1) Move the list of agents for each container to a global state within the Master Member code.

Ans: This has been briefly addressed by creating an ArrayList of type AID (jade.core.AID) within the setup() function of the Master Member. There are three ArrayLists, one for each type of member – Permanent Members, Temporary Members and Regular Members.

```
ArrayList<AID> temporary_members = new ArrayList<AID>();
ArrayList<AID> permanent_members = new ArrayList<AID>();
ArrayList<AID> regular_members = new ArrayList<AID>();
```

The Master Member is the one which has the highest priority amongst all the members, and it is created before everyone else, so it is being used to listen to platform-wide events from the AMS. This is done by setting up an AMSSubscriber (jade.domain.introspection.AMSSubscriber) behaviour so that the AMS (the white pages service) automatically messages the Master Member whenever an event happens. Whenever an agent is being born, the Master Member checks which container the agent attaches itself to. Once that is done, it appends the AID for the agent to the appropriate ArrayList.

```
AMSSubscriber myAMSSubscriber = new AMSSubscriber() {
            protected void installHandlers(Map handlers) {
            EventHandler creationsHandler = new EventHandler() {
            public void handle(Event ev) {
            BornAgent ba = (BornAgent) ev;
            if (ba.getWhere().getName().equals("Permanent Members"))
            {
                permanent_members.add(ba.getAgent());
            } else if (ba.getWhere().getName().equals("Temporary Members"))
            {
                temporary_members.add(ba.getAgent());
            } else if (ba.getWhere().getName().equals("Regular Members"))
            {
                regular_members.add(ba.getAgent());
            }
            }
            };
            handlers.put(IntrospectionVocabulary.BORNAGENT,
            creationsHandler);
            }
        };
```

This approach reduces the number of API calls. In the previous approach, the ArrayLists were freshly made every time some work needed to be done – which means one request from the AMS and one request from the Master Member respectively. In this approach, the Master Member only sends one request at the time of registration of the behaviours, and the AMS automatically responds whenever an agent is born.

2) Pick agent input from a file.

Changes are mainly done to the main calling function (stored within Main). File input is being taken by creating a file called **input.txt** within the main directory of the project and running the code. The agent creation is then handled accordingly depending on the input within the file.

```
try {
                acm = cc.createNewAgent("Master Member", "MasterMember", null)
;
                acm.start();
            } catch (Exception e) {
                e.printStackTrace();
            }
        int x=4;
        try {
            File myObj = new File("input.txt");
            Scanner myReader = new Scanner(myObj);
            while (myReader.hasNextLine()) {
              String data = myReader.nextLine();
              x = Integer.parseInt(data);
              if (x==1) {
                try {
                    System.out.println("Permanent Member");
                    ac1 = c1.createNewAgent("Member " + i, "PermanentMember",
null);
                    ac1.start();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                Main.i++;
            }
            else if (x==2) {
                try {
                    System.out.println("Regular Member");
                    ac2 = c2.createNewAgent("Member " + i, "RegularMember", nu
ll);
                    ac2.start();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                Main.i++;
            }
            else if (x==3) {
                try {
                    System.out.println("Temporary Member");
                    ac3 = c3.createNewAgent("Member " + i, "TemporaryMember",
null);
                    ac3.start();
                } catch (Exception e) {
                    e.printStackTrace();
                }
```

```
                Main.i++;
            }
            else if (x == 0) {
                System.out.println("Quit");
                System.exit(0);
            }
            }
            myReader.close();
          } catch (Exception e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
          }
```

3) Permanent Members only vote when they get a message.

