Spring Boot & Java Interview Questions

# Q1. How to troubleshoot a React, Spring Boot, MySQL, GCP Cloud project?

To troubleshoot a full-stack project using React (frontend), Spring Boot (backend), MySQL (database), and deployed on Google Cloud Platform (GCP), follow a systematic approach:  
  
1. Frontend (React):  
 - Check browser console logs (F12 → Console).  
 - Verify API URLs in .env or config.  
 - Handle CORS issues (check network tab).  
 - Use Postman or curl to test backend separately.  
 - Rebuild (`npm run build`) and redeploy if needed.  
  
2. Backend (Spring Boot):  
 - Run locally with `mvn spring-boot:run` and test endpoints.  
 - Check logs locally or on GCP (Cloud Logging).  
 - Configure CORS if needed using WebMvcConfigurer.  
 - Check Cloud Run/App Engine logs if deployed on GCP.  
  
3. Database (MySQL):  
 - Verify JDBC URL, credentials, and connectivity.  
 - Use MySQL client or Cloud SQL Proxy for manual DB connection.  
 - Review GCP SQL logs and authorized networks.  
  
4. GCP Services:  
 - Check IAM roles and permissions.  
 - Review Cloud Monitoring and Logging for all components.  
 - Use Cloud Shell or Cloud SQL Proxy for advanced debugging.

# Q2. How to make a custom Annotation in Spring?

To create a custom annotation in Spring:  
  
1. Define the annotation:  
```java  
@Target({ElementType.METHOD, ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
public @interface MyCustomAnnotation {  
 String value() default "default";  
}  
```  
  
2. Create an aspect or handler to process the annotation:  
```java  
@Aspect  
@Component  
public class MyCustomAspect {  
 @Before("@annotation(myCustomAnnotation)")  
 public void before(JoinPoint joinPoint, MyCustomAnnotation myCustomAnnotation) {  
 System.out.println("Custom annotation used with value: " + myCustomAnnotation.value());  
 }  
}  
```

# Q3. How to handle custom exceptions in Spring Boot?

1. Define a custom exception:  
```java  
public class ResourceNotFoundException extends RuntimeException {  
 public ResourceNotFoundException(String message) {  
 super(message);  
 }  
}  
```  
  
2. Create a global exception handler using @ControllerAdvice:  
```java  
@ControllerAdvice  
public class GlobalExceptionHandler {  
  
 @ExceptionHandler(ResourceNotFoundException.class)  
 public ResponseEntity<String> handleResourceNotFound(ResourceNotFoundException ex) {  
 return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT\_FOUND);  
 }  
}  
```

# Q4. Generics in Java

Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces, and methods.  
  
Example - Generic Class:  
```java  
public class Box<T> {  
 private T value;  
 public void setValue(T value) { this.value = value; }  
 public T getValue() { return value; }  
}  
```  
  
Example - Generic Method:  
```java  
public <T> void printArray(T[] array) {  
 for (T element : array) {  
 System.out.println(element);  
 }  
}  
```  
  
Benefits:  
- Type safety  
- Code reusability  
- Compile-time checking