SOFTWARE ENGINEERING FOUNDATIONS: ASSESSMENT 2

FLORIAN BERGMANN
H00020398

MACS
Heriot-Watt University

December 2010

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

GUI graphical user interface

Part I

CREATION OF CABINMANAGER

1 OVERVIEW

1.1 PLOT-CLASS

The plot class provides the following attributes:

ID: Unique identifier for a plot.

SIZE: The area in square-meters provided by the plot.

ELECTRICITY: Boolean depicting whether the plot provides electricity or not.

The cost is calculated by associating the size with the adequate cost - this is done in the `PriceMapping` class.

If the plot provides electricity another amount is added on top (can be changed in the plot class as constant - default value is 10).

1.2 STATUS REPORT

The application should meet the specifications fully.

2

UML EXERCISE ONE

From the textual description of the use case the following use case diagram was derived:

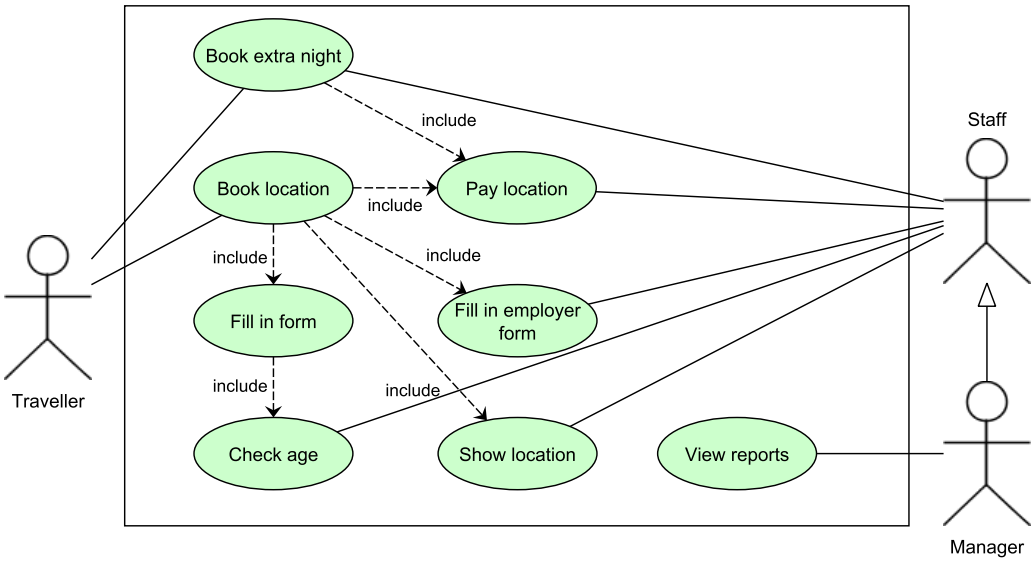

Figure 1: Use case diagram.

The complexity level with many *include* cases was chosen, to justify the use of a diagram and to show the possible sub-cases that might be reused.

Following from this description the *book location* high-ceremony use case would look like the following scenario:

2.1 HIGH CEREMONY USE CASES

2.1.1 *Use case: book location*

GOAL: Book a location.

ACTORS: Traveller, Staff.

PRECONDITION: A location is available.

POSTCONDITION: The traveller has booked a location for (at least) one night.

STEPS: 1. Traveller *fills in form*.
2. Staff *fills in employer form*.
3. Traveller *pays location*.
4. Staff *shows location* to traveller.

ALTERNATIVES: 1A. Fill in form use case fails.
The booking fails.
3A. Pay location use case fails.
The booking fails.

2.1.2 *Use case: fill in form*

GOAL: Traveller form is filled in.

ACTORS: Traveller, Staff.

PRECONDITION: User wants to book a location.

POSTCONDITION: Form was filled in correct.

STEPS: 1. Traveller enters personal details.
2. Traveller provides staff with passport.
3. Staff *checks age*.
4. Staff returns passport.

ALTERNATIVES: 3A. Check age use case fails.
The booking fails.

2.1.3 *Use case: pay location*

GOAL: Pay for stay.

ACTORS: Traveller, Staff.

PRECONDITION: Location and number of nights have been chosen.

POSTCONDITION: Payment is finished.

STEPS: 1. Traveller pays for desired number of nights.
2. Staff accepts payment.
3. Staff marks payment as received.

ALTERNATIVES: 1A. Traveller can not pay for stay.
The payment fails.

2.1.4 *Use case: show location*

GOAL: Traveller is shown location.

ACTORS: Traveller, Staff.

PRECONDITION: Payment is finished.

POSTCONDITION: -

STEPS: 1. Staff shows traveller the location.

ALTERNATIVES: -

2.1.5 *Use case: fill in employee form*

GOAL: Employee fills in form.

ACTORS: Staff.

PRECONDITION: -

POSTCONDITION: -

STEPS: 1. Staff enter details (location id, form number, nights to stay).

ALTERNATIVES: -

2.2 ACTIVITY DIAGRAMS

2.2.1 *Book location*

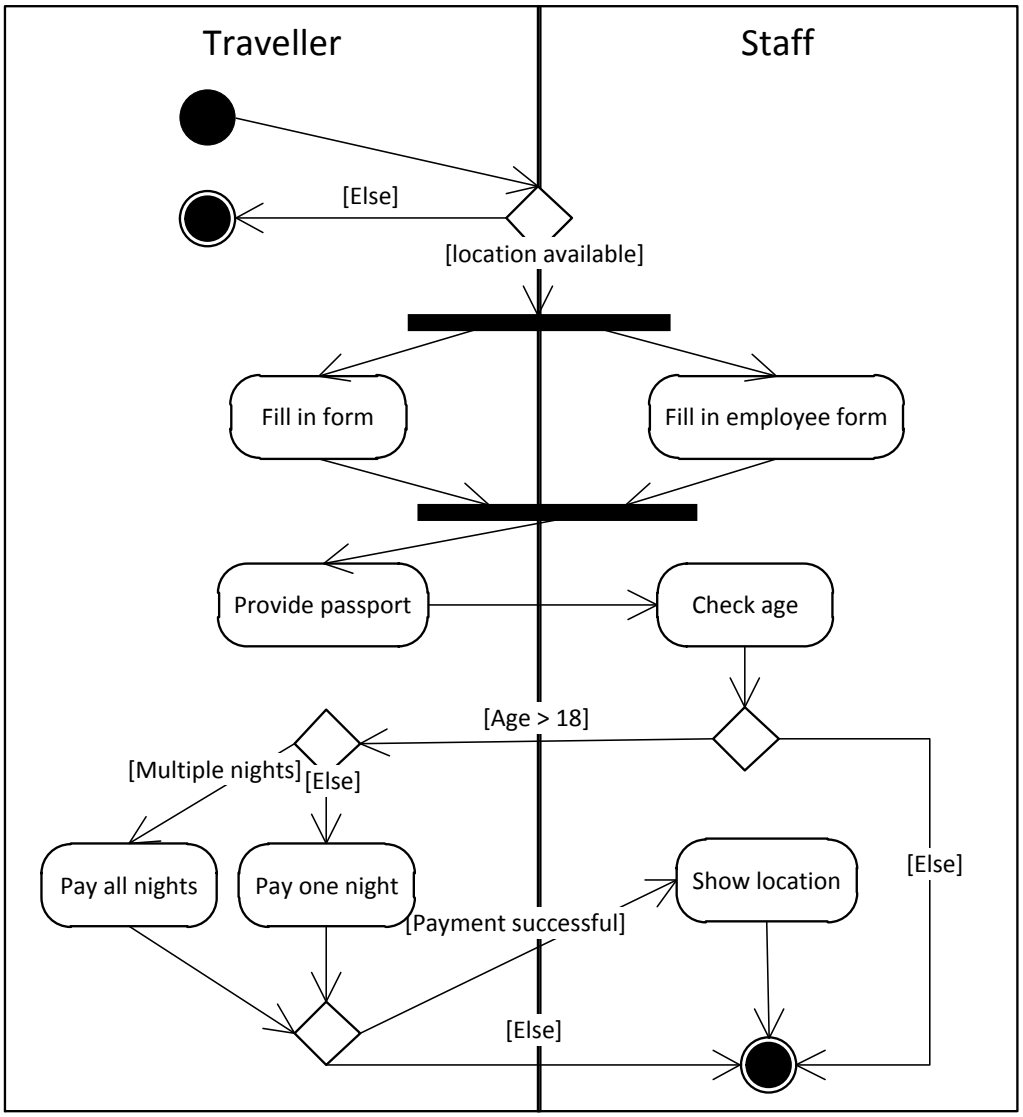

Figure 2: Activity diagram of book location.

Even though best practice is to only have one end-point per activity diagram, the provided activity diagram has two to improve readability.

2.2.2 *GetCost()*

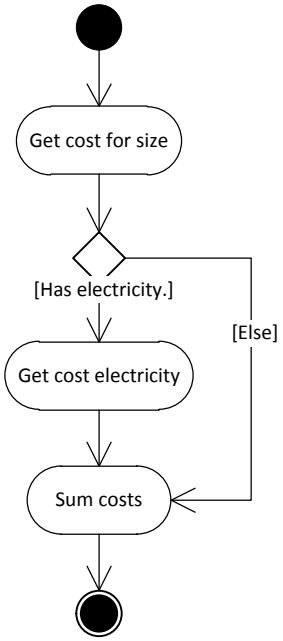

Figure 3: Activity diagram of getCost for plots.

3 CLASS DIAGRAMS

The class diagrams are provided on a per-package level:

3.1 VIEW

The view is connected to a presenter (holds locations that can be changed) and to the model (implements an observer pattern to notify the view about changes.

