Resilience4j

Fault Tolerance – Circuit Breaker

Resilience4j

Alternate to Hystrix

Resilience4j

Resilience4j is a lightweight fault tolerance library designed for Java applications, and it integrates seamlessly with Spring Boot.

It provides various fault tolerance and stability patterns:

1. **Circuit Breaker:** Prevents repeated failures by stopping calls to a failing service.
2. **Retry:** Automatically retries failed operations.
3. **Rate Limiter:** Controls the rate of requests to a service.
4. **Bulkhead:** Limits the number of concurrent calls to a service.
5. **Time Limiter:** Sets a time limit for operations.

Fault tolerance

Fault tolerance refers to the ability of a system to continue operating properly in the event of the failure of some of its components.

It ensures that the system can handle errors gracefully and maintain functionality without complete failure.

Techniques to achieve fault tolerance include:

- **Redundancy:** Having backup components that can take over in case of failure.
- **Graceful Degradation:** Reducing functionality in a controlled manner when parts of the system fail.
- **Retry Mechanisms:** Automatically retrying failed operations.
- **Fallbacks:** Providing alternative responses or services when the primary service fails.

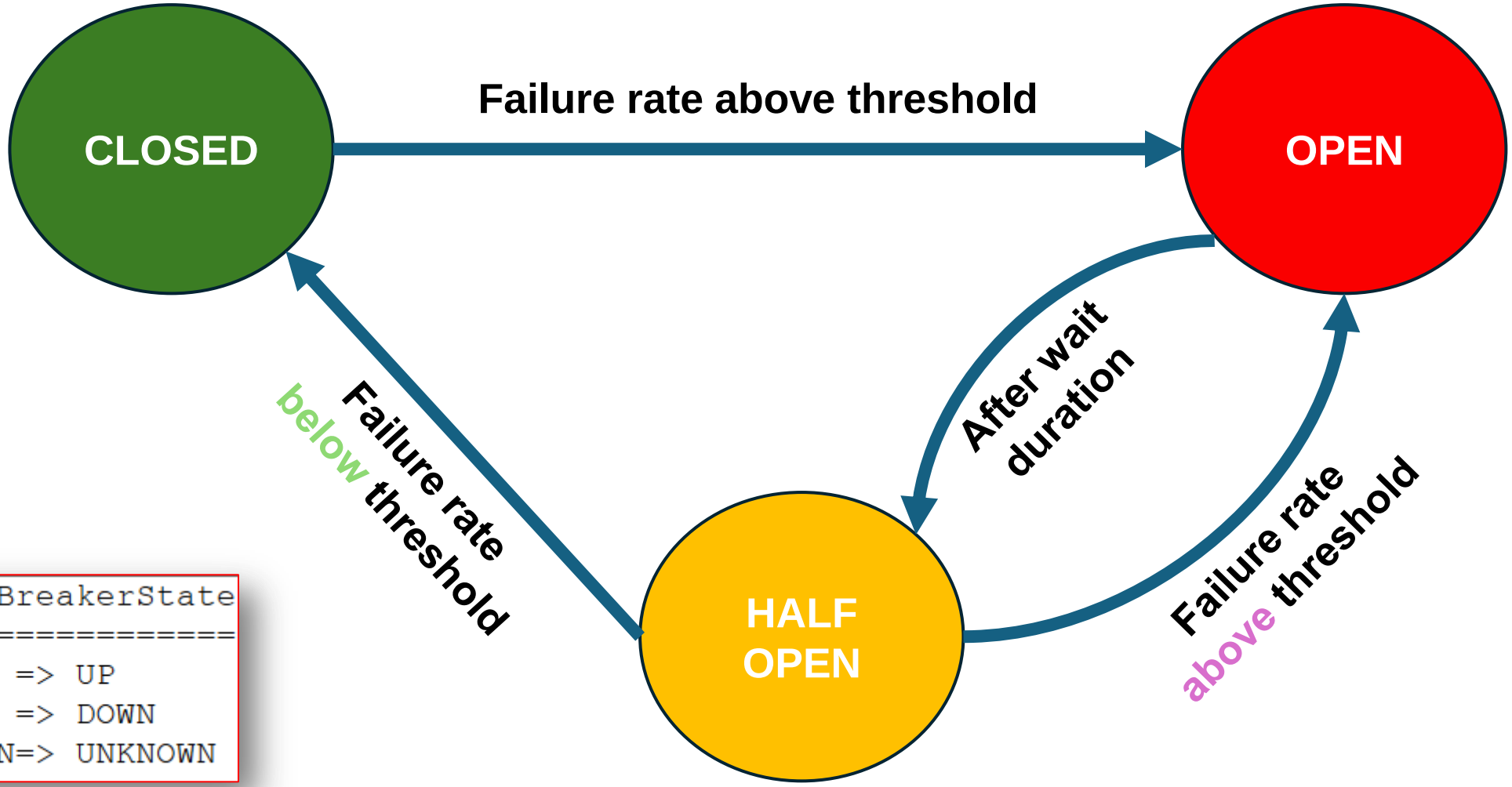
Circuit Breaker

The Circuit Breaker pattern is a design pattern used to detect failures and encapsulate the logic of preventing a failure from constantly recurring, which can help maintain the stability of the system.

Here’s how it works:

- **Closed State**: The circuit breaker allows requests to pass through to the service.
- **Open State**: If the service fails repeatedly, the circuit breaker opens, and all requests are immediately failed without attempting to call the service.
- **Half-Open State**: After a timeout period, the circuit breaker allows a limited number of test requests to pass through. If these requests succeed, the circuit breaker closes again; if they fail, it remains open.

Resilience4j Circuit Breaker




```
CircuitBreakerState
====================
CLOSED   => UP
OPEN     => DOWN
HALFOPEN=> UNKNOWN
```

Add dependencies (Student Service)

```
        <java.version>17</java.version>
        <spring-cloud.version>2023.0.3</spring-cloud.version>
    </properties>
    <dependencies>   Add Spring Boot Starters...
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
```

Properties File

application.properties

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
