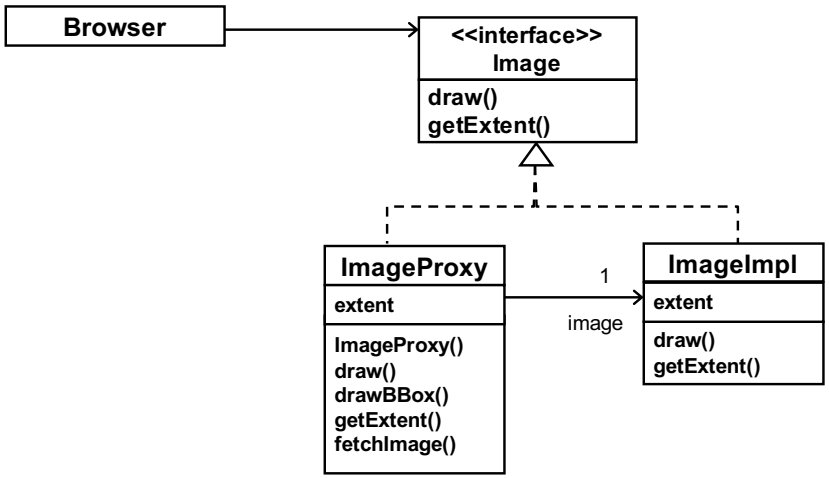
Proxy Design Pattern

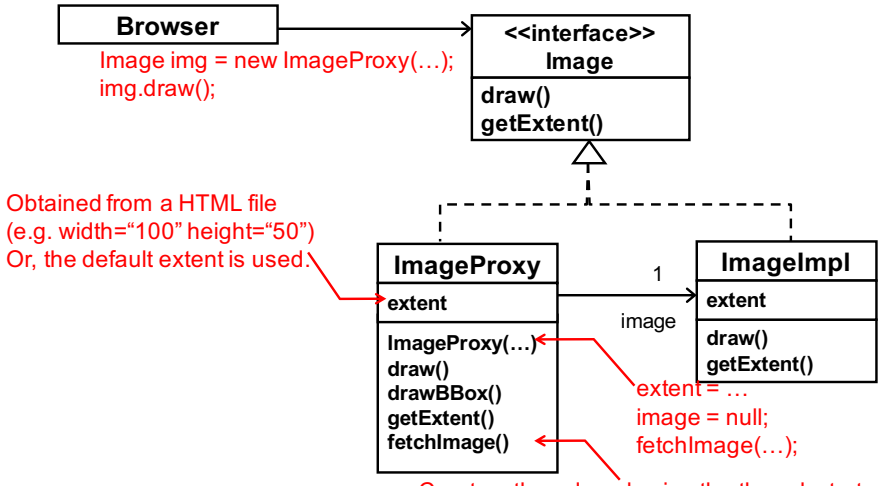
Proxy Design Pattern

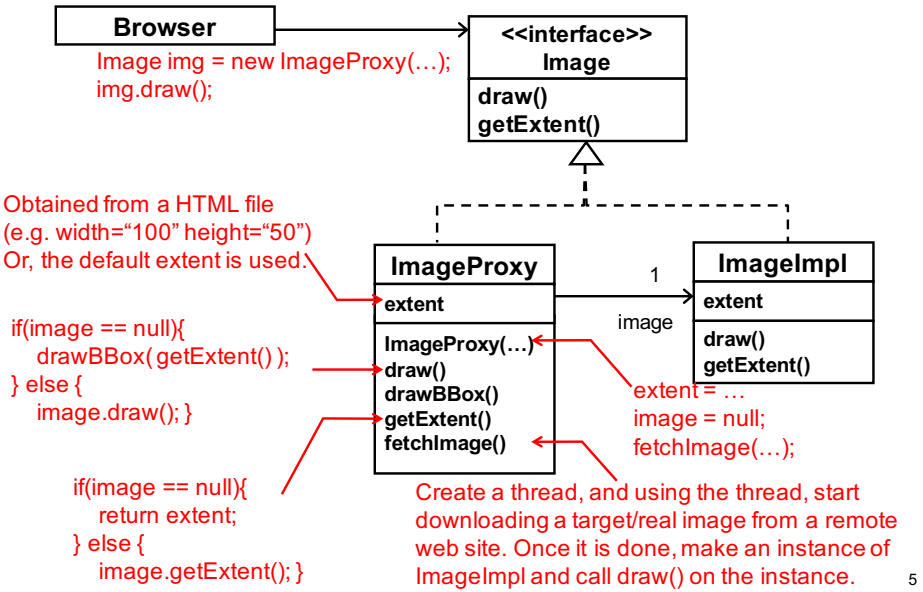
- Intent
  - Provide a surrogate or placeholder for another object to control access to it.
- An example: Image handling in a web browser
  - When an HTML file contains images,
    - A bounding box (placeholder) is displayed first for each image
      - Until the image is downloaded and ready to be displayed.
        - » Most users are not patient enough to keep watching blank browser windows until all text and images are downloaded and displayed.
    - Whenever the image is downloaded, the bounding box is replaced with the real image.

