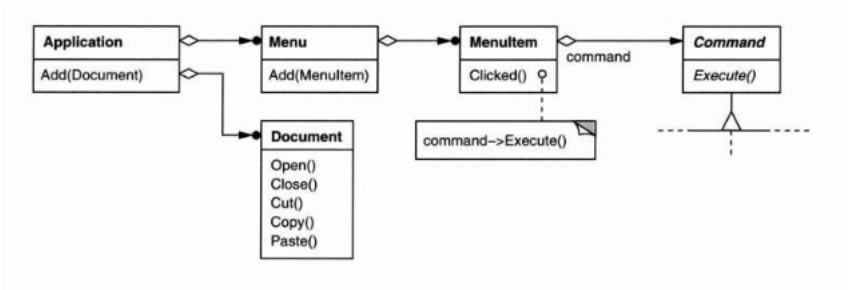
Command

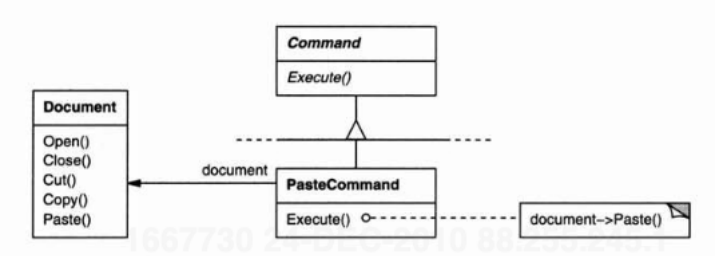
Command

- Intent
  - Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
- Motivation
  - issue requests to objects without knowing anything about the operation being requested or the receiver of the request
  - add new types of requests

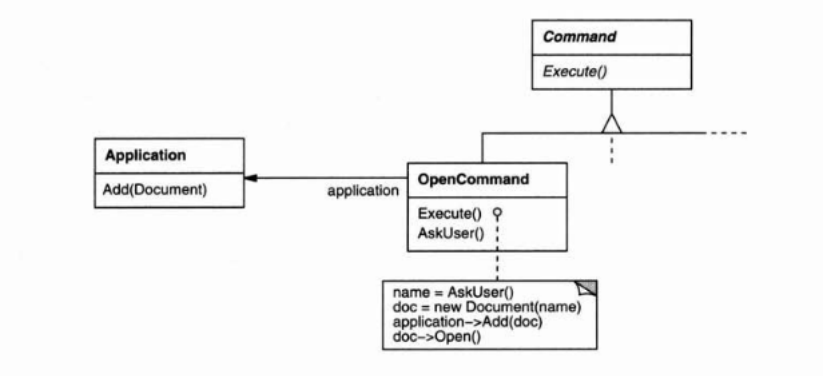
Command



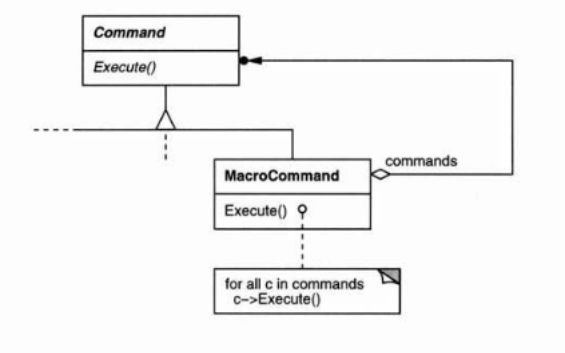
Command



Command



Sequence of Commands



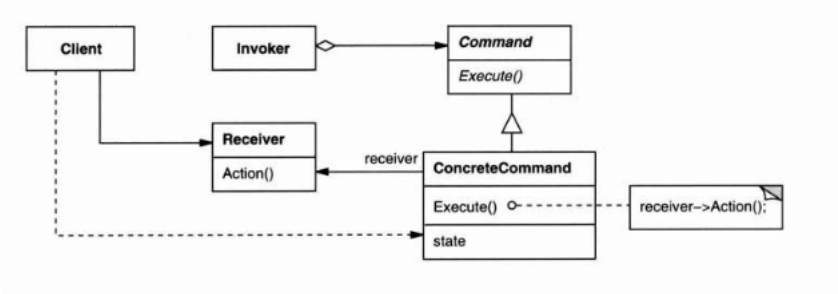
Applicability

- With Command, you can
  - specify, queue, and execute requests at different times. A Command object can have a lifetime independent of the original request.
  - transfer a command object for the request to a different process and fulfill the request there.
  - support undo. The Command's Execute operation can store state for reversing its effects in the command itself.
    - The Command interface must have an added Unexecute operation

Applicability

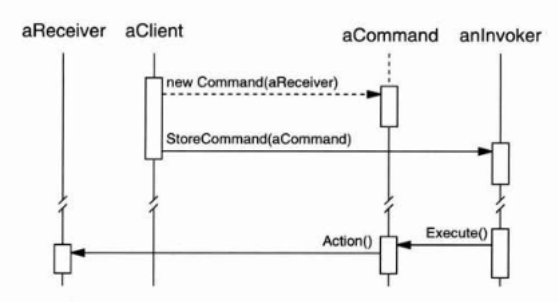
- With Command, you can
  - support logging changes so that they can be reapplied in case of a system crash.
  - extend the system with new transactions.
  - execute a group of Commands as a transaction

Structure



Collaborations

- Command decouples the invoker from the receiver



Consequences

- Commands are first-class objects. They can be manipulated and extended like any other object.
- You can assemble commands into a composite command. An example is the MacroCommand class described earlier.
- It's easy to add new Commands, because you don't have to change existing classes.


```
class Command {
public:
    virtual ~Command();
    virtual void Execute() = 0;
protected:
    Command();
};

class OpenCommand: public Command {
public:
    OpenCommand(Application*);
    virtual void Execute();
protected:
    virtual const char* AskUser();
private:
    Application* _application;
    char* _response;
};
```

```
OpenCommand::OpenCommand(Application* a) {
    _application = a;
}

void OpenCommand::Execute() {
    const char* name = AskUser();
    if (name != 0) {
        Document* document = new Document(name);
        _application->Add(document);
        document->Open();
    }
}

class PasteCommand: public Command {
public:
  PasteCommand(Document*);
  virtual void Execute();
private:
  Document* _document;
};
PasteCommand::PasteCommand(Document* doc) {
  _document = doc;
}
void PasteCommand::Execute() {
  _document->Paste();
}
```