalBtn.addActionListener(this);

    view.backBtn.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {



    if (e.getSource() == view.backBtn) {

        frame.setContentPane(mainView);

        frame.revalidate();

        frame.repaint();

        return;

    }
```

```
        try {

            double goalVal = Double.parseDouble(view.goalField.getText());

            tracker.setGoal(new Goal(goalVal));

            JOptionPane.showMessageDialog(null, "Goal saved!");

        } catch (Exception ex) {

            JOptionPane.showMessageDialog(null, "Invalid goal value.");

        }

    }

}
```

Filename: HistoryController.java

```
package cop4331.controller;

import cop4331.view.*;

import cop4331.model.*;

import javax.swing.*;

import java.awt.event.*;

import java.time.LocalDate;
import java.util.stream.Collectors;

public class HistoryController implements ActionListener {

    private HistoryView view;

    private WorkoutLog log;

    private JFrame frame;

    private MainView mainView;

    public HistoryController(HistoryView view, WorkoutLog log, JFrame frame, MainView
mainView) {

        this.view = view;

        this.log = log;

        this.frame = frame;

        this.mainView = mainView;



        view.filterBtn.addActionListener(this);

        view.clearFilterBtn.addActionListener(this);

        view.deleteBtn.addActionListener(this);

        view.editBtn.addActionListener(this);
    view.backBtn.addActionListener(this);

    loadTableData(log.getAllWorkouts());

}

private void loadTableData(java.util.List<Workout> workouts) {

    view.tableModel.setRowCount(0);

    for (Workout w : workouts) {

        view.tableModel.addRow(new Object[]{

                w.getId(),

                w.getDate(),

                w.getType(),

                w.getDurationMinutes(),

                w.getIntensity(),

                w.getCaloriesBurned()

        });

    }

}
```

```
@Override

public void actionPerformed(ActionEvent e) {

    if (e.getSource() == view.backBtn) {

        frame.setContentPane(mainView);

        frame.revalidate();

        frame.repaint();

        return;

    }

    Object src = e.getSource();

    if (src == view.filterBtn) {

        handleFilter();

    }

    if (src == view.clearFilterBtn) {

        loadTableData(log.getAllWorkouts());

    }

    if (src == view.deleteBtn) {
        handleDelete();

    }



    if (src == view.editBtn) {

        handleEdit();

    }

}

private void handleFilter() {

    String type = view.filterTypeField.getText();

    String dateStr = view.filterDateField.getText();

    java.util.List<Workout> list = log.getAllWorkouts();

    if (!type.isEmpty()) {

        list = list.stream()

                .filter(w -> w.getType().equalsIgnoreCase(type))

                .collect(Collectors.toList());

    }
    if (!dateStr.isEmpty()) {
        try {
            LocalDate d = LocalDate.parse(dateStr);
            list = list.stream()
                    .filter(w -> w.getDate().equals(d))
                    .collect(Collectors.toList());
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "Invalid date.");
        }
    }
    loadTableData(list);
}
private void handleDelete() {
    int row = view.workoutTable.getSelectedRow();
    if (row < 0) {
            JOptionPane.showMessageDialog(null, "Select a workout first.");

            return;

        }

        int id = (int) view.tableModel.getValueAt(row, 0);

        log.removeWorkout(id);

        loadTableData(log.getAllWorkouts());

    }

    private void handleEdit() {

        int row = view.workoutTable.getSelectedRow();

        if (row < 0) {

            JOptionPane.showMessageDialog(null, "Select a workout to edit.");

            return;

        }

        int id = (int) view.tableModel.getValueAt(row, 0);

        // find the workout by ID

        Workout w = log.getAllWorkouts().stream()
```

```
                .filter(x -> x.getId() == id)

                .findFirst()

                .orElse(null);

        if (w != null) {

            WorkoutEntryView editor = new WorkoutEntryView();

            new WorkoutEditController(editor, log, w, frame, mainView);

            frame.setContentPane(editor);

            frame.revalidate();

            frame.repaint();

