Name: _____

Student Id: _____

1	
2	
3	
4	
5	
6	
7	
8	
Total	

CS544 Midterm exam

Question 1 [10 points] {10 minutes}

a. Suppose we have a Spring application with the following given XML configuration

```
<bean id="customerService" class="basic.CustomerService">
   <constructor-arg ref="emailService"/>
</bean>
<bean id="emailService" class="basic.EmailService">
   <constructor-arg ref="customerService"/>
</bean>
```

When we run the application, Spring gives an error. Explain clearly why Spring gives an error based on the given XML configuration.

With constructor injection Spring first need to initialize the bean that is injected. Then it can initialize the bean by using the constructor of this bean and pass the injected bean as parameter. In the given configuration we have a circular dependency. Spring can only create the CustomerService bean after it has created the Emailservice. But it can only create the EmailService after it has created the CustomerService.

b. Explain how we can solve this problem. **Write the XML configuration** that solves the problem and gives the same wiring of beans given in part a. It is important that your XML configuration results in the same spring beans and corresponding bean wiring as the XML configuration given in part a. Assume the corresponding Spring beans have all the necessary methods.

Use setter injection.

```
<bean id="customerService" class="basic.CustomerService">
   <property name="emailService" ref="emailService"/>
</bean>
<bean id="emailService" class="basic.EmailService">
   <property name="customerService"  ref="customerService"/>
</bean>
Or at least 1 bean should have setter injection
<bean id="customerService" class="basic.CustomerService">
   <property name="emailService" ref="emailService"/>
</bean>
<bean id="emailService" class="basic.EmailService">
   <constructor-arg ref="customerService"/>
</bean>
```


If your answer does not fit, continue at the back of this page

Question 2 [10 points] {10 minutes}

Circle all statements that are correct:

a. With the dirty read problem you can read uncommitted data.
b. With the phantom read problem you can read uncommitted data.
c. We map 1 class on 2 tables using @Embedded and @Embeddable
d. When we add a version to an entity and we annotate this with @Version then you will never have the dirty read problem on this entity.
e. All Spring beans that are singletons should be thread-safe and should be stateless.
f. In JPA, a @ManyToMany is always bi-directional.
g. If we do not allow the phantom read problem in our application, we cannot run 2 transactions at the same time.
h. When you use @AfterReturning on an AOP aspect method, you cannot get the return value in this method.
i. In a Spring boot application that uses JPA, you cannot use dependency injection on JPA entities.
j. In JPA, if you do not use @JoinColumn or @JoinTable on a OneToMany relationship, then this OneToMany relationship is by default mapped with a join column.

Question 3 [15 points] {20 minutes}

Given are the following entities:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
      name="vehicle_type",
      discriminatorType=DiscriminatorType.STRING
)
public abstract class Vehicle {
   @Id
   @GeneratedValue
   private long id;
   private String brand;
   private String color;

   public Vehicle() { }

   public Vehicle(String brand, String color) {
      this.brand = brand;
      this.color = color;
   }
}
```


If your answer does not fit, continue at the back of this page

```java
@Entity
public abstract class Car extends Vehicle{
   private String licencePlate;
   public Car() { }
   public Car(String brand, String color, String licencePlate) {
      super(brand, color);
      this.licencePlate = licencePlate;
   }
}

@Entity
@DiscriminatorValue("RentalBycicle")
public class RentalBycicle extends Vehicle{
   private double pricePerHour;
   public RentalBycicle() {   }
   public RentalBycicle(String brand, String color, double pricePerHour) {
      super(brand, color);
      this.pricePerHour = pricePerHour;
   }
}


@Entity
@DiscriminatorValue("SellableCar")
public class SellableCar extends Car {
   private double sellPrice;
   public SellableCar() { }
   public SellableCar(String brand, String color, String licencePlate, double
sellPrice) {
      super(brand, color, licencePlate);
      this.sellPrice = sellPrice;
   }
}


@Entity
@DiscriminatorValue("RentalCar")
public class RentalCar extends Car {
   private double pricePerDay;
   public RentalCar() {   }
   public RentalCar(String brand, String color, String licencePlate, double
pricePerDay) {
      super(brand, color, licencePlate);
      this.pricePerDay = pricePerDay;
   }
}

public interface RentalBycicleRepository extends JpaRepository<RentalBycicle,
Long> {
}
public interface RentalCarRepository extends JpaRepository<RentalCar, Long> {
}
public interface SellableCarRepository extends JpaRepository<SellableCar,
Long> {
}
```


If your answer does not fit, continue at the back of this page

```
@SpringBootApplication
public class Application implements CommandLineRunner {
   @Autowired
   RentalCarRepository rentalCarRepository;
   @Autowired
   SellableCarRepository sellableCarRepository;
   @Autowired
   RentalBycicleRepository rentalBycicleRepository;

