CSC321/545, Instructor: Zhen Jiang Sample Assignment: UML design

**Assignment:**

Use UML to design classes/tables for a student grading system. The system needs to handle the following functions for a person:

* 1. The student who can view his/her grade of any homework, project, quiz, test, and final letter grade in any class he/she took.
  2. The teacher who can update the above information in the system, including addition of all student records at the beginning of class.

**Key part in your design:**

1. **Use cases with 4 steps**
2. **List of pieces of information**
3. **Class diagram with name, attributes, and relationships that represents the tables**

-----------------------------------Design Solution Part (a)-------------------------------------------

Q: What is Use Cases?

A: See material DB\_uc.ppt.

Q: What and where to start with?

A: Primary actor and its success story. Please follow the Q/A in the below.

Q: Who is going to use this grading system?

A: Students and teachers

Q: Are they (primary) actors?

A: Yes. But use a singular role, say a student, which represents every student.

Q: Can each actor have multiple roles?

A: Yes. For instance, the teacher/instructor can create the records and update the scores.

Q: Can one case include all roles?

A: No. Each case must have sole role. For instance, we have two actors, student and instructor, but total 3 roles, student view, instructor creation, and instructor update. Therefore, we need 3 Use cases.

Q: Is that one case one role?

A: Yes. See the below.

|  |  |
| --- | --- |
| Case # | 1 |
| Step 1: primary actor and goal | Student views the score (goal = case name) |
| Step 2: main scenario (cannot be simpler, always ok, happily ends) | 1. Provide with course information. 2. See the grades. 3. Repeat the above until no more class is needed |
| Step 3: Failures | In step 1 of main scenario, information can be mismatched –  **When** the target course is taken with   1. Different course name, 2. Different instructor, 3. Different year, and 4. Different section.   In step 1 of main scenario, the user is unauthorized and tried to view the informationbeyond his/her access right –  **When**   1. Identity of this student is incorrect.   In step 2 of main scenario, display information may be insufficient and incomplete –  **When**   1. The display is not specified and the detailed request is expected. |
| Step 4: alternative scenarios (from exceptions in the above) | a-d. The existence of the course and the relation between the course and this student must be confirmed by the instructor in the creation case in advance. If the process fails with any reason, the view access must be denied. Process go back to step 1 of main scenario.  e. Login process is needed before step 1 of main scenario, in order for the user toobtain the right data access. After that, step 1 can be executed.  f. See case 1.1 for a detail look.After that, the process reaches the end of the main scenario. Note that those failures caused by incompleteness or insufficiency must be implemented with more detailed subpart information, triggering a lower level case.  \*Note that any failure that is incapable to address its reason and time to occur will be ignored and be consideredas the precondition. |



|  |  |
| --- | --- |
| Case # | 1.1 (sub-case of 1) |
| Step 1: primary actor and goal | Student sees the grades. |
| Step 2: main scenario | Derived from step 2 in case 1:   1. Choose to display individual record or overallrecord. 2. Under the “**all**” mode, gather all the scores in record and display them. 3. Under the “**individual**” mode, select the target item and then see the display.   Note that both steps 2 and 3 are too complicated to fit in the single goal here. So we consider a lower level implementation (of UC1.1.1 and UC 1.1.2). |
| Step 3: Failures | None |
| Step 4: alternative scenario | N/A |

Rather than using the following complicated scenarios,



we implement UC1.1 with UC1.1.1 and UC1.1.2.



|  |  |
| --- | --- |
| Case # | 1.1.1 (sub-case of 1.1) |
| Step 1: primary actor and goal | Student sees all scores together. |
| Step 2: main scenario | Derived from step 2 in case 1.1:   1. Gather all the scores. 2. Display. |
| Step 3: Failures | None |
| Step 4: alternative scenario | N/A |



|  |  |
| --- | --- |
| Case # | 1.1.2 (sub-case of 1.1) |
| Step 1: primary actor and goal | Student sees each category scores. |
| Step 2: main scenario | Derived from step 3 in case 1.1:   1. Determine the category and display. 2. Repeat the above when there is another item to show. |
| Step 3: Failures | In step 1 of main scenario, selection could be wrong –  **When**  Unavailable category (quiz, project, homework, test, and final)is selected. |
| Step 4: alternative scenario | Repeat the selection process until the input is valid. |