        }

    }

}
```

Filename: MainController.java

```
package cop4331.controller;

import cop4331.view.*;

import cop4331.model.*;
import cop4331.util.CSVExporter;

import javax.swing.*;

import java.awt.event.*;

public class MainController implements ActionListener {

    private JFrame frame;

    private MainView mainView;

    private WorkoutLog workoutLog;

    private GoalTracker goalTracker;

    public MainController(JFrame frame, MainView mainView, WorkoutLog workoutLog,
GoalTracker goalTracker) {

        this.frame = frame;

        this.mainView = mainView;

        this.workoutLog = workoutLog;

        this.goalTracker = goalTracker;

        mainView.addWorkoutBtn.addActionListener(this);

        mainView.viewHistoryBtn.addActionListener(this);

        mainView.setGoalBtn.addActionListener(this);
    mainView.summaryBtn.addActionListener(this);

    mainView.exportBtn.addActionListener(this);

    mainView.saveBtn.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {

    Object src = e.getSource();

    if (src == mainView.addWorkoutBtn) {

        WorkoutEntryView view = new WorkoutEntryView();

        new WorkoutEntryController(view, workoutLog, frame, mainView);

        showPanel(view);

    }

    if (src == mainView.viewHistoryBtn) {

        HistoryView view = new HistoryView();

        new HistoryController(view, workoutLog, frame, mainView);

        showPanel(view);
```

```
}

if (src == mainView.setGoalBtn) {

    GoalView view = new GoalView();

    new GoalController(view, goalTracker, frame, mainView);

    showPanel(view);

}

if (src == mainView.summaryBtn) {

    SummaryView view = new SummaryView();

    new SummaryController(view, workoutLog, goalTracker, frame, mainView);

    showPanel(view);

}

if (src == mainView.exportBtn) {

    ExportDialog dialog = new ExportDialog();

    new ExportController(dialog, workoutLog);

}

if (src == mainView.saveBtn) {
```

```
            AppController.saveData(workoutLog, goalTracker);

        }

    }

    private void showPanel(JPanel panel) {

        frame.setContentPane(panel);

        frame.revalidate();

        frame.repaint();

    }

}
```

Filename: SummaryController.java

```
package cop4331.controller;

import cop4331.view.*;

import cop4331.model.*;



import javax.swing.*;

import java.awt.event.*;
```

```
public class SummaryController implements ActionListener {

    private SummaryView view;

    private WorkoutLog log;

    private GoalTracker tracker;

    private JFrame frame;

    private MainView mainView;

    public SummaryController(SummaryView view, WorkoutLog log, GoalTracker tracker,
JFrame frame, MainView mainView) {

        this.view = view;

        this.log = log;

        this.tracker = tracker;

        this.frame = frame;

        this.mainView = mainView;

        view.backBtn.addActionListener(this);

        loadSummary();

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == view.backBtn) {

            frame.setContentPane(mainView);

            frame.revalidate();

            frame.repaint();

            return;

        }



        loadSummary();

    }

    private void loadSummary() {

        ProgressSummary summary = tracker.evaluateProgress(log.getAllWorkouts());

        view.updateSummary(summary.getTotalCalories(), summary.isGoalReached());

    }

}
```

Filename: WorkoutEditController.java

```
package cop4331.controller;

import cop4331.view.*;

import cop4331.model.*;



import javax.swing.*;

import java.awt.event.*;

public class WorkoutEditController implements ActionListener {

    private WorkoutEntryView view;

    private WorkoutLog log;

    private Workout workout;

    private JFrame frame;

    private MainView mainView;

    public WorkoutEditController(WorkoutEntryView view, WorkoutLog log, Workout
workout, JFrame frame, MainView mainView) {

        this.view = view;

        this.log = log;
    this.workout = workout;

    this.frame = frame;

    this.mainView = mainView;

    // populate fields

    view.typeField.setText(workout.getType());

    view.durationField.setText(String.valueOf(workout.getDurationMinutes()));

    view.intensityBox.setSelectedItem(workout.getIntensity().name());

    view.saveBtn.setText("Update Workout");

    view.saveBtn.addActionListener(this);

    view.backBtn.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {



    if (e.getSource() == view.backBtn) {

        frame.setContentPane(mainView);
    }

            frame.revalidate();

            frame.repaint();

            return;

        }

        try {

            workout.setType(view.typeField.getText());

            workout.setDurationMinutes(Integer.parseInt(view.durationField.getText()));

            workout.setIntensity(IntensityLevel.valueOf((String)
view.intensityBox.getSelectedItem()));

