

Java SE Programming Language

Lab Guides


Document Code	25e-BM/HR/HDCV/FSOFT
Version	1.1
Effective Date	20/11/2012

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	01/Oct/2018	Create new	Draft		
2	17/Jun/2019	Update template	Fsoft template	DieuNT1	

Contents

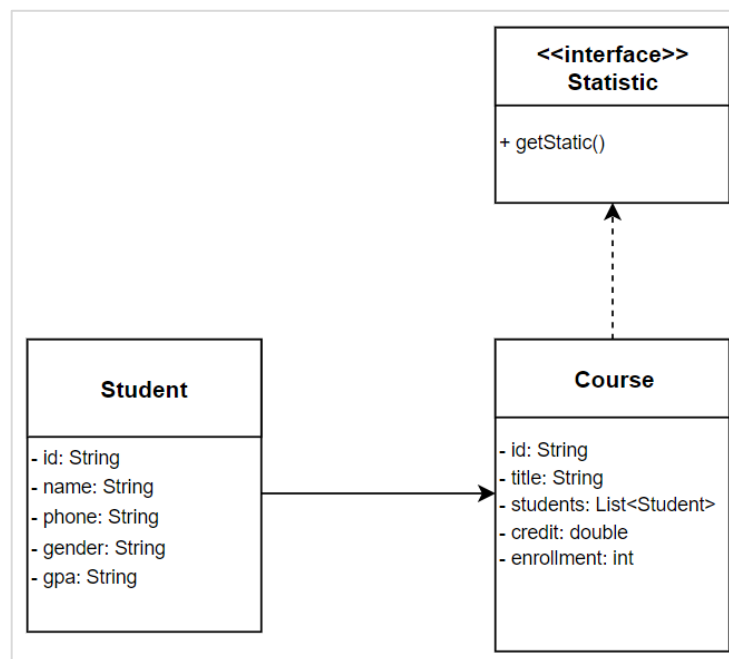
Unit 13: Java I/O Fundamentals	4
Lab Specifications:.....	4
Problem Descriptions:.....	5
Business Rules:	5
Guidelines:.....	5
Outputs:.....	18

	CODE: JPL.L.L101 TYPE: MEDIUM LOC: DURATION: 180 MINUTES
---	---

Unit 13: Java I/O Fundamentals

Lab Specifications:

For the hierarchy below, the trainee will create java classes that will implement this class diagram. Your classes should be able to show relationship between the entities.



Create a class called **Student** with the following information:

- » Four private instance variables: `id` (String), `name` (String), `phone` (String), `gender` (String), and `gpa` (double).
- » One constructor to initialize the `id`, `name`, `email`, and `gender` with the given values. Also including **getter** and **setter** method.

And, a class called **Course** with the following information:

- » Five private instance variables: `id` (String), `title` (String), `student` (of a class **Student** you have just created), `credit` (double), and `enrollment` (int).
- » One constructor to initialize all the given values. Also including **getter** and **setter** method. - Implements **Statistical** interface.

Finally, an interface **Statistical**:

- » For **Course**, it will return the summary of how many students get grade A, B, C, and D. You will use **Map** for this case.

Problem Descriptions:

- a. Write a method to enter Course data, users will be asked whether to continue or finish
Finish typing data when the user select 'n' or 'N' or the number of students in the course exceed *không quá* enrollment.
The students in a course and Course Id must be unique.
- b. Write a method to save the entire course list into "**course.dat**" file.
- c. The program has a method to sort course by id and display data in the following format:

