CSCI121: Lab 5
Sean Overton
SN: 6421490

TASK 1:
CreateCourse.java program:
Note: (the other classes that aren't the primary class is included in both programs)

```
/*-----------------------------------------------------
My name: Sean Overton
My student number: 6421490
My course code: CSIT121
My email address: so412@uowmail.edu.au
Lab: #5
-----------------------------------------------------*/

import java.util.ArrayList;
import java.util.Scanner;
import java.util.InputMismatchException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.io.*;
import java.util.Formatter;
import java.util.FormatterClosedException;
import java.lang.SecurityException;

class CreateCourse{
   private static ObjectOutputStream output;

   public static void main(String[] args){
      /*i)) to create all subjects, majors, and courses based on the
       Bachelor of Computer Science*/
      //Create Subject objects
      Subject CSIT111 = new Subject("CSIT111", "Programming Fundamentals", 6);
      Subject CSIT113 = new Subject("CSIT113", "Problem solving", 6);
      Subject CSIT114 = new Subject("CSIT114", "System Analysis", 6);
      Subject CSIT115 = new Subject("CSIT115", "Data Management and Security", 6);
      Subject CSIT121 = new Subject("CSIT121", "Object Oriented Design and Programming", 6);
      Subject CSIT127 = new Subject("CSIT127", "Networks and Communications", 6);
      Subject CSIT128 = new Subject("CSIT128", "Introduction to Web Technology", 6);
      Subject CSCI235 = new Subject("CSCI235", "Database Systems", 6);
      Subject CSCI251 = new Subject("CSCI251", "Advanced Programming", 6);
      Subject CSIT214 = new Subject("CSIT214", "IT Project Management", 6);
      Subject MATH221 = new Subject("MATH221", "Mathematics for Computer Science", 6);
      Subject CSCI203 = new Subject("CSCI203", "Data Structures and Algorithms", 6);
      Subject CSIT226 = new Subject("CSIT226", "Human Computer Interaction", 6);
      Subject CSIT314 = new Subject("CSIT314", "Software Development Methodologies", 6);
      Subject CSIT321 = new Subject("CSIT321", "Project", 12);
      Subject CSCI317 = new Subject("CSCI317", "Database Performance Tuning", 6);
      Subject INFO411 = new Subject("INFO411", "Data Mining and Knowledge Discovery", 6);
      Subject CSCI316 = new Subject("CSCI316", "Big Data Mining Techniques and Implementation", 6);
      Subject ISIT312 = new Subject("ISIT312", "Big Data Management", 6);
      Subject CSCI301 = new Subject("CSCI301", "Contemporary Topics in Security", 6);
      Subject CSCI262 = new Subject("CSCI262", "System Security", 6);
      Subject CSCI369 = new Subject("CSCI369", "Ethical Hacking", 6);
      Subject CSIT302 = new Subject("CSIT302", "Cybersecurity", 6);
      Subject CSCI361 = new Subject("CSCI361", "Cryptography and Secure Applications", 6);
      Subject CSCI368 = new Subject("CSCI368", "Network Security", 6);
      Subject CSCI376 = new Subject("CSCI376", "Multicore and GPU Programming", 6);
      Subject CSCI236 = new Subject("CSCI236", "3D Modelling and Animation", 6);
      Subject CSCI336 = new Subject("CSCI336", "Interactive Computer Graphics", 6);
      Subject CSCI366 = new Subject("CSCI366", "Mobile Multimedia", 6);
      Subject CSCI356 = new Subject("CSCI356", "Game Engine Essentials", 6);
Subject CSCI334 = new Subject("CSCI334", "Software Design", 6);
Subject ISIT219 = new Subject("ISIT219", "Knowledge and Information Engineering", 6);
Subject CSCI318 = new Subject("CSCI318", "Software Engineering Practices and Principles", 6);
Subject ISIT315 = new Subject("ISIT315", "Semantic Web", 6);

//Create Majors and add subjects to majors:
Subject[] bigDataSubjects = {CSCI317, INFO411, CSCI316, ISIT312};
Major bigData = new Major("Big Data", bigDataSubjects);

Subject[] cyberSecSubjects = {CSCI301, CSCI262, CSCI369, CSIT302};
Major cyberSec = new Major("Cyber Security", cyberSecSubjects);

Subject[] sysSecSubjects = {CSCI361, CSCI368, CSCI262, CSCI376};
Major sysSecurity = new Major("Digital System Security", sysSecSubjects);

Subject[] softwareEngSubjects = {CSCI334, ISIT219, CSCI318, ISIT315};
Major softwareEng = new Major("Software Engineering", softwareEngSubjects);

Subject[] gameDevSubjects = {CSCI236, CSCI336, CSCI356, CSCI366,
CSCI376};
Major gameDev = new Major("Game and Mobile Development", gameDevSubjects);

//Create Course object
Course compSci = new Course("Bachelor of Computer Science", 144);

    //add all majors to course
Major[] majorList = {bigData, cyberSec, sysSecurity, softwareEng,
gameDev};
compSci.addMajors(majorList);

    //add core subjects to Course
Subject[] coreSubjects = {CSIT111, CSIT113, CSIT114,
CSIT115, CSIT121, CSIT127, CSIT128, CSCI235, CSCI251, CSIT214,
MATH221, CSCI203, CSIT226, CSIT314, CSIT321};
compSci.addCores(coreSubjects);

    //add elective subjects to course
Subject[] elecSubjects = {CSCI317, INFO411, CSCI316, ISIT312, CSCI301,
CSCI262, CSCI369, CSIT302, CSCI361, CSCI368, CSCI376, CSCI236, CSCI336,
CSCI356, CSCI366, CSCI334, CSCI318, ISIT219, ISIT315};
compSci.addElectives(elecSubjects);

//for bachelor course
openFile("bcs.ser");
addRecords(compSci);
closeFile();

//create masters course structure
Subject CSCI814 = new Subject("CSCI814", "IT Porject Management", 6);
Subject CSCI851 = new Subject("CSCI851", "Advanced Programming", 6);
Subject CSCI803 = new Subject("CSCI803", "Algorithms and Data structures", 6);
Subject CSCI835 = new Subject("CSCI835", "Database Systems", 6);
Subject CSCI862 = new Subject("CSCI862", "System Security", 6);
Subject CSIT826 = new Subject("CSIT826", "Human Computer Interaction", 6);
Subject MTS9302 = new Subject("MTS9302", "Corporate Network Management", 6);
Subject ISIT925 = new Subject("ISIT925", "Strategic Network Design", 6);
Subject CSIT940 = new Subject("CSIT940", "Research Methodology", 6);
Subject CSCI920 = new Subject("CSCI920", "Contemporary Topics in Computer Science", 6);
Subject CSCI992 = new Subject("CSCI992", "Professional Project", 12);
Subject CSCI964 = new Subject("CSCI964", "Computational IntellisGenderce", 6);
Subject CSCI924 = new Subject("CSCI924", "Reasoning and Learning", 6);
Subject CSCI944 = new Subject("CSCI944", "Perception and Planning", 6);
Subject CSCI933 = new Subject("CSCI933", "Machine Learning Algorithms and Applications", 6);
Subject CSCI935 = new Subject("CSCI935", "Computer Vision Algorithms and Systems", 6);
Subject CSCI946 = new Subject("CSCI946", "Big Data Analytics", 6);
Subject CSCI968 = new Subject("CSCI968", "Advanced Network Security", 6);
```

```
        Subject INFO912 = new Subject("INFO912", "Mathematics for cryptography", 6);
        Subject CSCI971 = new Subject("CSCI971", "Advanced Computer Security", 6);
        Subject CSCI910 = new Subject("CSCI910", "Software Requirements, Specifications and Formal Methods", 6);
        Subject CSCI926 = new Subject("CSCI926", "Software Testing and Analysis", 6);
        Subject CSCI927 = new Subject("CSCI927", "Service-orientated software engineering", 6);
        Subject ECTE903 = new Subject("ECTE903", "Image and Video Processing", 6);
        Subject INFO911 = new Subject("INFO911", "Data mining and Knowledge Discovery", 6);
        Subject INFO913 = new Subject("INFO913", "Information Theory", 6);

        Major intSys = new Major("IntellisGendert System");
        Subject[] intSysCores = {CSCI964, CSCI924, CSCI944};
        intSys.addMCores(intSysCores);

        Major macLearBD = new Major("Machine Learning and Big Data");
        Subject[] macLearBDCores = {CSCI964, CSCI924, CSCI944};
        macLearBD.addMCores(macLearBDCores);

        Major netInfoSec = new Major("Network AND Information Security");
        Subject[] netInfoSecCores = {CSCI968, INFO912, CSCI971};
        netInfoSec.addMCores(netInfoSecCores);

        Major softEngM = new Major("Software Engineering");
        Subject[] softEngMCores = {CSCI964, CSCI924, CSCI944};
        softEngM.addMCores(softEngMCores);

        Major[] MCSMajors = {softEngM, intSys, netInfoSec, macLearBD};

        Subject[] cCoresM = {CSCI814, CSCI851, CSCI803, CSCI835, MTS9302, CSIT940,
        CSCI920, CSCI992};

        Subject[] cElesM = {CSCI862, CSIT826, ISIT925, CSCI964, CSCI924,
        CSCI944, CSCI933, CSCI935, CSCI946, CSCI968, INFO912, CSCI971,
        CSCI910, CSCI926, CSCI927, ECTE903, INFO911, INFO913};

        Course mcs = new Course("Master of Computer Science", 96);
        mcs.addCores(cCoresM);
        mcs.addElectives(cElesM);
        mcs.addMajors(MCSMajors);

        //for master course
        openFile("mcs.ser");
        addRecords(mcs);
        closeFile();
    }

    public static void openFile(String fileName){
        try{
            output = new ObjectOutputStream(Files.newOutputStream(Paths.get(fileName)));
        }
        catch(IOException ioException){
            System.err.println("Error opening file. Terminating.");
            System.exit(1); //terminates program?
        }
    }

    public static void addRecords(Course course){
        try{
            //for loop here to add all data
            output.writeObject(course);
        }
        catch(IOException ioException){
            System.err.println("Error writing to file. Terminating.");
        }
    }

    public static void closeFile(){
        try{
            if(output != null){
                output.close();
            }
        }
        catch(IOException ioException){
            System.err.println("Error closing file. Terminating.");
        }
    }
interface Enrolment{
    //abstract by default
    public void addRecord(Record record);
    public void deleteRecord(int index);
    public Record getRecord(int index);
    public ArrayList<Record> getRecords();
    public void setRecords(ArrayList<Record> records);
}

class Subject implements Serializable{
    //dataprivate fields
    private String sName;
    private String code;
    private int credit;

    //default constructor
    public Subject(){
        this(" ", " ", 0);
    }

    //constructor with parameters
    public Subject(String code, String name, int creditPoint){
        this.sName = name;
        this.code = code;
        this.credit = creditPoint;
    }

    //methods
    public String getName(){
        return sName;
    }

    public String getCode(){
        return code;
    }

    public int getCredit(){
        return credit;
    }

    public boolean isSame(Subject subject){
        if (subject.getCode().equals(this.getCode())){
            return true;
        }
        else{
            return false;
        }
    }

    //implementing clone interface
    public Subject clone() throws CloneNotSupportedException{
        return (Subject) super.clone();
    }

    public String toString(){
        String printOut = getCode() +  " (" +  getName() + ", " + Integer.toString(getCredit()) + "pt)";
        return printOut;
    }
}

abstract class Student implements Enrolment, Cloneable{
    //datafields
    private String stName;
    private final String DOB;
    private ArrayList<Record> records;
    private final int STNUM;
    private String sGender;

    //constructors
    public Student(){
        this("x", "x", "x", 0);
    }

    public Student(String name, String DOB, String sGender, int STUDENTID){
        this.stName = name;
        this.DOB = DOB;
        this.sGender = sGender;
        this.STNUM = STUDENTID;
        this.records = new ArrayList<Record>();
    }

    //methods
    public String getSName(){
        return stName;
    }

    public void setSName(String stName){
        this.stName = stName;
    }

    public String getDOB(){
        return DOB;
    }

    public int getsNumberM(){
        return STNUM;
    }

    public String getsGender(){
        return sGender;
    }

    public void setsGender(String sGender){
        this.sGender = sGender;
    }

    public String toString(){
        return String.format("Student: %s (%s, %s, %s)\n",
            getSName(), Integer.toString(getsNumberM()), getsGender(), getDOB());
    }

    //implementing interface methods
    public void addRecord(Record record){
        records.add(record);
    }

    public void deleteRecord(int index){
        records.remove(index);
    }

    public Record getRecord(int index){
        return records.get(index);
    }
    public ArrayList<Record> getRecords(){
        return records;
    }

    public void setRecords(ArrayList<Record> records){
        this.records = records;
    }

    //implementing cloneabe method
    @Override
    public Student clone() throws CloneNotSupportedException{
        Student studentClone = (Student) super.clone();
        studentClone.setRecords((ArrayList<Record>)this.getRecords().clone());
        return studentClone;
    }

    //empty abstract methods
    public abstract void setExpectedCompletion(String date);
    public abstract String getExpectedCompletion();
}

class Undergraduate extends Student{
    //datafields
    private String bachelorCompletion;

    //constructors
    public Undergraduate(){
        super();
        this.bachelorCompletion ="x";
    }

    public Undergraduate(String name, String DOB, int STUDENTID, String sGenderder, String expectedCompletionSession){
        super(name, DOB, sGenderder, STUDENTID);
        this.bachelorCompletion = expectedCompletionSession;
    }

    //methods
    public String getExpectedCompletion(){
        return bachelorCompletion;
    }

    public void setExpectedCompletion(String expectedCompletionSession){
        this.bachelorCompletion = expectedCompletionSession;
    }

    @Override
    public String toString(){
        String printOut = super.toString();
        for(int i = 0; i < getRecords().size(); i++){
            Record record = getRecord(i);
            printOut += String.format("\nCourse Name: %s, (%s) \n\n%s",
            record.getCourseName(), record.getStatus(), record);
        }

        return printOut;
    }
}

class Postgraduate extends Student{
    //datafields
    private String bachelorCompletion;
    private String masterCompletion;

    //constructors
    public Postgraduate(){
        super();
        this.bachelorCompletion ="x";
        this.masterCompletion = "x";
    }

    public Postgraduate(String name, String DOB, int STUDENTID, String sGenderder,
    String bachelorCompletion, String masterCompletion){
        super(name, DOB, sGenderder, STUDENTID);
        this.bachelorCompletion = bachelorCompletion;
        this.masterCompletion = masterCompletion;
    }

    //methods
    public String getExpectedCompletion(){
        return masterCompletion;
    }

    public void setExpectedCompletion(String expectedCompletionSession){
        this.masterCompletion = expectedCompletionSession;
    }

    public String getBachelorCompletion(){
        return bachelorCompletion;
    }

    public void setBachelorCompletion(String bachelorCompletion){
        this.bachelorCompletion = bachelorCompletion;
    }

    @Override
    public String toString(){
        String printOut = super.toString();
        for(int i = 0; i < getRecords().size(); i++){
            Record record = getRecord(i);
            printOut += String.format("\n\nCourse Name: %s, (%s) \n\n%s",
            record.getCourseName(), record.getStatus(), record);
        }

        printOut += "\nExpected Master Graduation: " + this.getExpectedCompletion();
        // printOut += "\n\nBachelor was received: " + getBachelorCompletion();

        return printOut;
    }
}

class Record implements Cloneable{
    //datafields
    private final String CNAME;
    private ArrayList<Subject> eCores;
    private Major eMajor;
    private ArrayList<Subject> eElectives;
    private int totalCredit;
    private Status status;

    public enum Status {ACTIVE, COMPLETE, NA}

    //default constructor
    public Record(){
        this("default");
    }

    //parameterised constructor
    public Record(String CNAME){
        this.CNAME = CNAME;
        this.eCores = new ArrayList<Subject>();
        this.eMajor = new Major();
        this.eElectives = new ArrayList<Subject>();
        this.totalCredit = 0;
        this.status = Status.NA;
    }

    //methods
    public void enrolCores(ArrayList<Subject> subjects){
        //maintains up to date credit points
        for(int i=0; i < subjects.size(); i++ ){
            this.totalCredit += subjects.get(i).getCredit();
        }

        //assigns core subjects
        this.eCores = subjects;
    }

    public void enrolMajor(Major major){
        //mtaintains up to date credit point count
        this.totalCredit += major.getCredit();
        this.eMajor = major;
    }

    public void enrolElective(Subject subject){
        this.eElectives.add(subject);
        this.totalCredit += subject.getCredit();
    }

    public void enrolElectives(ArrayList<Subject> subjects){
        //maintains up to date credit points
        for(int i=0; i < subjects.size(); i++ ){
            this.totalCredit += subjects.get(i).getCredit();
        }

        //assigns core subjects
        this.eCores = subjects;
    }

    public int getTotalCredit(){
        return totalCredit;
    }

    public void setTotalCredit(int credit){
        this.totalCredit = credit;
    }

    /*calculates number of electives left to reach minimum credit
    points for selected course*/
    public int howManyElectivesLeft(Course course){
        if(getTotalCredit() < course.getCCredit()){
            return ((course.getCCredit() - getTotalCredit())/6);
        }
        return 0;
    }

    //predicate method(just checks truth)
    public boolean isEnrolled(Subject name){
        if(eCores.contains(name)){
            return true;
        }
        else if(eElectives.contains(name)){
            return true;
        }
        else if(eMajor.isIncluded(name)){
            return true;
        }
        return false;
    }

    public Status getStatus(){
        return this.status;
    }

    public void setStatus(Status status){
        this.status = status;
    }

    public String toString(){
        String s = "";

        s+="Cores: \n";
        for(int i = 0; i < eCores.size(); i++)
            s+=eCores.get(i)+"\n";

        s+="\n";

        s+="Major: " + eMajor;

        s+="\n";

        s+="Electives: \n";
        for(int i = 0; i < eElectives.size(); i++)
            s+=eElectives.get(i)+"\n";

        s+="-----------------\n";

        s+="Total Enrolled Credit: "+ getTotalCredit() +"pt";

        return s;
    }

    public String getCourseName(){
        return CNAME;
    }

    //implementing cloneable interface
    public Record clone() throws CloneNotSupportedException{
        Record recordClone = (Record) super.clone();
        //clears credit points
        recordClone.setTotalCredit(0);

        //these methods all add relevent credit points
        recordClone.enrolMajor((Major)this.eMajor.clone());
        recordClone.enrolElectives((ArrayList<Subject>)this.eElectives.clone());
        recordClone.enrolCores((ArrayList<Subject>)this.eCores.clone());

        return recordClone;
    }
}

class Major implements Cloneable, Serializable{
    private String mName;
    private ArrayList<Subject> mCores;
    private int credit = 0;

    //default constructor
    public Major(){
        this("default");
    }

    //constructor with 1 parameter
    public Major(String name){
        this.mName = name;
        this.mCores = new ArrayList<Subject>();
    }

    //constructor with 2 parameters
    public Major(String name, Subject[] subjectList){
        this.mName = name;
        this.mCores = new ArrayList<Subject>();
        addMCores(subjectList);
    }

    public String getMName(){
        return mName;
    }

    public void setMName(String name){
        this.mName = name;
    }

    public int getCredit(){
        int credit = 0;

        for(int i = 0; i < mCores.size(); i++ ){
            credit += mCores.get(i).getCredit();
        }

        return credit;
    }

    //predicate method
    public boolean isIncluded(Subject name){
        ArrayList<Subject> subjects = this.getMCores();

        for(int i = 0; i < subjects.size(); i++ ){
            if(subjects.get(i).getName() == name.getName()){
                return true;
            }
        }
        return false;
    }

    public void addMCores(Subject[] subjects){
        for (Subject subject : subjects){
            this.mCores.add(subject);
        }
    }

    public ArrayList<Subject> getMCores(){
        return mCores;
    }

    //implementing clone interface
    @Override
    public Major clone() throws CloneNotSupportedException{
        return (Major) super.clone();
    }

    @Override
    public String toString(){
        String printOut = getMName() + "\n";
        ArrayList<Subject> subjects = getMCores();

        for(int i = 0; i < subjects.size(); i++ ){
            printOut += subjects.get(i) + "\n";
        }

        return printOut;
    }
}

class Course implements Serializable{
    private String cName;
    private ArrayList<Subject> cores;
    private ArrayList<Major> majors;
    private ArrayList<Subject> electives;
    private int cCredit;

    public Course(){
        this("default", 0);
    }

    public Course(String cName, int cCredit){
        this.cName = cName;
        this.cores = new ArrayList<Subject>();
        this.majors = new ArrayList<Major>();
        this.electives = new ArrayList<Subject>();
        this.cCredit = cCredit;
    }

    public void addCores(Subject[] subjects){
        for (Subject subject : subjects){
            this.cores.add(subject);
        }
    }

    public void addMajors(Major[] majors){
        for (Major major : majors){
            this.majors.add(major);
        }
    }

    public void addElectives(Subject[] electives){
        for (Subject elective : electives){
            this.electives.add(elective);
        }
    }

    //display all electives
    public void printElectives(){
        System.out.println("Elective Subjects:");
        for(int i = 0; i < getElectives().size(); i++){
            System.out.println(getElectives().get(i));
        }
        System.out.println();
    }

    public ArrayList<Subject> getCores(){
        return cores;
    }

    public ArrayList<Subject> getElectives(){
        return electives;
    }

    public ArrayList<Major> getMajors(){
        return majors;
    }

    public String toString(){
        String s = "Course: "+getCName() + "\n";

        s+="Cores: \n";
        for(int i = 0; i < getCores().size(); i++)
```

```
            s+=getCores().get(i)+"\n";

        s+="\n";

        s+="Majors: \n";
        for(int i = 0; i < getMajors().size(); i++){
            s+=getMajors().get(i) + "\n";
        }

        s+="\n";

        s+="Electives: \n";
        for(int i = 0; i < getElectives().size(); i++)
            s+=getElectives().get(i)+"\n";

        s+="-----------------\n";

        s+="Required Total Credit: "+ getCCredit() +"pt";

        return s;
    }

    public String getCName(){
        return cName;
    }

    public int getCCredit(){
        return cCredit;
    }
}

class InputOutOfRange extends Exception{
    //datafields
    private String range;

    //parameterised constructor
    public InputOutOfRange(String range){
        this.range = range;
    }

    public String getMessage(){
        if(range != null){
            return String.format("Input out of range. Should be between %s", getRange());
        }
        else{
            return "Input out of range.";
        }
    }

    public String getRange(){
        return this.range;
    }
}

interface Enrolment{
    //abstract by default
    public void addRecord(Record record);
    public void deleteRecord(int index);
    public Record getRecord(int index);
    public ArrayList<Record> getRecords();
    public void setRecords(ArrayList<Record> records);
}

class Subject implements Serializable{
    //dataprivate fields
    private String sName;
    private String code;
    private int credit;

    //default constructor
    public Subject(){
        this(" ", " ", 0);
    }

    //constructor with parameters
    public Subject(String code, String name, int creditPoint){
        this.sName = name;
        this.code = code;
        this.credit = creditPoint;
    }

    //methods
    public String getName(){
        return sName;
    }

    public String getCode(){
        return code;
    }

    public int getCredit(){
        return credit;
    }

    public boolean isSame(Subject subject){
        if (subject.getCode().equals(this.getCode())){
            return true;
        }
        else{
            return false;
        }
    }

    //implementing clone interface
    public Subject clone() throws CloneNotSupportedException{
        return (Subject) super.clone();
    }

    public String toString(){
        String printOut = getCode() +  " (" +  getName() + ", " + Integer.toString(getCredit()) + "pt)";
        return printOut;
    }
}

abstract class Student implements Enrolment, Cloneable{
    //datafields
    private String stName;
    private final String DOB;
    private ArrayList<Record> records;
    private final int STNUM;
    private String sGender;

    //constructors
    public Student(){
        this("x", "x", "x", 0);
    }

    public Student(String name, String DOB, String sGender, int STUDENTID){
        this.stName = name;
        this.DOB = DOB;
        this.sGender = sGender;
        this.STNUM = STUDENTID;
        this.records = new ArrayList<Record>();
    }

    //methods
    public String getSName(){
        return stName;
    }

    public void setSName(String stName){
        this.stName = stName;
    }

    public String getDOB(){
        return DOB;
    }

    public int getsNumberM(){
        return STNUM;
    }

    public String getsGender(){
        return sGender;
    }

    public void setsGender(String sGender){
        this.sGender = sGender;
    }

    public String toString(){
        return String.format("Student: %s (%s, %s, %s)\n",
            getSName(), Integer.toString(getsNumberM()), getsGender(), getDOB());
    }

    //implementing interface methods
    public void addRecord(Record record){
        records.add(record);
    }

    public void deleteRecord(int index){
        records.remove(index);
    }

    public Record getRecord(int index){
        return records.get(index);
    }

    public ArrayList<Record> getRecords(){
        return records;
    }

    public void setRecords(ArrayList<Record> records){
        this.records = records;
    }

    //implementing cloneabe method
    @Override
    public Student clone() throws CloneNotSupportedException{
        Student studentClone = (Student) super.clone();
        studentClone.setRecords((ArrayList<Record>)this.getRecords().clone());
        return studentClone;
    }

    //empty abstract methods
    public abstract void setExpectedCompletion(String date);
    public abstract String getExpectedCompletion();
}
```

```
class Undergraduate extends Student{
    //datafields
    private String bachelorCompletion;

    //constructors
    public Undergraduate(){
        super();
        this.bachelorCompletion ="x";
    }

    public Undergraduate(String name, String DOB, int STUDENTID, String sGenderder, String expectedCompletionSession){
        super(name, DOB, sGenderder, STUDENTID);
        this.bachelorCompletion = expectedCompletionSession;
    }

    //methods
    public String getExpectedCompletion(){
        return bachelorCompletion;
    }

    public void setExpectedCompletion(String expectedCompletionSession){
        this.bachelorCompletion = expectedCompletionSession;
    }

    @Override
    public String toString(){
        String printOut = super.toString();
        for(int i = 0; i < getRecords().size(); i++){
            Record record = getRecord(i);
            printOut += String.format("\nCourse Name: %s, (%s) \n\n%s",
            record.getCourseName(), record.getStatus(), record);
        }

        return printOut;
    }
}

class Postgraduate extends Student{
    //datafields
    private String bachelorCompletion;
    private String masterCompletion;

    //constructors
    public Postgraduate(){
        super();
        this.bachelorCompletion ="x";
        this.masterCompletion = "x";
    }

    public Postgraduate(String name, String DOB, int STUDENTID, String sGenderder,
    String bachelorCompletion, String masterCompletion){
        super(name, DOB, sGenderder, STUDENTID);
        this.bachelorCompletion = bachelorCompletion;
        this.masterCompletion = masterCompletion;
    }

    //methods
    public String getExpectedCompletion(){
        return masterCompletion;
    }

    public void setExpectedCompletion(String expectedCompletionSession){
        this.masterCompletion = expectedCompletionSession;
    }
    public String getBachelorCompletion(){
        return bachelorCompletion;
    }

    public void setBachelorCompletion(String bachelorCompletion){
        this.bachelorCompletion = bachelorCompletion;
    }

    @Override
    public String toString(){
        String printOut = super.toString();
        for(int i = 0; i < getRecords().size(); i++){
            Record record = getRecord(i);
            printOut += String.format("\n\nCourse Name: %s, (%s) \n\n%s",
            record.getCourseName(), record.getStatus(), record);
        }

        printOut += "\nExpected Master Graduation: " + this.getExpectedCompletion();
        // printOut += "\n\nBachelor was received: " + getBachelorCompletion();

        return printOut;
    }
}

class Record implements Cloneable{
    //datafields
    private final String CNAME;
    private ArrayList<Subject> eCores;
    private Major eMajor;
    private ArrayList<Subject> eElectives;
    private int totalCredit;
    private Status status;

    public enum Status {ACTIVE, COMPLETE, NA}

    //default constructor
    public Record(){
        this("default");
    }

    //parameterised constructor
    public Record(String CNAME){
        this.CNAME = CNAME;
        this.eCores = new ArrayList<Subject>();
        this.eMajor = new Major();
        this.eElectives = new ArrayList<Subject>();
        this.totalCredit = 0;
        this.status = Status.NA;
    }

    //methods
    public void enrolCores(ArrayList<Subject> subjects){
        //maintains up to date credit points
        for(int i=0; i < subjects.size(); i++ ){
            this.totalCredit += subjects.get(i).getCredit();
        }

        //assigns core subjects
        this.eCores = subjects;
    }

    public void enrolMajor(Major major){
        //mtaintains up to date credit point count
        this.totalCredit += major.getCredit();
        this.eMajor = major;
    }
```

