**public interface CheckCallButtonInterface {**

**public boolean getDownLit();**

**public boolean getUpLit();**

**}**

public class CallButton **implements CheckCallButtonInterface** {

…

}

public interface CallElevatorSystemInterface {

public void addFloor(int floor) throws IllegalArgumentException;

public void callElevator(int id, Direction.DIRECTION dir);

public int getNextFloor();

public void removeFloor(int floor) throws IllegalArgumentException;

**public boolean checkButton(int floor, Direction.DIRECTION dir);**

**public Direction.DIRECTION getDir();**

}

public class Elevator implements CallElevatorInterface, GetIDInterface {

…

/\*\*

\* Processes a clock tick<br>

\*

\* Precondition: N/A<br>

\* Postcondition: Moves the Elevator to the next scheduled Floor, if there

\* is one. In case of a departure, the Elevator Door is closed. In case of

\* an arrival, the Elevator Door is opened, the TargetFloorButton associated

\* with the current Floor is turned off, and the ElevatorSystem is informed

\* of the arrival<br>

\* Cleanup: N/A<br>

\*

\* @see Door#closeDoor()

\* @see Door#openDoor()

\* @see ElevatorSystem#removeFloor(int)

\* @see TargetFloorButton#setLit(boolean)

\*/

public void tick() {

nextFloor = sys.getNextFloor();

if (nextFloor == -1) {

door.openDoor();

return;

}

if (door.getStatus() == Door.DOOR\_STATUS.OPENED) {

door.closeDoor();

}

if (currentFloor < nextFloor) {

currentFloor++;

} else if (currentFloor > nextFloor) {

currentFloor--;

}

if (currentFloor == nextFloor &&

**(buttons[currentFloor].isLit || (**

**sys.checkButton(currentFloor, Direction.DIRECTION.UP) && sys.getDir() == Direction.DIRECTION.UP ||**

**sys.checkButton(currentFloor, Direction.DIRECTION.DOWN) && sys.getDir() == Direction.DIRECTION.DOWN)**

)){

door.openDoor();

buttons[currentFloor].setLit(false);

if (sys != null) {

sys.removeFloor(currentFloor);

}

nextFloor = -1;

}

}

}

import java.util.concurrent.ConcurrentSkipListMap;

public class ElevatorSystem implements CallElevatorSystemInterface {

…

**/\*\***

**\* Check a call button state for a given direction.**

**\***

**\* Precondition: N/A<br>**

**\* Postcondition: N/A<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @param floor is the floor to get the button for**

**\* @param dir is the direction to check, must be UP or DOWN**

**\* @return true if the button is lit, false if the button is not lit**

**\* @throws IllegalArgumentException if floor is out of range or direction is invalid**

**\*/**

**@Override**

**public boolean checkButton(int floor, Direction.DIRECTION dir) throws IllegalArgumentException {**

**if (floor < 0 || floor >= floors.length) {**

**throw new IllegalArgumentException();**

**}**

**if (dir == Direction.DIRECTION.DOWN) {**

**return floors[floor].getCallButtonInterface().getDownLit();**

**} else if (dir == Direction.DIRECTION.UP) {**

**return floors[floor].getCallButtonInterface().getUpLit();**

**} else {**

**throw new IllegalArgumentException();**

**}**

**}**

/\*\*

\* Computes the next Floor to visit<br>

\*

\* Precondition: N/A<br>

\* Postcondition: The direction of the Elevator and the next target Floor

\* have been set<br>

\* Cleanup: N/A<br>

\*

\*/

public void computeNextFloor() {

int currentFloor = elevator.getCurrentFloor();

if (dir == null) {

dir = Direction.DIRECTION.NONE;

}

if (dir == Direction.DIRECTION.NONE) {

dir = Direction.DIRECTION.UP;

}

if (dir == Direction.DIRECTION.UP) {

**// Does current floor still need servicing in this direction?**

**if (floors[currentFloor].getCallButtonInterface().getUpLit()) {**

**nextFloor = currentFloor;**

**} else {**

// We are headed up, can we go any higher?

nextFloor = (Integer) floorList.higherKey(currentFloor);

}

if (nextFloor != null) {

return;

}

// Nope, let's go down

dir = Direction.DIRECTION.DOWN;

}

// Does current floor still need servicing in this direction?