            // update calories

            workout.setCaloriesBurned(workout.getDurationMinutes() * 6.5);

            JOptionPane.showMessageDialog(null, "Workout updated!");



        } catch (Exception ex) {

            JOptionPane.showMessageDialog(null, "Invalid input.");

        }

    }
```

```
}
```

Filename: WorkoutEntryController.java

```
package cop4331.controller;

import cop4331.view.*;

import cop4331.model.*;

import javax.swing.*;

import java.awt.event.*;



public class WorkoutEntryController implements ActionListener {

    private WorkoutEntryView view;

    private WorkoutLog log;

    private JFrame frame;

    private MainView mainView;

    public WorkoutEntryController(WorkoutEntryView view, WorkoutLog log, JFrame
frame, MainView mainView) {

        this.view = view;
    this.log = log;

    this.frame = frame;

    this.mainView = mainView;

    view.saveBtn.addActionListener(this);

    view.backBtn.addActionListener(this);

}



@Override

public void actionPerformed(ActionEvent e) {

    if (e.getSource() == view.backBtn) {

        frame.setContentPane(mainView);

        frame.revalidate();

        frame.repaint();

        return;

    }

    try {
```

```
String type = view.typeField.getText();

int duration = Integer.parseInt(view.durationField.getText());

IntensityLevel level = IntensityLevel.valueOf(

        (String) view.intensityBox.getSelectedItem()

);



double calories = duration * 6.5;

Workout workout = new Workout(

        log.getAllWorkouts().size() + 1,

        java.time.LocalDate.now(),

        type,

        duration,

        level,

        calories

);

log.addWorkout(workout);
```

```
            JOptionPane.showMessageDialog(null, "Workout added successfully!");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "Invalid input.");
        }
    }
}
```

Filename: DataStore.java

```
package cop4331.model;
public class DataStore {
    private static DataStore instance;
    private DataStore() {}


    public static DataStore getInstance() {
        if (instance == null)
            instance = new DataStore();
        return instance;
    }

    public void saveData(WorkoutLog log, GoalTracker tracker) {
    }
}
```

Filename:Goal.java

```
package cop4331.model;

public class Goal {
    private double targetCalories;

    public Goal(double targetCalories) { this.targetCalories = targetCalories; }

    public double getTargetCalories() { return targetCalories; }
}
```

Filename:GoalEvaluationStrategy.java

```
package cop4331.model;
```

```
import java.util.List;

public interface GoalEvaluationStrategy {

    ProgressSummary evaluate(List<Workout> workouts, Goal goal);

}
```

Filename: GoalTracker.java

```
package cop4331.model;

import java.util.List;

public class GoalTracker {



    private Goal currentGoal;

    private GoalEvaluationStrategy strategy

    public GoalTracker() {

        this.strategy = new WeeklyCaloriesStrategy();

    }

    public void setGoal(Goal goal) {

        this.currentGoal = goal;
    }

    public Goal getCurrentGoal() {

        return currentGoal;

    }

    public void setStrategy(GoalEvaluationStrategy strategy) {

        if (strategy != null) {

            this.strategy = strategy;

        }

    }

    public ProgressSummary evaluateProgress(List<Workout> workouts) {

        if (currentGoal == null) {

            // No goal set → return 0 with "not reached"

            return new ProgressSummary(0, false);

        }

        return strategy.evaluate(workouts, currentGoal);

    }
```

```
}
```

Filename:IntensityLevel.java

```
package cop4331.model;

public enum IntensityLevel {

    LOW, MEDIUM, HIGH

}
```

Filename:ProgressSummary.java

```
package cop4331.model;



public class ProgressSummary {

    private double totalCalories;

    private boolean goalReached;

    public ProgressSummary(double totalCalories, boolean goalReached) {

        this.totalCalories = totalCalories;

        this.goalReached = goalReached;

    }
    public double getTotalCalories() { return totalCalories; }

    public boolean isGoalReached() { return goalReached; }

}
```

Filename: WeeklyCaloriesStrategy.java

```
package cop4331.model;

import java.util.List;

public class WeeklyCaloriesStrategy implements GoalEvaluationStrategy {


    @Override

    public ProgressSummary evaluate(List<Workout> workouts, Goal goal) {

        double total = workouts.stream()