|  |  |
| --- | --- |
| Case # | 2 |
| Step 1: primary actor and goal | Instructor creates records at the beginning of semester. |
| Step 2: main scenario | Precondition: In the routine, such as D2L, the enrolled students are assigned to each instructor’s class automatically. Here, we assume the instructor knows the correct list of students and will select his/her students to class from the existing student pool by inputtheir names one by one. |
| 1. Create the class record. 2. Create student records one by one. |
| Step 3: Failures | In step 1, the class input could be wrong –  **When**   1. Incorrect information (name, year, and section) is used. 2. Course (name, year, and section) exists.   In step 2, the system could go wrong –  **When**   1. Input student name is not found. 2. Input student name is not unique. |
| Step 4: alternative scenario | 1. Provide a confirmation page. In case already existing or incorrect, redo the input. 2. Go to update process, i.e., UC3. 3. Provide a warning message to force the instructor input the right name. 4. Provide a list for instructor to select. |



|  |  |
| --- | --- |
| Case # | 3 |
| Step 1: primary actor and goal | Instructor updates student score. |
| Step 2: main scenario | 1. Choose the course. 2. Choose the category (quiz, HW, project, test, or final). 3. Choose the student and write up the score. |
| Step 3: Failures | In step 1, the update could be wrong –  **When**   1. The instructor is not authorized, i.e., course is not created under his/her instructor ID.   In steps1-2, the system can go wrong –  **When**   1. Incorrect item is selected.   In steps 3, the system can go wrong –  **When**   1. Score is mistyped. |
| Step 4: alternative scenario | 1. Login process is needed as step 1 of UC2. 2. Asked to reenter. 3. Simply ignore it here. |



-----------------------------------Design Solution Part (b)-------------------------------------------

|  |  |
| --- | --- |
| Information covered | Object/Class |
| Instructor name (UC1) | Instructor |
| Instructor id (UC3) |
| Instructor login password (UC3) |
| Course name (UC1) | Course-for-grade |
| Year offered (UC1) |
| Section # (UC1) |
| Instructor (UC1) – relation, not attribute, |
| Registered students – relation, not attribute |
| Grading category |
| Grading score |
| Student name (UC2, UC3) | Student |
| Student id (UC1) |
| Student login password (UC1) |

Q: Why do we determine the objects like these?

A: See the definition of object in material DB\_uml.ppt.

-----------------------------------Design Solution Part (c)-------------------------------------------

-----Non-SQL Java Implementation with System and Scanner, without DB involved-------



public abstract class Course\_for\_Grade {

// find a key to determine one of the 1-to-m relation with instructors

// assume no collaborative class  
private Instructor ins;

// find a key to determine 1-to-n relation with students  
private Student stu;

// course info.

private String name;  
private int year;  
private int section;

// above identity are used to determine m-to-n relation between

// instructor and student

// each object represents a student in a specified course relation with

// a faculty. Many object will have identical student information but

// have different course link and grading records, which indicates

// the records in different courses taken by this student.

// only reference, no new (storage) involved

private String [] category;

private int [] score;

private Course\_for\_Grade(){

//This is called by an instructor.

}  
…

}//end of class Course\_for\_Grade

public abstractclass User {

//login table  
protected String name;  
protected String id;  
protected String passwd;

public boolean login(String name, String passwd){

…

}  
public abstract void write();

//for student to reset passwd and

// faculty to reset both passwd and student grading info.  
public abstract String read();  
// for matching the passwd and relevantgrading information

//for abstract class not being able to have constructor:

public String get\_name(){return name;}  
public String get\_id(){return id;}  
public String get\_passwd(){returnpasswd;}  
public void set\_name(String name){this.name=name;}  
public void set\_id(String id){this.id=id;}  
public void set\_passwd(String passwd){this.passwd = passwd;}

}//end of class User

public class Faculty extends User {

// each faculty can initiate the course relation since all classes

// are accessible.

