Design Patterns im Rahmen des objektorientierten Designs

Hans Fangohr

2016-02-01

http://github.com/uni-wuppertal/SE-designpattern

Outline

Einführung

Lernziele

- Sie können die Idee von Design Pattern erklären,
- Sie sind mit mindestens zwei Design Pattern vertraut, und können diese an einem Beispiel erläutern,
- Sie fangen an, einen Überblick von klassischen OO Design Pattern zu entwickeln, und
- Sie können Literatur benutzen, um selbständig weitere Design Pattern zu erlernen.

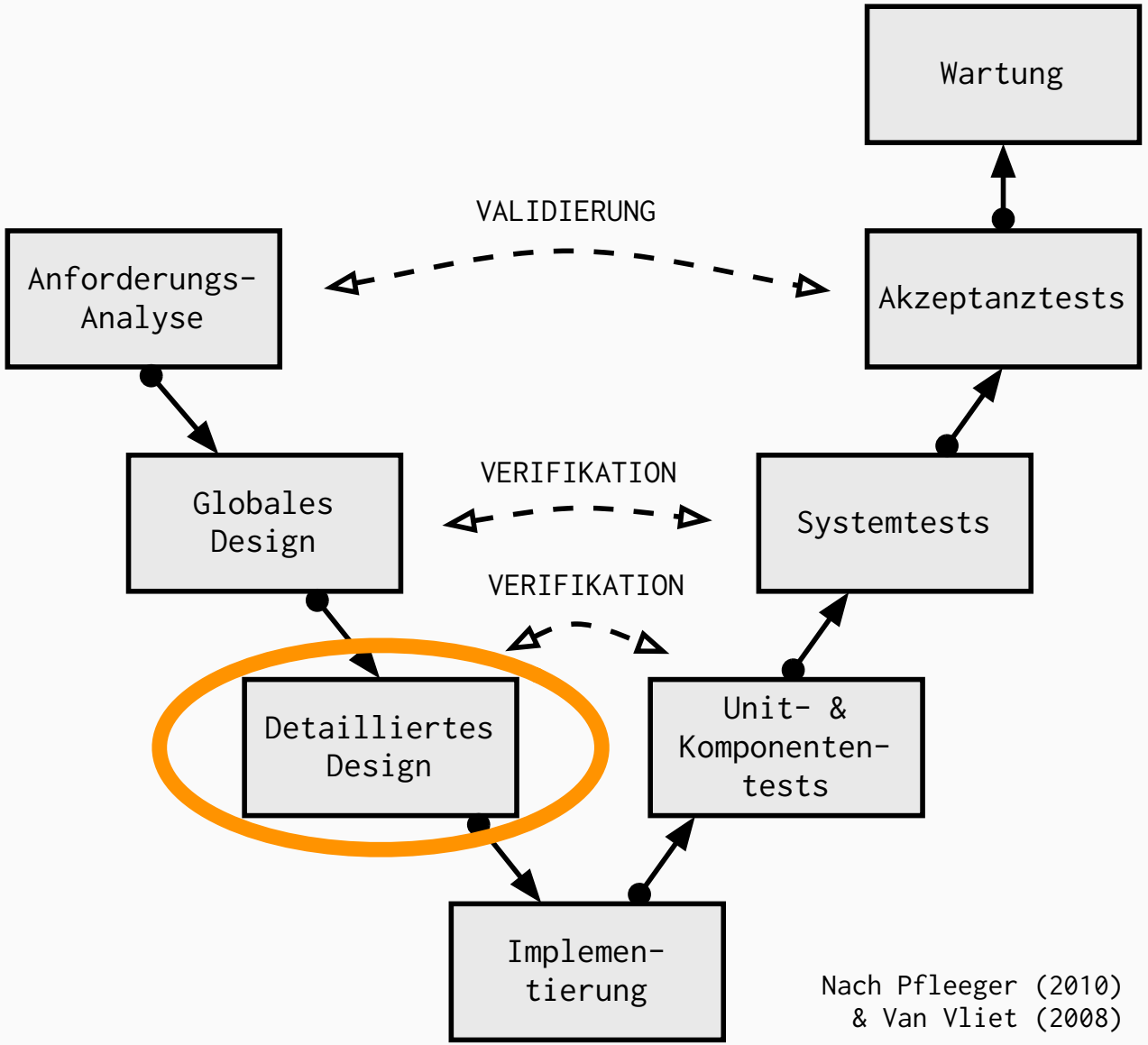
Was ist ein Entwurfsmuster ("Design Pattern")?

