```java
package set10111.simulation;

import java.util.ArrayList;
38
//was previously named as SellerAgent because similar to Seller Agent of practical 6
public class TimeTablingAgent extends Agent {
    private HashMap<String, ACLMessage> BothAgreed = new HashMap<>();
    private AID tickerAgent;
    private ArrayList<AID> studentAgents = new ArrayList<>();
    ArrayList<String> moduleNames = new ArrayList<String>();
    ArrayList<Student> studentList1 = new ArrayList<Student>();
    Random random = new Random();
    private HashMap<AID, Student> studentList = new HashMap<>();
    private boolean result = false;

    HashMap<String, Student> studentListFromMain;

    private Codec codec = new SLCodec();
    private Ontology ontology = TimetablingOntology.getInstance();

    @Override
    protected void setup() {
        // add this agent to the yellow pages
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("seller");
        sd.setName(getLocalName() + "-seller-agent");
        dfd.addServices(sd);
        try {
            DFService.register(this, dfd);
        } catch (FIPAException e) {
            e.printStackTrace();
        }

        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(ontology);

        // get the title of book from passed arguments to agent
            Object[] args = getArguments();
            studentListFromMain = ((Customm) args[0]).timetable;




        addBehaviour(new TickerWaiter(this));
//      addBehaviour(bnew SwapRequestServer(myAgent));
    }

    public class TickerWaiter extends OneShotBehaviour {

        // behaviour to wait for a new day
        public TickerWaiter(Agent a) {
            super(a);
        }

        @Override
        public void action() {
            MessageTemplate mt = MessageTemplate.or(MessageTemplate.MatchContent("new
  day"),
                    MessageTemplate.MatchContent("terminate"));
            ACLMessage msg = myAgent.receive(mt);
```

```
96 //          if (msg != null) {
97 //              if (tickerAgent == null) {
98 //                  tickerAgent = msg.getSender();
99 //              }
100 //             if (msg.getContent().equals("new day")) {
101                 myAgent.addBehaviour(new FindStudents(myAgent));
102
103                 myAgent.addBehaviour(new TimeTableGenerator());
104                 doWait(2000);
105                 myAgent.addBehaviour(new SendTimeTable(myAgent));
106
107                 CyclicBehaviour os = new SwapRequestServer(myAgent);
108                 myAgent.addBehaviour(os);
109 //                ArrayList<Behaviour> cyclicBehaviours = new ArrayList<>();
110 //                cyclicBehaviours.add(os);
111                 myAgent.addBehaviour(new End(myAgent));
112 //             } else {
113 //                 // termination message to end simulation
114 //                 myAgent.doDelete();
115 //             }
116 //         } else {
117 //             block();
118 //         }
119         }
120
121         public class TimeTableGenerator extends OneShotBehaviour {
122
123             @Override
124             public void action() {
125
126                 for (int i = 0; i < studentAgents.size(); i++) {
127                     AID studentAid = studentAgents.get(i);
128                     String name = studentAid.getLocalName();
129
130                     Student student = studentListFromMain.get(name);
131
132                     studentList.put(studentAid, student);
133                 }
134
135 //              moduleNames.clear();
136 //
137                 // add new dummy module
138                 moduleNames.add("Module1");
139                 moduleNames.add("Module2");
140                 moduleNames.add("Module3");
141
142 //              // create students list based upon number of student agents in system
143 //              for (AID sa : studentAgents) {
144 //                  Student student = new Student();
145 //                  student.name = sa.getLocalName();
146 //                  student.moduleList = new HashMap<>();
147 //
148 //                  // add to student list
149 //                  studentList.put(sa, student);
150 //
151 //              }
152 //
153 //              // Now assign random time slots and random tutorial groups to each student
    for
154 //              // each module
155 //              for (String moduleName : moduleNames) {
156 //                  System.out.println("");
```

```
157 //                  System.out.println(moduleName);
158 //                  System.out.println("---------");
159 //
160 //                  int totalStudentG1 = 0;
161 //                  int totalStudentG2 = 0;
162 //
163 //                  // generate 2 time slots for each tutorial
164 //                  ArrayList<TimeSlot> tutTime = new ArrayList<TimeSlot>();
165 //                  for (int i = 0; i < 2; i++) {
166 //
167 //                      int startTime = random.nextInt(9) + 9;
168 //                      int endTime = startTime + 1;
169 //                      int day = random.nextInt(5) + 1; // set the day of the tutorial
170 //
171 //                      TimeSlot slot = new TimeSlot();
172 //                      slot.setModuleName(moduleName);
173 //                      slot.setGroupId(0); // set to 0 because will be updated when actual
    timeslot is assigned
174 //                      slot.setDate(day);
175 //                      slot.setStartTime(startTime);
176 //                      slot.setEndTime(endTime);
177 //                      slot.setStatus("AsiignedByTA");
178 //                      tutTime.add(slot); // adding dummy slot to tutorial times arraylist
179 //                  }
180 //
181 //                  // assign each student to random timeslot of tutorial
182 //                  for (AID aid : studentAgents) {
183 //                      Student student = studentList.get(aid);
184 //
185 //                      int groupId = random.nextInt(2); // generate random number between
    0 and 1
186 //
187 //                      TimeSlot slot = tutTime.get(groupId);
188 //                      slot.setGroupId(groupId + 1); // assign the group id to slot.
