**What is Rate Limiter:**

A Rate Limiter limits the number of client requests allowed to be sent over a specified period. If the API request count exceeds the threshold defined by the rate limiter, all the excess calls are blocked.

**A “rate limit”** is applied to ensure the API can provide optimal quality of service for its users, while also ensuring the safety of the API’s users.

For example, rate limiting can protect the API from slow performance when too many bots are accessing the API for malicious purposes, or when a [DDoS attack](https://datadome.co/learning-center/ddos-layer-7-security-protection/) is currently affecting the API. Also, when too many legitimate users are accessing the API, a rate limit can be useful.

**Any request which will be declined will return http code 429.**

**Why Rate Limiter:**

In DDoS attack, attacker sends unwanted requests to a server and each server has limited resources (RAM, Disk space). Attacker sends 1000s or lakhs of requests simultaneously so all of resources get consumed and brings server down. So, when genuine user sends request, its request gets declined. To prevent this DDoS attack, Rate limiter is needed.

**Advantages of implementing API rate limiting:**

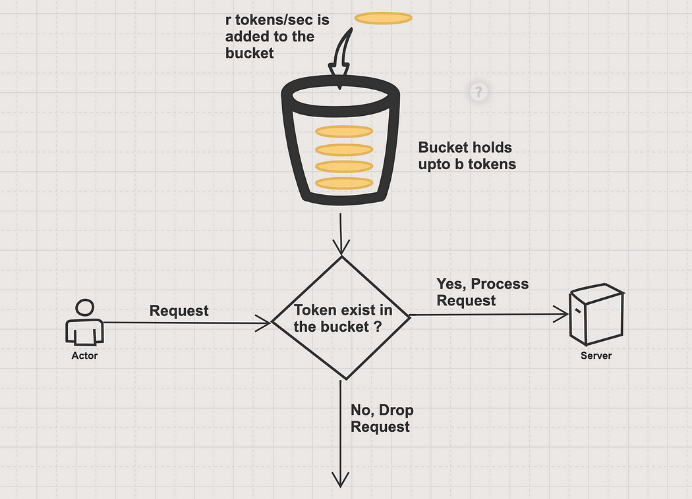
* **Protecting resource usage:** All APIs operate on finite resources, and rate limiting is essential to improve the availability of API service for as many users as possible by avoiding excessive resource usages. While resource starvation can be caused by attackers via DDoS attacks, there are actually many DoS incidents that are caused by errors in software rather than outside attacks.
* **Controlling Data Flow: This is especially important in APIs that process and transmit large volumes of data. Rate limiting can be implemented to control data flow, for example by merging many data streams into a single service.**

### Maximizing Cost-Efficiency (minimizing cost on resources consumption): Rate limiting can be implemented to control cost, for example, to prevent using too many resources, which may accumulate large costs. Any resource consumed will always generate a cost, and the more requests an API gets, the more costs it will accumulate. Rate limiting can be extremely important to ensure the profitability of the API.

### Controlling Quotas Between Users: When the capacity of an API’s service is shared among many users, rate limiting can (and should) be applied to individual users’ usage to ensure fair use without disrupting other users’ access. We can do this by applying the rate limit over a certain time period (i.e. per day) or by limiting the resource’s quantity when it’s possible. These allocation limits are often referred to as quotas.

**Algorithms to design Rate Limiter:**

1. **Token Bucket:** The token bucket algorithm works by allocating a fixed number of tokens to each user, which they can use to make requests. As tokens are used up, they are gradually replenished over time. If a user runs out of tokens, they must wait for more tokens to become available before they can make additional requests. This approach ensures that high-traffic users are limited in the number of requests they can make, while low-traffic users can still access the system without any restrictions



