

OBJECT ORIENTATED PROGRAMMING

660047784: 50%

660037119: 50%

Date	Start Time	End Time	Driver	Observer	
02/03/2017	13:30	14:30	660047784	660037119	
	14:30	16:00	660037119	660047784	
08/03/2017	12:30	13:30	660037119	660047784	
10/03/2017	12:30	14:30	660037119	660047784	
14/03/2017	14:40	16:30	660037119	660047784	
22/03/2017	12:30	15:30	660047784	660037119	
23/03/2017	13:30	14:30	660047784	660037119	
	14:30	15:30	660037119	660047784	
	14:30	16:00	660047784	660037119	
24/03/2017	12:30	13:30	660047784	660037119	
27/03/2017	11:30	13:30	660037119	660047784	
28/03/2017	14:30	16:30	660037119	660047784	
29/03/2017	12:30	23:59	660047784	660037119	Regular changes throughout
30/03/2017	00:00	05:00	660047784	660037119	Regular changes throughout

```
1 package university;
2
3 import java.io.Serializable;
4 import java.io.IOException;
5 import java.io.ObjectOutputStream;
6 import java.io.FileOutputStream;
7 import java.io.FileInputStream;
8 import java.io.ObjectInputStream;
9
10
11 /**
12  * UniversityAllocationManager
13  * <p>
14  * Allocation Manager to hold staff, students and modules in a univers
15  * ity and their relationships.
16  *
17  * @author 660037119, 660047784
18  * @date 28/03/2017
19  */
20 public class UniversityAllocationManager implements AllocationManager,
21     Serializable
22 {
23     private ObjectArrayList students;
24     private ObjectArrayList staff;
25     private ObjectArrayList modules;
26
27     /**
28      * Constructor for the university allocation manager.
29      */
30     public UniversityAllocationManager() {
31         students = new ObjectArrayList();
32         staff = new ObjectArrayList();
33         modules = new ObjectArrayList();
34     }
35
36     /**
37      * Binary search current students stored via ID.
38      *
39      * @param ID student ID to search
40      *
41      * @returns UniversityStudent stored on system if existent, els
42      * e null.
43      */
44     private UniversityStudent binarySearchStudentID( String ID ){
45         int lowerBound = 0;
46         int upperBound = students.size() - 1;
47         String currentIndexValue;
48
49         while (upperBound >= lowerBound){
50             int currentIndex = (lowerBound + upperBound) / 2;
51             currentIndexValue = (String)( (Student) students.get(curre
52 ntIndex) ).getID();
```

```

50
51         if( currentIndexValue.compareTo(ID) == 0 ){
52             return (UniversityStudent)students.get(currentIndex);
53         }
54         if( currentIndexValue.compareTo(ID) < 0 ){
55             lowerBound = currentIndex + 1;
56         }
57         if( currentIndexValue.compareTo(ID) > 0 ){
58             upperBound = currentIndex - 1;
59         }
60     }
61     return null;
62 }
63
64 /**
65  * Binary search current staff stored via ID.
66  *
67  * @param ID staff ID to search
68  *
69  * @returns UniversityStaff stored on system if existent, else
null.
70  */
71 private UniversityStaff binarySearchStaffID( String ID ){
72     int lowerBound = 0;
73     int upperBound = staff.size() - 1;
74     String currentIndexValue;
75
76     while (upperBound >= lowerBound){
77         int currentIndex = (lowerBound + upperBound) /2;
78         currentIndexValue = (String)( (Staff) staff.get(currentInd
ex) ).getID();
79
80         if( currentIndexValue.compareTo(ID) == 0 ){
81             return (UniversityStaff)staff.get(currentIndex);
82         }
83         if( currentIndexValue.compareTo(ID) < 0 ){
84             lowerBound = currentIndex + 1;
85         }
86         if( currentIndexValue.compareTo(ID) > 0 ){
87             upperBound = currentIndex - 1;
88         }
89     }
90     return null;
91 }
92
93 /**
94  * Binary search current modules stored via code.
95  *
96  * @param code module code to search
97  *
98  * @returns UniversityModule stored on system if existent, else
null.
99  */

```

```

100     private UniversityModule binarySearchModuleCode( String code ){
101         int lowerBound = 0;
102         int upperBound = modules.size() - 1;
103         String currentIndexValue;
104
105         while (upperBound >= lowerBound){
106             int currentIndex = (lowerBound + upperBound) /2;
107             currentIndexValue = (String)( (Module) modules.get(current
Index) ).getCode();
108
109             if( currentIndexValue.compareTo(code) == 0 ){
110                 return (UniversityModule)modules.get(currentIndex);
111             }
112             if( currentIndexValue.compareTo(code) < 0 ){
113                 lowerBound = currentIndex + 1;
114             }
115             if( currentIndexValue.compareTo(code) > 0 ){
116                 upperBound = currentIndex - 1;
117             }
118         }
119         return null;
120     }
121
122     /**
123      * Binary insert a student into the manager in order of ID string
lexographical ordering.
124      *
125      * @param student student to insert
126      */
127     private void binaryInsertStudent( UniversityStudent student ) {
128
129         if (students.size() == 0) {
130             students.add(student);
131             return;
132         }
133         int lowerBound = 0;
134         int upperBound = students.size() - 1;
135         int currentIndex = 0;
136         String currentIndexValue;
137
138         while (true) {
139             currentIndex = (upperBound + lowerBound) / 2;
140             currentIndexValue = (String)( (Student)students.get(current
tIndex) ).getID();
141             if ( currentIndexValue.compareTo( student.getID() ) == 0 )
{
142                 break;
143             } else if ( currentIndexValue.compareTo( student.getID() )
< 0 ) {
144                 lowerBound = currentIndex + 1;
145                 if ( lowerBound > upperBound ) {
146                     currentIndex += 1;
147                     break;

