**1. Inverted Class in Java with Syntax**

Java does not have a concept called "Inverted Class." However, if you are referring to **Inversion of Control (IoC)**, it is a design principle used in frameworks like Spring.

**Example using Spring IoC:**

java

CopyEdit

import org.springframework.stereotype.Component;

@Component // Marks this class as a Spring-managed component

public class MyService {

public void display() {

System.out.println("Hello from MyService!");

}

}

Here, control of object creation is inverted and managed by Spring.

**2. Try-With-Resource Concept in Java 8**

Introduced in Java 7 and improved in Java 9, **try-with-resources** automatically closes resources (like files, sockets, and database connections) without needing a finally block.

**Example:**

java

CopyEdit

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class TryWithResourceExample {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

System.out.println(br.readLine()); // Reads the first line

} catch (IOException e) {

e.printStackTrace();

}

}

}

Here, the BufferedReader is automatically closed when the try block ends.

**3. Java 8 Features Used in a Project**

Some key **Java 8 features** commonly used in projects:

1. **Lambda Expressions** - For concise function expressions
2. **Streams API** - For functional data processing
3. **Default & Static Methods in Interfaces** - Helps in backward compatibility
4. **Functional Interfaces** - Predicate, Function, Consumer, etc.
5. **Optional Class** - Avoids NullPointerException
6. **Collectors API** - For reducing and grouping operations in Streams

**Example (Streams + Lambda):**

java

CopyEdit

import java.util.Arrays;

import java.util.List;

public class Java8Example {

public static void main(String[] args) {

List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

names.stream().filter(name -> name.startsWith("A")).forEach(System.out::println);

}

}

Output: Alice

**4. Annotations Used in Spring Boot (With Explanation)**

1. **@SpringBootApplication** - Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
2. **@RestController** - Marks a class as a RESTful controller.
3. **@RequestMapping** - Maps HTTP requests to handler methods.
4. **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping** - HTTP-specific mappings.
5. **@Autowired** - Injects dependencies automatically.
6. **@Component, @Service, @Repository** - Used for component scanning and dependency injection.
7. **@Transactional** - Manages database transactions.
8. **@Value** - Injects property values from application properties file.

**Example:**

java

CopyEdit

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloController {

@GetMapping("/hello")

public String sayHello(@RequestParam String name) {

return "Hello, " + name;

}

}

**5. @Service vs @Repository - Can We Interchange Them?**

Yes, technically **we can interchange @Service and @Repository**, but it is **not recommended**.

* @Service is meant for **business logic layers**.
* @Repository is meant for **DAO (Data Access Layer)** and provides **exception translation**.

**What Happens If We Interchange?**

* **Using @Service instead of @Repository:** The application will work, but exception translation (e.g., converting JDBC exceptions to Spring’s DataAccessException) will not happen.
* **Using @Repository instead of @Service:** It will still function, but it may create confusion because the annotation suggests it’s a DAO layer class.

**Example:**

java

CopyEdit

@Service

public class UserService {

public String getUser() {

return "User Data";

}

}

@Repository

public class UserRepository {

public void saveUser(String user) {

System.out.println("User saved: " + user);

}

}

While both can technically work interchangeably, **it’s best practice to use them appropriately** for better maintainability.

**what** is the disadvantage of microservice.

what is circuit breaking

Problem 1:

group Employees object by Age group ( 21 - 30, 31 - 40) and print

Problem 2:

If you have a Employee class with emp\_id, emp\_name, emp\_salary, emp\_dept

- Get the Average salary for each department - should return Map<String, Double>

- Get the max salary per department - should return Map<String, Long>

- Get the Employee information for each department - should return Map<String, List<Employee>>

- Sort the Employees by Salary - should return List<Employee>

Problem 3: what is the result of this program

class A {

public int hashCode() {

return 0;

}

public boolean equals(Object obj) {

return true;

}

}

public class Main {

public static void main(String[] args) {

Map<A, Integer> map = new HashMap<>();

A a1 = new A();

A a2 = new A();

A a3 = a2;

map.put(a1, 1);

System.out.println(map.size());

map.put(a2, 2);

System.out.println(map.size());

map.put(a3, 3);

System.out.println(map.size());

map.put(new A(), 4);

System.out.println(map.size());

System.out.println(map.get(a1));

System.out.println(map.get(a2));

System.out.println(map.get(a3));

System.out.println(map.get(new A()));

}

}