**Algorithm/Code:**

****

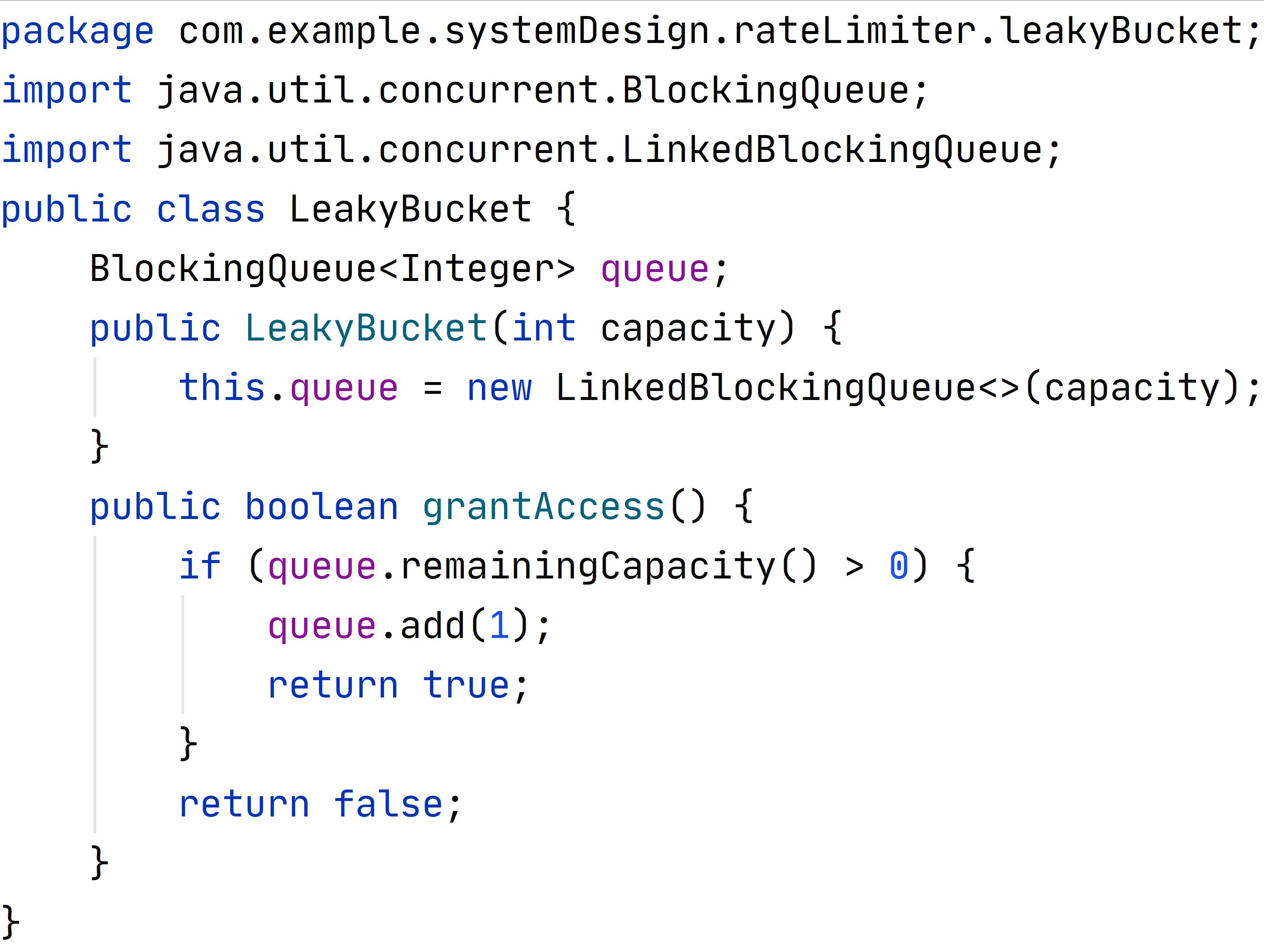
**Advantages:**  
1. The algorithm is easy to implement and memory efficient.  
2. Token bucket allows a burst of traffic for short periods. As long as there are tokens left, a request can be processed.

**Disadvantages:**  
1. It requires the tuning of two parameters, which may be difficult to achieve.  
*• Bucket size: the maximum number of tokens allowed in the bucket*  
*• Refill rate: number of tokens put into the bucket every second.*

1. **Leaky Bucket:** This algorithm uses queue as data structure for implementation. The bucket has a maximum capacity, which limits the burstiness of the requests. When the bucket is full, any incoming request is rejected or queued until there is space in the bucket. The requests are processed and sent to the API at a fixed rate, equal to the leakage rate of the bucket. The leaky bucket algorithm ensures that the requests are evenly distributed over time, regardless of how fast they arrive.