                .mapToDouble(Workout::getCaloriesBurned)

                .sum();

        boolean reached = total >= goal.getTargetCalories();

        return new ProgressSummary(total, reached);

    }
}
```

Filename:Workout.java

```
package cop4331.model;

import java.time.LocalDate;

public class Workout {

    private int id;

    private LocalDate date;

    private String type;

    private int durationMinutes;

    private IntensityLevel intensity;

    private double caloriesBurned;

    public Workout(int id, LocalDate date, String type, int durationMinutes,

                   IntensityLevel intensity, double caloriesBurned) {

        this.id = id;

        this.date = date;

        this.type = type;
        this.durationMinutes = durationMinutes;

        this.intensity = intensity;

        this.caloriesBurned = caloriesBurned;

    }

    // ======== GETTERS =========

    public int getId() {

        return id;

    }

    public LocalDate getDate() {

        return date;

    }

    public String getType() {

        return type;

    }

    public int getDurationMinutes() {

        return durationMinutes;
```

```
}

public IntensityLevel getIntensity() {

    return intensity;

}

public double getCaloriesBurned() {

    return caloriesBurned;

}

public void setType(String type) {

    this.type = type;

}

public void setDurationMinutes(int durationMinutes) {

    this.durationMinutes = durationMinutes;

}

public void setIntensity(IntensityLevel intensity) {

    this.intensity = intensity;

}
```

```
    public void setCaloriesBurned(double caloriesBurned) {

        this.caloriesBurned = caloriesBurned;

    }

}
```

Filename:WorkoutLog.java

```
package cop4331.model;

import java.util.ArrayList;

import java.util.List;

public class WorkoutLog {

    private List<Workout> workouts = new ArrayList<>();

    public void addWorkout(Workout w) { workouts.add(w); }

    public void removeWorkout(int id) {

        workouts.removeIf(w -> w.getId() == id);

    }

    public List<Workout> getAllWorkouts() {

        return new ArrayList<>(workouts);
    }

}
```

Filename:CSVExporter.java

```
package cop4331.util;

import cop4331.model.Workout;



import java.io.FileWriter;

import java.io.IOException;

import java.util.List;

public class  CSVExporter {

    /**

     * Exports a list of workouts to a CSV file.

     *

     * @param workouts list of Workout objects to export

     * @param filePath full file path chosen by user (ex: C:/output/workouts.csv)
    * @throws IOException if the file cannot be written

    */

   public static void exportToCSV(List<Workout> workouts, String filePath) throws
IOException {

        FileWriter writer = new FileWriter(filePath);

        // Write header row

        writer.write("ID,Date,Type,Duration,Intensity,Calories\n");

        // Write each workout row

        for (Workout w : workouts) {

            writer.write(

                    w.getId() + "," +

                            w.getDate() + "," +

                            safe(w.getType()) + "," +

                            w.getDurationMinutes() + "," +

                            w.getIntensity() + "," +

                            w.getCaloriesBurned() + "\n"

            );
```

```java
        }

        writer.flush();

        writer.close();

    }

    /**

     * Safely formats strings for CSV.

     *

     * @param s input string

     * @return sanitized version without commas or quotes

     */

    private static String safe(String s) {

        if (s == null) return "";

        return s.replace(",", ";").replace("\"", "");

    }

}
```

Filename:ExportDialog.java

```
package cop4331.view;

import javax.swing.*;

import java.io.File;

public class ExportDialog {

    public String chooseFileLocation() {

        JFileChooser chooser = new JFileChooser();

        chooser.setDialogTitle("Select Export Location");

        int result = chooser.showSaveDialog(null);

        if (result == JFileChooser.APPROVE_OPTION) {

            File selected = chooser.getSelectedFile();

            return selected.getAbsolutePath();

        }

        return null;

    }

}
```

```
}
```

Filename:GoalView.java

```
package cop4331.view;

import javax.swing.*;

import java.awt.*;

public class GoalView extends JPanel {



    public JTextField goalField = new JTextField(20);

    public JButton saveGoalBtn = new JButton("Save Goal");

    public JButton backBtn = new JButton("Back");

    public GoalView() {

        setLayout(new GridLayout(3, 2, 10, 10));

        add(new JLabel("Weekly Calorie Goal:"));

        add(goalField);

        add(saveGoalBtn);

        add(backBtn);