    adding one because want tp statr
189 //                                                      // group ids from 1 instead of 0
190 //
191 ////                      Module module = new Module();
192 ////                      module.name = moduleName;
193 ////                      module.timeSlot = slot;
194 //
195 //                      // add this slot to the module list of student
196 //                      studentList.get(aid).moduleList.put(moduleName, slot);
197 //
198 //                      if (groupId == 0) {
199 //                          totalStudentG1++;
200 //                      } else {
201 //                          totalStudentG2++;
202 //                      }
203 //
204 ////                      System.out.println(student.name);
205 //                      System.out.println(
206 //                              student.name + "   ----   slot : " + slot.getModuleName() +
    " " + slot.getGroupId()
207 //                                      + " " + slot.getDate() + " " + slot.getStartTime()
    + " " + slot.getEndTime());
208 //                  }
209 //
210 //                  System.out.println("Total Students enrolled on module : " +
    studentAgents.size());
211 //                  System.out.println(" student in Group 1 : " + totalStudentG1);
212 //                  System.out.println(" student in Group 2 : " + totalStudentG2);
```

```
213 //              }
214             }
215 
216         }
217 
218         public class FindStudents extends OneShotBehaviour {
219 
220             public FindStudents(Agent a) {
221                 super(a);
222             }
223 
224             @Override
225             public void action() {
226                 DFAgentDescription studentTemplate = new DFAgentDescription();
227                 ServiceDescription sd = new ServiceDescription();
228                 sd.setType("student");
229                 studentTemplate.addServices(sd);
230                 try {
231                     studentAgents.clear();
232                     DFAgentDescription[] agentsType1 = DFService.search(myAgent,
    studentTemplate);
233                     for (int i = 0; i < agentsType1.length; i++) {
234                         studentAgents.add(agentsType1[i].getName()); // this is the AID
235                     }
236                 } catch (FIPAException e) {
237                     e.printStackTrace();
238                 }
239 
240             }
241 
242         }
243 
244         public class End extends CyclicBehaviour {
245 
246             public End(Agent a) {
247                 super(a);
248             }
249 
250             @Override
251             public void action() {
252                 addBehaviour(new FindStudents(myAgent));
253                 if (studentAgents.size() < 1) {
254                     myAgent.doDelete();
255                     System.out.println("No more students active so shutting down TA
    agent");
256                 }
257                 else {
258 //                    block();
259                 }
260 
261             }
262 
263         }
264 
265         public class SendTimeTable extends OneShotBehaviour {
266 
267             public SendTimeTable(Agent a) {
268                 super(a);
269             }
270 
271             @Override
272             public void action() {
```

```
273                 try {
274
275                     // Prepare the action request message
276 //                  ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
277 //                  msg.setLanguage(codec.getName());
278 //                  msg.setOntology(ontology.getName());
279
280                     // include following code in for loop if want to send timetable to all
    student
281                     // agents
282
283                     for (int i = 0; i < studentAgents.size(); i++) {
284
285                         // Prepare the action request message
286                         ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
287                         msg.setLanguage(codec.getName());
288                         msg.setOntology(ontology.getName());
289
290                         AID student = studentAgents.get(i);
291
292                         // Prepare the content.