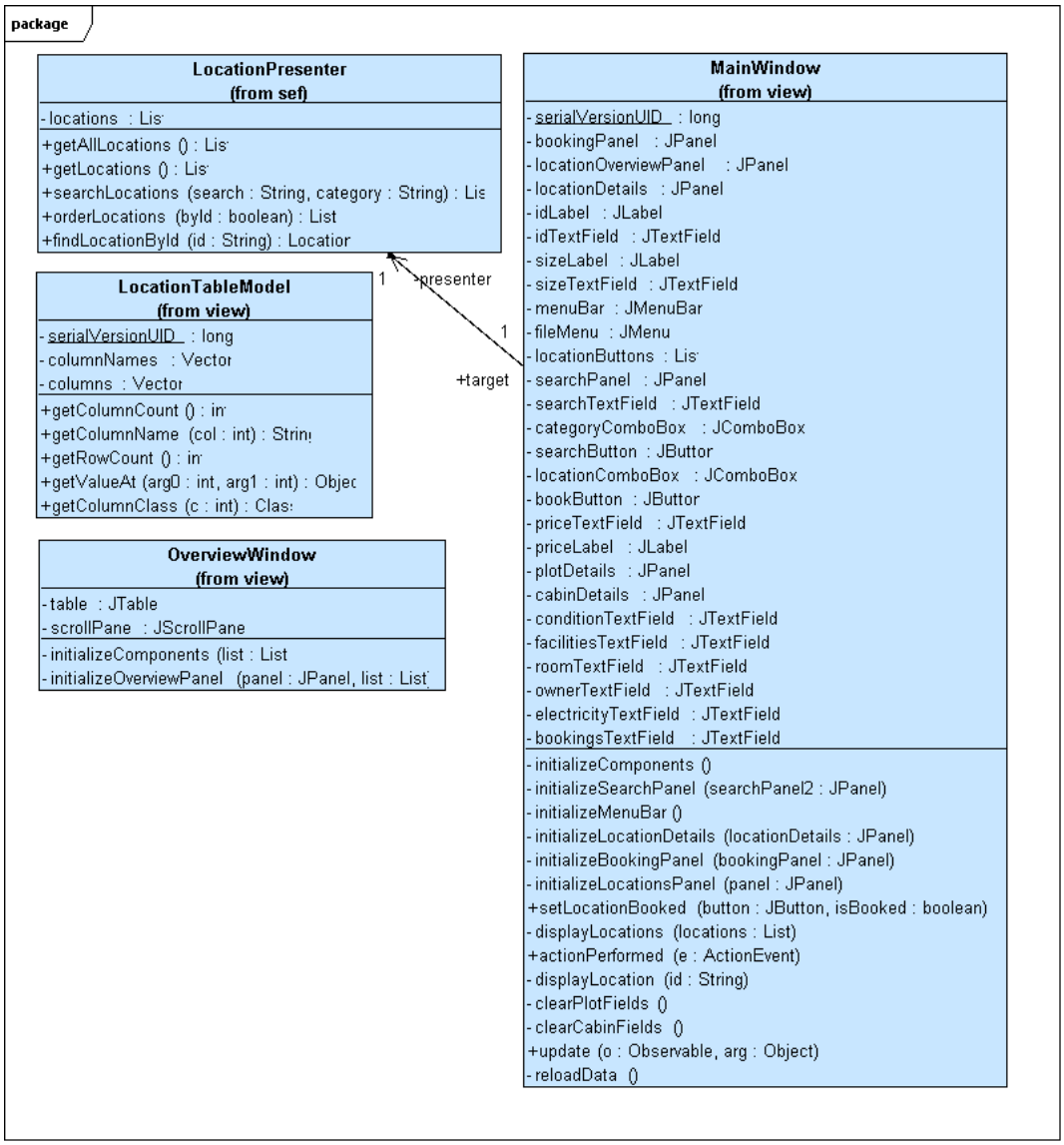

Figure 4: Class diagram view package.

3.2 MODEL

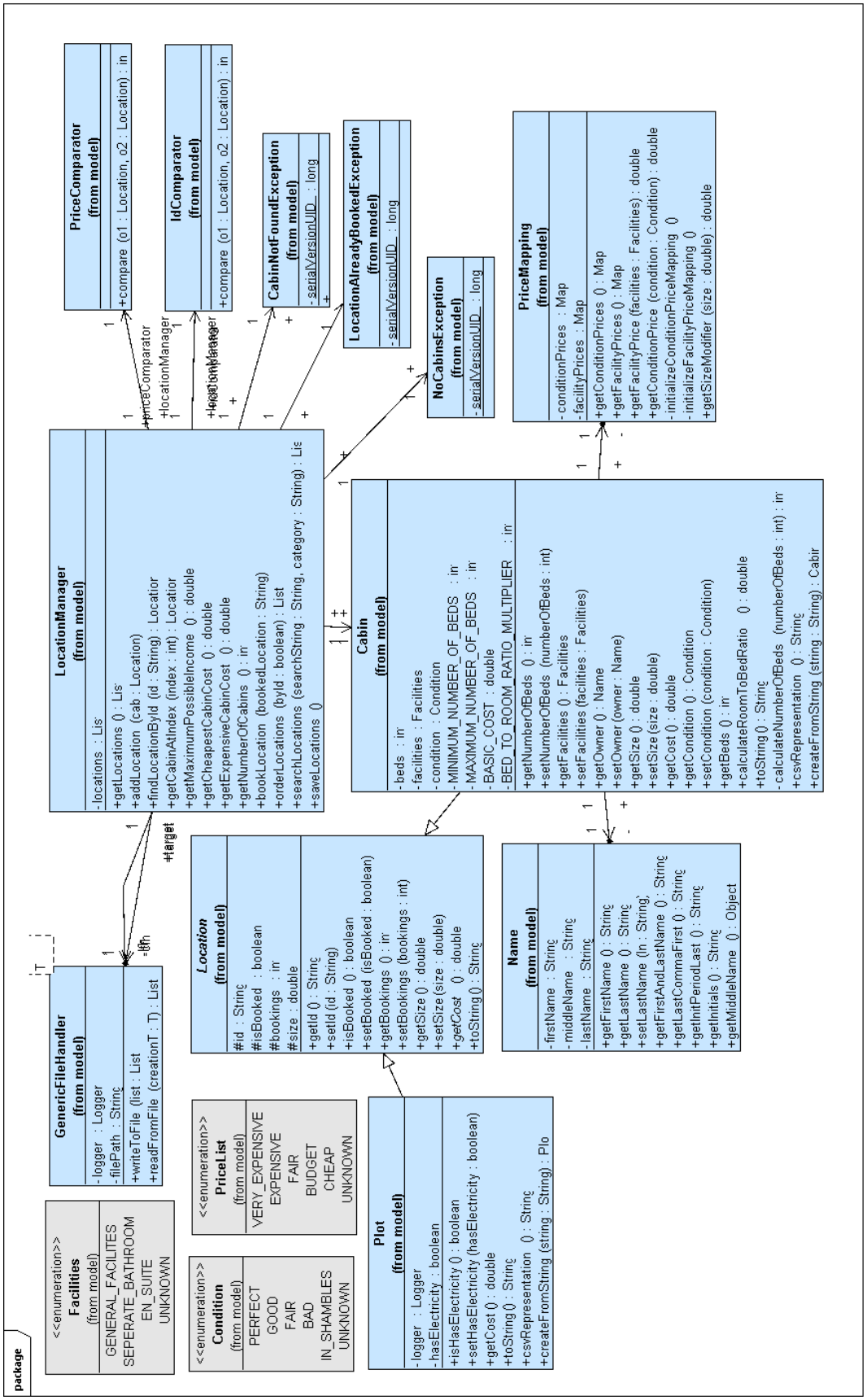

Figure 5: Class diagram model package.

3.3 MODEL.INTERFACES

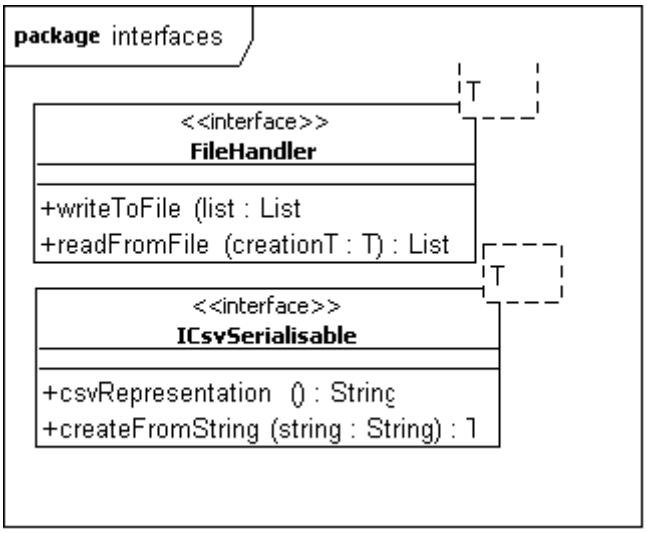

Figure 6: Class diagram model.interfaces package.

4 HOW IT WORKS

All code of the examples can be found in the appendix and will not be printed here.

4.1 BOOKING A LOCATION

To book a location the user has to select a location from the `DropDown`-box and click the book button:

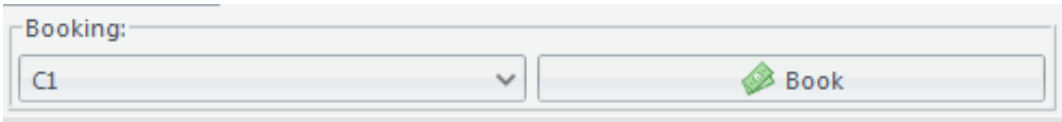

Figure 7: Booking a location.

This will trigger the `actionListener` of the `bookButton` (implemented as an anonymous inner class) that will receive the event from the `EventDispatcher`-thread.

