Assignment 5

Contributors:

Designer: HU Chunfeng, CHEN xingyu, XIONG zhuochen

Tester： HU Chunfeng

update history:

May 7, 2021

1. revised description of **Student.courseExist()**

to check whether the course is already in schedule or not. Since a student cannot select 2 courses with the same code, name and type. ***If there is already a course with the same code, name and type in the student's schedule as the parameters given, it returns true. Otherwise, it returns false.***

2. add some **getter and setter** methods into classes already used in test cases.

   Building.setRooms
   Building.getRooms
   Course.setTime
   Course.setType
   Course.getStudents

May 6, 2021

1. revised interface name **changeCourse()** in **CourseOperator** . The method in class student and teacher is correct before.

```
 boolean changeCourse( Course oldCourse1 , Course newCourse2 );
// to change oldCourse1 to newCourse2 and update the schedules.
```

2. revised description of public method **Student.changeCourse()** implements the interface

A student could change a new course with different time or teacher but with the same course code and type. Both with the same or different teachers are OK. Both with the same or different classrooms are OK. If it returns true, the old course would be deleted from the student's schedule. And the new course would be added to the schedule. ***The student should also be deleted from the student list of the old course and added to the new course*.** If there is not an available course for the student to choose at the time, it returns false and does not change the schedule.

```
public boolean changeCourse( Course oldCourse1 , Course newCourse2 )
```

3. some typing and grammar errors in description.

May 5, 2021

1. revised error messages of **Classroom.addCourse()** if more than 1 error(May 5) :

```
public String addCourse(Course course){}
```

Return messages are as follows:

> **OK: Adding course to classroom success.**
> **ERROR: Another course already exists at the time.**
> **ERROR: Course type not same as classroom.**
> **ERROR: Not enough seats in the classroom for this course.**

If there are more than 2 errors to add a course to a classroom at the time, it will show only 1 error message at the order above.

2. revised description of **Course.getAbbrevName()**(May 5)

getAbbrevName() should get the abbreviation of the course name if it has been set. But if is empty, it will be automatically generated by every first character in caption of each word in course name, except some words like **to, of, the, in,and**. For example, **getAbbrevName()** of the course **"Introduction to Computer Programming A"** returns **"ICPA"** if the AbbrevName is not set before. It will return **"JavaA"** if it have been set as setAbbrevName("JavaA").

3. revised description of **Teacher.getFreeClassroom()**(May 5)

to the get available classrooms from the classroom list in **Db.buildings**. According to the teacher's prefer location and it returns only the classrooms in the prefer location. If there is no available classrooms in prefer location. It returns classrooms in other locations. It returns an empty list if there isn't any available classroom for the course.

4. **LocalJude.java, Db.java** updates according to this document.(May 5)

Problems:

Problem 1: Design a class named **Building** and some basic enum structures and simple classes[10 marks]

Problem 2: Design a class named **Classroom** [20 marks]

Problem 3: Design a class named **Course** [20 marks]

Problem 4: Design a class named **Teacher** inheriting **Person** implements **CourseOperator** interface [25 marks]

Problem 5: Design a class named **Student** inheriting **Person** implements **CourseOperator** interface [25 marks]

Notice:

> 1. There have been some static ArrayList of students, teachers, building and courses to initiate some basic data in a class **Db** provided for you. There are also classes **CourseTime** and interface **CourseOperator** already for you. Of course, it needs invoke some methods you should finish. You can design more classes and more methods in classes if you need, and submit them all together.
> 2. It is **not necessary** to consider data consistency in the class above except for those mentioned in description below. For example, a teacher could drop a course while not

> checking whether some students have selected this course. It should be used before students to select course.
> 3. There is also a **junit** test file for you to test the classes above you need finish. Most but not all method would be tested in the test cases.
> 4. You should strictly write the class with the same names, attributes and methods with no package in source codes.

It is a simple educational administration system for the teachers and students to arrange classes in SUSTech. It should have at least the following functions:

1. The teachers can use this system to arrange their courses, such as choosing the time and the location of the class he teaches before new semester.
2. The students can use this system to choose their courses they want to take and drop the class they don't like.
3. The system can print the classes schedule of a teacher, a student and a classroom.

Problem 1: Design a class named Building.