An Example of *Proxy*



Proxy Class

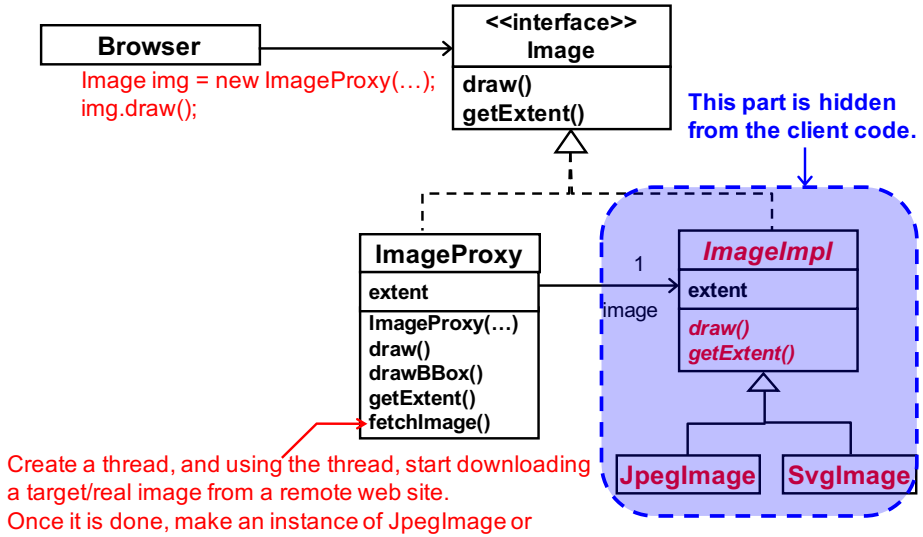




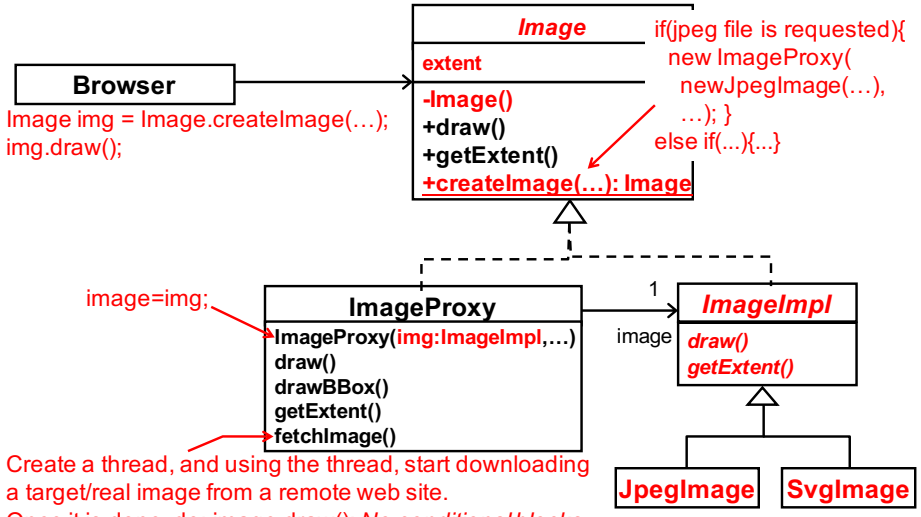
What's the Point?

- Loosely couple bounding box placement and image rendering.
- Why is that important?
  - Changes are expected for
    - Image formats that the browser supports.
    - Rendering algorithms
  - Bounding box placement is independent from those changes.
  - Separate *what can change often* from *what wouldn't* to improve maintainability.

