Design Patterns

Design Patterns

- Tested, proven and documented solutions for recurring design problems in given contexts.
- Each design pattern is structured as
  - Pattern name
  - Intent
  - Motivation
  - Applicability
  - Class structure
  - Participants
  - …etc.

Resources

- *Design Patterns: Elements of Reusable Object-Oriented Software*
  - By Eric Gamma et al.
  - Addison-Wesley
- *Head Start Design Patterns*
  - by Elizabeth Freeman et al.
  - O'Reilly
- Web
  - http://en.wikipedia.org/wiki/Design_patterns_(computer_science)
  - http://sourcemaking.com/design_patterns

Benefits of Design Patterns

- Useful information source to learn and practice good object-oriented designs
- Useful as a communication tool among developers
  - e.g., Recursion, collection (array, stack, queue, etc.), sorting, buffers, infinite loops

Recap: Brief History to OOD

- In good, old days… programs had no structures.
  - One dimensional code.
    - From the first line to the last line on a line-by-line basis.
    - "Go to" statements to control program flows.
      - Produced a lot of "spaghetti" code
        - » "Go to" statements considered harmful.
  - No notion of structures (or modularity)
    - Making a chunk of code (module) self-contained and independent from the other code
      - Improve reusability and maintainability
        - » Higher reusability → higher productivity, less production costs
        - » Higher maintainability → higher productivity and quality, less maintenance costs

Modules in SD and OOD

- Modules in Structured Design (SD)
  - Structure = a set of variables (data fields)
  - Function = a block of code
- Modules in OOD
  - Class = a set of data fields and functions
  - Interface = a set of abstract functions
- Key design questions/challenges:
  - how to define modules
  - how to separate a module from others
  - how to let modules interact with each other

SD v.s. OOD

- OOD
  - Intends coarse-grained modularity
    - The size of each code chuck is often bigger.
  - Extensibility in mind in addition to reusability and maintainability
    - How easy (cost effective) to add and revise existing modules (classes and interfaces) to accommodate new/modified requirements.
    - How to make software more flexible/robust against changes in the future.
  - How to gain reusability, maintainability and extensibility?
    - Design patterns as example good OODs.

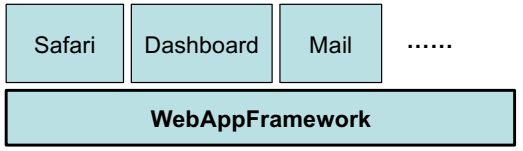
Factory Method

Factory Method

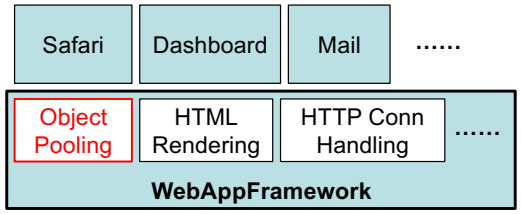
- A method to instantiate a class and initializes a class instance without using its constructors
  - Uses a regular (i.e., non-constructor) method.
  - Lets a class *defer* instantiation to subclasses.
    - Define an abstract class (or an interface) for creating an instance.
    - Let its subclasses (or implementation classes) decide *which class to instantiate*.

