Name: _____ StudentID:_____

Final Exam 2022-09

CS544 Enterprise Architecture

Theory Section

A. [3 pts] Explain what Domain Driven Design (DDD) is:

B. [3 pts] Explain what Component Scan is for the Spring Context:

C. [3 pts] Explain what the @Bean annotation does for Spring

D. [3 pts] Explain what is meant with Target in the context of AOP:

E. [3 pts] Give 3 types of method calls where Spring's Proxy Based Weaving doesn't work:

F. [3 pts] What does the @PathVariable annotation do in Spring MVC?

G. [3 pts] Explain why the order is important in the Spring Security config:

H. [3 pts] Explain what the use of ResponseEntity is in Spring MVC:

Name: _____ StudentID:_____

1. [20 pts] What is the output of the following application:

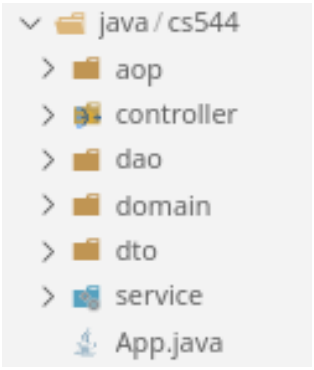
```
@Configuration
@ComponentScan("cs544")
@EnableAspectJAutoProxy
public class Config {
}
---------------------------------
public class App {
  public static void main(String[] args) {
    ConfigurableApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
    System.out.println("Testing Spring Startup");
    MyClass mc = context.getBean("myClass", MyClass.class);
    mc.sayHello();
    context.close();
  }
}
---------------------------------
public abstract class MySuper {
  @Value("From Super")
  private String text;

  public MySuper() { System.out.println("MySuper Constructor - text: " + getText()); }

  @PostConstruct
  public void init() { this.setText("From Super Init"); }

  public String getText() { return text; }
  public void setText(String text) {
    System.out.println("Setting Text to: " + text);
    this.text = text;
  }
}
---------------------------------
@Scope("prototype")
@Component
public class MyClass extends MySuper {

  public MyClass() { setText("From Class Constructor"); }

  public void sayHello() { System.out.println("Hello is: " + getText()); }

  @PreDestroy
  public void destroy() { System.out.println("Destroying MyClass"); }
}
---------------------------------
@Aspect
@Component
public class TraceAspect {
  @Autowired
  private MyClass myClass;

  @Before("execution(* cs544.*.*(..))")
  public void beforeTrace(JoinPoint jp) {
    System.out.println(jp.getSignature().getName() + " is about to execute");
    if (jp.getTarget() instanceof MyClass) {
      MyClass my = (MyClass)jp.getTarget();
      my.setText("From Advice");
    }
  }
}
```

Name: _____ StudentID:_____

All of the code exercises after this belong together. In essence you are going to make a simple Cookie Eating CRUD application (based on the Cookie domain from the Midterm). The package structure for this application is shown in the screenshot below – not all packages need to have classes.

```
@Data
@Entity
public class Child {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  private int age;
  @Embedded
  private Address address;
  @OneToMany
  private List<Eats> eats = new ArrayList<>();
}
@Data
@Embeddable
public class Address {
  private String city;
  private String country;
}
@Data
@Entity
public class Eats {
  @Id
  @GeneratedValue
  private Long id;
  @Temporal(TemporalType.DATE)
  private Date date;
  private String enjoyment;
  @ManyToOne
  @JsonManagedReference
  private Cookie cookie;
}
```

```
@Data
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Cookie {
  @Id
  @GeneratedValue
  private Long id;
  private double size;
  @OneToMany(mappedBy = "cookie")
  @JsonBackReference
  private List<Eats> eats = new ArrayList<>();
}
@Data
@Entity
public class ChocolateChip extends Cookie {
  private int numberOfChips;
}
@Data
@Entity
public class Shortbread extends Cookie {
  private String extraIngredient;
}
@Data
@Entity
public class Thumbprint extends Cookie {
  private String jamType;
}
```



Code Exercises:

2. [7 pts] Write an EatsDao, with methods for:
    1. All Eats with a given enjoyment
    2. All Eats for a given cookie id
    3. Write the code for any additional repositories here as well

3. [15 pts] Create a service that uses the repositories you make for the previous question. The easiest way to know what it should do is to first implement the controller on the next page.

Name: _____ StudentID:_____

4. [15 pts] Write a RestController class that can correctly respond to the following requests:
   - `GET /eats/{id}` returns the eats with that id
   - `GET /eats/cookie/{id}` returns list of eats for the given cookie id
   - `POST /eats/` Receives childId, cookieId, and enjoyment (no date, use current date)
   - `PUT /eats/{id}` Receives date and enjoyment to update by id
   - `DELETE /eats/{id}` wants eatsId as URL param

Name: _____ StudentID:_____

5. [10 pts] Create a ChocolateChipLover Aspect class with an advice method that detects when a new Eats object is created (hooks into the service method that receives the parameters to make the Eats object). If the CookieId is for a ChocolateChip cookie it should change the value of the enjoyment parameter passed to the method to “Great”