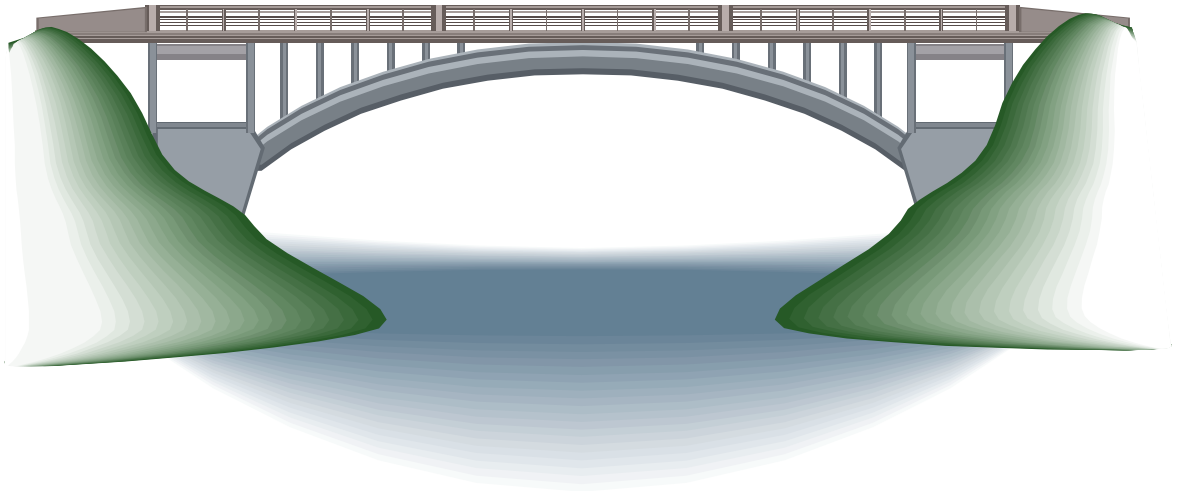
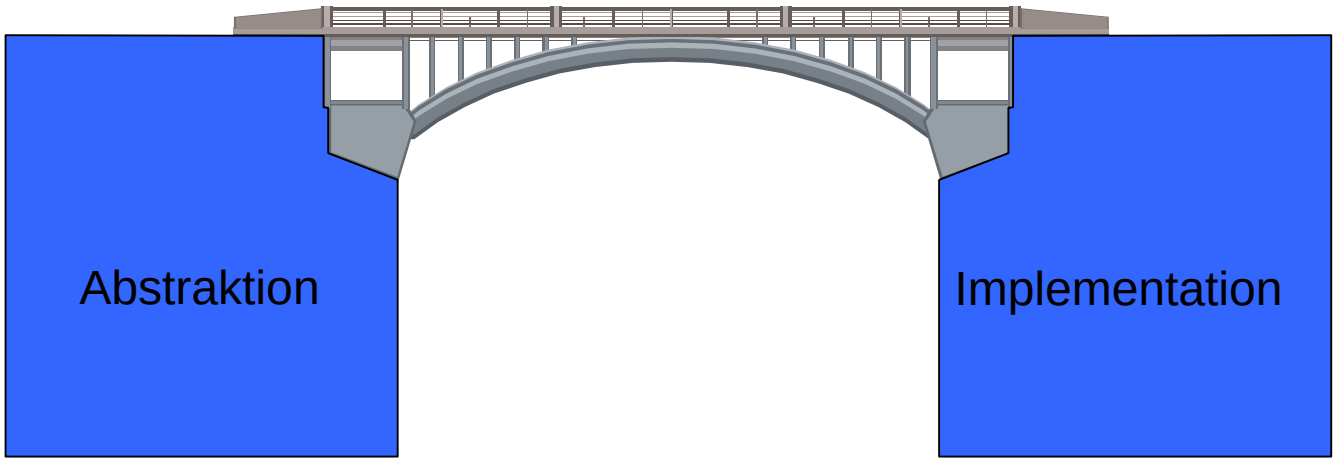
Bridge

(auch bekannt als Handle oder Body)