An Example: Web App Dev Framework

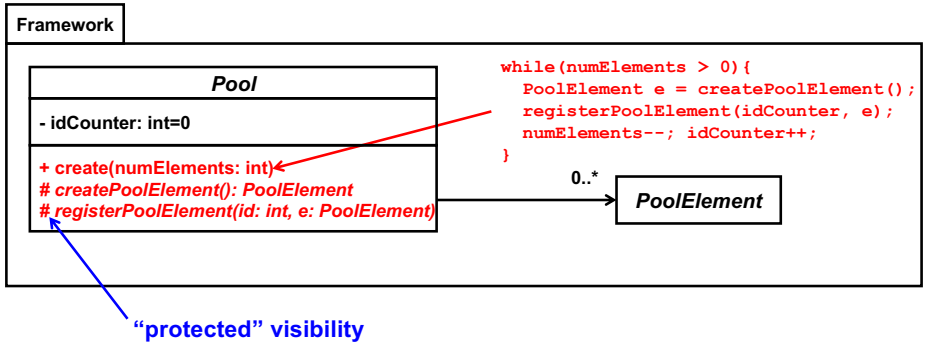
- Application framework
  - A set of foundation APIs to implement and run a series of applications.
    - Implement the standard/common functionalities (structures and behaviors) among individual applications
    - Make them reusable/available for individual apps.
    - Make app development easier and faster.
- Assume you are implementing a development framework for various types of web apps
  - Frameworks for web applications
    - e.g., WebKit, Struts, Ruby on Rails, etc.
    - WebKit (http://www.webkit.org/)
      - Web browsers (incl. Safari), Dashboard, Mail and many other Mac OS X apps.