**Algorithm/Code:**





**Advantages:**

* The main benefit of the leaky bucket algorithm is that it provides fairness and stability to the API.

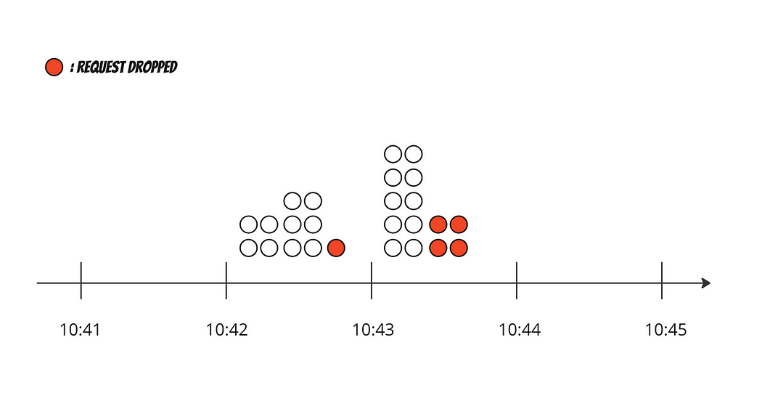
The algorithm prevents any client from dominating or flooding the API with requests, and ensures that all clients have an equal chance of accessing the API.

* The algorithm also smooths out the spikes and fluctuations of the incoming requests, and creates a predictable and consistent rate of requests. This can improve the performance and reliability of the API, and reduce the risk of overloading or crashing.
* Leaky Bucket is a more robust algorithm compared to a token bucket. It is also memory efficient because it uses a fixed-sized bucket

**Disadvantages:**

* The main drawback of the leaky bucket algorithm is that it can be inefficient and unresponsive to the client. The algorithm imposes a fixed rate of requests, regardless of the actual demand and traffic patterns. This means that the client cannot use the available bandwidth optimally, and may waste resources or miss opportunities. The client also cannot make a burst of requests when needed, and may experience delays or rejections due to the queueing effect of the bucket. The leaky bucket algorithm is also more complex and harder to implement and tune than the token bucket algorithm.

1. **Fixed Window Counter:** During the fixed time range, the rate limiter processes the request by increasing the counter. If the counter reaches the maximum value, further requests are dropped until the counter is reset by the new time window.



**Algorithm/Code:**

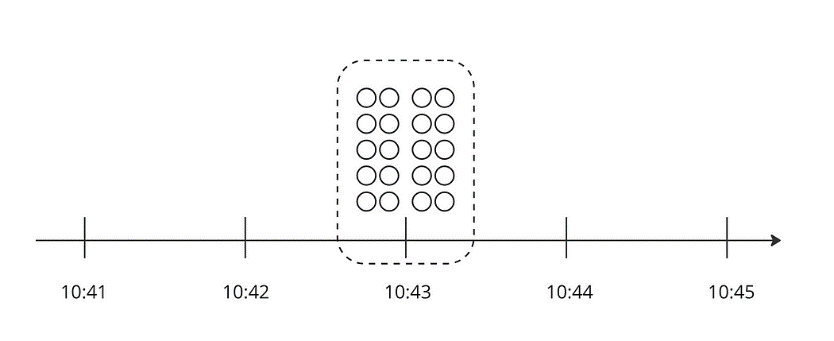
****

**Advantages:**

The advantage of this rate-limiting algorithm is that it lets you control the amount of data transferred per request.