```

```

148         }
149     } else {
150         upperBound = currentIndex - 1;
151         if ( lowerBound > upperBound )
152             break;
153     }
154 }
155
156 ObjectArrayList newArray = new ObjectArrayList();
157 for( int i=0; i<currentIndex; i++ ) {
158     newArray.add( students.get(i) );
159 }
160 newArray.add( student );
161 for( int i=currentIndex; i<students.size(); i++ ) {
162     newArray.add( students.get(i) );
163 }
164
165 students = newArray;
166 }
167
168 /**
169  * Binary insert a module into the manager in order of code str
170  * ing lexicographical ordering.
171  *
172  * @param module module to insert
173  */
174 private void binaryInsertModule( UniversityModule module ) {
175     if ( modules.size() == 0 ) {
176         modules.add(module);
177         return;
178     }
179     int lowerBound = 0;
180     int upperBound = modules.size() - 1;
181     int currentIndex = 0;
182     String currentIndexValue;
183
184     while (true) {
185         currentIndex = (upperBound + lowerBound) / 2;
186         currentIndexValue = (String)( (Module)modules.get(currentI
187 ndex) ).getCode();
188         if ( currentIndexValue.compareTo( module.getCode() ) == 0
189 ) {
190             break;
191         } else if ( currentIndexValue.compareTo( module.getCode()
192 ) < 0 ) {
193             lowerBound = currentIndex + 1;
194             if ( lowerBound > upperBound ) {
195                 currentIndex += 1;
196                 break;
197             }
198         } else {
199             upperBound = currentIndex - 1;

```

```

197         if ( lowerBound > upperBound )
198             break;
199     }
200 }
201
202 ObjectArrayList newArray = new ObjectArrayList();
203 for( int i=0; i<currentIndex; i++ ) {
204     newArray.add( modules.get(i) );
205 }
206 newArray.add( module );
207 for( int i=currentIndex; i<modules.size(); i++ ) {
208     newArray.add( modules.get(i) );
209 }
210
211 modules = newArray;
212 }
213
214 /**
215  * Binary insert a staff member into the manager in order of ID
string lexicographical ordering.
216  *
217  * @param staff staff to insert
218  */
219 private void binaryInsertStaff( UniversityStaff staff ) {
220
221     if (this.staff.size() == 0) {
222         this.staff.add(staff);
223         return;
224     }
225     int lowerBound = 0;
226     int upperBound = this.staff.size() - 1;
227     int currentIndex;
228     String currentIndexValue;
229
230     while (true) {
231         currentIndex = (upperBound + lowerBound) / 2;
232         currentIndexValue = (String)( (Staff)this.staff.get(current
tIndex) ).getID();
233         if ( currentIndexValue.compareTo( staff.getID() ) == 0 ) {
234             break;
235         } else if ( currentIndexValue.compareTo( staff.getID() ) <
0 ) {
236             lowerBound = currentIndex + 1;
237             if ( lowerBound > upperBound ) {
238                 currentIndex += 1;
239                 break;
240             }
241         } else {
242             upperBound = currentIndex - 1;
243             if ( lowerBound > upperBound )
244                 break;
245         }
246     }

```

```
247     ObjectArrayList newArray = new ObjectArrayList();
248     for( int i=0; i<currentIndex; i++ ) {
249         newArray.add( this.staff.get(i) );
250     }
251     newArray.add( staff );
252     for( int i=currentIndex; i<this.staff.size(); i++ ) {
253         newArray.add( this.staff.get(i) );
254     }
255
256     this.staff = newArray;
257 }
258
259
260
261
262
263 /**
264  * @inheritDoc
265  */
266 public String addStudent(String forename, String surname, byte sta
267 ge) throws InvalidStageException {
268     UniversityStudent student = new UniversityStudent( forename, s
269 urname, stage );
270     String studentID = generateStudentID();
271
272     try {
273         student.setID( studentID );
274     }
275     catch( InvalidIDException e ) {
276         e.printStackTrace();
277     }
278     catch( IDAlreadySetException e ) {
279         e.printStackTrace();
280     }
281
282     binaryInsertStudent( student );
283     return student.getID();
284 }
285
286 /**
287  * @inheritDoc
288  */
289 public void addStudent(Student student) throws IDAlreadySetExcepti
290 on {
291     UniversityStudent uniStudent = null;
292     try {
293         uniStudent = new UniversityStudent( student.getForename(),
294 student.getSurname(), student.getStage() );
295     }
296     catch( InvalidStageException e ) {
297         e.printStackTrace();
298     }
```



```

296     }
297
298     if( student.getID() == null ) {
299         try {
300             String studentID = generateStudentID();
301             uniStudent.setID( studentID );
302             student.setID( studentID );
303         }
304         catch( InvalidIDException e ) {
305             e.printStackTrace();
306         }
307     }
308     else {
309         throw new IDAlreadySetException( "Student already has ID:
" + student.getID() + " and so cannot be added to the system." );
310     }
311
312     binaryInsertStudent( uniStudent );
313 }
314
315
316 /**
317  * Generate a random student ID
318  *
319  * @returns random student ID
320  */
321 private String generateStudentID() {
322     String num1 = Integer.toString((int)(Math.random() * 99999 + 0
323 ));
324     String num2 = Integer.toString((int)(Math.random() * 99999 + 1
325 ));
326     String ID = num1 + num2;
327
328     int zeros = 10 - ID.length();
329     while( zeros > 0 ) {
330         ID = "0" + ID;
331         zeros--;
332     }
333
334     if( binarySearchStudentID(ID) != null ) {
335         //try again if the random ID generated was already an exis
336         tent in manager
337         return generateStudentID();
338     }
339
340     assert ID != null;
341     return ID;
342 }
343
344 /**
345  * @inheritDoc
346  */
347 public String addModule(String name, byte credits, byte stage, int