   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }

   @Override
   public void run(String... args) throws Exception {
      RentalCar rentalCar = new RentalCar("BMW", "Black", "KL-980-1", 67.00);
      rentalCarRepository.save(rentalCar);
      SellableCar sellableCar = new SellableCar("Audi", "White", "KM-956-2",
45980.00);
      sellableCarRepository.save(sellableCar);
      RentalBycicle rentalBycicle = new RentalBycicle("Moof", "Grey", 10.50);
      rentalBycicleRepository.save(rentalBycicle);

   }

}
```

a. In the given code above, add the necessary mapping annotations so that the whole inheritance hierarchy is mapped according the **single table per hierarchy** strategy. Do **NOT** rewrite any code. Only write the correct annotations in the given code.

b.

c. Explain **ALL** advantages and disadvantages we learned about the **single table per hierarchy** strategy.

Advantages

+ Simple, Easy to implement
+ Good performance on all queries, polymorphic and non-polymorphic

Disadvantages

- Nullable columns / de-normalized schema
- Table may have to contain lots of columns
- A change in any class results in a change of this table

If your answer does not fit, continue at the back of this page

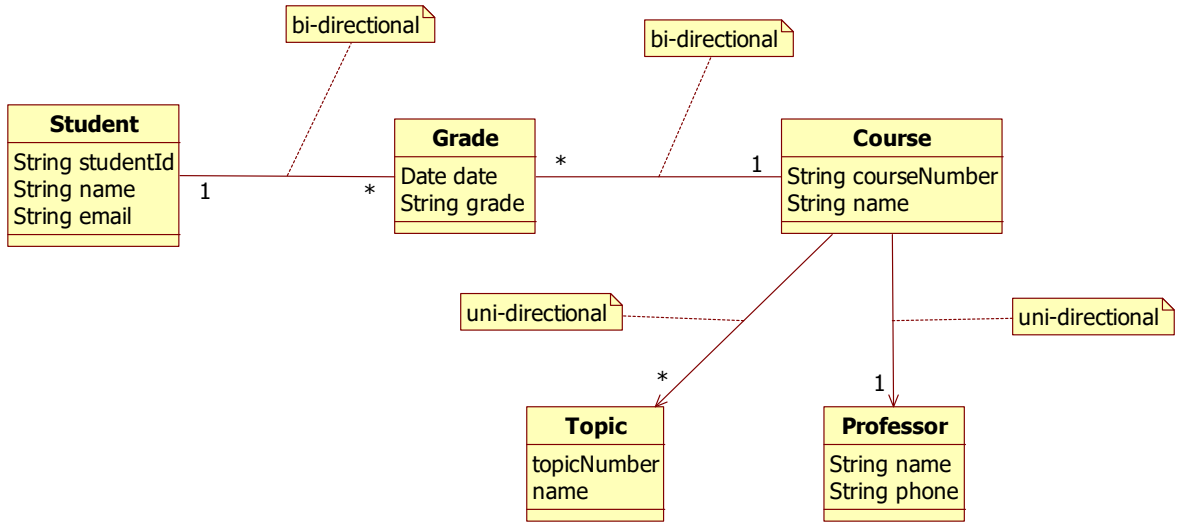
d. Draw the corresponding database table with all the columns and corresponding data if we run Application.java.

VEHICLE_TYPE	ID	BRAND	COLOR	LICENCE_PLATE	PRICE_PER_HOUR	PRICE_PER_DAY	SELL_PRICE
RentalCar	1	BMW	Black	KL-980-1		67	
SellableCar	2	Audi	White	KM-956-2			45980.0
RentalBycicle	3	Moof	Grey	[null]	10.5		

If your answer does not fit, continue at the back of this page

Question 4 [20 points] {25 minutes}

Given is the following domain model:



This domain model has the following requirements:

- The Course has a **Map** of Topics. The topicNumber is the key in the Map.
- The Course has a **List** of Grades. The grades in this List should always be ordered by grade descending.

a. Below you find the corresponding Java classes. The constructor or any getter or setter methods are not shown. In the given code, add the necessary JPA annotations so that we can save these classes to the database. Do **NOT** implement fetching or cascading configuration.

