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

```
package university;

import java.io.Serializable;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;


/**
 * UniversityAllocationManager
 * <p>
 * Allocation Manager to hold staff, students and modules in a university and their relationships.
 *
 * @author 660037119, 660047784
 * @date 28/03/2017
 */

public class UniversityAllocationManager implements AllocationManager, Serializable
{
private ObjectArrayList students;
private ObjectArrayList staff;
private ObjectArrayList modules;

    /**
        * Constructor for the university allocation manager.
     */
    public UniversityAllocationManager() {
        students = new ObjectArrayList();
        staff = new ObjectArrayList();
        modules = new ObjectArrayList();
    }

    /**
        * Binary search current students stored via ID.
        *
        * @param ID student ID to search
        *
        * @returns UniversityStudent stored on system if existent, else null.
     */
    private UniversityStudent binarySearchStudentID( String ID ){
        int lowerBound = 0;
        int upperBound = students.size() - 1;
        String currentIndexValue;

        while (upperBound >= lowerBound){
            int currentIndex = (lowerBound + upperBound) / 2;
            currentIndexValue = (String)( (Student) students.get(currentIndex) ).getID();

            if( currentIndexValue.compareTo(ID) == 0 ){
                return (UniversityStudent)students.get(currentIndex);
            }
            if( currentIndexValue.compareTo(ID) < 0){
                lowerBound = currentIndex + 1;
            }
            if( currentIndexValue.compareTo(ID) > 0){
                upperBound = currentIndex - 1;
            }
        }
        return null;
    }

    /**
        * Binary search current staff stored via ID.
        *
        * @param ID staff ID to search
        *
        * @returns UniversityStaff stored on system if existent, else
null.
     */
    private UniversityStaff binarySearchStaffID( String ID ){
        int lowerBound = 0;
        int upperBound = staff.size() - 1;
        String currentIndexValue;

        while (upperBound >= lowerBound){
            int currentIndex = (lowerBound + upperBound) /2;
            currentIndexValue = (String)( (Staff) staff.get(currentInd
ex) ).getID();

            if( currentIndexValue.compareTo(ID) == 0 ){
                return (UniversityStaff)staff.get(currentIndex);
            }
            if( currentIndexValue.compareTo(ID) < 0){
                lowerBound = currentIndex + 1;
            }
            if( currentIndexValue.compareTo(ID) > 0){
                upperBound = currentIndex - 1;
            }
        }
        return null;
    }

    /**
        * Binary search current modules stored via code.
        *
        * @param code module code to search
        *
        * @returns UniversityModule stored on system if existent, else
 null.
     */
    private UniversityModule binarySearchModuleCode( String code ){
        int lowerBound = 0;
        int upperBound = modules.size() - 1;
        String currentIndexValue;

        while (upperBound >= lowerBound){
            int currentIndex = (lowerBound + upperBound) /2;
            currentIndexValue = (String)( (Module) modules.get(currentIndex) ).getCode();

            if( currentIndexValue.compareTo(code) == 0 ){
                return (UniversityModule)modules.get(currentIndex);
            }
            if( currentIndexValue.compareTo(code) < 0){
                lowerBound = currentIndex + 1;
            }
            if( currentIndexValue.compareTo(code) > 0){
                upperBound = currentIndex - 1;
            }
        }
        return null;
    }

    /**
        * Binary insert a student into the manager in order of ID string lexographical ordering.
        *
        * @param student student to insert
     */
    private void binaryInsertStudent( UniversityStudent student ) {

       if (students.size() == 0) {
           students.add(student);
           return;
       }
       int lowerBound = 0;
       int upperBound = students.size() - 1;
       int currentIndex = 0;
       String currentIndexValue;

       while (true) {
            currentIndex = (upperBound + lowerBound) / 2;
            currentIndexValue = (String)( (Student)students.get(currentIndex) ).getID();
            if ( currentIndexValue.compareTo( student.getID() ) == 0 ) {
                break;
            } else if ( currentIndexValue.compareTo( student.getID() ) < 0 ) {
                lowerBound = currentIndex + 1;
                if ( lowerBound > upperBound ) {
                    currentIndex += 1;
                    break;
                }
            } else {
                upperBound = currentIndex - 1;
                if ( lowerBound > upperBound )
                    break;
            }
       }

       ObjectArrayList newArray = new ObjectArrayList();
       for( int i=0; i<currentIndex; i++ ) {
           newArray.add( students.get(i) );
       }
       newArray.add( student );
       for( int i=currentIndex; i<students.size(); i++ ) {
           newArray.add( students.get(i) );
       }

       students = newArray;
    }

    /**
        * Binary insert a module into the manager in order of code string lexographical ordering.