Christopher Alexander (Architekt), 1977:

> *Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*

- Alexander bezog sich auf Gebäude und Städte
- Wir sprechen über Objekte und Interfaces anstelle von Wänden und Türen
- Kernidee: Allgemeine Lösung für ein wiederkehrendes Problem in speziellem Kontext

Wo im Softwarezyklus sind Design Pattern relevant?



Nach Pfleeger (2010)
& Van Vliet (2008)

Was sind Design Patterns im objektorientierten (OO) Design?

Design Patterns

- sind generelle, wiederverwendbare Lösungen für häufig auftretende Entwurfsprobleme im Softwaredesign,
- beschreiben interagierenden Klassen und Objekten,
- stellen getestete Wissen zur Verfügung, und können den Entwurfprozess beschleunigen

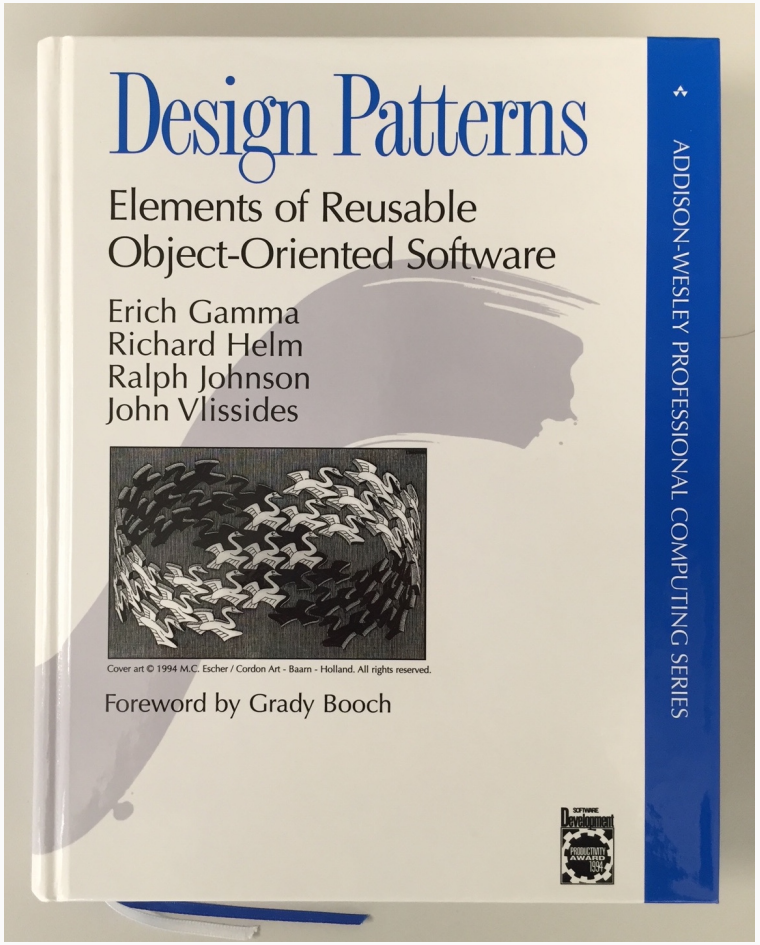
Design Patterns

- sind keine fertigen Entwürfe, die nur noch implementiert werden müssen

Design Patterns Buch

Design Patterns – Elements of Reusable Object-Oriented Software (1995)

- Autoren werden oft als "Gang of Four" (GoF) beschrieben.
- Anfang der Design Patterns in Objektorientiertem Design