```
@Entity
public class Topic {

    @Id
    @GeneratedValue
    private Long topicNumber;
    private String name;

@Entity
public class Professor {

    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private String phone;
```


If your answer does not fit, continue at the back of this page

```java
@Entity
public class Student {

    @Id
    private Long studentId;
    private String name;
    private String email;

    @OneToMany(mappedBy = "student")
    private List<Grade> grades= new ArrayList<>();


@Entity
public class Grade {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    @JoinColumn(name="grade_course")
    private Course course;

    @ManyToOne
    @JoinColumn(name="grade_student")
    private Student student;

    private Date date;
    private String grade;


@Entity
public class Course {

    @Id
    @GeneratedValue
    private Long courseNumber;

    private String name;

    @OneToMany(mappedBy = "course")
    @OrderBy("grade DESC")
    private List<Grade> grades = new ArrayList<Grade>();

    @ManyToOne
    private Professor professor;

    @OneToMany
    @MapKey(name="topicNumber")
    private Map<Long, Topic> topics = new HashMap<>();
```


If your answer does not fit, continue at the back of this page

b. Draw the tables of the database including their columns according your JPA mapping of part a. You do not need to show the data in the tables. Only show the tables and their columns.

Topic table:

TOPIC_NUMBER	NAME

Professor table:

ID	NAME	PHONE

Student table:

STUDENT_ID	EMAIL	NAME

Course table:

COURSE_NUMBER	NAME	PROFESSOR_ID

CourseTopic table:

COURSE_COURSE_NUMBER	TOPICS_TOPIC_NUMBER

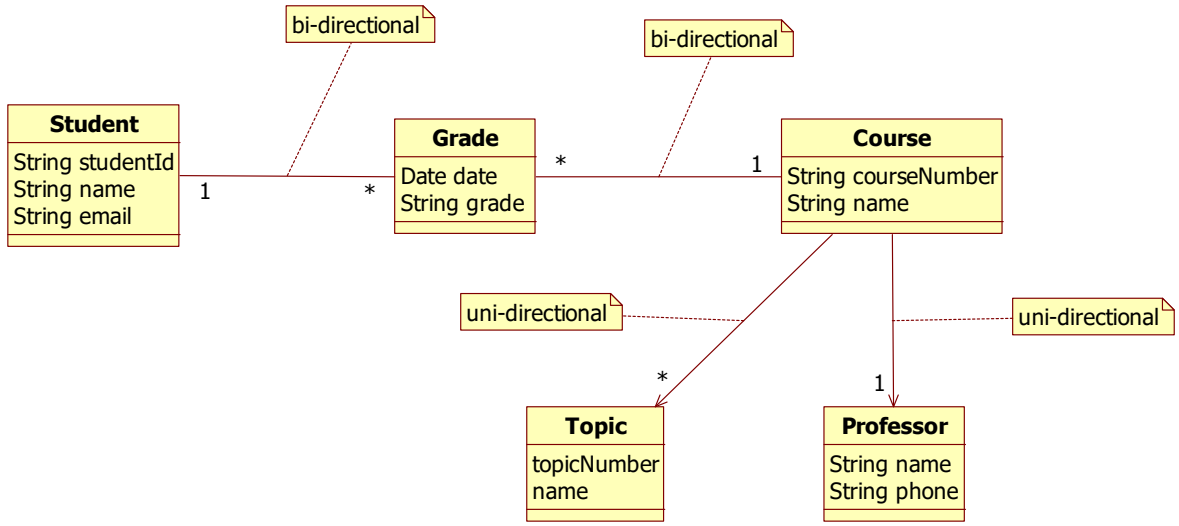
Grade table:

ID	DATE	GRADE	GRADE_COURSE	GRADE_STUDENT

If your answer does not fit, continue at the back of this page

Question 5 [10 points] {15 minutes}

Given is the domain model of question 4:



a. Write the code for the **GradeRepository** so that the GradeRepository contains the following query:

   Get all grades for the student with a given studentId. The studentId is a parameter. Use a named parameter.

```
public interface GradeRepository extends JpaRepository<Grade, Long> {

    @Query("select g.grade from Grade g where g.student.studentId =
:studentId")
    List<String> getGradesBySId2(@Param("studentId") Long studentId);

}
```


If your answer does not fit, continue at the back of this page

b. Write the code for the **StudentRepository** so that the StudentRepository contains the following 2 queries:
   - **Get all unique Students who took the course with a given course name. The course name is a parameter. Use a named parameter.**
   - **Get all unique Students who got a given grade in the course with a given name. The grade and course name are parameters. Use named parameters.**

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

    @Query("select distinct s from Student s join s.grades gr where
gr.course.name = :courseName")
    List<Student> findStudentTakenCourse(@Param("courseName") String
courseName);


    @Query("select distinct s from Student s join s.grades gr where
gr.course.name = :courseName and gr.grade = :grade")
    List<Student> findStudentGotGradeInCourse(@Param("courseName") String
courseName, @Param("grade") String grade);

}
```

c. Write the code for the **CourseRepository** so that the CourseRepository contains the following query:
   - **Get all unique Courses given by the professor with a given name. The professor name is a parameter. Use a named parameter.**

```
public interface CourseRepository extends JpaRepository< Course, Long> {

    @Query("select distinct c from Course c where c.professor.name= :name")
    List<Course> findCourseGivenByProfessor(@Param("name")String name);

}
```


