Name: Thanh Do Nguyen StudentID: 615941

93 inc review

Final Exam 2023-06

CS544 Enterprise Applications

Theory Section

A. [3 pts] What does the JavaBean standard specify?

@Bean is used in java config for the factory methods. x

0

B. [3 pts] Explain what autowiring is

3 the way to implement injection by using @Autowired. it gain greater flexibility, loose coupling.

C. [3 pts] What does the @Lazy annotation do in Spring?

3 to tell spring not create that Beans when startup. It will be create when call.

D. [3 pts] What is meant with Advice in the context of AOP?

3 Advice implement action of the cross cutting concern.

E. [3 pts] Explain what JoinPoint is in the context of AOP

3 JoinPoint is a specific point in code.

F. [3 pts] Explain what the difference is between BMT and CMT (in the context of transactions)

3 BMT: manage by super transaction / global transaction.
CMT: manage multiple transactions for Each Situation.

G. [3 pts] Explain the difference between @RequestParam and @PathVariable

3 @RequestParam: bind a variable in the URL path like "/book?id=1"
@PathVariable: bind a variable in the URL path like "/book/{id}" > for Example: "book/1"

H. [3 pts] Explain what profiles are in the context of Spring Boot

3 using diffrent configuration for diffrent environments.

Name: Thanh Do Nguyen StudentID: 615941

1. [15 pts] What is the output of the following application (all classes are in the cs544 package):

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

```
public class App {
    public static void main(String[] args) {
        ConfigurableApplicationContext context = new
            AnnotationConfigApplicationContext(Config.class);
        System.out.println("Testing Spring Startup");
        ClassC c = context.getBean(ClassC.class);
        System.out.println("In main: " + c.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("ClassB")) {
            System.out.println("MyAspect doesn't like ClassB");
            return "Something";
        }
        System.out.println("MyAspect says: Fine, be that way");
        return pjp.proceed();
    }
}
```

```
@Component
public class ClassA {
  private ClassB classB;

  public ClassA() {
    System.out.println("ClassA constructor - ClassB is: " + classB);
  }
  @Autowired
  public void setClassB(ClassB classB) {
    System.out.println("Setting classB");
    this.classB = classB;
  }
  public ClassB getClassB() {
    System.out.println("ClassA returning ClassB");
    return classB;
  }
}
```

```
@Component
public class ClassB {
  @Value("Thing")
  private String text;

  public ClassB() {
    String text = "Value";
    System.out.println("ClassB constructor text is: " + text);
    this.text = text;
  }
  public String getText() {
    return text;
  }
}
```

```
@Lazy
@Component
public class ClassC extends ClassA {
  private String text;

  public ClassC() {
    System.out.println("ClassC constructor");
  }

  @Value("Random")
  public void setText(String text) {
    System.out.println("ClassC setting text to: " + text);
    this.text = text;
  }
  public String getText() {
    System.out.println("ClassC getText " + text);
    return text + " " + this.getClassB().getText();
  }
  @PreDestroy
  public void end() {
    System.out.println("ClassC exiting - text: " + text);
  }
}
```

(1) ClassA constructor - ClassB is: null
(2) classB Constructor text is: Value
(3) Setting ClassB
(4) MyAspect start method - text: Test
(5) Testing Spring Startup.
(6) ClassC constructor
(7) MyAspect says: fine, be that way
(8) ClassC getText Random
(9) MyAspect Says: Fine, be that way
(10) ClassA Returning ClassB
(11) MyAspect doesn't like ClassB.
(12) In Main: Random Something
(13) ClassC exiting - text: Random.

Name: Thanh Do Nguyen StudentID: 815941

All of the code exercises after this belong together. In essence you are going to make a simple Gemstone CRUD application. The package structure for this application is shown in the screenshot below – not all packages need to have classes.

```
@Data
@Entity
public class Gemstone {
  @Id
  @GeneratedValue
    (strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
  private double price;
  private String color;
  private double translucency;
  private int hardness;
  @Embedded
  private Cut cut;
}
```

```
@Data
@Embeddable
public class Cut {
  private String shape;
  @Temporal(TemporalType.DATE)
  private Date date;
}
```

Code Exercises:

2. [12 pts] Write an GemStoneDao, with finder methods to:
   1. Retrieve all gemstones with a specific name (for example "Diamond")
   2. Retrieve all gemstones whose translucency is greater than a given value
   3. Retrieve all gemstones where the price is between a given low and high