Beispiel

Beispiel: Sortierproblem 1/2

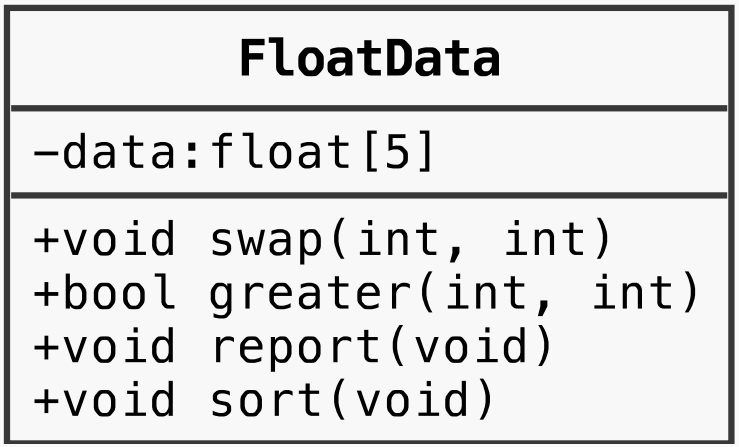
Anforderungen für `FloatData` Klasse:

- 5 Gleitkommazahlen speichern, mit Index 0 bis 4
- `swap(i, j)` Methode: zwei Zahlen mit Index `j` und `j` vertauschen
- `greater(i, j)` Methode: `True` falls Zahl mit Index `i` groesser als Zahl mit Index `j`
- `report()` Methode: Bericht mit Zahlen und Reihenfolge
- `sort()` Methode: sortiert die 5 Elemente

Beispiel: Sortierproblem 1/2

Anforderungen für `FloatData` Klasse:

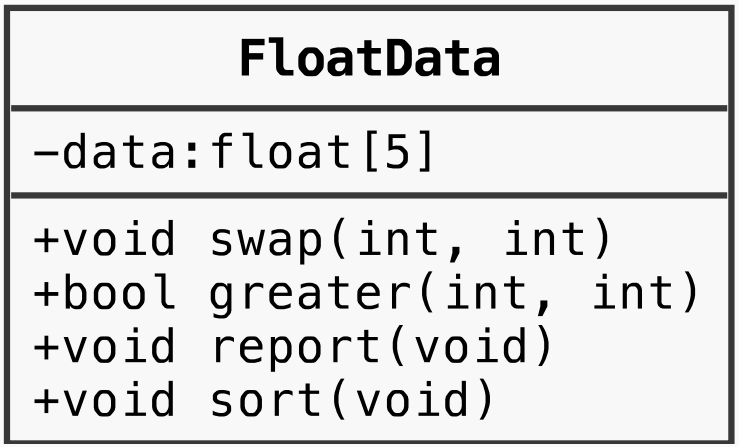
- 5 Gleitkommazahlen speichern, mit Index 0 bis 4
- `swap(i, j)` Methode: zwei Zahlen mit Index `j` und `j` vertauschen
- `greater(i, j)` Methode: `True` falls Zahl mit Index `i` groesser als Zahl mit Index `j`
- `report()` Methode: Bericht mit Zahlen und Reihenfolge
- `sort()` Methode: sortiert die 5 Elemente



Beispiel: Sortierproblem 1/2

Anforderungen für `FloatData` Klasse:

- 5 Gleitkommazahlen speichern, mit Index 0 bis 4
- `swap(i, j)` Methode: zwei Zahlen mit Index `j` und `j` vertauschen
- `greater(i, j)` Methode: `True` falls Zahl mit Index `i` groesser als Zahl mit Index `j`
- `report()` Methode: Bericht mit Zahlen und Reihenfolge
- `sort()` Methode: sortiert die 5 Elemente



Example implementation (C++) in `https://github.com/uni-wuppertal/SE-designpattern/blob/master/Code/sort5-floatonly.cpp`

Beispiel: Sortierproblem 2/2

Zusatz: wir brauchen die gleichen Fähigkeiten für Integerdaten

FloatData
`data: float[5]`

`void swap(int, int)`
`bool greater(int, int)`
`void report(void)`
`void sort(void)`

IntData
`data: int[5]`

`void swap(int, int)`
`bool greater(int, int)`
`void report(void)`
`void sort(void)`

Beispiel: Sortierproblem 2/2

Zusatz: wir brauchen die gleichen Fähigkeiten für Integerdaten

FloatData
`data: float[5]`

`void swap(int, int)`
`bool greater(int, int)`
`void report(void)`
`void sort(void)`