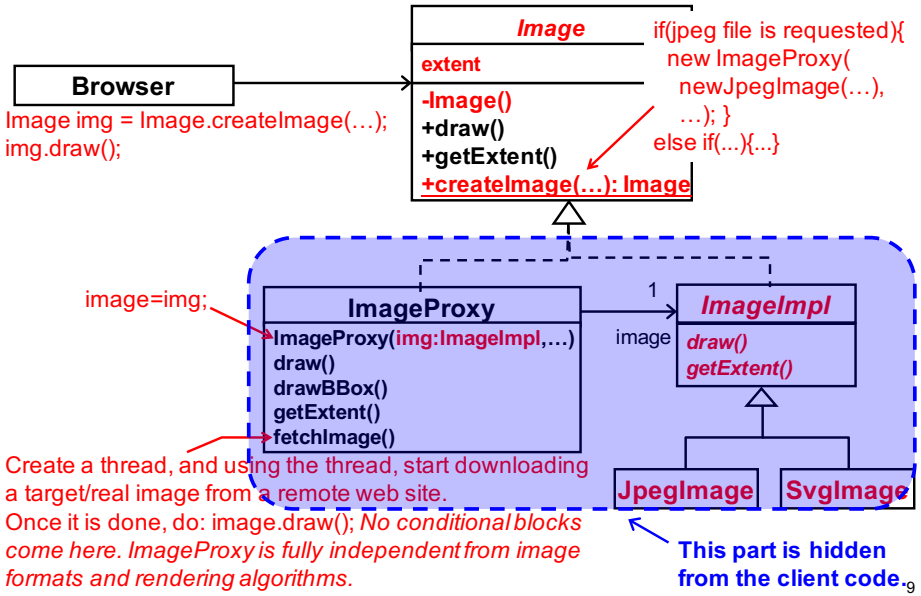
Supporting Multiple Image Formats



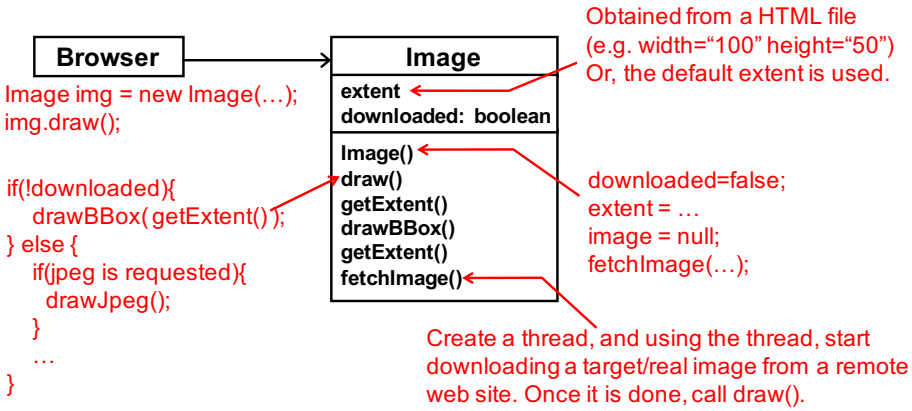
One Step Further with *Factory Method*



One Step Further with *Factory Method*



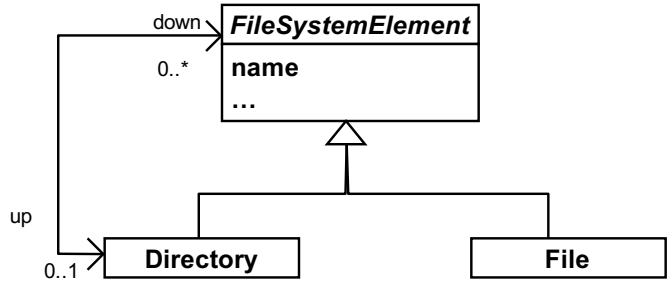
What if Everything is Integrated into a Single Class?



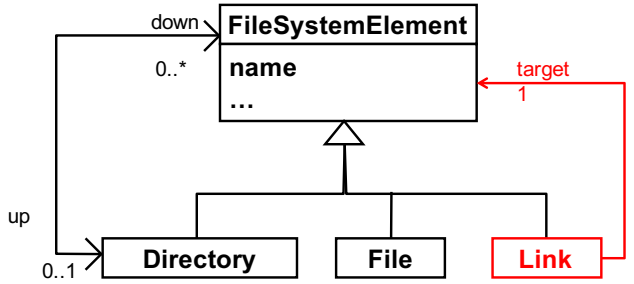
Bounding box placement, choices of image formats and image rendering are all mixed up in a single class, which will become fat and spaghetti (i.e. unmaintainable) soon.

Better design strategy: Separation of concerns

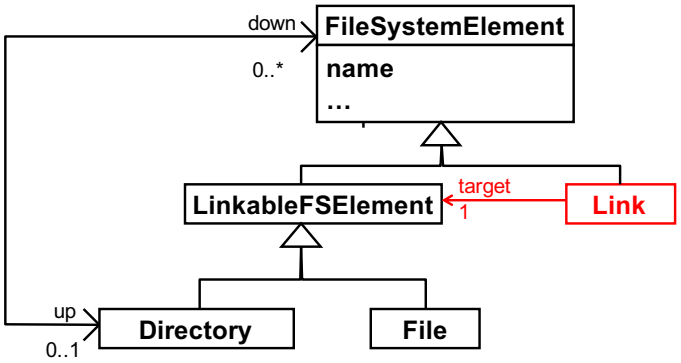
Add Proxy to your File System



- Add a symbolic link feature
  - a.k.a. alias (Mac), shortcut (Windows)
- A link acts as a proxy of a directory or file.
- Use the Proxy design pattern.