Permanent Members now use a CyclicBehaviour (jade.core.behaviours.CyclicBehaviour) for performing the voting action. Voting is assigning a boolean value to the Regular Members or the Temporary Members (which is randomly generated). However, this voting process is not triggered if there are multiple Permanent Members but no Regular Members or Temporary Members. This is because voting is triggered on a message from the Master Member, otherwise the Permanent Member remains in a blocked state. Once the Permanent Member gets the values it immediately sets up a response to the Master Member. A conversation ID is attached to the ACLMessage (jade.lang.acl.ACLMessage) so that the multiple behaviours being run on the Master Member side do not confuse the messages being sent from the Permanent Members with the messages being sent from the AMS.

```
Behaviour receiveMessage = new CyclicBehaviour(this) {
            public void action()
            {
                MessageTemplate mt = MessageTemplate.MatchConversationId("C");
                ACLMessage ip = receive(mt);
                if (ip != null)
                {
                    try {
                        ArrayList<Type> data = (ArrayList<Type>) ip.getContent
Object();
                        ArrayList<Type> votes = new ArrayList<Type>();
                        for (Type i : data)
                    {
                        System.out.println(i.name.toString() + " - " + i.type)
;
                        double chance = Math.random();
                        if (chance <= 0.5)
                        {
                            votes.add(new Type(i,true));
                        }
                        else
                        {
                            votes.add(new Type(i,false));
```

```
                    }
                }
                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.setContentObject(votes);
                msg.addReceiver(ip.getSender());
                msg.setSender(getAID());
                msg.setConversationId("A");
                send(msg);
                } catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
            else
            {
                block();
            }

        }
    };
```

4) Master Member should send messages to all the Permanent Members after “ticks” or fixed timed schedules to initiate the voting process.

The Master Member now uses a TickerBehaviour (jade.core.behaviours.TickerBehaviour) to inform the Permanent Members to start the voting process. It appends the list of Regular Members and Temporary Members and sends it to the Permanent Members so that they can vote on them. Voting takes place on all of the Regular Members and Temporary Members present at the time of voting. If no Regular Members or Temporary Members are present, voting does not proceed. The Permanent Members are dependent on the Master Member for its messages, which occur at intervals of 5000 ms.

```
Behaviour startVoting = new TickerBehaviour(this,5000) {
            protected void onTick()
            {
               for (AID i : permanent_members)
               {
                   try {
                    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                    ArrayList<Type> obj = new ArrayList<Type>();
                    for (AID k : temporary_members)
                    {
                        obj.add(new Type(k,"Temporary Member"));
                    }
                    for (AID k : regular_members)
                    {
                        obj.add(new Type(k,"Regular Member"));
                    }
                    msg.setContentObject(obj);
```

```
                    msg.addReceiver(i);
                    msg.setSender(getAID());
                    msg.setConversationId("C");
                    send(msg);
                   } catch (Exception e)
                   {
                       e.printStackTrace();
                   }

               }
           }
       };
```

The Master Member also has another CyclicBehaviour (jade.core.behaviours.CyclicBehaviour) to receive members from the Permanent Members.

```
Behaviour cyclic = new CyclicBehaviour(this) {
            public void action()
            {
                MessageTemplate mt = MessageTemplate.MatchConversationId("A");
                ACLMessage msg = id.receive(mt);
                if (msg != null)
                {
                    System.out.println("Sent by: " + msg.getSender());
                    try {
                    ArrayList<Type> data = (ArrayList<Type>) msg.getContentObj
ect();
                    for (Type i : data)
                    {
                        System.out.println(i.name.toString() + " - " + i.type
+ " - " + i.vote);
                    }
                    } catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                }
            }
        };
```

(In multiple sections of the code, a “Type” class can be found. Type is actually a user-implemented class which implements the Serializable - java.io.Serializable - interface). It uses constructor overloading to allow for more flexibility in initializing the objects of the class. Objects of this class are used in agent communication (as described above).

```
import jade.core.AID;
import java.io.Serializable;
```

```java
public class Type implements Serializable {
    AID name;
    String type;
    boolean vote;
    public Type(AID name,String type,boolean vote)
    {
        this.name = name;
        this.type = type;
        this.vote = vote;
    }
    public Type(AID name,String type)
    {
        this.name = name;
        this.type = type;
        this.vote = false;
    }
    public Type(Type obj,boolean vote)
    {
        this.name = obj.name;
        this.type = obj.type;
        this.vote = vote;
    }
}
```