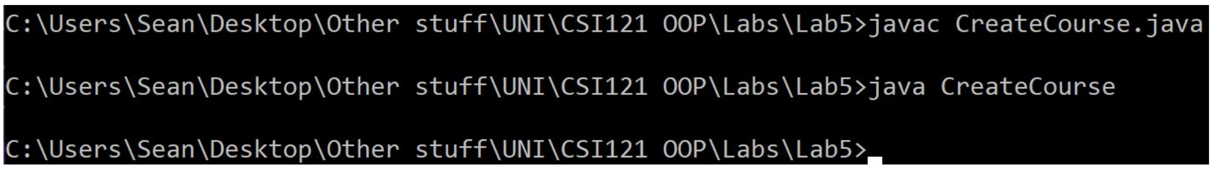
```
public void enrolElective(Subject subject){
    this.eElectives.add(subject);
    this.totalCredit += subject.getCredit();
}

public void enrolElectives(ArrayList<Subject> subjects){
    //maintains up to date credit points
    for(int i=0; i < subjects.size(); i++ ){
        this.totalCredit += subjects.get(i).getCredit();
    }

    //assigns core subjects
    this.eCores = subjects;
}

public int getTotalCredit(){
    return totalCredit;
}

public void setTotalCredit(int credit){
    this.totalCredit = credit;
}

/*calculates number of electives left to reach minimum credit
points for selected course*/
public int howManyElectivesLeft(Course course){
    if(getTotalCredit() < course.getCCredit()){
        return ((course.getCCredit() - getTotalCredit())/6);
    }
    return 0;
}

//predicate method(just checks truth)
public boolean isEnrolled(Subject name){
    if(eCores.contains(name)){
        return true;
    }
    else if(eElectives.contains(name)){
        return true;
    }
    else if(eMajor.isIncluded(name)){
        return true;
    }
    return false;
}

public Status getStatus(){
    return this.status;
}

public void setStatus(Status status){
    this.status = status;
}

public String toString(){
    String s = "";

    s+="Cores: \n";
    for(int i = 0; i < eCores.size(); i++)
        s+=eCores.get(i)+"\n";

    s+="\n";

    s+="Major: " + eMajor;
```

```
        s+="\n";

        s+="Electives: \n";
        for(int i = 0; i < eElectives.size(); i++)
            s+=eElectives.get(i)+"\n";

        s+="-----------------\n";

        s+="Total Enrolled Credit: "+ getTotalCredit() +"pt";

        return s;
    }

    public String getCourseName(){
        return CNAME;
    }

    //implementing cloneable interface
    public Record clone() throws CloneNotSupportedException{
        Record recordClone = (Record) super.clone();
        //clears credit points
        recordClone.setTotalCredit(0);

        //these methods all add relevent credit points
        recordClone.enrolMajor((Major)this.eMajor.clone());
        recordClone.enrolElectives((ArrayList<Subject>)this.eElectives.clone());
        recordClone.enrolCores((ArrayList<Subject>)this.eCores.clone());

        return recordClone;
    }
}

class Major implements Cloneable, Serializable{
    private String mName;
    private ArrayList<Subject> mCores;
    private int credit = 0;

    //default constructor
    public Major(){
        this("default");
    }

    //constructor with 1 parameter
    public Major(String name){
        this.mName = name;
        this.mCores = new ArrayList<Subject>();
    }

    //constructor with 2 parameters
    public Major(String name, Subject[] subjectList){
        this.mName = name;
        this.mCores = new ArrayList<Subject>();
        addMCores(subjectList);
    }

    public String getMName(){
        return mName;
    }

    public void setMName(String name){
        this.mName = name;
    }

    public int getCredit(){
        int credit = 0;
        for(int i = 0; i < mCores.size(); i++ ){
            credit += mCores.get(i).getCredit();
        }

        return credit;
    }

    //predicate method
    public boolean isIncluded(Subject name){
        ArrayList<Subject> subjects = this.getMCores();

        for(int i = 0; i < subjects.size(); i++ ){
            if(subjects.get(i).getName() == name.getName()){
                return true;
            }
        }
        return false;
    }

    public void addMCores(Subject[] subjects){
        for (Subject subject : subjects){
            this.mCores.add(subject);
        }
    }

    public ArrayList<Subject> getMCores(){
        return mCores;
    }

    //implementing clone interface
    @Override
    public Major clone() throws CloneNotSupportedException{
        return (Major) super.clone();
    }

    @Override
    public String toString(){
        String printOut = getMName() + "\n";
        ArrayList<Subject> subjects = getMCores();

        for(int i = 0; i < subjects.size(); i++ ){
            printOut += subjects.get(i) + "\n";
        }

        return printOut;
    }
}

class Course implements Serializable{
    private String cName;
    private ArrayList<Subject> cores;
    private ArrayList<Major> majors;
    private ArrayList<Subject> electives;
    private int cCredit;

    public Course(){
        this("default", 0);
    }

    public Course(String cName, int cCredit){
        this.cName = cName;
        this.cores = new ArrayList<Subject>();
        this.majors = new ArrayList<Major>();
        this.electives = new ArrayList<Subject>();
        this.cCredit = cCredit;
    }
```