If your answer does not fit, continue at the back of this page

Question 6 [20 points] {20 minutes}

Suppose we have the following entity classes. (Only the class name and attributes are shown)

```
@Entity
public class CreditCard {
    @Id
    @GeneratedValue
    private int id;
    private String number;
    private String name;
    private Date expiration;


@Entity
public class Address {
    @Id
    @GeneratedValue
    private int id;
    private String street;
    private String city;
    private String zip;


@Embeddable
public class Account {
    private String username;
    private String password;

@Entity
@SecondaryTable(name="customerstatus")
public class Customer {
   @Id
   @GeneratedValue
   private long id;
   @Column(name="first")
   private String firstname;
   @Column(name="last")
    private String lastname;

   @ManyToOne
   @JoinTable(name="customerAddress")
   private Address address;

   @OneToMany
   @JoinColumn(name="customerCreditcards")
    private Collection<CreditCard> creditcards=new ArrayList<CreditCard>();

   @Embedded
   private Account account;

   @Column(table="customerstatus")
   private String status;
```


If your answer does not fit, continue at the back of this page

For this question it is important that the entities map correctly on the given tables below. You are **NOT** allowed to change the table structure.

Write the annotations in the given code. **Do not rewrite all the code.**

Table: address

ID	CITY	STREET	ZIP
2	Chicago	Mainstreet 5	54778
6	Chicago	Mainstreet 33	54778

Table: creditcard

ID	EXPIRATION	NAME	NUMBER	CUSTOMER_CREDITCARDS
3	2022-06-29 21:52:39.528	Frank Brown	123	1
4	2022-06-29 21:52:39.528	Frank Brown	345	1
7	2022-06-29 21:52:39.6	Frank Johnson	123	5
8	2022-06-29 21:52:39.6	Frank Johnson	345	5

Table: customer

ID	PASSWORD	USERNAME	FIRST	LAST
1	pass1	user1	Frank	Brown
5	pass2	user2	Frank	Johnson

Table: customeraddress

ADDRESS_ID	ID
2	1
6	5

Table: customerstatus

STATUS	ID
Active	1
Not Active	5

If your answer does not fit, continue at the back of this page

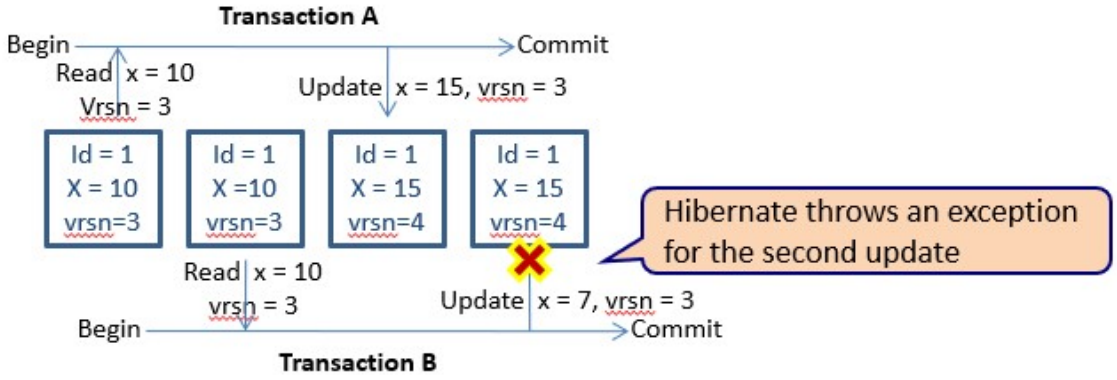
Question 7 [10 points] {10 minutes}

a. Explain clearly what problem is solved with the @Version attribute. Explain clearly which classes need a @Version attribute.

The @Version attribute avoids the Lost Update problem. Without the @Version attribute we get last commit wins. With the @Version attribute we get first commit wins.

All entity classes that can be changed at the same time by 2 or more users of the application need to have a @Version attribute to avoid the Lost Update problem.

b. Explain clearly **how** the @Version attribute solves this problem.



If your answer does not fit, continue at the back of this page

Question 8 [5 points] {10 minutes}

Describe how we can relate **Spring Boot** to one or more principles of SCI. Your answer should be about **2 to 3 paragraphs**. The number of points you get for this questions depends how well you explain the relationship between **Spring Boot** and one or more principles of SCI.