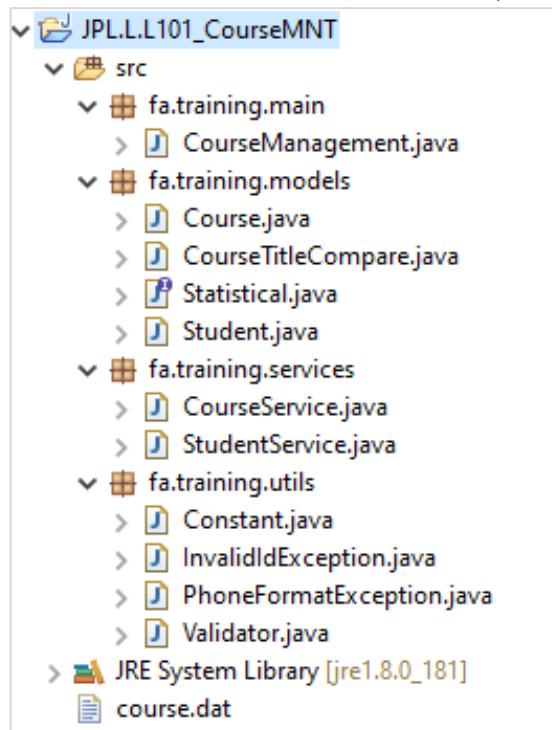
N01	C# programming	200.000	10	[Author [name=Nhung, email=nhung@fsoft.com, gender=female]]
J01	Java	1.000	1	[Author [name=Dieu, email=dieunt1@fsoft.com, gender=male]]
- d. The program has a method to search for courses that a student has taken by him/her id.
- e. Write a method to remove a course from course file.
- f. Write a method to get the ranking of all the courses.

Business Rules:

- » Phone: must be at least 10 digits
- » Student Id: 6 digits (i.e. 123456)
- » Course Id: 2 upper case letter follows by 3 digits (i.e. CS141)

Guidelines:

- » Step1. Create a project named **JPL.L.L101_CourseMNT** in Eclipse as below:



- » Step2. Create package **fa.training.models** that contains **Course**, **Student**, **CourseTitleCompare**, **Statistical** classes:

Course class:

```
1. package fa.training.models;
2.
3. // imports
4.
5. public class Course implements Serializable, Statistical {
6.
7.     private static final long serialVersionUID = 1L;
8.     private String id;
9.     private String title;
10.    /**
11.     * List of Students enrolled.
12.     */
13.    private Set<Student> students;
14.    private double credit;
15.    private int enrollment;
16.
17.    public Course(String id, String title, Set<Student> students,
18.                  double credit, int enrollment) {
19.        super();
20.        this.id = id;
21.        this.title = title;
22.        this.students = students;
23.        this.credit = credit;
24.        this.enrollment = enrollment;
25.    }
26.
27.    public Course() {
28.
29.    }
30.
31.    public String getId() {
32.        return id;
33.    }
34.
35.    public void setId(String id) throws InvalidIdException {
36.        if ((Validator.isCourseId(id)) && (Validator.isExisted(id))) {
37.            this.id = id;
38.        } else {
39.            throw new InvalidIdException("Id is invalid");
40.        }
41.    }
42.
43.    public String getTitle() {
44.        return title;
45.    }
46.
47.    public void setTitle(String title) {
48.        this.title = title;
49.    }
50.
51.    public Set<Student> getStudents() {
52.        return students;
53.    }
54.
55.    public void setStudents(Set<Student> students) {
56.        this.students = students;
57.    }
58.
59.    public double getCredit() {
60.        return credit;
61.    }
62.
63.    public void setCredit(double credit) {
64.        this.credit = credit;
```

```
65.     }
66.
67.     public int getEnrollment() {
68.         return enrollment;
69.     }
70.
71.     public void setEnrollment(int enrollment) {
72.         this.enrollment = enrollment;
73.     }
74.
75.     @Override
76.     public Map<String, Integer> getStatistic() {
77.         Map<String, Integer> stats = new HashMap<String, Integer>();
78.         stats.put("A", 0);
79.         stats.put("B", 0);
80.         stats.put("C", 0);
81.         stats.put("D", 0);
82.         for (Student student : students) {
83.             if (student.getGpa() >= 8.5) {
84.                 stats.put("A", stats.get("A") + 1);
85.             } else if (student.getGpa() < 8.5 && student.getGpa() >= 7) {
86.                 stats.put("B", stats.get("B") + 1);
87.             } else if (student.getGpa() < 7 && student.getGpa() >= 6) {
88.                 stats.put("C", stats.get("C") + 1);
89.             } else {
90.                 stats.put("D", stats.get("D") + 1);
91.             }
92.         }
93.         return stats;
94.     }
95. }
```