public void create(int [] numbers, String [] names){

//for create the course object and relevant student records

//at the beginning

…

}

public void create(String name, String course\_name, intnew\_score){

//update the student record individually

…

}  
…

}//end of class Faculty

public class Student extends User {

public void write(…){

//need authorization passed from the instructor

…

}

public void read(…){

//no constraint, can be called at everywhere, by student or faculty

…

}

} //end of class Student

------------------SQL Java Implementation, involved with JDBC and MySQL---------

1. Student, Instructor, User are of the same size -> User only
2. Table User vs. class User
3. Table course\_for\_grade vs. class

User (id, name, passwd)

Course\_for\_Grade( Name, Year, Section, tid, did, category, score)



1. Implement all users and interface in java classes.



See DB server from the result in the above.

The application client will be implemented as following:



-----------------------------------Design Solution Part (d) Demo-----------------------------------

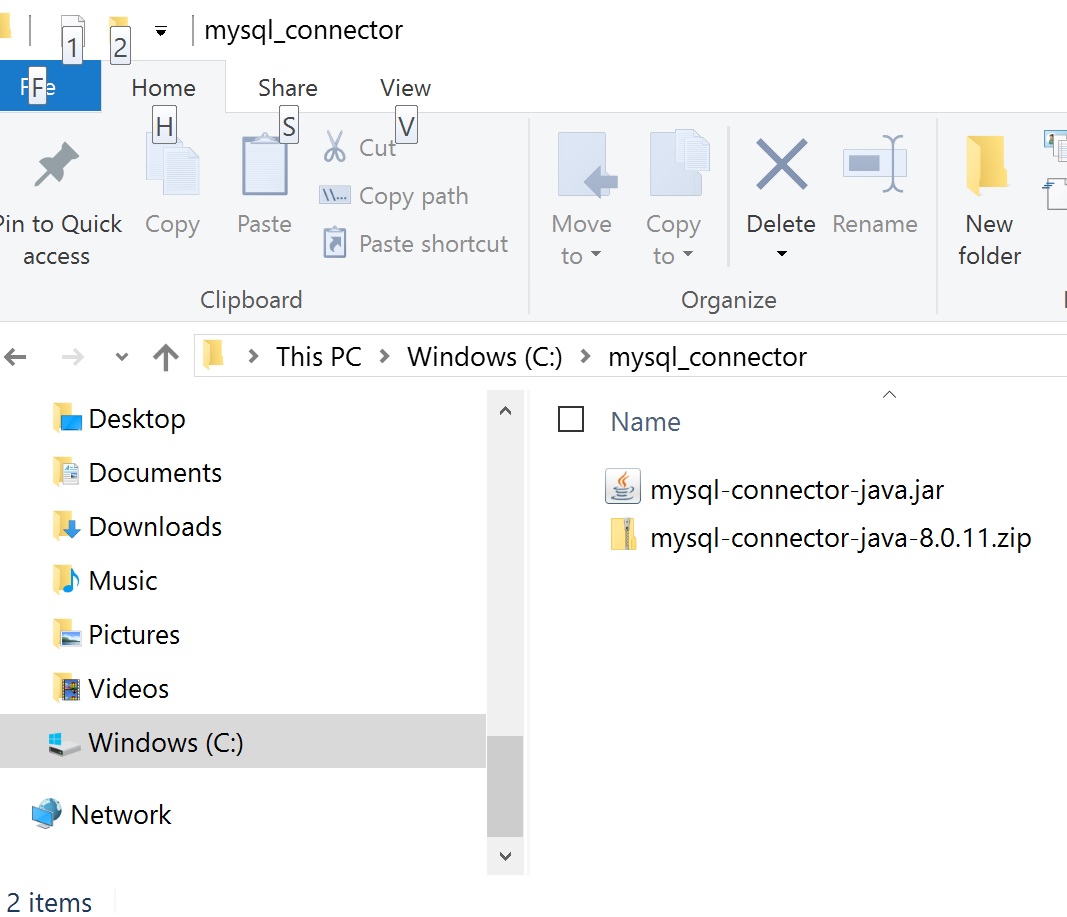
1. Download and install Java (e.g., JDK from

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>, not using JRE version)