        *
        * @param module module to insert
     */
    private void binaryInsertModule( UniversityModule module ) {

       if ( modules.size() == 0 ) {
           modules.add(module);
           return;
       }
       int lowerBound = 0;
       int upperBound = modules.size() - 1;
       int currentIndex = 0;
       String currentIndexValue;

       while (true) {
            currentIndex = (upperBound + lowerBound) / 2;
            currentIndexValue = (String)( (Module)modules.get(currentIndex) ).getCode();
            if ( currentIndexValue.compareTo( module.getCode() ) == 0 ) {
                break;
            } else if ( currentIndexValue.compareTo( module.getCode() ) < 0 ) {
                lowerBound = currentIndex + 1;
                if ( lowerBound > upperBound ) {
                    currentIndex += 1;
                    break;
                }
            } else {
                upperBound = currentIndex - 1;
```

```
                 if ( lowerBound > upperBound )
                     break;
             }
        }

        ObjectArrayList newArray = new ObjectArrayList();
        for( int i=0; i<currentIndex; i++ ) {
            newArray.add( modules.get(i) );
        }
        newArray.add( module );
        for( int i=currentIndex; i<modules.size(); i++ ) {
            newArray.add( modules.get(i) );
        }

        modules = newArray;
    }

    /**
         * Binary insert a staff member into the manager in order of ID string lexographical ordering.
         *
         * @param staff staff to insert
     */
    private void binaryInsertStaff( UniversityStaff staff ) {

        if (this.staff.size() == 0) {
            this.staff.add(staff);
            return;
        }
        int lowerBound = 0;
        int upperBound = this.staff.size() - 1;
        int currentIndex;
        String currentIndexValue;

        while (true) {
             currentIndex = (upperBound + lowerBound) / 2;
             currentIndexValue = (String)( (Staff)this.staff.get(currentIndex) ).getID();
             if ( currentIndexValue.compareTo( staff.getID() ) == 0 ) {
                 break;
             } else if ( currentIndexValue.compareTo( staff.getID() ) < 0 ) {
                 lowerBound = currentIndex + 1;
                 if ( lowerBound > upperBound ) {
                     currentIndex += 1;
                     break;
                 }
             } else {
                 upperBound = currentIndex - 1;
                 if ( lowerBound > upperBound )
                     break;
             }
        }

        ObjectArrayList newArray = new ObjectArrayList();
        for( int i=0; i<currentIndex; i++ ) {
            newArray.add( this.staff.get(i) );
        }
        newArray.add( staff );
        for( int i=currentIndex; i<this.staff.size(); i++ ) {
            newArray.add( this.staff.get(i) );
        }

        this.staff = newArray;
    }




    /**
        * @inheritDoc
     */
    public String addStudent(String forename, String surname, byte stage) throws InvalidStageException {

        UniversityStudent student = new UniversityStudent( forename, surname, stage );
        String studentID = generateStudentID();

        try {
            student.setID( studentID );
        }
        catch( InvalidIDException e ) {
            e.printStackTrace();
        }
        catch( IDAlreadySetException e ) {
            e.printStackTrace();
        }

        binaryInsertStudent( student );
        return student.getID();
    }


    /**
        * @inheritDoc
     */
    public void addStudent(Student student) throws IDAlreadySetException {
        UniversityStudent uniStudent = null;
        try {
            uniStudent = new UniversityStudent( student.getForename(), student.getSurname(), student.getStage() );
        }
        catch( InvalidStageException e ) {
            e.printStackTrace();
```

```java
        }

        if( student.getID() == null ) {
            try {
                String studentID = generateStudentID();
                uniStudent.setID( studentID );
                student.setID( studentID );
            }
            catch( InvalidIDException e ) {
                e.printStackTrace();
            }
        }
        else {
            throw new IDAlreadySetException( "Student already has ID: 
" + student.getID() + " and so cannot be added to the system." );
        }

        binaryInsertStudent( uniStudent );
    }


    /**
        * Generate a random student ID
        *
        * @returns random student ID
     */
    private String generateStudentID() {
        String num1 = Integer.toString((int)(Math.random() * 99999 + 0
));
        String num2 = Integer.toString((int)(Math.random() * 99999 + 1
));
        String ID = num1 + num2;

        int zeros = 10 - ID.length();
        while( zeros > 0 ) {
            ID = "0" + ID;
            zeros--;
        }

        if( binarySearchStudentID(ID) != null ) {
            //try again if the random ID generated was already an exis
tent in manager
            return generateStudentID();
        }

        assert ID != null;
        return ID;
    }

    /**
        * @inheritDoc
     */
    public String addModule(String name, byte credits, byte stage, int
 capacity, Staff[] staff)
    throws InvalidStageException, InvalidCreditsException, InvalidCapacityException,
    DuplicateStaffException, StaffNotInSystemException {

        if( capacity < 1 ) {
            throw new InvalidCapacityException( "Attempted to add module " + name + " with invalid capacity less than 1. " );
        }
        for( int i=0; i<staff.length; i++ ) {
            for( int j=i+1; j<staff.length; j++ ) {
                if( staff[i].equals(staff[j]) ) {