In the `actionPerformed`-method the text (equals to the id of the location) of the `comboBox` will be retrieved and the `bookLocation`-method of the `LocationManager` will be invoked with it.

The method will set the `booked`-flag to true and increase the bookings if the location is not already booked. If the location is already booked an exception will be thrown and an error message displayed by the graphical user interface (GUI).

If the location is successfully booked, the manager will notify the GUI via an `observer`-pattern, the GUI will reload the locations that are displayed and redisplay them - now the flag of the booked location will be coloured red, as it is booked.

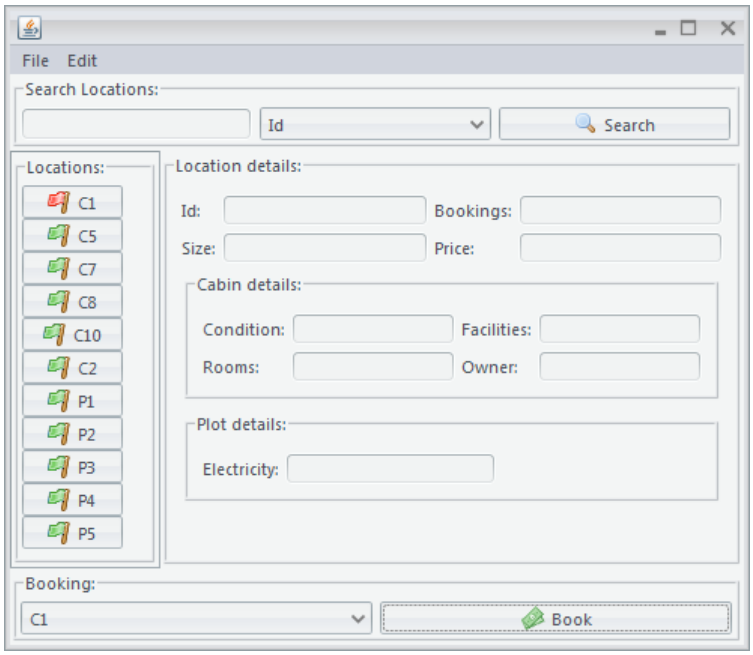

Figure 8: Successfully Booked location marked with red flag.

A short sequence-diagram of the successful booking looks as follows:

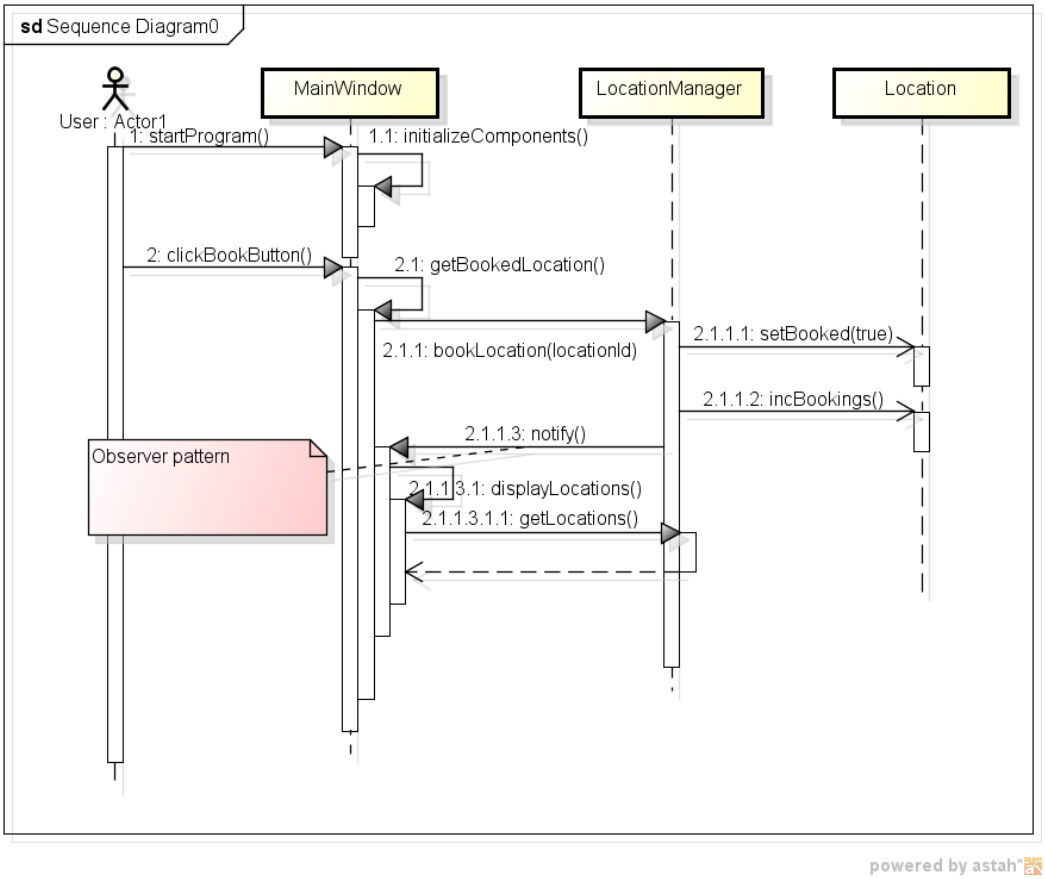

Figure 9: Sequence diagram of a successful booking process.

4.2 DISPLAY LOCATION INFORMATION

To display a location's information, the user simply clicks on one of the buttons listed in the main GUI.

This will trigger the `actionPerformed`-event of the button that is linked to the `actionListener` implemented in the `MainWindow`.

In the actionListener the source of the event is cast to a `JButton`, the text of the button (equal to a location's id) is read and the `displayLocation`-method is called with the string `id`.

The `displayLocation`-method first retrieves the location associated with the `id`, then fills out the fields common to cabins and plots (id, bookings, price, size). Then it is determined whether the displayed location is a cabin or plot. According to the result the location is cast to the appropriate type and the remaining fields are filled in.

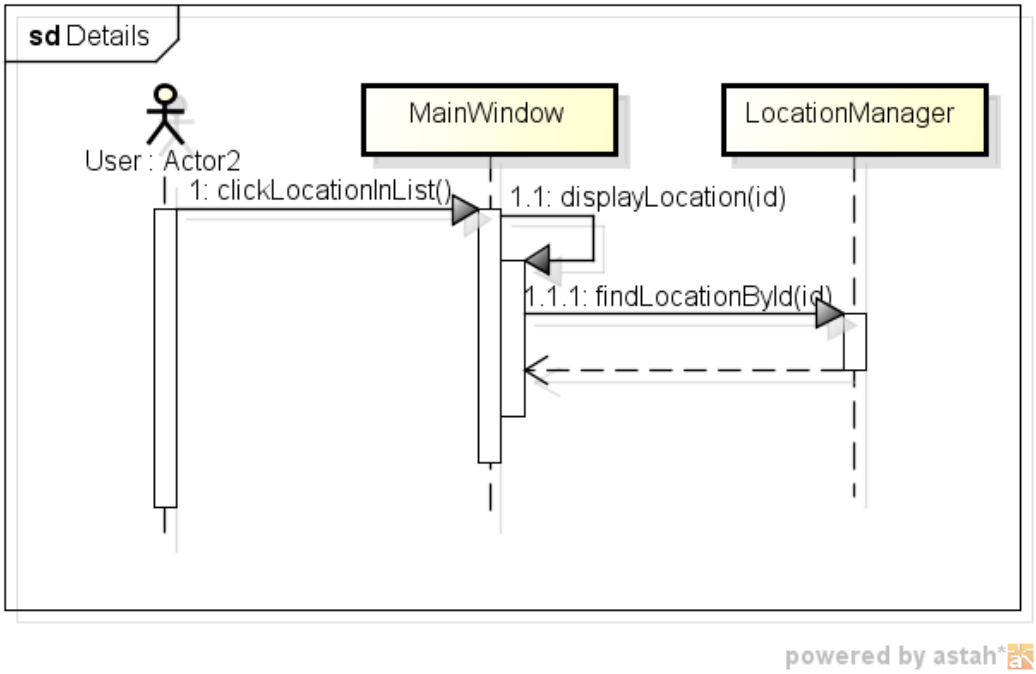

Figure 10: Sequence diagrams of displaying location details.

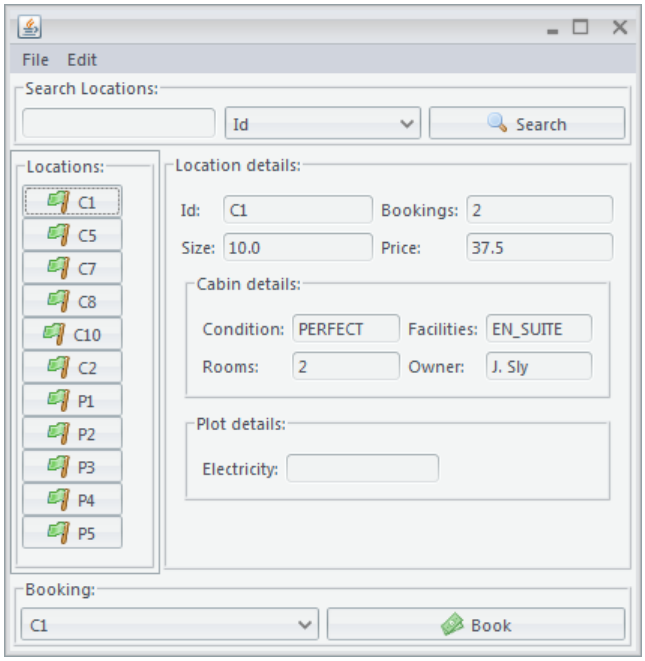

Figure 11: Location details are displayed in the gui.

