Student Name:- _____ Student ID: _____

CS545: Web Application Architecture
Midterm Exam

Computer Professionals Program

Date: 07 - 29 -2021

Theory		Cognitive skills		
Q1 (6)	Q2 (2)	Q3 (4)	Q4 (6)	Q5 (7)

(United States) Focus

Question 1: Circle the correct answer _____ **(6 points – each 1)**

1) What is the suitable component stereotype annotation to place on classes that will be functioning in the presentation layer?

a) @service
b) @controller
c) @repository
d) @bean

2) Which of the following operations is not idempotent?

a) GET
b) POST
c) PUT
d) DELETE

3) To get the value of id in the URL below:

`http://localhost:8080/products?id=1234`

```
@RequestMapping("/products")
public List<Product> findProductById ( ???????????  ){
      return productService.findProductById(id);
}
```

The correct method parameter would be?

1753 words English (United States) Focus

Question 1: Circle the correct answer _____(6 points – each 1)

1) What is the suitable component stereotype annotation to place on classes that will be functioning in the presentation layer?

a) @service
b) @controller
c) @repository
d) @bean

2) Which of the following operations is not idempotent?

a) GET
b) POST
c) PUT
d) DELETE

3) To get the value of id in the URL below:

`http://localhost:8080/products?id=1234`

```
@RequestMapping("/products")
public List<Product> findProductById ( ???????????  ){
```

a) GET
b) POST
c) PUT
d) DELETE

3) To get the value of id in the URL below:

`http://localhost:8080/products?id=1234`

```
@RequestMapping("/products")
public List<Product> findProductById ( ???????????  ){
      return productService.findProductById(id);
}
```

The correct method parameter would be?

a) @ModelAttribute("id") long id
b) @PathVariable("id") long id
c) @RequestParam("id") long id
d) @RequestBody("id") long id

4) If we place `@RequestMapping(method = RequestMethod.GET)` **over a method, it is equivalent to** `@GetMapping()`

4) If we place `@RequestMapping(method = RequestMethod.GET)` over a method, it is equivalent to `@GetMapping()`

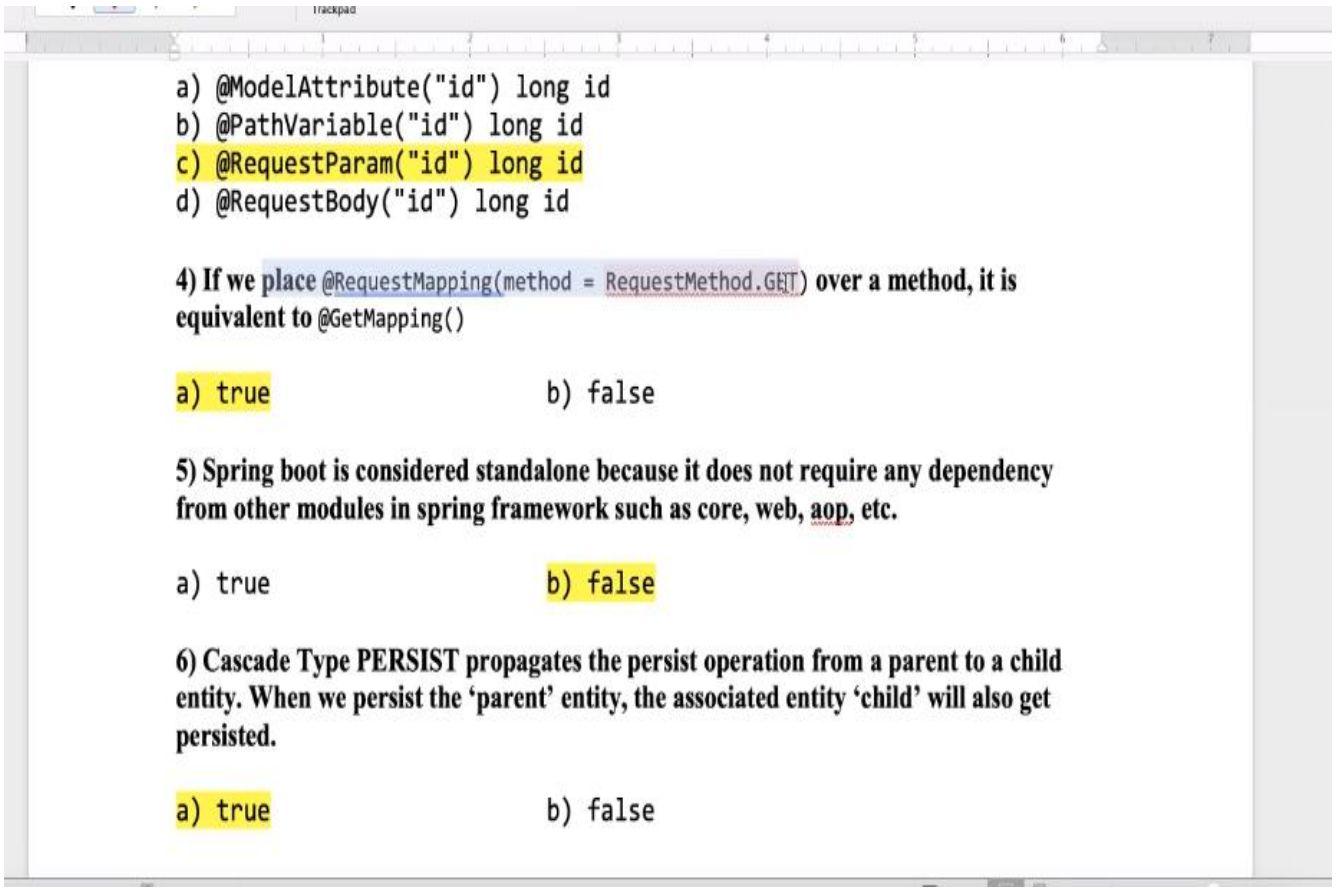
a) true b) false

5) Spring boot is considered standalone because it does not require any dependency from other modules in spring framework such as core, web, aop, etc.

a) true b) false

6) Cascade Type PERSIST propagates the persist operation from a parent to a child entity. When we persist the 'parent' entity, the associated entity 'child' will also get persisted.

a) true b) false



a) @ModelAttribute("id") long id
b) @PathVariable("id") long id
c) @RequestParam("id") long id
d) @RequestBody("id") long id

4) If we place `@RequestMapping(method = RequestMethod.GET)` over a method, it is equivalent to `@GetMapping()`

a) true b) false

5) Spring boot is considered standalone because it does not require any dependency from other modules in spring framework such as core, web, aop, etc.

a) true b) false

6) Cascade Type PERSIST propagates the persist operation from a parent to a child entity. When we persist the 'parent' entity, the associated entity 'child' will also get persisted.

a) true b) false

Question 2: What does `@SpringBootApplication` annotation do internally? (2 points)

The @SpringBootApplication annotation is a combination of following three Spring annotations and provides the functionality of all three with just one line of code.

@Configuration
This annotation marks a class as a Configuration class for Java-based configuration. This is particularly important if you favor Java-based configuration over XML configuration.

@ComponentScan
This annotation enables component-scanning so that the web controller classes and other components you create will be automatically discovered and registered as beans in Spring's Application Context. All the@Controller classes you write are discovered by this annotation.