eureka.instance.hostname=localhost

#CircuitBreaker
resilience4j.circuitbreaker.instances.courseService.sliding-window-size=10
resilience4j.circuitbreaker.instances.courseService.failure-rate-threshold=50
resilience4j.circuitbreaker.instances.courseService.wait-duration-in-open-state=30000
resilience4j.circuitbreaker.instances.courseService.automatic-transition-from-open-to-half-open-enabled=true
resilience4j.circuitbreaker.instances.courseService.permitted-number-of-calls-in-half-open-state=5

#CircuitBreaker & Actuator
resilience4j.circuitbreaker.instances.courseService.allow-health-indicator-to-fail=true
resilience4j.circuitbreaker.instances.courseService.register-health-indicator=true

#Actuator
management.health.circuitbreakers.enabled=true
management.endpoints.web.exposure.include=health
management.endpoint.health.show-details=always

```

Resilience4j properties

resilience4j.circuitbreaker.instances.courseService

- (number of last calls to be checked)
- sliding-window-size=10
- (If exceeds 50% threshold, move to open state - DOWN)
- failure-rate-threshold=50
- (wait in milliseconds)
- wait-duration-in-open-state=30000
- (moves to half open - UNKNOWN)
- automatic-transition-from-open-to-half-open-enabled=true
- (check with 5 calls and change to UP or Down – on Threshold)
- permitted-number-of-calls-in-half-open-state=5
- (if the health check for the courseService fails,
- Not to stop the circuit breaker)
- allow-health-indicator-to-fail=true
- Registration of a health indicator for the courseService.
- register-health-indicator=true

Actuator properties

- (Enables the health indicators for all circuit breakers)
- management.health.circuitbreakers.enabled=true
- (Health status will be accessible through a web endpoint)
- management.endpoints.web.exposure.include=health
- (Ensures that detailed health information is always shown)
- management.endpoint.health.show-details=always

Steps to implement CircuitBreaker

- Create a new class CommonService in the service package
- Add @Service annotation
- Copy & Paste the FeignClient Method from studentService to this class
- Add @CircuitBreaker annotation with name same as the service name used in the properties file
- Create a fallback method, must have the same signature of FeignClient getCourseById signature
- Autowire commonService in StudentService and update both the methods

Create a Common Service class with
copied feignclient method and a fallback method

CommonService.java | StudentService.java