The class **Building** should have the following attributes.

```
private List<Classroom> rooms;
private Location location;
private int id;
```

The Class **Building** should have the following methods:

1. **Constructor** with **location, id** or with no argument

```
public Building( )
public Building( Location location , int id )
```

2. some getter and setter methods

```
public Location getLocation() {}
public void setLocation( Location location ) {}
public int getId() {}
public void setId( int number ) {}
public List<Classroom> getRooms() {}
public void setRooms( List<Classroom> rooms ) {}
```

3. public method **addRoom(Classroom classroom), deleteRoom(Classroom classroom)**

To add or delete a classroom to the building. Since there is a reference attribute of Building of Classroom, it will be added or deleted successfully if they are the same building. Otherwise, it would return false. It returns false if to delete a classroom not in the building.

```
public boolean addRoom( Classroom room ) {}
public boolean deleteRoom( Classroom room ) {}
```

4. override method **toString()**

to get the building location name and id. For example, Building 6 of LycheePark should be

> LP#6

and Building 2 of TeachingBuilding should be

> TB#2

Other basic Enum types and classes.

1. Enum type of **Location** with values of **LycheePark, TeachingBuilding**
2. Enum type of **CourseType** with values of **Lecture, Lab**
3. enum type of **Day** with values of **Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday**
4. design an **abstract** class named **Person** with attributes and constructor

```
    String id;
    String name;
    Map<CourseTime, Course> schedule;
```

The schedule is used to store the courses at any CourseTime in a week. You might use some of the codes with **HashMap** below. And you might get more details from books and internet.

```
schedule = new HashMap<>( );
schedule.put( course.getTime( ) , course );
schedule.remove( course.getTime( ) );
```

5. A class named **CourseTime** with attributes, constructor and override method of equals() and hashCode() is already provided.

```
private Day day;
private int time;//1 for 8:00-9:50, 2 for 10:20-12:10, 3 for 14:00-15:50, 4 for 16:20-18:10, 5 for 19:00-20:50

public CourseTime( Day day , int time ) {
    this.day = day;
    this.time = time;
}

@Override
public boolean equals( Object o ) {
    if ( this == o ) return true;
    if ( o == null || getClass( ) != o.getClass( ) ) return false;
    CourseTime that = (CourseTime) o;
    return time == that.time && day == that.day;
}

@Override
public int hashCode() {
    return Objects.hash( day , time );
}
```

Problem 2: Design a class named Classroom.

The class **Classroom** should have the following variables.

```
int id;//eg:101
int seatNum;//eg:50
CourseType type;// Lecture or Lab
Building building;
Map<CourseTime, Course> schedule;
```

1. **Constructor** with **id, seatNum, type, building, CourseType** or with no argument and some getter and setter methods.

```
public Classroom( int id , int seatNum , Building building, CourseType type )
```

2. override method **toString()**

to get the classroom type, id and seatNum, and building name and id. For example, Lecture Room 401, Lab Room 402, Building 6 of LycheePark should be like **Classroom.type+"R"+Classroom.id+Classroom.SeatNum+Building.toString()**

> LectureR401(60)LP#6
> LabR402(50)LP#6

3. public method **addCourse()** with **Course**

to judge whether this course could be arranged in the classroom, and add the course to the schedule of the classroom.

It will be successful, if the classroom is empty at the course time, and the course type is the same as the classroom's courseType, and the capacity of course is not more than the seat numbers. Then this course will be added to the **schedule** of the classroom. Otherwise, the course can not be held in the classroom, return some error messages, and do not add to the **schedule**.

```
public String addCourse(Course course){}
```

Return messages are as follows:

> **OK: Adding course to classroom success.**
> **ERROR: Another course already exists at the time.**
> **ERROR: Course type not same as classroom.**
> **ERROR: Not enough seats in the classroom for this course.**

If there are more than 2 errors to add a course to a classroom at the time, it will show only 1 error message at the order above.

4. public method **deleteCourse()** with **Course**

It will be true, if the course is in the schedule of the classroom and delete it from the **schedule**.

```
public boolean deleteCourse(Course course){}
```

5. public method **getCourse()** by **CourseTime** to get the class arranged in the classrom at the time. If there is no class at the time , return **null** .

```
public Course getCourse(CourseTime courseTime){}
```

6. public method **printSchedule()**

```
public String printSchedule()
```

To print the schedule of a classroom as follow to the returned string:

> Classroom.toString() Schedule
> Monday
> 1 Course.code, Course.abbrevName, Teacher.name
> 2
> 3 Course.code, Course.abbrevName, Teacher.name
> 4 Course.code, Course.abbrevName, Teacher.name
> 5
> Tuesday
> ...
> Sunday
> ...

Notice:

> If there is not a course at the time slot, it may print just the number and blank in a line in every weekdays.

7. Public method **getScheduleCourseNum()**

To get the course number of the classroom schedule.

```
public int getScheduleCourseNum()
```

Problem 3: Design a class named Course.