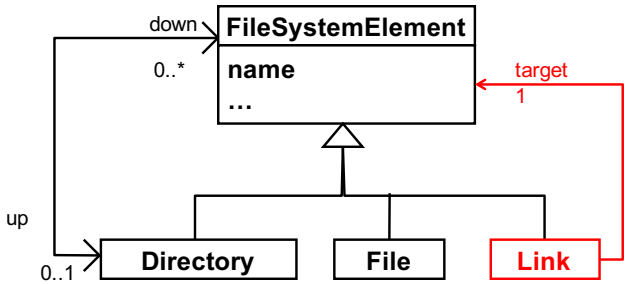


- A link acts as a proxy of a directory or file.
  - A link can act as a proxy of a link too.

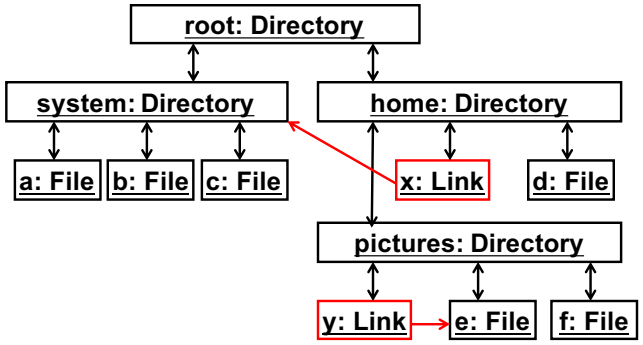


- A link acts as a proxy of a directory or file.
  - A link can act as a proxy of a link too.

HW15-1: Add Link to your HW 14 Code



- If a Link refers to a file, Link.getSize() returns the size of that file.
- If a Link refers to a directory, Link.getSize() returns the total amount of disk consumption by the directory's all child nodes.
- If a Link refers to another link, Link.getSize() goes through a chain of links until it reaches a file or directory.



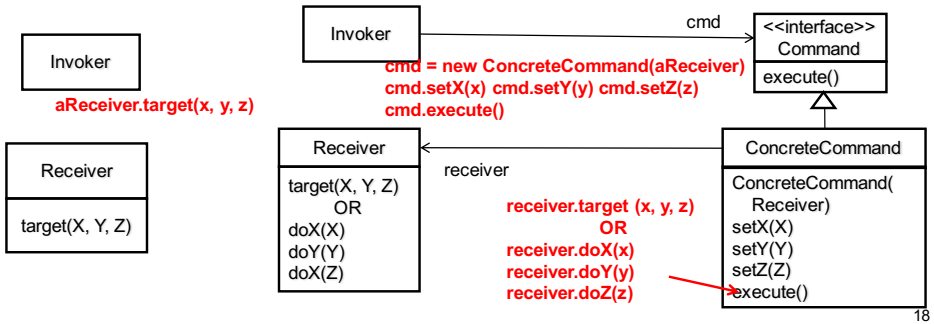
- Make this tree structure in your test case.
  - Assign values to data fields (size, owner, etc.) as you want.
  - Call getSize() on the root directory.
  - Call showAllElements() to print out this tree structure.
    - You can define your own textual format.

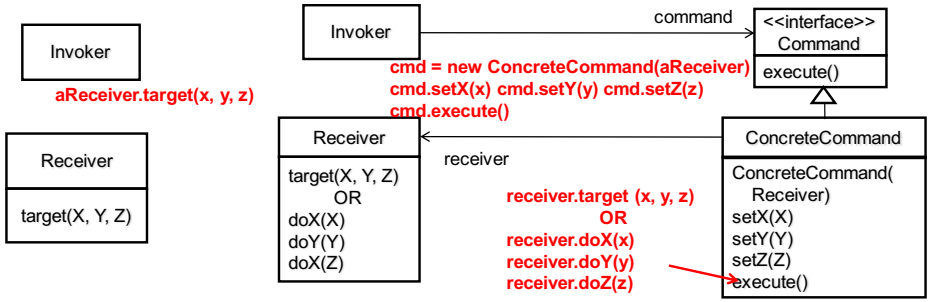
Command Design Patten

Command Design Pattern

- Intent
  - Encapsulate a request/command (a method call) and its relevant information (e.g., parameters) as a class.
    - Replace a method call with a class.

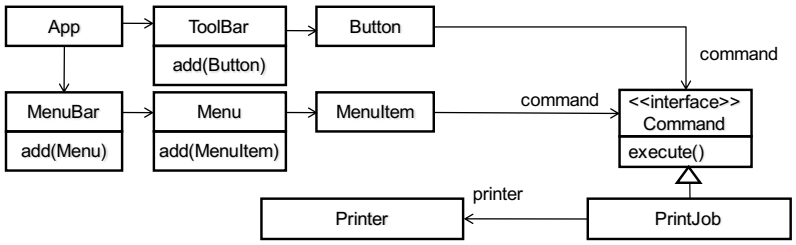
Simple method call **Command design pattern**

