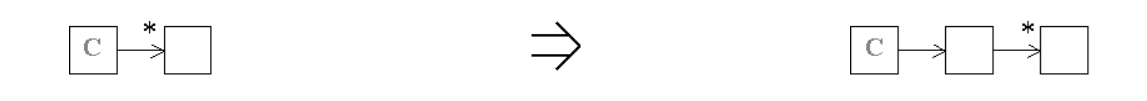
**Encapsulate Multiplicity**

|  |  |  |
| --- | --- | --- |
| *a class it's inner logic has become complex due to the multiplicity of components* |  | *extract the logic for dealing with the multiplicity of components and bring it into a class of its own* |



**Context:**

When a client's code has become complex and cluttered due to the multiplicity of its components [Encapsulate Multiplicity](https://web.archive.org/web/20090914100954/http:/www.refactoring.be/thumbnails/rtn-em.html) often helps to simplify the client's code.

[Encapsulate Multiplicity](https://web.archive.org/web/20090914100954/http:/www.refactoring.be/thumbnails/rtn-em.html) is an example of [Introduce Indirection](https://web.archive.org/web/20090914100954/http:/www.refactoring.be/thumbnails/rtn-ii.html) where we use the indirection level to deal with the extra complexity of having a multitude of components locally.

When [Evolving to the Composite Pattern](https://web.archive.org/web/20090914100954/http:/www.refactoring.be/thumbnails/ec-composite.html) this refactoring prepares a design for [Enable Component Subcomponent Substitution](https://web.archive.org/web/20090914100954/http:/www.refactoring.be/thumbnails/rtn-ecscs.html).