The class **Course** should have the following variables.

```
    static int idCnt = 0;// number of courses created
    int id;//generated automatically from 1
    String name; // Introduction to Computer Programming A
    String abbrevName;// JavaA
    String code; // CS102A
    CourseTime time;
    Teacher teacher;
    Classroom room;
    List<Student> students;// who selected this course
    int capacity;// maxium number of students
    CourseType type;//Lecture, Lab
```

Notice:

> You may use a static **idCnt** to count the number of courses created, and set it as **id**.

The class **CourseType** and **CourseTime** are all the enumeration class.

The Class **Course** should have the following methods:

1. some constructors of **Course**

The student list should be initialized when a course constructed.

```
    public Course( String code , String name , String abbrevName , Teacher
teacher , int capacity , CourseType type )
    public Course( String code , String name , String abbrevName , Teacher
teacher , int capacity , CourseType type, CourseTime time , Classroom room)
```

Notice:

> If the course constructor is with classroom and time, it would just set the attributes of the course, but not be added to the classroom's schedule in the constructor automatically. It would be added after checking the classroom's schedule by **Classroom.addCourse()**.

2. some getter and setter methods

```
    public String getAbbrevName() { }

    public void setAbbrevName(String abbrevName) { }

    public void setAbbrevName() {  }

    public CourseTime getTime() {  }

    public void setTime(CourseTime time) {  }

    public Teacher getTeacher() {  }

    public void setTeacher(Teacher teacher) { }

    public Classroom getRoom() { }

    public void setRoomTime(Classroom room, CourseTime time) {  }

    public int getCapacity() {  }

    public void setCapacity(int capacity) {  }

    public void setType(CourseType type) {  }

    public CourseType getType() {  }

    public List<Student> getStudents() {  }
```

Notice:

> **getAbbrevName()** should get the abbreviation of the course name if it has been set. But if is empty, it will be automatically generated by every first character in caption of each word in course name, except some words like **to, of, the, in,and**. For example, **getAbbrevName()** of the course **"Introduction to Computer Programming A"** returns **"ICPA"** if the AbbrevName is not set before. It will return **"JavaA"** if it have been set as setAbbrevName("JavaA").

3. public method **setRoomTime()**

To set the time and classroom for the course. If the course is created by constructor without classroom and time, this method is needed.

```
    public void setRoomTime( Classroom room, CourseTime time )
```

4. public method **addStudent()**

To add a student to the list of those students who choose the course. If the student is already in the list, it returns false to add into the list. Otherwise, the student would be added to the list and it returns true.

```
public boolean addStudent (Student student){}
```

5. public method **deleteStudent()**

To delete a student from the list of those students who choose the course. If the student is already in the list, it returns true to delete from the list. Otherwise, it returns false.

```
public boolean deleteStudent (Student student){}
```

Problem 4: Design a class named Teacher inheriting Person implements CourseOperator interface

The class **Teacher** should have a new attribute and getter, setter methods.

```
private Location preferLocation;

public Location getPreferLocation() {}

public void setPreferLocation( Location preferLocation )
```

It shows that the teacher would like the course to be in the preferred location than other locations. For example, If there are both some classroom available in two locations, the teacher will choose the classroom in the preferred location. The teacher will choose the classroom in other locations only if there is not any available classroom in the preferred location.

1. **Constructor** with **id, name** or with no argument

```
    public Teacher(){}
    public Teacher( String id , String name ) {}
```

2. public method **getFreeClassroom** by **CourseTime, capacity, type**

to the get available classrooms from the classroom list in **Db.buildings**. According to the teacher's prefer location and it returns only the classrooms in the prefer location. If there is no available classrooms in prefer location. It returns classrooms in other locations. It returns an empty list if there isn't any available classroom for the course.

```
  public List<Classroom> getFreeClassroom(CourseTime time, int capacity,
CourseType type )
```

3. public method **createCourse()**

to create a course and add it to the teacher's schedule.

It should be checked whether the course in already on the teacher's schedule. It cannot create the course and return false if the course already exists on the teacher's schedule.

A course with course time and classroom could be created successfully only the classroom is empty at the course time, and the course type is the same as the classroom's courseType, and the capacity of course is not more than the seat numbers. Then this course will be added to the teacher's **schedule** and to the classroom's **schedule**. Otherwise the course can not be in the classroom, return **false** and do not add the the **schedule**.

```
boolean createCourse(Course course ) {}
boolean createCourse(String code , String name , String abbrevName, CourseTime
time , Classroom room , int capacity , CourseType type ) {}
```

4. public method **dropCourse()** implements the interface

If the course is not already on the schedule of the teacher, return false.

Else remove the selected course in teacher's schedule and classroom's schedule, return true.

```
public boolean dropCourse( Course course )
```

Notice:

> The teacher can only drop or change a course before students select courses. So it is not necessary to check students' schedules in this method.