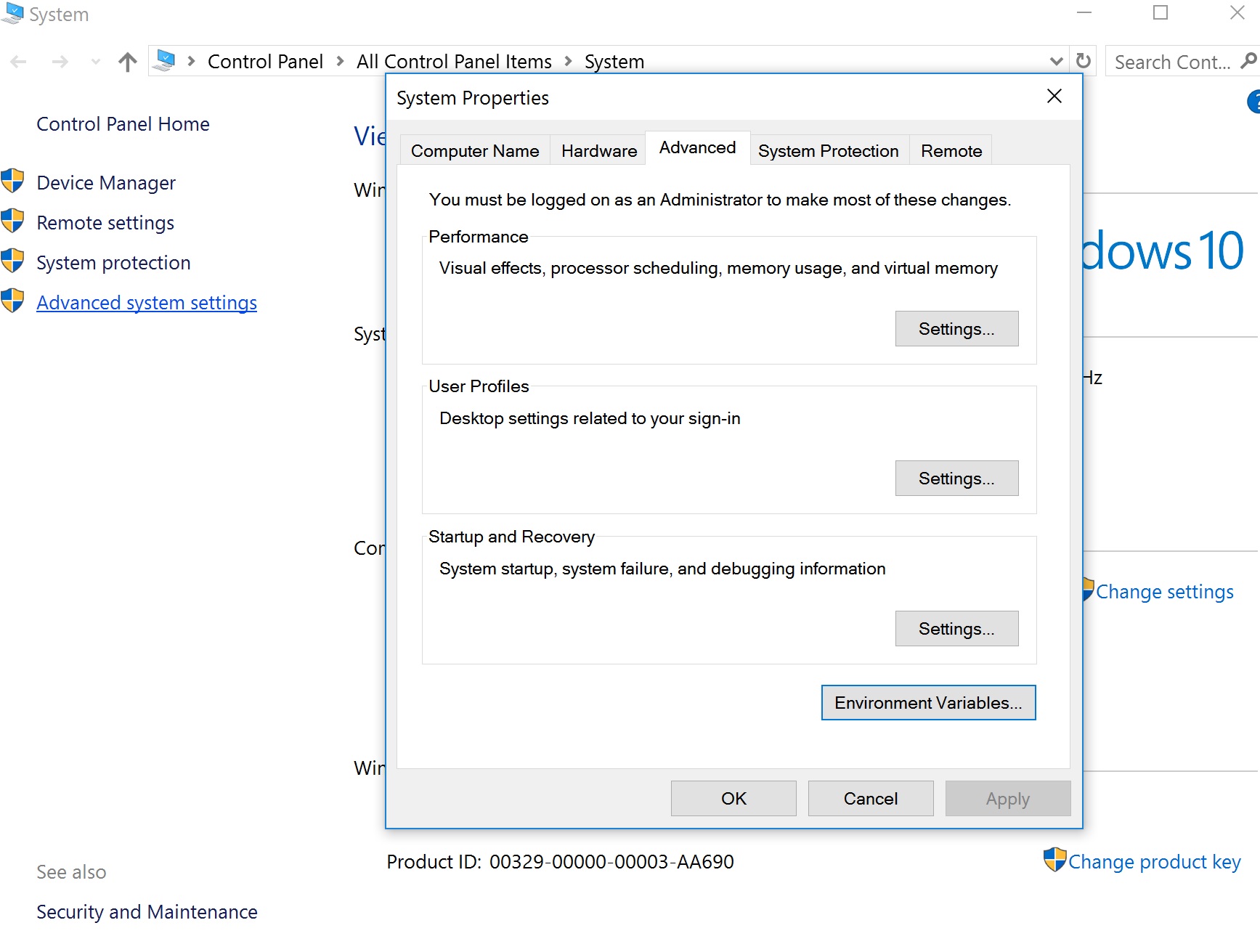
1. Download and install MySql (from

<https://dev.mysql.com/downloads/file/?id=411874> or run the install file from umlDemo.zip available in D2L).