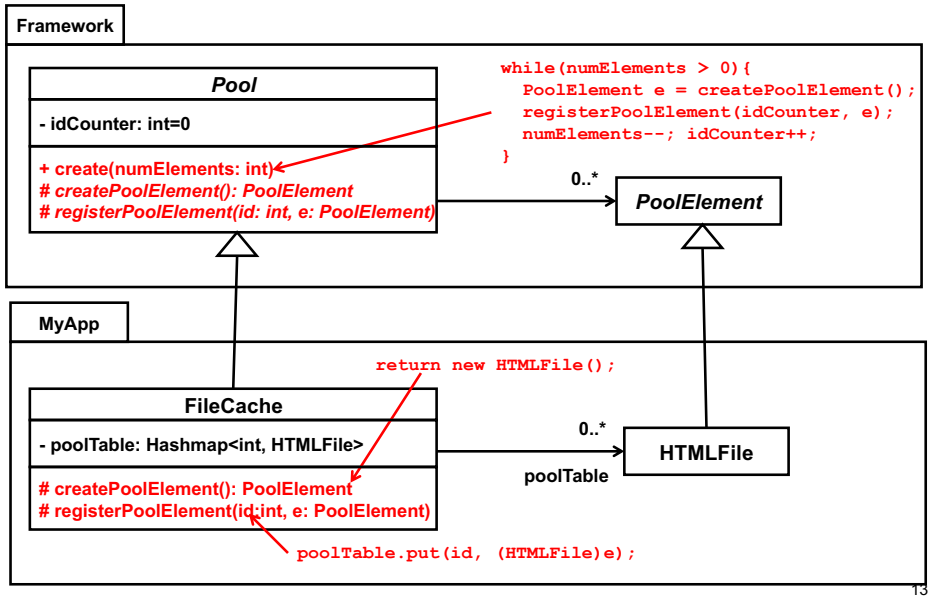


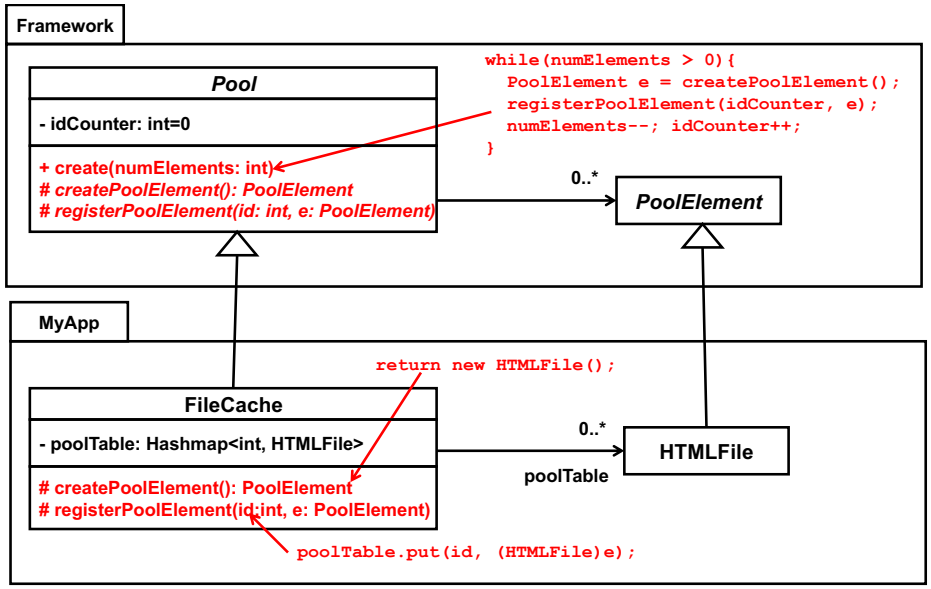
- Assume you are implementing an object pooling API in this framework.
  - Creates and manages a pool of (the same kind/type of) objects
    - e.g., a pool of browser windows, a pool of tabs in each browser window, a pool (cache) of HTML files, a pool of HTTP connections, a pool of threads, etc.
- Here, we focus on the *creation* of pools.



Factory Method in Object Pooling







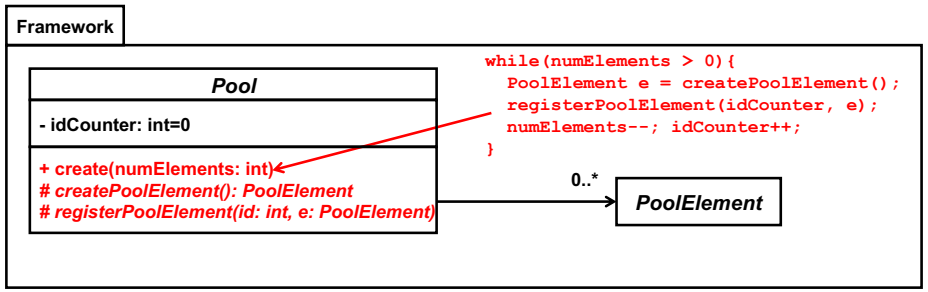

```
FileCache cache = new FileCache();
cache.create(10);
```


What's the Point?

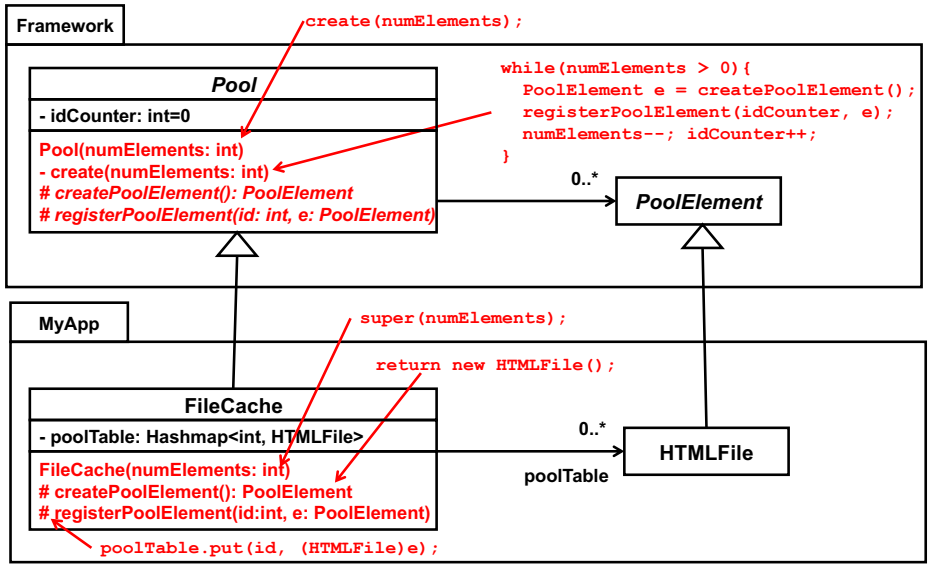
- The framework
  - Provides a skeleton (or template) for pool creation logic.
    - Partially implements the pool creation logic.
  - Never specify specific types (specific class names) for a pool and its elements
- MyApp (framework client)
  - Reuses the skeleton/template of pool creation logic and completes it
    - By specifying which pool class and which pool element class are used.

