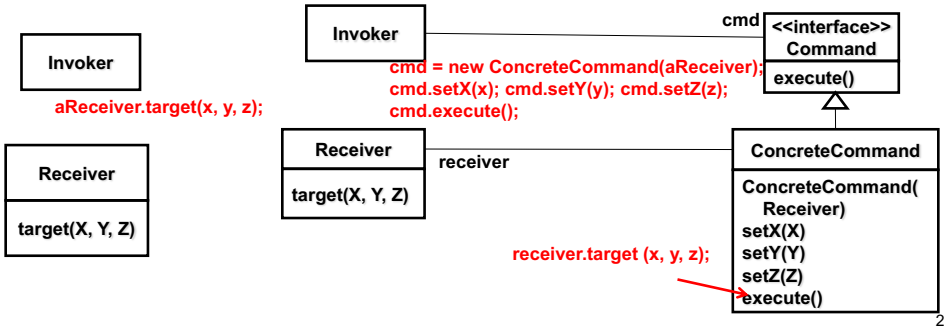
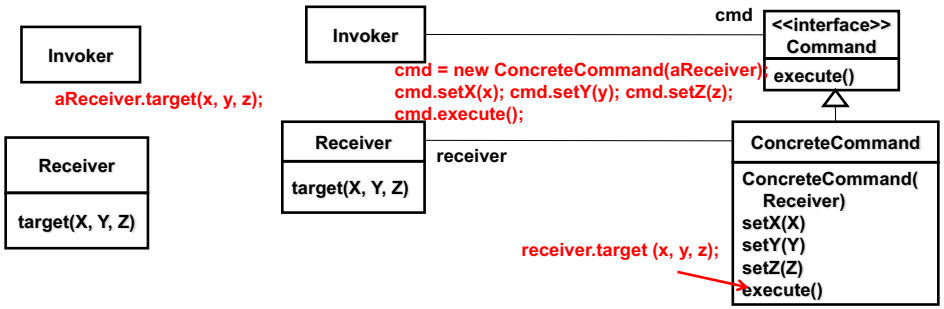
Command Design Patten

Command Design Pattern

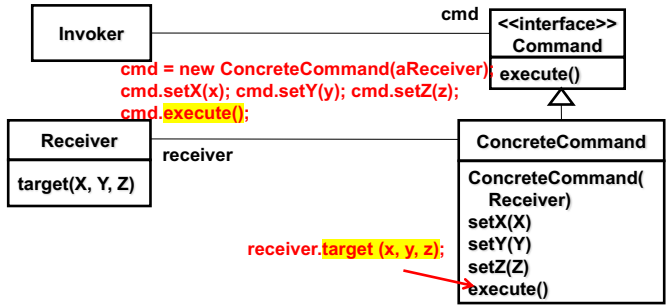
- Intent
  - Encapsulate a request/command (or a method call) and its relevant information (or method parameters) as a class.
    - Replace a method call with a class.

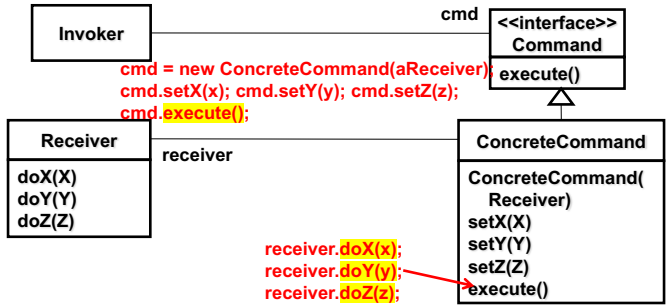
Simple method call **Command design pattern**



