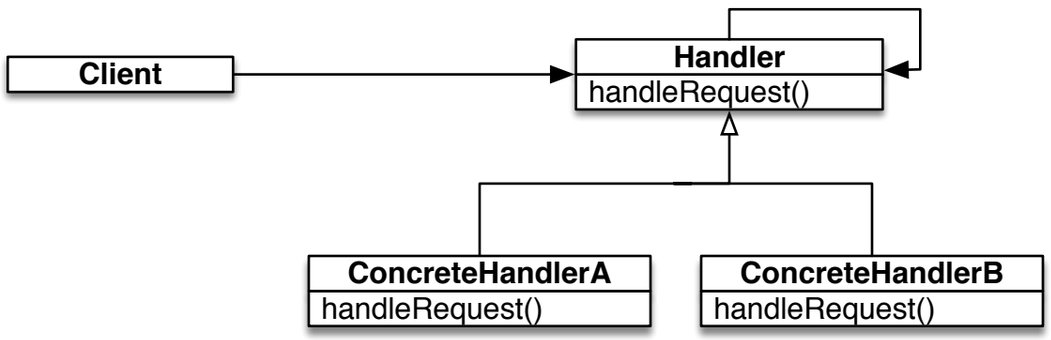
## Chain of Responsibility

**Intent**: avoid coupling the sender of a request to its receiver by giving more than one object to handle the request.
Chain the receiving objects and pass the request along the chain until an object handles it.
The object that made the request has no explicit knowledge of who will handle it (request has an implicit receiver)

For example, consider a show_tooltip function where whether onmouseover of a button shows button-specific tooltip, or dialog-specific toolkit is not known to the button being moused over.




Class Diagram


Object Structure

Chain of responsibility **features**
- Reduced coupling (sender and receiver have no explicit knowledge of each other)
- Added flexibility in assigning responsibilities to objects (can change the chain at runtime)
- Receipt isn't guaranteed

**Implementing** chain of responsibility
- Can use new chain (parent references), or existing ones (like Composite pattern)
- If one were to implement new chains, by default a HelpHandler (like the example above) contains a successor (HelpHandler) pointer, and the default behavior of a HelpHandler's handleRequest call forwards the call to the successor. One can override HelpHandler and use the overridden one to provide actual functionality
- Request can be represented with an int, a string (flexibility but more handling), an object (pass to handleRequest calls), or the Handler class itself can be specific enough (like a HelpHandler)
- (Client can have a HelpHandler, or can be a HelpHandler themselves)

Chain of responsibility is often applied in conjunction with **Composite** (successor in the chain being parent)