Midterm Review – ISTE 240

Question 1

Create a Spring Context Configuration class:

a. Add 2 beans of type string representing primary color and secondary color for an app.

b. Add a bean that maintains a list of products (i.e., global data store); each product has a name and a price.

c. Create a consumer class that grabs the list of products and returns the one with min/max price.

Here's how to create a configuration class:

```
1. // Product class definition
 2. class Product {
 3.     private String name;
 4.     private double price;
 5.
 6.     public Product(String name, double price) {
 7.         this.name = name;
 8.         this.price = price;
 9.     }
10.
11.     public String getName() {
12.         return name;
13.     }
14.
15.     public double getPrice() {
16.         return price;
17.     }
18. }
```

```
 1. // Consumer class that will find min/max priced products
 2. class ProductConsumer {
 3.     private List<Product> products;
 4.
 5.     public ProductConsumer(List<Product> products) {
 6.         this.products = products;
 7.     }
 8.

 9.     public Product getMinPricedProduct() {
10.         return products.stream()
11.                 .min(Comparator.comparing(Product::getPrice))
12.                 .orElse(null);
13.     }
14.
15.     public Product getMaxPricedProduct() {
16.         return products.stream()
17.                 .max(Comparator.comparing(Product::getPrice))
18.                 .orElse(null);
```

```java
@Configuration  // This annotation tells Spring this is a configuration class
public class AppConfig {

    // a. Add 2 beans of type string for colors
    @Bean
    public String primaryColor() {
        return "Blue";
    }

    @Bean
    public String secondaryColor() {
        return "Green";
    }

    // b. Add a bean that maintains a list of products
    @Bean
    public List<Product> productList() {
        List<Product> products = new ArrayList<>();
        products.add(new Product("Laptop", 999.99));
        products.add(new Product("Phone", 699.99));
        products.add(new Product("Headphones", 149.99));
        return products;
    }

    // c. Create a consumer class that can find min/max priced products
    @Bean
    public ProductConsumer productConsumer(List<Product> productList) {
        return new ProductConsumer(productList);
    }
}
```

Explanation:

- Spring Context is the central part of a Spring application that manages beans. A bean is simply an object that is managed by Spring.
- `@Configuration` marks this class as a source of bean definitions
- `@Bean` methods produce objects that Spring manages
- For the colors, we simply return String objects
- For products, we create a List containing Product objects
- For the consumer, we inject the product list using the parameter of the method
- The consumer has methods to find minimum and maximum priced products using Java streams

Question 2

Spring Context with stereotype annotation a. Given the following classes, create a spring context that maintains beans using stereotype annotations noting that any document must have a header and a footer.

class Document ()

class Header ()

class Footer ()

Stereotype annotations are a way to mark classes as components that Spring should automatically detect and register as beans. Common stereotype annotations include `@Component`, `@Service`, `@Repository`, and `@Controller`.

Here's how to implement this:

```
@Component
class Header {
    public String getHeaderContent() {
        return "This is the document header";
    }
}

@Component
class Footer {
    public String getFooterContent() {
        return "This is the document footer";
    }
}

@Component
class Document {
    private final Header header;
    private final Footer footer;

    // Constructor injection - Spring will automatically inject
    // the Header and Footer beans when creating the Document bean
    @Autowired
    public Document(Header header, Footer footer) {
        this.header = header;
        this.footer = footer;
    }

    public Header getHeader() {
        return header;
    }

    public Footer getFooter() {
        return footer;
    }

    public String getFullDocument() {
        return header.getHeaderContent() + "\nDocument content goes here\n" +
footer.getFooterContent();
    }
}

```

```
1. // This configuration enables component scanning
2. @Configuration
3. @ComponentScan  // This tells Spring to look for @Component classes in this package
4. public class DocumentConfig {
5.     // No explicit bean definitions needed here since we're using stereotype annotations
6. }
7.
```

Explanation:

- `@Component` marks classes that should be registered as Spring beans
- `@Autowired` tells Spring to inject dependencies (here we're using constructor injection)
- `@ComponentScan` tells Spring to search for @Component classes in the current package
- Each Document object requires a Header and Footer, and Spring automatically wires them together

Question 3

Spring Boot question: create a controller method to serve the user a dynamic view that indicates his/her age based on a given birth year, write the template for this view.

The controller class basically handles HTTP requests and responses in spring.

```java
@Controller  // Marks this as a web controller
public class AgeController {

    @GetMapping("/age/{birthYear}")  // Maps to URLs like /age/1990
    public String calculateAge(@PathVariable int birthYear, Model model) {
        // Calculate age based on birth year
        int currentYear = Year.now().getValue();  // Gets the current year
        int age = currentYear - birthYear;

        // Add calculated age to the model
        model.addAttribute("birthYear", birthYear);
        model.addAttribute("currentYear", currentYear);
        model.addAttribute("age", age);

        // Return the view name (Mustache template)
        return "age-result";
    }
}

```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Age Calculator</title>
</head>
<body>
    <h1>Age Calculator Results</h1>
    <p>Birth Year: {{birthYear}}</p>
    <p>Current Year: {{currentYear}}</p>
    <p>Your age is: {{age}} years old</p>
</body>
</html>

```

Explanation:

- `@Controller` marks this class as a web controller
- `@GetMapping` maps HTTP GET requests to this method
- `@PathVariable` extracts the birthYear from the URL
- `Model` is used to pass data to the view
- Mustache templates use `{{variableName}}` syntax to insert dynamic content
- The template is returned by the controller method and rendered with the provided model attributes

Some technical words explained:

- Bean: A Java object that is managed by the Spring container
- Dependency Injection: Spring's way of providing objects that a class needs (dependencies)
- @Configuration: Marks a class as a source of bean definitions
- @Component: A generic stereotype annotation that marks a class as a Spring component
- @Autowired: Tells Spring to automatically inject dependencies
- @Controller: A stereotype annotation that marks a class as a web controller
- @GetMapping: Maps HTTP GET requests to specific handler methods
- @PathVariable: Extracts values from the URL path
- Model: A container for data that will be passed to the view
- Mustache: A logic-less template engine that allows you to create dynamic HTML