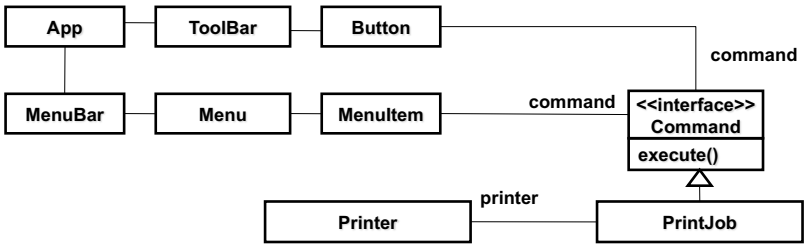


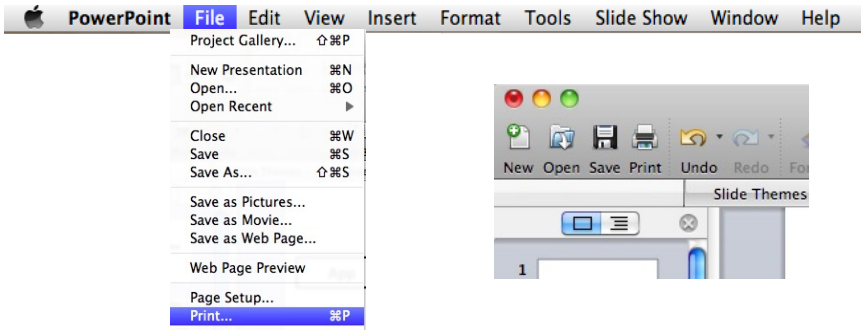
- Benefits
  - Separate (loosely couple) an invoker and a receiver
    - The invoker doesn't have to know how to perform a command (i.e., which method to call on the receiver).
    - The invoker can be intact when the receivers is changed.
  - Make it easy to add new commands in addition to existing ones.