Factory Method

- createPoolElement()
  - Allows create() to avoid stating the class name for pool elements.
  - Allows the framework to be independent (or de-coupled) from individual applications (framework clients).
    - Allows applications to be pluggable to the framework.



An Alternative Design

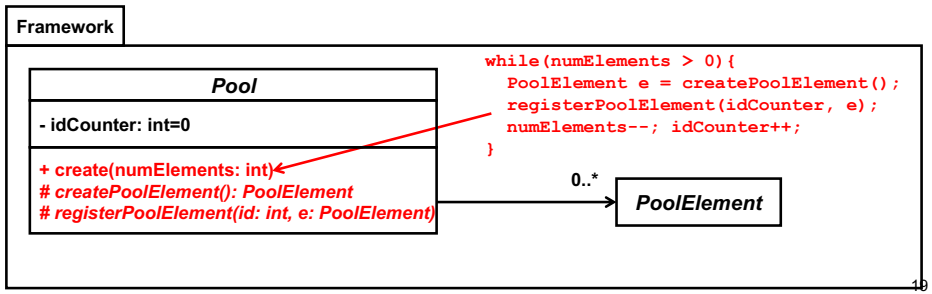


Client code gets simpler: `FileCache cache = new FileCache(10);`

- Client code can be simpler in this alternative design
  - `FileCache cache = new FileCache(10);`
  - No need to call create() as in the previous design.

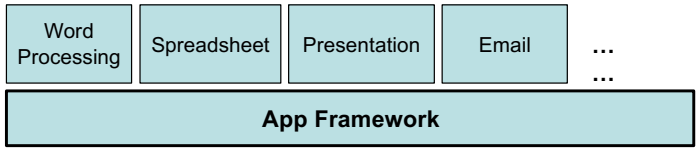
Other Considerations

- idCounter
  - Generics for Pool



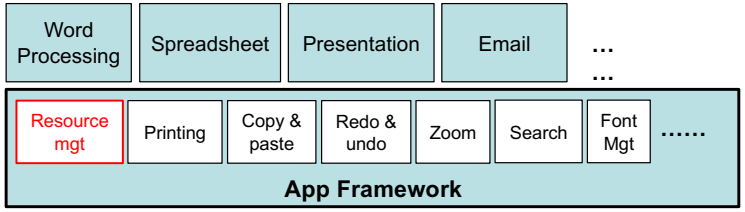
Another Example: A Framework for Productivity Applications

- Application framework
  - A set of foundation APIs to implement and run a series of applications.
    - Implement the standard/common functionalities (structures and behaviors) among individual applications
    - Make them reusable/available for individual apps.
    - Make app development easier and faster.
    -
  - Frameworks for productivity (“office”) applications
    - e.g., .Net Framework, Microsoft Foundation Class (MFC), Cocoa, OpenOffice Framework, GNOME, KDE, etc.