```

```

344     capacity, Staff[] staff)
345     throws InvalidStageException, InvalidCreditsException, InvalidCapa
cityException,
346     DuplicateStaffException, StaffNotInSystemException {
347
348         if( capacity < 1 ) {
349             throw new InvalidCapacityException( "Attempted to add modu
le " + name + " with invalid capacity less than 1. " );
350         }
351         for( int i=0; i<staff.length; i++ ) {
352             for( int j=i+1; j<staff.length; j++ ) {
353                 if( staff[i].equals(staff[j]) ) {
354                     assert i != j;
355                     throw new DuplicateStaffException( "Duplicate staf
f in array, therefore cannot add to module" );
356                 }
357             }
358         }
359
360
361         UniversityModule module = null;
362         String moduleCode = generateModuleCode();
363
364         for( Staff staffToAdd : staff ) {
365             if( binarySearchStaffID(staffToAdd.getID()) == null ) {
366                 throw new StaffNotInSystemException( "Attempted to add
staff member '" + staffToAdd.getForename() + " " + staffToAdd.getSurn
ame() +
367                                     "' who does not e
xist in the system onto module: " + name );
368             }
369         }
370
371         try {
372             module = new UniversityModule( name, credits, stage, capac
ity, moduleCode );
373         }
374         catch( InvalidIDException e ) {
375             e.printStackTrace();
376         }
377         catch( IDAlreadySetException e ) {
378             e.printStackTrace();
379         }
380
381
382         for( Staff staffToAdd : staff ) {
383             UniversityStaff staffInSystem = binarySearchStaffID( staff
ToAdd.getID() );
384             try {
385                 module.addStaff( staffInSystem );
386             }
387             catch( IDNotSetException e ) {
388                 e.printStackTrace();

```

```
389         }
390         catch( ModuleDiscontinuedException e ) {
391             e.printStackTrace();
392         }
393         staffInSystem.addTeachingModule( module );
394     }
395
396     binaryInsertModule( module );
397     return module.getCode();
398 }
399
400 /**
401  * @inheritDoc
402  */
403 public void addModule(Module module) throws IDAlreadySetException
404 {
405     UniversityModule uniModule = null;
406     try {
407         uniModule = new UniversityModule( module.getName(), module
408 .getCredits(), module.getStage(), module.getCapacity() );
409     }
410     catch( InvalidStageException e ) {
411         e.printStackTrace();
412     }
413     catch( InvalidCreditsException e ) {
414         e.printStackTrace();
415     }
416
417     if( module.getCode() == null ) {
418         try {
419             String moduleCode = generateModuleCode();
420             uniModule.setCode( moduleCode );
421             module.setCode( moduleCode );
422         }
423         catch( InvalidIDException e ) {
424             e.printStackTrace();
425         }
426     }
427     else {
428         throw new IDAlreadySetException( "Module already has code:
429 " + module.getCode() + " and so cannot be added to the system." );
430     }
431
432     binaryInsertModule( uniModule );
433 }
434
435 /**
436  * Generate a random module code
437  *
438  * @returns random module code
439  */
440 public String generateModuleCode() {
441     String code = Integer.toString((int)(Math.random() * 99999 + 1
```

```
438 ));
439
440     int zeros = 5 - code.length();
441     while( zeros > 0 ) {
442         code = "0" + code;
443         zeros--;
444     }
445
446     if( binarySearchModuleCode(code) != null ) {
447         //try again
448         return generateModuleCode();
449     }
450
451     assert code != null;
452     return code;
453 }
454
455 /**
456  * @inheritDoc
457  */
458 public String addStaff(String forename, String surname) {
459     UniversityStaff staff = new UniversityStaff( forename, surname
460 );
461
462     String staffID = generateStaffID();
463     try {
464         staff.setID( staffID );
465     }
466     catch( InvalidIDException e ) {
467         e.printStackTrace();
468     }
469     catch( IDAlreadySetException e ) {
470         e.printStackTrace();
471     }
472
473     binaryInsertStaff( staff );
474     return staff.getID();
475 }
476
477 /**
478  * @inheritDoc
479  */
480 public void addStaff(Staff staff) throws IDAlreadySetException {
481     UniversityStaff uniStaff = new UniversityStaff( staff.getForen
482 ame(), staff.getSurname() );
483
484     if( staff.getID() == null ) {
485         try {
486             String staffID = generateStaffID();
487             uniStaff.setID( staffID );
488             staff.setID( staffID );
489         }
```

```

489         catch( InvalidIDException e ) {
490             e.printStackTrace();
491         }
492     }
493     else {
494         throw new IDAlreadySetException( "Staff member already has
ID: " + staff.getID() + " and so cannot be added to the system." );
495     }
496
497     binaryInsertStaff( uniStaff );
498 }
499
500 /**
501  * Generate a random staff ID
502  *
503  * @returns random staff ID
504  */
505 public String generateStaffID(){
506     String ID = Integer.toHexString((int)(Math.random() * 1048575
+ 0));
507
508     int zeros = 5 - ID.length();
509     while( zeros > 0 ) {
510         ID = "0" + ID;
511         zeros--;
512     }
513
514     if( binarySearchStaffID(ID) != null ){
515         //try again
516         return generateStaffID();
517     }
518
519     assert ID != null;
520     return ID;
521 }
522
523 /**
524  * @inheritDoc
525  */
526 public void discontinue(String moduleCode) throws InvalidIDExcepti
on,
527     IDNotRecognisedException {
528     UniversityModule.checkValidCode(moduleCode);
529     UniversityModule module = binarySearchModuleCode(moduleCode);
530
531     if( module == null ) {
532         throw new IDNotRecognisedException( "Module code: " + modu
leCode + " not found in the system." );
533     }
534     Student[] students = module.getStudents();
535
536     for( int i=0; i<students.length; i++ ) {
537         UniversityStudent student = ( (UniversityStudent) students

