Assignment 1 Guide

This guide is intended to help students to get started on the assignment. You may skip to the section that you need help with.

The development steps are:

1. Declare the classes
2. Declare the fields
3. Declare the constructors
4. Declare the toString() functions
5. Code each goal in the order that PLATE tests it.

Declare the classes and fields

You can copy these declarations from the assignment specification. Then submit to PLATE. It is very important that you fix any errors that PLATE tells you about before continuing on to the constructors.

Declare the constructors

Define the constructors in this order: Person, Lift, Building. After completing each constructor, submit your project to PLATE, and don't continue onto the later parts of the assignment until PLATE says that your constructors are correct.

Person

The FAQ makes a correction to the specification and tells you this about the Person constructor:

- The id field is initialised from a parameter
- The name field is initialised from a parameter
- The level field is initialised from a parameter
- The destination field is initialised to the same value as the level
- The aboard field is initialised to false

This means that your constructor has 5 lines to initialise 5 fields(*) but has only 3 parameters, because only 3 of the fields are initialised from parameters.

The pattern for initialising a field from a parameter is:

```
this.id = id;
```

(*) It is not strictly necessary to initialise the aboard field since a boolean field will be initialised to false by default. So the constructor could be done in 4 lines if you accept the default initialiser for the boolean field.

Lift

The specification tells you on page 9 that the Lift constructor initialises the number, bottom, top, and level fields from parameters. The direction field always starts at zero. This means your constructor has 5 lines to initialise 5 fields(*) but has only 4 parameters, because only 4 of the fields are initialised from parameters.

(*) It is not strictly necessary to initialise the direction field since an int field will be initialised to 0 by default.

Building

The specification tells you on page 10 that the mode field is initialised by reading the mode from the user. This means you need to create a readMode() function and you need to call it while initialising the field. You have plenty examples of this in the lecture notes, videos in the Study 4 and Study 5 modules, in the tutor demos, and in your lab exercises.

Your constructor **also** needs to initialise the list of lifts by creating 3 new `Lift` objects and adding them to the `lifts` list. You also have several examples of this, in video #2 of the Study 5 module, in the tutor demo for Lab 5, and in your Lab 5 exercise. For example, in the tutor demo, you will find code like this to create 2 new `Account` objects and add them to the `accounts` list:

```
accounts.add(new Account("Savings"));
accounts.add(new Account("Loan"));
```

The main point you need to understand here is how to create a new object and knowing how many parameters you need to pass in. In the tutor demo example above, the code `new Account("Savings")` creates a new Account passing in one parameter value “Savings”. The reason you need to supply one parameter value is that the Account constructor was defined like this:

```
public Account(String type) { ...
```

It’s important you understand that your Lift constructor will take 4 parameters, and so when you create a new Lift, you will need to pass in 4 parameter values, each separated by a comma.

toString functions

Define the toString() functions in this order: Person, and then Lift.

Person toString()

To code this toString() function, you need to understand that the Person's level field stores the person's current level within the building, and the destination field stores the level that the person wishes to go to. If the person's level and destination are the same, it means the person is happy to stay at their current level and is not going anywhere. You also need to understand that the aboard field stores whether or not the person is currently aboard a lift.

The person's toString() function should produce a string of one of these forms:

- Sam(1) on level 2 waiting to go to level 4
- Sam(1) on level 2 going to level 4
- Jane(2) on level 5

The " waiting to go to level L" ending is shown only if the person is waiting to go to a particular destination level. That is, their destination has to be different from their current level, and they must not already be aboard.

The " going to level L" ending is shown only if the person is aboard.

Since the first part of the string is always shown while the last part of the string is only sometimes shown, you need a way to build the string in parts, where the last part is conditional. You may find it helpful to use the technique recommended in Lab 4 which is to introduce a String variable where you add parts onto the string incrementally like this:

```
String s = "";
s += "Part A";
s += "Part B";
s += "Part C";
return s;
```

This will allow you to place an if statement around one part of the string so that it is only sometimes included.

Lift toString()

The solution to the Lab 4 Lift exercise has now been posted right next to the link you clicked on to open this document. It provides you with the basic idea for how to build the Lift string. However, the Assignment version of the toString() function will be slightly different from the Lab 4 Lift exercise version, so you will need to read Page 10 of the specification to see what is required for the assignment.

Goals

Implement each goal in the order in which PLATE tests it. Therefore, start with the menu goal. Submit to PLATE and see what goal PLATE wants to test next.