Disadvantages:

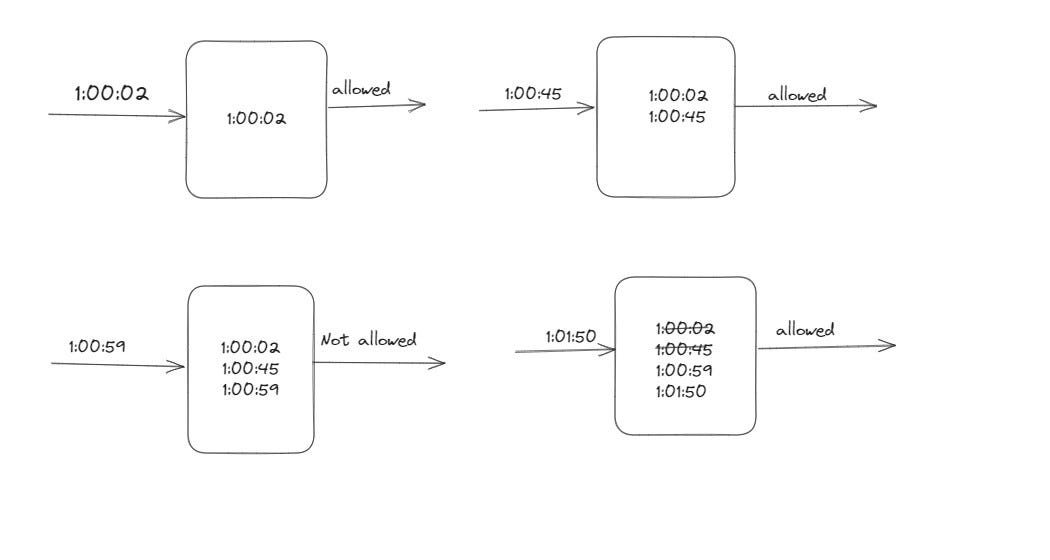
The disadvantage of this rate-limiting algorithm is that it’s hard to understand and implement, the window needs to be of the right size, and it doesn’t guarantee that all the requests will be serviced in a certain period of time. The fixed window rate limiting algorithm is used to control the amount of data per request.



