Proxy Design Pattern

Affan Rauf

December 18, 2024

Introduction to Proxy Design Pattern

- The Proxy Design Pattern provides a surrogate or placeholder for another object to control access to it.
- Useful when direct access to an object should be controlled or managed.
- Examples of Proxy usage:
  - Remote Proxy: To access an object in a different address space.
  - Virtual Proxy: To delay expensive object creation until needed.
  - Protection Proxy: To restrict access to sensitive operations.

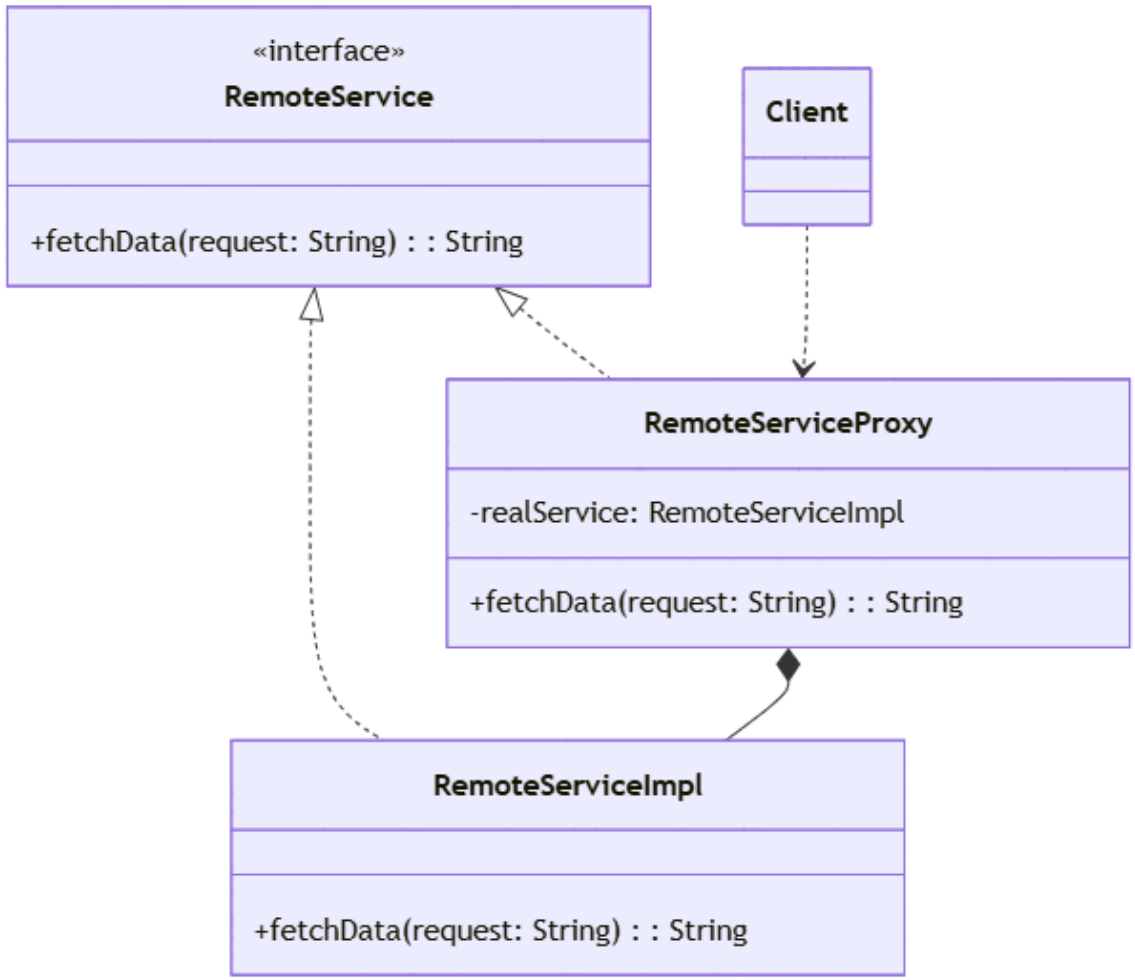
Structure of the Proxy Pattern

- The Proxy pattern involves:
  - **Interface**: Defines the methods that can be called on the object.
  - **Proxy Class**: Implements the interface and controls access to the real object.
  - **Concrete Class**: The actual object that the proxy represents.
- The proxy forwards requests to the concrete class as needed.