                    assert i != j;
                    throw new DuplicateStaffException( "Duplicate staff in array, therefore cannot add to module" );
                }
            }
        }


        UniversityModule module = null;
        String moduleCode = generateModuleCode();

        for( Staff staffToAdd : staff ) {
            if( binarySearchStaffID(staffToAdd.getID()) == null )  {
                throw new StaffNotInSystemException( "Attempted to add staff member '" + staffToAdd.getForename() + " " + staffToAdd.getSurname() +
                                                     "' who does not exist in the system onto module: " + name );
            }
        }

        try {
            module = new UniversityModule( name, credits, stage, capacity, moduleCode );
        }
        catch( InvalidIDException e ) {
            e.printStackTrace();
        }
        catch( IDAlreadySetException e ) {
            e.printStackTrace();
        }


        for( Staff staffToAdd : staff ) {
            UniversityStaff staffInSystem = binarySearchStaffID( staffToAdd.getID() );
            try {
                module.addStaff( staffInSystem );
            }
            catch( IDNotSetException e ) {
                e.printStackTrace();
            }
            catch( ModuleDiscontinuedException e ) {
                e.printStackTrace();
            }
            staffInSystem.addTeachingModule( module );
        }

        binaryInsertModule( module );
        return module.getCode();
    }

    /**
        * @inheritDoc
     */
    public void addModule(Module module) throws IDAlreadySetException
{
        UniversityModule uniModule = null;
        try {
            uniModule = new UniversityModule( module.getName(), module
.getCredits(), module.getStage(), module.getCapacity() );
        }
        catch( InvalidStageException e ) {
            e.printStackTrace();
        }
        catch( InvalidCreditsException e ) {
            e.printStackTrace();
        }

        if( module.getCode() == null ) {
            try {
                String moduleCode = generateModuleCode();
                uniModule.setCode( moduleCode );
                module.setCode( moduleCode );
            }
            catch( InvalidIDException e ) {
                e.printStackTrace();
            }
        }
        else {
            throw new IDAlreadySetException( "Module already has code:
 " + module.getCode() + " and so cannot be added to the system." );
        }

        binaryInsertModule( uniModule );
    }

    /**
        * Generate a random module code
        *
        * @returns random module code
     */
    public String generateModuleCode() {
        String code = Integer.toString((int)(Math.random() * 99999 + 1
));

        int zeros = 5 - code.length();
        while( zeros > 0 ) {
            code = "0" + code;
            zeros--;
        }

        if( binarySearchModuleCode(code) != null ) {
            //try again
            return generateModuleCode();
        }

        assert code != null;
        return code;
    }

    /**
        * @inheritDoc
     */
    public String addStaff(String forename, String surname) {
        UniversityStaff staff = new UniversityStaff( forename, surname
 );

        String staffID = generateStaffID();
        try {
            staff.setID( staffID );
        }
        catch( InvalidIDException e ) {
            e.printStackTrace();
        }
        catch( IDAlreadySetException e ) {
            e.printStackTrace();
        }

        binaryInsertStaff( staff );
        return staff.getID();
    }

    /**
        * @inheritDoc
     */
    public void addStaff(Staff staff) throws IDAlreadySetException {

        UniversityStaff uniStaff = new UniversityStaff( staff.getForen
ame(), staff.getSurname() );

        if( staff.getID() == null ) {
            try {
                String staffID = generateStaffID();
                uniStaff.setID( staffID );
                staff.setID( staffID );
            }
            catch( InvalidIDException e ) {
                e.printStackTrace();
            }
        }
        else {
            throw new IDAlreadySetException( "Staff member already has
 ID: " + staff.getID() + " and so cannot be added to the system." );
        }

        binaryInsertStaff( uniStaff );
    }

    /**
        * Generate a random staff ID
        *
        * @returns random staff ID
     */
    public String generateStaffID(){
        String ID = Integer.toHexString((int)(Math.random() * 1048575
+ 0));

        int zeros = 5 - ID.length();
        while( zeros > 0 ) {
            ID = "0" + ID;
            zeros--;
        }

        if( binarySearchStaffID(ID) != null ){
            //try again
            return generateStaffID();
        }

        assert ID != null;
        return ID;
    }

    /**
        * @inheritDoc
     */
    public void discontinue(String moduleCode) throws InvalidIDExcepti
on,
    IDNotRecognisedException {
        UniversityModule.checkValidCode(moduleCode);
        UniversityModule module = binarySearchModuleCode(moduleCode);

        if( module == null ) {
            throw new IDNotRecognisedException( "Module code: " + modu
leCode + " not found in the system." );
        }
        Student[] students = module.getStudents();

        for( int i=0; i<students.length; i++ ) {
            UniversityStudent student = ( (UniversityStudent) students
[i] );
            student.removeModule( module );
        }

        Staff[] staff = module.getTeachingStaff();
        for( Staff staffMember : staff ) {
            ( (UniversityStaff) staffMember ).removeTeachingModule( mo
dule );
        }
        module.discontinue();