@EnableAutoConfiguration
This annotation enables the magical auto-configuration feature of Spring Boot, which can automatically configure a lot of stuff for you.

@Configuration
This annotation marks a class as a Configuration class for Java-based configuration. This is particularly important if you favor Java-based configuration over XML configuration.

@ComponentScan
This annotation enables component-scanning so that the web controller classes and other components you create will be automatically discovered and registered as beans in Spring's Application Context. All the@Controller classes you write are discovered by this annotation.

@EnableAutoConfiguration
This annotation enables the magical auto-configuration feature of Spring Boot, which can automatically configure a lot of stuff for you.

Question 3: Using JSR-303 Bean Validation API, annotate the given codes (domain models) on page 4, validate them according to the following requirements: (4 points)

- `id` will be auto-generated.
- `firstName` and `lastName` are both **required** with a **minimum** of 1 letter up to 50

Question 3: Using JSR-303 Bean Validation API, annotate the given codes (domain models) on page 4, validate them according to the following requirements: (4 points)

- `id` will be auto-generated.
- `firstName` and `lastName` are both **required** with a **minimum** of 1 letter up to 50 **maximum**.
- `email` is **required** and should be a valid email.
- `salary` is **required** and should have at maximum 6 **digits** and no decimal points.
- `address.street` is **required**
- `address.state` should be only **2** characters, example → 'IA'
- `address.zipCode` should only have 5 **characters**, example → 52557

- All validation constraints should have a corresponding message taken from the **errorMesssages** file. You may use the existing messages and add based on your requirement.

- The user should NOT see nested labels such as '`address.street`', it should be '`street`' and same thing for all other nested labels.

- Assume that all configurations are successfully adjusted, write your answers only for the given codes on page 4

- You may refer to the Bean validation annotations on page 9.

Employee.java

```
public class Employee {

  private long id;

@NotBlank
@Size(min = 1 , max = 50 , message = "{Size.name.validation}")
  private String firstName;

@NotBlank
@Size(min = 1 , max = 50 , message = "{Size.name.validation }")
  private String lastName;

@NotNull
@Digit(integer = 6 , fraction = 0, message ="exceed maximum
numbers of digits")
  private double salary;

@NotBlank
@email(message = "{email.validation}")
  private String email;

@Valid
```

Address.java

```
public class Address {

@NotBlank
  private String street;

@NotBlank
  private String city;


@NotEmpty
@Size(min = 2 , max = 2 , message = "{Size.state}")
  private String state;


@NotEmpty
@Size(min = 5 , max = 5 , message = "{Size.zipcode}")
  private String zipcode;
```

Employee.java

```
public class Employee {

   private long id;

@NotBlank
@Size(min = 1, max = 50 , message = "{Size.name.validation}")
   private String firstName;

@NotBlank
@Size(min = 1 , max = 50 , message = "{Size.name.validation }")
   private String lastName;

@NotNull
@Digit(integer = 6 , fraction = 0, message ="exceed maximum
numbers of digits")
   private double salary;

@NotBlank
@email(message = "{email.validation}")
   private String email;

@Valid
   private Address address;
}
```

Address.java

```
public class Address {

@NotBlank
   private String street;

@NotBlank
   private String city;


@NotEmpty
@Size(min = 2 , max = 2 , message = "{Size.state}")
   private String state;


@NotEmpty
@Size(min = 5 , max = 5 , message = "{Size.zipcode}")
   private String zipcode;




}
```


```
   private String email;

@Valid
   private Address address;
}
```

```
}
```


errorMessages.properties

```
NotNull = {0} is a required field
NotEmpty = {0} field must have a value
NotBlank = {0} field must not be blank

Size.state = State must have two characters
Size.name.validation = Size of the {0} must be between {2} and {1}
Size.zipcode = Zipcode must have five characters


email.validation = The entered email is not a valid email.


firstName = First Name
lastName = Last Name


address.street = street
address.state = state
```

Question 4: Write the code to complete the EmployeeController. This controller should implement the following calls: **(6 marks)**

1- GET localhost:8080/employees → This should retrieve all the employees in the database.
2- GET localhost:8080/employees/1 → This should retrieve the employee with id = 1
3- POST localhost:8080/employees → This should create and save a new employee
4- PUT localhost:8080/employees/1 → This should update the employee with id = 1
5- DELETE localhost:8080/employees/1 → This should delete the employee with id = 1
6- GET localhost:8080/employees/1/phones → This should retrieve the phones of the employee with id = 1 assuming there is an associated domain class '**Phone**'

Note: 1 is just an example, the actual path should ask for 'id'

Assuming that **employeeSerivce** has the following methods:

```
employeeSerivce.findEmployee( long id )                    // This will retrieve an 'Employee' object on matching 'id'

employeeSerivce.findAll( )                                 // This will retrieve a List of all 'Employee' objects

employeeSerivce.save( Employee employee )                  // This will add an 'Employee' to the database

employeeService.update (long id, Employee employee)        // This will update an 'Employee' on the database with the
                                                           matching 'id'

employeeService.deleteById (long id)                       // This will delete an 'Employee' on the database with the
                                                           matching 'id'

employeeService.getPhones( long id )                       // This will retrieve a List of phones for an employee
```

4- PUT localhost:8080/employees/1 → This should update the employee with id = 1

5- DELETE localhost:8080/employees/1 → This should delete the employee with id = 1

6- GET localhost:8080/employees/1/phones → This should retrieve the phones of the employee with id = 1
assuming there is an associated domain class '**Phone**'

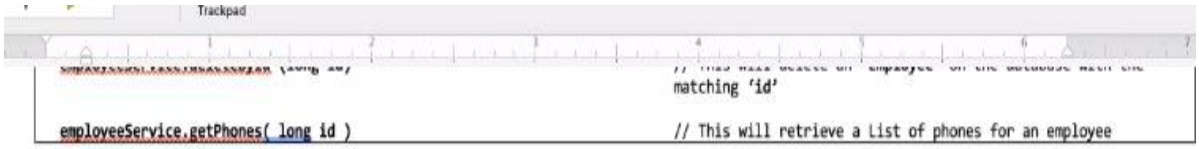
Note: 1 is just an example, the actual path should ask for 'id'

Assuming that **employeeSerivce** has the following methods:

```
employeeSerivce.findEmployee( long id )                    // This will retrieve an 'Employee' object on matching 'id'

employeeSerivce.findAll( )                                 // This will retrieve a List of all 'Employee' objects

employeeSerivce.save( Employee employee )                  // This will add an 'Employee' to the database

employeeService.update (long id, Employee employee)        // This will update an 'Employee' on the database with the
                                                           matching 'id'

employeeService.deleteById (long id)                       // This will delete an 'Employee' on the database with the
                                                           matching 'id'

employeeService.getPhones( long id )                       // This will retrieve a List of phones for an employee
```