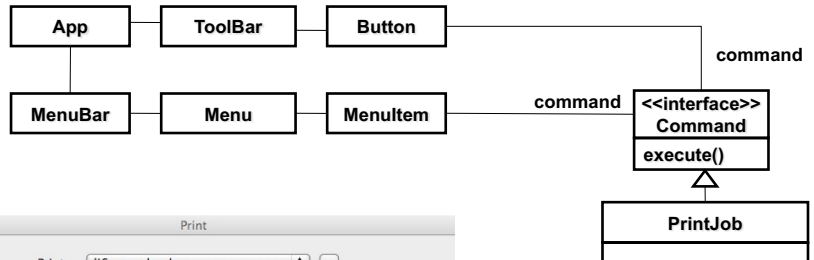


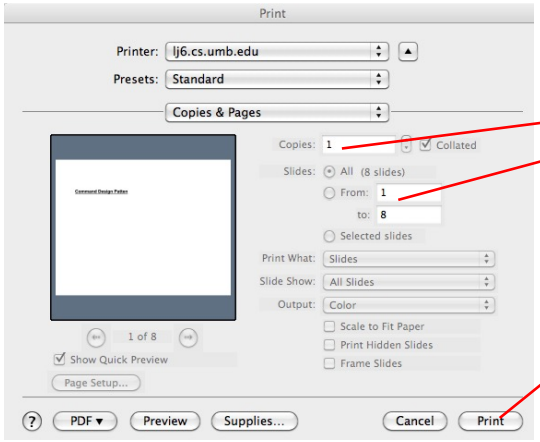


An Example: Print Command

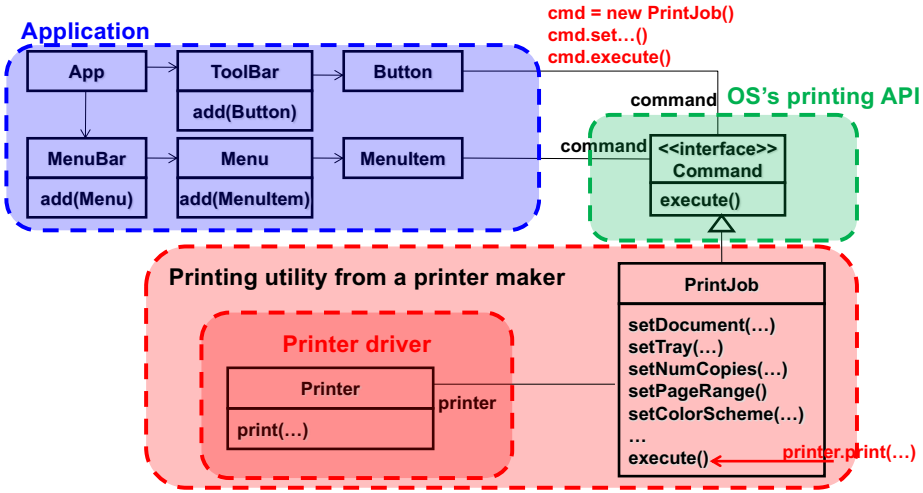




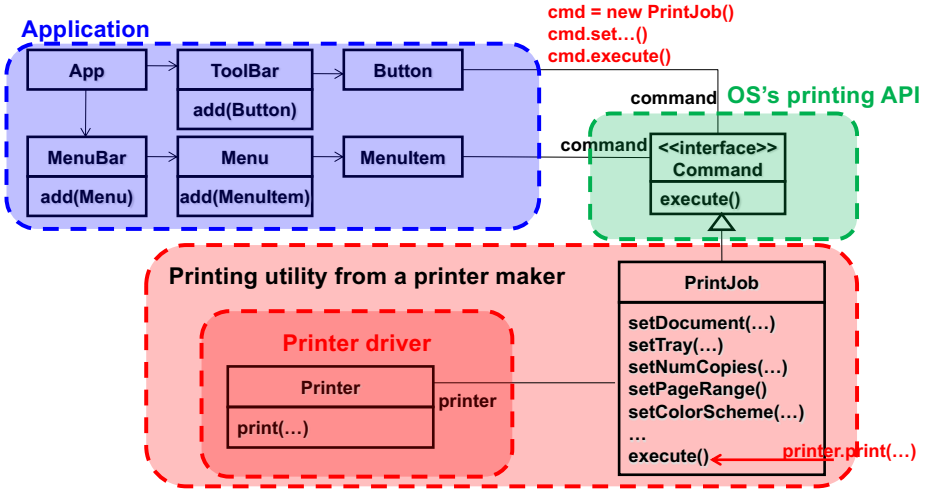




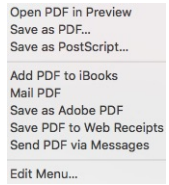
PrintJob cmd = new PrintJob()
cmd.set...()
cmd.execute()



- Separate invokers (the "print" button and the "print" menu item) and a receiver (printer driver)
  - Invokers don't have to know how to perform a command (i.e., which method to call on the printer driver)
    - They don't have to know low-level details in the underlying printing facility.
  - Invokers can be intact when the printer driver is changed.

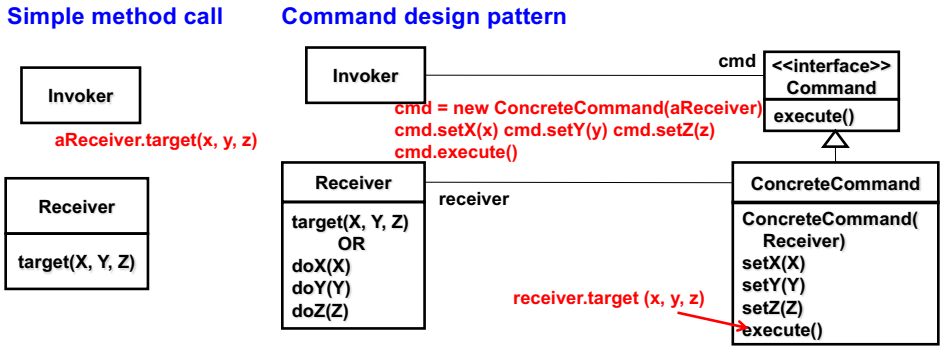


- Make it easy to add new commands such as "PDF generation" and "Send PDF via messages."
  - No need to change the printer driver nor invokers