Easy goals

First you need to implement the following goals which are very similar to the week 5 tutor demo and your week 5 lab exercise:

- The menu (main method + use method)
- help
- add person
- show people
- remove person
- show lifts

Complex goals

This is where the assignment becomes a bit more challenging. The remaining goals involve calling and operating the lifts, and switching modes. Because of the complexity of this task, it is a good idea to break it down into **incremental goals**.

- Incremental Goal #1: Call and operate lifts assuming that a person always calls lift #1.
- Incremental Goal #2: Support the various modes.
- Incremental Goal #3: Consider all 3 lifts and select the most suitable lift for a call.

The plan for most goals will be spread across classes. For example, “call” requires the involvement of several classes.

- The Building needs to select a person from its list.
- The Building needs to select a suitable lift from its list.
- The Person needs to update its destination.
- The Lift needs to add the person to its queue.

You may find it helpful to fill in a location table for each goal to help you to spread each goal across the classes:

Classes:	Building	Lift	Person
Fields:	entrance bottom top lifts people mode	number bottom top level direction passengers queue	id name level destination abroad
Goals:			
Call a lift			
Find best lift			
Operate lifts			
...			

Define a method in each of these classes to hand a different aspect of the goal, and link them together. Remember that each aspect of the goal needs to be coded in the same class as the field(s) that it needs to access. You can refer to the Ass1Design.pdf document for a sample class diagram which may give you some ideas on what methods to put in what classes. Note: This class diagram is intentionally incomplete so that you are forced to think a bit about how to structure your own code. The class diagram does include enough procedures, but you will want to add many more functions. The more functions you create, the easier it will be to implement your procedures. Don't do all the work in your procedures! (For more information on this principle, re-watch the last video of the Study 3 module)

Incremental Goal #1 - Assume a person always calls lift #1

PLATE's test cases are designed in such a way that initially, the first lift is always selected as the best lift. This means that in the beginning, you do not need a suitability function to determine the best lift.

Calling a lift

The "call" menu option has the following sample I/O:

```
Choice (a/r/p/c/l/o/x): c
Person ID: 2
Destination level: 6
```

So you need to read the Person ID and the destination level from the user. But you also need to lookup that person, find a suitable lift(*), update the person's destination, and add

that person to the lift's queue. As mentioned above, the plan should be spread across the various classes so that code to update a particular field goes into the same class as that field.

(*) Because we will always select lift #1 in the first incremental goal, you can use code such as the following:

```
private void call() {
      ...
      Lift bestLift = lifts.getFirst(); // simple!
      ...
}
```

Operating a lift

The algorithm for operating a lift is given in full on Page 4 of the assignment specification. Here is a summary of the steps:

- If the lift is stationary, set the direction of the lift toward the next pickup or dropoff point
- Let people alight and board
- If the lift is now empty, set the lift's direction to 0
- Otherwise, move the lift and its passengers one level in the current direction

You must apply the "push it right" design principle so that these steps are implemented inside the Lift class. You must also push some of the code into the Person class to make the person's current level change along with the lift's current level.

The key is to create lots of helper functions (as shown in week 3: break it down, build it up).

To help you understand what to get working first, here is a description of the 3 test cases PLATE uses to test the operate menu item:

- Test operate1: A single person calls a lift from level 2 and rides in the UP direction.
- Test operate2: Two people call the same lift from level 2 and ride in the UP direction, but alight at different levels
- Test operate3: Many people call the same lift, and sometimes more than one person will get alight at the same level. Sometimes the call is to travel in the UP direction, sometimes the call is to travel in the DOWN direction.

The order of these tests means that you can focus on getting your lift working with a single passenger travelling in the UP direction first, before getting the other scenarios working.

Incremental Goal #2 - Modes

Apply patterns from week 1 and week 5.

Incremental Goal #3 - Multiple lifts

In the last set of test cases on PLATE, your program will be tested in scenarios where Lift 1 is not always the most suitable lift. You will therefore need to define a function to calculate the suitability of a particular lift and also a function to select which of the 3 lifts has the highest suitability.

For the suitability function, PLATE will look for a function in your Lift class called suitability that takes 3 parameters as follows:

```
int suitability(int buildingDistance, int fromLevel, int toLevel)
```

This method should public and should return the lift's suitability based on these 3 parameters:

1. buildingDistance is the highest minus the lowest level of the building
2. fromLevel is what level the call originated from
3. toLevel is the desired destination of the caller

You are free to name these parameters what you like, but they must be listed in that order to be recognised by PLATE.