

# GreenhouseControls

## **Default Constructor: public GreenhouseControls()**

Input: `GreenhouseControls gc = new GreenhouseControls();`

(This gc object will be used throughout the test plan)

Expected Output: New GreenhouseControls object, gc, is created

## **Set the greenhouse state: void setVariable(String key, Object value)**

Input: `gc.setVariable("Light", true)`

Expected Output: It set the HashMap attribute, "variables", with key of "Light" and the value of true

## **Start the Event: void addEvent(Event c)**

Input: `gc.addEvent(new LightOn(0, gc));`

Expected Output: It adds the LightOn event to the eventlist and starts thread with the LightOn object, that will call run() method of the Event. The event will be executed right away as the delay time is 0, prints out "Light is on"

## **Start the Malfunction Event: void addEvent(Event c)**

Input: `gc.addEvent(new WindowMalfunction(0, gc));`

Expected Output: It adds the LightOn event to the eventlist and starts thread with the LightOn object, that will call run() method of the Event. The event will be executed right away as the delay time is 0, prints out "Light is on" but the WindowMalfunction event throws ControllerException when its actioned, so the shutdown(long delaytime) will be called, the Greenhouse will be serialized then program will be finished

## **Add Event to the controller: void addEvent(String eventName, long delayTime, Controller controller)**

Input: `gc.addEvent("LightOn", 0, gc);`

Expected Output: It creates the LightOn event with the delay time of 0 to the gc object's. Then call addEvent(Event c) method to run the thread with the event

## **Add Ring Event to the controller: void addEvent(String eventName, long delayTime, int ring, Controller controller)**

Input: `gc.addEvent("Ring", 0, 3, gc);`

Expected Output: It creates Ring event with the delay time of 0 and 3 rings to the gc object's eventList list attribute. Then call `addEvent(Event c)` method to run the thread with the event

**Add Other Event to the controller using the one with the ring parameter: `void addEvent(String eventName, long delayTime, int ring, Controller controller)`**

Input: `gc.addEvent("LightOn", 0, 3, gc);`

Expected Output: Exception will occurs as LightOn Event does not have extra integer parameters in its constructor.

**Getting the list of events: `List<Event> getEvents()`**

Input: `gc.addEvent("LightOn", 50000, gc);`

`gc.addEvent("LightOff", 50000, gc);`

`gc.getEvents();`

Expected Output: will return the list of events. The LightOn object and the LightOff object will be in the list in this case.

**Remove event from the list: `void removeEvent(Event event)`**

Input: `LightOn event = new LightOn(5000, gc)`

`gc.addEvent(event);`

`gc.removeEvent(event);`

Expected Output: will remove the LightOn object from the eventList attribute.

**Getting an Error Code: `int getError()`**

Input: `gc.getError();`

Expected Output: return the errorcode. 0 which is a default error code will be returned in this case.

**Getting Fixable for the Error Code: `Fixable getFixable(int errorcode)`**

Input: `gc.getFixable();`

Expected Output: return the Fixable object depends on the errorcode. FixWindow object for errorcode 1, PowerOn object for errorcode 2. Null will be returned in this case for the default errorcode 0.

### **Restore the greenhouse: void Restore(String dump)**

Input: gc.restore("dump.out");

Expected Output: Restore object will be created and prints "Restoring System". The Restore object will be deserialized the dump.out file to create the new GreenhouseControls object then fix the issue of the restored object then resumes the rest of the events from where its left

### **Restore with the file that does not exist: public Restore(String dump)**

Input: gc.restore("somethingnotexists");

Expected Output: Prints "No file found" to the console then the program will be finished.

### **Shutdown: void shutdown(long delayTime)**

Input: gc.shutdown(300);

Expected Output: the log file called "error.log" will be created with the errors depends on the errorcode attribute then set the lastEventDelayTime attribute to the delayTime parameter and serialize the self(gc) to "dump.out" file. "error.log" file will be empty as the default errorcode is 0 which indicates no error.

## **LightOn**

### **Constructor: public LightOn(long delayTime, Controller controller)**

Input: new LightOn(0, gc);

Expected Output: LightOn object will be created with the delay time of 0 and the controller of gc.

### **Action: void action()**

Input: new LightOn(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Light" and value of true.

### **What does the event done: String toString()**

Input: new LightOn(0, gc).toString()

Expected Output: Will return "Light is on" string.