293                         TimeTable timetable = new TimeTable();
294                         timetable.setName(student.getLocalName());
295
296                         ArrayList<TimeSlot> timeSlots = new ArrayList<TimeSlot>();
297
298                         // todo: remove module class and instead store the time slots in
    the module list
299                         // and name list to time slots
300                         TimeSlot slot =
    studentList.get(student).moduleList.get(moduleNames.get(0));
301                         timeSlots.add(slot);
302
303                         slot = studentList.get(student).moduleList.get(moduleNames.get(1));
304                         timeSlots.add(slot);
305 //
306                         // comment this third module if test case involve only 2 modules
    but you will also have to change
307                         // constrints in onltology
308                         slot = studentList.get(student).moduleList.get(moduleNames.get(2));
309                         timeSlots.add(slot);
310
311                         timetable.setTutorialAssignment(timeSlots);
312
313                         AcceptTimetable order = new AcceptTimetable();
314                         order.setSenderAgentAid(myAgent.getAID()); // setting the aid of
    the sender which is this agent
315                         order.setTimeTable(timetable);
316
317                         msg.addReceiver(student); // sellerAID is the AID of the Seller
    agent
318
319                         // IMPORTANT: According to FIPA, we need to create a wrapper Action
    object
320                         // with the action and the AID of the agent
321                         // we are requesting to perform the action
322                         // you will get an exception if you try to send the sell action
    directly
323                         // not inside the wrapper!!!
324                         Action request = new Action();
325                         request.setAction(order);
326                         request.setActor(student); // the agent that you request to perform
```

```
   the action
327
328                         // Let JADE convert from Java objects to string
329                         getContentManager().fillContent(msg, request); // send the wrapper
   object
330                         send(msg);
331                     }
332
333                 } catch (CodecException ce) {
334                     ce.printStackTrace();
335
336                 } catch (OntologyException oe) {
337                     oe.printStackTrace();
338                 }
339
340             }
341
342         }
343
344     }
345
346     public class SwapRequestServer extends CyclicBehaviour {
347
348         public SwapRequestServer(Agent a) {
349             super(a);
350         }
351
352         @Override
353         public void action() {
354
355             try {
356                 MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
357                 ACLMessage msg = myAgent.receive(mt);
358                 if (msg != null) {
359                     ACLMessage reply = msg.createReply();
360                     ContentElement ce = null;
361
362                     // Let JADE convert from String to Java objects
363                     // Output will be a ContentElement
364                     ce = getContentManager().extractContent(msg);
365
366                     // check if content is an Action
367                     if (ce instanceof Action) {
368                         Action act = ((Action) ce);
369                         Concept action = act.getAction();
370
371                         // check if action is of type OwnsTimeSlot
372                         if (action instanceof UpdateTimetable) {
373                             UpdateTimetable swap = (UpdateTimetable) action;
374                             // Extract the TimeSlot from action
375                             TimeSlot timeslot = swap.getTimeSlot();
376                             String moduleName = timeslot.getModuleName();
377                             AID student1 = swap.getStudentRequested();
378                             // change following to AID of second student once fillContent()
   issue is fixed in case1 of student agent's SendSwapRequest behaviour
379                             AID student2 = swap.getStudentOffered();
380                             boolean second=false;
381                             // call the method to perform swap
382 //                          boolean isSwapped = swapTimeSlots(student1, moduleName,
   student2);
383
384
```

```
385                             TimeSlot s1_slot =
    studentList.get(student1).moduleList.get(moduleName);
386                             TimeSlot s2_slot =
    studentList.get(student2).moduleList.get(moduleName);
387
388                             String id1
    =String.valueOf(s1_slot.getDate()).concat(String.valueOf(s1_slot.getStartTime())).concat(St
    ring.valueOf(s1_slot.getEndTime()));
389                             String id2 =
    String.valueOf(s2_slot.getDate()).concat(String.valueOf(s2_slot.getStartTime())).concat(Str
    ing.valueOf(s2_slot.getEndTime()));
390
391                             String firstRequest
    =moduleName.concat(student1.getLocalName()).concat("-"+student2.getLocalName()).concat(id1)
    .concat(id2);
392                             BothAgreed.put(firstRequest, msg);
393                             String secondRequest
    =moduleName.concat(student2.getLocalName()).concat("-"+student1.getLocalName()).concat(id2)
    .concat(id1);
394
395                             second = BothAgreed.get(secondRequest) == null ? false : true;
396
397
398
399                             if (studentList.containsKey(student1) &&
    studentList.containsKey(student2) && second == true) {
400
401                                 // get the time slot of student's specified module and
    replace it with other
402                                 // student
403                                  s1_slot =
    studentList.get(student1).moduleList.remove(moduleName);
404                                  s2_slot =
    studentList.get(student2).moduleList.remove(moduleName);
405
406 //                                swap slots  use uncommented 2 lines or the next syntax
    where assign each property separately.