    }
}
```

Filename:HistoryView.java

```
package cop4331.view;

import javax.swing.*;

import javax.swing.table.DefaultTableModel;

import java.awt.*;

public class HistoryView extends JPanel {

    public JTable workoutTable;

    public DefaultTableModel tableModel;

    public JButton filterBtn = new JButton("Apply Filter");

    public JButton clearFilterBtn = new JButton("Clear Filter");

    public JButton deleteBtn = new JButton("Delete Selected");

    public JButton editBtn = new JButton("Edit Selected");

    public JButton backBtn = new JButton("Back");



    public JTextField filterTypeField = new JTextField(15);
public JTextField filterDateField = new JTextField(15);

public HistoryView() {

    setLayout(new BorderLayout());

    tableModel = new DefaultTableModel(new String[]{

            "ID", "Date", "Type", "Duration", "Intensity", "Calories"

    }, 0);

    workoutTable = new JTable(tableModel);

    JScrollPane scroll = new JScrollPane(workoutTable);

    add(scroll, BorderLayout.CENTER);

    JPanel filterPanel = new JPanel(new GridLayout(3, 2));

    filterPanel.add(new JLabel("Type:"));

    filterPanel.add(filterTypeField);

    filterPanel.add(new JLabel("Date (YYYY-MM-DD):"));

    filterPanel.add(filterDateField);

    filterPanel.add(filterBtn);

    filterPanel.add(clearFilterBtn);
```

```
        add(filterPanel, BorderLayout.NORTH);

        JPanel actionPanel = new JPanel();

        actionPanel.add(editBtn);

        actionPanel.add(deleteBtn);

        actionPanel.add(backBtn);

        add(actionPanel, BorderLayout.SOUTH);

    }

}
```

Filename:MainView.java

```
package cop4331.view;

import javax.swing.*;

import java.awt.*;

public class MainView extends JPanel {

    public JButton addWorkoutBtn = new JButton("Add Workout");

    public JButton viewHistoryBtn = new JButton("View History");
    public JButton setGoalBtn = new JButton("Set Goal");

    public JButton summaryBtn = new JButton("Progress Summary");

    public JButton exportBtn = new JButton("Export to CSV");

    public JButton saveBtn = new JButton("Save Data");

    public MainView() {

        setLayout(new GridLayout(6, 1, 10, 10));

        add(addWorkoutBtn);

        add(viewHistoryBtn);

        add(setGoalBtn);

        add(summaryBtn);

        add(exportBtn);

        add(saveBtn);

    }

}
```

Filename:SummaryView.java

```
}

package cop4331.view;

import javax.swing.*;

import java.awt.*;

public class SummaryView extends JPanel {

    public JLabel totalCaloriesLabel = new JLabel("Total Calories: 0");

    public JLabel goalReachedLabel = new JLabel("Goal Reached: No");

    public JButton backBtn = new JButton("Back");

    public SummaryView() {

        setLayout(new GridLayout(3, 1, 10, 10));

        totalCaloriesLabel.setHorizontalAlignment(SwingConstants.CENTER);

        goalReachedLabel.setHorizontalAlignment(SwingConstants.CENTER);



        add(totalCaloriesLabel);

        add(goalReachedLabel);

        add(backBtn);

    }
    public void updateSummary(double totalCalories, boolean reached) {

        totalCaloriesLabel.setText("Total Calories: " + totalCalories);

        goalReachedLabel.setText("Goal Reached: " + (reached ? "Yes" : "No"));

    }

}
```

Filename:WorkoutEntryView.java

```
package cop4331.view;

import javax.swing.*;

import java.awt.*;

public class WorkoutEntryView extends JPanel {

    public JTextField typeField = new JTextField(20);

    public JTextField durationField = new JTextField(20);

    public JComboBox<String> intensityBox = new JComboBox<>(new String[]{"LOW",
"MEDIUM", "HIGH"});

    public JButton saveBtn = new JButton("Save Workout");

    public JButton backBtn = new JButton("Back");

    public WorkoutEntryView() {
        setLayout(new GridLayout(5, 2, 10, 10));

        add(new JLabel("Workout Type:"));

        add(typeField);



        add(new JLabel("Duration (minutes):"));

        add(durationField);

        add(new JLabel("Intensity:"));

        add(intensityBox);

        add(saveBtn);

        add(backBtn);

