Chain of Responsibility

Introduction

- Allows a request to be sent implicitly through a chain of candidate objects
- Any candidate may fulfill the request depending on the conditions set at run-time
- The number of candidates is open-ended
- Candidate objects working together can often handle a request that could not be handled by individual objects

Solution

- The client issues the request to the first object in the chain. This object receives the request and either handles it or forwards it to the next candidate on the chain (successor), which does likewise
- Each object on the chain shares a common interface for handling requests and for accessing its successor on the chain

Problem Context

- A client may send a request and not know which object will be able to handle that request
- Need to find a method to separate the initiator of the request from objects that might provide the solution

Solution Context

- More than one object can handle a request, and the handler is not known in advance
- Desire to issue a request to one of several objects without overtly specifying the receiver
- The set of objects that can handle the request should be specified at run-time

GoF Pattern Types

- **Creation**al — Concerned with the process of object creation
- **Structural** — Concerned with the composition of objects or classes
- **Behavior**al — Concerned with the communication between objects and the distribution of responsibility
### Chain of Responsibility

#### Structure

- **Client** — Initiates a request
- **Candidate Object** — Handles request if able to, otherwise forwards request to a successor
- **Handler** — Defines an interface for handling requests

#### Advantages/Disadvantages

- **Reduced Coupling:**
  - Frees an object from knowing the final handler of request (an object only knows that the request was handled appropriately).
  - Objects in the chain can be unaware of the chain’s structure.
  - Objects keep a single reference to their successor instead of maintaining references to all candidate receivers.

- **Added Flexibility**
  - Allows objects to share responsibility.
  - Responsibilities for handling a request can be modified at run-time by changing the chain.

- **No Guarantee**
  - The request can fall off the end of the chain without having been handled because it has no predetermined receiver.

#### Implementation

```java
public class It_Is_All_Greek_To_Me
{
    public boolean canTranslate()
    {
        //can the student read Greek?
        return true or false;
    }

    public void setSuccessor()
    { Set successor when each candidate object is created }

    public void forwardRequest()
    { if the object has a successor, then forward the request to that successor and call forwardRequest() }
}
```

#### Implementation (cont.)

Each subclass will implement these methods:

```java
public class Student extends It_Is_All_Greek_To_Me
{
    public void setSuccessor()
    { Set successor (4 front…); }

    public void readGreek()
    { if (canTranslated())
        { readGreek(); }
    else
        forwardRequest(); //forward to successor
    }
}
```