```

```

537 [i] );
538         student.removeModule( module );
539     }
540
541     Staff[] staff = module.getTeachingStaff();
542     for( Staff staffMember : staff ) {
543         ( (UniversityStaff) staffMember ).removeTeachingModule( mo
544         dule );
545     }
546     module.discontinue();
547 }
548 /**
549  * @inheritDoc
550  */
551 public void enrol(String studentID, String moduleCode) throws Inva
552 lidIDException,
553 IDNotRecognisedException, ModuleAtCapacityException,
554 InsufficientAvailableCreditsException, ModuleDiscontinuedException
555 ,
556 ModuleStageTooHighException, EnrollingWouldPreventHonoursException
557 {
558     UniversityStudent.checkValidID( studentID );
559     UniversityModule.checkValidCode( moduleCode );
560
561     UniversityStudent student = binarySearchStudentID(studentID);
562     UniversityModule module = binarySearchModuleCode(moduleCode);
563
564     if( student == null || module == null ) {
565         throw new IDNotRecognisedException( "The student ID : " +
566 studentID + " or the module code: " + moduleCode +
567 " does not reference a student or module on the system.");
568     }
569
570     if( module.isAtCapacity() ) {
571         throw new ModuleAtCapacityException( "The module with code
572 : " + moduleCode + " is at capacity and cannot enrol students" );
573     }
574
575     if( student.getTotalCredits() + module.getCredits() > 120 ) {
576         throw new InsufficientAvailableCreditsException( "The stud
577 ent with ID: " + studentID + " does not have enough credits left to en
578 rol onto module with code : "
579 + moduleCode );
580     }
581
582     if( module.isDiscontinued() ) {
583         throw new ModuleDiscontinuedException( "The module with co
584 de: " + moduleCode + " is discontinued and so cannot enrol students" )
585 ;
586     }
587 }

```

```

580         if( module.getStage() > student.getStage() ) {
581             throw new ModuleStageTooHighException( "The module with co
de: " + moduleCode + " is of a higher stage than student with ID: " +
studentID );
582         }
583
584         if( module.getStage() != student.getStage() ) {
585             if( module.getCredits() + student.getLowerStageCredits() >
30 ) {
586                 throw new EnrollingWouldPreventHonoursException( "Enro
lling student with ID: " + studentID + " to module with code: " + modu
leCode
587                     + "would give the student more than 30 credits at a lo
wer stage, preventing honours.");
588             }
589         }
590
591         student.assignModule( module );
592         module.addStudent( student );
593     }
594
595     /**
596      * @inheritDoc
597      */
598     public void loadAllocationManager(String filename) throws IOExcept
ion,
599     ClassNotFoundException {
600         UniversityAllocationManager manager = null ;
601         FileInputStream fileInput = null ;
602         ObjectInputStream objectInput = null ;
603         fileInput = new FileInputStream ( filename );
604         objectInput = new ObjectInputStream ( fileInput );
605         manager = (UniversityAllocationManager)objectInput.readObject(
);
606         objectInput.close ();
607     }
608
609     /**
610      * @inheritDoc
611      */
612     public int getNumberOfStaff() {
613         return staff.size();
614     }
615
616     /**
617      * @inheritDoc
618      */
619     public int getNumberOfStudents() {
620         return students.size();
621     }
622
623     /**
624      * @inheritDoc

```

```

625     */
626     public int getNumberOfModules() {
627         return modules.size();
628     }
629
630     /**
631      * @inheritDoc
632      */
633     public Staff[] getStaff() {
634         Staff[] staff = new Staff[this.staff.size()];
635
636         for( int i=0; i<this.staff.size(); i++ ) {
637             staff[i] = (Staff)this.staff.get(i);
638         }
639         return staff;
640     }
641
642     /**
643      * @inheritDoc
644      */
645     public Staff[] getStaff(String moduleCode) throws InvalidIDExcepti
on,
646     IDNotRecognisedException {
647
648         UniversityModule.checkValidCode( moduleCode );
649         UniversityModule module = binarySearchModuleCode(moduleCode);
650
651         if( module == null ) {
652             throw new IDNotRecognisedException( "The module code " + m
oduleCode + " does not exist on the system.");
653         }
654         return module.getTeachingStaff();
655     }
656
657     /**
658      * @inheritDoc
659      */
660     public Student[] getStudents() {
661         Student[] students = new Student[this.students.size()];
662
663         for( int i=0; i<this.students.size(); i++ ) {
664             students[i] = (Student)this.students.get(i);
665         }
666         return students;
667     }
668
669     /**
670      * @inheritDoc
671      */
672     public Student[] getStudents(String moduleCode) throws InvalidIDEx
ception,
673     IDNotRecognisedException {
674

