Resilience4j also provides a **RateLimiter** module, which is used to **control the number of calls allowed in a certain time period** — protecting your service (or downstream service) from being overwhelmed.

Below is a **complete example** using Spring Boot + Resilience4j for a service named currencyExchangeService.

**🧰 1️⃣ Add Dependency**

If you haven't added it yet:

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot2</artifactId>

</dependency>

For Spring Boot 3.x, use resilience4j-spring-boot3.

**⚙️ 2️⃣ application.yml Configuration**

resilience4j:

ratelimiter:

instances:

currencyExchangeService:

limitForPeriod: 5 # Number of calls allowed per refresh period

limitRefreshPeriod: 10s # How often the limit is refreshed

timeoutDuration: 2s # How long to wait for permission

registerHealthIndicator: true # Show in /actuator/health

**📝 Explanation:**

| **Property** | **Description** |
| --- | --- |
| limitForPeriod | Max number of calls allowed during one limitRefreshPeriod. |
| limitRefreshPeriod | Duration after which the limit counter is reset. |
| timeoutDuration | How long a thread waits to get permission when limit is exceeded. |
| registerHealthIndicator | Optional — lets you monitor via /actuator/health. |

👉 Example: With limitForPeriod: 5 and limitRefreshPeriod: 10s, only **5 calls are allowed every 10 seconds**. If a 6th call comes, it will either wait up to timeoutDuration or fail immediately.

**🧠 3️⃣ Annotate the Method**

In your controller or service where you make the external call:

import io.github.resilience4j.ratelimiter.annotation.RateLimiter;

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

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

@RestController

public class CurrencyConversionController {

@GetMapping("/convert")

@RateLimiter(name = "currencyExchangeService", fallbackMethod = "rateLimiterFallback")

public String convertCurrency() {

// Simulate actual call to currency-exchange service

return "Currency conversion successful at " + System.currentTimeMillis();

}

public String rateLimiterFallback(Throwable t) {

return "Too many requests - please try again later";

}

}

**🧪 4️⃣ Test**

* Hit the /convert endpoint **more than 5 times within 10 seconds**.
* First 5 will succeed ✅
* 6th and above will return the **fallback message** ❌

**🌿 Optional: Combine with CircuitBreaker or Retry**

You can combine annotations too:

@CircuitBreaker(name = "currencyExchangeService", fallbackMethod = "fallback")

@RateLimiter(name = "currencyExchangeService")

@GetMapping("/convert")

public String convertCurrency() {

// ...

}

👉 Resilience4j will apply **both** behaviors: first check rate limiting, then circuit breaker logic.

**📊 Monitor via Actuator**

If you have spring-boot-starter-actuator, you can check:

GET http://localhost:8080/actuator/ratelimiters

GET http://localhost:8080/actuator/ratelimiter-events

**✅ Summary**

| **Feature** | **Purpose** |
| --- | --- |
| **RateLimiter** | Controls **traffic volume** (e.g., 5 requests / 10s) |
| **CircuitBreaker** | Handles **fault tolerance** when service is failing |
| **Retry** | Retries failed calls intelligently |

Would you like me to show a **RateLimiter + CircuitBreaker** **together with RestTemplate or Feign client** example (as in microservices)? 🧠⚡

Top of Form



Bottom of Form