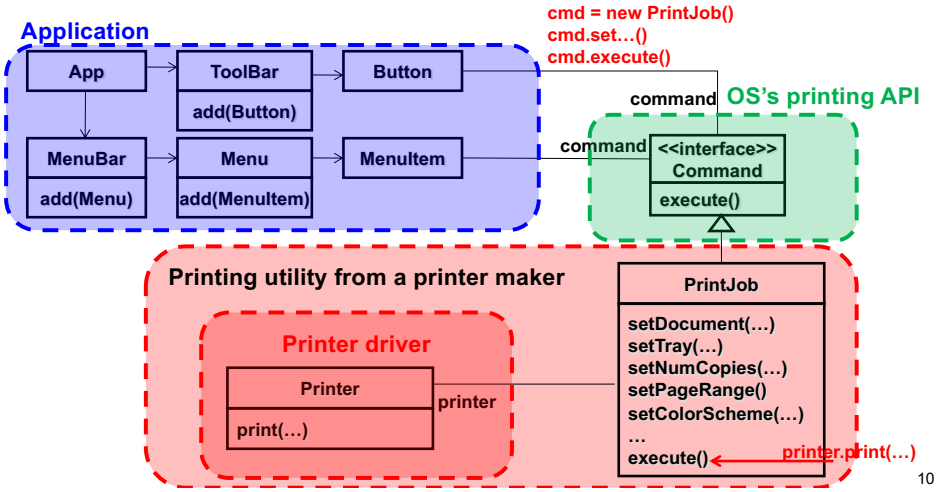


When to Use *Command*?

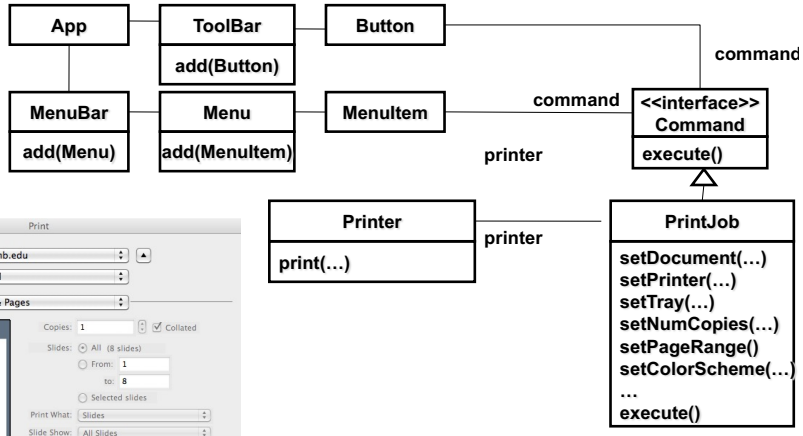
- When do you want to use *Command*, rather than a regular method call?
  - Why not using a regular method call?

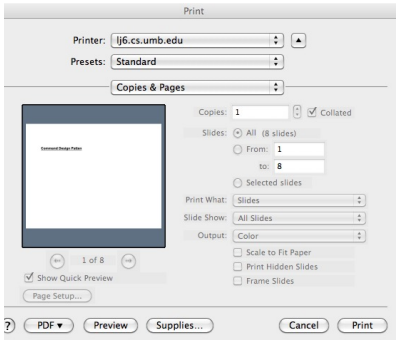


- When you want invokers and receivers to be loosely-coupled.

