Win Ei Khaing

613403

SCI: 4
Total: 95.5

Theory Questions:

a. [3 pts] What is an Aspect in the context of Aspect Oriented Programming? (give the definition)

3

An Aspect is a combination of advice and pointcut (collection of points)
It is what (advice) should execute where (pointcut).

b. [3 pts] Explain why Spring cannot perform AOP on local calls:

3

Because of IoC and DI, Spring can inject something else (a proxy) in between. Proxy then calls the desired code(advice) before and/or after. In that case, while proxy calls the real method, if that method calls another real method, Spring couldn't inject other proxies between them. It is the same when advice method calls joinpoint.getThis() or joinpoint.getTarget();

c. [3 pts] Explain why transactions are not optional when using a database

3

If we want to make a process that will touch two tables in database, if there is no transactions, when something's wrong at processing with second tables, it will not rollback for the first table. So, transaction is not optional for reliability. But there is no database which has no transaction for default. @Transactional(propagation = Propagation.REQUIRED) is default.

d. [3 pts] Explain what URI Templates are:

3

URI templates are Strings like URI with variables e.g "/posts/{id}"
When we substitute the variables, it becomes real URI
e.g "/posts/1"

e. [3 pts] Explain what Spring Boot profiles are:

3

Spring Boot profiles are for development environment e.g dev, prod
As Spring Boot can't configure automatically for that, we have to change profile (only if we want) in external application.properties

f. [3 pts] What is the difference is between @Controller and @RestController?

3

@Controller is just a specification of @Component.
@RestController is the combination of @Controller and @ResponseBody, by adding @ResponseBody on the class level, applying for all methods in that class.

g. [3 pts] Give the different HTTP methods that RESTFull uses. Be sure to explain what each verb should be used for.

3

POST - for inserting a new one
PUT - For updating the existing one
DELETE - for deleting the existing on
GET - for retrieving the data
PATCH - for inserting or updating (we can't use directly like other. this one needs additional configuration)

Win Ei Khaing 613403

Code Exercises:

1. [15 pts] What is the output of the following application? All classes are inside the cs544 package.

```
@Configuration
@ComponentScan("cs544")
@EnableAspectJAutoProxy
public class Config {
}
```

```
public class App {
    public static void main(String[] args) {
        ConfigurableApplicationContext context = new
            AnnotationConfigApplicationContext(Config.clas
s);
        System.out.println("Testing Spring Startup");
        Third t = context.getBean(Third.class);
        System.out.println("In main: " + t.getText());
        context.close();
    }
}
```

```
@Component
@Aspect
public class MyAspect {
    @Value("Test")
    private String text;
    @PostConstruct
    public void start() {
        System.out.println(
            "MyAspect start method - text: " + text);
    }
    @Around("execution(* cs544.*.get*(..))")
    public Object beforeTrace(ProceedingJoinPoint pjp)
                                    throws Throwable {
        String name =
            pjp.getTarget().getClass().getSimpleName();
        if (name.equals("Second")) {
            return "Something";
        }
        return pjp.proceed();
    }
}
```

```
@Component
public class First {
    protected Second second;
    public First() {
        System.out.println("First construction - Second:"
                                                + second);
    }
    @Autowired
    public void setSecond(Second second) {
        System.out.println("Setting second");
        this.second = second;
    }
    public Second getSecond() {
        return second;
    }
}
```

```
@Component
public class Second {
    @Value("Thing")
    private String text;
    public Second() {
        String text = "Value";
        System.out.println("Second constructor text is: "
                                                + text);
        this.text = text;
    }
    public String getText() {
        return text;
    }
}
```

```
@Lazy
@Component
public class Third extends First {
    @Value("Random")
    private String text;
    public String getText() {
        return text + " " + second.getText();
    }
    @PostConstruct
    public void start() {
        System.out.println("Third starting - text: "
                                                + text);
    }
    @PreDestroy
    public void end() {
        System.out.println("Third exiting - text: "
                                                + text);
    }
}
```

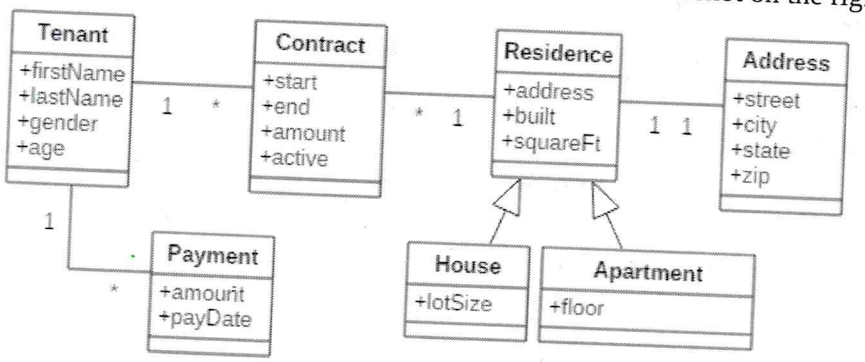
1.