**if (floors[currentFloor].getCallButtonInterface().getDownLit()) {**

**nextFloor = currentFloor;**

**} else {**

// We are going down, can we go any lower?

nextFloor = (Integer) floorList.lowerKey(currentFloor);

}

if (nextFloor != null) {

return;

}

// Nope. OK, time to rest

dir = Direction.DIRECTION.NONE;

nextFloor = -1;

}

**/\*\***

**\* Get the current direction<br>**

**\***

**\* Precondition: N/A<br>**

**\* Postcondition: N/A<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @return the elevator's direction, UP or DOWN**

**\*/**

**@Override**

**public Direction.DIRECTION getDir() {**

**return dir;**

**}**

**/\*\***

**\* Gets an interface to the selectFloor method of an elevator.**

**\***

**\* Precondition: N/A<br>**

**\* Postcondition: N/A<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @return a CallElevatorInterface object**

**\*/**

**public CallElevatorInterface getCallElevatorInterface() {**

**return (CallElevatorInterface)elevator;**

**}**

**/\*\***

**\* Get an array of floors with callElevator and getID methods.**

**\***

**\* Precondition: N/A<br>**

**\* Postcondition: N/A<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @return**

**\*/**

**public CallFloorInterface[] getCallFloorInterface() {**

**return this.floors;**

}

/\*\*

\* Removes a Floor from the list of scheduled Floors<br>

\*

\* Precondition: N/A<br>

\* Postcondition: The given Floor has been removed from the list of

\* scheduled Floors and the given Floor has also been informed of the

\* Elevator arrival<br>

\* Cleanup: N/A<br>

\*

\* @param floor the floor to remove from the schedule

\* @throws IllegalArgumentException if the floor is out of range

\*

\* @see Floor#arrivedAtFloor(Direction.DIRECTION)

\*/

@Override

public void removeFloor(int floor) throws IllegalArgumentException {

if (floor < 0 || floor >= floors.length) {

throw new IllegalArgumentException();

}

**// Remove up call if moving up**

**if (this.dir == Direction.DIRECTION.UP && checkButton(floor, Direction.DIRECTION.UP)) {**

**floors[floor].arrivedAtFloor(Direction.DIRECTION.UP);**

**}**

**// Remove down call if moving down**

**if (this.dir == Direction.DIRECTION.DOWN && checkButton(floor, Direction.DIRECTION.DOWN)) {**

**floors[floor].arrivedAtFloor(Direction.DIRECTION.DOWN);**

**}**

**// Remove floor if fully serviced**

**if (!floors[floor].callButton.isDownLit && !floors[floor].callButton.isUpLit) {**

**floorList.remove(floor);**

**}**

computeNextFloor();

}

}

public class Floor implements CallFloorInterface, GetIDInterface {

…

**/\*\***

**\* Get a callButton that can be used to check the state of the lights.**

**\***

**\* Precondition: N/A<br>**

**\* Postcondition: N/A<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @return a callButton with getUpLit() and getDownLit() methods**

**\*/**

**public CheckCallButtonInterface getCallButtonInterface() {**

**return callButton;**

**}**

public class UIController implements UIControllerInterface {

…

**/\*\***

**\* Presses the up button on a given floor<br>**

**\***

**\* Preconditions: floor is valid<br>**

**\* Postconditions: The up button has been pressed<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @param floor the floor where the call was made**

**\***

**\* @see CallButton#callElevator(Direction.DIRECTION)**

**\*/**

**@Override**

**public void callUp(int floor) {**

**floors[floor].callElevator(Direction.DIRECTION.UP);**

**}**

**/\*\***

**\* Presses the down button on a given floor<br>**

**\***

**\* Preconditions: floor is valid<br>**

**\* Postconditions: The down button has been pressed<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @param floor the floor where the call was made**

