

Design Patterns In Java

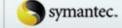
Design Patterns Overview











Chapter content





- Design Patterns Overview
- Motivation
- Notes
- Basic Design Concepts
- Preliminary Example
- Design Patterns Taxonomy

Overview



"Most papers in Computer Science describe how their author learned what someone else already knew" Peter Landin

- DP's aim is to catalog common object structures/ interactions that programmers have found useful.
- Learn to recognize problems that fall into familiar categories.
- Rely on literature such as Design Patterns -Elements of Reusable Software Gamma et-al.

Motivation



Less re-inventing & re-thinking:

Decome familiar with common solutions and trade-offs associated with them.

>Humility:

learn from the accumulated experience of others.

>flexible programs:

Use it to create sophisticated, easy-tomaintain

A common vocabulary:

amongst guild members.

Motivation - cont.



When maintaining code written by others, learn to recognize (and perhaps criticize) the design choices of other programmers.

Note



There is, of course, more to design than fancy patterns...

- Organization skills, look-ahead, commons sense may be more valuable than knowledge of some rarely-used pattern
 - > Proper system analysis & spec review.
 - > Maintainable, reusable, configurable code.
 - > Avoiding over-design. Knowing when to prefer simplicity over complex structures.
 - Exploiting available human/programming resources.
 - Avoiding assignment of too much/too little responsibility per class, method or programmer.

Note - cont.



This course, however, focuses on Patterns.

- Design Patterns may help achieve the above mentioned goals, e.g:
 - > Factory for configurability.
 - > Bridge for re-usability & dividing class responsibilities.
 - Strategy for avoiding code duplication.
 - Facade for simplifying code interface.

Basic Design Concepts



Separate abstract requirements (interface) from implementation. Program to an interface, not to implementation!

Isolate the minimal factor of change

- Separate factors that change from those that can be re-used.
- 2. Avoid code duplication.
- 3. When adding extensions, minimize the need to edit the existing code.

Basic Design Concepts - cont. In



1. Remember your code will most probably change (over time).

Usually assume the worst!

1. Quite often we will favor object composition over inheritance.

Preliminary Example



Code duplication:

```
public class FileUtils {
   // DELETE file, or directory+entire content:
    public void delete(File file) throws Exception{
         if (file.isDirectory()){
            File[] children = file.listFiles();
           for(File child : children)
              delete(child);
         file.delete();
    }
    // Declare READ-ONLY for file, or for directory+entire content:
    public void setReadOnly(File file) throws Exception{
    // same recursion, just call "setReadOnly" instead of "delete"
```

Preliminary Example - cont.

// cont ->



Isolate the minimal change factor:

```
public interface FileAction {
    public void perform(File file) throws Exception;
public class FileUtils {
    public void recurse(File file, FileAction action) throws Exception{
        if (file.isDirectory()){
            File[] children = file.listFiles
            for(File child : children)
               recurse(child, action);
                                                            The Command Pattern
                                                            (Some may call it Visitor,
         action.perform(file);
                                                            though it doesn't match the
                                                            classical Visitor definition of
```

overcoming the lack of **Multiple Polymorphism**)

Preliminary Example - cont.



```
// FileUtils - cont'd:
private static FileAction DEL_ACTION = new FileAction(){
    public void perform(File file) throws Exception {
        file.delete();
    }
};
The Facade Pattern
public void delete(File file) throws Exception {
    recurse (file, DEL_ACTION);
}
```

- Is our recurse() method general enough?
- Is it always worth the trouble?

Design Patterns Taxonomy



- We shall discuss 23 patterns, divided into three categories:
 - Creational patterns
 - > strategies for creating new objects (instances).
 - Structural patterns
 - how objects may be grouped into more complex structures.
 - > Behavioral patterns
 - define the flow of communication between objects.