```java
@Service
public class CommonService {

    @Autowired
    private CourseFeignClient courseFeignClient;

    @CircuitBreaker(name="courseService", fallbackMethod="fallbackGetCourseById")
    public CourseResponse getCourseById(long courseId) {
        return courseFeignClient.getById(courseId);
    }

    public CourseResponse fallbackGetCourseById(long courseId, Throwable th) {
        return new CourseResponse();
    }
}
```

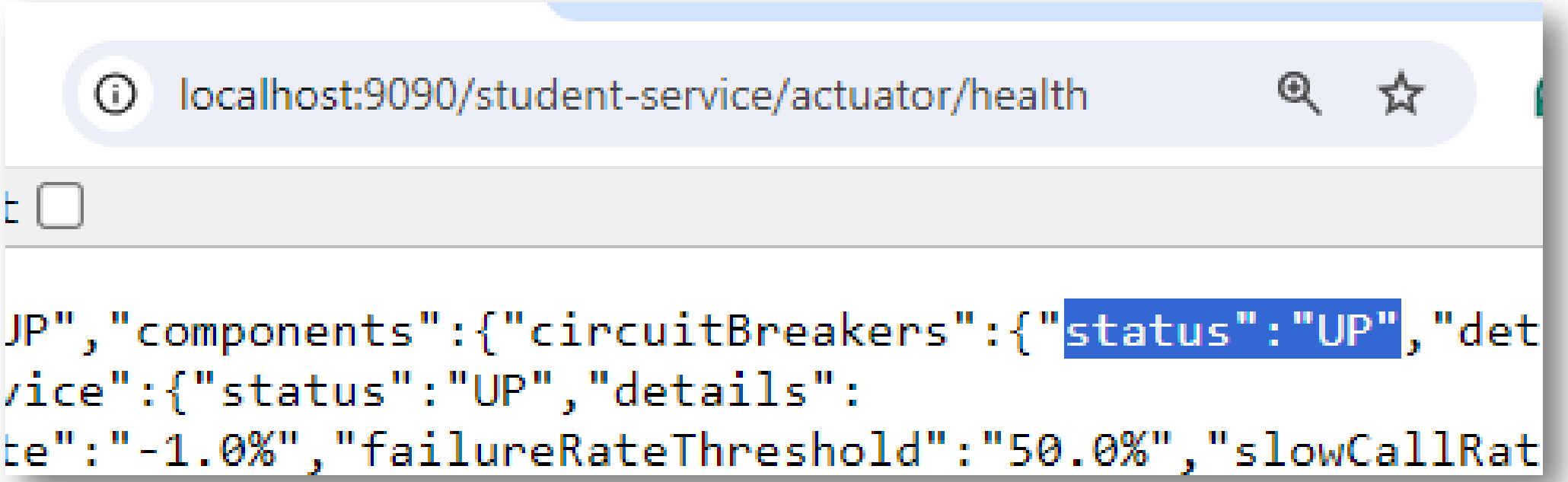
```java
@Autowired
private CommonService commonService ;

public StudentResponse createStudent(StudentRequest studentRequest) {
    Student student = new Student();
    student.setFirstName(studentRequest.getFirstName());
    student.setLastName(studentRequest.getLastName());
    student.setEmail(studentRequest.getEmail());
    student.setCourseId(studentRequest.getCourseId());
    studentRepository.save(student);

    StudentResponse studentResponse = new StudentResponse(student);
    studentResponse.setCourse(commonService.getCourseById(student.getCourseId()));
    return studentResponse;
}

public StudentResponse getById(long id) {
    logger.info("In getById : "+id);
    Student student = studentRepository.findById(id).get();