**\***

**\* @see CallButton#callElevator(Direction.DIRECTION)**

**\*/**

**@Override**

**public void callDown(int floor) {**

**floors[floor].callElevator(Direction.DIRECTION.DOWN);**

**}**

**/\*\***

**\* Presses a floor button<br>**

**\***

**\* Preconditions: floor is valid<br>**

**\* Postconditions: The given button has been pressed<br>**

**\* Cleanup: N/A<br>**

**\***

**\* @param floor the floor button that was pressed**

**\***

**\* @see TargetFloorButton#selectFloor()**

**\*/**

**@Override**

**public void selectFloor(int floor) {**

**e.selectFloor(floor);**

**}**

}

public class UIView extends JFrame {

…

final static int PANE\_WIDTH = 700;

final static int PANE\_HEIGHT = 700;

final static int CALL\_BUTTON\_H\_OFFSET = 25;

final static double CALL\_BUTTON\_V\_OFFSET = 2.5;

final static int DOOR\_H\_OFFSET = 100;

final static int TARGET\_BUTTON\_H\_OFFSET = 500;

final static int ELEVATOR\_H\_OFFSET = 200;

protected static int numFloors;

protected JButton[] callUpButtons;

**protected JButton[] callDownButtons;**

**protected JButton[] targetFloorButtons;**

protected ImageIcon upOn;

protected ImageIcon upOff;

protected ImageIcon downOn;

protected ImageIcon downOff;

protected ImageIcon lightOn;

protected ImageIcon lightOff;

protected ImageIcon doorClosed;

protected ImageIcon doorOpen;

protected ImageIcon elevatorDoorClosed;

protected ImageIcon elevatorDoorOpen;

protected JLabel[] callUpIcon;

protected JLabel[] callDownIcon;

protected JLabel[] targetFloorIcon;

protected JLabel[] floorDoorIcon;

protected JLabel elevatorDoorIcon;

protected UIController controller;

/\*\*

\* Sets the up or down call button to a given status<br>

\*

\* Precondition: The floor is valid<br>

\* Postcondition: If the Direction is UP, the up button status has been set

\* to the given status. If the Direction is DOWN, the down button status has

\* been set to the given status. Other Directions are ignored<br>

\* Cleanup: N/A<br>

\*

\* @param dir the button that is to be given a new status

\* @param status true if the light is to be lit, false otherwise

\* @param floor the floor on which the button is located

\*/

public void setCallButtonLit(Direction.DIRECTION dir, boolean status, int floor) {

if (dir == Direction.DIRECTION.UP) {

callUpIcon[floor].setIcon(status ? upOn : upOff);

} else if (dir == Direction.DIRECTION.DOWN) {

callDownIcon[floor].setIcon(status ? downOn : downOff);

}

}

/\*\*

\* Sets the target floor button to a given status<br>

\*

\* Precondition: The floor is valid<br>

\* Postcondition: The target button status has been set to the given

\* status<br>

\* Cleanup: N/A<br>

\*

\* @param status true if the light is to be lit, false otherwise

\* @param floor the floor tied to the target button

\*/

public void setTargetButtonLit(boolean status, int floor) {

targetFloorIcon[floor].setIcon(status ? lightOn : lightOff);

}

/\*\*

\* Sets the door to the given status<br>

\*

\* Precondition: The floor is valid<br>

\* Postcondition: The door status has been set to the given status<br>

\* Cleanup: N/A<br>

\*

\* @param status true if the door is to be opened, false otherwise

\* @param floor the floor tied to the door

\*/

public void setFloorDoorOpen(boolean status, int floor) {

floorDoorIcon[floor].setIcon(status ? doorOpen : doorClosed);

}

/\*\*

\* Sets the elevator floor to the given floor<br>

\*

\* Precondition: The floor is valid<br>

\* Postconditions: The elevator has been moved to the given floor and the

\* door has been set to the given status<br>

\* Cleanup: N/A<br>

\*

\* @param status true if the door is to be opened, false otherwise

\* @param floor the floor where the elevator should go

\*/

public void setElevatorDoorOpen(boolean status, int floor) {

Insets insets = getContentPane().getInsets();

Dimension size = callUpButtons[0].getPreferredSize();

elevatorDoorIcon.setBounds(ELEVATOR\_H\_OFFSET + insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - floor - 1) + insets.top), 50, 50);

elevatorDoorIcon.setIcon(status ? elevatorDoorOpen : elevatorDoorClosed);