5 EXCEPTION HANDLING

Input	Output
Already booked location should be booked	Info-window is displayed
Non-existent location should be booked	Not possible via combo-box
Non-matching search string entered	No results will be returned
No search string entered	All locations will be displayed
Already booked location should be booked	Info-window is displayed
Invalid cabin shall be loaded	Entry is skipped
Invalid plot shall be loaded	Entry is skipped
Invalid location shall be displayed	Not possible due to dynamically generated buttons

Table 1: Performed tests.

Part II

APPENDIX

APPENDIX

A.1 SOURCE CODE

Package: uk.heriotwatt.sef

Listing 1: uk.heriotwatt.sef.Starter.java

```
package uk.heriotwatt.sef;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;

import uk.heriotwatt.sef.model.Cabin;
import uk.heriotwatt.sef.model.GenericFileHandler;
import uk.heriotwatt.sef.model.LocationManager;
import uk.heriotwatt.sef.model.Plot;
import uk.heriotwatt.sef.view.MainWindow;

public class Starter {

        /**
         * @param args
         */
        public static void main(String[] args) {
                JFrame.setDefaultLookAndFeelDecorated(true);
                SwingUtilities.invokeLater(new Runnable() {
                      public void run() {
                        try {
                          UIManager.setLookAndFeel("org.pushingpixels.
                              substance.api.skin.
                              SubstanceOfficeSilver2007LookAndFeel");
                        } catch (Exception e) {
                          System.out.println("Substance Graphite failed to
                              initialize");
                        }
                        GenericFileHandler<Cabin> cfh = new GenericFileHandler
                            <Cabin>("cabins.csv");
                        GenericFileHandler<Plot> lfh = new GenericFileHandler<
                            Plot>("plots.csv");
```

```
                        LocationManager manager = new LocationManager(cfh, lfh
                            );
                        MainWindow mw = new MainWindow(manager);
                                mw.setVisible(true);
                    }
                });
        }

}
```

Package: uk.heriotwatt.sef.view

Listing 2: uk.heriotwatt.sef.view.MainWindow.java

```
package uk.heriotwatt.sef.view;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.LinkedList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;

import javax.swing.BoxLayout;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;

import net.miginfocom.swing.MigLayout;
import uk.heriotwatt.sef.model.Cabin;
```

```
import uk.heriotwatt.sef.model.CabinNotFoundException;
import uk.heriotwatt.sef.model.Location;
import uk.heriotwatt.sef.model.LocationAlreadyBookedException;
import uk.heriotwatt.sef.model.LocationManager;
import uk.heriotwatt.sef.model.Plot;

public class MainWindow extends JFrame implements ActionListener, Observer {

        /**
         *
         */
        private static final long serialVersionUID = -2899688509417738813L;
        private JPanel bookingPanel;
        private JPanel locationOverviewPanel;
        private JPanel locationDetails;
        private JLabel idLabel;
        private JTextField idTextField;
        private JLabel sizeLabel;
        private JTextField sizeTextField;
        private JMenuBar menuBar;
        private JMenu fileMenu;

        private List<JButton> locationButtons;
        private JPanel searchPanel;
        private JTextField searchTextField;
        private JComboBox categoryComboBox;
        private JButton searchButton;
        private JComboBox locationComboBox;
        private JButton bookButton;
        private JTextField priceTextField;
        private JLabel priceLabel;

        private LocationManager manager;
        private JPanel plotDetails;
        private JPanel cabinDetails;
        private JTextField conditionTextField;
        private JTextField facilitiesTextField;
        private JTextField roomTextField;
        private JTextField ownerTextField;
        private JTextField electricityTextField;
        private JTextField bookingsTextField;
        private LocationPresenter presenter;

        public MainWindow(LocationManager manager) {
                this.manager = manager;
                this.manager.addObserver(this);
                this.presenter = new LocationPresenter(manager);
```

```
        this.initializeComponents();
        this.displayLocations(manager.getLocations());
    }

    private void initializeComponents() {

        initializeMenuBar();
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        searchPanel = new JPanel();
        this.initializeSearchPanel(searchPanel);
        locationOverviewPanel = new JPanel();
        this.initializeLocationsPanel(locationOverviewPanel);
        JScrollPane scrollPane = new JScrollPane(locationOverviewPanel
            );
        locationDetails = new JPanel(new MigLayout("", "[][grow][][]["
            grow]"));
        this.initializeLocationDetails(locationDetails);
        bookingPanel = new JPanel(new MigLayout());
        this.initializeBookingPanel(bookingPanel);

        panel.add(searchPanel, BorderLayout.NORTH);
        panel.add(scrollPane, BorderLayout.WEST);
        panel.add(locationDetails, BorderLayout.CENTER);
        panel.add(bookingPanel, BorderLayout.SOUTH);

        this.add(panel);
        this.setSize(400, 400);
    }

    private void initializeSearchPanel(JPanel searchPanel2) {
        Border border = new TitledBorder("Search Locations:");
        searchPanel2.setBorder(border);

        GridLayout gl = new GridLayout(0, 3, 5, 0);
        searchPanel2.setLayout(gl);

        searchTextField = new JTextField();
        categoryComboBox = new JComboBox();
        categoryComboBox.addItem("Id");
        categoryComboBox.addItem("Price");
        categoryComboBox.addItem("Area");
        categoryComboBox.addItem("Booked");
        searchButton = new JButton("Search", new ImageIcon(
                        "resources/magnifier.png"));
```

```
            searchButton.addActionListener(new ActionListener() {

                    @Override
                    public void actionPerformed(ActionEvent e) {
                            String searchString = searchTextField.getText
                                ();
                            String category = categoryComboBox.
                                getSelectedItem().toString();
                            List<Location> searchedList = presenter.
                                searchLocations(searchString, category);
                            displayLocations(searchedList);
                    }
            });
            searchPanel2.add(searchTextField);
            searchPanel2.add(categoryComboBox);
            searchPanel2.add(searchButton);
    }

    private void initializeMenuBar() {
            menuBar = new JMenuBar();

            /*
             * Creating the file submenu.
             */
            fileMenu = new JMenu("File");
            JMenuItem exitItem = new JMenuItem("Close");
            exitItem.setIcon(new ImageIcon("resources/door_out.png"));
            exitItem.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                            manager.saveLocations();
                            System.exit(NORMAL);
                    }
            });

            fileMenu.add(exitItem);

            /*
             * Creating the edit submenu.
             */
            JMenu editMenu = new JMenu("Edit");
            JMenu orderMenu = new JMenu("Order by:");
            orderMenu.setIcon(new ImageIcon("resources/sort_ascending.png"
                ));
            JMenuItem orderByIdItem = new JMenuItem("Id");
            orderByIdItem.setIcon(new ImageIcon("resources/key.png"));
            orderByIdItem.addActionListener(new ActionListener() {
```

```
                @Override
                public void actionPerformed(ActionEvent e) {
                        presenter.orderLocations(true);
                }
        });

        JMenuItem orderByCostItem = new JMenuItem("Price");
        orderByCostItem.setIcon(new ImageIcon("resources/money_dollar.
            png"));
        orderByCostItem.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                        presenter.orderLocations(false);
                }
        });

        JMenuItem showSummaryItem = new JMenuItem("Show Summary");
        showSummaryItem.setIcon(new ImageIcon("resources/table.png"));
        showSummaryItem.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                        OverviewWindow window = new OverviewWindow(
                            manager.getLocations());
                        window.setVisible(true);
                }
        });
        orderMenu.add(orderByIdItem);
        orderMenu.add(orderByCostItem);
        editMenu.add(orderMenu);
        editMenu.add(showSummaryItem);

        menuBar.add(fileMenu);
        menuBar.add(editMenu);
        this.setJMenuBar(menuBar);
}

private void initializeLocationDetails(JPanel locationDetails) {
        Border border = new TitledBorder("Location details:");
        locationDetails.setBorder(border);
        idLabel = new JLabel("Id:");
        locationDetails.add(idLabel);
        idTextField = new JTextField();
        locationDetails.add(idTextField, "growx");
        JLabel bookingsLabel = new JLabel("Bookings:");
```

```
            locationDetails.add(bookingsLabel);
            bookingsTextField = new JTextField();
            locationDetails.add(bookingsTextField, "wrap, growx");
            sizeLabel = new JLabel("Size:");
            locationDetails.add(sizeLabel);
            sizeTextField = new JTextField();
            locationDetails.add(sizeTextField, "growx");
            priceLabel = new JLabel("Price:");
            priceTextField = new JTextField();
            locationDetails.add(priceLabel);
            locationDetails.add(priceTextField, "growx, wrap");

            cabinDetails = new JPanel(new MigLayout("", "[][grow][][grow]"
                ));

            Border cabinBorder = new TitledBorder("Cabin details:");
            cabinDetails.setBorder(cabinBorder);
            Component conditionLabel = new JLabel("Condition:");
            cabinDetails.add(conditionLabel);
            conditionTextField = new JTextField();
            cabinDetails.add(conditionTextField, "growx");
            JLabel facilitiesLabel = new JLabel("Facilities:");
            cabinDetails.add(facilitiesLabel);
            facilitiesTextField = new JTextField();
            cabinDetails.add(facilitiesTextField, "growx, wrap");
            JLabel cabinRoomsLabel = new JLabel("Rooms:");
            cabinDetails.add(cabinRoomsLabel);
            roomTextField = new JTextField();
            cabinDetails.add(roomTextField, "growx");
            JLabel ownerLabel = new JLabel("Owner:");
            cabinDetails.add(ownerLabel);
            ownerTextField = new JTextField();
            cabinDetails.add(ownerTextField, "growx");

            plotDetails = new JPanel(new MigLayout("", "[][grow][][grow]")
                );
            Border plotBorder = new TitledBorder("Plot details:");
            plotDetails.setBorder(plotBorder);
            JLabel electricityJLabel = new JLabel("Electricity:");
            plotDetails.add(electricityJLabel);
            electricityTextField = new JTextField();
            plotDetails.add(electricityTextField, "growx");

            locationDetails.add(cabinDetails, "grow, wrap, span 4");
            locationDetails.add(plotDetails, "grow, span 4");
    }
```

```
    private void initializeBookingPanel(JPanel bookingPanel) {
            Border border = new TitledBorder("Booking:");
            bookingPanel.setBorder(border);

            GridLayout gl = new GridLayout(0, 2, 5, 0);
            bookingPanel.setLayout(gl);

            locationComboBox = new JComboBox();
            bookButton = new JButton("Book", new ImageIcon("resources/
                money.png"));
            bookButton.addActionListener(new ActionListener() {

                    @Override
                    public void actionPerformed(ActionEvent e) {
                            String bookedLocation = locationComboBox.
                                getSelectedItem()
                                            .toString();
                            try {
                                    manager.bookLocation(bookedLocation);
                            } catch (NumberFormatException e2) {
                                    e2.printStackTrace();
                            } catch (LocationAlreadyBookedException e1) {
                                    JOptionPane.showMessageDialog(null, "
                                        Location already booked.");
                                    e1.printStackTrace();
                            }

                    }
            });
            bookingPanel.add(locationComboBox);
            bookingPanel.add(bookButton);
    }

    private void initializeLocationsPanel(JPanel panel) {
            this.locationButtons = new LinkedList<JButton>();
            panel.setLayout(new BoxLayout(locationOverviewPanel,
                            BoxLayout.PAGE_AXIS));
            Border border = new TitledBorder("Locations:");
            panel.setBorder(border);
    }

    public void setLocationBooked(JButton button, boolean isBooked) {
            if (isBooked) {
                    Icon bookedIcon = new ImageIcon("resources/flag_red.
                        png");
                    button.setIcon(bookedIcon);
            } else {
```

```
                Icon freeIcon = new ImageIcon("resources/flag_green.
                    png");
                button.setIcon(freeIcon);
        }
}

private void displayLocations(List<Location> locations) {
        locationButtons.clear();
        this.locationOverviewPanel.removeAll();
        for (Location location : locations) {
                JButton button = new JButton(location.getId());
                this.setLocationBooked(button, location.isBooked());
                button.addActionListener(this);
                locationButtons.add(button);
                this.locationOverviewPanel.add(button);

                this.locationComboBox.addItem(location.getId());
        }
        this.repaint();
}

@Override
public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source instanceof JButton) {
                JButton button = (JButton) source;
                String id = button.getText();
                this.displayLocation(id);
        }

}

private void displayLocation(String id) {
        Location location;
        try {
                location = presenter.findLocationById(id);
                this.idTextField.setText(location.getId());
                this.bookingsTextField.setText(String.valueOf(location
                    .getBookings()));
                String cost = String.valueOf(location.getCost());
                this.priceTextField.setText(cost);
                String size = String.valueOf(location.getSize());
                this.sizeTextField.setText(size);
                if (location instanceof Cabin){
                        Cabin cab = (Cabin) location;
                        this.facilitiesTextField.setText(cab.
                            getFacilities().toString());
```

```
                                this.conditionTextField.setText(cab.
                                    getCondition().toString());
                                this.ownerTextField.setText(cab.getOwner().
                                    getInitPeriodLast());
                                this.roomTextField.setText(String.valueOf(cab.
                                    getBeds()));
                                this.clearPlotFields();
                        }
                        else {
                                Plot plot = (Plot) location;
                                this.electricityTextField.setText(String.
                                    valueOf(plot.isHasElectricity()));
                                this.clearCabinFields();
                        }
                } catch (CabinNotFoundException e) {
                        e.printStackTrace();
                }
        }

        private void clearPlotFields() {
                this.electricityTextField.setText("");
        }

        private void clearCabinFields() {
                this.facilitiesTextField.setText("");
                this.conditionTextField.setText("");
                this.ownerTextField.setText("");
                this.roomTextField.setText("");
        }

        @Override
        public void update(Observable o, Object arg) {
                this.reloadData();
        }

        private void reloadData() {
                this.displayLocations(this.presenter.getLocations());
        }
}
```