    }

    /**
        * @inheritDoc
     */
    public void enrol(String studentID, String moduleCode) throws Inva
lidIDException,
    IDNotRecognisedException, ModuleAtCapacityException,
    InsufficientAvailableCreditsException, ModuleDiscontinuedException
,
    ModuleStageTooHighException, EnrollingWouldPreventHonoursException
 {

        UniversityStudent.checkValidID( studentID );
        UniversityModule.checkValidCode( moduleCode );

        UniversityStudent student = binarySearchStudentID(studentID);
        UniversityModule module = binarySearchModuleCode(moduleCode);

        if( student == null || module == null ) {
            throw new IDNotRecognisedException( "The student ID : " +
studentID + " or the module code: " + moduleCode +
            " does not reference a student or module on the system.");
        }

        if( module.isAtCapacity() ) {
            throw new ModuleAtCapacityException( "The module with code
: " + moduleCode + " is at capacity and cannot enrol students" );
        }

        if( student.getTotalCredits() + module.getCredits() > 120 ) {
            throw new InsufficientAvailableCreditsException( "The stud
ent with ID: " + studentID + " does not have enough credits left to en
rol onto module with code : "
            + moduleCode );
        }

        if( module.isDiscontinued() ) {
            throw new ModuleDiscontinuedException( "The module with co
de: " + moduleCode + " is discontinued and so cannot enrol students" )
;
        }

```

```java
        if( module.getStage() > student.getStage() ) {
            throw new ModuleStageTooHighException( "The module with code: " + moduleCode + " is of a higher stage than student with ID: " + studentID );
        }

        if( module.getStage() != student.getStage() ) {
            if( module.getCredits() + student.getLowerStageCredits() > 30 ) {
                throw new EnrollingWouldPreventHonoursException( "Enrolling student with ID: " + studentID + " to module with code: " + moduleCode
                + "would give the student more than 30 credits at a lower stage, preventing honours.");
            }
        }

        student.assignModule( module );
        module.addStudent( student );
    }

    /**
        * @inheritDoc
     */
    public void loadAllocationManager(String filename) throws IOException,
    ClassNotFoundException {
        UniversityAllocationManager manager = null ;
        FileInputStream fileInput = null ;
        ObjectInputStream objectInput = null ;
        fileInput = new FileInputStream ( filename );
        objectInput = new ObjectInputStream ( fileInput );
        manager = (UniversityAllocationManager)objectInput.readObject( );
        objectInput.close ();
    }

    /**
        * @inheritDoc
     */
    public int getNumberOfStaff() {
        return staff.size();
    }

    /**
        * @inheritDoc
     */
    public int getNumberOfStudents() {
        return students.size();
    }

    /**
        * @inheritDoc
     */
    public int getNumberOfModules() {
        return modules.size();
    }

    /**
        * @inheritDoc
     */
    public Staff[] getStaff()  {
        Staff[] staff = new Staff[this.staff.size()];

        for( int i=0; i<this.staff.size(); i++ ) {
            staff[i] = (Staff)this.staff.get(i);
        }
        return staff;
    }

    /**
        * @inheritDoc
     */
    public Staff[] getStaff(String moduleCode) throws InvalidIDExcepti
on,
    IDNotRecognisedException  {

        UniversityModule.checkValidCode( moduleCode );
        UniversityModule module = binarySearchModuleCode(moduleCode);

        if( module == null ) {
            throw new IDNotRecognisedException( "The module code " + m
oduleCode + " does not exist on the system.");
        }
        return module.getTeachingStaff();
    }

    /**
        * @inheritDoc
     */
    public Student[] getStudents() {
        Student[] students = new Student[this.students.size()];

        for( int i=0; i<this.students.size(); i++ ) {
            students[i] = (Student)this.students.get(i);
        }
        return students;
    }

    /**
        * @inheritDoc
     */
    public Student[] getStudents(String moduleCode) throws InvalidIDEx
ception,
    IDNotRecognisedException {

        UniversityModule.checkValidCode( moduleCode );
        UniversityModule module = binarySearchModuleCode(moduleCode);

        if( module == null ) {
            throw new IDNotRecognisedException( "The module code " + m
oduleCode + " does not exist on the system.");
        }
        return module.getStudents();
    }

    /**
        * @inheritDoc
     */
    public Module[] getModules() {
        Module[] modules = new Module[this.modules.size()];

        for( int i=0; i<this.modules.size(); i++ ) {
            modules[i] = (Module)this.modules.get(i);
        }
        return modules;
    }

    /**
        * @inheritDoc
     */
    public Module[] getRunningModules() {

        ObjectArrayList modules = new ObjectArrayList();

        for( int i=0; i<this.modules.size(); i++ ) {
            Module module = (Module)this.modules.get(i);

            if( !module.isDiscontinued() ) {
                modules.add( module );
            }
        }
        Module[] runningModules = new Module[modules.size()];

        for( int i=0; i<modules.size(); i++ ) {
            runningModules[i] = (Module)modules.get(i);
        }
        return runningModules;
    }

    /**
        * @inheritDoc
     */
    public Module[] getAvailableModules()  {
        ObjectArrayList modules = new ObjectArrayList();

        for( int i=0; i<this.modules.size(); i++ ) {
            UniversityModule module = (UniversityModule)this.modules.g
et(i);