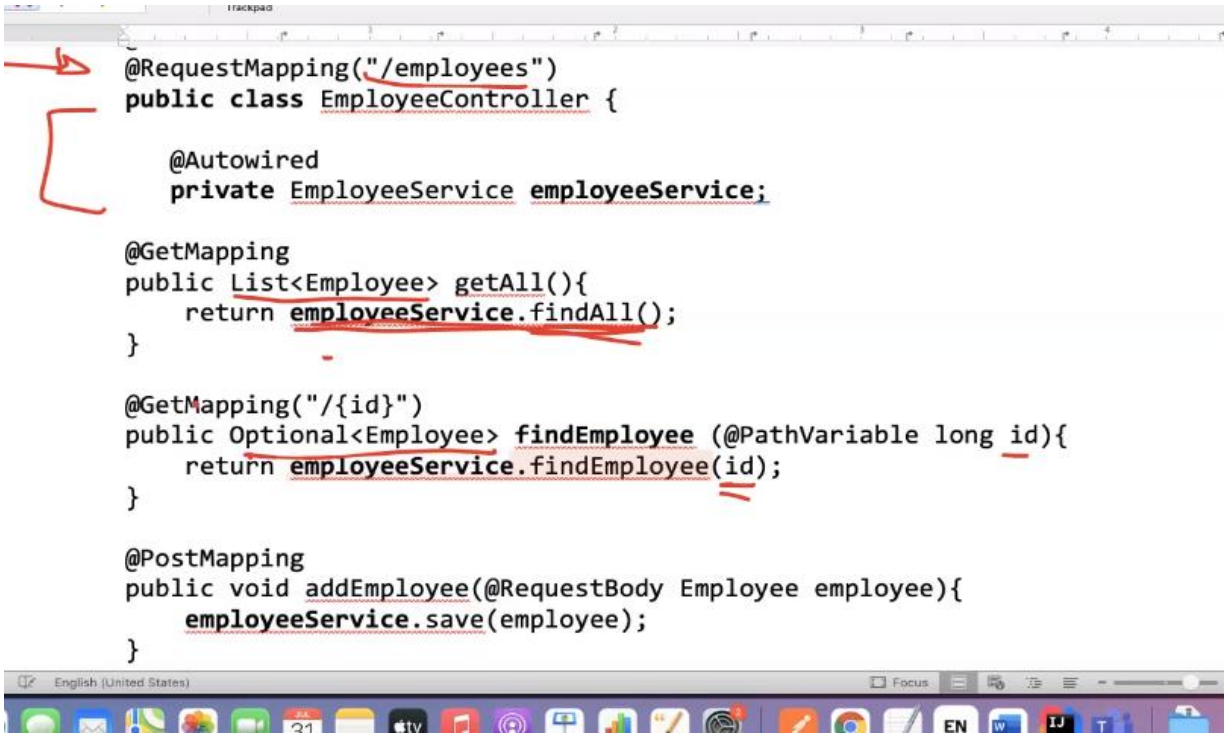
EmployeeController

```
package edu.miu.controller;
import …

@RestController
@RequestMapping("/employees")
public class EmployeeController {

   @Autowired
   private EmployeeService employeeService;
```