Student class:

```
1. package fa.training.models;
2.
3. // imports
4.
5. public class Student implements Serializable {
6.
7.     private static final long serialVersionUID = 1L;
8.     private String id;
9.     private String name;
10.    private String phone;
11.    private String gender;
12.    private double gpa;
13.
14.    public Student(String name, String phone, String gender, String id,
15.        double gpa) {
16.        super();
17.        this.id = id;
18.        this.name = name;
19.        this.phone = phone;
20.        this.gender = gender;
21.        this.gpa = gpa;
22.    }
23.    public Student() {
24.
25.    }
26.
27.    public String getId() {
28.        return id;
29.    }
30.
31.    public void setId(String id) throws InvalidIdException {
32.        if (Validator.isStudentId(id)) {
33.            this.id = id;
34.        } else {
35.            throw new InvalidIdException("Id is invalid!");
36.        }
37.    }
38. }
```

```
37.     }
38.
39.     public String getName() {
40.         return name;
41.     }
42.
43.     public void setName(String name) {
44.         this.name = name;
45.     }
46.
47.     public String getPhone() {
48.         return phone;
49.     }
50.
51.     public void setPhone(String phone) throws PhoneFormatException {
52.         if (Validator.isPhone(phone)) {
53.             this.phone = phone;
54.         } else {
55.             throw new PhoneFormatException("Email is invalid!");
56.         }
57.     }
58.
59.     public String getGender() {
60.         return gender;
61.     }
62.
63.     public void setGender(String gender) {
64.         this.gender = gender;
65.     }
66.
67.     public double getGpa() {
68.         return gpa;
69.     }
70.
71.     public void setGpa(double gpa) {
72.         this.gpa = gpa;
73.     }
74.
75.     @Override
76.     public String toString() {
77.         return "Student [id=" + id + ", name=" + name + ", phone=" + phone
78.             + ", gender=" + gender + ", gpa=" + gpa + "]";
79.     }
80.
81.     @Override
82.     public int hashCode() {
83.         final int prime = 31;
84.         int result = 1;
85.         result = prime * result + ((id == null) ? 0 : id.hashCode());
86.         return result;
87.     }
88.
89.     @Override
90.     public boolean equals(Object obj) {
91.         if (this == obj)
92.             return true;
93.         if (obj == null)
94.             return false;
95.         if (getClass() != obj.getClass())
96.             return false;
97.         Student other = (Student) obj;
98.         if (id == null) {
99.             if (other.id != null)
100.                return false;
101.            } else if (!id.equals(other.id))
102.                return false;
103.            return true;
104.        }
105.    }
```


CourseTitleCompare class:

```
1.
2. package fa.training.models;
3.
4. import java.util.Comparator;
5.
6. public class CourseTitleCompare implements Comparator<Course> {
7.
8.     @Override
9.     public int compare(Course o1, Course o2) {
10.         return o1.getId().compareToIgnoreCase(o2.getId());
11.     }
12.
13. }
```

Statistical interface:

```
1.
2. package fa.training.models;
3.
4. import java.util.Map;
5.
6. public interface Statistical {
7.
8.     /**
9.      * Show the statistic of all the students in the class
10.     *
11.     * @return
12.     */
13.     public Map<String, Integer> getStatistic();
14.
15. }
```

- » Step3. Create package **fa.training.utils** that contains **Constant**, **Validator**, **PhoneFormatException**, **InvalidIdException** classes:

Constant class

```
1.
2. package fa.training.utils;
3.
4. public class Constant {
5.
6.     public static final String FILE_PATH = "course.dat";
7.     public static final String SUCCESS = "success";
8.     public static final String FAIL = "fail";
9.     public static final String INPUT = "1";
10.    public static final String SAVE = "2";
11.    public static final String SORT = "3";
12.    public static final String SEARCH = "4";
13.    public static final String REMOVE = "5";
14.    public static final String STATS = "6";
15.    public static final String EXIT = "7";
16.
17. }
```