Listing 3: uk.heriotwatt.sef.view.OverviewWindow.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package uk.heriotwatt.sef.view;
```

```
import java.awt.BorderLayout;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import uk.heriotwatt.sef.model.Location;

/**
 *
 * @author Florian Bergmann
 */
public class OverviewWindow extends JFrame {

        private JTable table;
        private JScrollPane scrollPane;

        public OverviewWindow(List<Location> list) {
                this.initializeComponents(list);
        }

        private void initializeComponents(List<Location> list) {
                JPanel panel = new JPanel(new BorderLayout());
                this.initializeOverviewPanel(panel, list);
                this.add(panel);
                this.setSize(400, 400);
        }

        private void initializeOverviewPanel(JPanel panel, List<Location> list
            ) {
                Border border = new TitledBorder("Overview:");
                panel.setBorder(border);
                table = new JTable(new LocationTableModel(list));
                scrollPane = new JScrollPane(table);
                table.setFillsViewportHeight(true);

                panel.add(scrollPane, BorderLayout.CENTER);
        }
}
```

Listing 4: uk.heriotwatt.sef.view.LocationTableModel.java

```
package uk.heriotwatt.sef.view;
```

```
import java.util.Arrays;
import java.util.List;
import java.util.Vector;

import javax.swing.table.AbstractTableModel;

import uk.heriotwatt.sef.model.Cabin;
import uk.heriotwatt.sef.model.Location;
import uk.heriotwatt.sef.model.Plot;

public class LocationTableModel extends AbstractTableModel {

        /**
         *
         */
        private static final long serialVersionUID = -8059706006230392340L;
        private Vector<String> columnNames;
        private Vector<Location> columns;

        public LocationTableModel(List<Location> columns) {
                String[] columnName = { "Id", "Size", "Booked", "Price", "
                    Condition",
                                "Facilities", "Rooms", "Owner", "Electricity"
                                    };
                this.columnNames = new Vector<String>();
                for (String string : columnName) {
                        this.columnNames.add(string);
                }
                this.columns = new Vector<Location>();
                for (Location location : columns) {
                        this.columns.add(location);
                }
        }

        @Override
        public int getColumnCount() {
                return this.columnNames.size();
        }

        public String getColumnName(int col) {
                return columnNames.get(col).toString();
        }

        @Override
        public int getRowCount() {
                return this.columns.size();
        }
```

```
@Override
public Object getValueAt(int arg0, int arg1) {
        Location loc = columns.get(arg0);
        if (arg1 < 4) {
                switch (arg1) {
                case 0:
                        return loc.getId();
                case 1:
                        return loc.getSize();
                case 2:
                        return loc.isBooked();
                case 3:
                        return loc.getCost();
                default:
                        break;
                }
        } else if (arg1 < 8) {
                if (loc instanceof Cabin) {
                        Cabin cab = (Cabin) loc;
                        switch (arg1) {
                        case 4:
                                return cab.getCondition().toString();
                        case 5:
                                return cab.getFacilities().toString();
                        case 6:
                                return cab.getBeds();
                        case 7:
                                return cab.getOwner().
                                    getInitPeriodLast();
                        default:
                                break;
                        }
                } else {
                        return "——";
                }
        } else {
                if (loc instanceof Plot) {
                        Plot plot = (Plot) loc;
                        switch (arg1) {
                        case 8:
                                return plot.isHasElectricity();
                        default:
                                break;
                        }
                } else {
                        return "——";
```

```
                        }
                }
                return "—";
        }

        @Override
        public Class getColumnClass(int c) {
                return getValueAt(0, c).getClass();
        }

}
```

Listing 5: uk.heriotwatt.sef.view.LocationPresenter.java

```
package uk.heriotwatt.sef.view;


import java.util.List;

import uk.heriotwatt.sef.model.CabinNotFoundException;
import uk.heriotwatt.sef.model.Location;
import uk.heriotwatt.sef.model.LocationManager;


public class LocationPresenter {

        private LocationManager manager;
        private List<Location> locations;

        public LocationPresenter(LocationManager manager2) {
                this.manager = manager2;
                this.locations = this.manager.getLocations();
        }

        public List<Location> getAllLocations()
        {
                this.locations = this.manager.getLocations();
                return this.locations;
        }

        public List<Location> getLocations()
        {
                return this.locations;
        }

        public List<Location> searchLocations(String search, String category)
            {
```

```
                this.locations = this.manager.searchLocations(search, category
                    );
                return this.locations;
        }

        public List<Location> orderLocations(boolean byId)
        {
                return this.locations = this.manager.orderLocations(this.
                    locations, byId);
        }

        public Location findLocationById(String id) throws
            CabinNotFoundException {
                return this.manager.findLocationById(id);
        }
}
```

Package: uk.heriotwatt.sef.model

Listing 6: uk.heriotwatt.sef.model.Cabin.java

```
package uk.heriotwatt.sef.model;

import java.util.Formatter;
import java.util.Locale;

import uk.heriotwatt.sef.model.interfaces.ICsvSerialisable;

/**
 * Stores values associated with a cabin.
 *
 * @author fhb2
 *
 */
public class Cabin extends Location implements ICsvSerialisable<Cabin> {

        private int[] beds;
        private Facilities facilities;
        private Name owner;
        private Condition condition;

        private PriceMapping data;

        private final int MINIMUM_NUMBER_OF_BEDS = 2;
        private final int MAXIMUM_NUMBER_OF_BEDS = 8;
```