    }

}
```

Filename:Main.java

```
package cop4331.main;

import cop4331.model.*;
import cop4331.view.*;

import cop4331.controller.*;

import javax.swing.*;

public class Main {

    public static void main(String[] args) {

        WorkoutLog log = new WorkoutLog();

        GoalTracker tracker = new GoalTracker();

        JFrame frame = new JFrame("Fitness Tracker");

        frame.setSize(400, 450);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        MainView mainView = new MainView();

        new MainController(frame, mainView, log, tracker);



        frame.setContentPane(mainView);

        frame.setLocationRelativeTo(null);

        frame.setVisible(true);
    }

}
```

Jar Code

Filename: FitnessTracker.jar

TEST

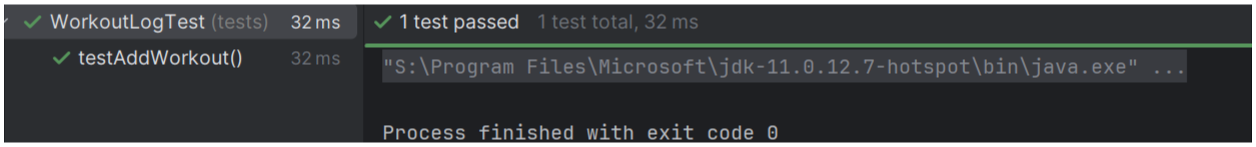
Filename: WorkoutLogTest.java

```
package tests;

import cop4331.model.*;

import org.junit.jupiter.api.Test;

import java.time.LocalDate;

import static org.junit.jupiter.api.Assertions.*;

public class WorkoutLogTest {

    @Test

    public void testAddWorkout() {
        WorkoutLog log = new WorkoutLog();

        Workout w = new Workout(1, LocalDate.now(), "Run", 30, IntensityLevel.MEDIUM,
250);

        log.addWorkout(w);

        assertEquals(1, log.getAllWorkouts().size());

    }

}
```

Description:

This testing strategy for the WorkoutLog class provides a way to demonstrate that when creating a Workout object using an instance of the WorkoutLog class, the proper function ADD creates a valid entry in the database (i.e., ADD returns true or false depending on whether the ADD worked or not). This was accomplished by creating a new Workout object with acceptable attributes (Readonly ID, current date, workout type, duration, and burn calories) and passing these details into the method call log.Add().

The next step in this process was to retrieve the total number of entries via the log.Show() method. As expected, there should be now one workout listed (because we just added it) and therefore the log.Show() method returns back an integer value representing that one entry in the logged workouts. In summary, both methods have been verified to perform as desired in that when you create a new workout from within an instance of the WorkoutLog class, it can be added to the underlying data structure (ArrayList) and then, when you want to, you can retrieve that workout via its collection of workouts. Thus, the methods of the WorkoutLog class provide for accurate insertion and retrieval of workout data.



```
✓ WorkoutLogTest (tests) 32 ms    ✓ 1 test passed  1 test total, 32 ms
  ✓ testAddWorkout()     32 ms    "S:\Program Files\Microsoft\jdk-11.0.12.7-hotspot\bin\java.exe" ...

                                  Process finished with exit code 0
```

Filename: GoalTrackerTest.java

```
package tests;

import cop4331.model.*;

import org.junit.jupiter.api.Test;

import java.time.LocalDate;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

public class GoalTrackerTest {

    @Test

    public void testSetAndGetGoal() {

        GoalTracker tracker = new GoalTracker();
    Goal goal = new Goal(1500);

    tracker.setGoal(goal);

    assertEquals(1500, tracker.getCurrentGoal().getTargetCalories());

}

@Test

public void testEvaluateProgressNoGoal() {

    GoalTracker tracker = new GoalTracker();

    List<Workout> workouts = List.of(

            new Workout(1, LocalDate.now(), "Run", 30, IntensityLevel.MEDIUM, 300)

    );

    ProgressSummary summary = tracker.evaluateProgress(workouts);

    assertEquals(0, summary.getTotalCalories());

    assertFalse(summary.isGoalReached());