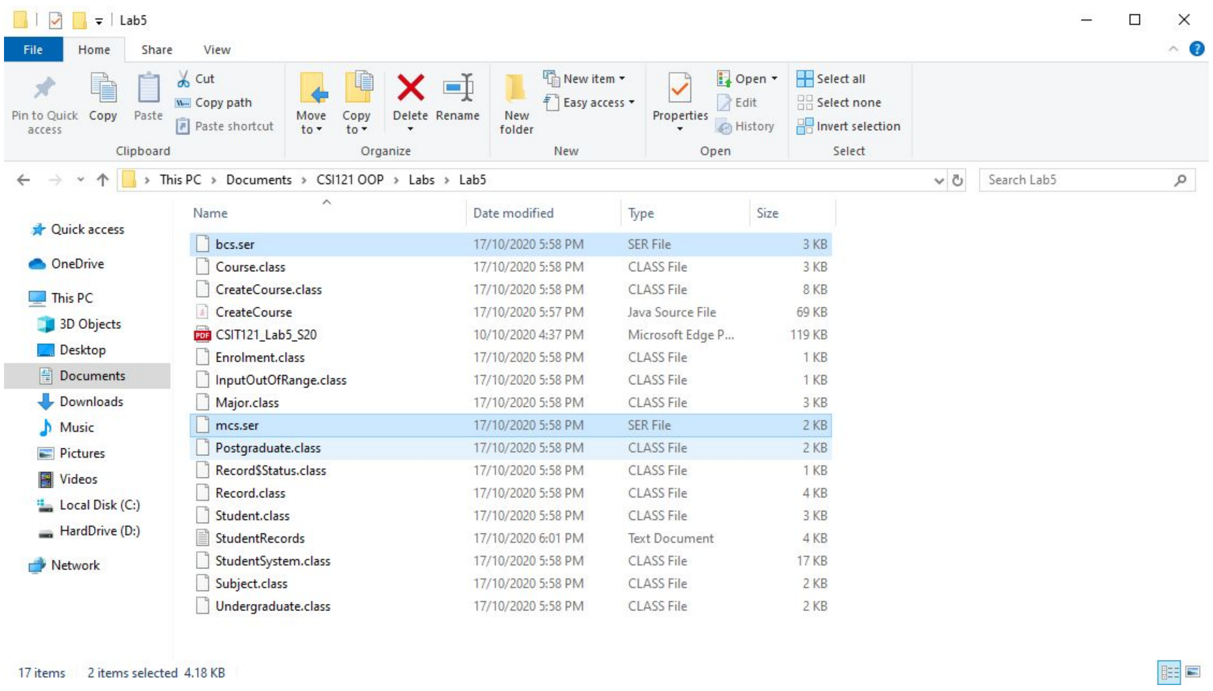
```
public void addCores(Subject[] subjects){
    for (Subject subject : subjects){
        this.cores.add(subject);
    }
}

public void addMajors(Major[] majors){
    for (Major major : majors){
        this.majors.add(major);
    }
}

public void addElectives(Subject[] electives){
    for (Subject elective : electives){
        this.electives.add(elective);
    }
}

//display all electives
public void printElectives(){
    System.out.println("Elective Subjects:");
    for(int i = 0; i < getElectives().size(); i++){
        System.out.println(getElectives().get(i));
    }
    System.out.println();
}

public ArrayList<Subject> getCores(){
    return cores;
}

public ArrayList<Subject> getElectives(){
    return electives;
}

public ArrayList<Major> getMajors(){
    return majors;
}

public String toString(){
    String s = "Course: "+getCName() + "\n";

    s+="Cores: \n";
    for(int i = 0; i < getCores().size(); i++)
        s+=getCores().get(i)+"\n";

    s+="\n";

    s+="Majors: \n";
    for(int i = 0; i < getMajors().size(); i++){
        s+=getMajors().get(i) + "\n";
    }

    s+="\n";

    s+="Electives: \n";
    for(int i = 0; i < getElectives().size(); i++)
        s+=getElectives().get(i)+"\n";

    s+="-----------------\n";

    s+="Required Total Credit: "+ getCCredit() +"pt";

    return s;
}
```

```
    public String getCName(){
        return cName;
    }

    public int getCCredit(){
        return cCredit;
    }
}

class InputOutOfRange extends Exception{
    //datafields
    private String range;

    //parameterised constructor
    public InputOutOfRange(String range){
        this.range = range;
    }

    public String getMessage(){
        if(range != null){
            return String.format("Input out of range. Should be between %s", getRange());
        }
        else{
            return "Input out of range.";
        }
    }

    public String getRange(){
        return this.range;
    }
}
```

Screen shots and successful compilation:



```
C:\Users\Sean\Desktop\Other stuff\UNI\CSI121 OOP\Labs\Lab5>javac CreateCourse.java

C:\Users\Sean\Desktop\Other stuff\UNI\CSI121 OOP\Labs\Lab5>java CreateCourse

C:\Users\Sean\Desktop\Other stuff\UNI\CSI121 OOP\Labs\Lab5>
```



Name	Date modified	Type	Size
bcs.ser	17/10/2020 5:58 PM	SER File	3 KB
Course.class	17/10/2020 5:58 PM	CLASS File	3 KB
CreateCourse.class	17/10/2020 5:58 PM	CLASS File	8 KB
CreateCourse	17/10/2020 5:57 PM	Java Source File	69 KB
CSIT121_Lab5_S20	10/10/2020 4:37 PM	Microsoft Edge P...	119 KB
Enrolment.class	17/10/2020 5:58 PM	CLASS File	1 KB
InputOutOfRange.class	17/10/2020 5:58 PM	CLASS File	1 KB
Major.class	17/10/2020 5:58 PM	CLASS File	3 KB
mcs.ser	17/10/2020 5:58 PM	SER File	2 KB
Postgraduate.class	17/10/2020 5:58 PM	CLASS File	2 KB
Record$Status.class	17/10/2020 5:58 PM	CLASS File	1 KB
Record.class	17/10/2020 5:58 PM	CLASS File	4 KB
Student.class	17/10/2020 5:58 PM	CLASS File	3 KB
StudentRecords	17/10/2020 6:01 PM	Text Document	4 KB
StudentSystem.class	17/10/2020 5:58 PM	CLASS File	17 KB
Subject.class	17/10/2020 5:58 PM	CLASS File	2 KB
Undergraduate.class	17/10/2020 5:58 PM	CLASS File	2 KB

17 items 2 items selected 4.18 KB

-the bcs.ser and mcs.ser files located in the same directory

TASK 2:

Note: (the other classes that aren't the primary class is included in both programs)

