



The Elevator Challenge

The Fun Educational Resource for Object-Oriented Design and Programming

Introduction

The goal of the Elevator Challenge is to provide students and hobbyists a chance to practice their programming skills by addressing a real-life problem – how to most efficiently control the elevators or lifts of a building. At the heart of the challenge is an elevator simulator which allows programmers to design, code and then test their implementation in efficiently managing elevators in different real-world scenarios. You also get the chance to test your skills by comparing results with others, including the option to submit results here to compete against other programmers from around the world.

The Elevator Challenge has been accepted as an [ACM SIGSCE educational resource](#).

[Subscribe](#) now to receive news and updates.

Contents

This guide comprises three sections:

1. The Simulator. This section describes operation of the Elevator Challenge Simulator.
2. The Model. This section describes the underlying model of the Simulator.
3. The Code. This section gives an overview of the Simulator code and the development of the elevator group Controller.

Quick Peek

To get a quick feel for what the Simulator looks like, simply install the Simulator and run the file *challenge.bat*. As long as you have Java installed, it will give you a chance to play with the functionality before creating your own control logic.

The Simulator

The Elevator Challenge is run using a Simulator which allows for the creation of a *Scenario* which describes a particular building and the characteristics of its occupants. Using the logic provided by in the Challenge class, the performance of the elevator group is measured in a *Simulation*. Once the logic of the elevator group *Controller* is programmed by the user, the Elevator Challenge Simulator does all the work!

Running the Simulator

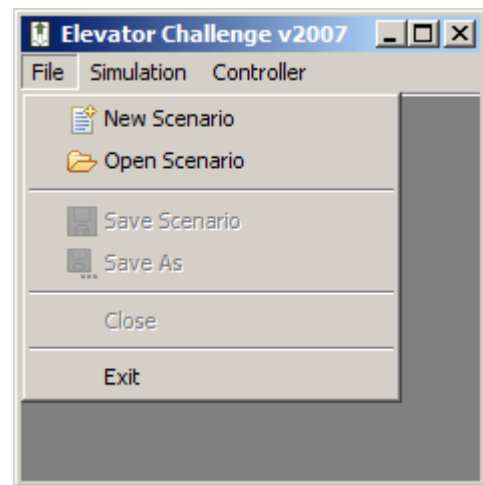
The Simulator is run either through a development environment, like Eclipse, or through the command line. Programming the Controller and running the simulator is described later in “The Code.” There are a few pre-requisites, specifically Windows and Java 1.6.

Scenario Creation

A Scenario is created by entering the characteristics of the building, elevator and occupants. New Scenarios can be created by selecting *New Scenario* from the *File* menu of the Simulator. Once created, Scenarios can be saved and opened from the *File* menu.

Once the Scenario window is opened, the necessary parameters can be entered. Note that initially, the New Scenario is created with default values that are then updated. At any point, the Scenario can be saved by accessing the *Save Scenario* command from the *File* Menu.

To provide greater flexibility, changes entered into any of the fields of the Scenario are not applied until specifically chosen by the user as detailed below.



Scenario Toolbar

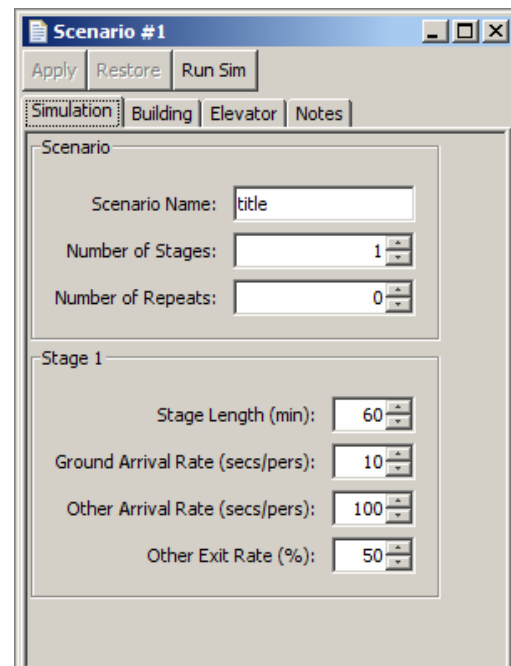
Each Scenario provides a toolbar for general commands:

- *Apply*. This command applies changes in the various entry fields to the Scenario. Before running the simulation or saving a scenario, you will be prompted to accept or reject and changes that have yet to be applied.
- *Restore*. This command restores the Scenario fields to values since the Scenario was loaded or since the last *Apply*.
- *Run Sim*. This command executes the Simulation engine as described later.

The Scenario window provides four parameter entry panes as described below.