}

@Test

public void testEvaluateProgressGoalReached() {
    GoalTracker tracker = new GoalTracker();

    tracker.setGoal(new Goal(500));

    List<Workout> workouts = List.of(

            new Workout(1, LocalDate.now(), "Run", 30, IntensityLevel.MEDIUM, 300),

            new Workout(2, LocalDate.now(), "Bike", 45, IntensityLevel.MEDIUM, 250)

    );

    ProgressSummary summary = tracker.evaluateProgress(workouts);



    assertEquals(550, summary.getTotalCalories());

    assertTrue(summary.isGoalReached());

}

@Test

public void testEvaluateProgressGoalNotReached() {

    GoalTracker tracker = new GoalTracker();

    tracker.setGoal(new Goal(1000));

    List<Workout> workouts = List.of(
            new Workout(1, LocalDate.now(), "Run", 30, IntensityLevel.MEDIUM, 300),

            new Workout(2, LocalDate.now(), "Bike", 45, IntensityLevel.MEDIUM, 250)

        );

        ProgressSummary summary = tracker.evaluateProgress(workouts);

        assertEquals(550, summary.getTotalCalories());

        assertFalse(summary.isGoalReached());

    }

    @Test

    public void testChangeStrategy() {

        GoalTracker tracker = new GoalTracker();

        GoalEvaluationStrategy strategy = new WeeklyCaloriesStrategy();

        tracker.setStrategy(strategy);



        assertNotNull(tracker);

    }

}
```

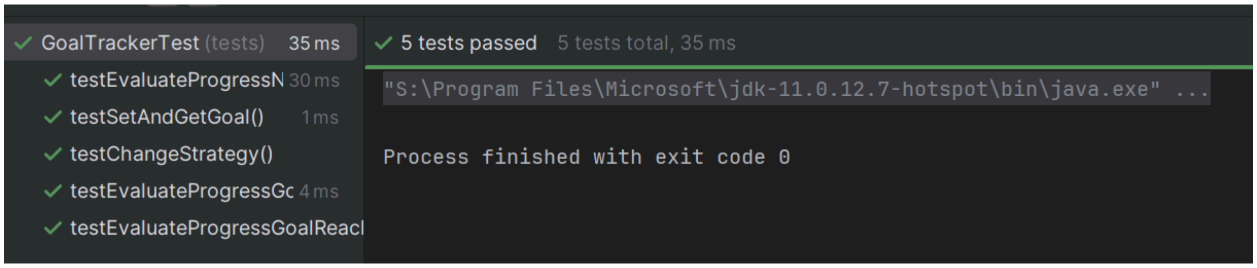
Description:

The GoalTrackerTest is a unit testing framework designed to ensure that the GoalTracker model meets the expectations for managing fitness goals, as well as evaluating the user's progress based on the specifications for your project. The first set of tests verify the fundamental operations of setting and retrieving a user-defined goal.

One of the tests examines the situation where progress is calculated without a goal, which is an important edge case, to confirm that the application correctly returns a value of zero with respect to the user's progress and indicates that they have not yet reached their goal.

These tests confirm that the GoalTracker can handle both normal and edge-case data inputs with respect to the integrity of the data and the expected behavior of the application. The next set of tests validate the calculation of the user's progress regarding their goal based on records of their physical activities and how those records provide accurate information regarding their progress.

To accomplish this, the tests had to provide lists of Workout Object instances with different calorie values. The test cases include progress beyond goal met, did not meet the goal, and changing evaluation methods. Collectively, these tests will assure that there are no errors in the way in which an application uses the Strategy Pattern, the display of progress summaries, and the accuracy of the calculation logic that is used for key use cases.



```
✓ GoalTrackerTest (tests)  35 ms
  ✓ testEvaluateProgressN 30 ms
  ✓ testSetAndGetGoal()    1 ms
  ✓ testChangeStrategy()
  ✓ testEvaluateProgressGc 4 ms
  ✓ testEvaluateProgressGoalReacl
```

```
✓ 5 tests passed  5 tests total, 35 ms
"S:\Program Files\Microsoft\jdk-11.0.12.7-hotspot\bin\java.exe" ...

Process finished with exit code 0
```