StudentSystem class:

```
/*------------------------------------------------------
My name: Sean Overton
My student number: 6421490
My course code: CSIT121
My email address: so412@uowmail.edu.au
Lab: #5
-------------------------------------------------------*/

import java.util.ArrayList;
import java.util.Scanner;
import java.util.InputMismatchException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.io.*;
import java.util.Formatter;
import java.util.FormatterClosedException;
import java.lang.SecurityException;

class StudentSystem{
  private static ArrayList<Student> students = new ArrayList<Student>();
  private static ObjectInputStream fileInput;
  private static Formatter output;

  public static void main(String[] args) throws CloneNotSupportedException{
    //deserialise bachelor course object
    openFile("bcs.ser");
    Course compSci = readRecords();
    closeFile();

    //deserialise master course object
    openFile("mcs.ser");
    Course mcs = readRecords();
```

```
closeFile();

System.out.println("(Enrol first student to Bachelor of Computer Science Course)");
System.out.println("Welcome to enrol into Bachelor of Computer Science.");
System.out.println("The Course structure is as follows: ");
System.out.println("------------------------------------------");

//i) to display the entire course structure to the student;
System.out.println(compSci);

/*ii) to ask the student to input the personal information
(by using the Scanner); */
System.out.println("The following information is required to complete student enrolment:");

//boolean created for while loops for try statements on inputs
boolean inputsRequired = true;

//used to gather input from user/student with validation checks
Scanner input = new Scanner(System.in);
String sName = "default";
String sGender = "default";
String sDOB = "default";
int sNumber = 0;

System.out.print("Please input your full name: ");

do{
  try{
    sName = input.nextLine();

    if(sName.equals("")){
      throw new InputMismatchException("Blank input not accepted.");
    }

    //if no exception thrown
    inputsRequired = false;
  }
  catch(InputMismatchException inputMismatchException){
    //clears buffer
    input.nextLine();
    if(inputMismatchException.getMessage() != null){
      System.err.println(inputMismatchException.getMessage());
    }
    else{
      System.err.println("Invalid datatype.");
    }
    System.err.println("Please enter valid full-name:");
  }
  catch(Exception e){
    //clears buffer
    input.nextLine();

    //catches all other unforseen exceptions
    System.err.printf("Error thrown: %s", e);
    System.err.println("Please enter valid full-name:");
  }
}while(inputsRequired);

inputsRequired = true;

System.out.print("Please input your student number: ");

String number = "";

do{
  try{
        number = input.nextLine();
        sNumber = Integer.parseInt(number);

        if(sNumber < 10000 || sNumber >=100000){
            throw new InputOutOfRange("10000-99999");
        }
        //if no exception thrown
        inputsRequired = false;
    }
    catch(NumberFormatException e){
        //clears buffer
        input.nextLine();

        if(number.equals("")){
            System.err.println("Blank input not accepted.");
        }

        System.err.println("Invalid datatype.");
        System.err.println("Please enter valid integer input.");
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }
        System.err.println("Please enter valid Student number:");
    }
    catch(InputOutOfRange inputOutOfRange){
        //clears buffer
        input.nextLine();

        System.err.println(inputOutOfRange.getMessage());
        System.err.println("Please enter valid Student number:");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();

        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s", e);
        System.err.println("Please enter valid Student number:");
    }
}while(inputsRequired);

inputsRequired = true;

System.out.print("Please input your gender: ");

do{
    try{
        sGender = input.nextLine();

        if(sGender.equals("")){
            throw new InputMismatchException("Blank input not accepted.");
        }
        else{
            //asertion used to enforce input constraints
            sGender = sGender.toLowerCase();

            assert(sGender.equals("male") || sGender.equals("female")) : "Please enter a valid gender. Not " + sGender;
            inputsRequired = false;
        }
    }
    catch(AssertionError assertionError){
        System.err.printf("Assertion error caught. Please enter 'male' OR 'female'. \n%s: ", assertionError.getMessage());
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }
        System.err.println("Please enter a valid gender:");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();

        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s", e);
        System.err.println("Please enter a valid gender:");
    }
}while(inputsRequired);

System.out.print("Please input your date of birth (dd/mm/yyyy): ");

inputsRequired = true;

do{
    try{
        sDOB = input.nextLine();

        if(sDOB.equals("")){
            throw new InputMismatchException("Blank input not accepted.");
        }
        String[] subjects = sDOB.split("/");

        boolean isValid = false;
        int y = Integer.parseInt(subjects[2]);
        int m = Integer.parseInt(subjects[1]);

        /*arrayList contains method requires integer object instead of
        primitive type int*/
        Integer intM = new Integer(m);
        int d = Integer.parseInt(subjects[0]);

        ArrayList<Integer> monthArray31 = new ArrayList<Integer>();
        monthArray31.add(1);
        monthArray31.add(3);
        monthArray31.add(5);
        monthArray31.add(7);
        monthArray31.add(8);
        monthArray31.add(10);
        monthArray31.add(12);
        ArrayList<Integer> monthArray30 = new ArrayList<Integer>();
        monthArray30.add(4);
        monthArray30.add(6);
        monthArray30.add(9);
        monthArray30.add(11);

        boolean isLeapYear = false;
        //checks whether the year is a leap year
        if (y % 4 != 0)
            isLeapYear = false;
            // not divisible by 4
        else if (y % 100 != 0)
            isLeapYear = true;
            // not divisible by 100
        else if (y % 400 != 0)
            isLeapYear = false;
            // not divisible by 400
        else
            isLeapYear = true;

        //checks if leap year feb has valid day
        if (isLeapYear && m == 2 && d <= 29 && d > 0 && y > 999){
            isValid = true;
        }
        else if (d > 0 && y > 999){
            //otherwise checks if any month has valid day
            if (monthArray31.contains(intM) && d <= 31){
                isValid = true;
            }
            else if (monthArray30.contains(intM) && d <= 30){
                isValid = true;
            }
            else if (m == 2 && d <= 28){
                isValid = true;
            }
        }

        if(!isValid){
            throw new InputMismatchException("The date entered has invalid numbers.");
        }

        inputsRequired = false;
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }

        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
    catch(ArrayIndexOutOfBoundsException e){
        //clears buffer
        input.nextLine();

        System.err.println("Invalid date format. Must be (dd/mm/yyyy): ");
        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
    catch(NumberFormatException numberFormatException){
        //clears buffer
        input.nextLine();

        System.err.println("Invalid number format.");
        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();
        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s\n", e);
        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
}while(inputsRequired);

/*ii) to enrol the first student to the Bachelor of Computer Science
course. Then the system will create the first student
object The system will automatically enrol the
student to all BCS core subjects and ask the student to select a major
and several elective subjects to
complete the enrolment.*/

//using above gathered information student object created
Student newStudent1 = new Undergraduate(sName, sDOB, sNumber, sGender, "Spring/2023");

/*Record object shall be created first based on the
first student's enrolment
information, then the Record object shall be added to the first
student's records list.*/
Record student1Record = new Record("Bachelor of Computer Science");

// //iii) to automatically enrol the student to all core subjects;
student1Record.enrolCores(compSci.getCores());

/*iv) to ask the student to select a major and automatically
enrol the student to all subjects of the major; */
System.out.println("Thanks for your information.");
System.out.println("In order to complete your enrolment, please select a major from the list:");
System.out.println();
System.out.println("1. Big Data");
System.out.println("2. Cyber Security");
System.out.println("3. Digital System Security");
System.out.println("4. Game and Mobile Development");
System.out.println("5. Software Engineering");
System.out.println();
System.out.print("Please input the index number before the major:");

inputsRequired = true;

/*required so it is initialised but in reality won't get past
do/while loop that will guarruntee a valid user input */
String majorChoice = "default";

do{
    try{
        majorChoice = input.nextLine();
        int m = Integer.parseInt(majorChoice);

        if(majorChoice.equals("")){
            throw new InputMismatchException("Blank input not accepted.");
        }

        if(m <= 0 || m > 5){
            throw new InputOutOfRange("1-5");
        }

        //if no exception thrown
        inputsRequired = false;
    }
    catch(InputOutOfRange inputOutOfRange){
        //clears buffer
        input.nextLine();

        System.err.println(inputOutOfRange.getMessage());
```

```
            System.err.println("Please enter valid integer input.");
        }
        catch(NumberFormatException e){
            //clears buffer
            input.nextLine();

            System.err.println("Invalid datatype.");
            System.err.println("Please enter valid integer input.");
        }
        catch(InputMismatchException inputMismatchException){
            //clears buffer
            input.nextLine();

            if(inputMismatchException.getMessage() != null){
                System.err.println(inputMismatchException.getMessage());
            }
            else{
                System.err.println("Invalid datatype.");
            }
            System.err.println("Please enter valid integer input.");
        }
        catch(Exception e){
            //clears buffer
            input.nextLine();

            //catches all other unforseen exceptions
            System.err.printf("Error thrown: %s", e);
            System.err.println("Please enter valid integer input.");
        }
    }while(inputsRequired);

    ArrayList<Major> majors = compSci.getMajors();

    System.out.println("You enrolled into:");
    switch(majorChoice){
        case "1":
            //big data major
            student1Record.enrolMajor(majors.get(0));
            System.out.println(majors.get(0));
            break;
        case "2":
            //cyber security
            student1Record.enrolMajor(majors.get(1));
            System.out.println(majors.get(1));
            break;
        case "3":
            //system security
            student1Record.enrolMajor(majors.get(2));
            System.out.println(majors.get(2));
            break;
        case "4":
            //game dev
            student1Record.enrolMajor(majors.get(3));
            System.out.println(majors.get(3));
            break;
        case "5":
            //software development
            student1Record.enrolMajor(majors.get(4));
            System.out.println(majors.get(4));
            break;
        default:
            student1Record.enrolMajor(majors.get(4));
            System.out.println("Default enrolment:");
            System.out.println(majors.get(4));
    }
```

```
/*v) to ask the student to select sufficient elective
subjects for reaching the minimal credit points requirement
of the course, i.e., 144 credit points; */
System.out.println();
System.out.println("In order to complete the enrolment, please select elective subjects from the list.");
System.out.println();

int electiveCount = student1Record.howManyElectivesLeft(compSci);

/*default case: it is outside of elective subjects because it
is a core subject. Therefore shouldn't cause any issues.*/
Subject subjectChoiceSub = compSci.getCores().get(1);

compSci.printElectives();

//required for correct exception handling
boolean matched = false;

ArrayList<Subject> electives = compSci.getElectives();

do{
    matched = false;
    //prompt for subject selection
    System.out.printf("Please select %s more elective subjects\n", electiveCount);

    try{
        String subjectChoiceStr = input.nextLine();
        String[] subjects = subjectChoiceStr.split(", ");

        //uses each subject from input
        for(int a = 0; a < subjects.length; a++){
            //locates subject object from string choice
            if(subjects[a].equals("")){
                throw new Exception("Blank input not accepted.");
            }

            for(int i = 0; i < electives.size(); i++){
                if(electives.get(i).getCode().equals(subjects[a])){
                    //assigned actual subject
                    subjectChoiceSub = electives.get(i);
                    matched = true;
                }
            }

            //if subject already enrolled
            if(!matched){
                throw new InputMismatchException(subjects[a]);
            }
            //if subject already enrolled
            else if(student1Record.isEnrolled(subjectChoiceSub)){
                throw new Exception("Already enrolled in subject.");
            }
            else{
                //add subject to elective subjects
                student1Record.enrolElective(subjectChoiceSub);
                electiveCount -= 1;
            }
            matched = false;
        }
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() !=null){
            System.err.printf("%s is an invalid subject code.\n", inputMismatchException.getMessage());
            }
            else{
                System.err.println("Invalid data type input.");
            }
            System.err.println("Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '");
        }
        catch(ArrayIndexOutOfBoundsException e){
            //clears buffer
            input.nextLine();

            System.err.println("Invalid subject code input: ");
            System.err.println("Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '");
        }
        catch(Exception e){
            //clears buffer
            input.nextLine();

            if(e.getMessage() !=null){
                System.err.println(e.getMessage());
            }
            else{
                System.err.printf("Error thrown: %s", e);
            }

            System.err.println("Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '");
        }
    //checks if course requirement is met
    }while(student1Record.howManyElectivesLeft(compSci) > 0);

    // automatically enrol student in course
    newStudent1.addRecord(student1Record);

    /*vi) once the student enrols sufficient subjects
    to make up at least minimum credit points enrolment
    is complete. */
    student1Record.setStatus(Record.Status.ACTIVE);

    System.out.println("(Enrol the second student into Master of Computer Science Course and add his/her undergraduate
record)");
    System.out.println("Welcome to enrol into Master of Computer Science.");
    System.out.println("The Course structure is as follows: ");
    System.out.println("------------------------------------------");
    //i) to display the entire course structure to the student;
    System.out.println(mcs);

    /*ii) to ask the student to input the personal information
    (by using the Scanner); */
    System.out.println();
    System.out.println("The following information is required to complete MASTER's student enrolment:");

    inputsRequired = true;

    System.out.print("Please enter your full-name: ");

    //use old scanner object to gather input from user/student
    do{
        try{
            sName = input.nextLine();

            if(sName.equals("")){
                throw new InputMismatchException("Blank input not accepted.");
            }

            //if no exception thrown
            inputsRequired = false;
        }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();
        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }
        System.err.println("Please enter valid full-name:");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();

        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s", e);
        System.err.println("Please enter valid full-name:");
    }
}while(inputsRequired);

inputsRequired = true;

System.out.print("Please input your student number: ");

do{
    try{
        number = input.nextLine();
        sNumber = Integer.parseInt(number);

        if(sNumber < 10000 || sNumber >=100000){
            throw new InputOutOfRange("10000-99999");
        }
        //if no exception thrown
        inputsRequired = false;
    }
    catch(NumberFormatException e){
        //clears buffer
        input.nextLine();

        if(number.equals("")){
            System.err.println("Blank input not accepted.");
        }

        System.err.println("Invalid datatype.");
        System.err.println("Please enter valid integer input.");
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }
        System.err.println("Please enter valid Student number:");
    }
    catch(InputOutOfRange inputOutOfRange){
        //clears buffer
        input.nextLine();

        System.err.println(inputOutOfRange.getMessage());
        System.err.println("Please enter valid Student number:");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();

        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s", e);
        System.err.println("Please enter valid Student number:");
    }
}while(inputsRequired);

inputsRequired = true;

System.out.print("Please input your gender: ");

do{
    try{
        sGender = input.nextLine();

        if(sGender.equals("")){
            throw new InputMismatchException("Blank input not accepted.");
        }
        else{
            //asertion used to enforce input constraints
            sGender = sGender.toLowerCase();

            assert(sGender.equals("male") || sGender.equals("female")) : "Please enter a valid gender. Not " +  sGender;

            inputsRequired = false;
        }
    }
    catch(AssertionError assertionError){
        System.err.printf("Assertion error caught. Please enter 'male' OR 'female'. \n%s: ", assertionError.getMessage());
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }
        System.err.println("Please enter a valid gender:");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();

        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s", e);
        System.err.println("Please enter a valid gender:");
    }
}while(inputsRequired);

System.out.print("Please input your date of birth (dd/mm/yyyy): ");

inputsRequired = true;

do{
    try{
        sDOB = input.nextLine();

        if(sDOB.equals("")){
            throw new InputMismatchException("Blank input not accepted.");
        }
        String[] subjects = sDOB.split("/");

        boolean isValid = false;
        int y = Integer.parseInt(subjects[2]);
        int m = Integer.parseInt(subjects[1]);

        /*arrayList contains method requires integer object instead of
        primitive type int*/
        Integer intM = new Integer(m);
        int d = Integer.parseInt(subjects[0]);

        ArrayList<Integer> monthArray31 = new ArrayList<Integer>();
        monthArray31.add(1);
        monthArray31.add(3);
        monthArray31.add(5);
        monthArray31.add(7);
        monthArray31.add(8);
        monthArray31.add(10);
        monthArray31.add(12);
        ArrayList<Integer> monthArray30 = new ArrayList<Integer>();
        monthArray30.add(4);
        monthArray30.add(6);
        monthArray30.add(9);
        monthArray30.add(11);

        boolean isLeapYear = false;

        //checks whether the year is a leap year
        if (y % 4 != 0)
            isLeapYear = false;
            // not divisible by 4
        else if (y % 100 != 0)
            isLeapYear = true;
            // not divisible by 100
        else if (y % 400 != 0)
            isLeapYear = false;
            // not divisible by 400
        else
            isLeapYear = true;

        //checks if leap year feb has valid day
        if (isLeapYear && m == 2 && d <= 29 && d > 0 && y > 999){
            isValid = true;
        }
        else if (d > 0 && y > 999){
            //otherwise checks if any month has valid day
            if (monthArray31.contains(intM) && d <= 31){
                isValid = true;
            }
            else if (monthArray30.contains(intM) && d <= 30){
                isValid = true;
            }
            else if (m == 2 && d <= 28){
                isValid = true;
            }
        }

        if(!isValid){
            throw new InputMismatchException("The date entered has invalid numbers.");
        }

        inputsRequired = false;
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
```

```
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }

        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
    catch(ArrayIndexOutOfBoundsException e){
        //clears buffer
        input.nextLine();

        System.err.println("Invalid date format. Must be (dd/mm/yyyy): ");
        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
    catch(NumberFormatException numberFormatException){
        //clears buffer
        input.nextLine();

        System.err.println("Invalid number format.");
        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
    catch(Exception e){
        //clears buffer
        input.nextLine();

        //catches all other unforseen exceptions
        System.err.printf("Error thrown: %s\n", e);
        System.err.println("Please enter a valid date(dd/mm/yyyy):");
    }
}while(inputsRequired);

inputsRequired = true;

System.out.print("Please enter the time (Session/year) you received your bachelor degree: ");

String bachelorCompletion  = "";

do{
    try{
        bachelorCompletion = input.nextLine();

        if(bachelorCompletion.equals("")){
            throw new InputMismatchException("Blank input not accepted.");
        }

        inputsRequired = false;
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() != null){
            System.err.println(inputMismatchException.getMessage());
        }
        else{
            System.err.println("Invalid datatype.");
        }

        System.err.println("Please enter a valid session (session/year):");
    }
    catch(Exception e){
        //clears buffer
```

```
            input.nextLine();

            //catches all other unforseen exceptions
            System.err.printf("Error thrown: %s\n", e);
            System.err.println("Please enter a valid session (session/year):");
        }
    }while(inputsRequired);

    System.out.println("(It is assumed that Amy completed Bachelor of Computer Science with the exact same subjects as
Bob. \nA same BCS record is added to Amy's enrolment record automatically.)");
    System.out.println();

    //using above gathered information student object created
    Student newStudent2 = new Postgraduate(sName, sDOB, sNumber, sGender, bachelorCompletion, "Spring/2022");

    /*Record object shall be created first based on the
    first student’s enrolment
    information, then the Record object shall be added to the first
    student’s records list.*/
    Record student2Record = new Record("Master of Computer Science");

    // automatically enrol student in course of previous student
    Record student1RecordClone = student1Record.clone();

    /*change status and checks that deep clone implementation
     has worked*/
    student1RecordClone.setStatus(Record.Status.COMPLETE);

    newStudent2.addRecord(student1RecordClone);

    newStudent2.addRecord(student2Record);

    // //iii) to automatically enrol the student to all core subjects;
    student2Record.enrolCores(mcs.getCores());

    /*iv) to ask the student to select a major and automatically
    enrol the student to all subjects of the major; */
    System.out.println("Thanks for your information. You have been enrolled into a Master of Computer Science.");
    System.out.println("In order to complete your enrolment, please select a major from the list:");
    System.out.println();
    System.out.println("1. Intelligent Systems");
    System.out.println("2. Machine Learning and Big data");
    System.out.println("3. Network and Information Security");
    System.out.println("4. Software Engineering");
    System.out.println();
    System.out.print("Please input the index number before the major:");

    inputsRequired = true;

    do{
        try{
            majorChoice = input.nextLine();
            int m = Integer.parseInt(majorChoice);

            if(majorChoice.equals("")){
                throw new InputMismatchException("Blank input not accepted.");
            }

            if(m <= 0 || m > 4){
                throw new InputOutOfRange("1-4");
            }

            //if no exception thrown
            inputsRequired = false;
        }
        catch(InputOutOfRange inputOutOfRange){
            //clears buffer
            input.nextLine();

            System.err.println(inputOutOfRange.getMessage());
            System.err.println("Please enter valid integer input.");
        }
        catch(NumberFormatException e){
            //clears buffer
            input.nextLine();

            System.err.println("Invalid datatype.");
            System.err.println("Please enter valid integer input.");
        }
        catch(InputMismatchException inputMismatchException){
            //clears buffer
            input.nextLine();

            if(inputMismatchException.getMessage() != null){
                System.err.println(inputMismatchException.getMessage());
            }
            else{
                System.err.println("Invalid datatype.");
            }
            System.err.println("Please enter valid integer input.");
        }
        catch(Exception e){
            //clears buffer
            input.nextLine();

            //catches all other unforseen exceptions
            System.err.printf("Error thrown: %s", e);
            System.err.println("Please enter valid integer input.");
        }
    }while(inputsRequired);

    majors = mcs.getMajors();

    System.out.println("You enrolled into:");
    switch(majorChoice){
        case "1":
            //intelligent systems
            student2Record.enrolMajor(majors.get(0));
            System.out.println(majors.get(0));
            break;
        case "2":
            //machine learning and big data
            student2Record.enrolMajor(majors.get(1));
            System.out.println(majors.get(1));
            break;
        case "3":
            //network and information security
            student2Record.enrolMajor(majors.get(2));
            System.out.println(majors.get(2));
            break;
        case "4":
            //software eng
            student2Record.enrolMajor(majors.get(3));
            System.out.println(majors.get(3));
            break;
        default:
            student2Record.enrolMajor(majors.get(3));
            System.out.println("Default enrolment:");
            System.out.println(majors.get(3));
    }

    /*v) to ask the student to select sufficient elective
```

```
subjects for reaching the minimal credit points requirement
of the course, i.e., 144 credit points; */
System.out.println();
System.out.println("In order to complete the enrolment, please select elective subjects from the list.");
System.out.println();

electiveCount = student2Record.howManyElectivesLeft(mcs);

/*default case: it is outside of elective subjects because it
is a core subject. Therefore shouldn't cause any issues.*/
subjectChoiceSub = mcs.getCores().get(1);

mcs.printElectives();

electives = mcs.getElectives();

do{
    matched = false;

    //prompt for subject selection
    System.out.printf("Please select %s more elective subjects\n", electiveCount);

    try{
        String subjectChoiceStr = input.nextLine();
        String[] subjects = subjectChoiceStr.split(", ");

            //uses each subject from input
        for(int a = 0; a < subjects.length; a++){
            //locates subject object from string choice

            //blank input check first
            if(subjects[a].equals("")){
                throw new Exception("Blank input not accepted.");
            }

            matched = false;

            for(int i = 0; i < electives.size(); i++){
                if(electives.get(i).getCode().equals(subjects[a])){
                    //assigned actual subject
                    subjectChoiceSub = electives.get(i);
                    matched = true;
                }
            }
            //if subject already enrolled
            if(!matched){
                throw new InputMismatchException(subjects[a]);
            }
            //if subject already enrolled
            else if(student2Record.isEnrolled(subjectChoiceSub)){
                throw new Exception("Already enrolled in subject.");
            }
            else{
                //add subject to elective subjects
                student2Record.enrolElective(subjectChoiceSub);
                electiveCount -= 1;
            }
        }
    }
    catch(InputMismatchException inputMismatchException){
        //clears buffer
        input.nextLine();

        if(inputMismatchException.getMessage() !=null){
            System.err.printf("%s is an invalid subject code.\n", inputMismatchException.getMessage());
        }
```

```
            else{
                System.err.println("Invalid data type input.");
            }
            System.err.println("Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '");
        }
        catch(ArrayIndexOutOfBoundsException e){
            //clears buffer
            input.nextLine();

            System.err.println("Invalid subject code input: ");
            System.err.println("Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '");
        }
        catch(Exception e){
            //clears buffer
            input.nextLine();

            if(e.getMessage() !=null){
                System.err.println(e.getMessage());
            }
            else{
                System.err.printf("Error thrown: %s", e);
            }

            System.err.println("Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '");
        }
    //checks if course requirement is met
    }while(student2Record.howManyElectivesLeft(mcs) > 0);

    student2Record.setStatus(Record.Status.ACTIVE);

    /*E)to add the Undergraduate student and the Postgraduate student objects to the static students
    ArrayList;*/
    students.add(newStudent1);
    students.add(newStudent2);

    /*f) to use a for loop to display all students' enrolment information in the students ArrayList.
    For each student, you shall display the student's personal information first, and then all
    enrolment records of the student. The Undergraduate student shall have one enrolment
    record and the Postgraduate student shall have two enrolment records.*/
    System.out.println("Congratulations you have successfully completed enrolment into Master of Computer Science.");
    System.out.println("Below is the current enrolments: ");

    openTextFile();
    saveRecords();
    closeTextFile();
}

public static void openFile(String fileName){
    try{
        fileInput = new ObjectInputStream(Files.newInputStream(Paths.get(fileName)));
    }
    catch(IOException ioException){
        System.err.println("Error opening file. Terminating.");
        System.exit(1); //terminates program?
    }
}

public static void closeFile(){
    if(output != null){
        output.close();
    }
}

public static Course readRecords(){
    try{
        //reads object from file and casts as course
```

```
            Course course = (Course)fileInput.readObject();

            return course;
        }
        catch(EOFException eofException){
            System.err.printf("%No more Records%n");
        }
        catch(ClassNotFoundException classNotFoundException){
            System.err.println("Invalid object type. Terminating.");
        }
        catch(IOException ioException){
            System.err.println("Error reading file. Terminating.");
        }

        return null;
    }

    public static void openTextFile(){
        try{
            output = new Formatter("student.txt");
        }
        catch(SecurityException e){
            System.err.println("Write Permission Denied. Terminating.");
            System.exit(1); //terminates
        }
        catch(FileNotFoundException e){
            System.err.println("Error opening file. Terminating.");
            System.exit(1); //terminates
        }
    }

    public static void saveRecords(){
        try{
            for(int i = 0; i < students.size(); i++){
                //saves strings to text file

                Student student = students.get(i);
                /*g) to use the instanceof and down casting in the for loop above to show the bachelor
                completion time for the Postgraduate student (calling the specific getBachelorCompletion()
                method defined in the Postgraduate class).*/
                if(student instanceof Undergraduate){
                    output.format(student.toString());
                }
                else if(student instanceof Postgraduate){
                    output.format("\n\n" + student.toString() + "\n");

                    //downcasting so we can call specific method from subclass
                    Postgraduate gradStudent = (Postgraduate) student;

                    output.format("\nBachelor Degree was received (using downcasting): " + gradStudent.getBachelorCompletion());
                }
            }
        }
        catch(FormatterClosedException e){
            System.err.println("Formatter closed. Terminating.");
        }
    }

    public static void closeTextFile(){
        if(output != null){
            output.close();
        }
    }
}

interface Enrolment{
    //abstract by default
    public void addRecord(Record record);
    public void deleteRecord(int index);
    public Record getRecord(int index);
    public ArrayList<Record> getRecords();
    public void setRecords(ArrayList<Record> records);
}

class Subject implements Serializable{
    //dataprivate fields
    private String sName;
    private String code;
    private int credit;

    //default constructor
    public Subject(){
        this(" ", " ", 0);
    }

    //constructor with parameters
    public Subject(String code, String name, int creditPoint){
        this.sName = name;
        this.code = code;
        this.credit = creditPoint;
    }

    //methods
    public String getName(){
        return sName;
    }

    public String getCode(){
        return code;
    }

    public int getCredit(){
        return credit;
    }

    public boolean isSame(Subject subject){
        if (subject.getCode().equals(this.getCode())){
            return true;
        }
        else{
            return false;
        }
    }

    //implementing clone interface
    public Subject clone() throws CloneNotSupportedException{
        return (Subject) super.clone();
    }

    public String toString(){
        String printOut = getCode() +  " (" +  getName() + ", " + Integer.toString(getCredit()) + "pt)";
        return printOut;
    }
}

abstract class Student implements Enrolment, Cloneable{
    //datafields
    private String stName;
    private final String DOB;
    private ArrayList<Record> records;
    private final int STNUM;
    private String sGender;
```