Scenario Simulation Pane

The *Simulation* Pane describes overall parameters of the



Elevator Challenge Guide

simulation itself, including:

Parameter	Description
Scenario Name	The Name of the Scenario. This is not necessarily the file name.
Stages	The number of stages in the simulation. For each stage, there are specific passenger behaviors including arrival rates and exit floors. Multiple stages can be defined to reflect different times of day when passenger behavior varies.
Stage Times	The length of each stage in minutes. The total of these times represents the length of the entire simulation run.
Repeat	The number of times the full run of stages are repeated. A zero value indicates only running through all stages once. Values of one or more denote the number of additional times the simulation will be run.

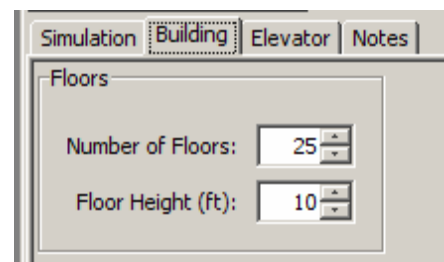
For each Stage, specific passenger behaviors are defined, as follows:

Parameter	Description
Ground Floor Arrival Rate	The period at which passengers arrive on the ground floor, measured in seconds. For example, a period of 2 seconds means that a passenger arrives on the ground floor every two seconds.
Other Floor Arrival Rate	The period at which passengers arrive on all other floors, measured in seconds.
Other Floor Exit Rate	The rate at which passengers on other floors leave the building versus traveling to other floors. The remaining passengers not going to the ground floor will travel to floors based on their relative position in the building. For example, the closer the floor is to the top of the building, the more likely the passenger is to be going to a floor below.

Scenario Building Pane

The *Building Pane* describes overall parameters of the building itself, including:

Parameter	Description
Floors	The number of floors in the building. All buildings have a ground floor including in this count.
Floor Height	The height of each floor in feet. It is assumed that all floors have uniform height.

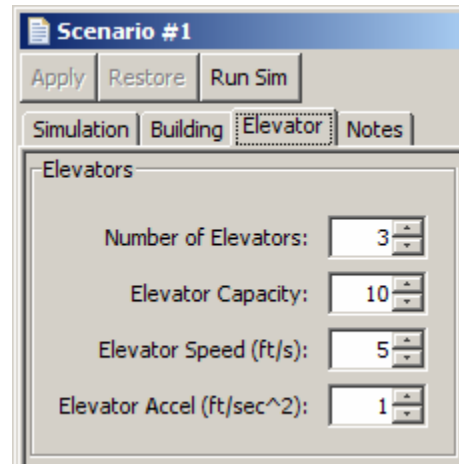


Elevator Challenge Guide

Scenario Elevator Pane

The *Elevator* Pane describes overall parameters of the elevator group, including:

Parameter	Description
Number of Elevators	The number of elevators in the building.
Elevator Capacity	The maximum number of passengers each elevator can hold.
Elevator Speed	The speed of the elevators in feet per second.
Elevator Acceleration	The acceleration of the elevators in feet per second ² .



Scenario Notes Pane

The *Notes* Pane provides a simple text document which can be used to record any notes or comments required.

Saving and Opening Scenarios

Scenarios can be open, saved and closed like documents in a word processing or spreadsheet application. The Simulator uses *xml* format to save and open Scenario files so the default file extension is *xml*.

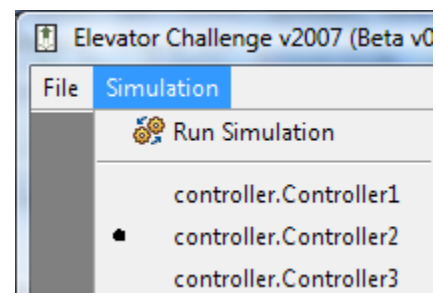
Running the Simulation

Once a Scenario has been created, the Simulation can be run. Multiple Simulations can be run from a single scenario but any changes made to a Scenario's parameters will not be reflected in Simulations that have been initiated. The Simulation itself is opened in a separate window as shown.

Controller Selection

When a Simulation is run, it uses the defined Scenario as well as one of the available Controllers. Controllers are automatically loaded with the Simulator is loaded. As described later in the programming section, each Controller is programmed as a Controller object according to require specifications. With the Simulator, two sample Controllers have been provided and more may be added as they are created.

The Controller selection is made via the *Simulation* menu and the selecting the name of the Controller as defined by the Java class name. That Controller is active for all Simulations that are run until such a time as a different Controller is selected.