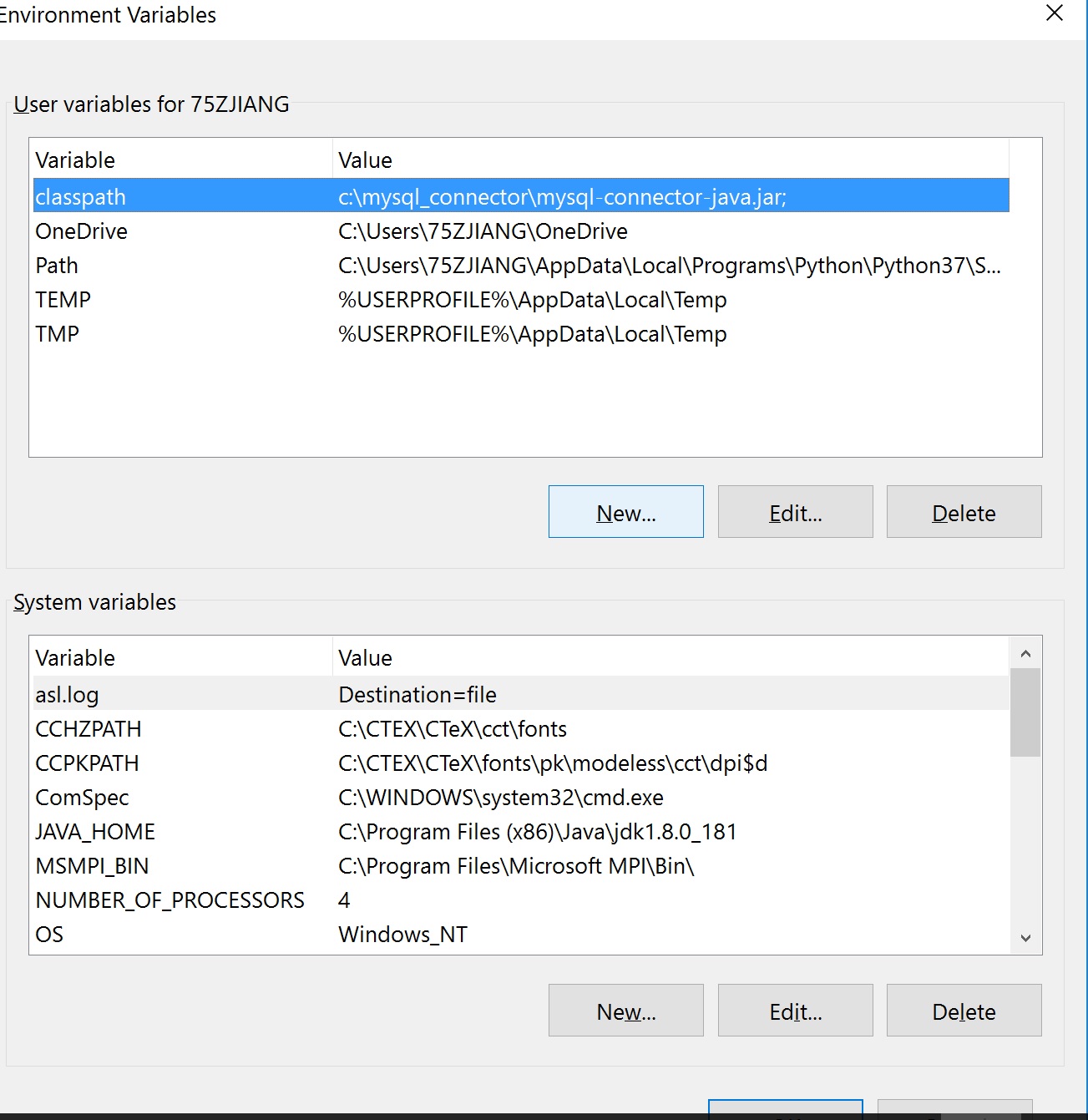
1. Place “mysql-connector-java.jar” to “c:\mysql\_connector”

