Full Name: kirs Gebregeworgis **Student ID:** 610765 **CS544 Midterm - April 2020**

Note Write your name on all pages of exam (including extra sheets of paper given to you). *Please make sure you read all questions. There are 4 pages of questions. You can answer on exam paper.*

1) [10 points] – List 5 different tiers/layers of **Service Oriented Architecture**. No explanation necessary.
1view 2, Controller 3, Service 4, domain model 5) persistance layer 10/10

2) [6 points] – Explain the difference between "**field**" vs. "**property**" accesses in Hibernate? 6

3) [4 points] – What are two main advantages of using an ORM (versus using plain database calls)? 4

10 4) [10 points] – Explain the difference between implicit vs. explicit updates in Hibernate?

10 5) [10 points] – Explain the **Single-Table** inheritance strategy in JPA?

10 6) [10 points] – Name two advantages and two disadvantages of using the **Joined-Table** inheritance strategy. Provide a little explanation so that I know that you understand it.

7) [10 points] – Under what situations Hibernate flushes the cache and writes all updates to DB? Name 3.
10

8) [10 points] – Is this the proper way to map a bi-directional mapping in JPA? Explain.
10

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;
  @OneToMany
  @JoinColumn(name="person_id")
  private List<Car> cars =
          new ArrayList();
```

```
@Entity
public class Car {
  @Id
  @GeneratedValue
  private int id;
  private short year;
  private String model;
  private String maker;
  @ManyToOne
  @JoinColumn(name="owner_id")
  private Person owner;
```

9) [10 points] – Given the requirements of next questions (DB and Java design), write an HQL query to find out:
10

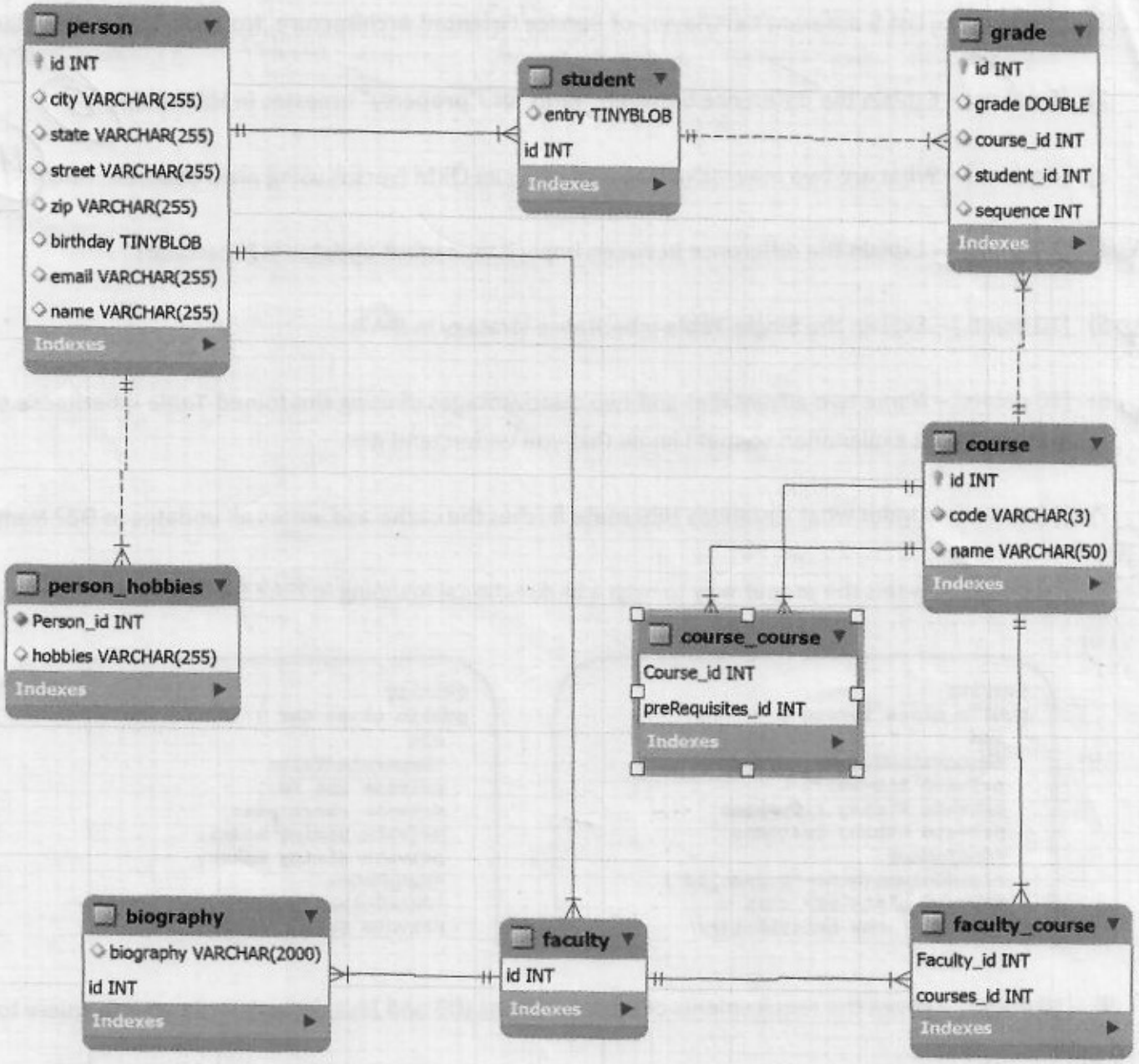
a) List of students who have taken course with code 544 and have a grade of 3.8 or better
"select s from students s join s.grades g where g.grade ≥ 3.8 and g.course.code='544'" +5

b) List of faculty whose biography is more than 1000 characters
"Select f from faculty f where f.biography.length > 1000" +5

120 = 100/100 !!!

40

10) [40 points] – Given the following database ER (Entity Relationship) diagram:



Annotate the following classes with JPA annotations to map them to the above DB.

Note: Pay attention to relationships in Java and where each attribute is potentially going to be mapped to

Use field access.