Simulation Controls

The execution of the simulation is managed by three controls:

- *Play/Pause/Resume*. This first button controls execution of the Simulation. *Play* is used to start the Simulation when at the beginning of the simulation clock. *Pause* is used to

Elevator Challenge Guide

temporarily stop Simulation execution. *Resume* triggers the continuation of the Simulation from a *Pause*.

- *Stop*. This button stops the Simulation without the ability to resume. This is used when you want to restart the Simulation from the start by pressing *Stop* followed by the *Play* button described above.
- *Speed*. This control varies the speed of the Simulation. Moving the slider to the left slows the simulation. Moving the slider to the right speeds it with the right most setting resulting in the fastest execution of the Simulation.

Simulation Time Displays

At the bottom of the Simulation window, three time displays are provided:

- *Clock*. This display provides the overall time of the Simulation including all stages and all repeats. Its shows total hours, minutes and seconds.
- *Repeat*. This display shows the Simulation's current Repeat and the total time through that Repeat. Each repeat represents one run through all of the Simulation Stages.
- *Stage*. This display shows the Simulation's current Stage and the time through that Stage.

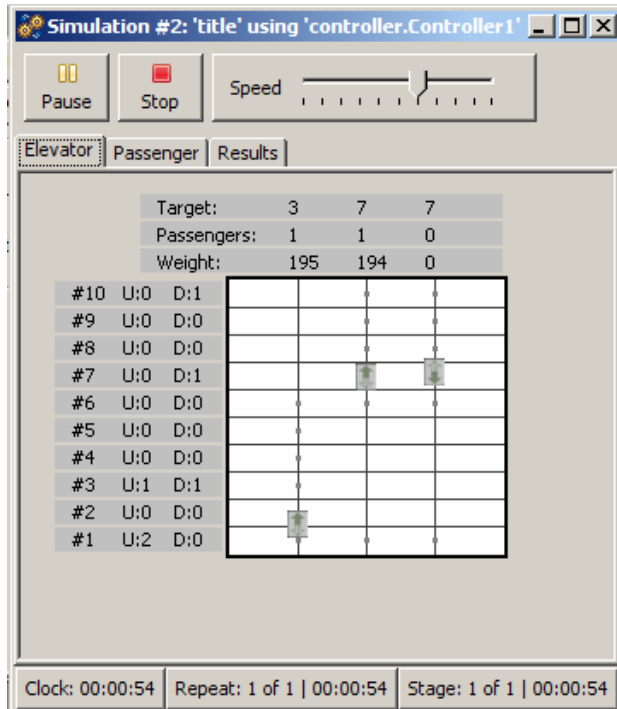
For version 0.2 on, these have been updated to show tenths of seconds.

Simulation Elevator Pane

The *Elevator* Pane provides a view of elevator and floor status as the Simulation runs.

For the Elevator, the following information is provided:

- *Position*. The current elevator position is indicated by its vertical position in the graphical representation of the building. Each floor is labeled with the floor # and increases further up the building.
- *Floors Serviced*. The Simulator can control whether or not an elevator services a particular floor. Floors that are serviced by a given elevator are so indicated by the presence of a small grey rectangle on the subject floor.
- *Committed Direction*. Each elevator has a committed direction (the direction in which it will take passengers). This is indicated by a green arrow on the elevator itself.
- *Target Floor*. The floor to which elevator has been dispatched is shown above the building, preceded by the label "Target:".
- *Passenger Count*. The number of passengers riding on each elevator is shown above the building, preceded by the label "Passengers:".
- *Passenger Weight*. The total weight of the passengers riding on each elevator is shown above the building, preceded by the label "Weight:".



Elevator Challenge Guide

In addition, the number of passengers waiting on each floor is shown in the rows immediately to the left of the building. The number passengers weighting to go up and down are shown with the labels “U:” and “D:”, respectively.

Simulation Passenger Pane

The *Passenger* Pane provides details on all passengers current traveling or waiting to travel on an elevator. Information presented in this pane includes:

- *ID*. Every passenger created has a unique, sequential identification number.
- *Depart*. The floor from which the passenger will board the elevator.
- *Target*. The destination floor from which they will depart the elevator. If the Controller has implemented split floors, the destination floor will be shown along with the cross-over floor selected by the passenger in parentheses. For example, if the passenger is going to floor #5 via floor #6, it will be shown as 10 (6).
- *Created*. Simulation clock time, in seconds, when the passenger was created.
- *Waiting*. The amount of time, in seconds, the passenger has waited for an elevator to arrive.
- *Riding*. The total amount of time, in seconds, the passenger has been riding their current elevator.
- *On*. The elevator number the user is riding on.