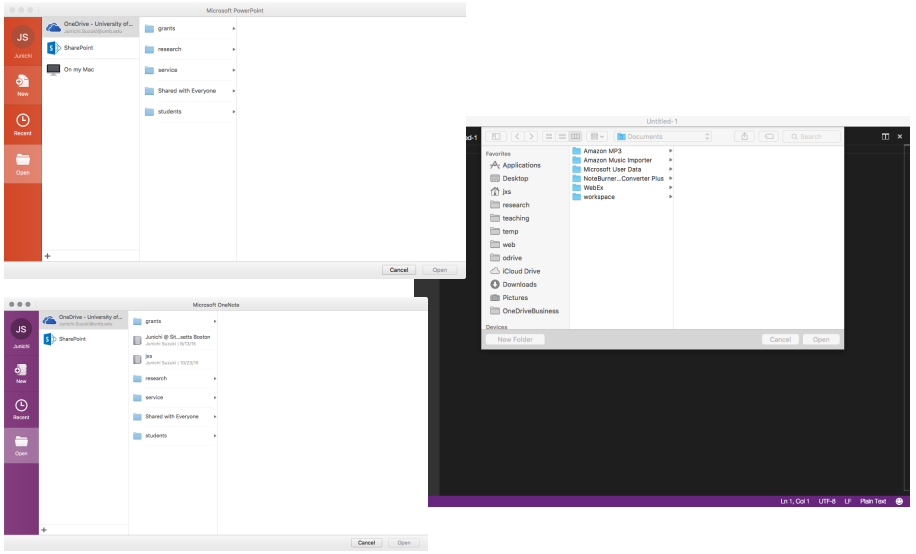


Resource Mgt in App Framework

- Resource management
  - Creating, opening and closing resources (e.g. documents, spreadsheets, presentation sheets, emails and notes)
  - Saving resources in the local disk or a remote cloud.
  - Exporting resources as other formats.
- Here, we focus on the logic of ***creating*** resources.



In Microsoft Applications…

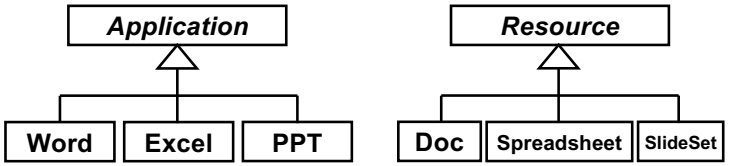


Assumptions

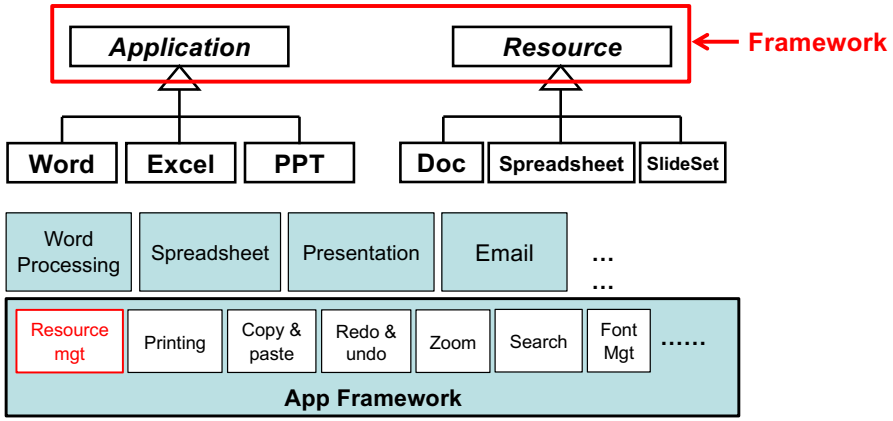
- Multiple applications exist.
  - Extra applications may be developed in the near future.
  - Different applications create and use different types of resources.
  - When an application creates a new resource, it opens a blank resource.
  - Each application creates one resource at a time, but can keep multiple resources open.
  - Each application records the list of resources that it opened recently.

Major Players: Apps and Resources

- Multiple applications exist.
  - Extra applications may be developed in the near future.
- Different applications create and use different types of resources.



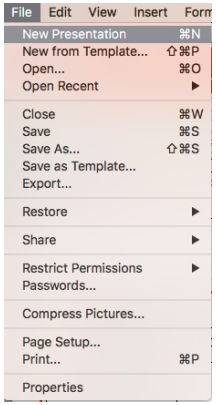
- When an application creates a new resource, it opens a blank resource.
- Each application creates one resource at a time, but can keep multiple resources open.
- Each application records the list of resources that it opened recently.