```
@Embeddable
public class Address {

	private String street;

	private String city;

	private String state;

	private String zip;

… // getters and setters
}
```

```
@Entity
public class Course {
	@Id
	@GeneratedValue
	private Integer id;
	@Column(name="code", length=3, nullable=false)
	private String code; // Maximum of 3 characters (e.g. "544"), required field
	@Column(name="name", length=50, nullable=false)
	private String name; // Maximum of 50 characters, required field
	@OneToMany
	@JoinTable(name="course_course", joinColumns=@JoinColumn(name="course_id"),
	inverseJoinColumns=@JoinColumn(name="prerequisite_id"))
	// Set of pre-requisite courses for this course. Can be empty.
	private Set<Course> preRequisites = new HashSet<>();

… // Getters and setters
}
```

```
@Entity
public class Grade {
	@Id
	@GeneratedValue
	private Integer id;

	private Double grade;
	@ManyToOne
	@JoinColumn(name="course_id")
	private Course course;

	@ManyToOne
	@JoinColumn(name="student_id")
	private Student student;
… // Getters and setters
}
```

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Person {
        @Id
        @GeneratedValue
        private Integer id;

        private String name;

        private String email;
        @Temporal(TemporalType.DATE)
        private LocalDate birthday;
        @ElementCollection
        private List<String> hobbies = new ArrayList<>();
        @Embedded
        private Address address;

    … // Getters and setters
}

@Entity
public class Student extends Person {

        private LocalDate entry;
        @OneToMany(mappedBy="student", cascade = CascadeType.PERSIST)
        @OrderColumn(name="sequence")
        private List<Grade> grades = new ArrayList<>();

    … // Getters and setters
}

@Entity
@SecondaryTable(name="biography")
public class Faculty extends Person {

        @Column(table="biography", length=2000, nullable=true)
        private String biography; //optional field with a maximum length of 2000 characters
        @ManyToMany
        @JoinTable(name="faculty_course")
        private Set<Course> courses = new HashSet<>();

    … // Getters and setters
}
```

kinos Gebregewergs ID 6/07/05

1) 1. view layer
2. Controller layer
3. service layer
4. domain model layer
5. persistance layer

10/10

2) - field access is when we put @Id annotation before the ~~field~~ hibernate will gain field access. and all other annotation has to be below this

- property access is when we put @Id annotation before the setters and getters hibernate will have property access. all other annotation has to be below this annotation

6

The property access give information hiding which is the main OO concept Encapsulation.

3) . . enable you to work directly with domain object
- supports inheritance and associations

4/4

4) - implicit update is if we have managed entity and moding the object using it's setters the hibernate have dirty checking to see if the object is modified after the transaction is commited. if there's change the change persisted to the database implicitly.

10

- Explicit update - We use this method if we want to update detached object from persistance Context. but we have to be careful when we use this method because it will throw nonunique exception if we have already managed entity with same id. but we can still solve this using another method called merge.

5) It creates a single table per heirarchy of the inheritance
- All the property of the sub class will be add to that table as column.
- it differentiate the subclass by using @discriminatory Column.
- the subclass column allow them to be null which results denormalized schema.
- It's good to use this strategy if the subclass doesn't have many property

6) Advantages
1) will result normalized schema - because every entity is going to be stored on it's own table with foreign key constraint on the primary key
2) Database view is similar to the domain class - because all class mapped into that table.

10

8 disadvantages
1 insert or update query results multiple insert or update statement in database
2 Neccassary join results poor query performance. because when we do polymorphic query it join multiple tables that will result poor performance. beca this is due to join query results poor performance.

7) 1 when we call session.flush() method
2 When we call transaction.commit method
3 When we execute query the hibernate checks first if there's any before the query then executes them first to database then execute the query.

10/10.

8) No this mapping isn't proper mapping because it misses some mapping properties

1) If we have one to many bidirectional association We have to set the owner of the relation to be the many side by setting mappedby property on the one to many side. We don't @JoinColumn on the @OneToMany side because it's redundant Column.

2) the second thing it misses cascaded property saying person will not persist the car and many other Cascade properties

I would correct the mapping like this

@OneToMany(mappedBy = "owner", Cascade = CascadeType.ALL, PERSIST)

10/10.

kiros Gebregewergs 6/07/65