```
private final double BASIC_COST = 10;
private final int BED_TO_ROOM_RATIO_MULTIPLIER = 5;

public Cabin() {
        super(null, 0, 0);
        this.data = new PriceMapping();
}

/**
 * Creates a new cabin object with the specified values.
 *
 * @param cabinNumber
 *            The cabin number.
 * @param beds
 *            The array of beds in the cabin..
 * @param size
 *            The size of the cabin.
 * @param facilities
 *            The facilites of the cabin.
 * @param owner
 *            The owner of the cabin.
 * @param condition
 *            The condition of the cabin.
 * @param bookings
 */
public Cabin(String cabinNumber, int[] beds, double size,
                Facilities facilities, Name owner, Condition condition
                    , int bookings) {
        super(cabinNumber, size, bookings);
        this.beds = beds;
        this.facilities = facilities;
        this.owner = owner;
        this.condition = condition;
        this.data = new PriceMapping();
}

/**
 * @return Array of beds.
 */
public int[] getNumberOfBeds() {
        return beds;
}

/**
 * Sets the beds.
 *
 * @param numberOfBeds
```

```
     *            The new array of beds.
     */
    public void setNumberOfBeds(int[] numberOfBeds) {
            if (numberOfBeds.length > 0) {
                    int bedsInArray = this.calculateNumberOfBeds(
                        numberOfBeds);
                    if (bedsInArray >= MINIMUM_NUMBER_OF_BEDS
                                    && bedsInArray <=
                                        MAXIMUM_NUMBER_OF_BEDS) {
                            this.beds = numberOfBeds;
                    } else {
                            throw new IllegalArgumentException(
                                            String.format(
                                                            "Only between
                                                                %d and %d
                                                                beds can
                                                                be placed
                                                                in a cabin
                                                                .",
                                                            MINIMUM_NUMBER_OF_BEDS
                                                                ,
                                                                MAXIMUM_NUMBER_OF_BEDS
                                                                ));
                    }
            } else {
                    throw new IllegalArgumentException(
                                    "The number of beds must be greater
                                        than 0.");
            }
    }

    /**
     * @return The facilities of the cabin.
     */
    public Facilities getFacilities() {
            return facilities;
    }

    /**
     * Attempts to set the facilites of the cabin.
     *
     * @param facilities
     */
    public void setFacilities(Facilities facilities) {
            this.facilities = facilities;
    }
```

```
/**
 * @return The owner of the cabin.
 */
public Name getOwner() {
        return owner;
}

/**
 * Sets the owner of the cabin.
 *
 * @param owner
 */
public void setOwner(Name owner) {
        this.owner = owner;
}

/**
 * @return The size of the cabin.
 */
public double getSize() {
        return size;
}

/**
 * Sets the size of the cabin.
 *
 * @param size
 *            The new size (must be bigger than 0)
 */
public void setSize(double size) {
        if (size >= 0) {
                this.size = size;
        } else {
                throw new IllegalArgumentException("Size must be
                    positive.");
        }
}

/**
 * The cost is calculated based on different factors: - The condition.
     - The
 * facilities. - The size. - The beds/rooms present (The less beds per
     room
 * the more expensive). The values associated with the first three are
 * stored in {@link PriceMapping}
 *
 * @return The cost if the cabin.
```

```
     */
    public double getCost() {
            double cost = BASIC_COST;

            double conditionModifier = this.data.getConditionPrice(this.
                condition);
            double faciltiesModifier = this.data.getFacilityPrice(this.
                facilities);
            double sizeModifier = this.data.getSizeModifier(this.size);
            double bedToRoomRatio = this.calculateRoomToBedRatio();

            cost = BASIC_COST + conditionModifier + faciltiesModifier
                            + sizeModifier
                            + (BED_TO_ROOM_RATIO_MULTIPLIER *
                                bedToRoomRatio);

            return cost;
    }

    /**
     * @return The condition.
     */
    public Condition getCondition() {
            return condition;
    }

    /**
     * Sets the condition of the cabin.
     *
     * @param condition
     *            New condition to be set.
     */
    public void setCondition(Condition condition) {
            this.condition = condition;
    }

    /**
     * @return The number of beds in the cabin.
     */
    public int getBeds() {
            return this.calculateNumberOfBeds(this.beds);
    }

    /**
     * Calculates the room to bed ratio.
     *
     * @return The room to bed ratio.
```

```
 */
public double calculateRoomToBedRatio() {
        int rooms = this.getNumberOfBeds().length;
        int beds = this.calculateNumberOfBeds(this.beds);
        double bedToRoomRatio = rooms / beds;
        return bedToRoomRatio;
}

public String toString() {
        String string = String
                        .format("Number: %s\nCondition: %s\nFacilities
                            : %s\nOwner: %s\nNumer of beds: %d",
                                        this.getId(), this.
                                            getCondition(),
                                        this.getFacilities(), this.
                                            owner.toString(),
                                        this.getBeds());
        return string;
}

private int calculateNumberOfBeds(int[] numberOfBeds) {
        int result = 0;
        for (int i : numberOfBeds) {
                result += i;
        }
        return result;
}

@Override
public String csvRepresentation() {
        StringBuilder sb = new StringBuilder();
        Formatter formatter = new Formatter(sb, Locale.UK);
        formatter.format("%s,%f,%s,%s,%s,%s,%s,%d", this.getId(), this
                        .getSize(), this.facilities.toString(), this.
                            condition
                        .toString(), this.getOwner().getFirstName(),
                            this.getOwner()
                        .getMiddleName(), this.getOwner().getLastName
                            (), this.getBookings());
        for (int i : this.beds)
        {
                formatter.format(",%d", i);
        }
        return sb.toString();
}

@Override
```

```
public Cabin createFromString(String string) {
        Cabin cabin = null;
        String[] splitList = string.split(",");
        String errorString = "";

        String cabinNumber = "";
        double size = 0;
        Facilities facilities = null;
        Condition condition = null;
        Name name;
        int bookings = 0;
        int[] beds;
        try {
                cabinNumber = splitList[0];
        } catch (NumberFormatException e) {
                errorString += splitList[0];
        }
        try {
                size = Double.parseDouble(splitList[1]);
        } catch (NumberFormatException e) {
                errorString += ", " + splitList[1];
        }
        try {
                facilities = Facilities.valueOf(splitList[2]);
        } catch (IllegalArgumentException e) {
                errorString += ", " + splitList[2];
        }
        try {
                condition = Condition.valueOf(splitList[3]);
        } catch (IllegalArgumentException e) {
                errorString += ", " + splitList[3];
        }
        name = new Name(splitList[4], splitList[5], splitList[6]);
        try {
                bookings = Integer.parseInt(splitList[7]);
        } catch (IllegalArgumentException e) {
                errorString += ", " + splitList[7];
        }
        beds = new int[splitList.length - 8];
        for (int i = 8; i < splitList.length; i++) {
                try {
                        beds[i - 8] = Integer.parseInt(splitList[i]);
                } catch (Exception e) {
                        errorString += ", " + beds[i - 8];
                }
        }
        if (errorString.length() == 0) {
```

```
                        cabin = new Cabin(cabinNumber, beds, size, facilities,
                             name,
                                        condition, bookings);
                        return cabin;
                } else {
                        throw new IllegalArgumentException(
                                        String.format(
                                                        "There were errors
                                                            parsing the line:\
                                                            n%s\nThe following
                                                             arguments were
                                                            violating the
                                                            format:%s",
                                                        string, errorString));
                }
        }
}
```

Listing 7: uk.heriotwatt.sef.model.LocationManager.java

```
package uk.heriotwatt.sef.model;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Observable;

public class LocationManager extends Observable {

        private List<Location> locations;

        public List<Location> getLocations() {
                return locations;
        }

        private GenericFileHandler<Cabin> cfh;
        private GenericFileHandler<Plot> lfh;

        public LocationManager(GenericFileHandler<Cabin> cfh,
                GenericFileHandler<Plot> lfh) {
                this.locations = new ArrayList<Location>();
                this.cfh = cfh;
                this.lfh = lfh;
                this.locations.addAll(this.cfh.readFromFile(new Cabin()));
                this.locations.addAll(this.lfh.readFromFile(new Plot()));
        }
```

```
public void addLocation(Location cab) {
        this.locations.add(cab);
}

/**
 * Attempts to find a cabin with the provided cabinNumber in the cabin
    -List.
 *
 * @param id
 *            The cabinnumber of the cabin to be returned
 * @return The first cabin in the list with the corresponding
    cabinnumber.
 * @throws CabinNotFoundException
 *             If no cabin with the provided number could be found.
 */
public Location findLocationById(String id) throws
    CabinNotFoundException {
        Location locationFound = null;
        for (Location location : this.locations) {
                if (location.getId().equals(id)) {
                        locationFound = location;
                        break;
                }
        }
        if (locationFound != null) {
                return locationFound;
        } else {
                throw new CabinNotFoundException(String.format(
                        "The cabin with number %d was not in the list.
                            ", id));
        }
}

/**
 * Retrieves that cabin at the specified position in the cabin list.
 *
 * @param index
 *            The position for which the cabin should be returned.
 * @return Tha cabin.
 */
public Location getCabinAtIndex(int index) {
        if (index < this.getNumberOfCabins()) {
                return this.locations.get(index);
        } else {
                throw new IndexOutOfBoundsException();
        }
```

```
}

/**
 * Returns the maximum possible income that could be achieved.
    Therefore the
 * cost for all cabins are added up.
 *
 * @return The added cost of all cabins.
 */
public double getMaximumPossibleIncome() {
        double result = 0;
        for (Location cabin : this.locations) {
                result += cabin.getCost();
        }
        return result;
}

