Task 1:

Taking cue from the help docuement given in the homework page, the module is started and screenshots containing the working of the application are taken and are being submitted along with the homework.

Key Learnings:

IOFMessageListener is the core module of the controller which listens onto all OF messages from all the switches and hence all the modules implementing this interface will receive the OF packets sent to controller.

IOFMessageListener - receive method: This method is called as and when an OF packet reaches the controller. This is very powerful method since this is core method where a pkt is observed and a decision is taken regarding the flow table entry in the switches.

Task 2:

Extending the Mac learning logic from the task above: Once we learn the MAC, its location (Switch, Switchport). This task is about controller working as a learning L3 switch. I have completed this task in two ways.

Way 1:

Instructions:

1)File: MySimpleSwitch.java
2)Please rename to MACTracker.java to run it. I have extended the above Task code.
3)add net.floodlightcontroller.mactracker.MACTracker,\ to /floodlight/src/main/resources/floodlightdefault.properties
4)add net.floodlightcontroller.mactracker.MACTracker to
/floodlight/src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule

Algorithm:

1)To send the packet(ARP or ICMP) to a host whose location is known, write a PACKET OUT to the destination switch.
2)To send the packet(ARP) to a host whose location is not known broadcast the packet using OFPP_FLOOD option of OFPort on OF switch. This is done by writing an action with OFPP_FLOOD added to it and sending this message to the switch which will broadcast on all interfaces other than the interface the original message came from.
3)When this ARP request is broadcast, the original destination replies(ARP reply) from which destination location is known and this information is used as in step 1.

Results:

pingall was used to test the connectivity in single switch, linear, tree topologies on Mininet and screenshots for some of the test cases are being submitted. The controller is successfully functioning as a L3 switch switching every packet.

Way 2:

Instructions:

1)File: MySimpleSwitchFlowTables.java
2)Please rename to MACTracker.java to run it. I have extended the above Task code.
3)add net.floodlightcontroller.mactracker.MACTracker,\ to /floodlight/src/main/resources/floodlightdefault.properties
4)add net.floodlightcontroller.mactracker.MACTracker to
/floodlight/src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule

Algorithm:

1)To send the packet(ARP or ICMP) to a host whose location is known, get the route from source host to destination host which involves all the switches along the path. For this part, IRoutingService module was used. Along a valid route, all the switches are programmed to allow this flow from source switchport to destination switchport. The OFFlowMod message is sent to all the switches along the path with all the FLAGS set and filling the action with corresponding output port.

2)To send the packet(ARP) to a host whose location is not known broadcast the packet using OFPP_FLOOD option of OFPort on OF switch. This is done by writing an action with OFPP_FLOOD added to it and sending this message to all the switches which will broadcast on all interfaces other than the interface the original message came from.
3)When this ARP request is broadcast, the original destination replies(ARP reply) from which destination location is known and this information is used as in step 1.

Results:
pingall was used to test the connectivity in single switch, linear, tree topologies on Mininet and screenshots for some of the test cases are being submitted. The controller is successfully functioning and once rules are written to switches, controller is nto receiving any more packets and the switch is able to send the packets along the necessary path.

Task 3:

Instructions:
1)File: MySimpleFirewall.java
2)Please rename to MACTracker.java to run it. I have extended the above Task code.
3)add net.floodlightcontroller.mactracker.MACTracker,\ to /floodlight/src/main/resources/floodlightdefault.properties
4)add net.floodlightcontroller.mactracker.MACTracker to
/floodlight/src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule

Algorithm:
1)In addition to above logic, the traffic from host h2 to h3 and h3 to h2 is dropped by writing a OFFlowMod message to the switch and not filling any action. If action is empty, the packets are dropped according to description.

Results:
pingall was used to test the connectivity in single switch 3 hosts on Mininet and screenshots for some of the test cases are being submitted. The controller is successfully functioning and once rules are written to switches, h2 to h3, h3 to h2 packets are dropped.

Task 4:

Simple app:
Description:
This application serves as a rest service for providing bandwidth as a service. As it provides a rest interface, it is written as 4 files instead of one mentioned in the homework. Please note that to make this application dynamic and interactive, it is written as a rest service. Once the request comes for providing bandwidth between two hosts, the path between the requested hosts is marked and any flows not matching this source and destination will be dropped. IRoutingService module is used to get the route between the hosts, this path stored and verified for every new flow which shares this path. Two options are available from REST API, reserve for reserving path, free for freeing the reserved path.

1)Files: BWaaS.java IBWaaService.java BWaaServiceResource.java BWaaSWebRoutable.java
2)add net.floodlightcontroller.bwaas.BWaaS,\ to /floodlight/src/main/resources/floodlightdefault.properties
3)add net.floodlightcontroller.bwaas.BWaaS to
/floodlight/src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule

Unit Test:
1) Start the controller with above module added
2) start mininet tree topo:
   sudo mn --controller=remote --switch ovsk --topo=tree,2,2
3) start tcp server on host 3:
   xterm h3
   iperf -s -p 500

4) start tcp client with time limit on host 2:
   xterm h2
   iperf -c 10.0.0.3 -p 500 -t 30
5) start tcp client with time limit on host 1 when above client is still running:
   xterm h1
   iperf -c 10.0.0.3 -p 500 -t 10
6) Observe the speed of the transfer on host 1.
7) Request bandwidth reserve using REST API:
   curl -s http://localhost:8080/wm/bwaas/10.0.0.1/10.0.0.3/reserve
8) Now repeat steps 4, 5 and observe the speed of transfer on host 1.
9) Request bandwidth free using REST API:
   curl -s http://localhost:8080/wm/bwaas/10.0.0.1/10.0.0.3/free
10) Repeat steps 4, 5 to see the state was restored.

Results:
For reserve and free options, the tested speed was captured using screenshots and is being attached with this assignment.

Learnings:
I have learnt how to write a rest service in java.
In depth understanding of architecture of Controller.