ID	Depart	Target	Waiting	Riding	On
7	3	2	45	0	
8	3	6	22	16	1
9	7	5 (6)	8	21	
10	1	8	27	0	3
11	10	1	27	0	
12	1	5	25	0	
13	1	8	22	0	
14	4	6	15	7	1
15	2	3	9	0	
16	2	5	6	0	
17	1	5	3	0	

At the end of the simulation, all remaining passengers will be processed and removed from this table.

Simulation Results Pane

The *Results* Pane provides a summary of the simulation and the performance statistics based on passengers that have completed their journey.

The Summary info includes the title of the Scenario as well as Controller information.

Scenario results information includes:

- *Passengers processed*. The total number of passengers who have completed their travel.
- *Average wait time*. For all passengers processed, the average time spent waiting for an elevator to arrive.
- *Maximum wait time*. The highest wait time of any processed passenger.
- *Average total time*. For all passengers processed, the average total time spent traveling from arriving at the departure floor to exiting the elevator on their destination floor.

The screenshot shows the 'Results' pane with the following content:

- Summary**
 - Scenario title: title
 - Run with Controller: controller.Controller1
 - Controller Description: Example Controller with Splits
- Scenario**
 - Passengers processed: 0
 - Average wait time: 0
 - Maximum wait time: 0
 - Average total time: 0
 - Maximum total time: 0
- At the bottom, there are three status boxes: Clock: 00:00:00, Repeat: 1 of 1 | 00:00:00, and Stage: 1 of 1 | 00:00:00.

Elevator Challenge Guide

- *Maximum total time.* The highest total travel time of any processed passenger.

At the end of the simulation, all remaining passengers will be processed using the greater of average versus actual wait and total travel times. These results will be used to determine the rating of the Controller implementation.

The Model

This section describes technical details of the Simulation model itself. The model is based on the following real-world objects:

- Building. This is the building in which the elevators run and passengers arrive, travel and exit.
- Elevators. These are the elevators that carry passengers within the building.
- Controller. This is the logic that, based on inputs from the building and elevators, tells the elevators where to go.
- Passengers. These are passengers that arrive, travel and exit on the various building floors.

Each of these interacts based on some behavioral assumptions as well as Scenario-specified parameters.

The Building

The building is a relatively simple object that has floors and elevators. As such, it is described as by the following scenario parameters:

- Number of floors. This is the numbers of floors in the building, including the ground floor. It is assumed there are no floors below ground level. The number of floors substantially impacts the passenger load that elevators are subject to.
- Floor height. This is the height of the floors in feet. It is assumed that each floor is the same height. The higher the floor, the more time it takes elevators to travel – effectively reducing their productivity.
- Number of elevators. This is the number of elevators in the building. All elevators service the ground floor but may, as described below, not necessarily service all other floors. The more elevators, the greater the service to passengers.

Each floor has an up and a down button for the purposes of passengers calling the elevator. The model automatically activates and deactivates the appropriate button upon passenger arrival and boarding. Activation occurs when any passenger is waiting for an elevator in the specific direction. Deactivation occurs when all passengers traveling in a particular direction have boarded an elevator.

The three scenario parameters are available to the elevator Controller along with the status of floor up and down buttons. In the code itself, floor numbering starts at 0 for floor 1. The Passenger

Passengers represent the individuals who travel on the elevators. Their behavior is governed by the following scenario parameters:

- Ground floor arrival rate. This is how often a new passenger arrives at the ground floor (ie. every x seconds). Whether a passenger arrives at a particular moment is governed by a random number generation reflecting the arrival rate. Thus there will be variances in particular periods but over time, the arrival rate will be achieved.
- Other floor arrival rate. Similar to ground floor arrival rate, this dictates the passenger arrival rate for each of the other floors. Thus, even a low rate here can generate substantial traffic. It is assumed to be uniform across all *other* floors.
- Other floor exit rate. For passengers arriving at other floors (non-ground floor), this is the portion that will travel to the ground floor and exit the building. The remainder will pick a random floor not including the ground floor (and not the departing floor).

Elevator Challenge Guide

- Passenger weight. When created, each passenger has a weight that is randomly distributed from 100 to 200 pounds. This is not directly accessible but total passenger weight on a given elevator can be determined as described later on.

Some elevator behavior is inherent in the model and cannot be controlled:

- Selecting departing floors. Based on the rates defined for the simulation passengers will be created. Amongst the *other* floors, passengers will be randomly assigned departing floors, not including the ground floor.
- Selecting destination floors. Passengers arriving on the ground floor will randomly select a floor to travel to while *other* floor passengers will likewise have their destination randomly chosen as described above.
- Floor buttons. Upon creation, passengers will automatically select the appropriate *up* or *down* button on their floor.
- Boarding the elevator. Passengers will automatically board an elevator that arrives as long as it is not full and it services their target floor (and the doors are open). Passengers are orderly and will board one at a time, taking half a second each.
- Selecting the target floor. In the elevator, passengers automatically press their desired target floor.
- Exiting the elevator. Passengers will automatically exit an elevator that arrives at their target floor. They are again orderly and will exit one at a time at half a second per passenger. Passengers will not board and exit at the same time so the total time taken per elevator stop includes one half second for each passenger departing as well as each boarding.