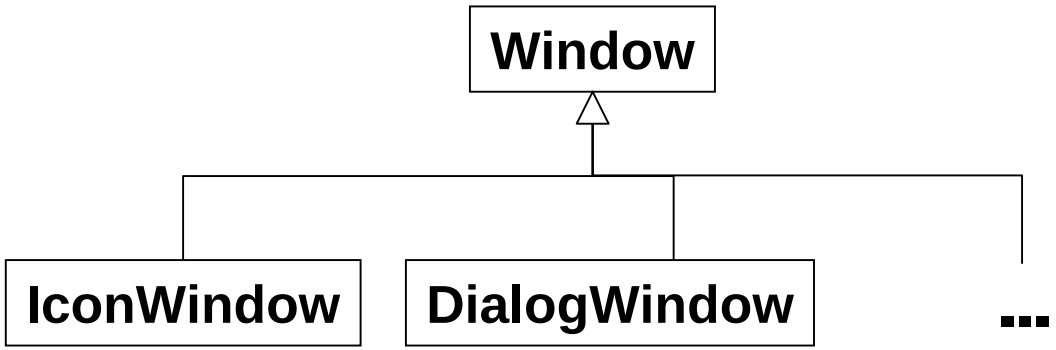
Die Bridge – Überblick

- Name: Bridge, Handle, Body
- Ziel: Trennung von Abstraktion und Implementation, um beiden eine unabhängige Weiterentwicklung zu ermöglichen.
- Einordnung: Strukturelles Pattern, arbeitet mit Objekten.
- Variabler Aspekt: Implementierung



Die Bridge – Ausgangspunkt

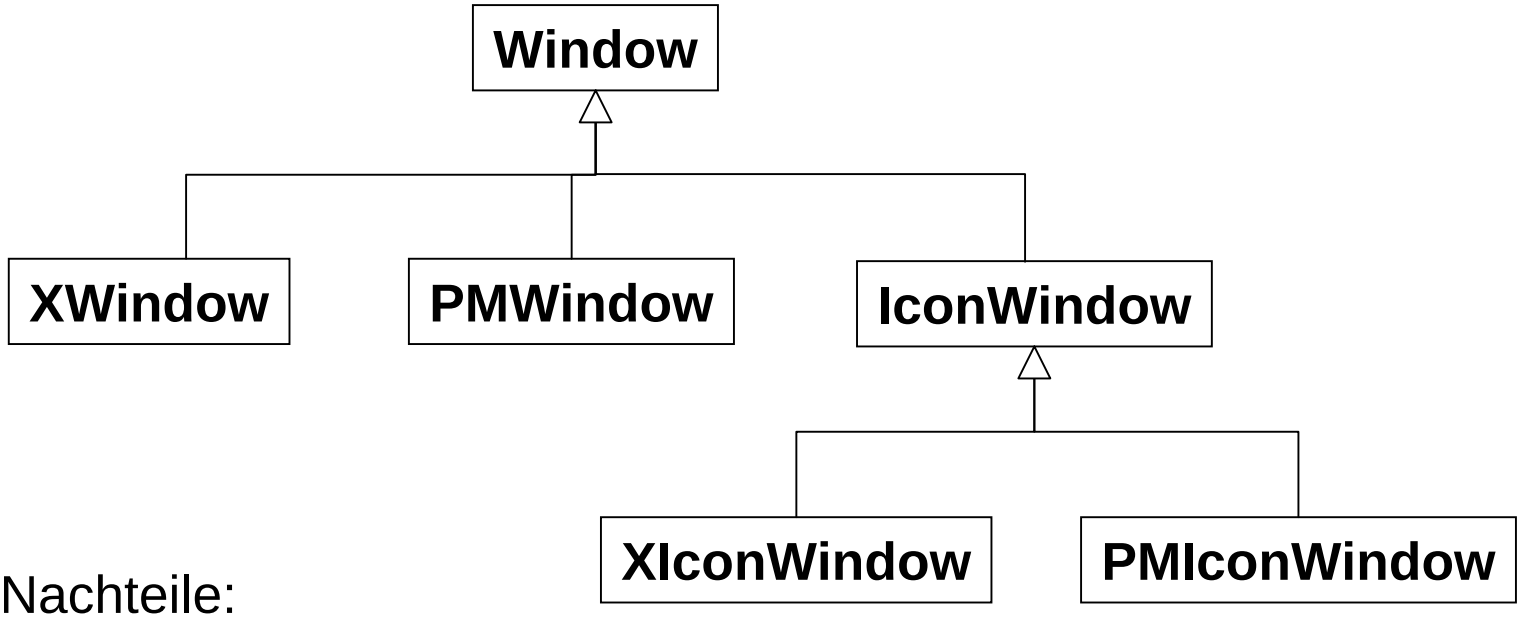
- Beispiel (stark) angelehnt an das, aus dem GOF Buch.
- Für die Entwicklung eines Editor werden eine Klasse **Window** und diverse Subklassen wie **IconWindow**, **DialogWindow** usw. benötigt.