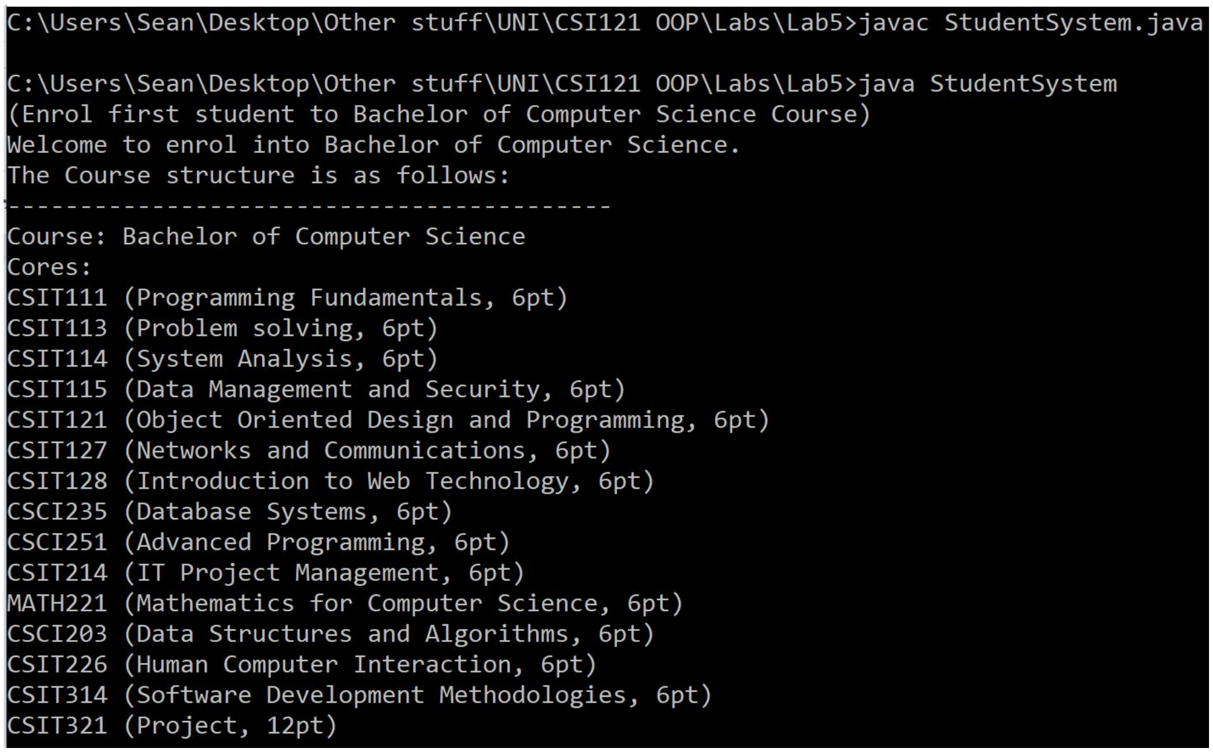
```
//constructors
public Student(){
    this("x", "x", "x", 0);
}

public Student(String name, String DOB, String sGender, int STUDENTID){
    this.stName = name;
    this.DOB = DOB;
    this.sGender = sGender;
    this.STNUM = STUDENTID;
    this.records = new ArrayList<Record>();
}

//methods
public String getSName(){
    return stName;
}

public void setSName(String stName){
    this.stName = stName;
}

public String getDOB(){
    return DOB;
}

public int getsNumberM(){
    return STNUM;
}

public String getsGender(){
    return sGender;
}

public void setsGender(String sGender){
    this.sGender = sGender;
}

public String toString(){
    return String.format("Student: %s (%s, %s, %s)\n",
        getSName(), Integer.toString(getsNumberM()), getsGender(), getDOB());
}

//implementing interface methods
public void addRecord(Record record){
    records.add(record);
}

public void deleteRecord(int index){
    records.remove(index);
}

public Record getRecord(int index){
    return records.get(index);
}

public ArrayList<Record> getRecords(){
    return records;
}

public void setRecords(ArrayList<Record> records){
    this.records = records;
}

//implementing cloneabe method
```

```
    @Override
    public Student clone() throws CloneNotSupportedException{
        Student studentClone = (Student) super.clone();
        studentClone.setRecords((ArrayList<Record>)this.getRecords().clone());
        return studentClone;
    }

    //empty abstract methods
    public abstract void setExpectedCompletion(String date);
    public abstract String getExpectedCompletion();
}

class Undergraduate extends Student{
    //datafields
    private String bachelorCompletion;

    //constructors
    public Undergraduate(){
        super();
        this.bachelorCompletion ="x";
    }

    public Undergraduate(String name, String DOB, int STUDENTID, String sGenderder, String expectedCompletionSession){
        super(name, DOB, sGenderder, STUDENTID);
        this.bachelorCompletion = expectedCompletionSession;
    }

    //methods
    public String getExpectedCompletion(){
        return bachelorCompletion;
    }

    public void setExpectedCompletion(String expectedCompletionSession){
        this.bachelorCompletion = expectedCompletionSession;
    }

    @Override
    public String toString(){
        String printOut = super.toString();
        for(int i = 0; i < getRecords().size(); i++){
            Record record = getRecord(i);
            printOut += String.format("\nCourse Name: %s, (%s) \n\n%s",
            record.getCourseName(), record.getStatus(), record);
        }

        return printOut;
    }
}

class Postgraduate extends Student{
    //datafields
    private String bachelorCompletion;
    private String masterCompletion;

    //constructors
    public Postgraduate(){
        super();
        this.bachelorCompletion ="x";
        this.masterCompletion = "x";
    }

    public Postgraduate(String name, String DOB, int STUDENTID, String sGenderder,
    String bachelorCompletion, String masterCompletion){
        super(name, DOB, sGenderder, STUDENTID);
        this.bachelorCompletion = bachelorCompletion;
        this.masterCompletion = masterCompletion;
    }

    //methods
    public String getExpectedCompletion(){
        return masterCompletion;
    }

    public void setExpectedCompletion(String expectedCompletionSession){
        this.masterCompletion = expectedCompletionSession;
    }

    public String getBachelorCompletion(){
        return bachelorCompletion;
    }

    public void setBachelorCompletion(String bachelorCompletion){
        this.bachelorCompletion = bachelorCompletion;
    }

    @Override
    public String toString(){
        String printOut = super.toString();
        for(int i = 0; i < getRecords().size(); i++){
            Record record = getRecord(i);
            printOut += String.format("\n\nCourse Name: %s, (%s) \n\n%s",
            record.getCourseName(), record.getStatus(), record);
        }

        printOut += "\nExpected Master Graduation: " + this.getExpectedCompletion();
        // printOut += "\n\nBachelor was received: " + getBachelorCompletion();

        return printOut;
    }
}

class Record implements Cloneable{
    //datafields
    private final String CNAME;
    private ArrayList<Subject> eCores;
    private Major eMajor;
    private ArrayList<Subject> eElectives;
    private int totalCredit;
    private Status status;

    public enum Status {ACTIVE, COMPLETE, NA}

    //default constructor
    public Record(){
        this("default");
    }

    //parameterised constructor
    public Record(String CNAME){
        this.CNAME = CNAME;
        this.eCores = new ArrayList<Subject>();
        this.eMajor = new Major();
        this.eElectives = new ArrayList<Subject>();
        this.totalCredit = 0;
        this.status = Status.NA;
    }

    //methods
    public void enrolCores(ArrayList<Subject> subjects){
        //maintains up to date credit points
        for(int i=0; i < subjects.size(); i++ ){
            this.totalCredit += subjects.get(i).getCredit();
        }

        //assigns core subjects
        this.eCores = subjects;
    }

    public void enrolMajor(Major major){
        //mtaintains up to date credit point count
        this.totalCredit += major.getCredit();
        this.eMajor = major;
    }

    public void enrolElective(Subject subject){
        this.eElectives.add(subject);
        this.totalCredit += subject.getCredit();
    }

    public void enrolElectives(ArrayList<Subject> subjects){
        //maintains up to date credit points
        for(int i=0; i < subjects.size(); i++ ){
            this.totalCredit += subjects.get(i).getCredit();
        }

        //assigns core subjects
        this.eCores = subjects;
    }

    public int getTotalCredit(){
        return totalCredit;
    }

    public void setTotalCredit(int credit){
        this.totalCredit = credit;
    }

    /*calculates number of electives left to reach minimum credit
    points for selected course*/
    public int howManyElectivesLeft(Course course){
        if(getTotalCredit() < course.getCCredit()){
            return ((course.getCCredit() - getTotalCredit())/6);
        }
        return 0;
    }

    //predicate method(just checks truth)
    public boolean isEnrolled(Subject name){
        if(eCores.contains(name)){
            return true;
        }
        else if(eElectives.contains(name)){
            return true;
        }
        else if(eMajor.isIncluded(name)){
            return true;
        }
        return false;
    }

    public Status getStatus(){
        return this.status;
    }

    public void setStatus(Status status){
        this.status = status;
    }
    public String toString(){
        String s = "";

        s+="Cores: \n";
        for(int i = 0; i < eCores.size(); i++)
            s+=eCores.get(i)+"\n";

        s+="\n";

        s+="Major: " + eMajor;

        s+="\n";

        s+="Electives: \n";
        for(int i = 0; i < eElectives.size(); i++)
            s+=eElectives.get(i)+"\n";

        s+="-----------------\n";

        s+="Total Enrolled Credit: "+ getTotalCredit() +"pt";

        return s;
    }

    public String getCourseName(){
        return CNAME;
    }

    //implementing cloneable interface
    public Record clone() throws CloneNotSupportedException{
        Record recordClone = (Record) super.clone();
        //clears credit points
        recordClone.setTotalCredit(0);

        //these methods all add relevent credit points
        recordClone.enrolMajor((Major)this.eMajor.clone());
        recordClone.enrolElectives((ArrayList<Subject>)this.eElectives.clone());
        recordClone.enrolCores((ArrayList<Subject>)this.eCores.clone());

        return recordClone;
    }
}

class Major implements Cloneable, Serializable{
    private String mName;
    private ArrayList<Subject> mCores;
    private int credit = 0;

    //default constructor
    public Major(){
        this("default");
    }

    //constructor with 1 parameter
    public Major(String name){
        this.mName = name;
        this.mCores = new ArrayList<Subject>();
    }

    //constructor with 2 parameters
    public Major(String name, Subject[] subjectList){
        this.mName = name;
        this.mCores = new ArrayList<Subject>();
        addMCores(subjectList);
    }
    public String getMName(){
        return mName;
    }

    public void setMName(String name){
        this.mName = name;
    }

    public int getCredit(){
        int credit = 0;

        for(int i = 0; i < mCores.size(); i++ ){
            credit += mCores.get(i).getCredit();
        }

        return credit;
    }

    //predicate method
    public boolean isIncluded(Subject name){
        ArrayList<Subject> subjects = this.getMCores();

        for(int i = 0; i < subjects.size(); i++ ){
            if(subjects.get(i).getName() == name.getName()){
                return true;
            }
        }
        return false;
    }

    public void addMCores(Subject[] subjects){
        for (Subject subject : subjects){
            this.mCores.add(subject);
        }
    }

    public ArrayList<Subject> getMCores(){
        return mCores;
    }

    //implementing clone interface
    @Override
    public Major clone() throws CloneNotSupportedException{
        return (Major) super.clone();
    }

    @Override
    public String toString(){
        String printOut = getMName() + "\n";
        ArrayList<Subject> subjects = getMCores();

        for(int i = 0; i < subjects.size(); i++ ){
            printOut += subjects.get(i) + "\n";
        }

        return printOut;
    }
}

class Course implements Serializable{
    private String cName;
    private ArrayList<Subject> cores;
    private ArrayList<Major> majors;
    private ArrayList<Subject> electives;
    private int cCredit;
```