    StudentResponse studentResponse = new StudentResponse(student);
    studentResponse.setCourse(commonService.getCourseById(student.getCourseId()));
    return studentResponse;
}
```

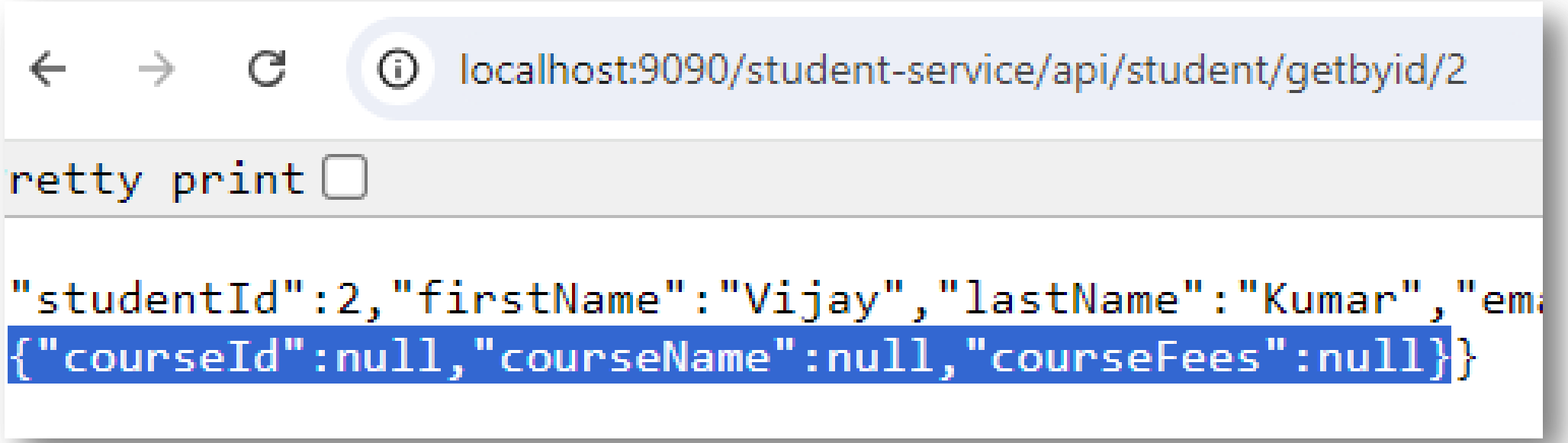
Add logger to test