Answer:

If your answer does not fit, continue at the back of this page

Name: _____

Student Id: _____

1	
2	
3	
4	
5	
6	
7	
8	
Total	

CS544 Midterm exam

Question 1 [10 points] {10 minutes}

a. Suppose we have a Spring application with the following given XML configuration

```
<bean id="customerService" class="basic.CustomerService">
   <constructor-arg ref="emailService"/>
</bean>
<bean id="emailService" class="basic.EmailService">
   <constructor-arg ref="customerService"/>
</bean>
```

When we run the application, Spring gives an error. Explain clearly why Spring gives an error based on the given XML configuration.

With constructor injection Spring first need to initialize the bean that is injected. Then it can initialize the bean by using the constructor of this bean and pass the injected bean as parameter. In the given configuration we have a circular dependency. Spring can only create the CustomerService bean after it has created the Emailservice. But it can only create the EmailService after it has created the CustomerService.

b. Explain how we can solve this problem. **Write the XML configuration** that solves the problem and gives the same wiring of beans given in part a. It is important that your XML configuration results in the same spring beans and corresponding bean wiring as the XML configuration given in part a. Assume the corresponding Spring beans have all the necessary methods.

Use setter injection.

```
<bean id="customerService" class="basic.CustomerService">
   <property name="emailService" ref="emailService"/>
</bean>
<bean id="emailService" class="basic.EmailService">
   <property name="customerService"  ref="customerService"/>
</bean>
Or at least 1 bean should have setter injection
<bean id="customerService" class="basic.CustomerService">
   <property name="emailService" ref="emailService"/>
</bean>
<bean id="emailService" class="basic.EmailService">
   <constructor-arg ref="customerService"/>
</bean>
```


If your answer does not fit, continue at the back of this page

Question 2 [10 points] {10 minutes}

Circle all statements that are correct:

a. With the dirty read problem you can read uncommitted data.
b. With the phantom read problem you can read uncommitted data.
c. We map 1 class on 2 tables using @Embedded and @Embeddable
d. When we add a version to an entity and we annotate this with @Version then you will never have the dirty read problem on this entity.
e. All Spring beans that are singletons should be thread-safe and should be stateless.
f. In JPA, a @ManyToMany is always bi-directional.
g. If we do not allow the phantom read problem in our application, we cannot run 2 transactions at the same time.
h. When you use @AfterReturning on an AOP aspect method, you cannot get the return value in this method.
i. In a Spring boot application that uses JPA, you cannot use dependency injection on JPA entities.
j. In JPA, if you do not use @JoinColumn or @JoinTable on a OneToMany relationship, then this OneToMany relationship is by default mapped with a join column.

Question 3 [15 points] {20 minutes}

Given are the following entities:

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
      name="vehicle_type",
      discriminatorType=DiscriminatorType.STRING
)
public abstract class Vehicle {
   @Id
   @GeneratedValue
   private long id;
   private String brand;
   private String color;

   public Vehicle() { }

   public Vehicle(String brand, String color) {
      this.brand = brand;
      this.color = color;
   }
}
```


If your answer does not fit, continue at the back of this page

```java
@Entity
public abstract class Car extends Vehicle{
   private String licencePlate;
   public Car() { }
   public Car(String brand, String color, String licencePlate) {
      super(brand, color);
      this.licencePlate = licencePlate;
   }
}

@Entity
@DiscriminatorValue("RentalBycicle")
public class RentalBycicle extends Vehicle{
   private double pricePerHour;
   public RentalBycicle() {   }
   public RentalBycicle(String brand, String color, double pricePerHour) {
      super(brand, color);
      this.pricePerHour = pricePerHour;
   }
}


@Entity
@DiscriminatorValue("SellableCar")
public class SellableCar extends Car {
   private double sellPrice;
   public SellableCar() { }
   public SellableCar(String brand, String color, String licencePlate, double
sellPrice) {
      super(brand, color, licencePlate);
      this.sellPrice = sellPrice;
   }
}


@Entity
@DiscriminatorValue("RentalCar")
public class RentalCar extends Car {
   private double pricePerDay;
   public RentalCar() {   }
   public RentalCar(String brand, String color, String licencePlate, double
pricePerDay) {
      super(brand, color, licencePlate);
      this.pricePerDay = pricePerDay;
   }
}

public interface RentalBycicleRepository extends JpaRepository<RentalBycicle,
Long> {
}
public interface RentalCarRepository extends JpaRepository<RentalCar, Long> {
}
public interface SellableCarRepository extends JpaRepository<SellableCar,
Long> {
}
```


If your answer does not fit, continue at the back of this page

```
@SpringBootApplication
public class Application implements CommandLineRunner {
   @Autowired
   RentalCarRepository rentalCarRepository;
   @Autowired
   SellableCarRepository sellableCarRepository;
   @Autowired
   RentalBycicleRepository rentalBycicleRepository;