5. public method **changeCourse()** implements the interface

A teacher could change his course time and classroom. There must be the same course code and type and teacher. If the classroom is available at the new time and the teacher is free, it returns true. Otherwise it returns false. If it returns true, the old course would be deleted from the teacher's schedule and classroom's schedule, and the new one would be added in.

```
public boolean changeCourse( Course oldCourse1 , Course newCourse2 )
```

6. public method **printSchedule()**

```
public String printSchedule()
```

To print the schedule of a teacher as follow to returned string:

> Teacher.name's Schedule
> Monday
> 1 Course.code, Course.abbrevName, Classroom.toSting()
> 2
> 3 Course.code, Course.abbrevName, Classroom.toSting()
> 4 Course.code, Course.abbrevName, Classroom.toSting()
> 5
> Tuesday
> ...
> Sunday
> ...

Notice:

> If there is not a course at the time slot, it may print just the number and blank in a line in every weekdays.

7. Public method **getScheduleCourseNum()**

To get the course number of the schedule.

```
public int getScheduleCourseNum()
```

Problem 5: Design a class named Student inheriting Person implements CourseOperator interface

1. **Constructor** with **id, name** or with no argument

```
    public Student(){}
    public Student( String id , String name ) {}
```

2. public method **courseExist()**

to check whether the course is already in schedule or not. Since a student cannot select 2 courses with the same code, name and type. If there is a course with the same code, name and type in the student's schedule as the parameters given, it returns true. Otherwise, it returns false.

```
public boolean courseExist( String code , String name , CourseType type )
public boolean courseExist( Course course )
```

3. public method **chooseCourse()**

A student can choose a course from the course list if

> 1. hasn't had this course before. One can not choose 2 courses with same name, code and type, but different teachers. One can choose 2 courses with same name and code, but different types, eg JavaA lecture and lab, no matter by a same teacher or different teachers.
>   **Notice**: For example, students can choose **1** javaA lecture and **1** lab course from any teacher. It is not considered whether students of a lab class are from the same lecture, which is different from the actual situation in SUSTech.
> 2. and doesn't have class during the course time
> 3. and the course capacity is not full.

If a student choose the course successfully, the course will be added to the student's schedule and to add the student to the student list of the course, then it returns true. Otherwise, it return false.

```
boolean chooseCourse( Course course){}
```

4. public method **dropCourse()** implements the interface

If the course is not on the schedule of the student, return false.

Else remove the selected course in student’s schedule and remove the student from the student list of the course, return true.

```
public boolean dropCourse( Course course )
```

5. public method **changeCourse()** implements the interface

A student could change a new course with different time or teacher but with the same course code and type. Both with the same or different teachers are OK. Both with the same or different classrooms are OK. If it returns true, the old course would be deleted from the student's schedule. And the new course would be added to the schedule. The student should also be deleted from the student list from the old course and added to the new course. If there is not an available course for the student to choose at the time, it returns false and does not change the schedule.

```
public boolean changeCourse( Course oldCourse1 , Course newCourse2 )
```

6. public method **printSchedule()**

```
public String printSchedule()
```

To print the schedule of a student as follow to the returned string:

> Student.name()'s Schedule
> Monday
> 1 Course.code, Course.abbrevName, Teacher.name, Classroom.toSting()
> 2
> 3 Course.code, Course.abbrevName, Teacher.name, Classroom.toSting()
> 4 Course.code, Course.abbrevName, Teacher.name, Classroom.toSting()
> 5
> Tuesday
> ...
> Sunday
> ...

Notice:

> If there is not a course at the time slot, it may print just the number and blank in a line in every weekdays.

7. Public method **getScheduleCourseNum()**

To get the course number of the schedule.

```
public int getScheduleCourseNum()
```

CourseOperator interface

```
public interface CourseOperator {
// some common operators of coursees for both teachers and students.
// which may be implemented differently for teachers and students

    boolean dropCourse( Course course );
    // to drop the course and delete it from schedule.
```

```
    // There is not necessary to check whether some students have had the ccourse
for teachers in this method.

    boolean changeCourse( Course oldCourse1 , Course newCourse2 );
    // to change oldCourse1 to newCourse2 and update the schedules.
    // A student could change a new course with different time or teacher but
with the same course code and type. Both with the same or different teachers are
OK. Both with the same or different classrooms are OK. If it returns true, the
old course would be deleted from the student's schedule. And the new course would
be added to the schedule. If there is not an available course for the student to
choose at the time, it returns false and dose not change the schedule.
    // A teacher could change his course time and classroom. There must be the
same course code and type and teacher. If the classroom is available at the new
time and the teacher is free, it returns true. Otherwise it returns false. If it
returns true, the old course would be deleted from the teacher's schedule and
classroom's schedule, and the new one would be added in.
```