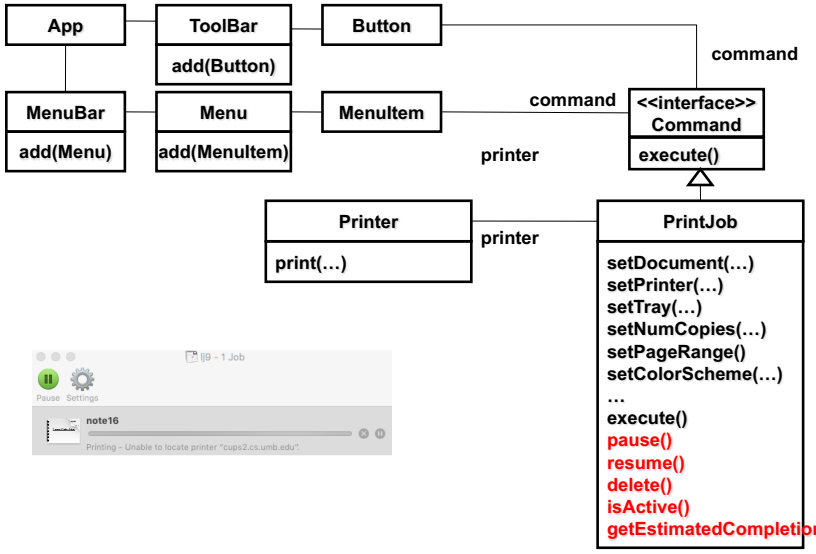


- When you have many invokers for each command.
- When a command has many relevant information (parameters).

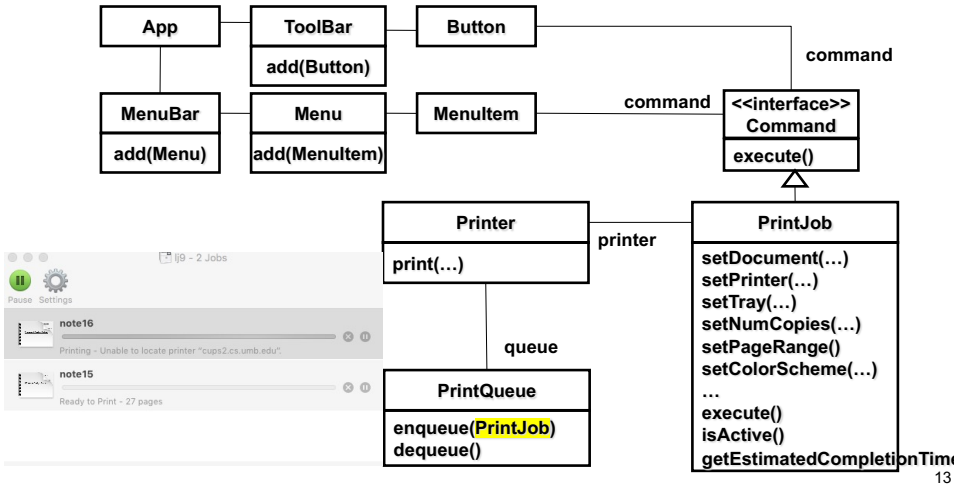




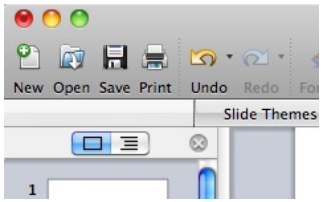
- When you want to perform some operations on a command.