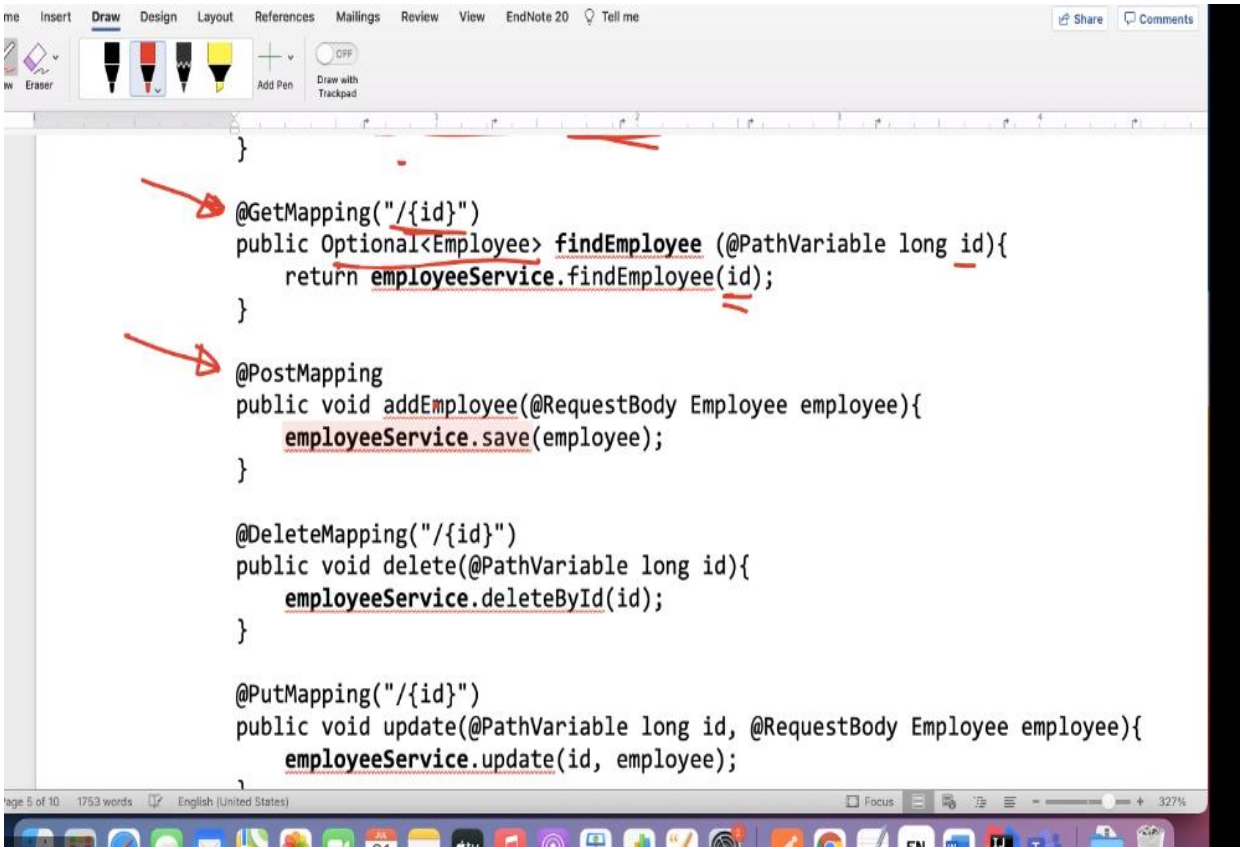


EmployeeController

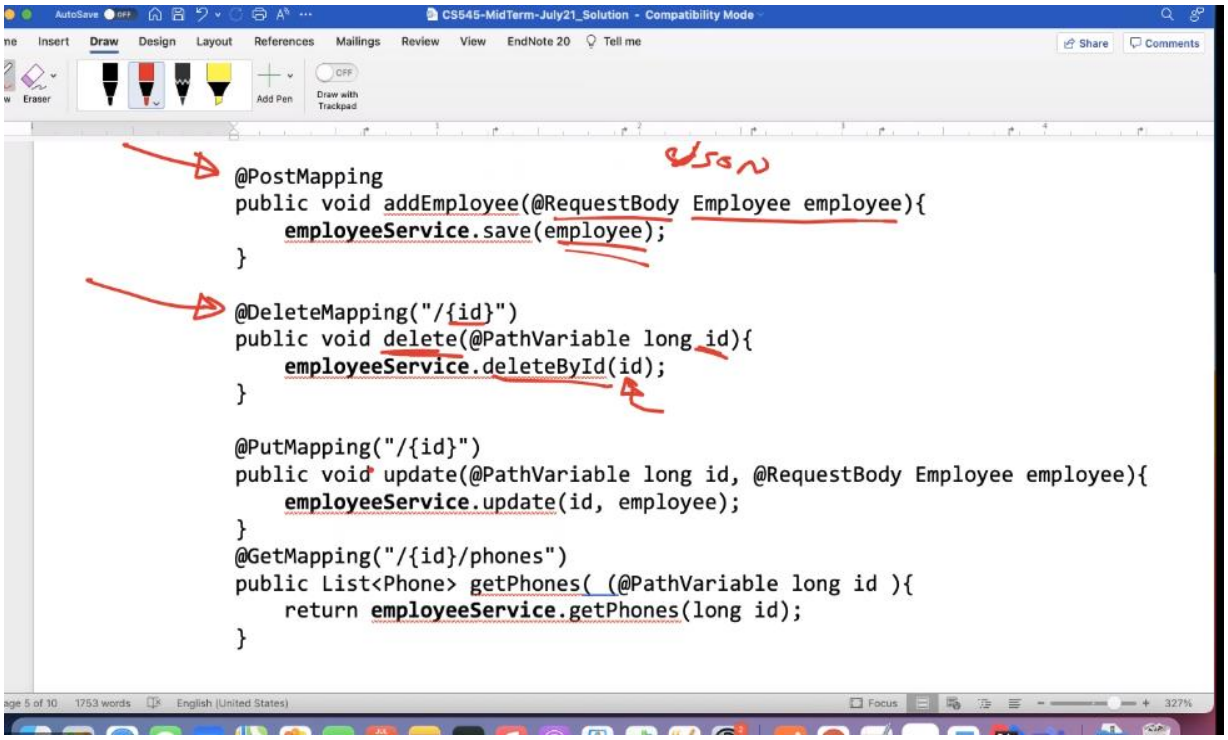
```
package edu.miu.controller;
import …

@RestController
@RequestMapping("/employees")
public class EmployeeController {

   @Autowired
   private EmployeeService employeeService;

@GetMapping
public List<Employee> getAll(){
    return employeeService.findAll();
}

@GetMapping("/{id}")
public Optional<Employee> findEmployee (@PathVariable long id){
    return employeeService.findEmployee(id);
}

@PostMapping
public void addEmployee(@RequestBody Employee employee){
    employeeService.save(employee);
}

@DeleteMapping("/{id}")
```



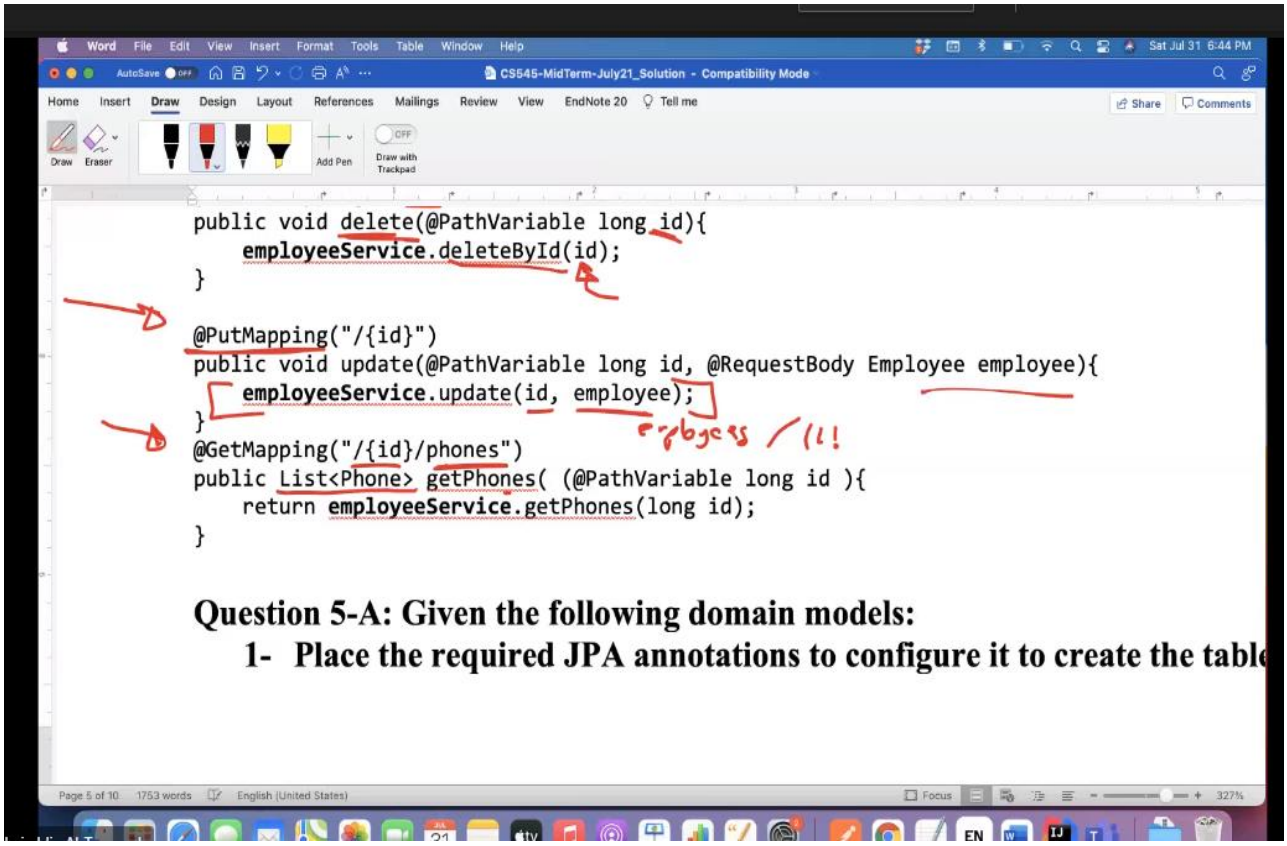
```java
@RequestMapping("/employees")
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

@GetMapping
public List<Employee> getAll(){
    return employeeService.findAll();
}

@GetMapping("/{id}")
public Optional<Employee> findEmployee (@PathVariable long id){
    return employeeService.findEmployee(id);
}

@PostMapping
public void addEmployee(@RequestBody Employee employee){
    employeeService.save(employee);
}
```



```java
}

@GetMapping("/{id}")
public Optional<Employee> findEmployee (@PathVariable long id){
    return employeeService.findEmployee(id);
}

@PostMapping
public void addEmployee(@RequestBody Employee employee){
    employeeService.save(employee);
}

@DeleteMapping("/{id}")
public void delete(@PathVariable long id){
    employeeService.deleteById(id);
}

@PutMapping("/{id}")
public void update(@PathVariable long id, @RequestBody Employee employee){
    employeeService.update(id, employee);
}
```