```



```
675         UniversityModule.checkValidCode( moduleCode );
676         UniversityModule module = binarySearchModuleCode(moduleCode);
677
678         if( module == null ) {
679             throw new IDNotRecognisedException( "The module code " + moduleCode + " does not exist on the system.");
680         }
681         return module.getStudents();
682     }
683
684     /**
685      * @inheritDoc
686      */
687     public Module[] getModules() {
688         Module[] modules = new Module[this.modules.size()];
689
690         for( int i=0; i<this.modules.size(); i++ ) {
691             modules[i] = (Module)this.modules.get(i);
692         }
693         return modules;
694     }
695
696     /**
697      * @inheritDoc
698      */
699     public Module[] getRunningModules() {
700
701         ObjectArrayList modules = new ObjectArrayList();
702
703         for( int i=0; i<this.modules.size(); i++ ) {
704             Module module = (Module)this.modules.get(i);
705
706             if( !module.isDiscontinued() ) {
707                 modules.add( module );
708             }
709         }
710         Module[] runningModules = new Module[modules.size()];
711
712         for( int i=0; i<modules.size(); i++ ) {
713             runningModules[i] = (Module)modules.get(i);
714         }
715         return runningModules;
716     }
717
718     /**
719      * @inheritDoc
720      */
721     public Module[] getAvailableModules() {
722         ObjectArrayList modules = new ObjectArrayList();
723
724         for( int i=0; i<this.modules.size(); i++ ) {
725             UniversityModule module = (UniversityModule)this.modules.get(i);
```

```
726         if( !module.isDiscontinued() && !module.isAtCapacity() ) {
727             modules.add( module );
728         }
729     }
730     Module[] availableModules = new Module[modules.size()];
731     for( int i=0; i<modules.size(); i++ ) {
732         availableModules[i] = (Module)modules.get(i);
733     }
734     return availableModules;
735 }
736
737 /**
738  * @inheritDoc
739  */
740 public Module[] getModules(String studentID) throws InvalidIDException,
741 IDNotRecognisedException {
742     UniversityStudent.checkValidID( studentID );
743     UniversityStudent student = binarySearchStudentID(studentID);
744     if( student == null ) {
745         throw new IDNotRecognisedException( "The student ID : " +
746 studentID + " does not exist on the system.");
747     }
748     return student.getEnrolledModules();
749 }
750
751 /**
752  * @inheritDoc
753  */
754 public int getNumberOfFullyAllocatedStudents() {
755     int numberFullStudents = 0;
756     for( int i=0; i<students.size(); i++ ) {
757         UniversityStudent student = (UniversityStudent)students.get(i);
758         if( student.getTotalCredits() == 120 ) {
759             numberFullStudents++;
760         }
761     }
762     return numberFullStudents;
763 }
764
765 /**
766  * @inheritDoc
767  */
768 public int getNumberOfModulesAtCapacity() {
769     int numberFullModules = 0;
```

```

776         for( int i=0; i<modules.size(); i++ ) {
777             UniversityModule module = (UniversityModule)modules.get(i)
778         ;
779
780             if( module.isAtCapacity() ) {
781                 numberFullModules++;
782             }
783         }
784         return numberFullModules;
785     }
786
787     /**
788     * @inheritDoc
789     */
790     public Module[] remove(Staff staff) throws InvalidIDException,
791     IDNotRecognisedException, IDNotSetException {
792
793         if( staff.getID() == null ) {
794             throw new IDNotSetException( "Staff member with name " + s
795 taff.getForename() + staff.getSurname() + " has no ID set and so cannot
796 be removed." );
797         }
798         UniversityStaff.checkValidID( staff.getID() );
799         UniversityStaff staffToRemove = binarySearchStaffID( staff.getID() );
800         ObjectArrayList nowNoStaff = new ObjectArrayList();
801
802         for( Module module : staffToRemove.getTeachingModules() ) {
803             try {
804                 module.removeStaff( staffToRemove );
805             }
806             catch( StaffNotInvolvedException e ) {
807                 e.printStackTrace();
808             }
809
810             if( ((UniversityModule)module).getTeachingStaff().length =
811 = 0 ) {
812                 nowNoStaff.add( module );
813             }
814         }
815         Module[] noStaff = new Module[nowNoStaff.size()];
816
817         for( int i=0; i<nowNoStaff.size(); i++ ) {
818             noStaff[i] = (Module)nowNoStaff.get(i);
819         }
820         this.staff.remove( staffToRemove );
821         return noStaff;
822     }
823     /**