}

/\*\*

\* Presses the up button on a given floor<br>

\*

\* Preconditions: The floor is valid<br>

\* Postconditions: The up button has been pressed<br>

\* Cleanup: N/A<br>

\*

\* @param evt the ActionEvent passed in from Swing

\* @param floor the floor where the call was made

\*

\* @see CallButton#callElevator(Direction.DIRECTION)

\*/

public void callUp(java.awt.event.ActionEvent evt, int floor) {

controller.callUp(floor);

}

/\*\*

\* Presses the down button on a given floor<br>

\*

\* Preconditions: The floor is valid<br>

\* Postconditions: The down button has been pressed<br>

\* Cleanup: N/A<br>

\*

\* @param evt the ActionEvent passed in from Swing

\* @param floor the floor where the call was made

\*

\* @see CallButton#callElevator(Direction.DIRECTION)

\*/

public void callDown(java.awt.event.ActionEvent evt, int floor) {

controller.callDown(floor);

}

/\*\*

\* Presses a floor button<br>

\*

\* Preconditions: The floor is valid<br>

\* Postconditions: The given button has been pressed<br>

\* Cleanup: N/A<br>

\*

\* @param evt the ActionEvent passed in from Swing

\* @param floor the floor button that was pressed

\*

\* @see TargetFloorButton#selectFloor()

\*/

public void selectFloor(java.awt.event.ActionEvent evt, int floor) {

controller.selectFloor(floor);

}

/\*\*

\* Creates the UIController<br>

\*

\* Preconditions: N/A<br>

\* Postconditions: The new UIController, its model, and its view have been

\* created<br>

\* Cleanup: N/A<br>

\*

\*/

public UIView() {

initializeUI();

controller = new UIController(numFloors, this);

}

private void initializeUI() {

Container pane = getContentPane();

pane.setLayout(null);

Insets insets = pane.getInsets();

Dimension size = null;

// Enable the close button to stop the program

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

// Create the buttons and doors

callUpButtons = new JButton[numFloors];

**callDownButtons = new JButton[numFloors];**

**targetFloorButtons = new JButton[numFloors];**

upOn = new ImageIcon(getClass().getResource("UpOn.jpeg"), "Up On");

upOff = new ImageIcon(getClass().getResource("UpOff.jpeg"), "Up Off");

downOn = new ImageIcon(getClass().getResource("DownOn.jpeg"), "Down On");

downOff = new ImageIcon(getClass().getResource("DownOff.jpeg"), "Down Off");

lightOn = new ImageIcon(getClass().getResource("LightOn.jpeg"), "Light On");

lightOff = new ImageIcon(getClass().getResource("LightOff.jpeg"), "Light Off");

doorClosed = new ImageIcon(getClass().getResource("DoorClosed.jpeg"), "Door Closed");

doorOpen = new ImageIcon(getClass().getResource("DoorOpen.jpeg"), "Door Open");

elevatorDoorClosed = new ImageIcon(getClass().getResource("DoorClosed.jpeg"), "Door Closed");

elevatorDoorOpen = new ImageIcon(getClass().getResource("DoorOpen.jpeg"), "Door Open");

callUpIcon = new JLabel[numFloors];

callDownIcon = new JLabel[numFloors];

targetFloorIcon = new JLabel[numFloors];

floorDoorIcon = new JLabel[numFloors];

elevatorDoorIcon = new JLabel();

for (int i = numFloors - 1; i >= 0; i--) {

// Up buttons

callUpButtons[i] = new JButton();

callUpButtons[i].setText("UP");

callUpButtons[i].setPreferredSize(new java.awt.Dimension(60, 25));

callUpButtons[i].addActionListener(new java.awt.event.ActionListener() {

int floor;

@Override

public void actionPerformed(java.awt.event.ActionEvent evt) {

callUp(evt, floor);

}

public java.awt.event.ActionListener init(int floorNum) {

floor = floorNum;

return this;