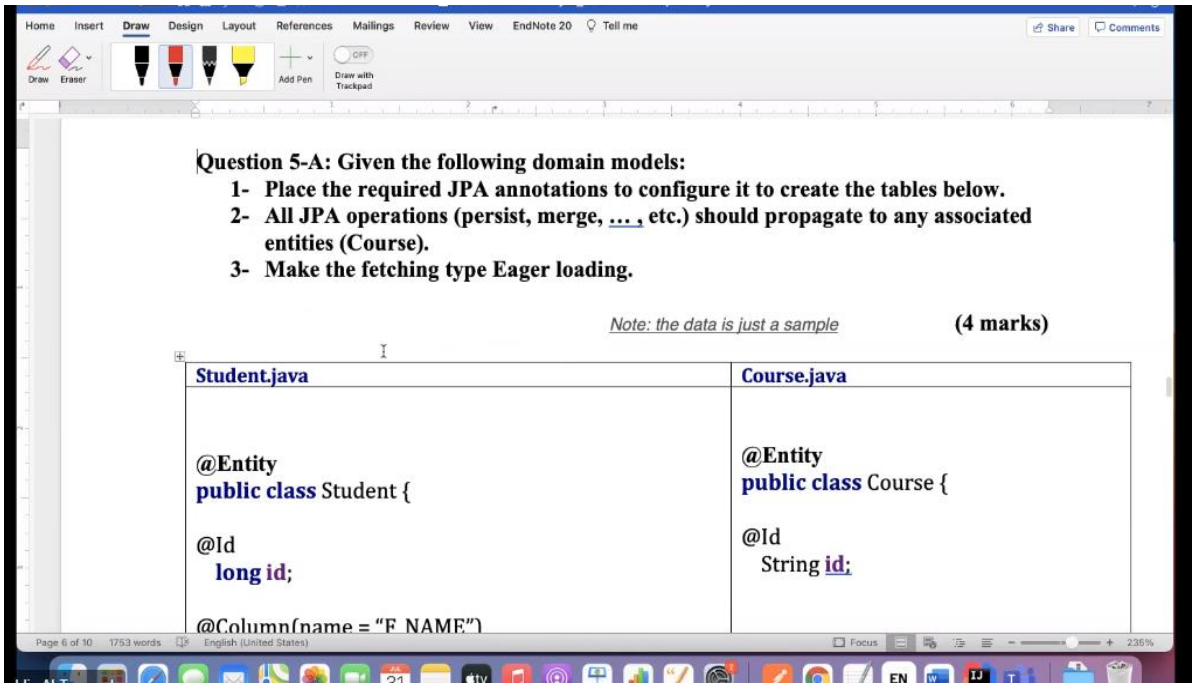
```
@PostMapping
public void addEmployee(@RequestBody Employee employee){
    employeeService.save(employee);
}

@DeleteMapping("/{id}")
public void delete(@PathVariable long id){
    employeeService.deleteById(id);
}

@PutMapping("/{id}")
public void update(@PathVariable long id, @RequestBody Employee employee){
    employeeService.update(id, employee);
}
@GetMapping("/{id}/phones")
public List<Phone> getPhones( (@PathVariable long id ){
    return employeeService.getPhones(long id);
}
```



```
public void delete(@PathVariable long id){
    employeeService.deleteById(id);
}

@PutMapping("/{id}")
public void update(@PathVariable long id, @RequestBody Employee employee){
    employeeService.update(id, employee);
}
@GetMapping("/{id}/phones")
public List<Phone> getPhones( (@PathVariable long id ){
    return employeeService.getPhones(long id);
}
```

Question 5-A: Given the following domain models:

1- Place the required JPA annotations to configure it to create the table

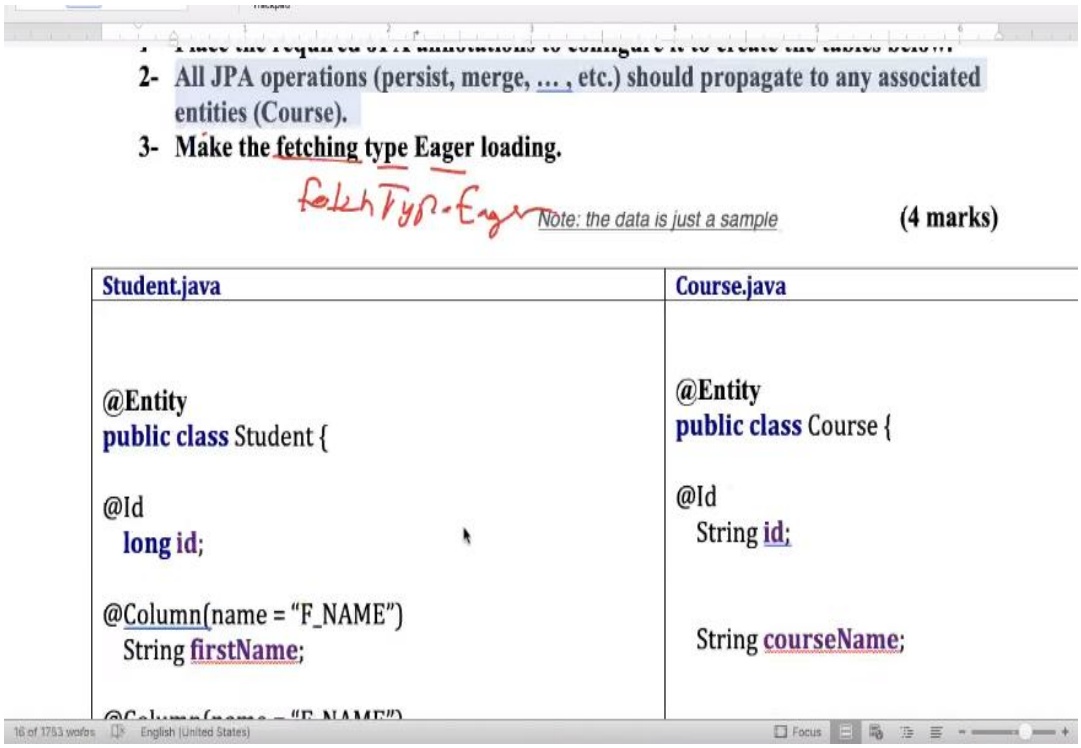


Question 5-A: Given the following domain models:

1- **Place the required JPA annotations to configure it to create the tables below.**
2- **All JPA operations (persist, merge, … , etc.) should propagate to any associated entities (Course).**
3- **Make the fetching type Eager loading.**