```

```
824      * @inheritDoc
825      */
826      public void remove(Student student) throws InvalidIDException,
827      IDNotRecognisedException, IDNotSetException {
828
829          if( student.getID() == null ) {
830              throw new IDNotSetException( "Student with name " + studen
831              t.getForename() + student.getSurname() + " has no ID set, and so canno
832              t be removed." );
833          }
834
835          UniversityStudent studentToRemove = binarySearchStudentID( stu
836          dent.getID() );
837          if( studentToRemove == null ) {
838              throw new IDNotRecognisedException( "The student ID : " +
839              studentToRemove.getID() + " does not exist on the system." );
840          }
841
842          for( Module module : studentToRemove.getEnrolledModules() ) {
843              unEnrol( studentToRemove.getID(), module.getCode() );
844          }
845
846          students.remove( studentToRemove );
847      }
848
849      /**
850       * @inheritDoc
851       */
852      public void saveAllocationManager(String filename) throws IOExcept
853      ion {
854          FileOutputStream fileOutput = null;
855          ObjectOutputStream objectOutput = null;
856          fileOutput = new FileOutputStream(filename);
857          objectOutput = new ObjectOutputStream(fileOutput);
858          objectOutput.writeObject(this);
859          objectOutput.close ();
860      }
861
862      /**
863       * @inheritDoc
864       */
865      public boolean unEnrol(String studentID, String moduleCode) throws
866      InvalidIDException, IDNotRecognisedException {
867
868          UniversityStudent.checkValidID(studentID);
869          UniversityModule.checkValidCode(moduleCode);
870
871          UniversityStudent student = binarySearchStudentID(studentID);
872          UniversityModule module = binarySearchModuleCode(moduleCode);
873
874          if( student == null || module == null ) {
875              throw new IDNotRecognisedException( "The student ID : " +
876              studentID + " or the module code: " + moduleCode +
```

```
871         " does not reference a student or module on the system.");
872     }
873     for( Module enrolledModule : student.getEnrolledModules() ) {
874         if( enrolledModule.getCode().equals(module.getCode()) ) {
875             ((UniversityModule)enrolledModule).removeStudent( stud
876 ent );
877             student.removeModule( (UniversityModule)enrolledModule
878 );
879             return true;
880         }
881     }
882     return false;
883 }
884 }
885
```

```
1 package university;
2
3 /**
4  * UniversityAllocationManager
5  * <p>
6  * Allocation Manager to hold staff, students and modules in a univers
7  * ity and their relationships.
8  *
9  * @author 660037119, 660047784
10 * @date 28/03/2017
11 */
12 public class UniversityStudent implements Student
13 {
14     private String forename;
15     private String surname;
16     private byte stage;
17     private ObjectArrayList modules;
18     private byte sameStageCredits;
19     private byte lowerStageCredits;
20     private byte totalCredits;
21     private String id;
22
23     /**
24      * Constructor for University Student without ID.
25      *
26      * @param forename forename of student
27      * @param surname surname of student
28      * @param stage stage of student
29      * @throws InvalidStageException if the student has a stage out of
30      * range 1-4 inclusive
31      */
32     public UniversityStudent( String forename, String surname, byte st
33     age ) throws InvalidStageException {
34         this.forename = forename;
35         this.surname = surname;
36         setStage( stage );
37         modules = new ObjectArrayList();
38     }
39
40     /**
41      * Constructor for University Student without ID.
42      *
43      * @param forename forename of student
44      * @param surname surname of student
45      * @param stage stage of student
46      * @param id id of the student
47      * @throws InvalidStageException if the student has a stage out of
48      * range 1-4 inclusive
49      */
50     public UniversityStudent( String forename, String surname, byte st
51     age, String id ) throws InvalidStageException, InvalidIDException, IDA
52     lreadySetException {
```

```
48         this.forename = forename;
49         this.surname = surname;
50         setStage( stage );
51         setID( id );
52         modules = new ObjectArrayList();
53     }
54
55     /**
56      * Check an ID is of correct student form.
57      *
58      * @param id id to be checked
59      * @throws InvalidIDException if the ID is of incorrect form
60      */
61     public static void checkValidID(String id) throws InvalidIDException {
62         boolean valid = true;
63
64         if( id.length() != 10 ) valid = false;
65
66         try {
67             Long.parseLong( id );
68         }
69         catch( NumberFormatException e ) {
70             valid = false;
71         }
72
73         if( !valid ) {
74             throw new InvalidIDException("Student ID provided must be
75             10 digits.");
76         }
77
78     /**
79      * @inheritDoc
80      */
81     public byte getStage() {
82         return stage;
83     }
84
85     /**
86      * Set the stage of a student.
87      *
88      * @param stage stage of student to be set
89      * @throws InvalidStageException if the stage is out of range 1-4
90      inclusive.
91      */
92     private void setStage( byte stage ) throws InvalidStageException {
93         if( stage < 1 || stage > 4 ) {
94             throw new InvalidStageException( "Stage for student '" + f
95             orename + surname + "' must be between 1 and 4 inclusive." );
96         }
97         else {
98             this.stage = stage;
99         }
100     }
```

```
97     }
98 }
99
100 /**
101  * @inheritDoc
102  */
103 public String getForename() {
104     return forename;
105 }
106
107 /**
108  * @inheritDoc
109  */
110 public String getSurname() {
111     return surname;
112 }
113
114 /**
115  * @inheritDoc
116  */
117 public String getID() {
118     return id;
119 }
120
121 /**
122  * Return the credits the student has from modules enrolled at
the same stage as them
123  *
124  * @returns credits at the same stage
125  */
126 public byte getSameStageCredits() {
127     return sameStageCredits;
128 }
129
130 /**
131  * Return the credits the student has from modules enrolled at
a lower stage than them
132  *
133  * @returns credits at a lower stage
134  */
135 public byte getLowerStageCredits() {
136     return lowerStageCredits;
137 }
138
139 /**
140  * Return the credits the student has from modules enrolled at
a higher stage
141  *
142  * @returns credits at a higher stage
143  */
144 public byte getTotalCredits() {
145     return totalCredits;
146 }
```



```
147
148     /**
149      * @inheritDoc
150      */
151     public void setID( String id ) throws IDAlreadySetException, InvalidIDException {
152         if( this.id == null) {
153             checkValidID(id);
154             this.id = id;
155         }
156         else {
157             throw new IDAlreadySetException ("Student '" + forename +
158 " " + surname + "' already has ID: " + this.id);
159         }
160     }
161
162     /**
163      * Return the modules this student is currently enrolled on.
164      *
165      * @returns modules enrolled
166      */
167     public Module[] getEnrolledModules() {
168         Module[] modules = new Module[this.modules.size()];
169
170         for( int i = 0; i < this.modules.size(); i++ ) {
171             modules[i] = (Module)this.modules.get(i);
172         }
173         return modules;
174     }
175
176     /**
177      * Assigns and links the student to a module.
178      *
179      * @param module module to be assigned
180      */
181     public void assignModule( UniversityModule module ) {
182         modules.add( module );
183
184         if( module.getStage() == stage ) {
185             sameStageCredits += module.getCredits();
186         }
187         else {
188             lowerStageCredits += module.getCredits();
189         }
190         totalCredits += module.getCredits();
191     }
192
193     /**
194      * Removes and unlinks the student from a module.
195      *
196      * @param module module to be assigned
197      */
198     public void removeModule( UniversityModule module ) {
```

```
198         modules.remove( module );
199
200         if( module.getStage() == stage ) {
201             sameStageCredits -= module.getCredits();
202         }
203         else {
204             lowerStageCredits -= module.getCredits();
205         }
206         totalCredits -= module.getCredits();
207     }
208 }
209
```

```
1 package university;
2
3
4 /**
5  * @author 660047784, 660037119
6  * @date 28/03/2017
7  */
8 public class UniversityStaff implements Staff
9 {
10     private String forename;
11     private String surname;
12     private String id;
13     private ObjectArrayList teachingModules;
14
15     /**
16      * Constructor for UniversityStaff without ID
17      *
18      * @param forename Forename of staff member
19      * @param surname Surname of staff member
20      */
21     public UniversityStaff( String forename, String surname ) {
22         this.forename = forename;
23         this.surname = surname;
24         teachingModules = new ObjectArrayList();
25     }
26
27     /**
28      * Constructor for UniversityStaff with ID
29      *
30      * @param forename Forename of staff member
31      * @param surname Surname of staff member
32      * @param id Staff member ID
33      *
34      * @throws IDAlreadySetException if there is already a staff member with that id in the system
35      * @throws InvalidIdException if the format of the id is invalid
36      */
37     public UniversityStaff( String forename, String surname, String id ) throws IDAlreadySetException, InvalidIDException {
38         this.forename = forename;
39         this.surname = surname;
40         setID( id );
41         teachingModules = new ObjectArrayList();
42     }
43
44     /**
45      * Checks if the id follows the valid conventions
46      *
47      * @throws InvalidIDException if id is invalid
48      */
49     public static void checkValidID( String id ) throws InvalidIDException {
50         boolean valid = true;
```

```