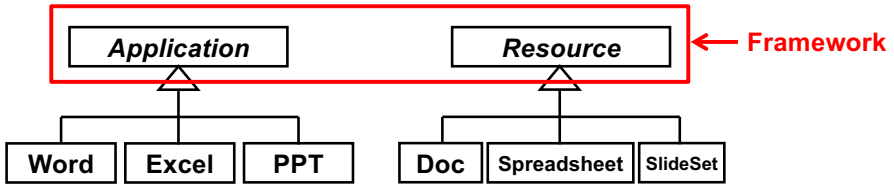


- When an application creates a new resource, it opens a blank resource.
  - Word should create a document.
  - Excel should create a spreadsheet.
  - PPT should create a slide set.

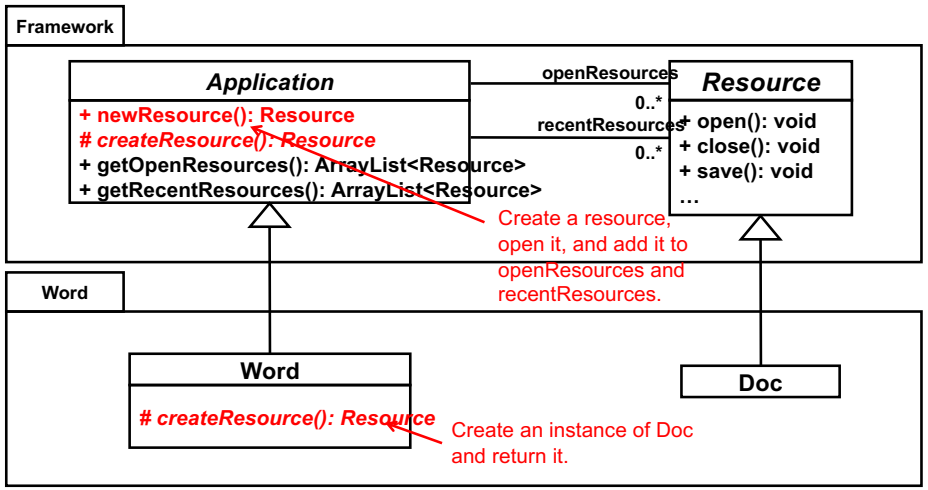
– Extra applications may be developed in the near future.
  - An app to be developed in the future should create a particular resource associated to that app.
    – We don't know about the app-resource pair now.

– How can we implement the *common creation logic* in the framework level (i.e. in Application) without knowing Application's subclasses and Resource's subclasses?