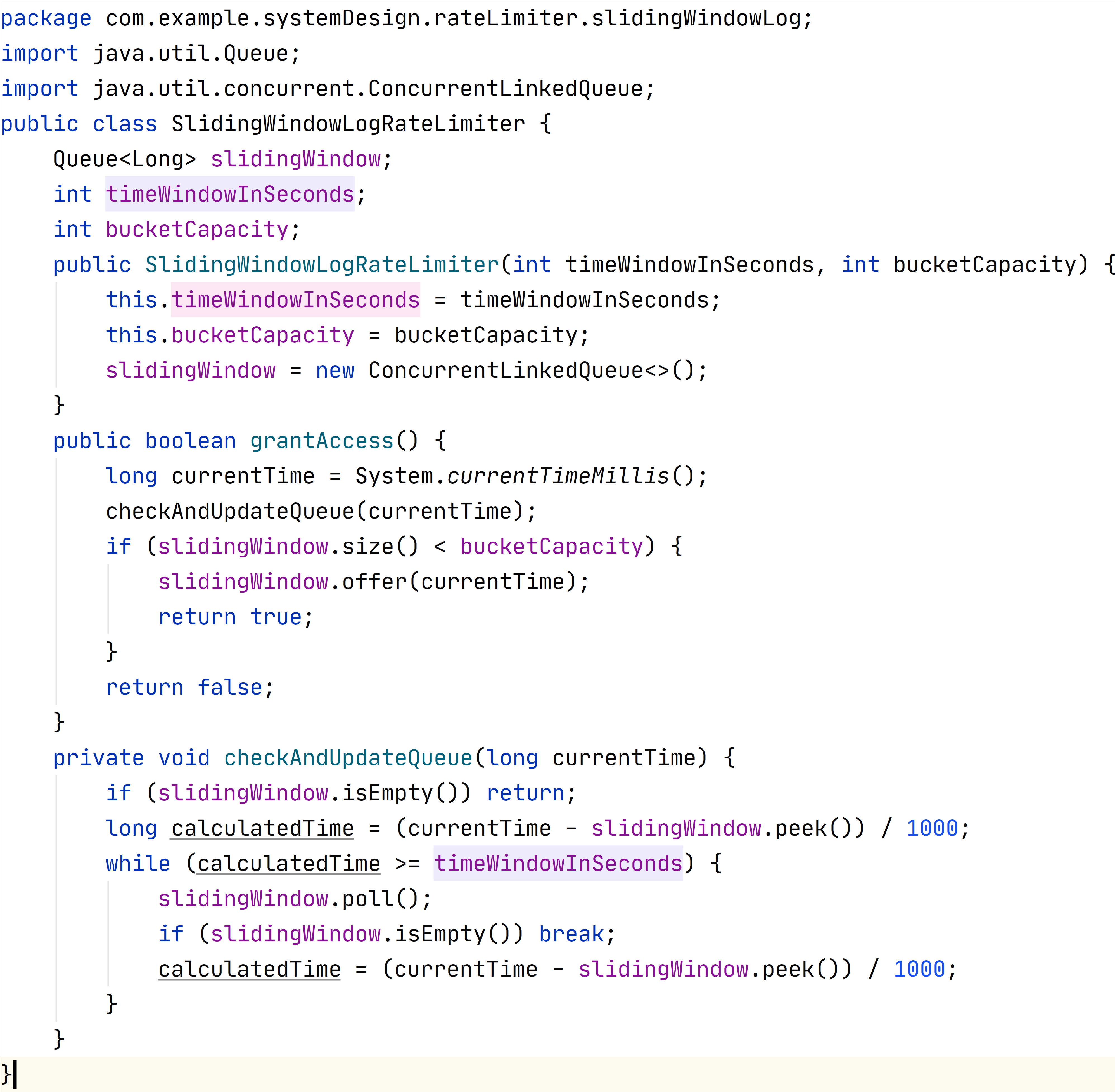
1. **Sliding Window Log:**

* The algorithm keeps track of request timestamps. Timestamp data is kept in cache, for faster retrieval.
* When a new request comes in, remove all the outdated timestamps. Outdated timestamps are those older than the start of the current time window.
* Add the timestamp of the new request to the log.
* If the log size is the same or lower than the allowed count, a request is accepted. Otherwise, it is dropped.

**In below example, rate limiter allows 2 requests per minute:**



**Algorithm/Code:**

****

****

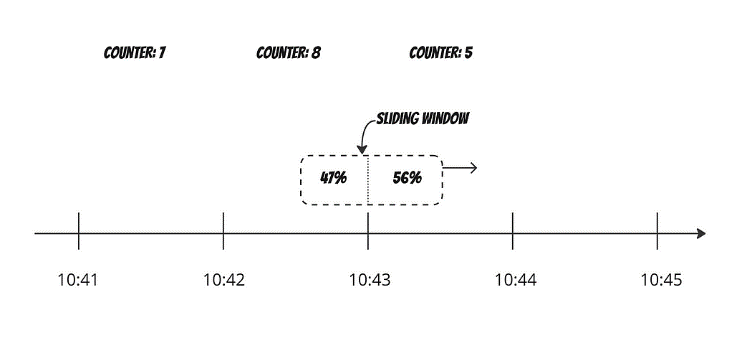
**Advantages:**

Rate limiting implemented by this algorithm is very accurate. In any rolling window, requests will not exceed the rate limit.

**Disadvantages:**

The algorithm consumes a lot of memory because even if a request is dropped, its timestamp might still be stored in memory.

1. **Sliding Window Counter:** **Sliding Window Counter is a hybrid solution that combines Fixed Window Counter and Sliding Window Log.** Fixed Window Counter is not a robust rate-limiting solution because it can’t correctly handle burst requests. But It is a memory-efficient solution. Sliding Window Log is a powerful solution since it enforces a hard limit on every time window. However, It isn’t memory efficient solution. **Sliding Window Counter tries to achieve efficient memory usage and a robust rate-limiting solution**



**Algorithm/Code:**

****

**Advantages:**

Sliding Window Counter is a robust and memory-efficient rate-limiting algorithm

**Disadvantages:**

Since it estimates the current sliding window counter by portion, It is not strictly correct estimation

* **These algorithms works well in single service but when it comes to distributed environment it fails due to data inconsistency and race conditions,**
* **We can solve the problem using locks, whenever any user is accesing data from any of the app server we will lock that data and release it once the updattion is done so ther app server will get the updated data.**

**Design Rate Limiter:**