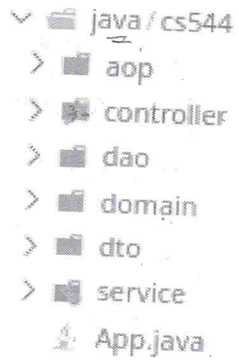
First construction - Second : null
Second constructor text is: Value
Setting second
MyAspect start method - text : Test
Testing Spring Startup
Third starting - text: Random
In main: Random Thing
Third existing - text : Random

12

// as NO AOP For local call

All of the code exercises after this belong together. In essence you are going to make one application. The package structure for this application is shown in the screenshot on the right, class diagram on the left:





The code for the entities shown on the class diagram can be found on the next page. The application is a property rental application (as might be used by a landlord). The core of our application should be a RentalService, in this exam you should implement the following methods on the interface:

```
public interface RentalService {
    public List<Residence> getResidences();
    public Integer addHouse(House house, Address address);
    public boolean deleteResidence(Integer residenceId);
    public Long updateContract(Long contractId, Contract contract);
    public List<Payment> underPayments(double amount);
}
```

I will briefly describe what each method should do:

- **getResidences()** returns a list of all the residences in the database
- **addHouse()** receives a House and an Address, sets the address on the house, persists the house and returns the newly created id of the house
- **deleteResidence()** receives a house Id, cehecks that there is no active contract for that residence, and deletes the house when there are no contracts. Returns false if there is a contract.
- **updateContract()** receives a contractId and a contract, where the incoming contract object doesn't have the tenant or residence set – it just functions as a DTO to bring updated start, end, amount, and active values. The method should use the contractId to retrieve the contract from the database, and then update it with the start/end/amount/active values from the passed contract.
- **underPayments()** takes an amount and returns all the payments that were less than that amount.

2. [10 pts] Start with your configuration on this page (and the back if needed). Use Spring Boot (which means you'll need a class with a main method and also an application.properties file for the hibernate config).

2.
```
@SpringBootApplication
//@ComponentScan("cs544") //optional but I add because it needs during my project
@EntityScan("cs544.domain")
@EnableJpaRepositories("cs544.dao")

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

10

Win Ei Khaing

613403

The following entities are part of this domain:

```
@Entity
public class Tenant {
  @Id
  @GeneratedValue
  private Long id;
  private String firstName;
  private String lastName;
  private String gender;
  private int age;
  @JsonIgnore
  @OneToMany(mappedBy = "tenant")
  private List<Contract> contracts = new ArrayList<>();
  @OneToMany(mappedBy = "tenant", cascade =
      {CascadeType.MERGE, CascadeType.PERSIST})
  @JsonIgnore
  private List<Payment> payments = new ArrayList<>();
}
```

```
@Entity
public class Contract {
  @Id
  @GeneratedValue
  private Long id;
  @ManyToOne(cascade = CascadeType.PERSIST)
  private Tenant tenant;
  @ManyToOne(cascade = CascadeType.PERSIST)
  private Residence residence;
  @Temporal(TemporalType.DATE)
  @JsonFormat(pattern="yyyy-MM-dd")
  private Date start;
  @Temporal(TemporalType.DATE)
  @JsonFormat(pattern="yyyy-MM-dd")
  @Column(name="stop")
  private Date end;
  private double amount;
  private boolean active;
}
```

```
@Entity
public abstract class Residence {
  @Id
  @GeneratedValue
  private Integer id;
  @Temporal(TemporalType.DATE)
  @JsonFormat(pattern="yyyy-MM-dd")
  private Date built;
  private int squareFt;
  @JsonIgnore
  @OneToMany(mappedBy = "residence")
  private List<Contract> contracts = new ArrayList<>();
  @Embedded
  private Address address;
}
```

```
@Entity
public class House extends Residence {
  private int lotSize;
}
```

```
@Entity
public class Apartment extends Residence {
  private int floor;
}
```

```
@Embeddable
public class Address {
  private String street;
  private String city;
  private String state;
  private String zip;
}
```