   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }

   @Override
   public void run(String... args) throws Exception {
      RentalCar rentalCar = new RentalCar("BMW", "Black", "KL-980-1", 67.00);
      rentalCarRepository.save(rentalCar);
      SellableCar sellableCar = new SellableCar("Audi", "White", "KM-956-2",
45980.00);
      sellableCarRepository.save(sellableCar);
      RentalBycicle rentalBycicle = new RentalBycicle("Moof", "Grey", 10.50);
      rentalBycicleRepository.save(rentalBycicle);

   }

}
```

a. In the given code above, add the necessary mapping annotations so that the whole inheritance hierarchy is mapped according the **single table per hierarchy** strategy. Do **NOT** rewrite any code. Only write the correct annotations in the given code.

b.

c. Explain **ALL** advantages and disadvantages we learned about the **single table per hierarchy** strategy.

Advantages

+ Simple, Easy to implement
+ Good performance on all queries, polymorphic and non-polymorphic

Disadvantages

- Nullable columns / de-normalized schema
- Table may have to contain lots of columns
- A change in any class results in a change of this table

If your answer does not fit, continue at the back of this page

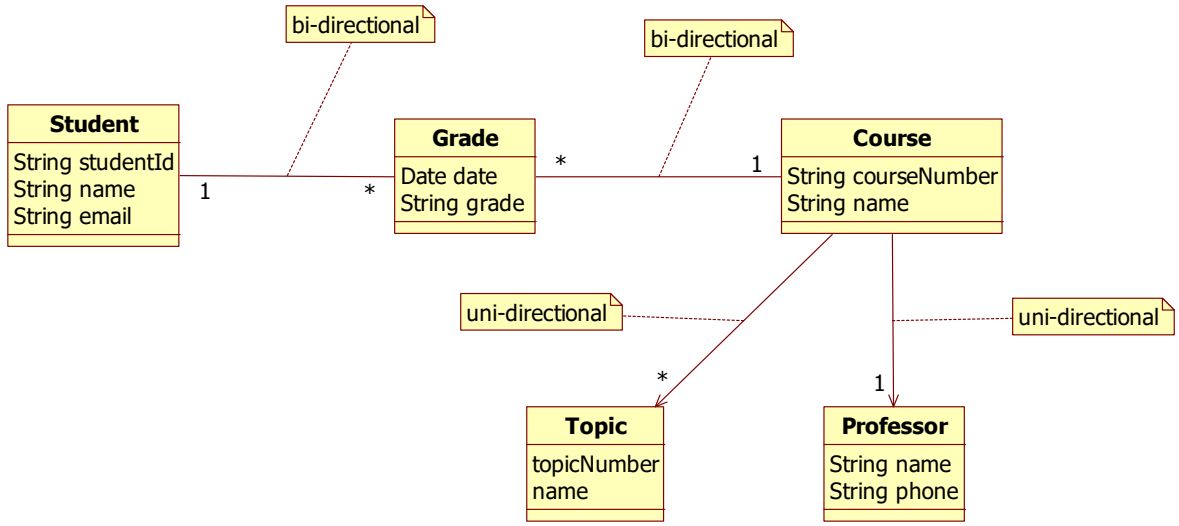
d. Draw the corresponding database table with all the columns and corresponding data if we run Application.java.

VEHICLE_TYPE	ID	BRAND	COLOR	LICENCE_PLATE	PRICE_PER_HOUR	PRICE_PER_DAY	SELL_PRICE
RentalCar	1	BMW	Black	KL-980-1		67	
SellableCar	2	Audi	White	KM-956-2			45980.0
RentalBycicle	3	Moof	Grey	[null]	10.5		

If your answer does not fit, continue at the back of this page

Question 4 [20 points] {25 minutes}

Given is the following domain model:



This domain model has the following requirements:

- The Course has a **Map** of Topics. The topicNumber is the key in the Map.
- The Course has a **List** of Grades. The grades in this List should always be ordered by grade descending.

a. Below you find the corresponding Java classes. The constructor or any getter or setter methods are not shown. In the given code, add the necessary JPA annotations so that we can save these classes to the database. Do **NOT** implement fetching or cascading configuration.