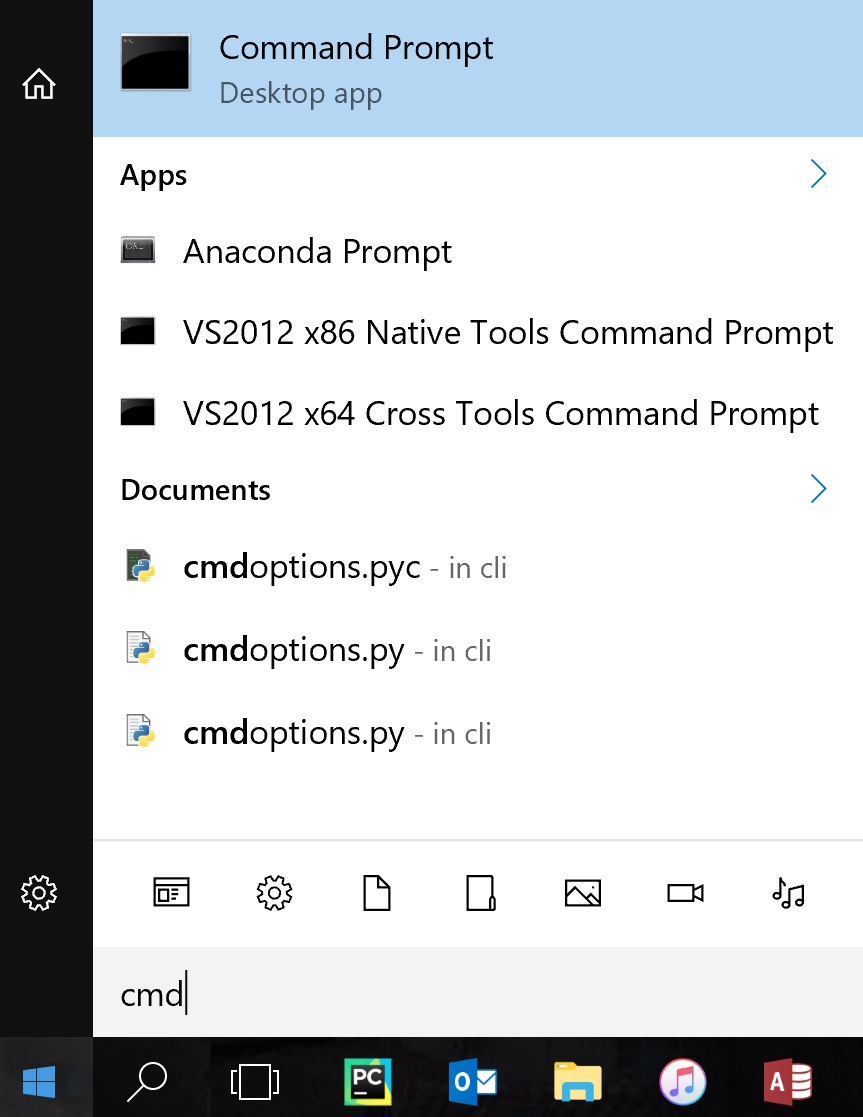


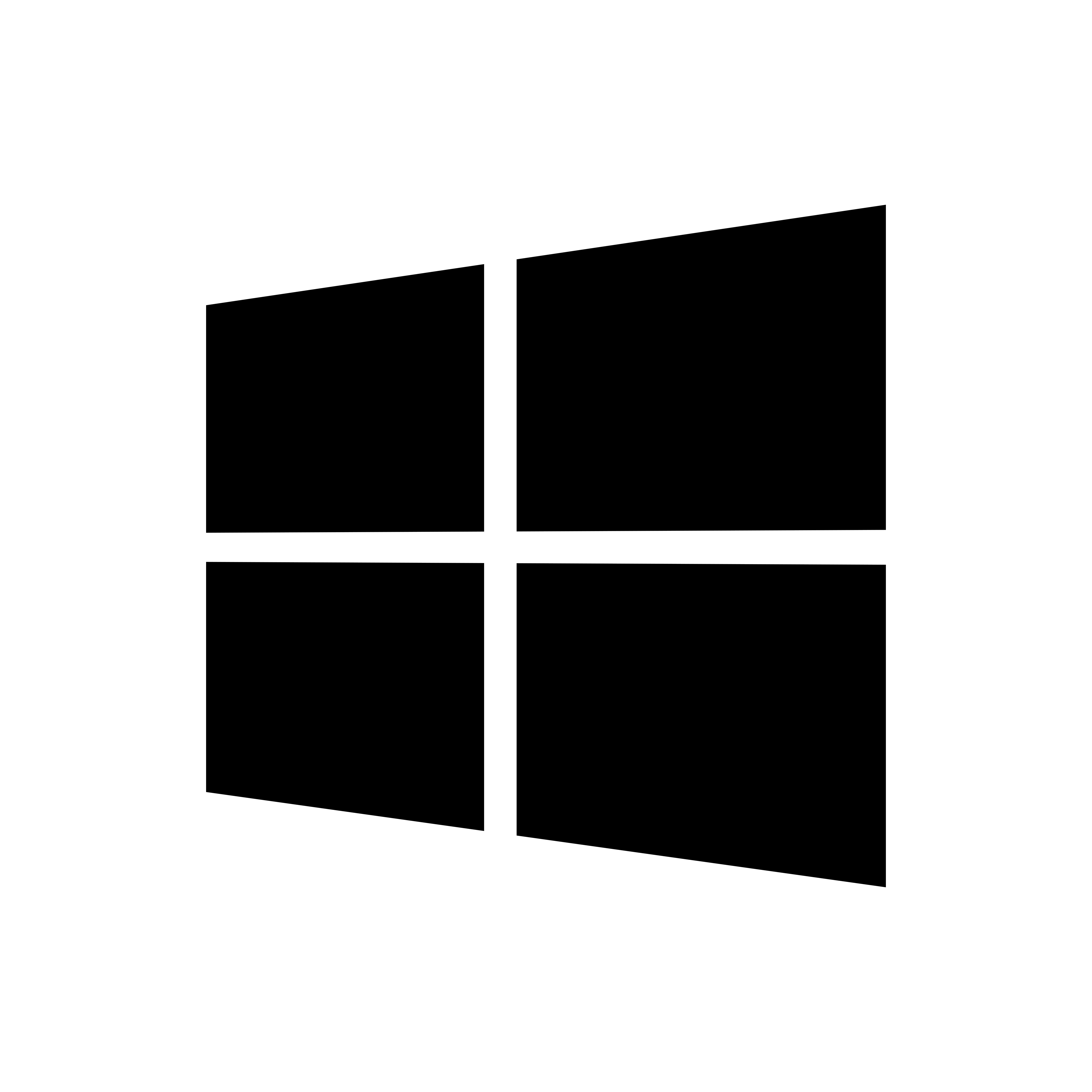
1. Set classpath:
   1. Enter <Advanced system settings> <Environment Variables>



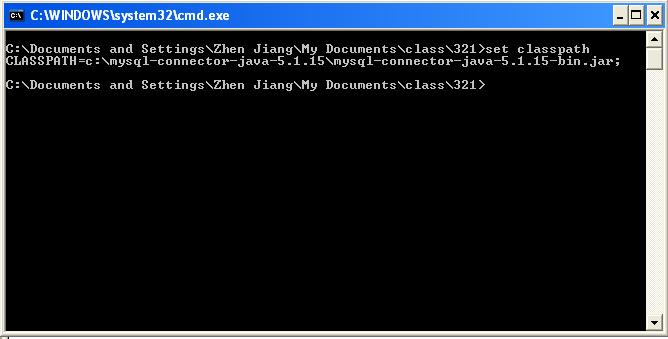
1. Click on <New> and then create “classpath” user variable “c:\mysql\_connector\mysql-connctor-java.jar;”