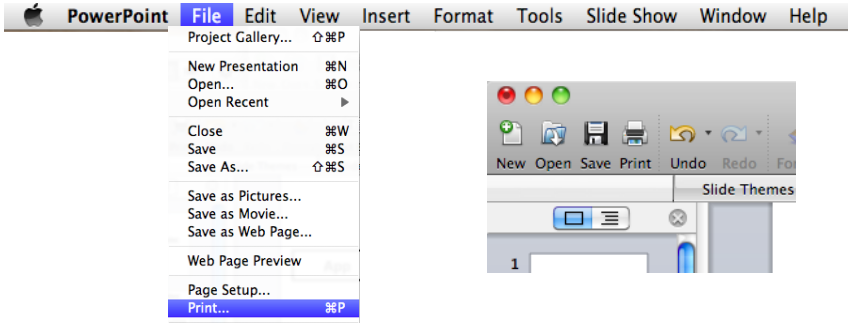


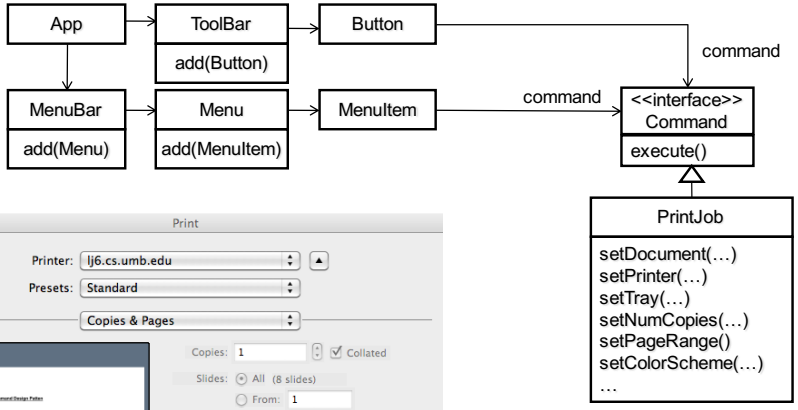


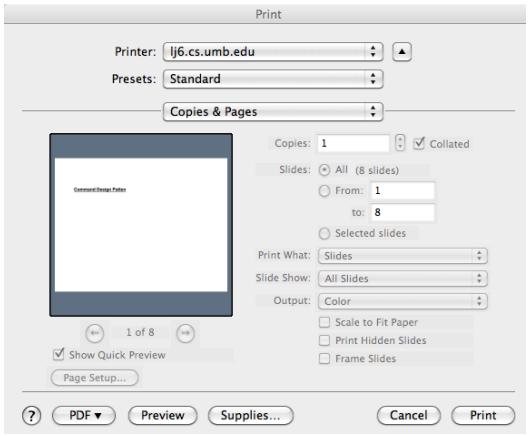
- Benefits
  - Loosely couple an invoker and a receiver
    - An invoker doesn't have to know how to perform a command.
    - Invokers can be intact when receivers are changed.
  - Make it easy to add new commands
    - No need to change invokers and receivers.