## LightOff

**Constructor: public LightOff(long delayTime, Controller controller)**

Input: new LightOff(0, gc);

Expected Output: LightOff object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new LightOff(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Light" and value of false

**What does the event done: String toString()**

Input: new LightOff(0).toString()

Expected Output: Will return "Light is off" string.

## WaterOn

**Constructor: public WaterOn(long delayTime, Controller controller)**

Input: new WaterOn(0, gc);

Expected Output: WaterOn object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new WaterOn(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Water" and value of true.

**What does the event done: String toString()**

Input: new WaterOn(0, gc).toString()

Expected Output: Will return "Greenhouse water is on" string.

## WaterOff

**Constructor: public WaterOff(long delayTime, Controller controller)**

Input: new WaterOff(0, gc);

Expected Output: WaterOff object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new WaterOff(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Water" and value of false.

**What does the event done: String toString()**

Input: new WaterOff(0, gc).toString()

Expected Output: Will return "Greenhouse water is off" string.

## FansOn

**Constructor: public FansOn(long delayTime, Controller controller)**

Input: new FansOn(0, gc);

Expected Output: FansOn object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new FansOn(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Fans" and value of true.

**What does the event done: String toString()**

Input: new FansOn(0, gc).toString()

Expected Output: Will return "Greenhouse fans are on" string.

## FansOff

**Constructor: public FansOff(long delayTime, Controller controller)**

Input: new FansOff(0, gc);

Expected Output: FansOff object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new FansOff(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Fans" and value of false.

**What does the event done: String toString()**

Input: new FansOff(0, gc).toString()

Expected Output: Will return "Greenhouse fans are off" string.

## ThermostatNight

**Constructor: public ThermostatNight(long delayTime, Controller controller)**

Input: new ThermostatNight(0, gc);

Expected Output: ThermostatNight object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new ThermostatNight(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Thermostat" and value of "Night".

**What does the event done: String toString()**

Input: new ThermostatNight(0, gc).toString()

Expected Output: Will return "Thermostat on night setting" string.

## ThermostatDay

**Constructor: public ThermostatDay(long delayTime, Controller controller)**

Input: new ThermostatDay(0, gc);

Expected Output: ThermostatDay object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new ThermostatDay(0, gc).action();

Expected Output: It will call the controller's setVariable method with the key of "Thermostat" and value of "Day".

**What does the event done: String toString()**

Input: new ThermostatDay(0, gc).toString()

Expected Output: Will return "Thermostat on day setting" string.

## Bell

**Constructor: public Bell(long delayTime, int ring, Controller controller)**

Input: new Bell(0, 3, gc);

Expected Output: Bell object will be created with the delay time of 0 and the controller of gc. And two more Bell object will be added to the controller using the addEvent method of the controller with the delay time of 2000ms and 4000ms respectively

**Action: void action()**

Input: new Bell(0, 3, gc).action()

Expected Output: Do nothing

**What does the event done: String toString()**

Input: new Bell(0, 3, gc).toString()

Expected Output: Will return "Bing!" string.

# WindowMalfunction

**Constructor: public WindowMalfunction(long delayTime, Controller Controller)**

Input: new WindowMalfunction(0, gc);

Expected Output: WindowMalfunction object will be created with the delay time of 0 and the controller of gc.

**Action: void action() throws ControllerException**

Input: new WindowMalfunction(0, gc).action();

Expected Output: It will call the controller's setVariable method twice with the key of "errorcode" and value of 1, and the key of "windowok" and value of false. Then throw ControllerException exception object with the error message "WindowMalfunction"

**What does the event done: String toString()**

Input: new WindowMalfunction(0, gc).toString()

Expected Output: Will return "Winwdow Malfunction!" string.

# PowerOut

**Constructor: public PowerOut(long delayTime, Controller Controller)**

Input: new PowerOut(0, gc);

Expected Output: PowerOut object will be created with the delay time of 0 and the controller of gc.

**Action: void action() throws ControllerException**

Input: new PowerOut(0, gc).action();

Expected Output: It will call the controller's setVariable method twice with the key of "errorcode" and value of 2, and the key of "poweron" and value of false. Then throw ControllerException exception object with the error message "PowerOut"

**What does the event done: String toString()**

Input: new WindowMalfunction(0, gc).toString()

Expected Output: Will return "Power Outage!" string.



# Restart

**Constructor: public Restart(long delayTime, Controller controller, String filename)**

Input: new Restart(0, gc, "test.txt");

Expected Output: Restart object will be created with the delay time of 0, the controller of gc, and set the String attribute "eventsFile" to the filename.