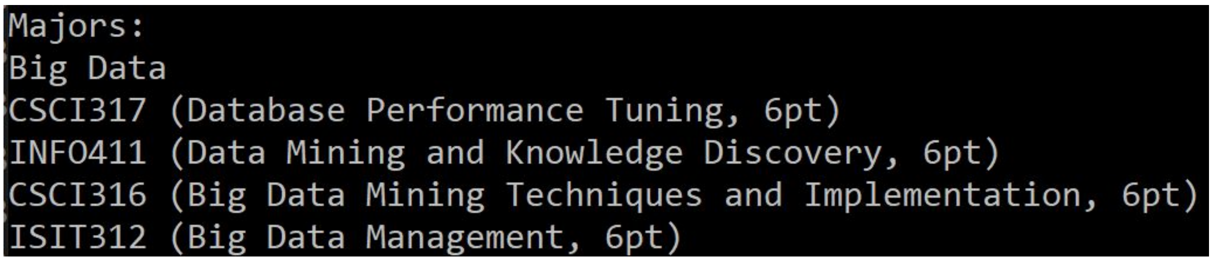
```
public Course(){
    this("default", 0);
}

public Course(String cName, int cCredit){
    this.cName = cName;
    this.cores = new ArrayList<Subject>();
    this.majors = new ArrayList<Major>();
    this.electives = new ArrayList<Subject>();
    this.cCredit = cCredit;
}

public void addCores(Subject[] subjects){
    for (Subject subject : subjects){
        this.cores.add(subject);
    }
}

public void addMajors(Major[] majors){
    for (Major major : majors){
        this.majors.add(major);
    }
}

public void addElectives(Subject[] electives){
    for (Subject elective : electives){
        this.electives.add(elective);
    }
}

//display all electives
public void printElectives(){
    System.out.println("Elective Subjects:");
    for(int i = 0; i < getElectives().size(); i++){
        System.out.println(getElectives().get(i));
    }
    System.out.println();
}

public ArrayList<Subject> getCores(){
    return cores;
}

public ArrayList<Subject> getElectives(){
    return electives;
}

public ArrayList<Major> getMajors(){
    return majors;
}

public String toString(){
    String s = "Course: "+getCName() + "\n";

    s+="Cores: \n";
    for(int i = 0; i < getCores().size(); i++)
        s+=getCores().get(i)+"\n";

    s+="\n";

    s+="Majors: \n";
    for(int i = 0; i < getMajors().size(); i++){
        s+=getMajors().get(i) + "\n";
    }

    s+="\n";
```

```
        s+="Electives: \n";
        for(int i = 0; i < getElectives().size(); i++)
            s+=getElectives().get(i)+"\n";

        s+="-----------------\n";

        s+="Required Total Credit: "+ getCCredit() +"pt";

        return s;
    }

    public String getCName(){
        return cName;
    }

    public int getCCredit(){
        return cCredit;
    }
}

class InputOutOfRange extends Exception{
    //datafields
    private String range;

    //parameterised constructor
    public InputOutOfRange(String range){
        this.range = range;
    }

    public String getMessage(){
        if(range != null){
            return String.format("Input out of range. Should be between %s", getRange());
        }
        else{
            return "Input out of range.";
        }
    }

    public String getRange(){
        return this.range;
    }
}
```

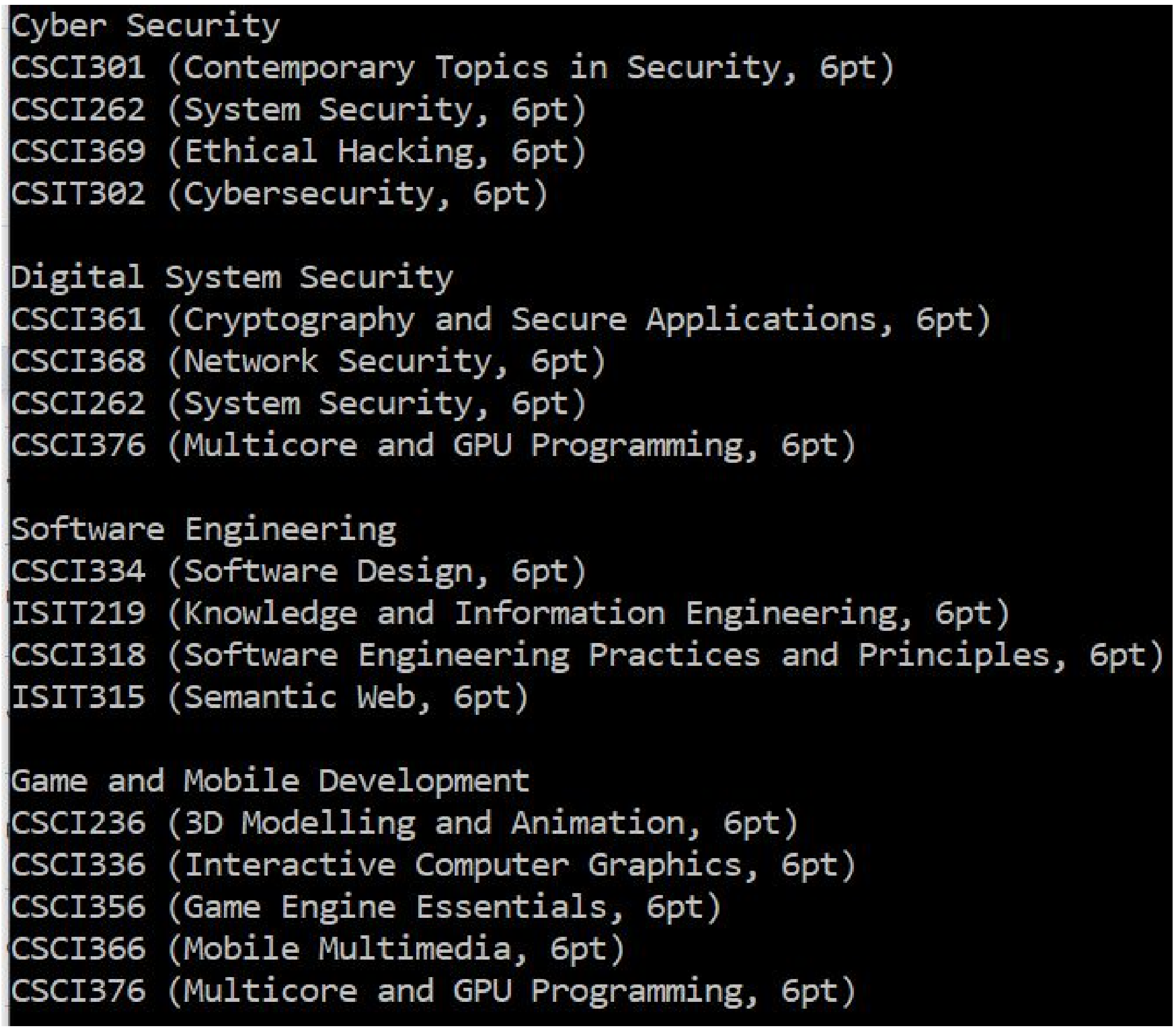
Screen shots and successful compilation:



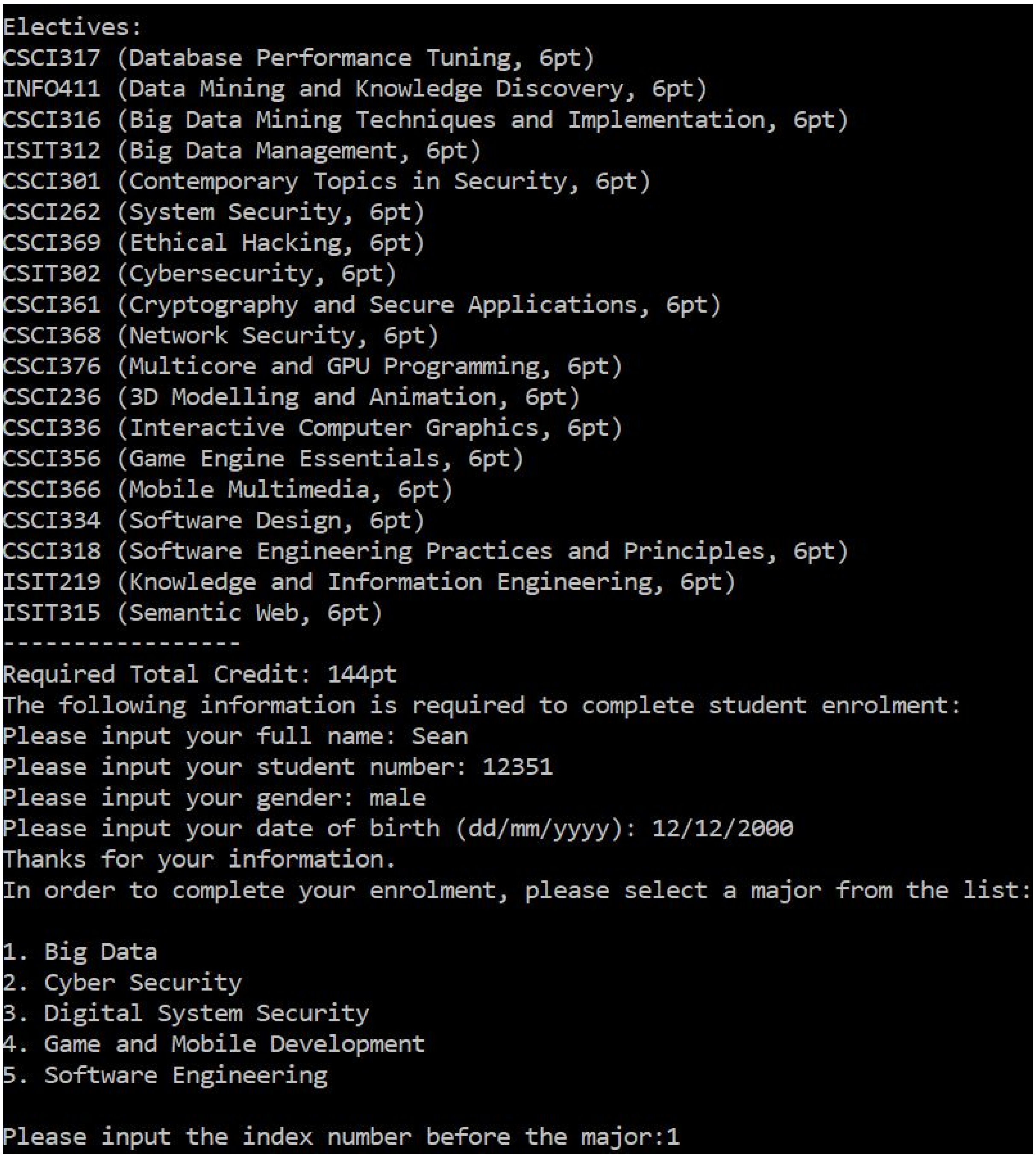
```
C:\Users\Sean\Desktop\Other stuff\UNI\CSI121 OOP\Labs\Lab5>javac StudentSystem.java

C:\Users\Sean\Desktop\Other stuff\UNI\CSI121 OOP\Labs\Lab5>java StudentSystem
(Enrol first student to Bachelor of Computer Science Course)
Welcome to enrol into Bachelor of Computer Science.
The Course structure is as follows:
------------------------------------------
Course: Bachelor of Computer Science
Cores:
CSIT111 (Programming Fundamentals, 6pt)
CSIT113 (Problem solving, 6pt)
CSIT114 (System Analysis, 6pt)
CSIT115 (Data Management and Security, 6pt)
CSIT121 (Object Oriented Design and Programming, 6pt)
CSIT127 (Networks and Communications, 6pt)
CSIT128 (Introduction to Web Technology, 6pt)
CSCI235 (Database Systems, 6pt)
CSCI251 (Advanced Programming, 6pt)
CSIT214 (IT Project Management, 6pt)
MATH221 (Mathematics for Computer Science, 6pt)
CSCI203 (Data Structures and Algorithms, 6pt)
CSIT226 (Human Computer Interaction, 6pt)
CSIT314 (Software Development Methodologies, 6pt)
CSIT321 (Project, 12pt)
```



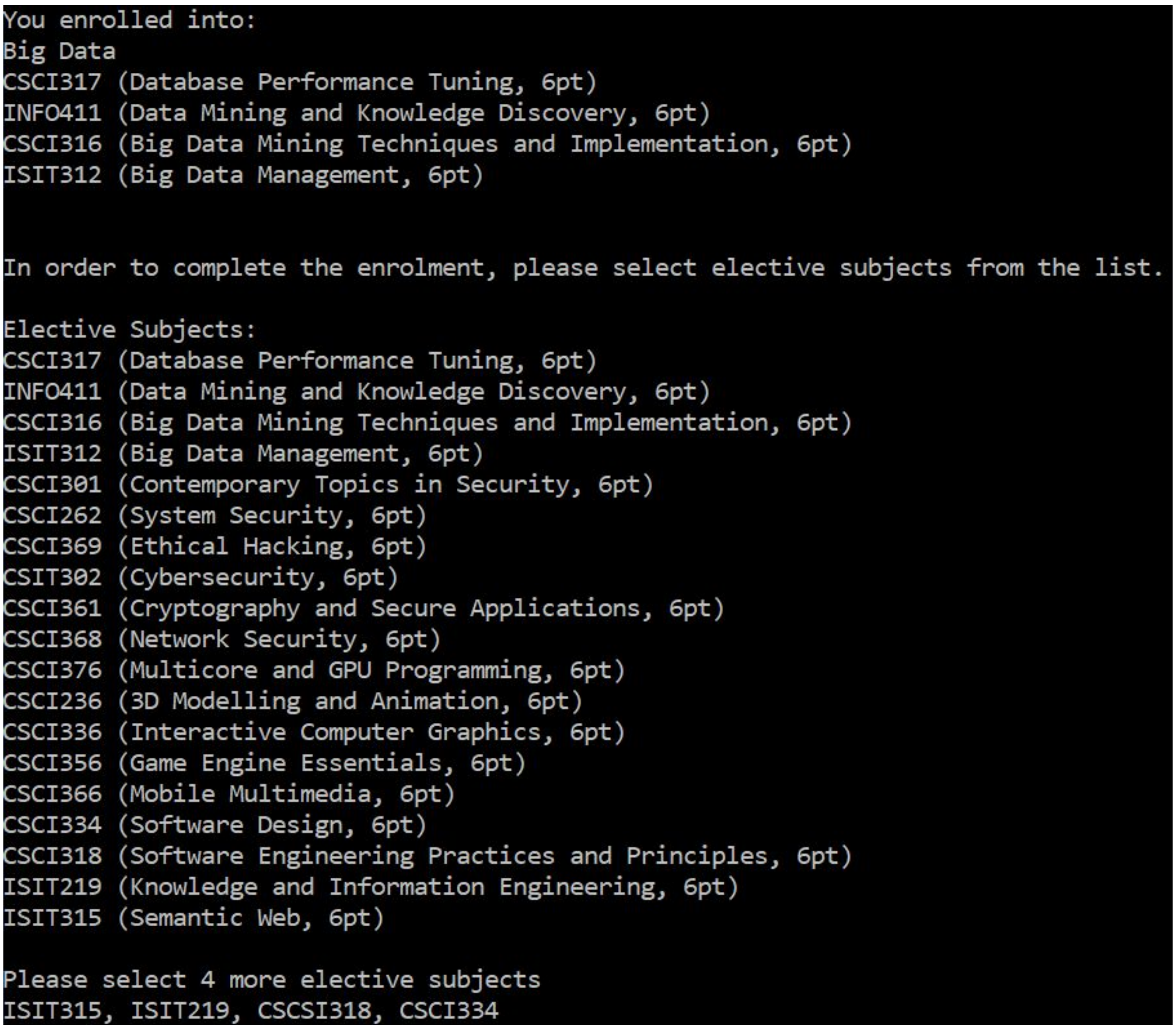
```
Majors:
Big Data
CSCI317 (Database Performance Tuning, 6pt)
INFO411 (Data Mining and Knowledge Discovery, 6pt)
CSCI316 (Big Data Mining Techniques and Implementation, 6pt)
ISIT312 (Big Data Management, 6pt)
```