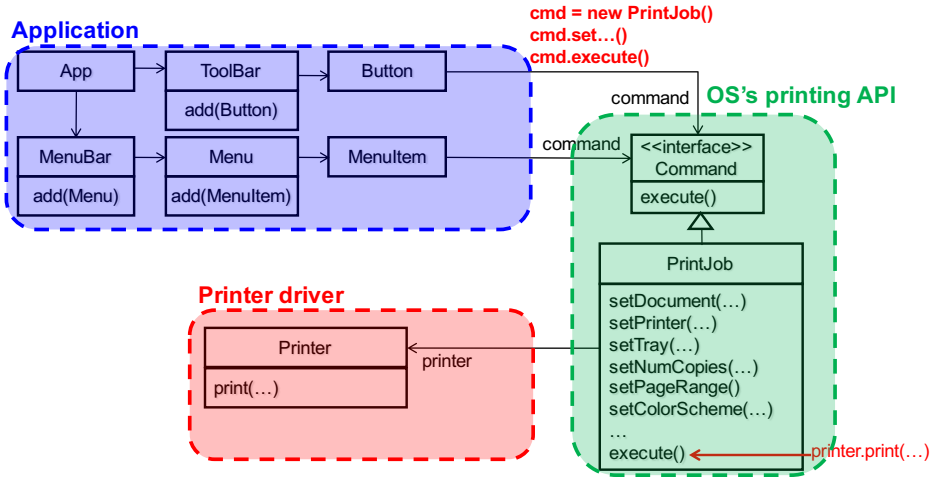
An Example: Print Command







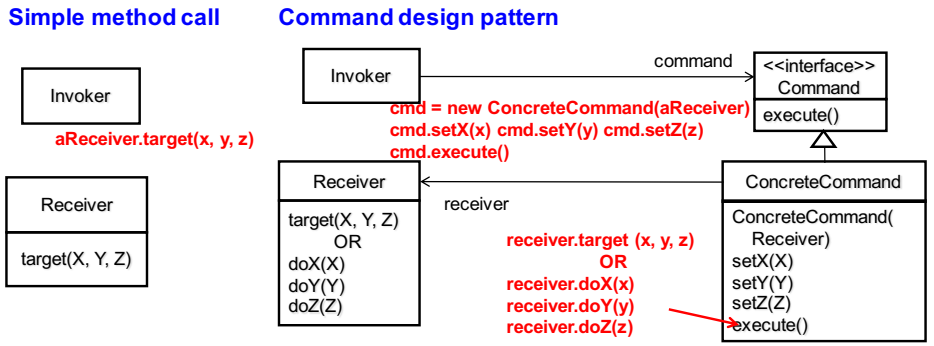




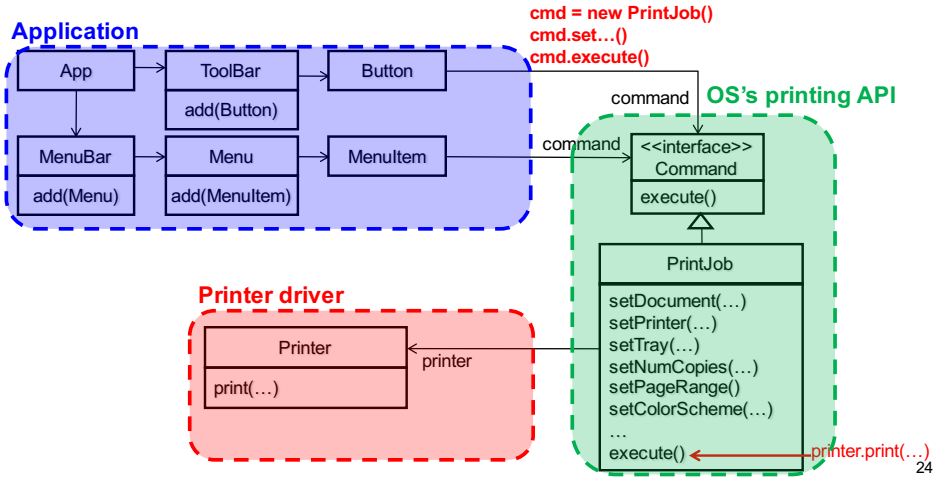
- Loosely couple invokers (a button and a menu item) and a printer
  - Invokers don't have to know how to perform a command.
    - They don't have to know the underlying printing facility (OS' printing API, printer drivers, etc.)
  - Invokers can be intact when the printer (its driver) is changed.
- Make it easy to add new commands such as faxing, PDF/PS generation and "Send PDF via email" and "Send PDF via text."
  - No need to change the printer and invokers

When to Use *Command*?

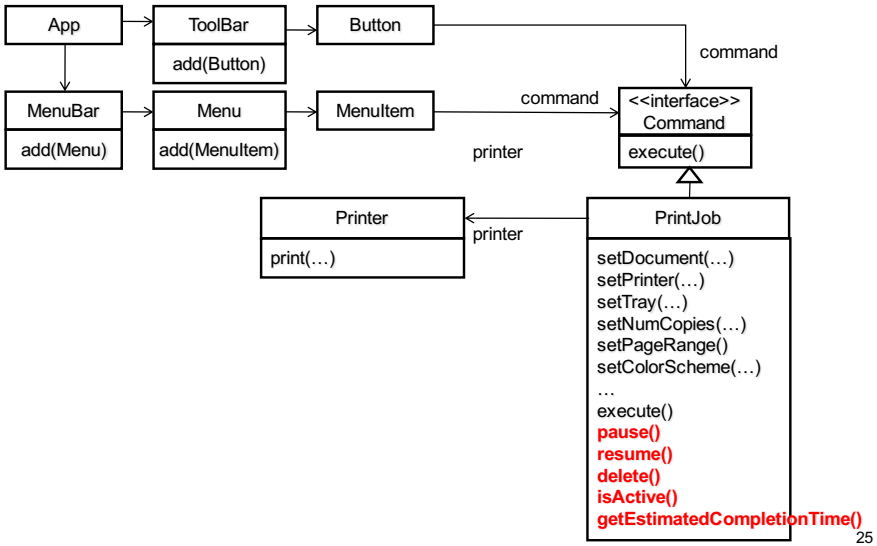
- When do you want to use *Command*, rather than a regular method call?
  - Why not using a regular method call?