As each passenger is created, boards an elevator and exits, the time is recorded including their total wait time (time between creation and boarding) as well as total travel time (time between creation and exiting). This is used to calculate the results. Any passengers who are still waiting or riding at the end of the simulation run will have their times set as follows:

- Wait time: the greater of their wait time so far or average wait time for all passengers to that point.
- Travel time: the greater of travel time so far or average total time for all passengers to that point.

The Elevator

The elevator is the workhorse of the simulation and is the only object controlled by content submissions. Its behavior is impacted by the following simulation parameters:

- Elevator capacity. This dictates the maximum number of passengers that can fit on an elevator. An elevator at capacity will still respond to all stops but passengers will not enter a full elevator (unless someone departs first). The higher the capacity, the greater passenger demand that can be serviced.
- Elevator speed. This dictates the maximum speed an elevator can travel at in feet/sec. It is assumed to be the same for all elevators. The faster the elevator, the better service it can provide.
- Elevator acceleration. This dictates the rate at which the elevator can increase or decrease speed in ft/sec^2 . It is assumed to be the same for all elevators. The higher the acceleration, the faster the elevator can accelerate and decelerate, allowing its average speed to be faster.

Some elevator behavior is inherent in the model and cannot be controlled:

Elevator Challenge Guide

- Door opening & closing. Elevator doors automatically open when arriving at a floor and close before departing. When passengers are waiting to board an elevator, the doors will automatically stay open until all passengers are boarded or elevator capacity is reached. Each open and close action takes 1 second.
- Elevator buttons. Arriving passengers automatically press the appropriate elevator button upon boarding the elevator and the light is automatically turned off when the elevator arrives at the associated floor.
- Elevator speed and acceleration. The elevator will automatically accelerate to the maximum speed possible until there is only enough time left to slow down to arrive at the target floor.

There is substantial elevator status information available to the Controller:

- Elevator speed. The current speed of the elevator. Positive speed indicates an elevator heading up while negative indicates an elevator going down.
- Elevator acceleration. The acceleration of the elevators.
- Elevator button status. For each elevator, which floor buttons on the elevator have been pressed.
- Elevator position. The current position of the elevator, both in feet and to the closest floor. Position is measured in feet above ground floor with zero being the bottom.
- Elevator door status. Whether the doors for an elevator are open or closed. The granularity of the model clock is such that status will never be a transition between open or closed.
- Elevator target. The current floor target of the elevator as instructed by the controller. The elevator will travel to that target and stop until directed to the next target.
- Elevator committed direction. The current committed direction of the elevator. Elevators responding to a passenger floor button must have a committed direction, *up* or *down*. Otherwise, passengers wishing to go up would board the same elevators as passengers wishing to go down. Accordingly, each elevator can have an *up*, *down* or *uncommitted* committed direction set by the controller managed in response to passenger travel.
- Elevator floor service. Whether a particular elevator services a particular floor. In some scenarios, elevators are allowed to only service certain floors. Set by the controller, this can help to achieve greater passenger service. Every elevator must service the ground floor.
- Elevator weight. This provides the current weight of the elevator less the weight of the empty elevator – hence the weight of the passengers on board. This can be useful for detecting when the elevator is getting full.

Content entries control the Elevators by setting the following parameters:

- Elevator target. Sets the floor target of a particular elevator. Typically, this target is set in response to a given passenger selecting a floor up or down button, or selecting a floor button in an elevator. The elevator will travel to that target and stop until directed to the next target.
- Elevator committed direction. Sets the current committed direction of the elevator. This is set so that passengers board elevators going in their desired direction. This is indicated by the controller by whether the *up* or *down* floor button has been pressed on a given floor. This is an important concept and it should be recognized that in some cases, an elevator may be going down, for example, but its committed direction is up – reflecting the fact it has been dispatched down to pick up a passenger who is traveling up. Whenever an elevator is being dispatched to a target floor, it should have a committed direction set.