```
@Entity
public class Topic {

    @Id
    @GeneratedValue
    private Long topicNumber;
    private String name;

@Entity
public class Professor {

    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private String phone;
```


If your answer does not fit, continue at the back of this page

```java
@Entity
public class Student {

    @Id
    private Long studentId;
    private String name;
    private String email;

    @OneToMany(mappedBy = "student")
    private List<Grade> grades= new ArrayList<>();


@Entity
public class Grade {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    @JoinColumn(name="grade_course")
    private Course course;

    @ManyToOne
    @JoinColumn(name="grade_student")
    private Student student;

    private Date date;
    private String grade;


@Entity
public class Course {

    @Id
    @GeneratedValue
    private Long courseNumber;

    private String name;

    @OneToMany(mappedBy = "course")
    @OrderBy("grade DESC")
    private List<Grade> grades = new ArrayList<Grade>();

    @ManyToOne
    private Professor professor;

    @OneToMany
    @MapKey(name="topicNumber")
    private Map<Long, Topic> topics = new HashMap<>();
```


If your answer does not fit, continue at the back of this page

b. Draw the tables of the database including their columns according your JPA mapping of part a. You do not need to show the data in the tables. Only show the tables and their columns.

Topic table:

TOPIC_NUMBER	NAME

Professor table:

ID	NAME	PHONE

Student table:

STUDENT_ID	EMAIL	NAME

Course table:

COURSE_NUMBER	NAME	PROFESSOR_ID

CourseTopic table:

COURSE_COURSE_NUMBER	TOPICS_TOPIC_NUMBER

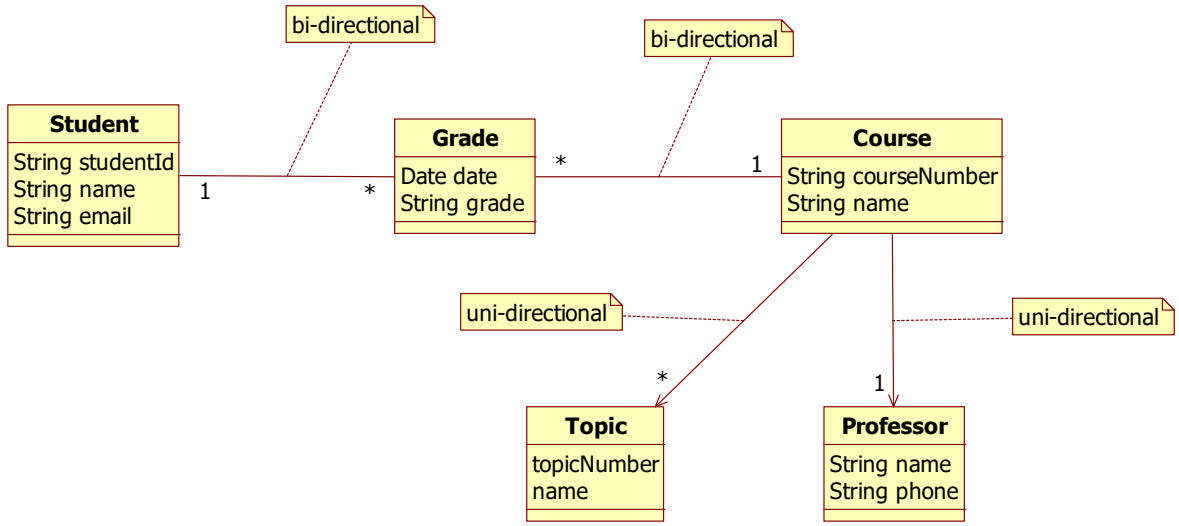
Grade table:

ID	DATE	GRADE	GRADE_COURSE	GRADE_STUDENT

If your answer does not fit, continue at the back of this page

Question 5 [10 points] {15 minutes}

Given is the domain model of question 4:



a. Write the code for the **GradeRepository** so that the GradeRepository contains the following query:

   Get all grades for the student with a given studentId. The studentId is a parameter. Use a named parameter.

```
public interface GradeRepository extends JpaRepository<Grade, Long> {

    @Query("select g.grade from Grade g where g.student.studentId =
:studentId")
    List<String> getGradesBySId2(@Param("studentId") Long studentId);

}
```


If your answer does not fit, continue at the back of this page

b. Write the code for the **StudentRepository** so that the StudentRepository contains the following 2 queries:
   - **Get all unique Students who took the course with a given course name. The course name is a parameter. Use a named parameter.**
   - **Get all unique Students who got a given grade in the course with a given name. The grade and course name are parameters. Use named parameters.**

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

    @Query("select distinct s from Student s join s.grades gr where
gr.course.name = :courseName")
    List<Student> findStudentTakenCourse(@Param("courseName") String
courseName);


    @Query("select distinct s from Student s join s.grades gr where
gr.course.name = :courseName and gr.grade = :grade")
    List<Student> findStudentGotGradeInCourse(@Param("courseName") String
courseName, @Param("grade") String grade);

}
```

c. Write the code for the **CourseRepository** so that the CourseRepository contains the following query:
   - **Get all unique Courses given by the professor with a given name. The professor name is a parameter. Use a named parameter.**

```
public interface CourseRepository extends JpaRepository< Course, Long> {

    @Query("select distinct c from Course c where c.professor.name= :name")
    List<Course> findCourseGivenByProfessor(@Param("name")String name);

}
```