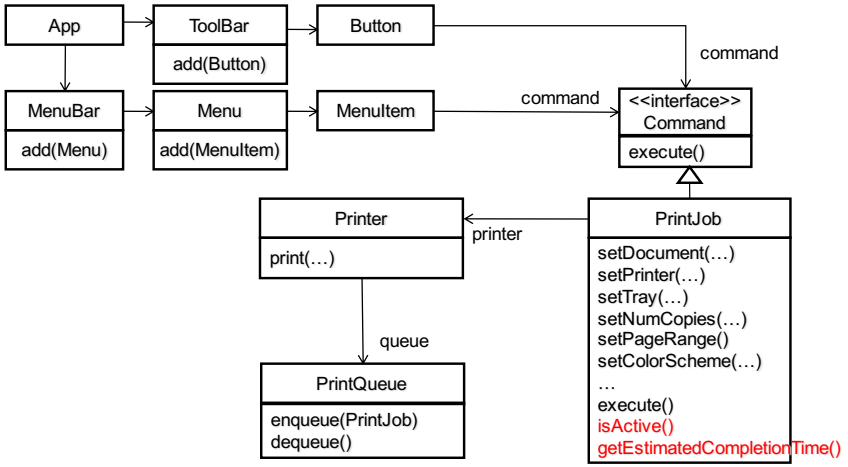
- When you want invokers and receivers to be loosely-coupled.



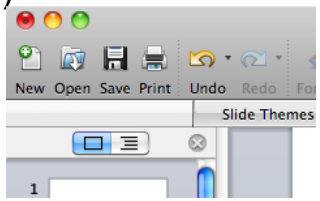
- When a command has many relevant information/parameters.
- When you have many invokers for each command.
- When you want to perform some operations on a command.

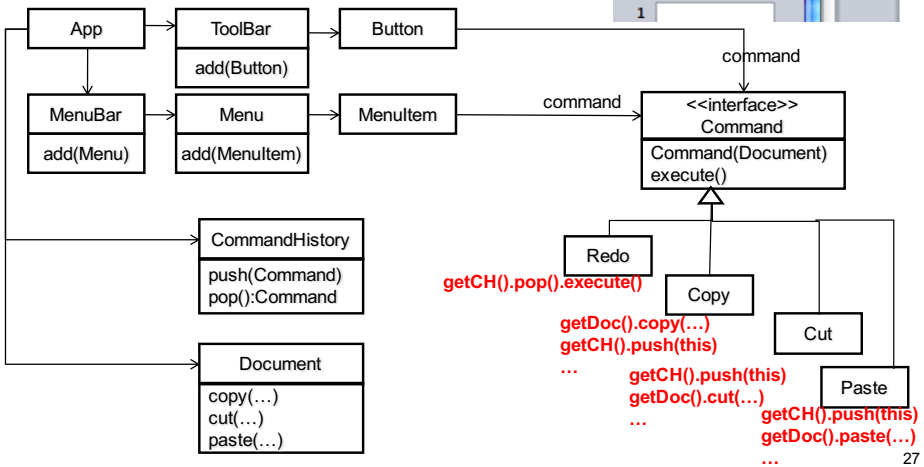


- When you would like to manage (or keep track of) multiple commands

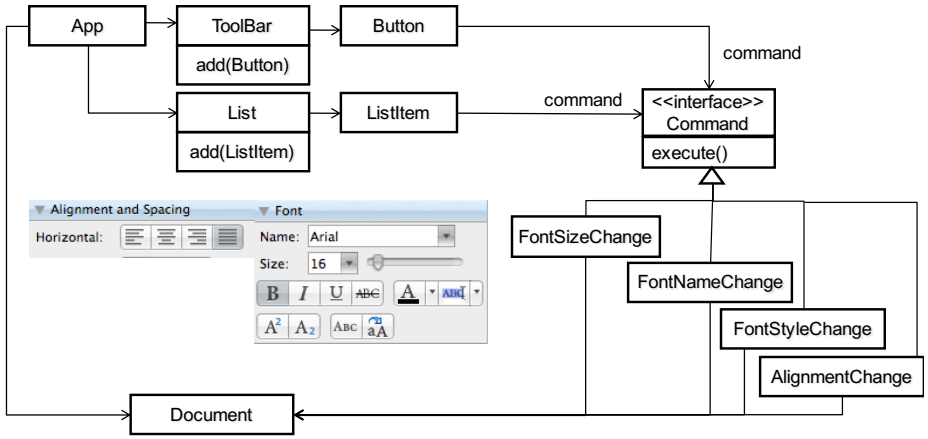


- When you would like to record (or log) command history
  - e.g., for implementing "undo" and "redo" function.