```

```java

            if( !module.isDiscontinued() && !module.isAtCapacity() ) {
                modules.add( module );
            }
        }
        Module[] avaliableModules = new Module[modules.size()];

        for( int i=0; i<modules.size(); i++ ) {
            avaliableModules[i] = (Module)modules.get(i);
        }
        return avaliableModules;
    }

    /**
        * @inheritDoc
     */
    public Module[] getModules(String studentID) throws InvalidIDExcep
tion,
    IDNotRecognisedException  {

        UniversityStudent.checkValidID( studentID );
        UniversityStudent student = binarySearchStudentID(studentID);

        if( student == null ) {
            throw new IDNotRecognisedException( "The student ID : " +
studentID + " does not exist on the system.");
        }

        return student.getEnrolledModules();
    }

   /**
        * @inheritDoc
     */
    public int getNumberOfFullyAllocatedStudents()  {
        int numberFullStudents = 0;

        for( int i=0; i<students.size(); i++ ) {
            UniversityStudent student = (UniversityStudent)students.ge
t(i);

            if( student.getTotalCredits() == 120 ) {
                numberFullStudents++;
            }
        }
        return numberFullStudents;
    }

    /**
        * @inheritDoc
     */
    public int getNumberOfModulesAtCapacity() {
        int numberFullModules = 0;

        for( int i=0; i<modules.size(); i++ ) {
            UniversityModule module = (UniversityModule)modules.get(i);

            if( module.isAtCapacity() ) {
                numberFullModules++;
            }
        }
        return numberFullModules;
    }


    /**
        * @inheritDoc
     */
    public Module[] remove(Staff staff) throws InvalidIDException,
    IDNotRecognisedException, IDNotSetException {

        if( staff.getID() == null ) {
            throw new IDNotSetException( "Staff member with name " + staff.getForename() + staff.getSurname() + " has no ID set and so cannot be removed." );
        }
        UniversityStaff.checkValidID( staff.getID() );

        UniversityStaff staffToRemove = binarySearchStaffID( staff.getID() );
        ObjectArrayList nowNoStaff = new ObjectArrayList();

        for( Module module : staffToRemove.getTeachingModules() ) {
            try {
                module.removeStaff( staffToRemove );
            }
            catch( StaffNotInvolvedException e ) {
                e.printStackTrace();
            }

            if( ((UniversityModule)module).getTeachingStaff().length == 0 ) {
                nowNoStaff.add( module );
            }
        }
        Module[] noStaff = new Module[nowNoStaff.size()];

        for( int i=0; i<nowNoStaff.size(); i++ ) {
            noStaff[i] = (Module)nowNoStaff.get(i);
        }
        this.staff.remove( staffToRemove );
        return noStaff;
    }

    /**
```

```java
        * @inheritDoc
     */
    public void remove(Student student) throws InvalidIDException,
    IDNotRecognisedException, IDNotSetException {

        if( student.getID() == null ) {
            throw new IDNotSetException( "Student with name " + student.getForename() + student.getSurname() + " has no ID set, and so cannot be removed." );
        }

        UniversityStudent studentToRemove = binarySearchStudentID( student.getID() );
        if( studentToRemove == null ) {
            throw new IDNotRecognisedException( "The student ID : " + studentToRemove.getID() + " does not exist on the system." );
        }

        for( Module module : studentToRemove.getEnrolledModules() ) {
            unEnrol( studentToRemove.getID(), module.getCode() );
        }

        students.remove( studentToRemove );
    }

    /**
        * @inheritDoc
     */
    public void saveAllocationManager(String filename) throws IOException {
        FileOutputStream fileOutput = null;
        ObjectOutputStream objectOutput = null;
        fileOutput = new FileOutputStream(filename);
        objectOutput = new ObjectOutputStream(fileOutput);
        objectOutput.writeObject(this);
        objectOutput.close ();
    }

    /**
        * @inheritDoc
     */
    public boolean unEnrol(String studentID, String moduleCode) throws
    InvalidIDException, IDNotRecognisedException {

        UniversityStudent.checkValidID(studentID);
        UniversityModule.checkValidCode(moduleCode);

        UniversityStudent student = binarySearchStudentID(studentID);
        UniversityModule module = binarySearchModuleCode(moduleCode);

        if( student == null || module == null ) {
            throw new IDNotRecognisedException( "The student ID : " + studentID + " or the module code: " + moduleCode +
                " does not reference a student or module on the system.");
        }

        for( Module enrolledModule : student.getEnrolledModules() ) {

            if( enrolledModule.getCode().equals(module.getCode()) ) {
                ((UniversityModule)enrolledModule).removeStudent( student );
                student.removeModule( (UniversityModule)enrolledModule );
                return true;
            }
        }
        return false;
    }
}
```

```
package university;

/**
 * UniversityAllocationManager
 * <p>
 * Allocation Manager to hold staff, students and modules in a university and their relationships.
 *
 * @author 660037119, 660047784
 * @date 28/03/2017
 */

public class UniversityStudent implements Student
{
    private String forename;
    private String surname;
    private byte stage;
    private ObjectArrayList modules;
    private byte sameStageCredits;
    private byte lowerStageCredits;
    private byte totalCredits;
    private String id;

    /**
     * Constructor for University Student without ID.