```
17      private CourseFeignClient courseFeignClient;
18
19      Logger logger = LoggerFactory.getLogger(CourseFeignClient.class);
20      long count = 1;
21
22      @CircuitBreaker(name="courseService", fallbackMethod="fallbackGetCourseById")
23      public CourseResponse getCourseById(long courseId) {
24          logger.info("Count = "+count++);
25          return courseFeignClient.getById(courseId);
26      }
27
28      public CourseResponse fallbackGetCourseById(long courseId, Throwable th) {
29          logger.info("Error : "+th);
30          return new CourseResponse();
31      }
32  }
```

Steps to execution

- Eureka Server
- API Gateway
- Student Service
- (Now not Course Service for testing)
- Open a browser to check the api response and actuator health details


localhost:9090/student-service/actuator/health
t
JP","components":{"circuitBreakers":{"status":"UP","det
/ice":{"status":"UP","details":
te":"-1.0%","failureRateThreshold":"50.0%","slowCallRat


localhost:9090/student-service/api/student/getbyid/2
retty print
"studentId":2,"firstName":"Vijay","lastName":"Kumar","em
{"courseId":null,"courseName":null,"courseFees":null}}

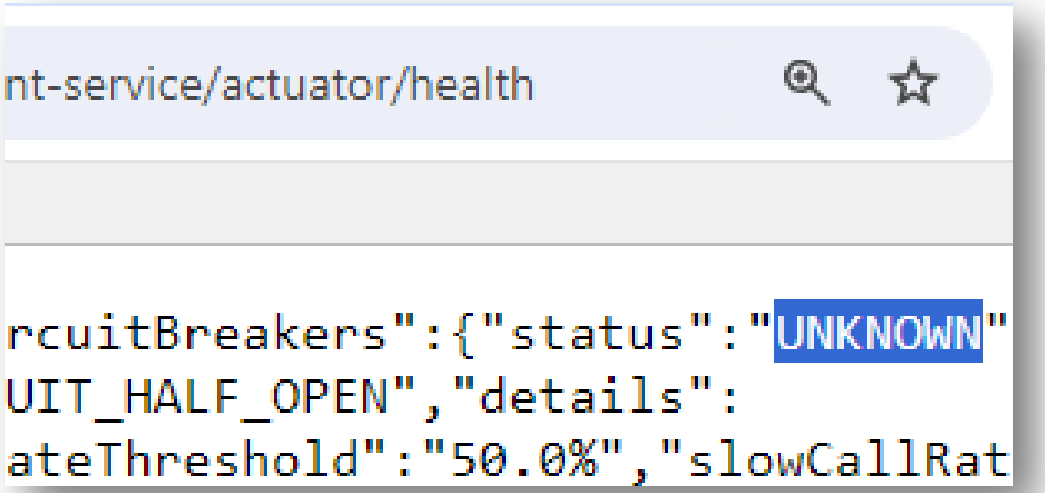
Call student api for 10 times, it fails with state changed to Down

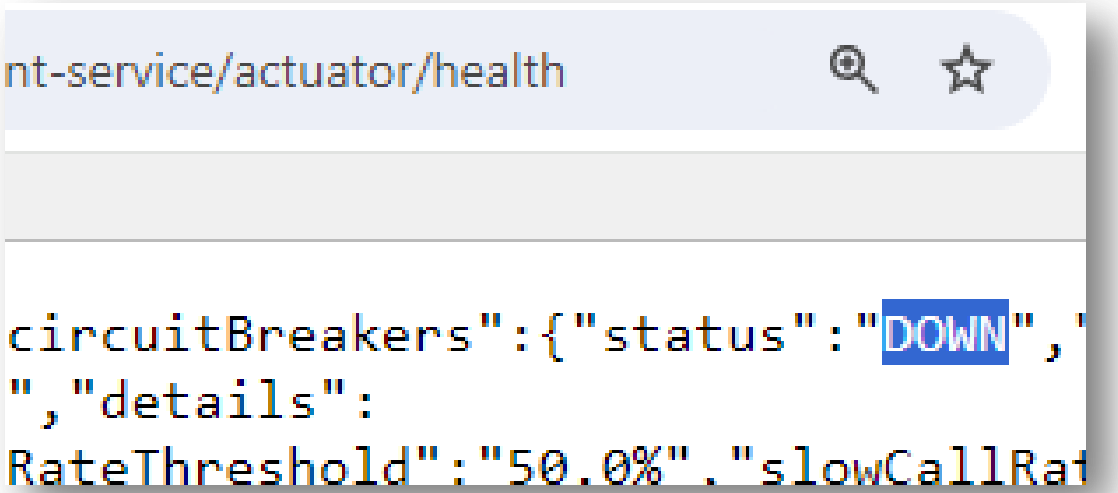
```
tService            : In getById : 2
s1_0 where s1_0.student_id=?
ourseFeignClient    : Count = 9
ourseFeignClient    : Error : feign.FeignException$NotF
tService            : In getById : 2
s1_0 where s1_0.student_id=?
ourseFeignClient    : Count = 10
ourseFeignClient    : Error : feign.FeignException$NotF
```

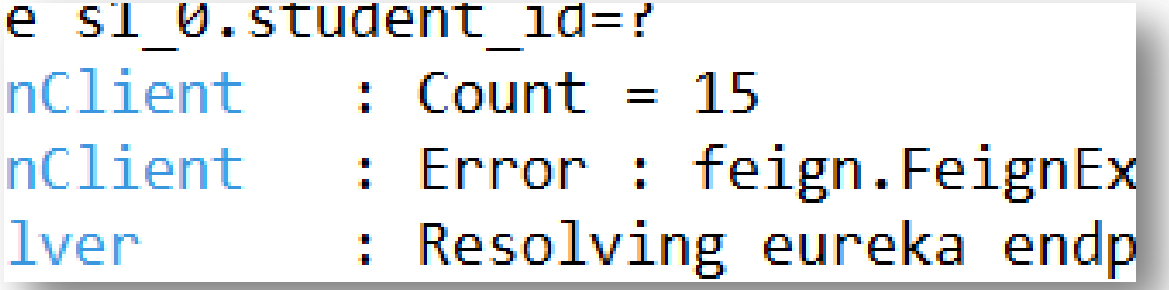
localhost:9090/student-service/actuator/health


```
N","components":{"circuitBreakers":{"status":"DOWN","
e":{"status":"DOWN","details":
:"100.0%","failureRateThreshold":"50.0%","slowCallRat
```

After 30 seconds, it turns to Unknown and again 5 more requests take it back to Down







Now, lets start the course service and hit 5 more student reque