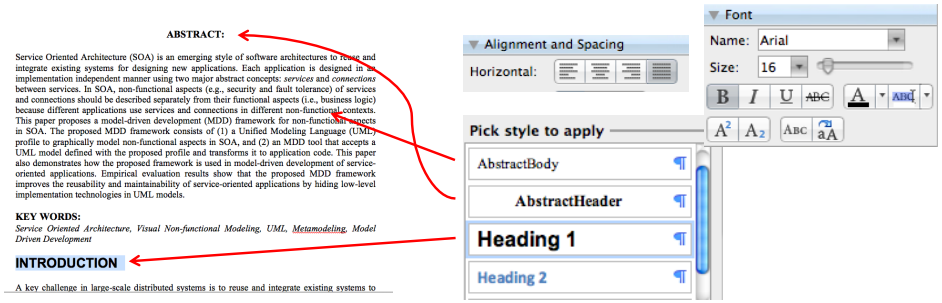


Another Example

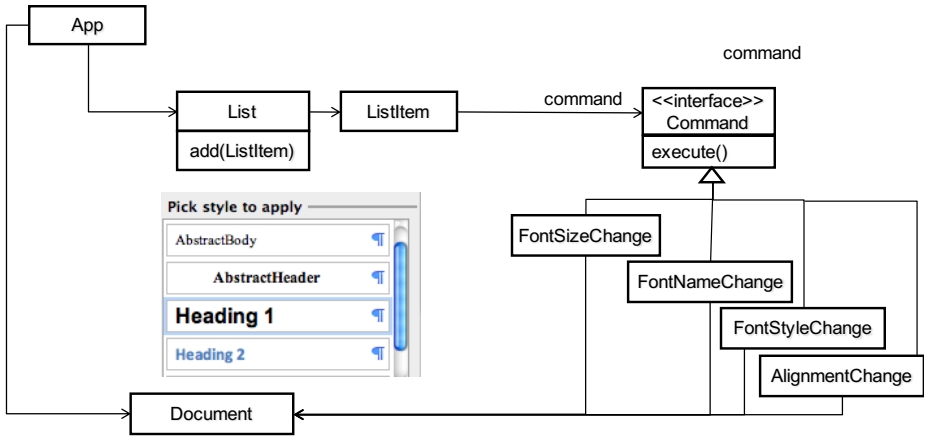


- These 4 command classes correspond to the buttons for changing font size, font name font style and alignment.

A Sequence of Commands (Composite Command)

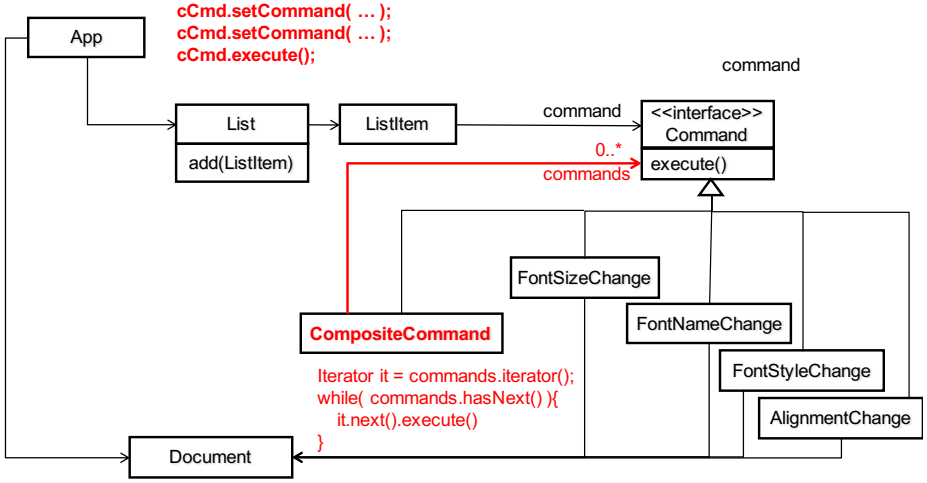


- Clicking "Abstract Header" style = performing the following 4 commands
  - Font size = 12pt
  - Font name = Times New Roman
  - Font style = bold
  - Alignment = center
- Clicking "Heading 1" style = performing the following 5 commands
  - Font size = 16pt
  - Font name = Arial
  - Font style = bold
  - Alignment = left
  - Effect = all caps



- How would you design command classes for list items to change formatting styles?
  - e.g., "Abstract Header"? (12pt, Times, bold, center)


```
Command cCmd = new CompositeCommand("Abstract Header");
Create 4 command objects to change font size, font name, font style and alignment.
For each command object, set a parameter value(s) (e.g. 12pt, Times, etc.).
cCmd.setCommand( … );
cCmd.setCommand( … );
cCmd.setCommand( … );
cCmd.execute();
```



Command + Composite design patterns (plus Iterator)

HW15-2

- Complete the pseudo code by
  - Replacing natural language descriptions with pseudo code
  - Replacing "…" with pseudo code
- You are free to add methods to command classes.
- Do not send me the entire pseudo code
  - I don't need the pseudo code for classes that appear in Slide 31.
  - I need the pseudo code for *client code* that uses the classes in Slide 31.

Wizards

- Wizard
  - e.g., installation wizard, project creation wizard, refactoring wizard
  - Shows a sequence of modal dialogs to collect a set of data from the user
    - e.g., one dialog to enter registration information (user name, serial number, etc.)
    - Another dialog to specify the directory to install an app in question
    - Another dialog to enable and disable application components/features
    - Moves one dialog to another with the "next" and "back" buttons
  - Performs a single action/command only when the "finish" button is clicked on the last dialog.

```
Command next = new Installation();
while (next != null) {
   next = next.execute();
}
```