51
52     //length of id must be 5 characters
53     if( id.length() != 5 ) valid = false;
54
55     //if the id can pass as a hex value
56     try {
57         Integer.parseInt( id, 16 );
58     }
59     catch( NumberFormatException e ) {
60         valid = false;
61     }
62
63     if( !valid ) {
64         throw new InvalidIDException( "ID provided must be 5 hex c
65         }
66     }
67
68     /**
69     * @inheritDoc
70     */
71     public String getForename() {
72         return forename;
73     }
74
75     /**
76     * @inheritDoc
77     */
78     public String getSurname() {
79         return surname;
80     }
81
82     /**
83     * @inheritDoc
84     */
85     public String getID() {
86         return id;
87     }
88
89     /**
90     * @inheritDoc
91     */
92     public void setID( String id ) throws IDAlreadySetException, InvalidIDException {
93         //if the staff member doesn't have an id
94         if( this.id == null ) {
95             checkValidID(id);
96             this.id = id;
97         }
98         else {
99             throw new IDAlreadySetException( "Staff member '" + forename + " " + surname + "' already has ID: " + id );
100         }
    }
```

```
101     }
102
103     /**
104      * Links a module to the staff member.
105      * @param module module to be linked
106      */
107     public void addTeachingModule( UniversityModule module ) {
108         teachingModules.add( module );
109     }
110
111     /**
112      * Remove link to a module.
113      * @param module module to be removed.
114      */
115     public void removeTeachingModule( UniversityModule module ) {
116         teachingModules.remove( module );
117     }
118
119     /**
120      * Get all modules the staff is linked and teaching on.
121      *
122      * @returns modules linked
123      */
124     public Module[] getTeachingModules() {
125         Module[] modules = new Module[teachingModules.size()];
126
127         //convert teachingModules into a module array
128         for( int i = 0; i < teachingModules.size(); i++ ) {
129             modules[i] = (Module)teachingModules.get(i);
130         }
131         return modules;
132     }
133 }
134
```

```
1 package university;
2
3 /**
4  *
5  *
6  * @author 660047784, 660037119
7  * @date 28/03/2016
8  */
9 public class UniversityModule implements Module
10 {
11     private String name;
12     private String code;
13     private byte credits;
14     private byte stage;
15     private ObjectArrayList teachingStaff;
16     private ObjectArrayList students;
17     private int capacity;
18     private int enrolled;
19     private boolean discontinued;
20
21     /**
22      * Constructor for UniversityModule without module code
23      *
24      * @param name Name of the module
25      * @param credits The number of credits that the module gives
26      * @param stage What stage the module can be taken at
27      * @param capacity The maximum number of students that can take th
28      e module
29      */
30     public UniversityModule( String name, byte credits, byte stage, in
31 t capacity ) throws InvalidStageException, InvalidCreditsException {
32         this.name = name;
33         setCredits( credits );
34         setStage( stage );
35         teachingStaff = new ObjectArrayList();
36         students = new ObjectArrayList();
37         this.capacity = capacity;
38         discontinued = false;
39     }
40
41     /**
42      * Constructor for UniversityModule with module code
43      *
44      * @param name Name of the module
45      * @param credits The number of credits that the module gives
46      * @param stage What stage the module can be taken at
47      * @param capacity The maximum number of students that can take th
48      e module
49      * @param code The module code
50      */
51     public UniversityModule( String name, byte credits, byte stage, in
52 t capacity, String code ) throws InvalidStageException, InvalidCredits
53 Exception, InvalidIDException,
```

```
49         IDAlreadySetException {
50             this.name = name;
51             setCredits( credits );
52             setStage( stage );
53             teachingStaff = new ObjectArrayList();
54             students = new ObjectArrayList();
55             this.capacity = capacity;
56             discontinued = false;
57             setCode( code );
58         }
59
60         /**
61          * Checks that the format of the module code is correct
62          *
63          * @param code The module code to check
64          * @return true if the module code is valid, else false
65          */
66         public static boolean checkValidCode( String code ) throws Invalid
IDException {
67             boolean valid = true;
68
69             //if the code isn't 5 characters
70             if( code.length() != 5 ) valid = false;
71
72             //if the code passes as an integer
73             try {
74                 Integer.parseInt( code );
75             }
76             catch( NumberFormatException e ) {
77                 valid = false;
78             }
79             if( !valid ) {
80                 throw new InvalidIDException("Module code provided must be
5 digits.");
81             }
82             return valid;
83         }
84
85         /**
86          * @inheritDoc
87          */
88         public String getCode(){
89             return code;
90         }
91
92         /**
93          * @inheritDoc
94          */
95         public void setCode(String code) throws InvalidIDException, IDAlre
adySetException {
96             //if the module doesn't have a module code
97             if( this.code == null) {
98                 checkValidCode(code);
```

```
99         this.code = code;
100     }
101     else { throw new IDAlreadySetException ("This module already h
as a code (" + this.code + " )" );
102     }
103 }
104
105 /**
106  * @inheritDoc
107  */
108 public String getName(){
109     return name;
110 }
111
112 /**
113  * @inheritDoc
114  */
115 public byte getStage(){
116     return stage;
117 }
118
119 /**
120  * Set the stage of the module.
121  *
122  * @param stage stage to be set.
123  * @throws InvalidStageException if the stage is not 1-4 inclusive
124  *
125  */
126 private void setStage( byte stage ) throws InvalidStageException {
127     //check the stage is between 1 and 4
128     if( stage < 1 || stage > 4 ) {
129         throw new InvalidStageException( "Stage for module '" + na
me + "' must be between 1 and 4 inclusive." );
130     }
131     else {
132         this.stage = stage;
133     }
134 }
135
136 /**
137  * Set the credits of the module.
138  *
139  * @param credits credits to be set.
140  * @throws InvalidCreditsException if the credits are not between
0-120 inclusive.
141  */
142 private void setCredits( byte credits ) throws InvalidCreditsExcep
tion {
143     //check that the number of credits is between 0 and 120
144     if( credits < 0 || credits > 120 ) {
145         throw new InvalidCreditsException( "Credits for module '"
+ name + "' must be between 0 and 120 inclusive." );
146     }
147 }
```



```
146         else {
147             this.credits = credits;
148         }
149     }
150
151     /**
152     * Returns the credits valued by the module
153     *
154     * @returns the credits of the module
155     */
156     public byte getCredits() {
157         return credits;
158     }
159
160     /**
161     * @inheritDoc
162     */
163     public Staff[] getTeachingStaff(){
164         Staff[] teachingStaff = new Staff[this.teachingStaff.size()];
165
166         //convert staff stored by module to a staff array
167         for( int i = 0; i < this.teachingStaff.size(); i++ ) {
168             teachingStaff[i] = (Staff)this.teachingStaff.get(i);
169         }
170         return teachingStaff;
171     }
172
173     /**
174     * @inheritDoc
175     */
176     public int getCapacity() {
177         return capacity;
178     }
179
180     /**
181     * Checks if the module has reached capacity
182     *
183     * @return true if module is at maximum capacity, false otherwise
184     */
185     public boolean isAtCapacity() {
186         if( enrolled == capacity ){
187             return true;
188         } else {
189             return false;
190         }
191     }
192
193     /**
194     * Discontinue the module, removing all links.
195     */
196     public void discontinue(){
197         teachingStaff = new ObjectArrayList();
198         students = new ObjectArrayList();
199     }
```

```

199         enrolled = 0;
200         discontinued = true;
201     }
202
203     /**
204      * @inheritDoc
205      */
206     public boolean isDiscontinued(){
207         return discontinued;
208     }
209
210     /**
211      * @inheritDoc
212      */
213     public void addStaff(Staff[] staff) throws DuplicateStaffException
, IDNotSetException, ModuleDiscontinuedException{
214
215         if( discontinued ) {
216             throw new ModuleDiscontinuedException( "Module " + name +
" is discontinued, and so cannot accept more staff." );
217         }
218         for( Staff staffToAdd : staff ){
219             //if the staff member doesn't have an id, throw IDNotSetEx
ception
220             if( staffToAdd.getID() == null ) {
221                 throw new IDNotSetException( "Attempted to add a staff
member '" + staffToAdd.getForename() + " " + staffToAdd.getSurname()
+
222                 "' whom has no ID set into module: " + name );
223             }
224             for( int i=0; i < teachingStaff.size(); i++ ) {
225                 //if there is a staff member with the same id in the m
odule
226                 if ( staffToAdd.equals(teachingStaff.get(i)) ) {
227                     throw new DuplicateStaffException( "Attempted to a
dd a duplicate staff member with ID: " + staffToAdd.getID() + ", in mo
dule " + name );
228                 }
229             }
230             teachingStaff.add( staffToAdd );
231         }
232     }
233
234     /**
235      * @inheritDoc
236      */
237     public void addStaff(Staff staff) throws DuplicateStaffException,
IDNotSetException, ModuleDiscontinuedException{
238         if( discontinued ) {
239             throw new ModuleDiscontinuedException( "Module '" + name +
"' is discontinued, and so cannot accept more staff." );
240         }
241         //if the staff member doesn't have an id, throw IDNotSetExcept