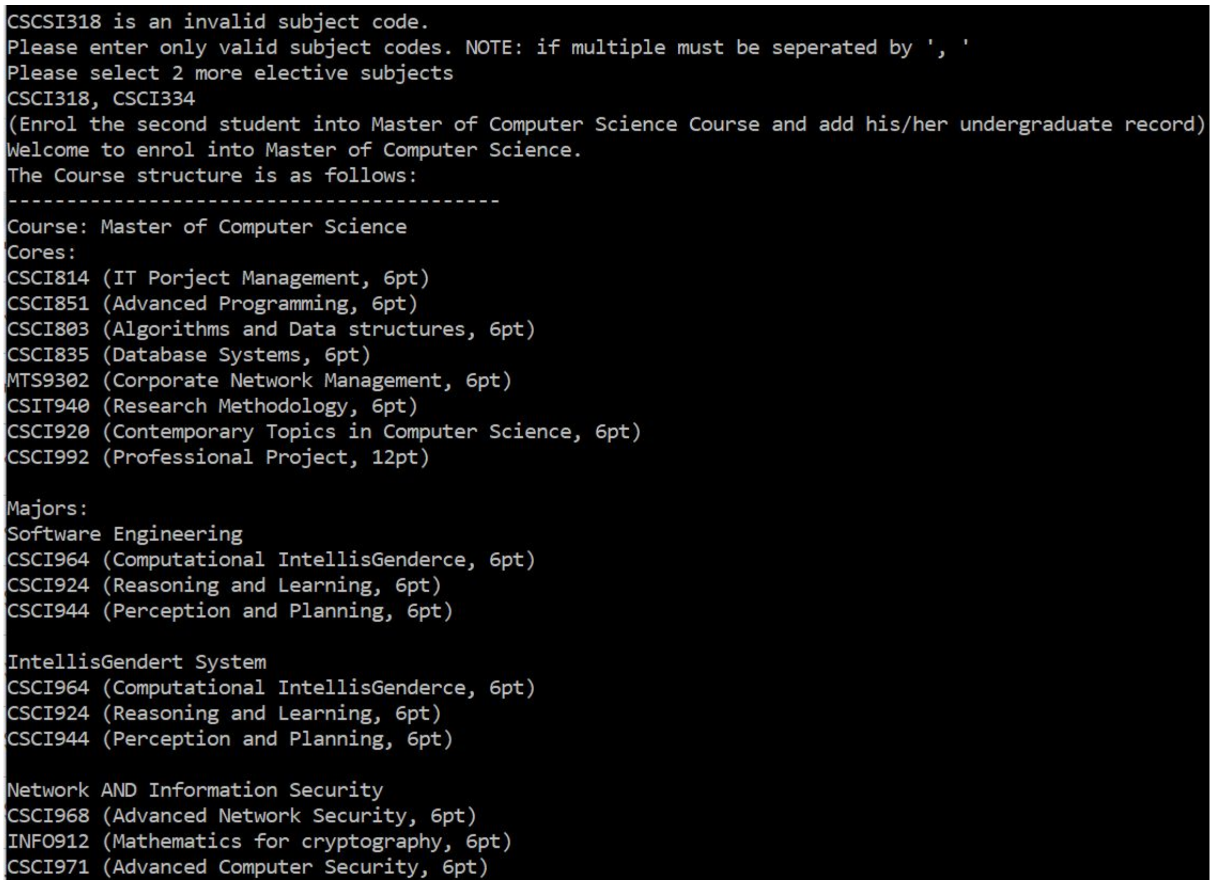
```
Cyber Security
CSCI301 (Contemporary Topics in Security, 6pt)
CSCI262 (System Security, 6pt)
CSCI369 (Ethical Hacking, 6pt)
CSIT302 (Cybersecurity, 6pt)

Digital System Security
CSCI361 (Cryptography and Secure Applications, 6pt)
CSCI368 (Network Security, 6pt)
CSCI262 (System Security, 6pt)
CSCI376 (Multicore and GPU Programming, 6pt)

Software Engineering
CSCI334 (Software Design, 6pt)
ISIT219 (Knowledge and Information Engineering, 6pt)
CSCI318 (Software Engineering Practices and Principles, 6pt)
ISIT315 (Semantic Web, 6pt)

Game and Mobile Development
CSCI236 (3D Modelling and Animation, 6pt)
CSCI336 (Interactive Computer Graphics, 6pt)
CSCI356 (Game Engine Essentials, 6pt)
CSCI366 (Mobile Multimedia, 6pt)
CSCI376 (Multicore and GPU Programming, 6pt)
```



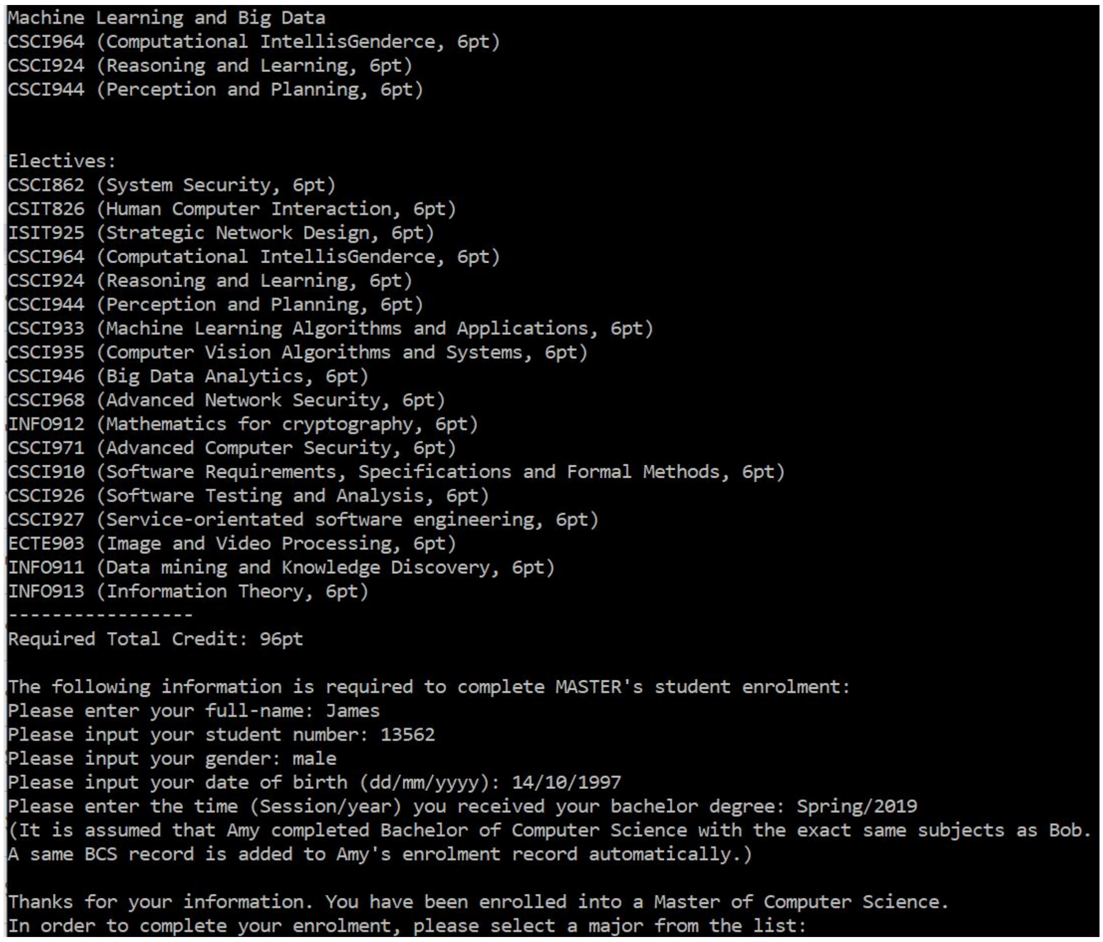
```
Electives:
CSCI317 (Database Performance Tuning, 6pt)
INFO411 (Data Mining and Knowledge Discovery, 6pt)
CSCI316 (Big Data Mining Techniques and Implementation, 6pt)
ISIT312 (Big Data Management, 6pt)
CSCI301 (Contemporary Topics in Security, 6pt)
CSCI262 (System Security, 6pt)
CSCI369 (Ethical Hacking, 6pt)
CSIT302 (Cybersecurity, 6pt)
CSCI361 (Cryptography and Secure Applications, 6pt)
CSCI368 (Network Security, 6pt)
CSCI376 (Multicore and GPU Programming, 6pt)
CSCI236 (3D Modelling and Animation, 6pt)
CSCI336 (Interactive Computer Graphics, 6pt)
CSCI356 (Game Engine Essentials, 6pt)
CSCI366 (Mobile Multimedia, 6pt)
CSCI334 (Software Design, 6pt)
CSCI318 (Software Engineering Practices and Principles, 6pt)
ISIT219 (Knowledge and Information Engineering, 6pt)
ISIT315 (Semantic Web, 6pt)
-----------------
Required Total Credit: 144pt
The following information is required to complete student enrolment:
Please input your full name: Sean
Please input your student number: 12351
Please input your gender: male
Please input your date of birth (dd/mm/yyyy): 12/12/2000
Thanks for your information.
In order to complete your enrolment, please select a major from the list:

1. Big Data
2. Cyber Security
3. Digital System Security
4. Game and Mobile Development
5. Software Engineering

Please input the index number before the major:1
```



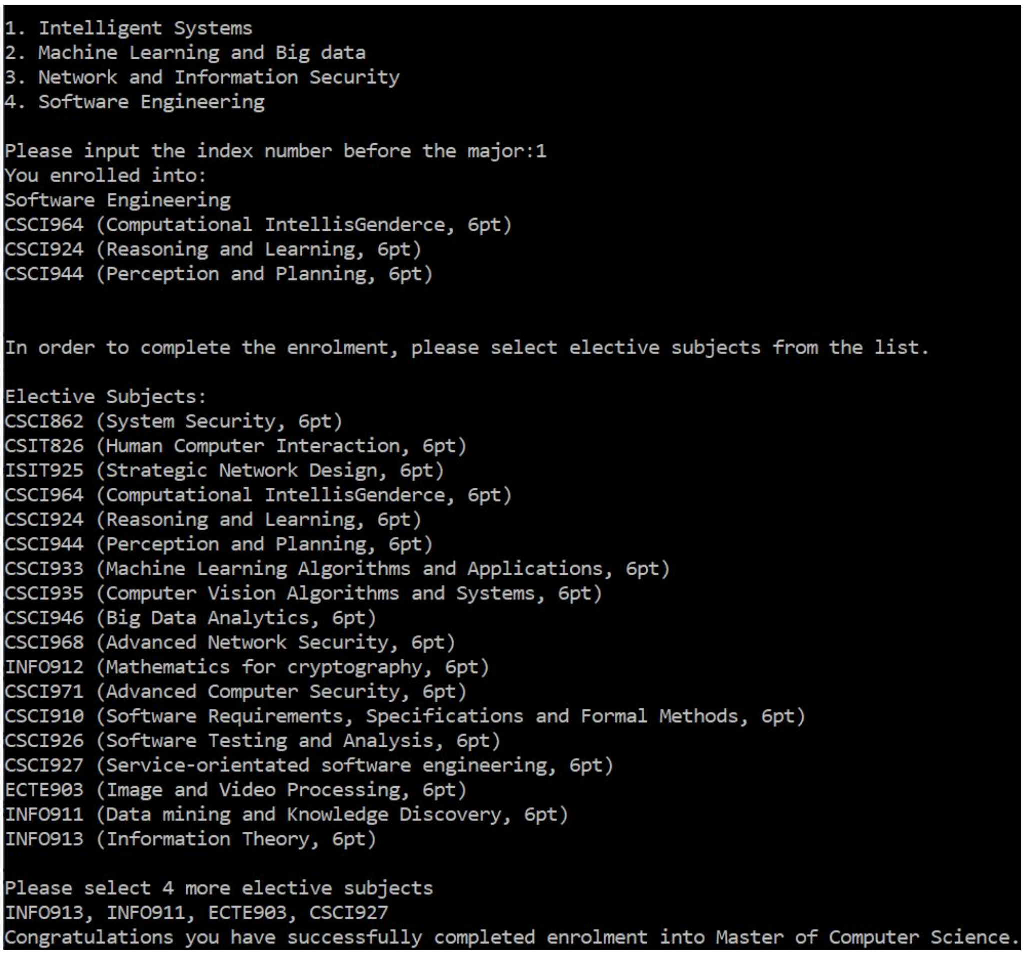
```
You enrolled into:
Big Data
CSCI317 (Database Performance Tuning, 6pt)
INFO411 (Data Mining and Knowledge Discovery, 6pt)
CSCI316 (Big Data Mining Techniques and Implementation, 6pt)
ISIT312 (Big Data Management, 6pt)


In order to complete the enrolment, please select elective subjects from the list.

Elective Subjects:
CSCI317 (Database Performance Tuning, 6pt)
INFO411 (Data Mining and Knowledge Discovery, 6pt)
CSCI316 (Big Data Mining Techniques and Implementation, 6pt)
ISIT312 (Big Data Management, 6pt)
CSCI301 (Contemporary Topics in Security, 6pt)
CSCI262 (System Security, 6pt)
CSCI369 (Ethical Hacking, 6pt)
CSIT302 (Cybersecurity, 6pt)
CSCI361 (Cryptography and Secure Applications, 6pt)
CSCI368 (Network Security, 6pt)
CSCI376 (Multicore and GPU Programming, 6pt)
CSCI236 (3D Modelling and Animation, 6pt)
CSCI336 (Interactive Computer Graphics, 6pt)
CSCI356 (Game Engine Essentials, 6pt)
CSCI366 (Mobile Multimedia, 6pt)
CSCI334 (Software Design, 6pt)
CSCI318 (Software Engineering Practices and Principles, 6pt)
ISIT219 (Knowledge and Information Engineering, 6pt)
ISIT315 (Semantic Web, 6pt)

Please select 4 more elective subjects
ISIT315, ISIT219, CSCSI318, CSCI334
```