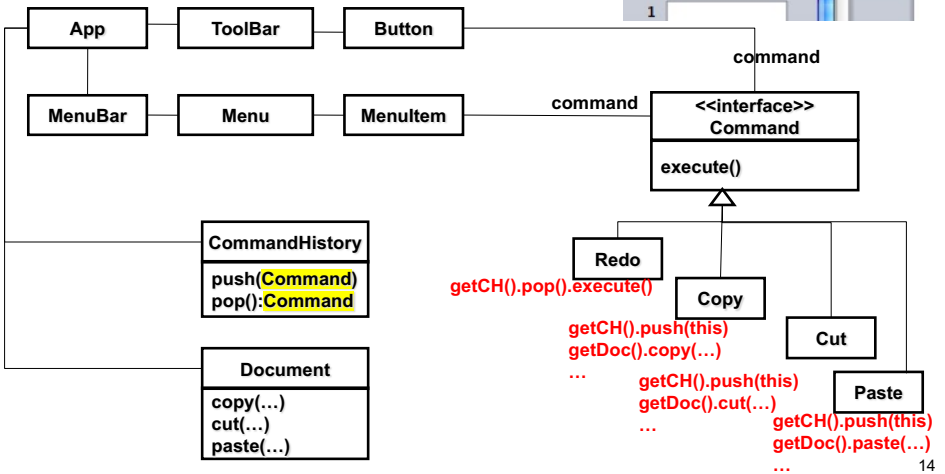


- When you want to manage (or keep track of) multiple commands



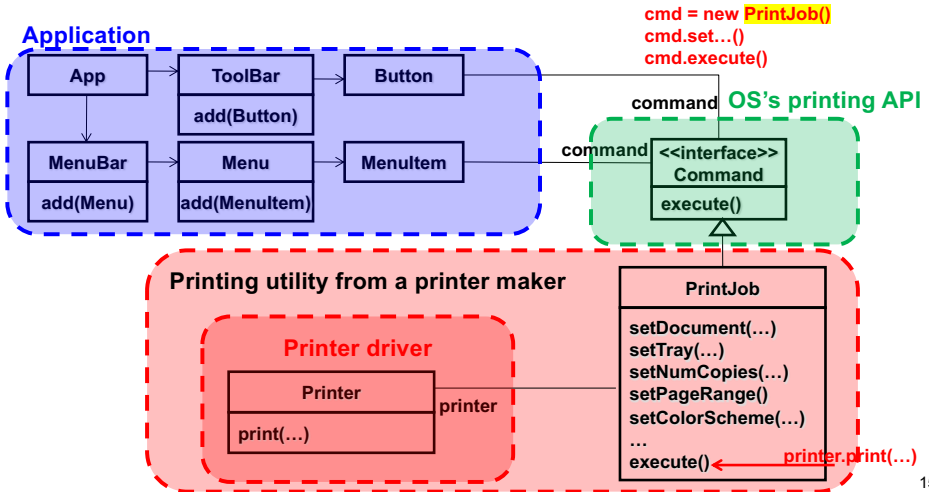
- When you want to record (or log) command history
  - e.g., for implementing "undo" and "redo" function.





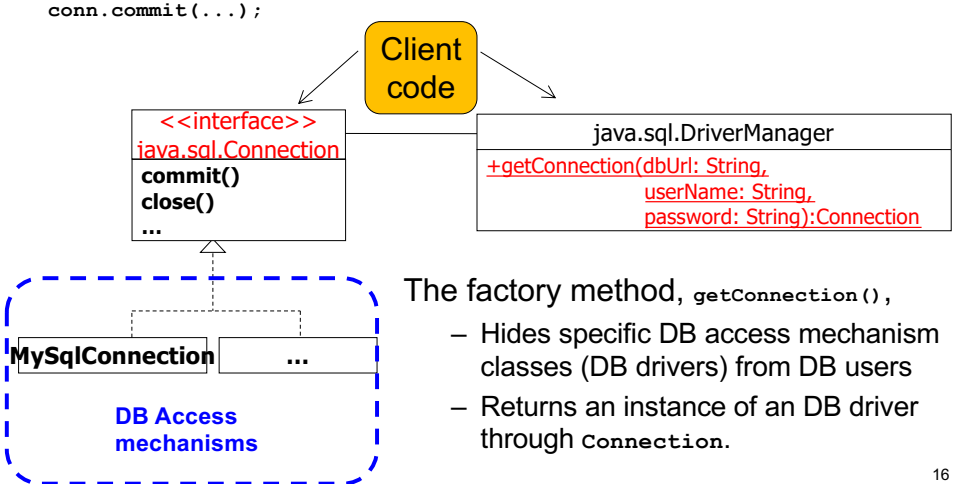
Coupling b/w PrintJob and its Callers

- `PrintJob` and its callers might be tightly-coupled.
  - You can separate them if you want.