```

```

241 ion
242     if( staff.getID() == null ) {
243         throw new IDNotSetException( "Attempted to add a staff
244         member '" + staff.getForename() + " " + staff.getSurname() +
245         "'' whom has no ID set into module: " + name );
246     }
247     for( int i=0; i < teachingStaff.size(); i++ ) {
248         //if there is a staff member with the same id in the modul
249         e
250             if( staff.equals(teachingStaff.get(i)) ) {
251                 throw new DuplicateStaffException( "Attempted to add a
252                 duplicate staff member with ID: " + staff.getID() + ", in module " +
253                 name );
254             }
255         }
256         teachingStaff.add( staff );
257     }
258     /**
259     * @inheritDoc
260     */
261     public void removeStaff(Staff staff) throws StaffNotInvolvedExcept
262     ion{
263         boolean staffFound = teachingStaff.remove(staff);
264
265         if( !staffFound ) {
266             throw new StaffNotInvolvedException( "Staff member with ID
267             : " + staff.getID() + " is not found in module: " + name + ", therefor
268             e cannot be removed" );
269         }
270     }
271
272     /**
273     * Allocates a student to the module array
274     *
275     * @param student Student object to add to module
276     */
277     public void addStudent( UniversityStudent student ) {
278         students.add( student );
279         enrolled++;
280     }
281
282     /**
283     * Removes a student from the module
284     *
285     * @param student Student to remove from module
286     */
287     public void removeStudent( UniversityStudent student ) {
288         students.remove( student );
289         enrolled--;
290     }
291
292     /**

```

```
287      * Gets all the students enrolled onto the module
288      *
289      * @return all students enrolled onto the module as a student array
290      */
291      public Student[] getStudents() {
292          Student[] students = new Student[this.students.size()];
293
294          //convert students stored by module to a Student array
295          for( int i = 0; i < this.students.size(); i++ ) {
296              students[i] = (Student)this.students.get(i);
297          }
298          return students;
299      }
300  }
301
302
```