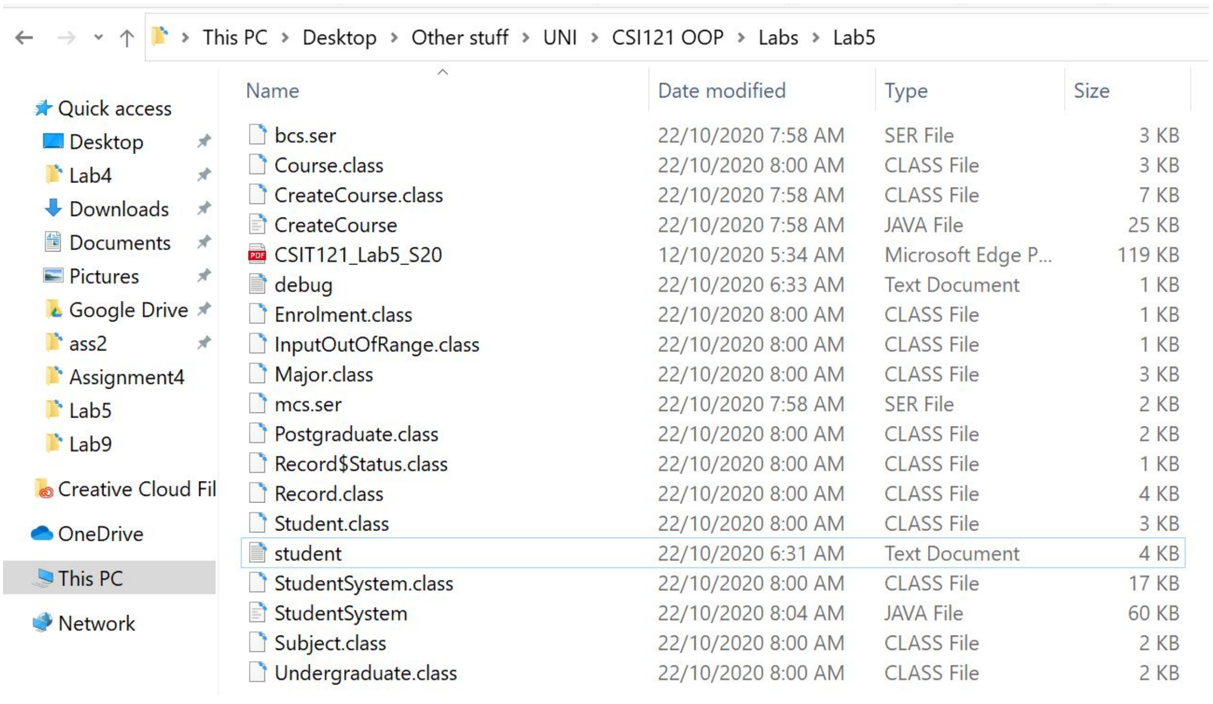
```
CSCSI318 is an invalid subject code.
Please enter only valid subject codes. NOTE: if multiple must be seperated by ', '
Please select 2 more elective subjects
CSCI318, CSCI334
(Enrol the second student into Master of Computer Science Course and add his/her undergraduate record)
Welcome to enrol into Master of Computer Science.
The Course structure is as follows:
------------------------------------------
Course: Master of Computer Science
Cores:
CSCI814 (IT Porject Management, 6pt)
CSCI851 (Advanced Programming, 6pt)
CSCI803 (Algorithms and Data structures, 6pt)
CSCI835 (Database Systems, 6pt)
MTS9302 (Corporate Network Management, 6pt)
CSIT940 (Research Methodology, 6pt)
CSCI920 (Contemporary Topics in Computer Science, 6pt)
CSCI992 (Professional Project, 12pt)

Majors:
Software Engineering
CSCI964 (Computational IntellisGendence, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)

IntellisGendert System
CSCI964 (Computational IntellisGendence, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)

Network AND Information Security
CSCI968 (Advanced Network Security, 6pt)
INFO912 (Mathematics for cryptography, 6pt)
CSCI971 (Advanced Computer Security, 6pt)
```



```
Machine Learning and Big Data
CSCI964 (Computational IntellisGendence, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)


Electives:
CSCI862 (System Security, 6pt)
CSIT826 (Human Computer Interaction, 6pt)
ISIT925 (Strategic Network Design, 6pt)
CSCI964 (Computational IntellisGendence, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)
CSCI933 (Machine Learning Algorithms and Applications, 6pt)
CSCI935 (Computer Vision Algorithms and Systems, 6pt)
CSCI946 (Big Data Analytics, 6pt)
CSCI968 (Advanced Network Security, 6pt)
INFO912 (Mathematics for cryptography, 6pt)
CSCI971 (Advanced Computer Security, 6pt)
CSCI910 (Software Requirements, Specifications and Formal Methods, 6pt)
CSCI926 (Software Testing and Analysis, 6pt)
CSCI927 (Service-orientated software engineering, 6pt)
ECTE903 (Image and Video Processing, 6pt)
INFO911 (Data mining and Knowledge Discovery, 6pt)
INFO913 (Information Theory, 6pt)
----------------
Required Total Credit: 96pt

The following information is required to complete MASTER's student enrolment:
Please enter your full-name: James
Please input your student number: 13562
Please input your gender: male
Please input your date of birth (dd/mm/yyyy): 14/10/1997
Please enter the time (Session/year) you received your bachelor degree: Spring/2019
(It is assumed that Amy completed Bachelor of Computer Science with the exact same subjects as Bob.
A same BCS record is added to Amy's enrolment record automatically.)

Thanks for your information. You have been enrolled into a Master of Computer Science.
In order to complete your enrolment, please select a major from the list:
```

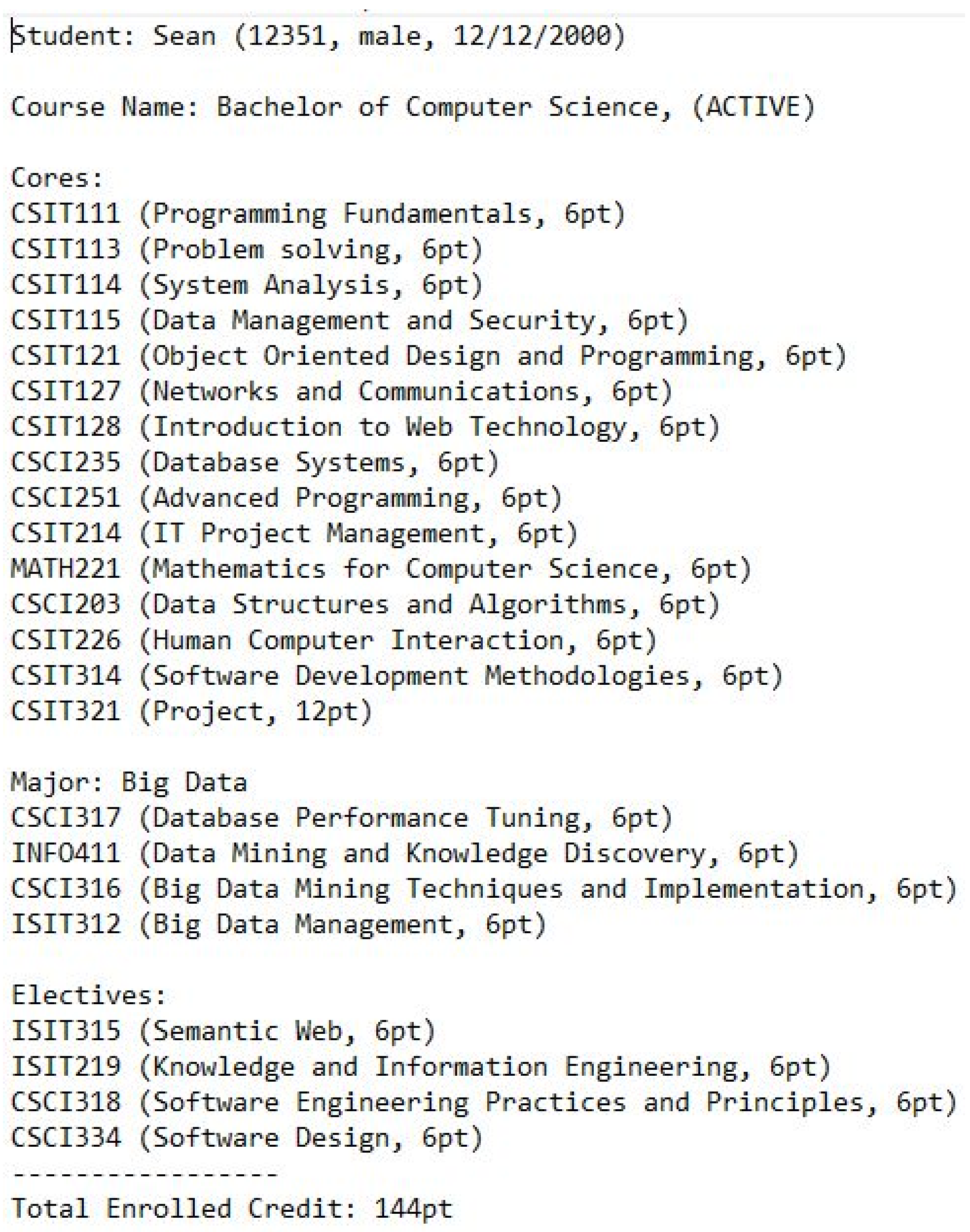


```
1. Intelligent Systems
2. Machine Learning and Big data
3. Network and Information Security
4. Software Engineering

Please input the index number before the major:1
You enrolled into:
Software Engineering
CSCI964 (Computational IntellisGenderce, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)


In order to complete the enrolment, please select elective subjects from the list.

Elective Subjects:
CSCI862 (System Security, 6pt)
CSIT826 (Human Computer Interaction, 6pt)
ISIT925 (Strategic Network Design, 6pt)
CSCI964 (Computational IntellisGenderce, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)
CSCI933 (Machine Learning Algorithms and Applications, 6pt)
CSCI935 (Computer Vision Algorithms and Systems, 6pt)
CSCI946 (Big Data Analytics, 6pt)
CSCI968 (Advanced Network Security, 6pt)
INFO912 (Mathematics for cryptography, 6pt)
CSCI971 (Advanced Computer Security, 6pt)
CSCI910 (Software Requirements, Specifications and Formal Methods, 6pt)
CSCI926 (Software Testing and Analysis, 6pt)
CSCI927 (Service-orientated software engineering, 6pt)
ECTE903 (Image and Video Processing, 6pt)
INFO911 (Data mining and Knowledge Discovery, 6pt)
INFO913 (Information Theory, 6pt)

Please select 4 more elective subjects
INFO913, INFO911, ECTE903, CSCI927
Congratulations you have successfully completed enrolment into Master of Computer Science.
```

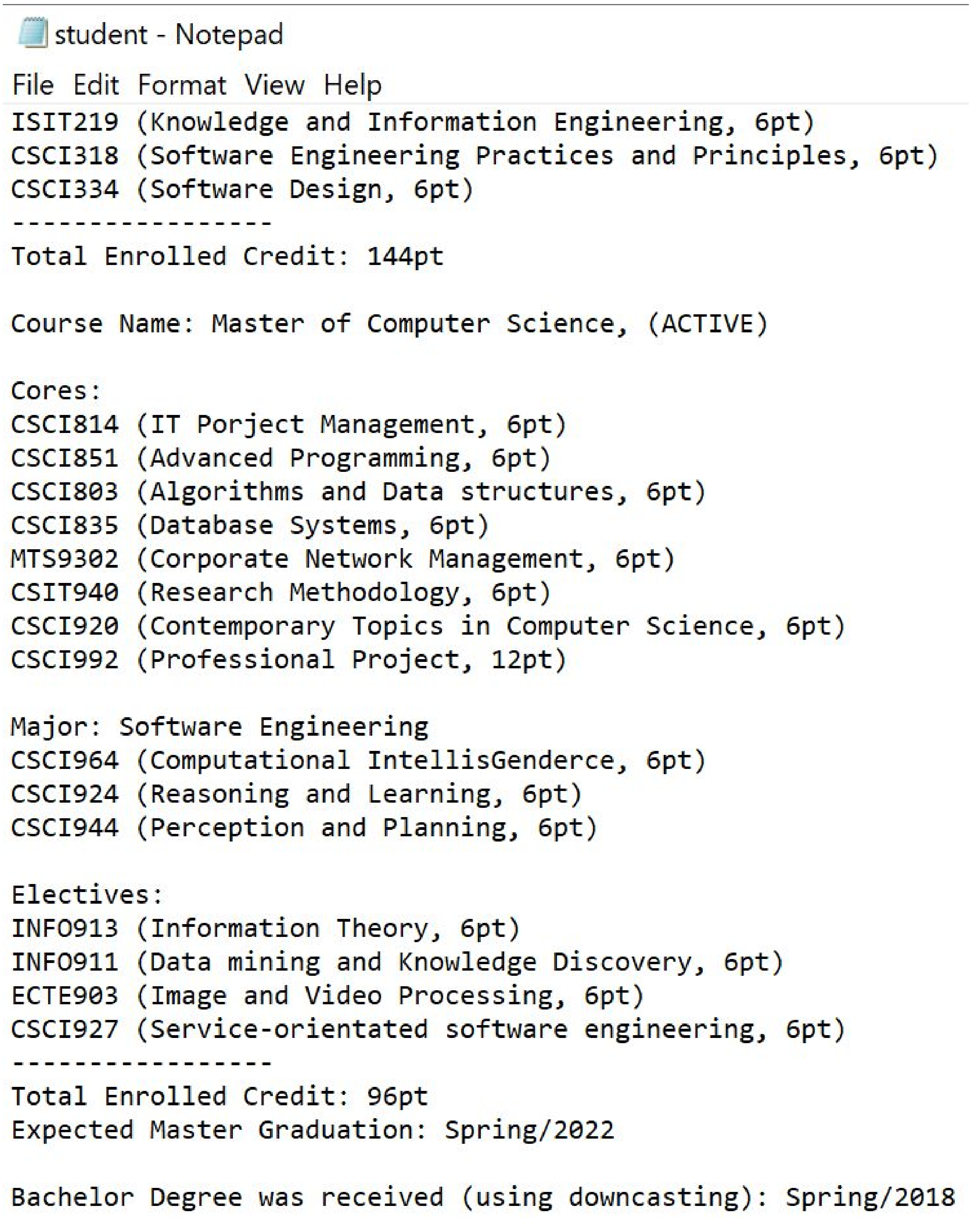


Name	Date modified	Type	Size
bcs.ser	22/10/2020 7:58 AM	SER File	3 KB
Course.class	22/10/2020 8:00 AM	CLASS File	3 KB
CreateCourse.class	22/10/2020 7:58 AM	CLASS File	7 KB
CreateCourse	22/10/2020 7:58 AM	JAVA File	25 KB
CSIT121_Lab5_S20	12/10/2020 5:34 AM	Microsoft Edge P...	119 KB
debug	22/10/2020 6:33 AM	Text Document	1 KB
Enrolment.class	22/10/2020 8:00 AM	CLASS File	1 KB
InputOutOfRange.class	22/10/2020 8:00 AM	CLASS File	1 KB
Major.class	22/10/2020 8:00 AM	CLASS File	3 KB
mcs.ser	22/10/2020 7:58 AM	SER File	2 KB
Postgraduate.class	22/10/2020 8:00 AM	CLASS File	2 KB
Record$Status.class	22/10/2020 8:00 AM	CLASS File	1 KB
Record.class	22/10/2020 8:00 AM	CLASS File	4 KB
Student.class	22/10/2020 8:00 AM	CLASS File	3 KB
student	22/10/2020 6:31 AM	Text Document	4 KB
StudentSystem.class	22/10/2020 8:00 AM	CLASS File	17 KB
StudentSystem	22/10/2020 8:04 AM	JAVA File	60 KB
Subject.class	22/10/2020 8:00 AM	CLASS File	2 KB
Undergraduate.class	22/10/2020 8:00 AM	CLASS File	2 KB

-the student.txt file located in the same directory



```
Student: Sean (12351, male, 12/12/2000)

Course Name: Bachelor of Computer Science, (ACTIVE)

Cores:
CSIT111 (Programming Fundamentals, 6pt)
CSIT113 (Problem solving, 6pt)
CSIT114 (System Analysis, 6pt)
CSIT115 (Data Management and Security, 6pt)
CSIT121 (Object Oriented Design and Programming, 6pt)
CSIT127 (Networks and Communications, 6pt)
CSIT128 (Introduction to Web Technology, 6pt)
CSCI235 (Database Systems, 6pt)
CSCI251 (Advanced Programming, 6pt)
CSIT214 (IT Project Management, 6pt)
MATH221 (Mathematics for Computer Science, 6pt)
CSCI203 (Data Structures and Algorithms, 6pt)
CSIT226 (Human Computer Interaction, 6pt)
CSIT314 (Software Development Methodologies, 6pt)
CSIT321 (Project, 12pt)

Major: Big Data
CSCI317 (Database Performance Tuning, 6pt)
INFO411 (Data Mining and Knowledge Discovery, 6pt)
CSCI316 (Big Data Mining Techniques and Implementation, 6pt)
ISIT312 (Big Data Management, 6pt)

Electives:
ISIT315 (Semantic Web, 6pt)
ISIT219 (Knowledge and Information Engineering, 6pt)
CSCI318 (Software Engineering Practices and Principles, 6pt)
CSCI334 (Software Design, 6pt)
-----------------
Total Enrolled Credit: 144pt
```


student - Notepad

File Edit Format View Help

```
ISIT219 (Knowledge and Information Engineering, 6pt)
CSCI318 (Software Engineering Practices and Principles, 6pt)
CSCI334 (Software Design, 6pt)
-----------------
Total Enrolled Credit: 144pt

Course Name: Master of Computer Science, (ACTIVE)

Cores:
CSCI814 (IT Porject Management, 6pt)
CSCI851 (Advanced Programming, 6pt)
CSCI803 (Algorithms and Data structures, 6pt)
CSCI835 (Database Systems, 6pt)
MTS9302 (Corporate Network Management, 6pt)
CSIT940 (Research Methodology, 6pt)
CSCI920 (Contemporary Topics in Computer Science, 6pt)
CSCI992 (Professional Project, 12pt)

Major: Software Engineering
CSCI964 (Computational IntellisGenderce, 6pt)
CSCI924 (Reasoning and Learning, 6pt)
CSCI944 (Perception and Planning, 6pt)

Electives:
INFO913 (Information Theory, 6pt)
INFO911 (Data mining and Knowledge Discovery, 6pt)
ECTE903 (Image and Video Processing, 6pt)
CSCI927 (Service-orientated software engineering, 6pt)
-----------------
Total Enrolled Credit: 96pt
Expected Master Graduation: Spring/2022

Bachelor Degree was received (using downcasting): Spring/2018
```

-the contents of the student.txt file