/**
 * Returns the cost for the cheapest cabin.
 *
 * @return The cost of the cheapest cabin.
 * @throws NoCabinsException
 */
public double getCheapestCabinCost() throws NoCabinsException {
        if (this.locations.size() > 0) {
                Location cheapestCab = null;
                for (Location cab : this.locations) {
                        if (cheapestCab == null) {
                                cheapestCab = cab;
                        }
                        if (cab.getCost() < cheapestCab.getCost()) {
                                cheapestCab = cab;
                        }
                }
                return cheapestCab.getCost();
        } else {
                throw new NoCabinsException(
                        "There are no cabins present. Insert cabins
                            first.");
        }

}

/**
 * Returns the cost for the most expensive cabin.
 *
 * @return The cost of the most expensive cabin.
```

```
     * @throws NoCabinsException
     */
    public double getExpensiveCabinCost() throws NoCabinsException {
            if (this.locations.size() > 0) {
                    Location expensiveCab = null;
                    for (Location cab : this.locations) {
                            if (expensiveCab == null) {
                                    expensiveCab = cab;
                            }
                            if (cab.getCost() > expensiveCab.getCost()) {
                                    expensiveCab = cab;
                            }
                    }
                    return expensiveCab.getCost();
            } else {
                    throw new NoCabinsException(
                            "There are no cabins present. Insert cabins
                                first.");
            }
    }

    /**
     * Returns the number of cabins currently registered in the list.
     *
     * @return The number of cabins.
     */
    public int getNumberOfCabins() {
            return this.locations.size();
    }

    public void bookLocation(String bookedLocation)
            throws LocationAlreadyBookedException {
            try {
                    Location loc = this.findLocationById(bookedLocation);
                    if (!loc.isBooked) {
                            loc.isBooked = true;
                            loc.bookings++;
                            this.setChanged();
                            this.notifyObservers(true);
                            this.clearChanged();
                    } else {
                            throw new LocationAlreadyBookedException(
                                String.format(
                                    "The location with id %s is already
                                        booked.",
                                    bookedLocation));
                    }
```

```
        } catch (CabinNotFoundException e) {
                e.printStackTrace();
        }
}

/**
 * Orders the present list of locations.
 * Either by their id, or by price.
 * @param locations2
 *
 * @param byId True of order by id, false if order by price.
 */
public List<Location> orderLocations(List<Location> locations2,
    boolean byId) {
        List<Location> returnList;
        if (byId) {
                Collections.sort(locations2, new IdComparator());
        } else {
                Collections.sort(locations2, new PriceComparator());
        }
        this.setChanged();
        this.notifyObservers(true);
        this.clearChanged();
        returnList = locations2;
        return returnList;
}

/**
 * Searches the list of location for matches with the provided
     parameters.
 *
 * @param searchString The string that should be matched. In case of
     the compared attribute being not a string, only a perfect match
     will return true.
 * @param category The category specifies the compared attributes: id,
      price, booked, area.
 * @return A list of the matching locations. If no search string is
     provided then all locations are returned.
 */
public List<Location> searchLocations(String searchString, String
    category) {
        List<Location> returnList = new LinkedList<Location>();
        if (searchString != null) {
                if (!searchString.isEmpty()) {
                        if (category.equals("Id")) {
                                for (Location location : this.
                                    locations) {
```

```
                                    if (location.getId().contains(
                                        searchString)) {
                                            returnList.add(
                                                location);
                                    }
                            }
                        } else if (category.equals("Price")) {
                                for (Location location : this.
                                    locations) {
                                    if (location.getCost() == (
                                        Double
                                            .parseDouble(
                                                searchString))) {
                                            returnList.add(
                                                location);
                                    }
                                }
                        } else if (category.equals("Area")) {
                                for (Location location : this.
                                    locations) {
                                    if (location.getSize() == (
                                        Double
                                            .parseDouble(
                                                searchString))) {
                                            returnList.add(
                                                location);
                                    }
                                }
                        } else if (category.equals("Booked")) {
                                for (Location location : this.
                                    locations) {
                                    if (location.isBooked() == (
                                        Boolean
                                            .parseBoolean(
                                                searchString))) {
                                            returnList.add(
                                                location);
                                    }
                                }
                        }
                } else {
                        returnList = this.locations;
                }
        } else {
                returnList = this.locations;
        }
        return returnList;
```

```
        }

        public void saveLocations() {
                List<Cabin> cabins = new LinkedList<Cabin>();
                List<Plot> plots = new LinkedList<Plot>();
                for (Location loc : this.locations) {
                        if (loc instanceof Cabin)
                        {
                                cabins.add((Cabin) loc);
                        }
                        else {
                                plots.add((Plot) loc);
                        }
                }
                this.cfh.writeToFile(cabins);
                this.lfh.writeToFile(plots);
        }

}
```

Listing 8: uk.heriotwatt.sef.model.Condition.java

```
package uk.heriotwatt.sef.model;

/**
 * Stores the different possibilities of conditions.
 *
 * @author fhb2
 *
 */
public enum Condition {

        PERFECT, GOOD, FAIR, BAD, IN_SHAMBLES, UNKNOWN

}
```

Listing 9: uk.heriotwatt.sef.model.Facilities.java

```
package uk.heriotwatt.sef.model;

/**
 * Stores the different possibilities of facilities.
 *
 * @author fhb2
 *
 */
public enum Facilities {
```

```
        GENERAL_FACILITES, SEPERATE_BATHROOM, EN_SUITE, UNKNOWN

}
```

Listing 10: uk.heriotwatt.sef.model.PriceList.java

```
package uk.heriotwatt.sef.model;

/**
 * Stores the price modifiers for certain discrete price categories.
 *
 * @author fhb2
 *
 */
public enum PriceList {

        VERY_EXPENSIVE(10.0), EXPENSIVE(7.5), FAIR(5.0), BUDGET(2.5), CHEAP
            (1.0), UNKNOWN(
                        0.0);

        private final double cost;

        private PriceList(double cost) {
                this.cost = cost;
        }

        public double cost() {
                return this.cost;
        }

}
```

Listing 11: uk.heriotwatt.sef.model.PriceMapping.java

```
package uk.heriotwatt.sef.model;

import java.util.HashMap;
import java.util.Map;

/**
 * Class to seperate the pricing mapping from the information of the cabin.
 *
 * @author florian
 *
 */
public class PriceMapping {
```

```
private Map<Condition, Double> conditionPrices;
private Map<Facilities, Double> facilityPrices;

public PriceMapping() {
        this.initializeConditionPriceMapping();
        this.initializeFacilityPriceMapping();
}

public Map<Condition, Double> getConditionPrices() {
        return conditionPrices;
}

public Map<Facilities, Double> getFacilityPrices() {
        return facilityPrices;
}

/*
 * Getters and setters
 */

public double getFacilityPrice(Facilities facilities) {
        return facilityPrices.get(facilities);
}

public double getConditionPrice(Condition condition) {
        return conditionPrices.get(condition);
}

/**
 * Adds Condition - Price pairs to a map. Will be used in the getCost
     ()
 * method.
 */
private void initializeConditionPriceMapping() {
        this.conditionPrices = new HashMap<Condition, Double>();
        this.conditionPrices.put(Condition.PERFECT,
                        PriceList.VERY_EXPENSIVE.cost());
        this.conditionPrices.put(Condition.GOOD, PriceList.EXPENSIVE.
            cost());
        this.conditionPrices.put(Condition.FAIR, PriceList.FAIR.cost()
            );
        this.conditionPrices.put(Condition.BAD, PriceList.BUDGET.cost
            ());
        this.conditionPrices.put(Condition.IN_SHAMBLES, PriceList.
            CHEAP.cost());
}
```

```
        /**
         * Adds Facilities - Price pairs to a map. Will be used in the getCost
             ()
         * method.
         */
        private void initializeFacilityPriceMapping() {
                this.facilityPrices = new HashMap<Facilities, Double>();
                this.facilityPrices.put(Facilities.EN_SUITE,
                                PriceList.VERY_EXPENSIVE.cost());
                this.facilityPrices.put(Facilities.SEPERATE_BATHROOM,
                                PriceList.FAIR.cost());
                this.facilityPrices.put(Facilities.GENERAL_FACILITES,
                                PriceList.BUDGET.cost());
        }

        /**
         * Return the size modifier that can be used to calculate a price for
             a
         * cabin.
         *
         * @param size
         *            The size of the cabin.
         * @return The size modifier according to the provided size of a room.
         */
        public double getSizeModifier(double size) {
                if (size < 20) {
                        return PriceList.BUDGET.cost();
                } else if (size >= 20 && size < 30) {
                        return PriceList.CHEAP.cost();
                } else if (size >= 30 && size < 40) {
                        return PriceList.FAIR.cost();
                } else if (size >= 40 && size < 50) {
                        return PriceList.EXPENSIVE.cost();
                } else {
                        return PriceList.VERY_EXPENSIVE.cost();
                }
        }
}
```

Listing 12: uk.heriotwatt.sef.model.LocationAlreadyBookedException.java

```
package uk.heriotwatt.sef.model;

public class LocationAlreadyBookedException extends Exception {

        /**
```

```
         *
         */
        private static final long serialVersionUID = 3180293198375887403L;

        public LocationAlreadyBookedException(String message)
        {
                super(message);
        }
}
```

Listing 13: uk.heriotwatt.sef.model.CabinNotFoundException.java

```
package uk.heriotwatt.sef.model;

public class CabinNotFoundException extends Exception {

        /**
         * Generated serialVersionUID to allow serialisation.
         */
        private static final long serialVersionUID = -7740644730079198039L;

        public CabinNotFoundException(String msg) {
                super(msg);
        }
}
```

Listing 14: uk.heriotwatt.sef.model.NoCabinsException.java

```
package uk.heriotwatt.sef.model;

public class NoCabinsException extends Exception {

        /**
         * Generated serialVersionUID to allow serialisation.
         */
        private static final long serialVersionUID = 2274177224545932291L;

        public NoCabinsException(String msg) {
                super(msg);
        }
}
```

Listing 15: uk.heriotwatt.sef.model.Name.java