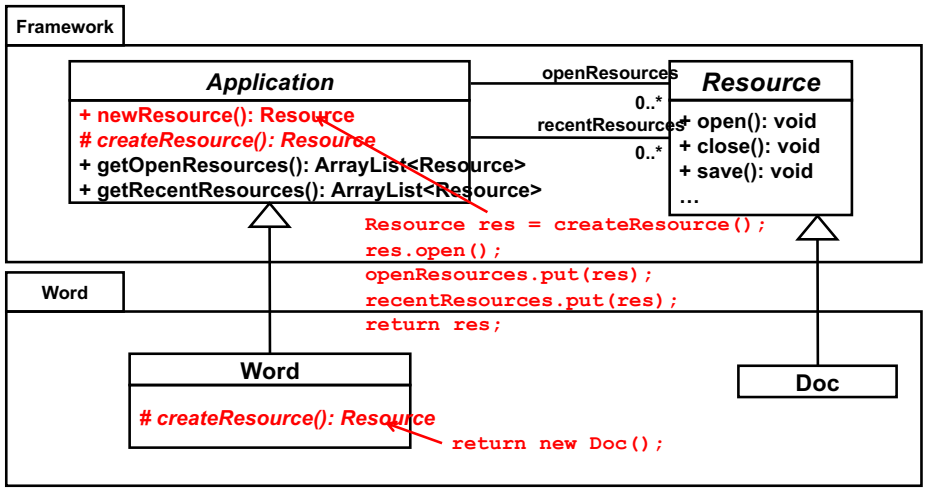




Solve this Design Issue with *Factory Method*



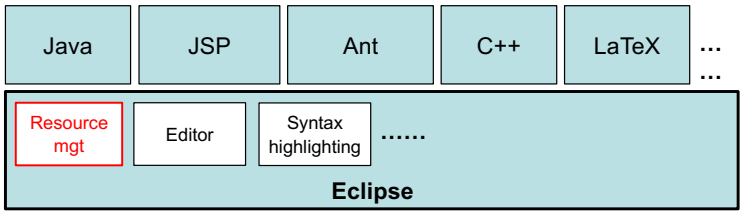

```
Word word = new Word(…);
word.newResource();
```

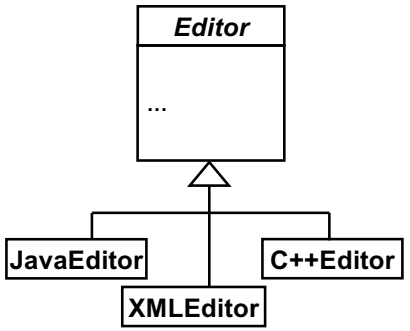

```
Word word = new Word(…);
word.newResource();
```

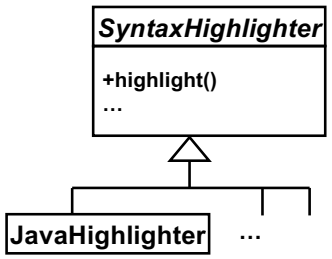
One More Example: Resource Mgt in Integrated Development Environments (IDEs)

- Imagine an IDE like Eclipse, for example.
- Resources in an IDE
  - Programs (e.g., Java, JavaScript, C++, etc.)
  - XML files (e.g., build.xml for Ant, web.xml for Servlet WAR)
  - …, etc.
- Many IDE components (plugins) use resources.
  - Editors, syntax highlighters, etc.

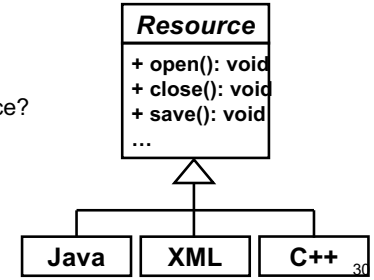


Editors, Syntax Highlighters and Resources

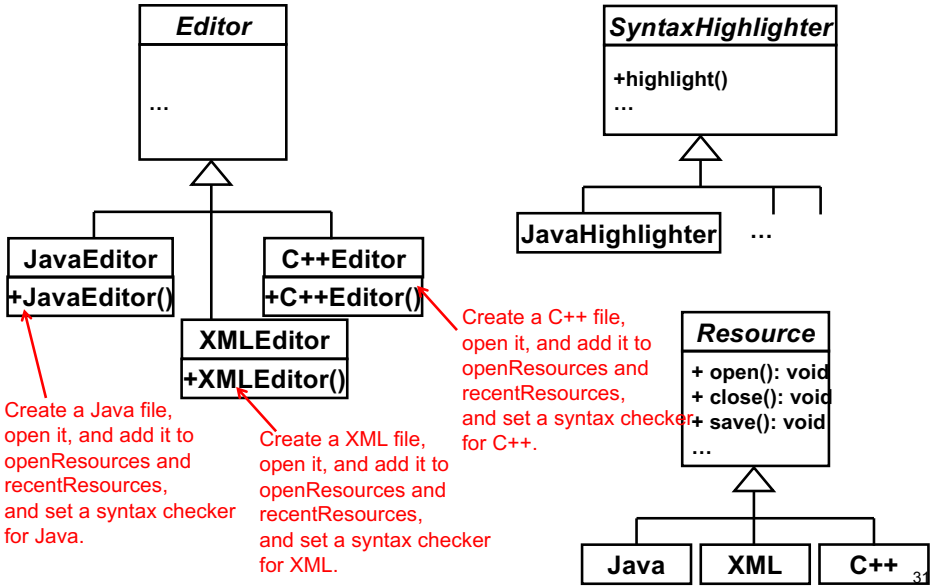




How does an editor create a resource?
How does it create a syntax highlighter for the resource?



