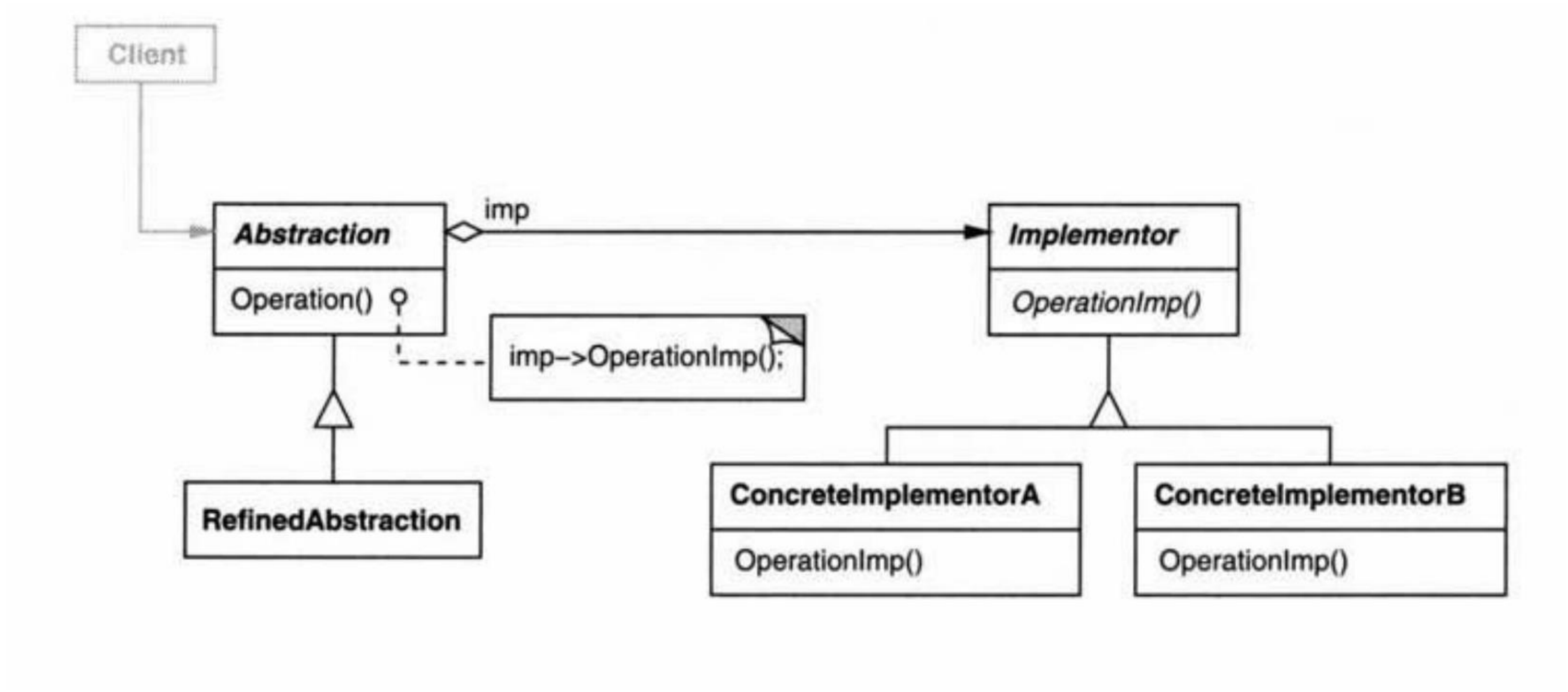
Bridge Pattern

Bridge Pattern bridges abstraction and it's implementation, letting them vary independently.

Use cases

- separate API from the implementation - hiding implementation to clients,
- decoupling interface and implementation,
- open for changes,
- Changes/new functionalities don’t affect existing code.

Bridge pattern structure (*Design Patterns: Elements of reusable software*)



Elements

Abstraction: Defines API used by client. Maintenance reference to the implementator.

Implementator: Provides rules for concrete implementators.

Concrete Implementator: Concrete representation of the Implementator.

Elements

Refined Abstraction: implements abstraction. Add specific implementation one level higher.

- Can implements low level functions from abstractions and run inside some functions specified by the implementator.
- Can implements high level functions from abstraction and add high level functionality which is different than in implementator.

Real life examples

- Socket class from java.net
- Button from java.awt