     *
     * @param forename forename of student
     * @param surname surname of student
     * @param stage stage of student
     * @throws InvalidStageException if the student has a stage out of range 1-4 inclusive
     */
    public UniversityStudent( String forename, String surname, byte stage ) throws InvalidStageException {
        this.forename = forename;
        this.surname = surname;
        setStage( stage );
        modules = new ObjectArrayList();
    }

    /**
     * Constructor for University Student without ID.
     *
     * @param forename forename of student
     * @param surname surname of student
     * @param stage stage of student
     * @param id id of the student
     * @throws InvalidStageException if the student has a stage out of range 1-4 inclusive
     */
    public UniversityStudent( String forename, String surname, byte stage, String id ) throws InvalidStageException, InvalidIDException, IDAlreadySetException {
```

```java
        this.forename = forename;
        this.surname = surname;
        setStage( stage );
        setID( id );
        modules = new ObjectArrayList();
    }

    /**
     * Check an ID is of correct student form.
     *
     * @param id id to be checked
     * @throws InvalidIDException if the ID is of incorrect form
     */
    public static void checkValidID(String id) throws InvalidIDException{
        boolean valid = true;

        if( id.length() != 10 ) valid = false;

        try {
            Long.parseLong( id );
        }
        catch( NumberFormatException e ) {
            valid = false;
        }

        if( !valid ) {
            throw new InvalidIDException("Student ID provided must be 10 digits.");
        }
    }

    /**
        * @inheritDoc
     */
    public byte getStage() {
        return stage;
    }

    /**
     * Set the stage of a student.
     *
     * @param stage stage of student to be set
     * @throws InvalidStageException if the stage is out of range 1-4 inclusive.
     */
    private void setStage( byte stage ) throws InvalidStageException {
        if( stage < 1 || stage > 4 ) {
            throw new InvalidStageException( "Stage for student '" + forename + surname + "' must be between 1 and 4 inclusive." );
        }
        else {
            this.stage = stage;
        }
    }

    /**
        * @inheritDoc
     */
    public String getForename() {
        return forename;
    }

    /**
        * @inheritDoc
     */
    public String getSurname() {
        return surname;
    }

    /**
        * @inheritDoc
     */
    public String getID() {
        return id;
    }

    /**
        * Return the credits the student has from modules enrolled at the same stage as them
        *
        * @returns credits at the same stage
     */
    public byte getSameStageCredits() {
        return sameStageCredits;
    }

    /**
        * Return the credits the student has from modules enrolled at a lower stage than them
        *
        * @returns credits at a lower stage
     */
    public byte getLowerStageCredits() {
        return lowerStageCredits;
    }

    /**
        * Return the credits the student has from modules enrolled at a higher stage
        *
        * @returns credits at a higher stage
     */
    public byte getTotalCredits() {
        return totalCredits;
    }
```

```java

    /**
        * @inheritDoc
     */
    public void setID( String id ) throws IDAlreadySetException, Inval
idIDException {
        if( this.id == null) {
            checkValidID(id);
            this.id = id;
        }
        else {
            throw new IDAlreadySetException ("Student '" + forename +
" " + surname + "' already has ID: " + this.id);
        }
    }

    /**
        * Return the modules this student is currently enrolled on.
        *
        * @returns modules enrolled
     */
    public Module[] getEnrolledModules() {
        Module[] modules = new Module[this.modules.size()];

        for( int i = 0; i < this.modules.size(); i++ ) {
            modules[i] = (Module)this.modules.get(i);
        }
        return modules;
    }

    /**
        * Assigns and links the student to a module.
        *
        * @param module module to be assigned
     */
    public void assignModule( UniversityModule module ) {
        modules.add( module );

        if( module.getStage() == stage ) {
            sameStageCredits += module.getCredits();
        }
        else {
            lowerStageCredits += module.getCredits();
        }
        totalCredits += module.getCredits();
    }

        /**
        * Removes and unlinks the student from a module.