Without *Factory Method*

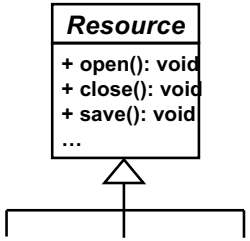


- This IDE
  - Is not that developer friendly.
    - Requires developers to write redundant code for their editors.
  - Can be more developer friendly
    - By defining a common sequence (or skeleton/template) to create and initialize a resource in Editor.
      - Does not have to require developers to write redundant code.

Dilemma

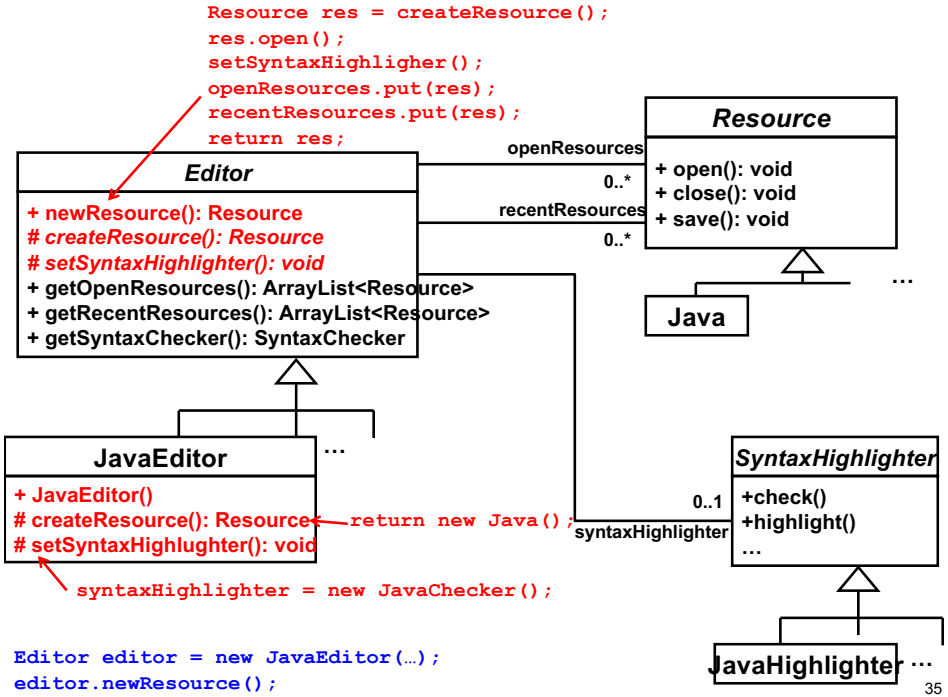
How can we implement a common sequence (skeleton or template) for instance creation and initialization in Editor **WITHOUT** knowing specific subclasses of Editor, Resource and SyntaxHighlighter AND relationships among them?

JavaEditor – Java – JavaHighlighter
XMLEditor – XML – XMLHighlighter



With *Factory Method*

- Define a factory method in Editor
- Have it implement a common sequence (skeleton or template) for instance creation and initialization with *empty protected methods*.



Benefits

- This IDE
  - Can define a common sequence for instance creation and initialization
  - Allows individual editors to reuse it.
    - Less redundant code
  - Can “force” every editor to follow the same behavior (i.e. same sequence for instance creation and initialization) when it creates a new resource.