```
@Entity
public class Payment {
  @Id
  @GeneratedValue
  private Long id;
  private double amount;
  @Temporal(TemporalType.DATE)
  private Date payDate;
  @ManyToOne
  private Tenant tenant;
}
```

For all of the following exercises write your code on additional pieces of paper.

15 3. [10 pts] Make a ResidenceDao, ContractDao, and PaymentDoa JpaRepository classes, and add finder methods where appropriate.

15 4. [15 pts] Create a RentalServiceImpl class that implements the RentalService interface shown earlier, and uses the DAOs from previous exercise

15 5. [15 pts] Write a RentalController RestController class that exposes the five methods of the RentalService. If you want you can create DTO objects to make input easier.

8.5 6. [10 pts] Create an DiscountAspect class with an advice for the updateContract() method. It should check if the firstName of the tenant is "Buddy", and if so reduce the amount on the incoming contract object by 100.

application.properties

```
spring.datasource.url = jdbc:mysql:///localhost/cs544?useSSL=false&serviceTimezone
                        = America/Chicago
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto = create-drop
spring.jpa.hibernate.use-new-id-generator-mappings = false
spring.mvc.view.prefix = /WEB/INF/view/
spring.mvc.view.suffix = .jsp
logging.level.root = WARN
```

3.

```
@Repository
public interface ResidenceDao extends JpaRepository<Residence, Integer>{

    public List<Residence> findAll();

    @Query("SELECT COUNT(c)
            FROM Residence r JOIN r.contracts c
            WHERE r.id = :residenceId AND
            c.active = 1")

    public Integer findActiveContractCount
                    (Integer residenceId);

}
```

```
@Repository
public interface ContractDao extends JpaRepository<Contract, Long> {



}
```

```
@Repository
public interface PaymentDao extends JpaRepository<Payment, Long
    @Query("FROM Payment p WHERE p.amount <:amount")
    public List<Payment> getUnderPayments(double amount);

}
```

4.

```
@Service
@Transactional
public class RentalServiceImpl implements RentalService {

    @Autowired
    private ResidenceDao residenceDao;

    @Autowired
    private ContractDao contractDao;

    @Autowired
    private PaymentDao paymentDao;

    @Override
    public List<Residence> getResidences() {
        return residenceDao.FindAll();

    }
```

```
@Override
public Integer addHouse(House house, Address address){
        house.setAddress(address);
        residenceDao.save(house);
        return house.getId();
}

@Override
public boolean deleteResidence(Integer residenceId){
    Integer activeContractCount = residenceDao.FindActiveContractCount
                                  (residenceId)

    if(activeContractCount > 0)
        return false;

    residenceDao.deleteById(residenceId);
}

@Override
public Long updateContract(Long contractId, Contract contract){

        Contract existingContract = contractDao.getById(contractId);
        existingContract.setStart(contract.getStart());
        existingContract.setEnd(contract.getEnd());
        existingContract.setAmount(contract.getAmount());
        existingContract.setActive(contract.getActive());

        contractDao.save(existingContract);

        return existingContract.getId();
}
```

```
@Override
public List<Payment> underPayments(double amount){
    return paymentDao.getUnderPayments(amount);
}
}
```

5.

```
@RestController
public class RentalController {
    @Autowired
    private RentalService rentalService;

    @GetMapping("/residences")
    public List<Residence> getResidences() {
        return rentalService.getResidences();
    }
    @PostMapping("/residences")
    public Integer addHouse(@RequestBody HouseDTO houseDto){
        return rentalService.addHouse(houseDto.house,
                                      houseDto.add);
    }
    //as there is no validations in entity classes,
    //I didn't add @Valid and didn't use BindingResult

    @DeleteMapping("/residences/{residenceId}")
    public boolean deleteResidence(Integer residenceId){
        return rentalService.deleteResidence(residenceId);
    }
```

```
public class HouseDTO
    private House hous
    private Address add
}
```

```
@PutMapping("/contracts/{contractId}")
public Long updateContract(@PathVariable Long contractId,
                           @RequestBody Contract contract){

    return rentalService.updateContract
                (contractId, contract);
}

@GetMapping("/payments")
public List<Payment> underPayments(@RequestParam double
                                   amount){

    return rentalService.underPayments(amount);
}
}
```

✓ ✓

```
@Component
@Aspect
public class DiscountAspect {

    @Around("execution(* cs544..*.updateContract(..))") {
    public Object beforeTrace(ProceedingJoinPoint pjp)
                throws Throwable {

        Object[] args = pjp.getArgs();
    Integer id = (Integer) args[0];
    Contract contract = (Contract) args[1];
    if(contract.tenant.firstName == "Buddy")
            contract.setAmount(contract.amount - 100);

    args[1] = contract;

    return pjp.proceed(args); // as I changed args[1]'s amount

    }
}
```

as mentioned on page 4 the incoming contract doesn't have a tenant set

-1.5