407 //                                if reference type does not cause issues then use following
    uncommented 2 lines otherwise dont
408                                 studentList.get(student1).moduleList.put(moduleName,
    s2_slot);
409                                 studentList.get(student2).moduleList.put(moduleName,
    s1_slot);
410
411
412 //                                studentList.get(student2).moduleList.get(moduleName).timeSl
    ot.replace("AsiignedByTA", slot1);
413
414                                 result = true;
415                                  id1
    =String.valueOf(s1_slot.getDate()).concat(String.valueOf(s1_slot.getStartTime())).concat(St
    ring.valueOf(s1_slot.getEndTime()));
416                                  id2 =
    String.valueOf(s2_slot.getDate()).concat(String.valueOf(s2_slot.getStartTime())).concat(Str
    ing.valueOf(s2_slot.getEndTime()));
417
418                                 System.out.println(student1.getLocalName() + "Swapped  " +
    id1 + " > "+ id2 );
419                             } else {
420                                 result = false;
421                             }
422
```

```
423 //                          -------------continue from here-------
424                             // which will allow manual creation of desired swapping
    scenarios // this checks
425                             // if student agent has not already agrred to swap this slot
    with other agent
426
427                             // send confirm message
428
429                             if (result) {
430                                 reply = BothAgreed.get(firstRequest).createReply();
431                                 reply.setPerformative(ACLMessage.INFORM);
432
433                                 Result result = new Result();
434
435                                 result.setAction(act);
436                                 result.setValue(timeslot); // use predicate
437
438                                 // craete content element list which will have orignal
    action requested by other
439                                 // student and offer of this student
440                                 ContentElementList cml = new ContentElementList();
441
442                                 // do we need to use content element list now because
    result itself has the previous action
443 //                              cml.add(act);
444
445                                 // add offer to the content list a swell
446                                 cml.add(result);
447
448                                 // Let JADE convert from Java objects to string
449                                 getContentManager().fillContent(reply, cml);
450 //                              System.out.println("--From TA ---" + cml);
451                                 myAgent.send(reply); // send reply/offer to other students
452
453
454                                 ACLMessage msg1 = BothAgreed.get(secondRequest);
455                                 ACLMessage reply1 =
    BothAgreed.get(secondRequest).createReply();
456                                 ContentElement ce1 = null;
457
458                                 // Let JADE convert from String to Java objects
459                                 // Output will be a ContentElement
460                                 ce1 = getContentManager().extractContent(msg1);
461                                 Action act1 = ((Action) ce1);
462                                 Concept action1 = act1.getAction();
463
464                                 UpdateTimetable swap1 = (UpdateTimetable) action1;
465                                 // Extract the TimeSlot from action
466                                 TimeSlot timeslot1 = swap1.getTimeSlot();
467
468                                 reply1.setPerformative(ACLMessage.INFORM);
469
470                                 Result result1 = new Result();
471
472                                 result1.setAction(act1);
473                                 result1.setValue(timeslot1); // use predicate
474
475                                 // craete content element list which will have orignal
    action requested by other
476                                 // student and offer of this student
477                                 ContentElementList cml1 = new ContentElementList();
478
```

```
479                                 // do we need to use content element list now because
    result itself has the previous action
480 //                              cml.add(act);
481
482                                 // add offer to the content list a swell
483                                 cml1.add(result1);
484
485                                 // Let JADE convert from Java objects to string
486                                 getContentManager().fillContent(reply1, cml1);
487 //                              System.out.println("--From TA ---" + cml);
488                                 myAgent.send(reply1); // send reply/offer to other students
489
490                             } else {
491
492 //                              reply.setPerformative(ACLMessage.FAILURE);
493 //                              // should this be failure or refuse
494 ////                                 reply.setPerformative(ACLMessage.REFUSE);
495 //                              System.out.println("Error swapping timeslots");
496 //
497 //                              myAgent.send(reply); // send reply/offer to other students
498                             }
499
500                         }
501                     }
502                 } else {
503                     block();
504                 }
505
506             } catch (CodecException ce) {
507                 ce.printStackTrace();
508             } catch (OntologyException oe) {
509                 oe.printStackTrace();
510             }
511
512         }
513
514 //Notes for all agents not only for timetabling agent
515         // todo- one way to do swap is that both student agents request timetabling
516         // agents to swap the agreed time slot
517         // In that scenario timetabling agent will wait for reply from both students for
518         // the same slotId but with different groups before Timetabling can perform
519         // actual swap
520
521 //      OR
522
523         // Other way can be only one of the student agent sends the swap request with
524         // details of both students and Timetabling agent performs the swap.