Note: the data is just a sample **(4 marks)**

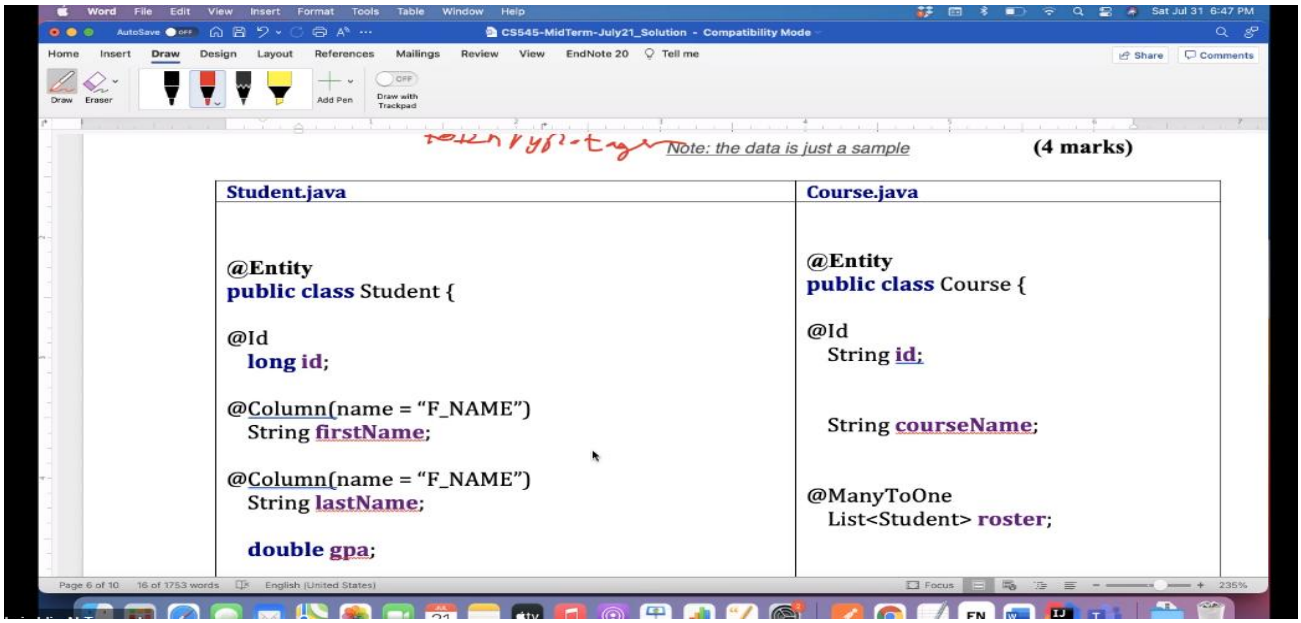
Student.java	Course.java
@Entity public class Student { @Id long id; @Column(name = "F_NAME")	@Entity public class Course { @Id String id;



2- All JPA operations (persist, merge, … , etc.) should propagate to any associated entities (Course).
3- Make the fetching type Eager loading.

fetchType-Eager *Note: the data is just a sample* **(4 marks)**

Student.java	Course.java
@Entity public class Student { @Id long id; @Column(name = "F_NAME") String firstName;	@Entity public class Course { @Id String id; String courseName;



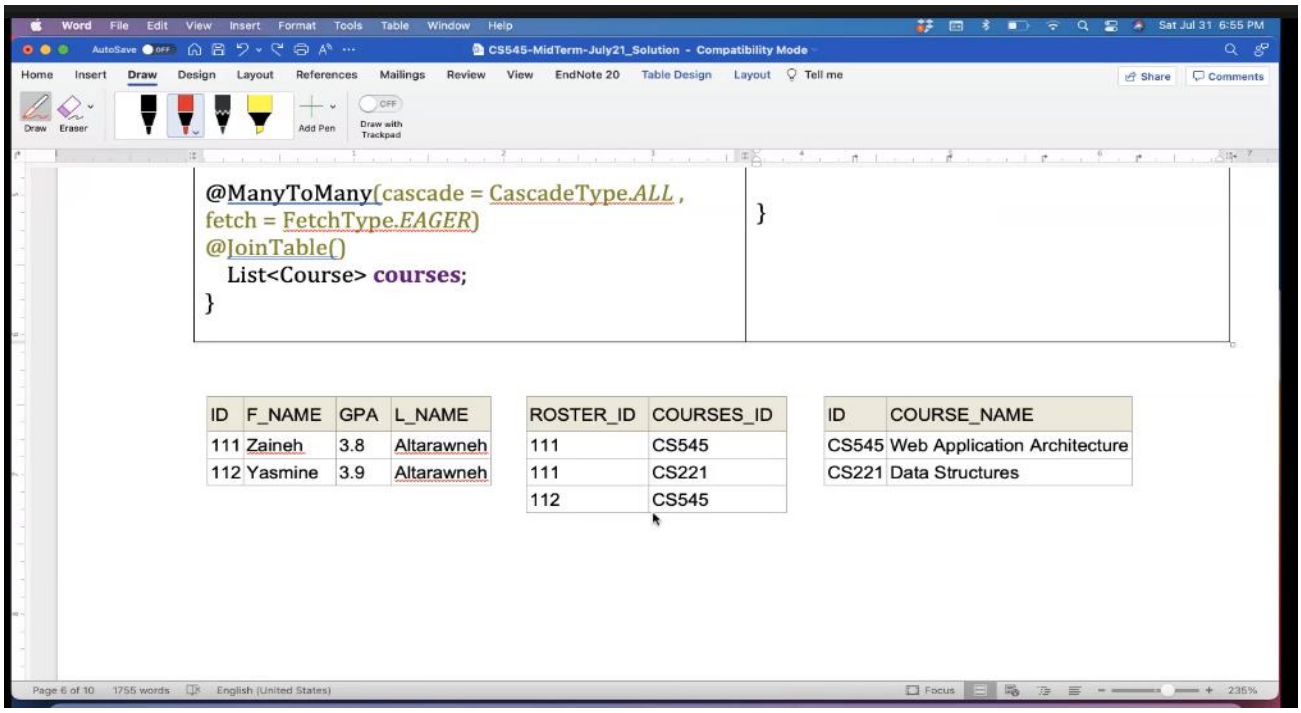
Note: the data is just a sample **(4 marks)**

Student.java

```
@Entity
public class Student {

@Id
   long id;

@Column(name = "F_NAME")
   String firstName;

@Column(name = "F_NAME")
   String lastName;

   double gpa;

@ManyToMany(cascade = CascadeType.ALL ,
fetch = FetchType.EAGER)
@JoinTable()
   List<Course> courses;
}
```