- Soweit noch kein Problem...

Die Bridge – Das Problem

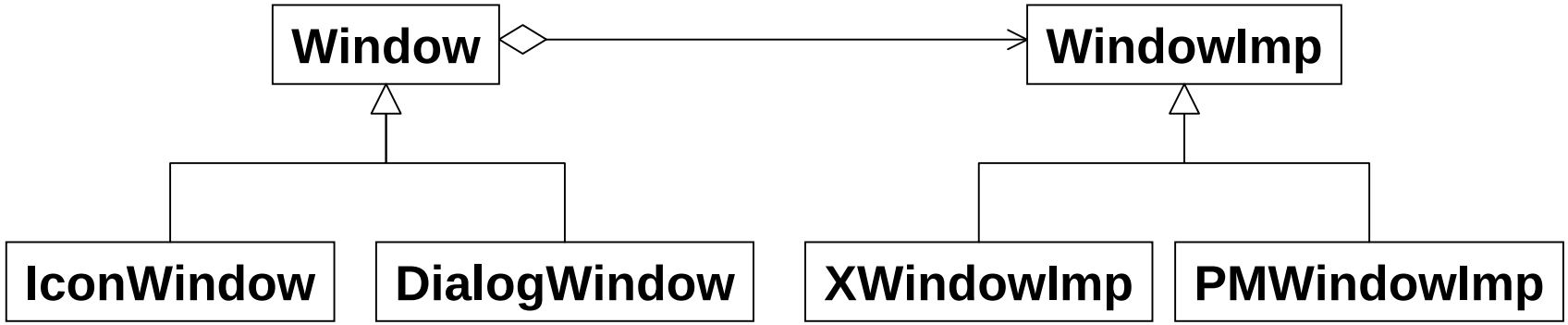
- Der Editor soll nun in verschiedenen Windowsumgebungen (X Windows, PM) laufen, **ohne jeweils individuell kompiliert werden zu müssen**.



- Nachteile:
  - Erweiterungen, wie Subklassen machen den Code schnell zu kompliziert, da für jede Umgebung je eine eigene Subklasse nötig wird
  - Der Code wird Plattformabhängig, dies belastet den Entwickler unnötig, und Portierungen oder Unterstützung für neue Umgebungen werden schwierig.

Die Bridge – Die Lösung

- Das Bridge Pattern bietet eine Lösung: es trennt die Abstraktion von der konkreten Umsetzung: Sie werden in verschiedene Klassen-hierachien gespalten.