```
package uk.heriotwatt.sef.model;
```

```
//First Name class
//F21SF - Monica
public class Name {
        private String firstName;
        private String middleName;
        private String lastName;

        // constructor to create object with first, middle and last name
        // if there isn't a middle name, that parameter could be an empty
            String
        public Name(String fName, String mName, String lName) {
                firstName = fName;
                middleName = mName;
                lastName = lName;
        }

        // returns the first name
        public String getFirstName() {
                return firstName;
        }

        // returns the last name
        public String getLastName() {
                return lastName;
        }

        // change the last name to the value provided in the parameter
        public void setLastName(String ln) {
                lastName = ln;
        }

        // returns the first name then a space then the last name
        public String getFirstAndLastName() {
                return firstName + " " + lastName;
        }

        // returns the last name followed by a comma and a space
        // then the first name
        public String getLastCommaFirst() {
                return lastName + ", " + firstName;
        }

        // returns name in the format initial, period, space, lastname
        public String getInitPeriodLast() {
                return firstName.charAt(0) + ". " + lastName;
        }
```

```
        public String getInitials() {
                return firstName.charAt(0) + ". " + lastName.charAt(0) + ".";
        }

        public Object getMiddleName() {
                return this.middleName;
        }
}
```

Listing 16: uk.heriotwatt.sef.model.Location.java

```
package uk.heriotwatt.sef.model;

public abstract class Location {

        protected String id;
        protected boolean isBooked;
        protected int bookings;
        protected double size;

        public Location(String number, double size, int bookings2) {
                super();
                this.id = number;
                this.size = size;
                this.bookings = bookings2;
        }

        public String getId() {
                return id;
        }

        public void setId(String id) {
                this.id = id;
        }

        public boolean isBooked() {
                return isBooked;
        }

        public void setBooked(boolean isBooked) {
                this.isBooked = isBooked;
        }

        public int getBookings() {
                return bookings;
        }
```

```
        public void setBookings(int bookings) {
                this.bookings = bookings;
        }

        public double getSize() {
                return size;
        }

        public void setSize(double size) {
                this.size = size;
        }

        public abstract double getCost();

        public String toString()
        {
                String string = String.format("Number: %s.\nBookings: %d.\
                    nBooked: %b\nSize: %d", this.id, this.bookings, this.
                    isBooked, this.size);
                return string;
        }
}
```

Listing 17: uk.heriotwatt.sef.model.Plot.java

```
package uk.heriotwatt.sef.model;

import java.util.Formatter;
import java.util.Locale;

import org.apache.log4j.Logger;

import uk.heriotwatt.sef.model.interfaces.ICsvSerialisable;

public class Plot extends Location implements ICsvSerialisable<Plot> {

        private static final int ELECTRICITY_SURPLUS = 10;

        private final Logger logger = Logger.getLogger(getClass());

        private boolean hasElectricity;

        /**
         * Initializes a new instance of the plot class with the provided
         * attributes.
         *
         * @param plotNumber
```

```
     *            The plot number.
     * @param hasElectricity
     *            The boolean to determine wether the plot has an electric
     *            outlet or not.
     * @param area
     *            The area provided to the traveller.
     */
    public Plot(String plotNumber, boolean hasElectricity, double area,
                    int bookings) {
            super(plotNumber, area, bookings);
            this.hasElectricity = hasElectricity;
    }

    public Plot() {
            super("", 0, 0);
    }

    public boolean isHasElectricity() {
            return hasElectricity;
    }

    public void setHasElectricity(boolean hasElectricity) {
            this.hasElectricity = hasElectricity;
    }

    public double getCost() {
            PriceMapping pm = new PriceMapping();
            double cost = pm.getSizeModifier(this.getSize());
            if (this.isHasElectricity()) {
                    cost += ELECTRICITY_SURPLUS;
            }
            return cost;
    }

    public String toString() {
            String string = String.format("Id: %s\nElectricty: %b\nCost:%d
                ",
                            this.getId(), this.isHasElectricity(), this.
                                getCost());
            return string;
    }

    @Override
    public String csvRepresentation() {
            StringBuilder sb = new StringBuilder();
            Formatter formatter = new Formatter(sb, Locale.UK);
            formatter.format("%s,%f,%b,%d", this.getId(),
```

```
                                this.getSize(), this.isHasElectricity(), this.
                                    getBookings());
                return sb.toString();
        }

        @Override
        public Plot createFromString(String string) {
                Plot plot = null;
                String[] splitList = string.split(",");

                String plotNumber = "";
                double size = 0;
                int bookings = 0;
                boolean hasElectricity = false;
                plotNumber = splitList[0];
                try {
                        size = Double.parseDouble(splitList[1]);
                } catch (NumberFormatException e) {
                        logger.error("Could not parse plot size. Argument was:
                             "
                                        + splitList[1]);
                }
                hasElectricity = Boolean.parseBoolean(splitList[2]);
                try {
                        bookings = Integer.parseInt(splitList[3]);
                } catch (NumberFormatException e) {
                        logger.error("Could not parse bookings. Argument was:
                            "
                                        + splitList[3]);
                }
                plot = new Plot(plotNumber, hasElectricity, size, bookings);
                return plot;
        }
}
```

Listing 18: uk.heriotwatt.sef.model.GenericFileHandler.java

```
package uk.heriotwatt.sef.model;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.LinkedList;
import java.util.List;
```

```
import java.util.Scanner;

import org.apache.log4j.Logger;

import uk.heriotwatt.sef.model.interfaces.FileHandler;
import uk.heriotwatt.sef.model.interfaces.ICsvSerialisable;

public class GenericFileHandler<T extends ICsvSerialisable<T>> implements
                FileHandler<T> {

        private final Logger logger = Logger.getLogger(getClass());

        private String filePath;

        public GenericFileHandler(String filePath) {
                this.filePath = filePath;
        }

        @Override
        public void writeToFile(List<T> list) {
                try {
                        File file = new File(this.filePath);
                        Writer w = new FileWriter(file);
                        BufferedWriter bw = new BufferedWriter(w);
                        for (T t : list) {
                                bw.write(t.csvRepresentation());
                                bw.write("\n");
                        }
                        bw.close();
                } catch (FileNotFoundException e) {
                        logger.error("File not found exception: " + e.toString
                            ());
                } catch (IOException e) {
                        logger.error("IO exception: " + e.toString());
                }
        }

        @Override
        public List<T> readFromFile(T creationT) {
                List<T> tList = new LinkedList<T>();
                try {
                        File file = new File(this.filePath);
                        Scanner scanner = new Scanner(file);
                        while (scanner.hasNext()) {
                                String nextLine = scanner.nextLine();
                                if (nextLine.trim().startsWith("#")) {
```

```
                                System.out.println("Ignoring a
                                    commented out line.");
                            } else {
                                try {
                                    T t = creationT.
                                        createFromString(nextLine)
                                        ;
                                    tList.add(t);
                                } catch (IllegalArgumentException e) {
                                    logger.error("
                                        IllegalArgumentException
                                        when creating object:"
                                                + e.toString()
                                                    );
                                }
                            }
                        }
                } catch (FileNotFoundException e) {
                        logger.error("File not found exception: " + e.toString
                            ());
                }
                return tList;
        }

}
```

Listing 19: uk.heriotwatt.sef.model.IdComparator.java

```
package uk.heriotwatt.sef.model;

import java.util.Comparator;

/**
 * Compares two locations based on their ids.
 *
 * @author florian
 */
public class IdComparator implements Comparator<Location> {

        @Override
        public int compare(Location o1, Location o2) {
                return o1.getId().compareTo(o2.getId());
        }

}
```

Listing 20: uk.heriotwatt.sef.model.PriceComparator.java

```
package uk.heriotwatt.sef.model;

import java.util.Comparator;

/**
 * Compares two locations based on their price.
 *
 * @author florian
 */
public class PriceComparator implements Comparator<Location> {

        @Override
        public int compare(Location o1, Location o2) {
                double costLocationOne = o1.getCost();
                double costLocationTwo = o2.getCost();
                return Double.compare(costLocationOne, costLocationTwo);
        }


}
```

Package: uk.heriotwatt.sef.model.interfaces

Listing 21: uk.heriotwatt.sef.model.interfaces.FileHandler.java

```
package uk.heriotwatt.sef.model.interfaces;

import java.util.List;

public interface FileHandler<T> {

        public void writeToFile(List<T> list);

        public List<T> readFromFile(T creationT);

}
```

Listing 22: uk.heriotwatt.sef.model.interfaces.ICsvSerialisable.java

```
package uk.heriotwatt.sef.model.interfaces;

public interface ICsvSerialisable<T> {

        public String csvRepresentation();
```

```
        public T createFromString(String string);
}
```

A.2 INPUT/OUTPUT FILES

Listing 23: Cabins input & output file

```
C1,10.000000,EN_SUITE,PERFECT,John,Rambo,Sly,2,1,1
C5,45.000000,EN_SUITE,FAIR,Sie,,Gurd,2,3
C7,39.000000,SEPERATE_BATHROOM,PERFECT,Perseus,,Nestor,1,2,2,2,2
C8,38.000000,SEPERATE_BATHROOM,FAIR,Robert,Bruce,Cotton,0,2,1,1
C10,27.000000,EN_SUITE,GOOD,Grimur,,Jonsson,0,2,4,2
C2,35.750000,EN_SUITE,PERFECT,Ben,,Franklin,0,1,1,1
```

Listing 24: Plots input & output file

```
P1,10.000000,true,1
P2,12.000000,false,0
P3,14.000000,true,0
P4,23.000000,false,0
P5,52.000000,true,0
```