Course.java

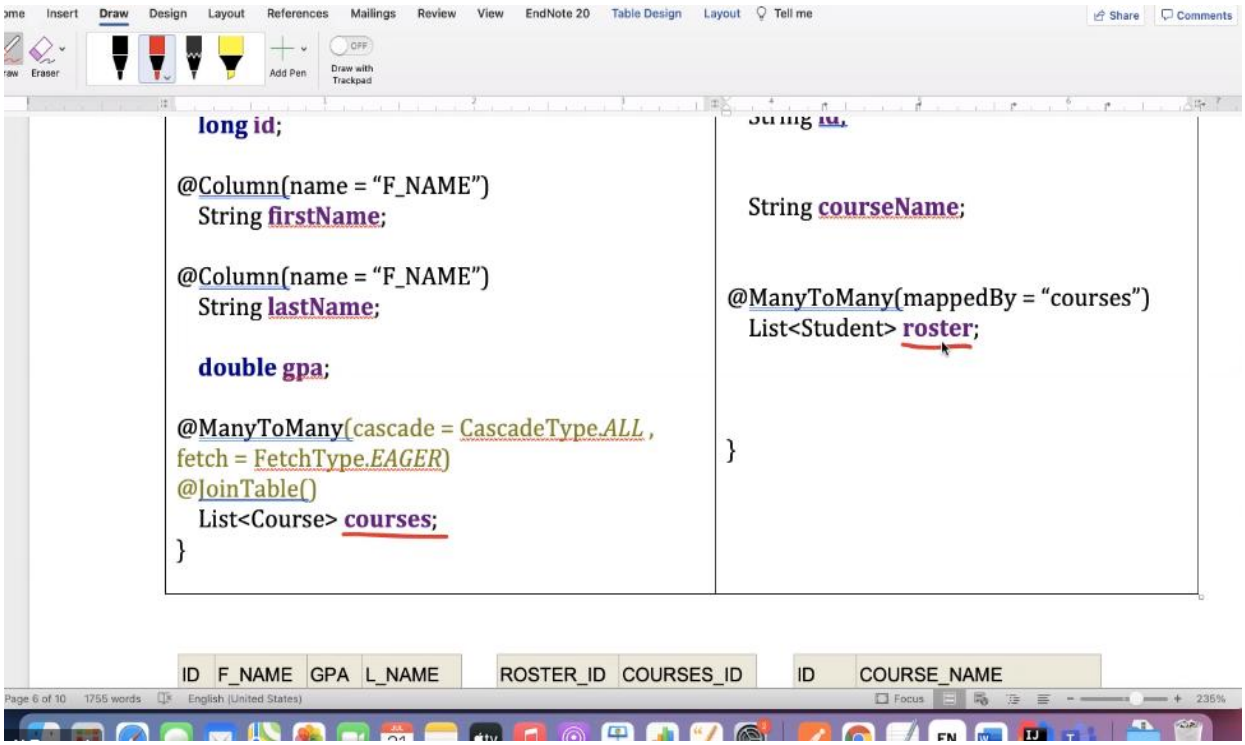
```
@Entity
public class Course {

@Id
   String id;

   String courseName;

@ManyToOne
   List<Student> roster;
}
```

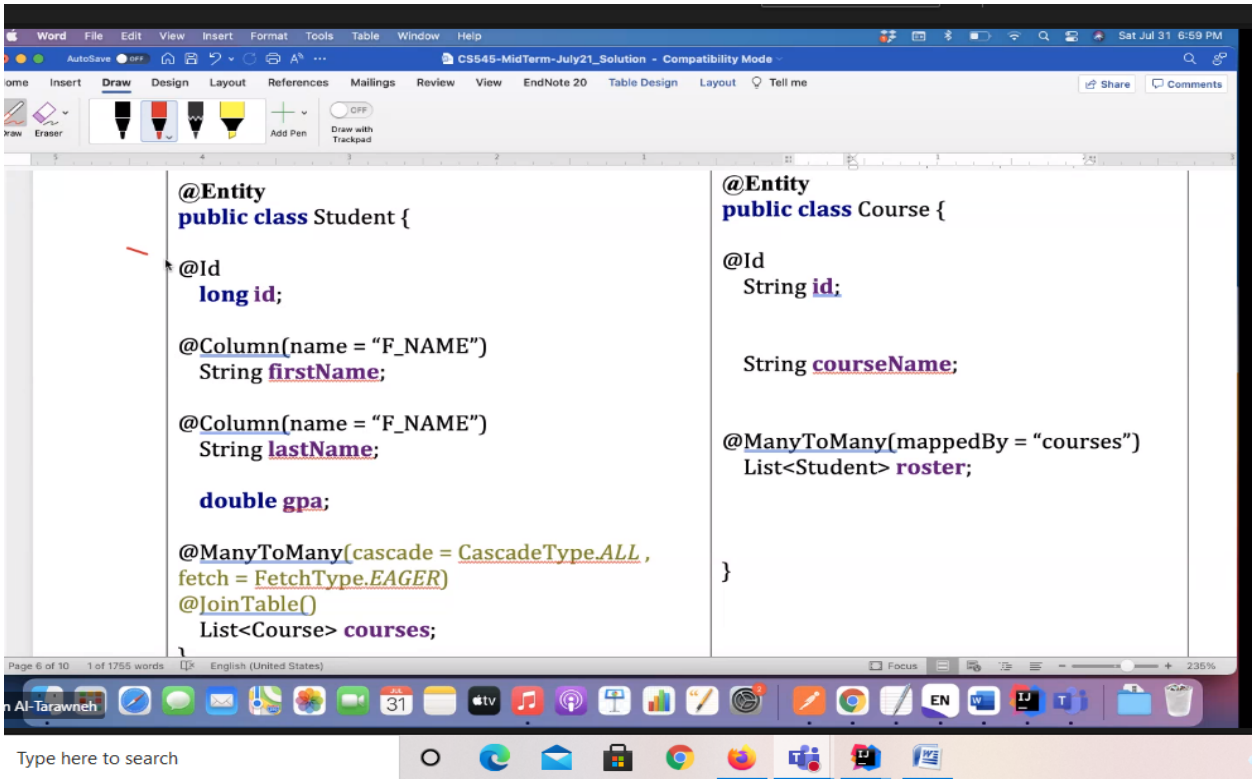


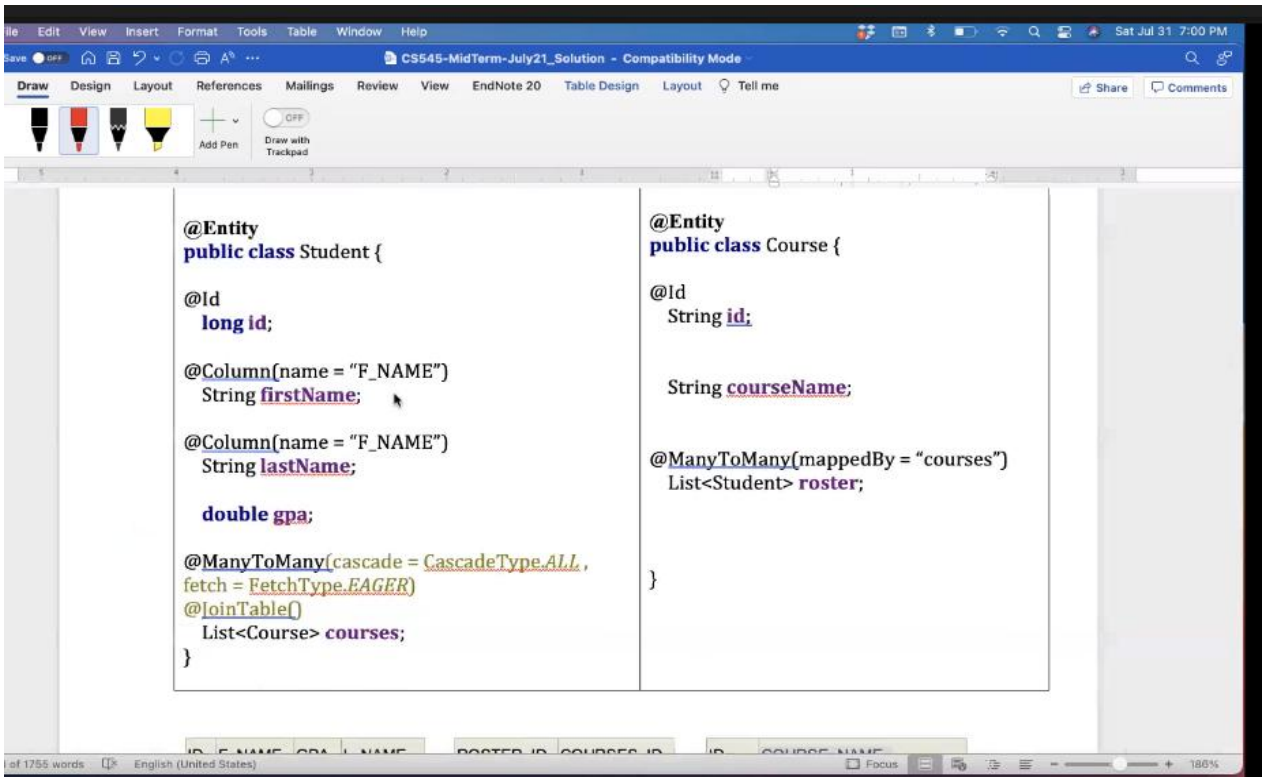
ID	F_NAME	GPA	L_NAME
111	Zaineh	3.8	Altarawneh
112	Yasmine	3.9	Altarawneh