If your answer does not fit, continue at the back of this page

Question 6 [20 points] {20 minutes}

Suppose we have the following entity classes. (Only the class name and attributes are shown)

```
@Entity
public class CreditCard {
    @Id
    @GeneratedValue
    private int id;
    private String number;
    private String name;
    private Date expiration;


@Entity
public class Address {
    @Id
    @GeneratedValue
    private int id;
    private String street;
    private String city;
    private String zip;


@Embeddable
public class Account {
    private String username;
    private String password;

@Entity
@SecondaryTable(name="customerstatus")
public class Customer {
   @Id
   @GeneratedValue
   private long id;
   @Column(name="first")
   private String firstname;
   @Column(name="last")
    private String lastname;

   @ManyToOne
   @JoinTable(name="customerAddress")
   private Address address;

   @OneToMany
   @JoinColumn(name="customerCreditcards")
    private Collection<CreditCard> creditcards=new ArrayList<CreditCard>();

   @Embedded
   private Account account;

   @Column(table="customerstatus")
   private String status;
```


If your answer does not fit, continue at the back of this page

For this question it is important that the entities map correctly on the given tables below. You are **NOT** allowed to change the table structure.

Write the annotations in the given code. **Do not rewrite all the code.**

Table: address

ID	CITY	STREET	ZIP
2	Chicago	Mainstreet 5	54778
6	Chicago	Mainstreet 33	54778

Table: creditcard

ID	EXPIRATION	NAME	NUMBER	CUSTOMER_CREDITCARDS
3	2022-06-29 21:52:39.528	Frank Brown	123	1
4	2022-06-29 21:52:39.528	Frank Brown	345	1
7	2022-06-29 21:52:39.6	Frank Johnson	123	5
8	2022-06-29 21:52:39.6	Frank Johnson	345	5

Table: customer

ID	PASSWORD	USERNAME	FIRST	LAST
1	pass1	user1	Frank	Brown
5	pass2	user2	Frank	Johnson

Table: customeraddress

ADDRESS_ID	ID
2	1
6	5

Table: customerstatus

STATUS	ID
Active	1
Not Active	5

If your answer does not fit, continue at the back of this page

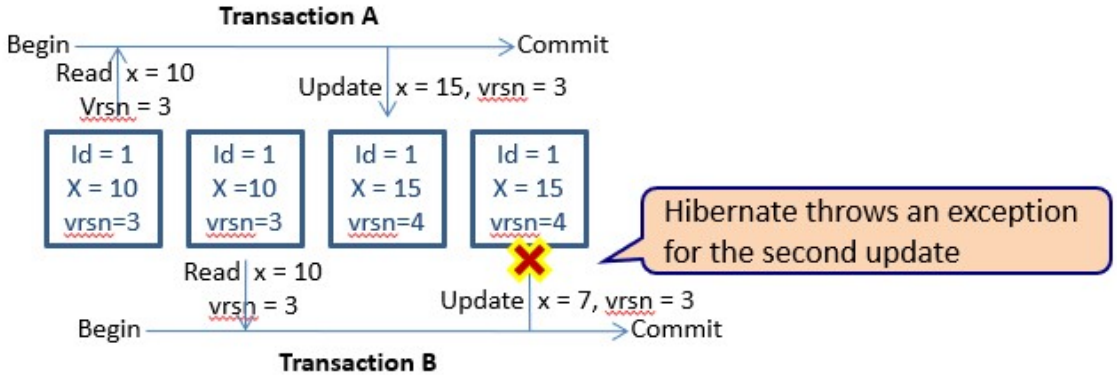
Question 7 [10 points] {10 minutes}

a. Explain clearly what problem is solved with the @Version attribute. Explain clearly which classes need a @Version attribute.

The @Version attribute avoids the Lost Update problem. Without the @Version attribute we get last commit wins. With the @Version attribute we get first commit wins.

All entity classes that can be changed at the same time by 2 or more users of the application need to have a @Version attribute to avoid the Lost Update problem.

b. Explain clearly **how** the @Version attribute solves this problem.



If your answer does not fit, continue at the back of this page

Question 8 [5 points] {10 minutes}

Describe how we can relate **Spring Boot** to one or more principles of SCI. Your answer should be about **2 to 3 paragraphs**. The number of points you get for this questions depends how well you explain the relationship between **Spring Boot** and one or more principles of SCI.

Answer:

If your answer does not fit, continue at the back of this page