}

}.init(i));

pane.add(callUpButtons[i]);

size = callUpButtons[i].getPreferredSize();

callUpButtons[i].setBounds(CALL\_BUTTON\_H\_OFFSET + insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - i - 1) + insets.top), size.width, size.height);

// Up icons

callUpIcon[i] = new JLabel(upOff);

pane.add(callUpIcon[i]);

callUpIcon[i].setBounds(insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - i - 1) + insets.top), 25, 25);

// Door icons

floorDoorIcon[i] = new JLabel(doorClosed);

pane.add(floorDoorIcon[i]);

floorDoorIcon[i].setBounds(DOOR\_H\_OFFSET + insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - i - 1) + insets.top), 50, 50);

// Down buttons

**callDownButtons[i] = new JButton();**

**callDownButtons[i].setText("DWN");**

**callDownButtons[i].setPreferredSize(new java.awt.Dimension(60, 25));**

**callDownButtons[i].addActionListener(new java.awt.event.ActionListener() {**

**int floor;**

**@Override**

**public void actionPerformed(java.awt.event.ActionEvent evt) {**

**callDown(evt, floor);**

**}**

**public java.awt.event.ActionListener init(int floorNum) {**

**floor = floorNum;**

**return this;**

**}**

**}.init(i));**

**pane.add(callDownButtons[i]);**

**size = callDownButtons[i].getPreferredSize();**

**callDownButtons[i].setBounds(CALL\_BUTTON\_H\_OFFSET + insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - i - 1) +25), size.width, size.height);**

// Down icons

callDownIcon[i] = new JLabel(downOff);

pane.add(callDownIcon[i]);

callDownIcon[i].setBounds(insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - i - 1) + insets.top + size.height), 25, 25);

// Floor buttons

**targetFloorButtons[i] = new JButton();**

**targetFloorButtons[i].setText("" + i);**

**targetFloorButtons[i].setPreferredSize(new java.awt.Dimension(60, 25));**

**targetFloorButtons[i].addActionListener(new java.awt.event.ActionListener() {**

**int floor;**

**@Override**

**public void actionPerformed(java.awt.event.ActionEvent evt) {**

**selectFloor(evt, floor);**

**}**

**public java.awt.event.ActionListener init(int floorNum) {**

**floor = floorNum;**

**return this;**

**}**

**}.init(i));**

**pane.add(targetFloorButtons[i]);**

**size = targetFloorButtons[i].getPreferredSize();**

**targetFloorButtons[i].setBounds(TARGET\_BUTTON\_H\_OFFSET + 40 + insets.left - size.width, size.height \* (numFloors - i - 1) + insets.top, size.width, size.height);**

// Target lights

targetFloorIcon[i] = new JLabel(lightOff);

pane.add(targetFloorIcon[i]);

targetFloorIcon[i].setBounds(TARGET\_BUTTON\_H\_OFFSET + insets.left - size.width, size.height \* (numFloors - i - 1) + insets.top, size.width, size.height);

}

callUpButtons[numFloors - 1].setVisible(false);

callUpIcon[numFloors - 1].setVisible(false);

// TODO callDownButtons

**callDownButtons[0].setVisible(false);**

callDownIcon[0].setVisible(false);

// Elevator icon

elevatorDoorIcon = new JLabel(doorClosed);

pane.add(elevatorDoorIcon);

size = callUpButtons[0].getPreferredSize();

elevatorDoorIcon.setBounds(ELEVATOR\_H\_OFFSET + insets.left, (int) (size.height \* CALL\_BUTTON\_V\_OFFSET \* (numFloors - 1) + insets.top), 50, 50);

}

/\*\*

\* Starts the program<br>

\*

\* Preconditions: args[0] > 1<br>

\* Postconditions: The new UIView, its model, and its view have been

\* created<br>

\* Cleanup: N/A<br>

\*

\* @param args argument[0] contains the number of floors

\*

\*/

public static void main(String args[]) {

// Read in the number of floors from the command line

if (args.length != 1) {

return;

}

numFloors = new Integer(args[0]);

// Create and display the form

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

UIView ctl = new UIView();

ctl.setSize(PANE\_WIDTH, PANE\_HEIGHT);

ctl.setVisible(true);

}

});

}

}