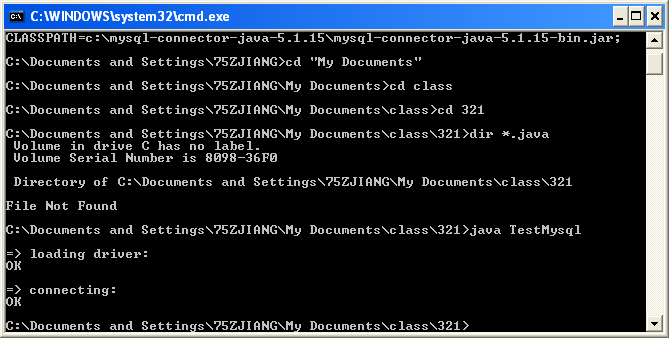




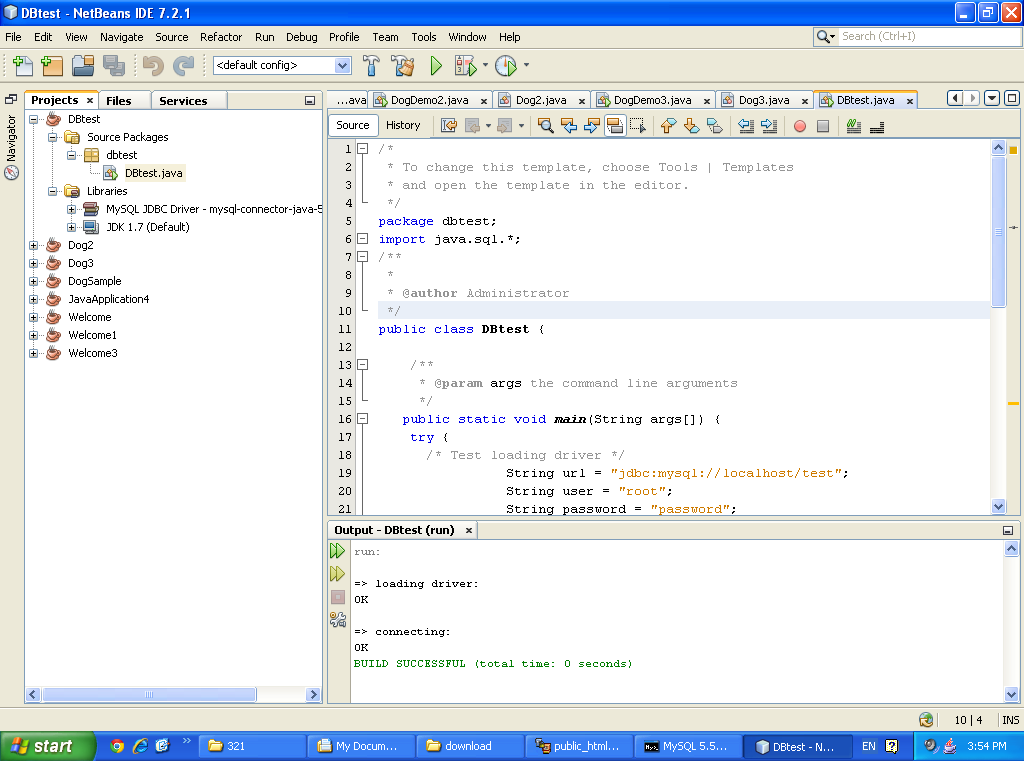
1. Click on <windows> icon on the bottom-left corner. Enter ‘c’+ ‘m’+ ‘d’ and then press <enter> key.
2. Run “set classpath” command and get the result:



1. Use TestMysql.java (available at [www.cs.wcupa.edu/~zjiang/TestMysql.java](http://www.cs.wcupa.edu/~zjiang/TestMysql.java)) to confirm the JDBC connection with the server of MySQL.



or



1. Run Grading.java and verify each UC in the above.

Summary:

1. All the records in the application layer are stored in Database server. The classes in that layer are used to determine the tables:

User (id, name, passwd)

Course\_for\_Grade( Name, Year, Section, tid, did, catogory)

where both tid and did comes out User instance/object.

1. Understand why 3NF is a copy, not a remove.
2. Java classes, including those implementation layer classes being used: System, Scanner, and Trans, are for remote reading/writing activities.
3. See how each activity in UC can be supported in a single task and this task can be triggered by different input. In standard keyboard input, this can be distinguished by a different sequence of input options. In GUI, it is simpler via different GUI button or mouse event. The corresponding task will be implemented in event.listener.
4. Think why there is no need for array of students?
5. What is the use of java class of User, Student, and Instructor?
6. If java accepts association and generalization, will the use of the association of class Course lose the accuracy, compared by the first UML class diagram?
7. What is the relation of tables in UML?