        *
        * @param module module to be assigned
     */
    public void removeModule( UniversityModule module ) {
        modules.remove( module );

        if( module.getStage() == stage ) {
            sameStageCredits -= module.getCredits();
        }
        else {
            lowerStageCredits -= module.getCredits();
        }
        totalCredits -= module.getCredits();
    }
}

```

```java
package university;


/**
 * @author 660047784, 660037119
 * @date 28/03/2017
 */
public class UniversityStaff implements Staff
{
    private String forename;
    private String surname;
    private String id;
    private ObjectArrayList teachingModules;

    /**
     * Constructor for UniversityStaff without ID
     *
     * @param forename Forename of staff member
     * @param surname Surname of staff member
     */
    public UniversityStaff( String forename, String surname ) {
        this.forename = forename;
        this.surname = surname;
        teachingModules = new ObjectArrayList();
    }

    /**
     * Constructor for UniversityStaff with ID
     *
     * @param forename Forename of staff member
     * @param surname Surname of staff member
     * @param id Staff member ID
     *
     * @throws IDAlreadySetException if there is already a staff member with that id in the system
     * @throws InvalidIdException if the format of the id is invalid
     */
    public UniversityStaff( String forename, String surname, String id ) throws IDAlreadySetException, InvalidIDException {
        this.forename = forename;
        this.surname = surname;
        setID( id );
        teachingModules = new ObjectArrayList();
    }

    /**
     * Checks if the id follows the valid conventions
     *
     * @throws InvalidIDException if id is invalid
     */
    public static void checkValidID( String id ) throws InvalidIDException {
        boolean valid = true;

        //length of id must be 5 characters
        if( id.length() != 5 ) valid = false;

        //if the id can pass as a hex value
        try {
            Integer.parseInt( id, 16 );
        }
        catch( NumberFormatException e ) {
            valid = false;
        }

        if( !valid ) {
            throw new InvalidIDException( "ID provided must be 5 hex c
haracters." );
        }
    }

    /**
     * @inheritDoc
     */
    public String getForename() {
        return forename;
    }

    /**
     * @inheritDoc
     */
    public String getSurname() {
        return surname;
    }

    /**
     * @inheritDoc
     */
    public String getID() {
        return id;
    }

    /**
     * @inheritDoc
     */
    public void setID( String id ) throws IDAlreadySetException, Inval
idIDException {
        //if the staff member doesn't have an id
        if( this.id == null ) {
            checkValidID(id);
            this.id = id;
        }
        else {
            throw new IDAlreadySetException( "Staff member '" + forena
me + " " + surname + "' already has ID: " + id );
        }
    }

    /**
     * Links a module to the staff member.
     * @param module module to be linked
     */
    public void addTeachingModule( UniversityModule module ) {
        teachingModules.add( module );
    }

    /**
     * Remove link to a module.
     * @param module module to be removed.
     */
    public void removeTeachingModule( UniversityModule module ) {
        teachingModules.remove( module );
    }

    /**
     * Get all modules the staff is linked and teaching on.
     *
     * @returns modules linked
     */
    public Module[] getTeachingModules() {
        Module[] modules = new Module[this.teachingModules.size()];

        //convert teachingModules into a module array
        for( int i = 0; i < this.teachingModules.size(); i++ ) {
            modules[i] = (Module)this.teachingModules.get(i);
        }
        return modules;
    }
}
```

```java
package university;

/**
 *
 *
 * @author 660047784, 660037119
 * @date 28/03/2016
 */
public class UniversityModule implements Module
{
    private String name;
    private String code;
    private byte credits;
    private byte stage;
    private ObjectArrayList teachingStaff;
    private ObjectArrayList students;
    private int capacity;
    private int enrolled;
    private boolean discontinued;

    /**
     * Constructor for UniversityModule without module code
     *
     * @param name Name of the module
     * @param credits The number of credits that the module gives
     * @param stage What stage the module can be taken at
     * @param capacity The maximum number of students that can take the module
     */
    public UniversityModule( String name, byte credits, byte stage, int capacity ) throws InvalidStageException, InvalidCreditsException {
        this.name = name;
        setCredits( credits );
        setStage( stage );
        teachingStaff = new ObjectArrayList();
        students = new ObjectArrayList();
        this.capacity = capacity;
        discontinued = false;
    }

    /**
     * Constructor for UniversityModule with module code
     *
     * @param name Name of the module
     * @param credits The number of credits that the module gives
     * @param stage What stage the module can be taken at
     * @param capacity The maximum number of students that can take the module
     * @param code The module code
     */
    public UniversityModule( String name, byte credits, byte stage, int capacity, String code ) throws InvalidStageException, InvalidCreditsException, InvalidIDException,
        IDAlreadySetException {
        this.name = name;
        setCredits( credits );
        setStage( stage );
        teachingStaff = new ObjectArrayList();
        students = new ObjectArrayList();
        this.capacity = capacity;
        discontinued = false;
        setCode( code );
    }

    /**
     * Checks that the format of the module code is correct
     *
     * @param code The module code to check
     * @return true if the module code is valid, else false
     */
    public static boolean checkValidCode( String code ) throws InvalidIDException {
        boolean valid = true;

        //if the code isn't 5 characters
        if( code.length() != 5 ) valid = false;

        //if the code passes as an integer
        try {
            Integer.parseInt( code );
        }
        catch( NumberFormatException e ) {
            valid = false;
        }
        if( !valid ) {
            throw new InvalidIDException("Module code provided must be 5 digits.");
        }
        return valid;
    }

    /**
     * @inheritDoc
     */
    public String getCode(){
        return code;
    }

    /**
     * @inheritDoc
     */
    public void setCode(String code) throws InvalidIDException, IDAlreadySetException {
        //if the module doesn't have a module code
        if( this.code == null) {
            checkValidCode(code);
            this.code = code;
        }
        else { throw new IDAlreadySetException ("This module already h
as a code (" + this.code + " )" );
        }
    }

    /**
     * @inheritDoc
     */
    public String getName(){
        return name;
    }

    /**
     * @inheritDoc
     */
    public byte getStage(){
        return stage;
    }

    /**
     * Set the stage of the module.
     *
     * @param stage stage to be set.
     * @throws InvalidStageException if the stage is not 1-4 inclusive
.
