Program :

Normal Condition

(1) Event implements Runnable, each type of event provides its own timing. Each event file should be a class of its own.

Input:

No subclasses should be present in GreenhouseControls. Event and all its subclasses are in their own class files. Since Event implements Runnable, therefore, all child classes also implement Runnable. Implementing Runnable allows these classes to be converted into threads. In Controller, an ExecutorService manages the implicit creation and execution of these Event threads. Event timing is maintained on event creation and addition to the eventList List.

Run any of the examples.
Ex: java GreenhouseControls -f examples1.txt

Expected output: Normal program execution
Restarting system test
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Bing!
Thermostat on day setting
Terminating

(2) Provide the means to create the event classes from their names. Use the same capability to be able to add new Event classes and modify Event classes without recompiling GreenhouseControls.java. Start the Event thread right after you have successfully created the Event object.

Input:

Uncomment the tester code in the main() method. Should be identifiable with comments for “Test createEvent for examples1.txt”

Ex: java GreenhouseControls -f examples1.txt

Do not forget to comment this code back in after running this test.

Expected output:
Restarting system test
Thermostat on night setting
Light is on
Light is off
Light is on
Greenhouse water is on
Terminating

(3) Events should be started once they are created. Devise a mechanism for GreenhouseControls to suspend and resume the Event threads. To do this, GreenhouseControls needs to keep a collection of threads of all events.

Input:
In Controller, Events are converted to threads using ExecutorService, which manages them implicitly and keeps a pool (collection) of the threads.

Uncomment the tester code in the main() method. Should be identifiable with comments for “Test suspend and resume”

Run any of the examples.
Ex: java GreenhouseControls -f examples1.txt

Do not forget to comment this code back in after running this test.

Expected output:

Program should output a 5 second pause between suspending and resuming. Resuming starts off at last event with no changes to event timing.

Restarting system test
Suspending
Resuming
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Bing!
Thermostat on day setting
Terminating

(4) Remove all existing state variables in GreenhouseControls and replace them using a collection of TwoTuple. Each time an event runs it should add an entry that identifies the variable the event is modifying as key and another object to this structure as the value the event is setting. Create a method in GreenhouseControls called setVariable to handle updating to this collection. Use the synchronization feature in Java to ensure that two Event classes are not trying to add to the structure at the same time.

Input:
No variables for events and their states present in GreenhouseControls. A new TwoTuple class manages the event:state pairs. The constructor for GreenhouseControls adds the TwoTuples of event:state pairs to a stateVariables List. Each Event subclass sets their respective variables into their wanted states in their action() methods.

Run any of the examples.
Ex: java GreenhouseControls -f examples1.txt

Expected output: Normal program execution

Restarting system test
Thermostat on night setting
Light is on
Light is off
Greenhouse water is on
Greenhouse water is off
Bing!
Thermostat on day setting
Terminating