525
526         // ---------
527         // once the swap is completed Timetabling agent can send the confirmation in the
528         // form of Result see workbook. That response can be
529         // used by student agents to update their local assignedtimetable varibale
530
531         // Whilst the swap is being carried out by timetabling agent, we should avoid
532         // the scenario of same student aganets agreeing swap of same timeslot with
533         // other
534         // students. To handle we can create an hasmap varriable called
535         // beingSwaapedByTimetablingAgent which can store the ids of time slots which
536         // are sent to timetabling agent
537         // for swap and not negotiate on the timeslots with other student agents if the
538         // ids of timeslots is also on the beingSwaapedByTimetablingAgent
539
```

```
540         // if swap was success then remove that time slot from both timeSlotsToSwap and
541         // beingSwaapedByTimetablingAgent
542         // if failed then remove that time slot from beingSwaapedByTimetablingAgent, so
543         // that student can start swapping negotiation with other students again it
544         // again
545 //      handle swapping of timeslots
546 //      public boolean swapTimeSlots(final AID student1, final String moduleName, final AID
    student2) {
547 //          try {
548 //              addBehaviour(new OneShotBehaviour() {
549 //                  @Override
550 //                  public void action() {
551 //                      // check both students are on the timetable list
552 //                      if (studentList.containsKey(student1) &&
    studentList.containsKey(student2)) {
553 //
554 //                          // get the time slot of student's specified module and replace
    it with other
555 //                          // student
556 //                          TimeSlot s1_slot =
    studentList.get(student1).moduleList.get(moduleName).timeSlot;
557 //                          TimeSlot s2_slot =
    studentList.get(student2).moduleList.get(moduleName).timeSlot;
558 //
559 ////                             swap slots  use uncommented 2 lines or the next syntax
    where assign each property separately.
560 ////                             if reference type does not cause issues then use following
    uncommented 2 lines otherwise dont
561 //                          studentList.get(student1).moduleList.get(moduleName).timeSlot =
    s2_slot;
562 //                          studentList.get(student2).moduleList.get(moduleName).timeSlot =
    s1_slot;
563 //
564 //                          // replace slot of student 1 with student 2
565 //
566 //                          TimeSlot slot2 = new TimeSlot();
567 //                          slot2.setModuleName(s2_slot.getModuleName());
568 //                          slot2.setGroupId(s2_slot.getGroupId());
569 //                          slot2.setDate(s2_slot.getDate());
570 //                          slot2.setStartTime(s2_slot.getStartTime());
571 //                          slot2.setEndTime(s2_slot.getEndTime());
572 //                          slot2.setStatus(s2_slot.getStatus());
573 //
574 ////                             studentList.get(student1).moduleList.get(moduleName).timeSl
    ot.replace("AsiignedByTA", slot2);
575 //
576 //                          // replace slot of student 2 with student 1
577 //
578 //                          TimeSlot slot1 = new TimeSlot();
579 //                          slot1.setModuleName(s2_slot.getModuleName());
580 //                          slot1.setGroupId(s2_slot.getGroupId());
581 //                          slot1.setDate(s2_slot.getDate());
582 //                          slot1.setStartTime(s2_slot.getStartTime());
583 //                          slot1.setEndTime(s2_slot.getEndTime());
584 //                          slot1.setStatus(s2_slot.getStatus());
585 //
586 ////                             studentList.get(student2).moduleList.get(moduleName).timeSl
    ot.replace("AsiignedByTA", slot1);
587 //
588 //                          result = true;
589 //                      } else {
590 //                          result = false;
```

```
591 //                      }
592 //
593 //                  }
594 //              });
595 //          } catch (Exception e) {
596 //              e.printStackTrace();
597 //          }
598 //
599 //          return result;
600 //      }
601     }
602
603     public class EndDayListener extends CyclicBehaviour {
604
605
606         public EndDayListener(Agent a) {
607             super(a);
608         }
609
610         @Override
611         public void action() {
612
613
614         }
615
616     }
617 }
618
```