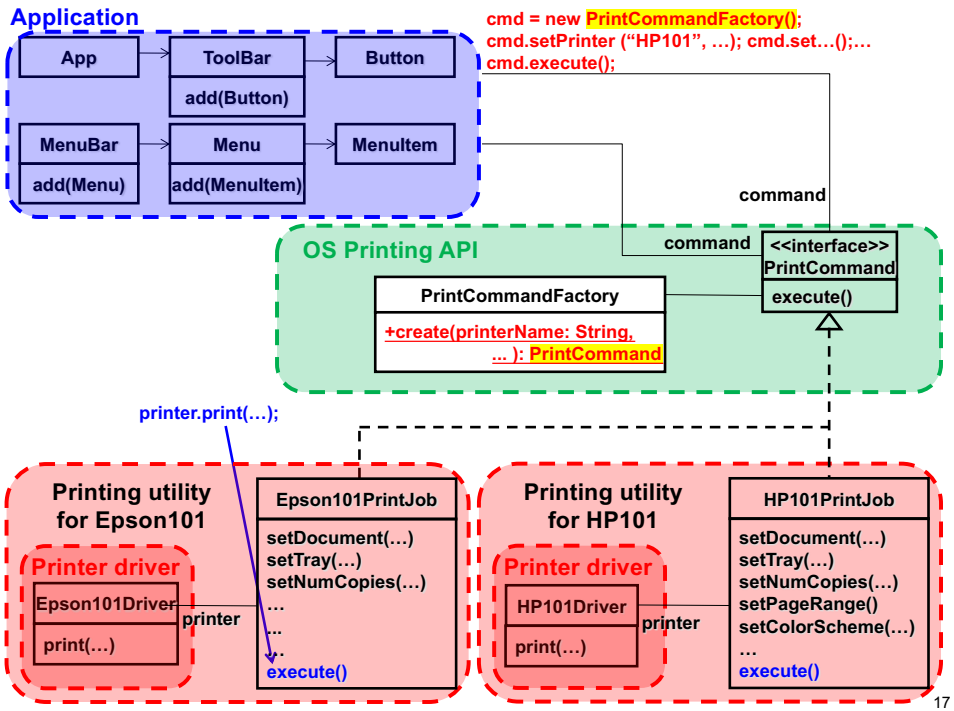


Recap:
`DriverManager.getConnection()` in JDBC API

- ```
  Connection conn =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb",
                                "johnDoe", "abcd");
  conn.commit(...);
  ```