**Action: void action()**

Input: new Restart(0, gc, "examples1.txt").action()

examples1.txt:

Event=ThermostatNight,time=0

Event=LightOn,time=2000

Event=WaterOff,time=8000

Event=ThermostatDay,time=10000

Event=Bell,time=9000

Event=WaterOn,time=6000

Event=LightOff,time=4000

Event=Terminate,time=12000

Expected Output: The Restart object that created will parse the "examples1.txt" file and add events to the controller, gc. 8 events will be added to the gc's eventlist.

**Action with the event name does not exist: void action()**

Input: new Restart(0, gc, "examples1.txt").action()

examples1.txt:

Event=ThermostatNight1,time=0

Expected Output: Will print "Wrong Even" and program will be finished.

**Action with the wrongly formatted event file: void action()**

Input: new Restart(0, gc, "examples1.txt").action()

examples1.txt:

Event1=ThermostatNight,time=0

Expected Output: Will print "Wrong Event File" and program will be finished.

**Action with the wrongly formatted event file: void action()**

Input: new Restart(0, gc, "examples1.txt").action()

examples1.txt:

Event=Bell,time=0,ring=-1

Expected Output: Will print "Wrong Event File" and program will be finished.

**Action with the filename that the file does not exist: void action()**

Input: new Restart(0, gc, "somethingnotexists").action()

Expected Output: Will print "No file exists" and program will be finished.

**What does the event done: String toString()**

Input: new Restart(0, gc, "test.txt").toString()

Expected Output: Will return "Restart System!" string.

## Terminate

**Constructor: public Terminate(long delayTime, Controller controller)**

Input: new Terminate(0, gc);

Expected Output: Terminate object will be created with the delay time of 0 and the controller of gc.

**Action: void action()**

Input: new Terminate(0, gc).action();

Expected Output: The program will be finished.

**What does the event done: String toString()**

Input: new Terminate(0, gc).toString()

Expected Output: Will return "Terminating" string.

## FixWindow

**Default Constructor: public FixWindow(Controller controller)**

Input: new FixWindow(gc);

Expected Output: FixWindow object will be created with the controller of gc.

**Fix issue: void fix()**

Input: new FixWindow(gc).fix()

Expected Output: It will call the controller's setVariable method twice with the key of "errorcode" and value of 0, and the key of "windowok" and value of true. Then calls its log() method.

**Log what does it done: void log()**

Input: new FixWindow(gc).log()

Expected Output: Will log out the time and the nature of the fix to the fix.log file to the current directory and prints "Window Fixed!" to the console.

## PowerOn

**Default Constructor: public PowerOn(Controller controller)**

Input: new PowerOn(gc);

Expected Output: PowerOn object will be created with the controller of gc.

**Fix issue: void fix()**

Input: new PowerOn(gc).fix()

Expected Output: It will call the controller's setVariable method twice with the key of "errorcode" and value of 0, and the key of "poweron" and value of true. Then calls its log() method.

**Log what does it done: void log()**

Input: new PowerOn(gc).log()

Expected Output: Will log out the time and the nature of the fix to the fix.log file to the current directory and prints "Power On!" to the console.

## Event

**Start time: public void start()**

Input: new LightOn(1000, gc).start();

Expected Output: set the eventTime attribute to current time + 1000msec

**Start for Restored Event: public void start(long lastEventDelayTime)**

Input: new LightOn(10000, gc).start(8000);

Expected Output: set the eventTime attribute to current time + 10000ms – 8000ms

**Get Delay Time: public long getDelayTime()**

Input: new LightOn(1000, gc).getDelayTime();

Expected Output: Returns the delay time of the event, so 1000 in this case

**Check if ready: public boolean ready()**

Input: LightOn e = new LightOn(1000, gc);

e.start();

e.ready();

Expected Output: return false as the ready() function called right after the start method but the delayTime is 1000msec. if ready(0) function called after 1000msec than it will be return true.

**Run the malfunction: public void run()**

Input: LightOn e = new LightOn(1000, gc)

Thread t = new Thread(e);

t.start();

Expected Output: the event thread will be started and will be keep checking in the loop if the event is ready to action by ready() method. Will trigger action() method and prints its action to the console.

**Stop the event: public void stop()**

Input: LightOn e = new LightOn(1000, gc)

Thread t = new Thread(e);

```
t.start();
```

```
e.stop();
```

Expected Output: the event thread will be stop as the stop function set the running flag to false which will quit from the loop for waiting the event ready.