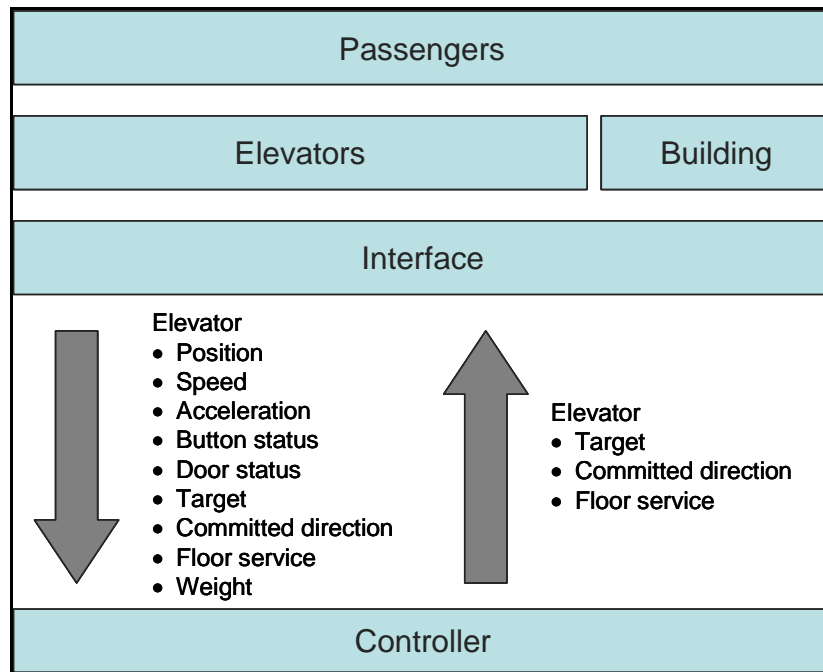
Elevator Challenge Guide

- Elevator floor service. Sets whether a particular elevator services a particular floor if allowed in a particular scenario. In larger buildings, it is typically better to set elevator service to a particular subset of floors. Note that the ground floor will always get service and that in most scenarios, there is a penalty in terms of increased ground arrival rate to reflect passengers using the ground floor to move between serviced floor ranges.

In the code itself, elevator numbering starts at zero for elevator 1.

The Controller

The Controller is the brains behind elevator travel. It accesses status information from an interface and sets the three parameters described above for each elevator. It is summarized in the following diagram.



The controller will be called in three ways – once for any initial setup required before the passengers start to arrive, and then once every clock tick (each 0.1 second). For each of these cycles updated information on the elevator and floor buttons can be read. As described previously, elevators are controlled by three variables for each elevator: target, committed direction and floor service. Finally, a single call also allows the Controller to provide description text.

It's that simple!

The Code

The Elevator Challenge package is composed of a single .zip file with the following directories and most significant files:

Directory	File	Description
controller/	Controller1.java Controller2.java Controller3.java	<p>Sample source code for three Controller classes. Each of the three have different features:</p> <p>Controller 1 uses split floors (limiting which floors are serviced by which elevator).</p> <p>Controller 2 does not use split floors.</p> <p>Controller 3 monitors elevator weight and guesses when an elevator is full and can't pick up any more passengers.</p> <p>These files can be modified and compiled to implement your own controller logic.</p>
	Controller1.class Controller2.class Controller3.class	Compiled controller classes. These classes (and any others in this directory) are automatically loaded when the Simulator is executed.
docs/	Elevator Challenge 2007 Guide.pdf	This guide.
	index.html	Javadocs for interface used by the Controller to set and retrieve simulation parameters.
scenarios/	simple.xml	Scenario file with a simple building scenario.
/	Elevator.jar	This is the jar file that contains all the necessary classes for the Simulator.
/	challenge.bat	This file provides a quick way to run the Simulator to get a feel for what it looks like without having to build that application.

Programming the Controller Class

As detailed previously, The Controller must implement three methods:

Elevator Challenge Guide

- **public void** `setup(Interface controllerInterface)`. This method is called when the Controller is first set up and passes the Interface through which the Controller will receive and set information for elevator group control.
- **public void** `tick(long clock)`. This method is called at each clock tick so that the Controller can receive new elevator group status information and make any updates required.
- **public** `String getText()`. This method provides the Simulator with a brief description of the controller for display.

Controller1.java, Controller2.java and Controller3.java are provided as programming examples. As shown in each, Controller classes must implement the IController Interface which enforces the presence of the three methods cited above. The two examples are identical apart from Controller1 uses split floors and Controller2 does not. Refer to the Interface class Javadocs and the example implementation for detailed information on implementing a Controller class.

Multiple Controller Classes can be created as long as they have unique names and are located in the same directory. All such classes are loaded when the Simulator is run and are available via the *Controller* menu.