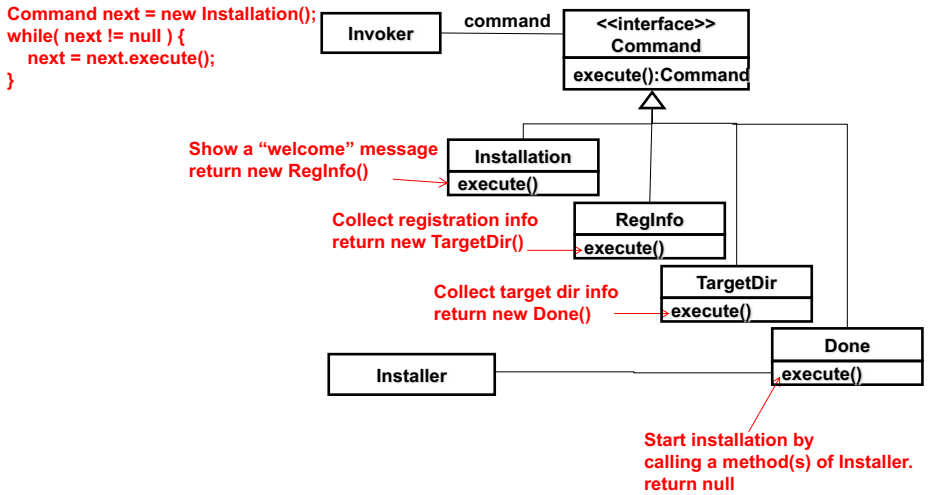


The factory method, `getConnection()`,
- Hides specific DB access mechanism classes (DB drivers) from DB users
- Returns an instance of an DB driver through `Connection`.
  ```



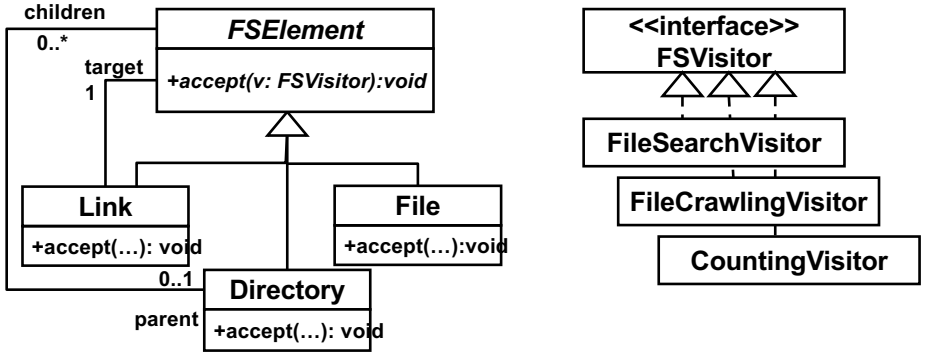
One More Example: Wizards

- Wizard
  - e.g., installation wizard, project creation wizard, refactoring wizard, etc.
  - 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
  - Starts a particular action only when the "finish" button is clicked on the last dialog.



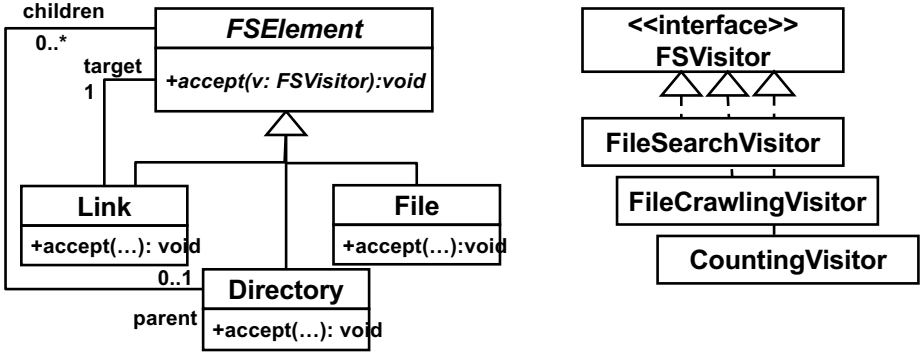
Recap: HW 9

- You implemented 3 visitors to perform 3 types of operations on FS-related data structures.




```
FileSearchVisitor visitor = new FileSearchVisitor("example.txt");
rootDir.accept( visitor );
visitor.getFoundFiles().size();
```

- Client code (i.e. callers of those operations) are tightly-coupled with visitors and FS-related data structures.

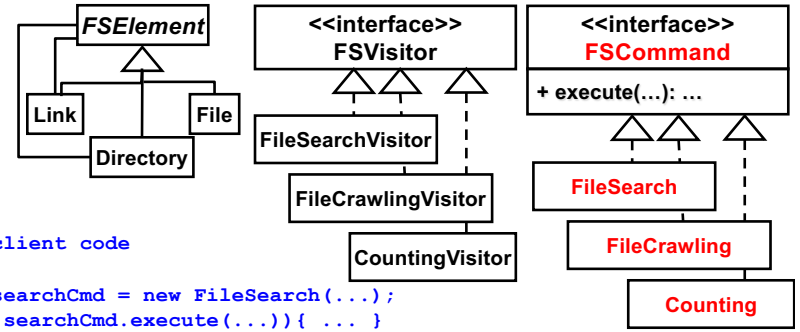



```
FileSearchVisitor visitor = new FileSearchVisitor("example.txt");
rootDir.accept( visitor );
visitor.getFoundFiles().size();

FileCrawlingVisitor visitor = new FileCrawlingVisitor();
rootDir.accept( visitor );
visitor.getFiles();
```


HW 10

- Separate client code (i.e. callers of those ops) from visitors and FS-related data structures
  - with the *Command* design pattern.




```
// Example client code

FileSearch searchCmd = new FileSearch(...);
for(File f: searchCmd.execute(...)){ ... }

FileCrawling crawlCmd = new FileCrawling(...);
for(File f: crawlCmd.execute(...)){ ... }
```