Validator class:

```
1. package fa.training.utils;
2.
3. import java.util.HashSet;
4. import java.util.Set;
5. import java.util.regex.Matcher;
6. import java.util.regex.Pattern;
7.
8. public class Validator {
9.
10.     private static final String VALID_PHONE_REGEX = "^\\d{10}$";
11.     private static final String VALID_STUDENT_ID_REGEX = "^\\d{6}$";
12.     private static final String VALID_COURSE_ID_REGEX = "[A-Z]{2}\\d{3}";
13.     private static Set<String> ids = new HashSet<String>();
```

```
14.  /**
15.   * Check phone number format is valid
16.   *
17.   * @param phone
18.   * @return
19.   */
20.  public static boolean isPhone(String phone) {
21.      Pattern pattern = Pattern.compile(VALID_PHONE_REGEX);
22.      Matcher matcher = pattern.matcher(phone);
23.      return matcher.matches();
24.  }
25.
26.  /**
27.   * Check student id format is valid
28.   *
29.   * @param id
30.   * @return
31.   */
32.  public static boolean isStudentId(String id) {
33.      Pattern pattern = Pattern.compile(VALID_STUDENT_ID_REGEX);
34.      Matcher matcher = pattern.matcher(id);
35.      return matcher.matches();
36.  }
37.
38.  /**
39.   * Check if course id format is valid
40.   *
41.   * @param id
42.   * @return
43.   */
44.  public static boolean isCourseId(String id) {
45.      Pattern pattern = Pattern.compile(VALID_COURSE_ID_REGEX);
46.      Matcher matcher = pattern.matcher(id);
47.      return matcher.matches();
48.  }
49.
50.  /**
51.   * Check if credit format is valid
52.   *
53.   * @param credit
54.   * @return
55.   */
56.  public static double isCredit(String credit) {
57.      double doCredit = 0d;
58.      try {
59.          doCredit = Double.parseDouble(credit);
60.      } catch (NumberFormatException exception) {
61.          throw new NumberFormatException();
62.      }
63.      return doCredit;
64.  }
65.
66.  /**
67.   * Check if enrollment format is valid
68.   *
69.   * @param enrollment
70.   * @return
71.   */
72.  public static int isEnrollment(String enrollment) {
73.      int intEnrollment = 0;
74.      try {
75.          intEnrollment = Integer.parseInt(enrollment);
76.      } catch (NumberFormatException exception) {
77.          throw new NumberFormatException();
78.      }
79.      return intEnrollment;
80.  }
81.
82.  /**
```

```
83.  * Check if an id exists or not
84.  *
85.  * @param id
86.  * @return
87.  */
88.  public static boolean isExisted(String id) {
89.      if (!ids.contains(id)) {
90.          ids.add(id);
91.          return true;
92.      } else {
93.          return false;
94.      }
95.  }
96.
97.  /**
98.   * Get the set
99.   *
100.   * @return
101.   */
102.   public static Set<String> getIds() {
103.       return ids;
104.   }
105.
106.   }
107.
```

PhoneFormatException class:

```
1.
2. package fa.training.utils;
3.
4. public class PhoneFormatException extends Exception {
5.
6.     private static final long serialVersionUID = 1L;
7.
8.     public PhoneFormatException() {
9.         super();
10.    }
11.
12.    public PhoneFormatException(String message) {
13.        super(message);
14.    }
15. }
```

InvalidIdException class

```
1. package fa.training.utils;
2.
3. public class InvalidIdException extends Exception {
4.
5.     private static final long serialVersionUID = 1L;
6.
7.     public InvalidIdException() {
8.         super();
9.     }
10.
11.    public InvalidIdException(String message) {
12.        super(message);
13.    }
14.
15. }
```

- » Step4. Create package **fa.training.services** package that contains **StudentService**, **CourseService** classes:

StudentService class

```
1. package fa.training.services;
2.
3. // imports
```

```
4.
5. public class StudentService {
6.
7.     /**
8.      * Create new Student
9.      *
10.     * @param scanner
11.     * @return
12.     */
13.     public Set<Student> createStudent(Scanner scanner, int maxSize) {
14.         String loop = "";
15.         String id, name, phone, gender, gpa;
16.         Student student;
17.         boolean addStatus = false;
18.
19.         Set<Student> students = new HashSet<Student>();
20.         do {
21.             student = new Student();
22.
23.             // Set student id
24.             do {
25.                 System.out.println("Enter student id:");
26.                 id = scanner.nextLine();
27.                 try {
28.                     student.setId(id);
29.                 } catch (InvalidIdException e) {
30.                     continue;
31.                 }
32.                 break;
33.             } while (true);
34.
35.             // Set student name
36.             System.out.println("Enter student name:");
37.             name = scanner.nextLine();
38.             student.setName(name);
39.
40.             // Set student gender
41.             System.out.println("Enter gender:");
42.             gender = scanner.nextLine();
43.             student.setGender(gender);
44.
45.             // Set student gpa
46.             System.out.println("Enter gpa:");
47.             gpa = scanner.nextLine();
48.             student.setGpa(Double.parseDouble(gpa));
49.
50.             // Set student phone
51.             do {
52.                 System.out.println("Enter phone:");
53.                 phone = scanner.nextLine();
54.                 try {
55.                     student.setPhone(phone);
56.                 } catch (PhoneFormatException e) {
57.                     continue;
58.                 }
59.                 break;
60.             } while (true);
61.
62.             // Add student to the list
63.             addStatus = students.add(student);
64.             if (!addStatus) {
65.                 System.out.println("<<Student existed in Enroll!>>");
66.             }
67.
68.             // Continue to input?
69.             if (students.size() < maxSize) {
70.                 System.out
71.                     .println("Do you want continue to input student for this course (Y/N)? ");
72.
```

```
73.         loop = scanner.nextLine();
74.     } else {
75.         loop = "N";
76.     }
77.
78.     // Reset the set
79.     if (loop.charAt(0) != 'Y' && loop.charAt(0) != 'y') {
80.         Validator.getIds().clear();
81.     }
82. } while (loop.charAt(0) == 'Y' || loop.charAt(0) == 'y');
83.
84. return students;
85. }
86.
87. }
```