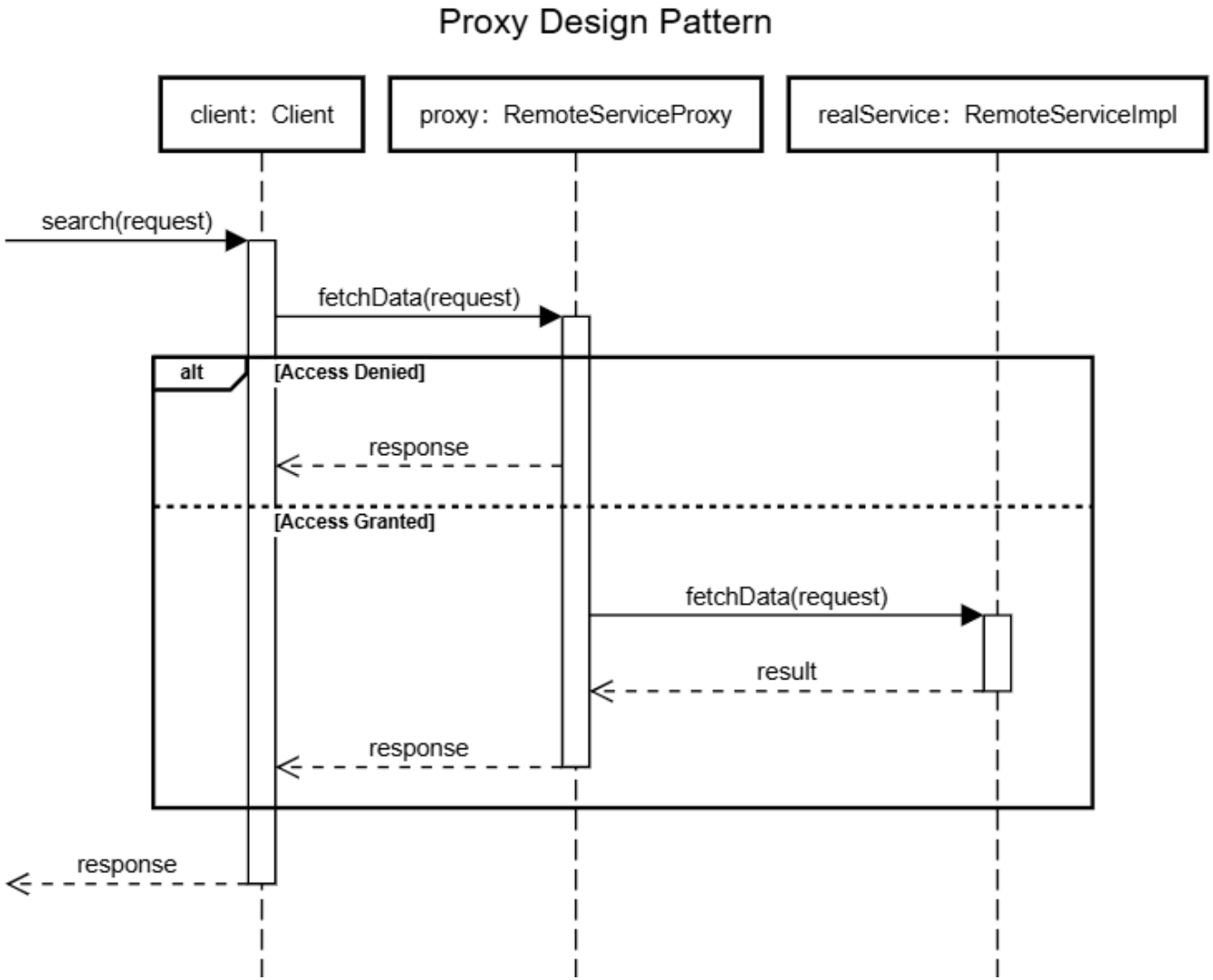
Roles in the Proxy Pattern

- **Interface**:
  - Provides a common type for the proxy and the concrete class.
  - Ensures the client interacts with a consistent API, unaware of whether it's working with the proxy or the real object.
- **Proxy Class**:
  - Holds a reference to the real object.
  - Controls access to the real object, adding additional behavior (e.g., security checks, logging).
- **Concrete Class**:
  - Implements the interface directly.
  - Represents the real object that performs the actual operations.

Class Diagram of Proxy Pattern



Sequence Diagram of Proxy Pattern



Example Code: Proxy Pattern in Java

Interface

```
public interface RemoteService {
    String fetchData(String request);
}
```

Concrete Class

```
public class RemoteServiceImpl implements RemoteService {
    public String fetchData(String request) {
        return "Data␣for␣" + request;
    }
}
```

Example Code (continued): Proxy Pattern in Java

Proxy Class

```
public class RemoteServiceProxy implements RemoteService {
    private RemoteServiceImpl realService;

    public RemoteServiceProxy() {
        this.realService = new RemoteServiceImpl();
    }

    public String fetchData(String request) {
        System.out.println("Logging request: " + request);
        return realService.fetchData(request);
    }
}
```

Benefits of the Proxy Pattern

- Allows controlled access to the real object.
- Can add extra functionality without modifying the real object's code.
- Useful in scenarios involving:
  - Security and access control.
  - Remote communication.
  - Lazy initialization.