ROSTER_ID	COURSES_ID
111	CS545
111	CS221
112	CS545

ID	COURSE_NAME
CS545	Web Application Architecture
CS221	Data Structures

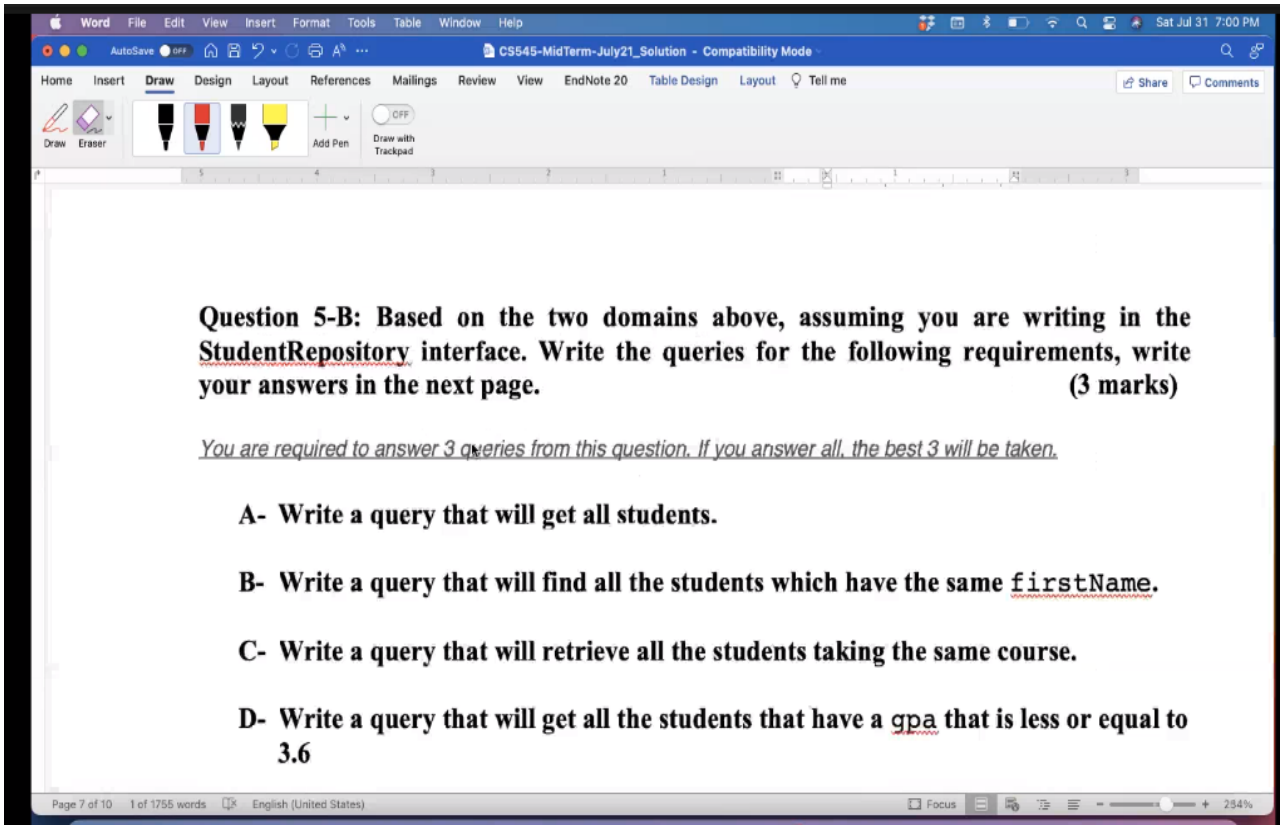

long id;
@Column(name = "F_NAME")
String firstName;
@Column(name = "F_NAME")
String lastName;
double gpa;
@ManyToMany(cascade = CascadeType.ALL ,
fetch = FetchType.EAGER)
@JoinTable()
List<Course> courses;
}
String courseName;
@ManyToMany(mappedBy = "courses")
List<Student> roster;
}
ID F_NAME GPA L_NAME
ROSTER_ID COURSES_ID
ID COURSE_NAME


@Entity
public class Student {
@Id
long id;
@Column(name = "F_NAME")
String firstName;
@Column(name = "F_NAME")
String lastName;
double gpa;
@ManyToMany(cascade = CascadeType.ALL ,
fetch = FetchType.EAGER)
@JoinTable()
List<Course> courses;
@Entity
public class Course {
@Id
String id;
String courseName;
@ManyToMany(mappedBy = "courses")
List<Student> roster;
}
Type here to search



```java
@Entity
public class Student {

@Id
  long id;

@Column(name = "F_NAME")
  String firstName;

@Column(name = "F_NAME")
  String lastName;

  double gpa;

@ManyToMany(cascade = CascadeType.ALL ,
fetch = FetchType.EAGER)
@JoinTable()
  List<Course> courses;
}
```

```java
@Entity
public class Course {

@Id
  String id;


  String courseName;


@ManyToMany(mappedBy = "courses")
  List<Student> roster;


}
```



Question 5-B: Based on the two domains above, assuming you are writing in the StudentRepository interface. Write the queries for the following requirements, write your answers in the next page. (3 marks)

You are required to answer 3 queries from this question. If you answer all, the best 3 will be taken.

A- Write a query that will get all students.

B- Write a query that will find all the students which have the same `firstName`.

C- Write a query that will retrieve all the students taking the same course.

D- Write a query that will get all the students that have a `gpa` that is less or equal to 3.6

C- Write a query that will retrieve all the students taking the same course.

D- Write a query that will get all the students that have a gpa that is less or equal to 3.6

StudentRepository

```
public interface StudentRepository extends CrudRepository<Student, Long>{

public List<Student> findAll();

// @Query("select  s  from Student s  where s.firstName = :firstName")
public List<Student> findAllByFisrtName(@Param("firstName") String firstName);

@Query("select c.roster from Course c where c.id = :id")
```


your answers in the next page. (3 marks)

You are required to answer 3 queries from this question. If you answer all, the best 3 will be taken.

A- Write a query that will get all students.

B- Write a query that will find all the students which have the same firstName.

C- Write a query that will retrieve all the students taking the same course.

D- Write a query that will get all the students that have a gpa that is less or equal to 3.6

StudentRepository

```
public interface StudentRepository extends CrudRepository<Student, Long>{
```

Page 7 of 10 1756 words English (United States) Focus 251%