Note: You don't have to use all these methods in the GemstoneService

```
public interface GemStone Dao extends Jpa Repository <GemStone, Long> {

    public List <Gemstone> find By Name (String name);

    public List <GemStone> find By Translucency Greater Than (double translucency);

    public List <GemStone> find By Price Less Than Equal And Greater Than Equal
                                (double high , double low );

}
```

Price

Name: Thanh Do Nguyen StudentID: 615941

10

3. [10 pts] Create a service that uses the repository you made for the previous question. The easiest way to know what it should do is to first implement the controller on the next page. Be sure to add transactions.

```
public interface GemStoneService {
    void add (GemStoneDto dto);
    List<GemStone> findByName (String name);
    GemStone findById (Long id);
    Void update (GemStoneDto dto);
    Void delete (Long id);
}

@Service
@Transactional
public Class GemStoneServiceImpl implements GemStoneService {
    @Autowired
    private GemStone Dao gemstoneDao;
    @Override
    public void add (GemStoneDto dto) {
        GemStone gs = new GemStone();
        gs.setName (dto.getName());
        gs.setPrice (dto.getPrice());
        gs.setColor (dto.getColor());
        gs.setTranslucency (dto.getTranslucency())
        gs.setHardness (dto.getHardness());
        Cut cut = new Cut();
        cut.setShape (dto.getShape());
        cut.setDate (dto.getDate());
        gs.setCut (cut);
        gemstoneDao.Save (gs);
    }
```

Name: thanh Do Nguyen StudentID: 615941

4. [20 pts] Write a RestController class, plus any DTO classes you might need, so that you can correctly respond to the following requests

20

- `POST /gemstone` adds a gemstone, including cut
- `GET /gemstone/name/{name}` returns all gemstones with that name
- `GET /gemstone/{id}` returns the gemstone/cut with the given id
- `PUT /gemstone/{id}` updates the gemstone/cut with the given id
- `DELETE /gemstone/{id}` deletes the gemstone/cut with the given id

```
@RestController
@RequestMapping("/gemstone")
public class GemStoneController {
    @Autowired
    private GemStoneService gemstoneService;

    @PostMapping
    public String add(@RequestBody GemstoneDto dto){
        gemstoneService.add(dto);
        return "OK";
    }
    @GetMapping("/name/{name}")
    public List<Gemstone> FindByName(@PathVariable String name){
        return gemstoneService.findByName(name);
    }
    @GetMapping("/{id}")
    public Gemstone FindById(@PathVariable Long id){
        return gemstoneService.findById(id)
    }
    @PutMapping("/{id}")
    public String update(@PathVariable Long id, @RequestBody GemstoneDto dto){
        dto.setId(id);
        gemstoneService.update(dto);
        return "OK";
    }
    @DeleteMapping("/{id}")
    public String delete(@pathVariable :id){
        gemstoneService.delete(id);
        return "OK";
    }
```

Name: _____ StudentID: _____

5. [15 pts] Create a DiamondDiscount Aspect class with advice method that applies to when a gemstone is created (at the service level). If the gemstone name is Diamond then the price on the gemstone object is multiplied by 0.9 so that a lower price is stored in the database.

15

```
@Aspect
@Component
public class DiamondDiscountAspect {

    @Around("execution(* CS544.Service.GemstoneService.add(..))")
    public Object invoke(ProceedingJoinPoint pjp) throws Throwable {
        Object[] args = pjp.getArgs();
        GemstoneDto dto = (GemstoneDto) args[0];
        if(dto.getName().equals("Diamond")){
            dto.setPrice(dto.getPrice() * 0.9);
            args[0] = dto;
        }
        return pjp.proceed(args);
    }
```

Thanh Do Nguyen 615921

3) Service ....

```
@Override
public List<GemStone> findByName (String name) {
    return gemstoneDao.findByName(name);
}.

@Override
public GemStone findById (Long id) {
    return gemstoneDao.findById().get();
}.

@Override
public void update (GemStoneDto dto) {.
    GemStone gs = new Gemstone ();
    gs.setName (dto.getName());
    gs.setPrice (dto.getprice());
    gs.setColor (dto.getColor());
    gs.setTranslucency (dto.getTranslucency());
    gs.setHardness (dto.getHardness());
    Cut cut = new Cut();
    cut.setShape (dto.getShape());
    Cut.setDate (dto.getDate());
    gs.setCut (cut);
    gs.setId (dto.getId());
    gemstoneDao.save (gs);
}.

@Override
public void delete (Long id) {
    gemstoneDao.deleteById(id);
}.
```

```
@Data
public Class GemstoneDto {

    private Long id;
    private String name;
    private double price;
    private String color;
    private double translucency;
    private int hardness;
    private String shape;
    private Date date;

}.
```