CourseService class:

```
1. package fa.training.services;
2.
3. // imports
4.
5. public class CourseService {
6.
7.     /**
8.      * Input list of course
9.      *
10.     * @param scanner
11.     * @return
12.     */
13.     public List<Course> createCourse(Scanner scanner) {
14.         String loop, id, title, credit, enrollment;
15.         double doCredit;
16.         int intEnrollment;
17.         Course course;
18.         Set<Student> students = new HashSet<Student>();
19.         List<Course> courses = new ArrayList<Course>();
20.         StudentService studentService = new StudentService();
21.
22.         do {
23.             course = new Course();
24.             // Set course id
25.             do {
26.                 System.out.println("Enter course id:");
27.                 id = scanner.nextLine();
28.                 try {
29.                     course.setId(id);
30.                 } catch (InvalidIdException e) {
31.                     continue;
32.                 }
33.                 break;
34.             } while (true);
35.
36.             // Set course title
37.             System.out.println("Enter course title:");
38.             title = scanner.nextLine();
39.             course.setTitle(title);
40.
41.             // Set course credit
42.             do {
43.                 System.out.println("Enter course credit:");
44.                 credit = scanner.nextLine();
45.                 try {
46.                     doCredit = Validator.isCredit(credit);
47.                     course.setCredit(doCredit);
48.                 } catch (NumberFormatException e) {
49.                     continue;
50.                 }
51.             } while (true);
52.         } while (true);
53.         return courses;
54.     }
55. }
```

```
51.         break;
52.     } while (true);
53.
54.     // Set course enrollment
55.     do {
56.         System.out.println("Enter course enrollment:");
57.         enrollment = scanner.nextLine();
58.         try {
59.             intEnrollment = Validator.isEnrollment(enrollment);
60.             course.setEnrollment(intEnrollment);
61.         } catch (NumberFormatException e) {
62.             continue;
63.         }
64.         break;
65.     } while (true);
66.
67.     // Set student to the course
68.     System.out.println("----Enter Student in the Course----");
69.     students = studentService.createStudent(scanner, intEnrollment);
70.     course.setStudents(students);
71.
72.     // Add course to list
73.     courses.add(course);
74.
75.     // Do you want to continue?
76.     System.out.println("Do you want continue to input course (Y/N)? : ");
77.     loop = scanner.nextLine();
78. } while (loop.charAt(0) == 'Y' || loop.charAt(0) == 'y');
79. return courses;
80. }
81.
82. /**
83.  * Save list of courses to file
84.  *
85.  * @param courses
86.  * @return
87.  * @throws Exception
88.  */
89. public String save(List<Course> courses) throws Exception {
90.     ObjectOutputStream objectOutputStream = null;
91.     try {
92.         objectOutputStream = new ObjectOutputStream(new FileOutputStream(
93.             Constant.FILE_PATH));
94.         objectOutputStream.writeObject(courses);
95.     } catch (Exception exception) {
96.         throw new Exception();
97.     } finally {
98.         if (objectOutputStream != null) {
99.             objectOutputStream.close();
100.        }
101.    }
102.    return Constant.SUCCESS;
103. }
104.
105. /**
106.  * Get all the course in file
107.  *
108.  * @return
109.  * @throws IOException
110.  */
111. @SuppressWarnings("unchecked")
112. public List<Course> getAll() throws IOException {
113.     ObjectInputStream objectInputStream = null;
114.     List<Course> courses;
115.     try {
116.         objectInputStream = new ObjectInputStream(new FileInputStream(
117.             Constant.FILE_PATH));
118.         courses = (List<Course>) objectInputStream.readObject();
119.     } catch (Exception exception) {
```

```
120.         throw new IOException();
121.     } finally {
122.         if (objectInputStream != null) {
123.             objectInputStream.close();
124.         }
125.     }
126.     return courses;
127. }
128.
129. /**
130.  * Sort and display the course by id
131.  *
132.  * @param listBook
133.  */
134. public void sortAndDisplay(List<Course> courses) {
135.
136.     Collections.sort(courses, new CourseTitleCompare());
137.
138.     System.out.println("-----COURSE LIST-----");
139.
140.     for (Course course : courses) {
141.         System.out.format("%s%20s%12.3f%5d%100s\n", course.getId(),
142.             course.getTitle(), course.getCredit(), course.getEnrollment(),
143.             course.getStudents());
144.     }
145. }
146.
147. public List<Course> getByStudent(String studentId) throws IOException {
148.     List<Course> courses = getAll();
149.     List<Course> coursesByStudent = new ArrayList<Course>();
150.
151.     if (courses != null) {
152.         for (Course course : courses) {
153.             for (Student studentCourse : course.getStudents()) {
154.                 if (studentId.equalsIgnoreCase(studentCourse.getId())) {
155.                     coursesByStudent.add(course);
156.                 }
157.             }
158.         }
159.     }
160.     return coursesByStudent;
161. }
162.
163. public String remove(String id) throws Exception {
164.     boolean removed = false;
165.
166.     List<Course> courses = getAll();
167.     if (courses == null) {
168.         throw new IOException();
169.     }
170.     Iterator<Course> iterator = courses.iterator();
171.     while (iterator.hasNext()) {
172.         Course course = iterator.next();
173.         if (id.equalsIgnoreCase(course.getId())) {
174.             iterator.remove();
175.             removed = true;
176.             break;
177.         }
178.     }
179.
180.     if (removed) {
181.         try {
182.             // update list
183.             save(courses);
184.         } catch (Exception e) {
185.             throw new Exception();
186.         }
187.
188.         return Constant.SUCCESS;
```

```
189.     }
190.     return Constant.FAIL;
191. }
192.
193. }
```

- » Step4. Create package **fa.training.main package** that contains **CourseManagement** classes:

CourseManagement class

```
1. package fa.training.main;
2.
3. // import
4.
5. public class CourseManagement {
6.
7.     private static List<Course> listNewCourse;
8.     private static List<Course> listCourse;
9.
10.    public static void main(String[] args) {
11.        String choice, status, studentId, courseId;
12.        Scanner scanner = null;
13.        List<Course> courseByStudent;
14.        CourseService courseService = new CourseService();
15.        try {
16.            scanner = new Scanner(System.in);
17.            do {
18.                System.out.println("-----MENU-----");
19.                System.out.println("1. Create new Course"
20.                    + "\n2. Save to File"
21.                    + "\n3. Sort by Id\n"
22.                    + "4. Find by Student"
23.                    + "\n5. Remove course"
24.                    + "\n6. Course Statistic\n7. Exit ");
25.
26.                System.out.print("Select: ");
27.                choice = scanner.nextLine();
28.                choice = choice.trim();
29.                switch (choice) {
30.                    // Create new Course
31.                    case Constant.INPUT:
32.                        if (listNewCourse != null) {
33.                            listNewCourse.clear();
34.                        }
35.                        listNewCourse = courseService.createCourse(scanner);
36.                        System.out.println("Input done!");
37.                        break;
38.
39.                    // Save to file
40.                    case Constant.SAVE:
41.                        try {
42.                            if (listNewCourse == null) {
43.                                throw new Exception();
44.                            }
45.                            status = courseService.save(listNewCourse);
46.                            System.out.println("Save: " + status);
47.                        } catch (Exception e) {
48.                            System.out.println("Save Fail!");
49.                        }
50.                        break;
51.
52.                    // Sort from file
53.                    case Constant.SORT:
54.                        if (listCourse != null) {
55.                            listCourse.clear();
56.                        }
57.                        try {
```



```
58.         listCourse = courseService.getAll();
59.         if (listCourse == null) {
60.             throw new IOException();
61.         }
62.         courseService.sortAndDisplay(listCourse);
63.     } catch (IOException e) {
64.         System.out.println("No data");
65.     }
66.     break;
67.     // Search all the courses that a student has taken
68. case Constant.SEARCH:
69.     try {
70.         System.out.println("Enter student id: ");
71.         studentId = scanner.nextLine();
72.         courseByStudent = courseService.getByStudent(studentId);
73.
74.         if (courseByStudent == null) {
75.             throw new IOException();
76.         }
77.         System.out.println("---List of Students---");
78.         for (Course course : courseByStudent) {
79.             System.out.println(course.getId() + "\t"
80.                 + course.getTitle() + "\t"
81.                 + course.getCredit()
82.                 + "\t" + course.getEnrollment());
83.         }
84.     } catch (IOException e) {
85.         System.out.println("No Data!");
86.     }
87.     break;
88.     // Remove a course from list courses
89. case Constant.REMOVE:
90.     System.out.println("Enter course id to remove: ");
91.     courseId = scanner.nextLine();
92.
93.     try {
94.         status = courseService.remove(courseId);
95.         System.out.println("Remove: " + status);
96.     } catch (Exception e) {
97.         System.out.println("Remove Fail!");
98.     }
99.
100.    break;
101.    // Course statistic
102. case Constant.STATS:
103.     if (listCourse != null) {
104.         listCourse.clear();
105.     }
106.     try {
107.         listCourse = courseService.getAll();
108.         if (listCourse == null) {
109.             throw new IOException();
110.         }
111.         for (Course course : listCourse) {
112.             System.out.println("---" + course.getId() + "----");
113.             for (Entry<String, Integer>entry: course.getStatistic()
114.                 .entrySet())
115.                 {
116.                     System.out.println(entry.getKey()
117.                         + " " +entry.getValue());
118.                 }
119.         }
120.     } catch (IOException e) {
121.         System.out.println("No data");
122.     }
123.     break;
124. default:
125.     choice = Constant.EXIT;
126.     break;
```

```

127.         }
128.         } while (!choice.equalsIgnoreCase(Constant.EXIT));
129.     } finally {
130.         if (scanner != null) {
131.             scanner.close();
132.         }
133.     }
134. }
135.
136. }

```

Outputs:

```

-----MENU-----
1. Create new Course
2. Save to File
3. Sort by Id
4. Find by Student
5. Remove course
6. Course Statistic
7. Exit
Select: 1
..

```

Select: 3

```

-----COURSE LIST-----
JA001  JAVA    2.000  2  [Student [id=123456, name=DUNG, phone=0987654321, gender=MALE, gpa=8.0]]
SQ001  SQL      3.000  3  [Student [id=246810, name=An, phone=0988776655, gender=Female, gpa=9.0]]

```

Select: 4

```

Enter student id:
123456
---List of Courses---
JA001    JAVA    2.0    2
SQ001    SQL     3.0    3

```

Select: 6

```

----JA001----
A 0
B 1
C 0
D 0
----SQ001----
A 1
B 1|
C 0
D 0

```

-- THE END --