     */
    private void setStage( byte stage ) throws InvalidStageException {
        //check the stage is between 1 and 4
        if( stage < 1 || stage > 4 ) {
            throw new InvalidStageException( "Stage for module '" + na
me + "' must be between 1 and 4 inclusive." );
        }
        else {
            this.stage = stage;
        }
    }

    /**
     * Set the credits of the module.
     *
     * @param credits credits to be set.
     * @throws InvalidCreditsException if the credits are not between
0-120 inclusive.
     */
    private void setCredits( byte credits ) throws InvalidCreditsExcep
tion {
        //check that the number of credits is between 0 and 120
        if( credits < 0 || credits > 120 ) {
            throw new InvalidCreditsException( "Credits for module '"
+ name + "' must be between 0 and 120 inclusive." );
        }
```

```java
        else {
            this.credits = credits;
        }
    }

    /**
     * Returns the credits valued by the module
     *
     * @returns the credits of the module
     */
    public byte getCredits() {
        return credits;
    }

    /**
     * @inheritDoc
     */
    public Staff[] getTeachingStaff(){
        Staff[] teachingStaff = new Staff[this.teachingStaff.size()];

        //convert staff stored by module to a staff array
        for( int i = 0; i < this.teachingStaff.size(); i++ ) {
            teachingStaff[i] = (Staff)this.teachingStaff.get(i);
        }
        return teachingStaff;
    }

    /**
     * @inheritDoc
     */
    public int getCapacity() {
        return capacity;
    }

    /**
     * Checks if the module has reached capacity
     *
     * @return true if module is at maximum capacity, false otherwise
     */
    public boolean isAtCapacity() {
        if( enrolled == capacity ){
            return true;
        } else {
            return false;
        }
    }

    /**
     * Discontinue the module, removing all links.
     */
    public void discontinue(){
        teachingStaff = new ObjectArrayList();
        students = new ObjectArrayList();
        enrolled = 0;
        discontinued = true;
    }

    /**
     * @inheritDoc
     */
    public boolean isDiscontinued(){
        return discontinued;
    }

    /**
     * @inheritDoc
     */
    public void addStaff(Staff[] staff) throws DuplicateStaffException
, IDNotSetException, ModuleDiscontinuedException{

        if( discontinued ) {
            throw new ModuleDiscontinuedException( "Module " + name +
" is discontinued, and so cannot accept more staff." );
        }
        for( Staff staffToAdd : staff ){
            //if the staff member doens't have an id, throw IDNotSetEx
ception
            if( staffToAdd.getID() == null ) {
                throw new IDNotSetException( "Attempted to add a staff
 member '" + staffToAdd.getForename() + " " + staffToAdd.getSurname()
+
                "' whom has no ID set into module: " + name );
            }
            for( int i=0; i < teachingStaff.size(); i++ ) {
                //if there is a staff member with the same id in the m
odule
                if ( staffToAdd.equals(teachingStaff.get(i)) ) {
                    throw new DuplicateStaffException( "Attempted to a
dd a duplicate staff member with ID: " + staffToAdd.getID() + ", in mo
dule " + name );
                }
            }
            teachingStaff.add( staffToAdd );
        }
    }

    /**
     * @inheritDoc
     */
    public void addStaff(Staff staff) throws DuplicateStaffException,
IDNotSetException, ModuleDiscontinuedException{
        if( discontinued ) {
            throw new ModuleDiscontinuedException( "Module '" + name +
 "' is discontinued, and so cannot accept more staff." );
        }
        //if the staff member doens't have an id, throw IDNotSetExcept
```

```
ion
        if( staff.getID() == null ) {
                throw new IDNotSetException( "Attempted to add a staff member '" + staff.getForename() + " " + staff.getSurname() +
                "' whom has no ID set into module: " + name );
        }
        for( int i=0; i < teachingStaff.size(); i++ ) {
            //if there is a staff member with the same id in the module
            if( staff.equals(teachingStaff.get(i)) ) {
                throw new DuplicateStaffException( "Attempted to add a duplicate staff member with ID: " + staff.getID() + ", in module " + name );
            }
        }
        teachingStaff.add( staff );
    }

    /**
     * @inheritDoc
     */
    public void removeStaff(Staff staff) throws StaffNotInvolvedException{
        boolean staffFound = teachingStaff.remove(staff);

        if( !staffFound ) {
            throw new StaffNotInvolvedException( "Staff member with ID: " + staff.getID() + " is not found in module: " + name + ", therefore cannot be removed" );
        }
    }

    /**
     * Allocates a student to the module array
     *
     * @param student Student object to add to module
     */
    public void addStudent( UniversityStudent student ) {
        students.add( student );
        enrolled++;
    }

    /**
     * Removes a student from the module
     *
     * @param student Student to remove from module
     */
    public void removeStudent( UniversityStudent student ) {
        students.remove( student );
        enrolled--;
    }

    /**
```

```
     * Gets all the students enrolled onto the module
     *
     * @return all students enrolled onto the module as a student arra
y
     */
    public Student[] getStudents() {
        Student[] students = new Student[this.students.size()];

        //convert students stored by module to a Student array
        for( int i = 0; i < this.students.size(); i++ ) {
            students[i] = (Student)this.students.get(i);
        }
        return students;
    }
}
```