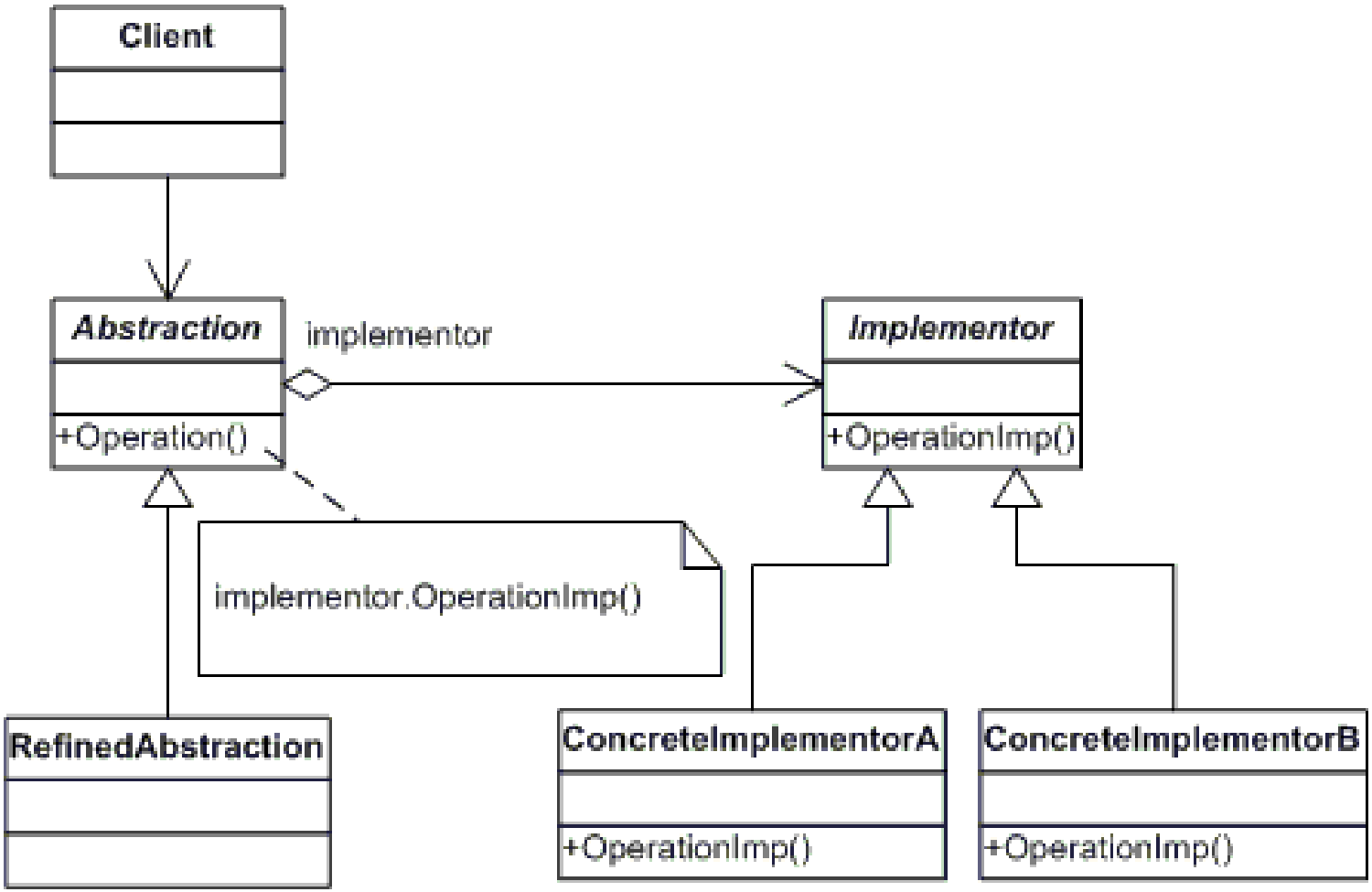


Die Bridge – Vorteile

- Änderungen der Umgebung und neue Plattformunterstützung ist leichter umzusetzen.
- Der Entwickler kann sich auf die Arbeit mit dem Modell konzentrieren, keine „Verunreinigung“ mit spezifischen, abhängigen Details.
- Die Bindung an eine konkrete Implementierung ist dynamisch, und kann sogar während der Laufzeit geändert werden.
- Beide Klassen können unabhängig voneinander erweitert werden.
- Details der Implementierung werden vor dem Client verborgen.

→Letztendlich ist die Bridge eine weitere Abstraktion und hebt damit das Modell auf eine höhere, überschaubarere Stufe und beseitigt Abhängigkeiten (wahrt Orthogonalität).

Die Bridge – Kontext



Quelle: http://www.dofactory.com/patterns/PatterBridge.aspx am 22.04.2003

Die Bridge – Kontext

- Abstraction definiert das Interface, und hält eine Referenz auf Implementor.
- RefinedAbstraction erweitert das Interface zu einer konkreteren Subklasse.
- Implementor definiert das Interface für die implementierende Klasse, es kann sich massiv von dem Interface von Abstraction unterscheiden.
- Typischerweise stellt Implementor nur primitive Funktionen zur Verfügung, während Abstraction eine Schnittstelle auf höherem Niveau bildet.
- ConcreteImplementor ist die Implementierung des Interfaces namens Implementor.
- Achtung, nicht durch den Namen täuschen lassen: Implementor ist von der Semantik her ein Interface!

Die Bridge – Anwendungsbereiche

Das Bridge Pattern kann eingesetzt werden, wenn:

- permanente Bindung an eine Implementierung vermieden werden soll.
- sowohl die Abstraktion als auch Implementierung erweiterbar sein sollen.
- Änderung der Implementierung (z.B. Plattform) keine Änderungen im Client notwendig machen soll.
- Eine Implementierung von verschiedenen Objekten geteilt werden soll.

Die Bridge – Umsetzung

- Wenn nur ein Implementor vorhanden ist, ist das Bridge Pattern grundsätzlich unnötig; die zusätzliche Komplexität bringt zu wenig/keine Vorteile.
- Ein Ausnahme ist es, wenn Änderungen der Implementierung den Client nicht betreffen sollen.
- Kritisch ist auch die Frage wann, wo und wie entschieden wird welche Implementierung genutzt wird. Möglichkeiten sind u.A. Delegation an ein anderes Objekt, oder im Konstruktor von Abstraction.
- Bei James O. Coplien, „Advanced C++ Programming Styles and Idioms“ finden sich Beispiele wie der Implementor von mehreren Objekten geteilt werden kann.