EA Practice midterm

Question 1 [10 points] {10 minutes}

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

```xml
<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.

Answer:

both beans depends on each other, this will cause an issue becuase each bean will be waiting for the other bean to get initiated, to fix this, one of them should be setter injection

b. Explain why we need an **init()** method in Spring Boot.

Answer:

this method will get called just after the constructor and we use it to to perform tasks after the bean is initilazed and has been injected with the dependicies.

Question 2 [15 points] {20 minutes}

Suppose we need to write a **Spring Boot** application that allow us to store and find Products. A Product consists of the following attributes: productNumber, name, price and categoryName. A categoryName is something like “clothing” or “toys” or “electronics” The application should allow us to store new Products and we should be able to find products with the following functionality:

- Give all products with a price bigger than a given amount
- Give all products from a certain category

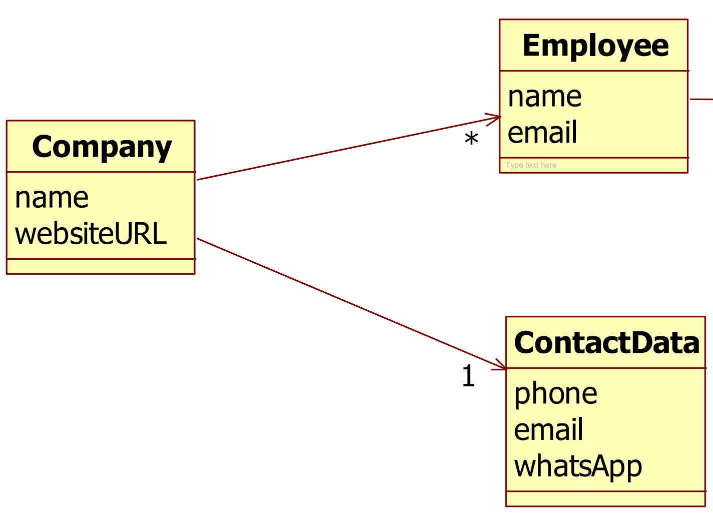
Write **ALL** necessary Java code including annotations. Do **NOT** write the Application class (that contains the main() method). Do **NOT** write imports and do **NOT** write getter and setter methods. Also do **NOT** write constructors.

Use all the best practices we learned in this course.

```
@Entity
public class Product  {
    @Id
    @GeneratedValue
    private long productNumber;
    private String name;
    private double price;
    private String categoryName;
}

public interface ProductRepository extends JpaRepository<Product, Long> {

public List<Product> findByPriceGreaterThan(double price);
public List<Product> findByCategoryName(String category);

}

@Repository
public class ProducService {
@Autowired
ProductRepository productRepository;

public void saveProduct(ProductDTO product) {
productRepository.save(ProductAdapter.productFromProductDTO(product));
}

}

public record ProductDTO(long productNumber, String name, double price, String categoryName){}


public class ProductAdapter {

public static ProductDTO productDTOfromProduct(Product product) {
return new ProductDTO(product.productNumber, product.name, product.price, product.categoryName);
}

public static Product productFromProductDTO( ProductDTO product) {
return new Product(product.productNumber, product.name, product.price, product.categoryName);
}

}
```

Question 3 [15 points] {15 minutes}

Suppose we have the following JPA entities:



We need to write the following queries:

These queries should be defined by the method name in the repository:

- **Give all Companies with a given name. Name is a parameter.**
- **Give all streets given a certain city and a certain zip**

These queries should be defined by **@Query** in the repository:

- **Give the name of all companies from a given city**
- **Give the name of the company given a certain phone number**
- **Give all Companies where an employee works with a certain given name.**

Write the queries in the corresponding repositories. Write the **complete Java code** of all necessary repositories including the methods and the annotations. **Do not write Java imports**

```
public interface CompanyRepository extends JpaRepository<Company, Long> {
public List<Company> findByName(String name);
@Query("Select c from Company c Join c.employees e Join e.address where a.city = :city")
public List<Company> findCompaniesByCity(@Param("city") String city);
@Query("Select cd.phone from Company c join c.contactData cd where cd.phone = :phone")
public List<String> findCompanyNameByPhone(@Param("phone") String phone);
@Query("Select c from Company c Join c.employees e where e.name = :name")
public List<Company> findCompanyEmployeeName(@Param("name") String name);
}

public interface AddressRepository extends JpaRepository<Address, Long> {
public List<String> findStreetByCityAndZip(String city, String zip);
}
```

Question 4 [20 points] {20 minutes}

Given are the following entities:

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
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;
   }
}
```

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
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
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
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
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> {
}

@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 **all 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. Explain **ALL** advantages and disadvantages we learned about the **single table per hierarchy** strategy.
Answer:

You don't have null as if you use single table
You have a table for each entity
flixibility becuase you can add more types without changing the actual table
the tables will have small size comparing to the single table

it's slower than single table becuase you will need to have joins
it's more complex than a single table

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

all tables will have the same columns that exist in the class plus the id of the parent with disreminator column

d. Suppose we map the given inheritance hierarchy with the strategy **Joined Tables**. Draw the corresponding database tables with all the columns and corresponding data if we use the strategy **Joined Tables**

all tables will have attributes that exist in the class plus the id of the parent
with discreminator column

e. Suppose we map the given inheritance hierarchy with the strategy **Table per concrete class**. Draw the corresponding database tables with all the columns and corresponding data if we use the strategy **Table per concrete class**

all tables will have all the attributes that exist in the class and in it's parent
without discreminator column

Question 5 [10 points] {15 minutes}

Circle all statements that are correct:

T a. When we add a version attribute to an entity and we annotate this with @Version then you will never have the dirty read problem on this entity.

T b. If we do not allow the phantom read problem in our application, we cannot run 2 transactions at the same time.

F c. In a Spring boot application that uses JPA, you cannot use dependency injection on JPA entities.

F d. When you make one method of a Spring bean transactional the 2 phase commit protocol will never be used. If you make 2 or more methods of a Spring bean transactional the 2 phase commit protocol will be used.

F e. Cascading is only applicable for inserts, updates and deletes.

T f. In JPA, a @OneToOne relation is stored in the database as a @ManyToOne relation.

F g. A named query cannot contain a join.

F h. An entity class in a Spring Boot JPA application is always a singleton.

F i. With the TransactionReadCommitted isolation level, you can never have the lost update problem

F j. In databases that use sequences, every table contains a sequence column.