Compiling and Running

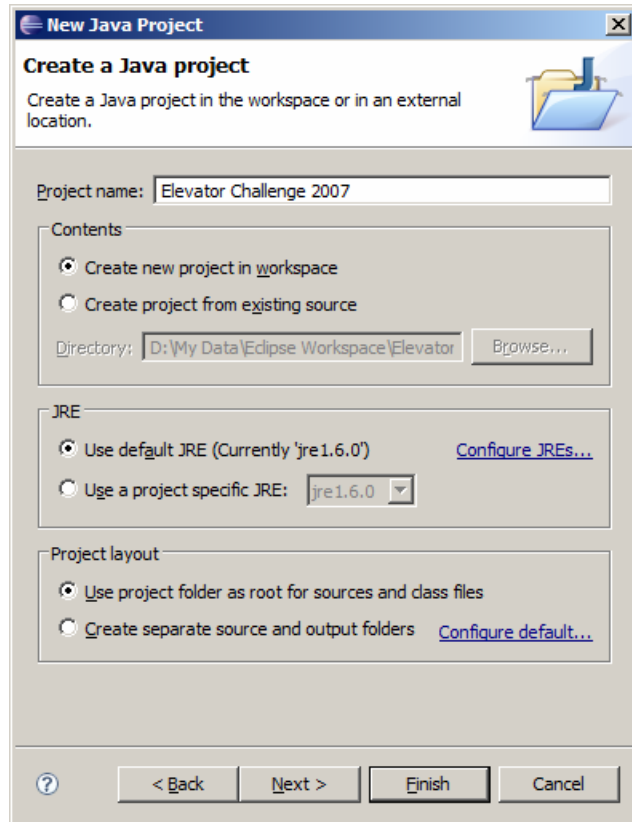
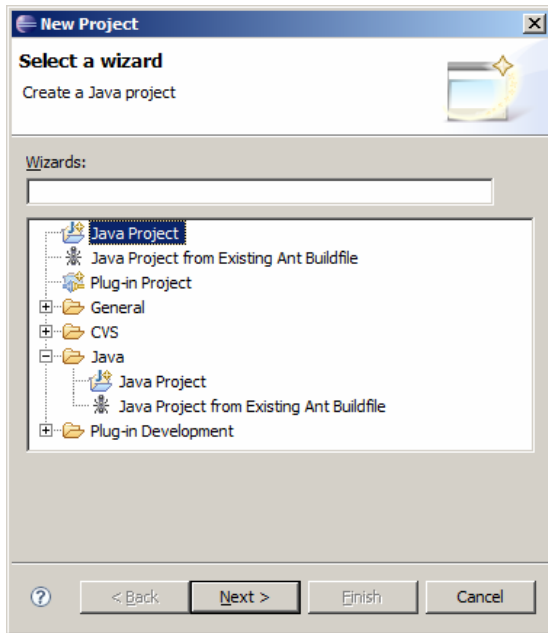
The Elevator Challenge package contains all the data to necessary to compile either using command line Java tools or with [Eclipse](#).

The first step is to unzip the package to a particular path (ex. `c:\elevator`). Make sure to use the folder names from the zip file. Once that has been done, follow the instructions depending on your programming environment.

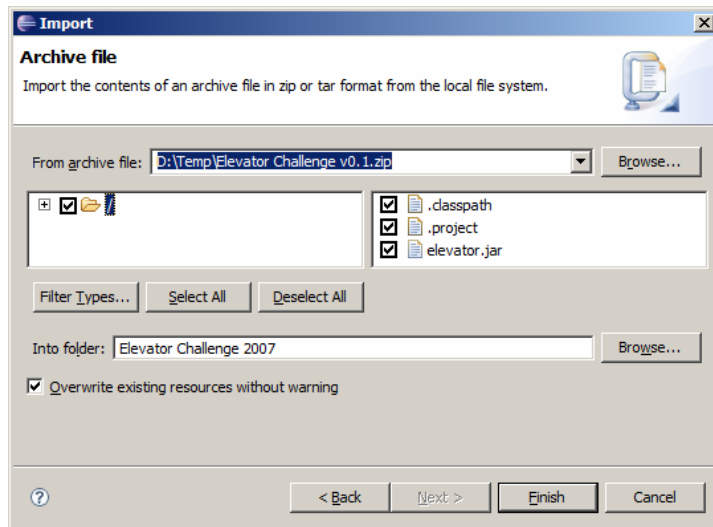
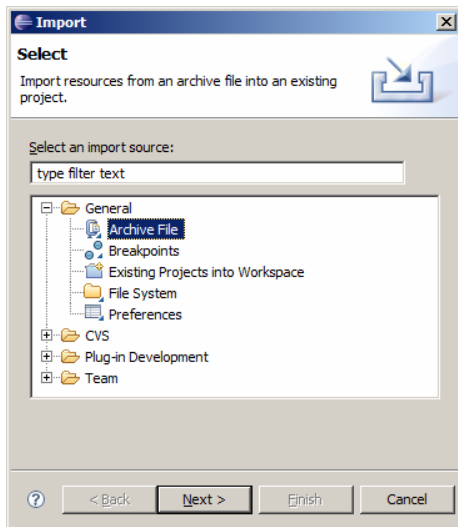
Using Eclipse

The first step with Eclipse is to create a new Java Project in which the Controller will be developed and the Simulator run. As shown below, this is done by selecting a new “Java Project” from the Eclipse “New” menu. It is very important when creating the project to select the “Use Project Folder as Root...” option under “Project layout” as shown.

Elevator Challenge Guide

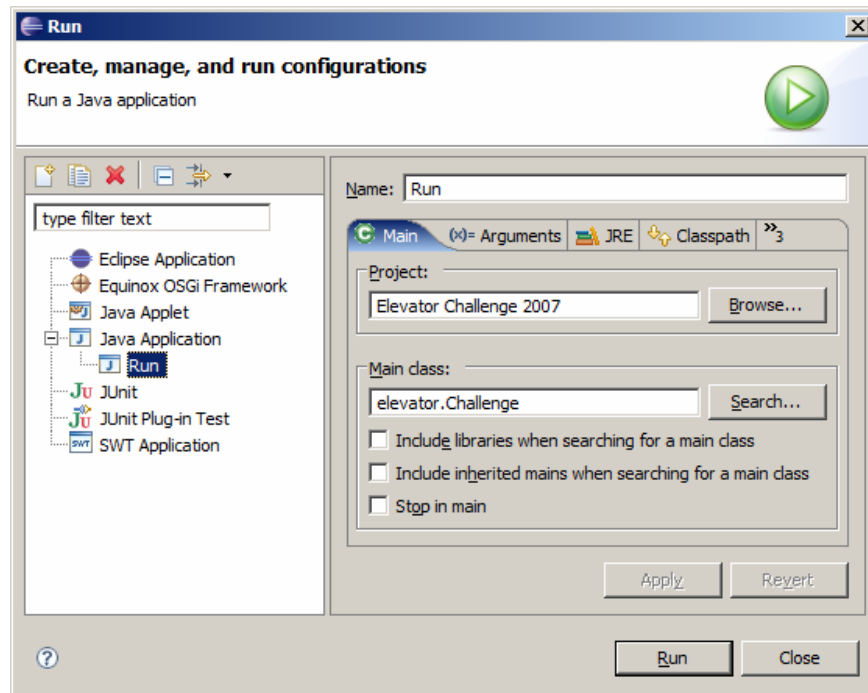


Next, importing the Simulator code into the project is done via the “Import” command under Eclipse’s “File” menu. Select the import “Archive File” option and then simply navigate to the downloaded Simulator zip file.



The Simulator should now be ready to run. To do so, a run configuration will be required. Simply navigate to the “Run...” command under the “Run” menu and select the “Java Application” option. You will now need to define the name of the configuration (arbitrary) and type in the following as the Main class: “elevator.Challenge”.

Elevator Challenge Guide



The Simulator is now ready to run!

Using Command Line

The first step is to unzip the package to a particular path (ex. `c:\elevator`). Make sure to use the folder names from the zip file.

Once this is done, then the commands for compiling and running would be as per normal Java compilation and execution. For example, with a path of `c:\elevator` and a Controller with source `Controller1.java` the command lines would be:

- `Javac -cp "c:\elevator;c:\elevator\elevator.jar" controller\Controller1.java`
- `Java -cp "c:\elevator;c:\elevator\elevator.jar" elevator.Challenge`

Remember that all Controllers must be built inside the "controller directory". Also, by default, the `elevator.jar` file is in the root directory of where the zip file is extracted.

More Information

See www.elevatorchallenge.com for the latest in news about the Elevator Challenge as well as forums to share ideas and submit issues. [Subscribe](#) now to receive news and updates.

Change History

Changes to the Elevator Challenge Simulator are as follows:

Version	Changes
v0.1 1 May 2007	Initial Release. This release shows the new interface for the simulator.
v0.2 18 Jun 2007	Update to a 10 msec clock and floating point motion calculation for greater accuracy. Includes updates to passenger load/unload times (0.5 sec) and door open/close times (1.0 sec). Minor updates to sample controllers to ensure compatibility with new model. Changes cited in source code.
v0.3 29 Jun 2007	Performance improvements of about 400% in running simulations. Most gain done by managing update rates of graphics and results. New example controller taking advantage of availability of weight data. If the elevator is full – no point in stopping to try and pick up new passengers. Improved look and feel for Windows Vista.

GOOD LUCK!