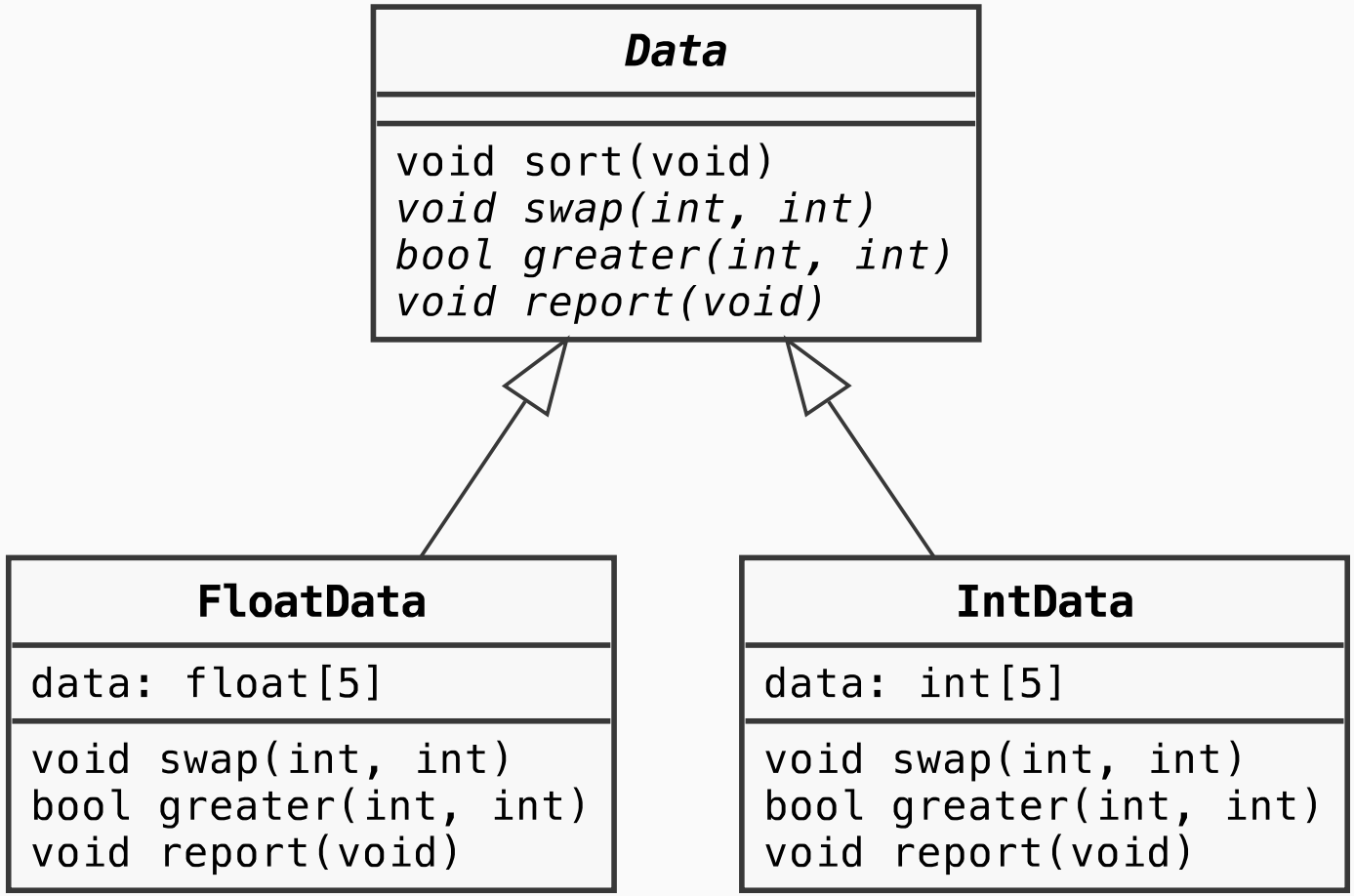
IntData
`data: int[5]`

`void swap(int, int)`
`bool greater(int, int)`
`void report(void)`
`void sort(void)`

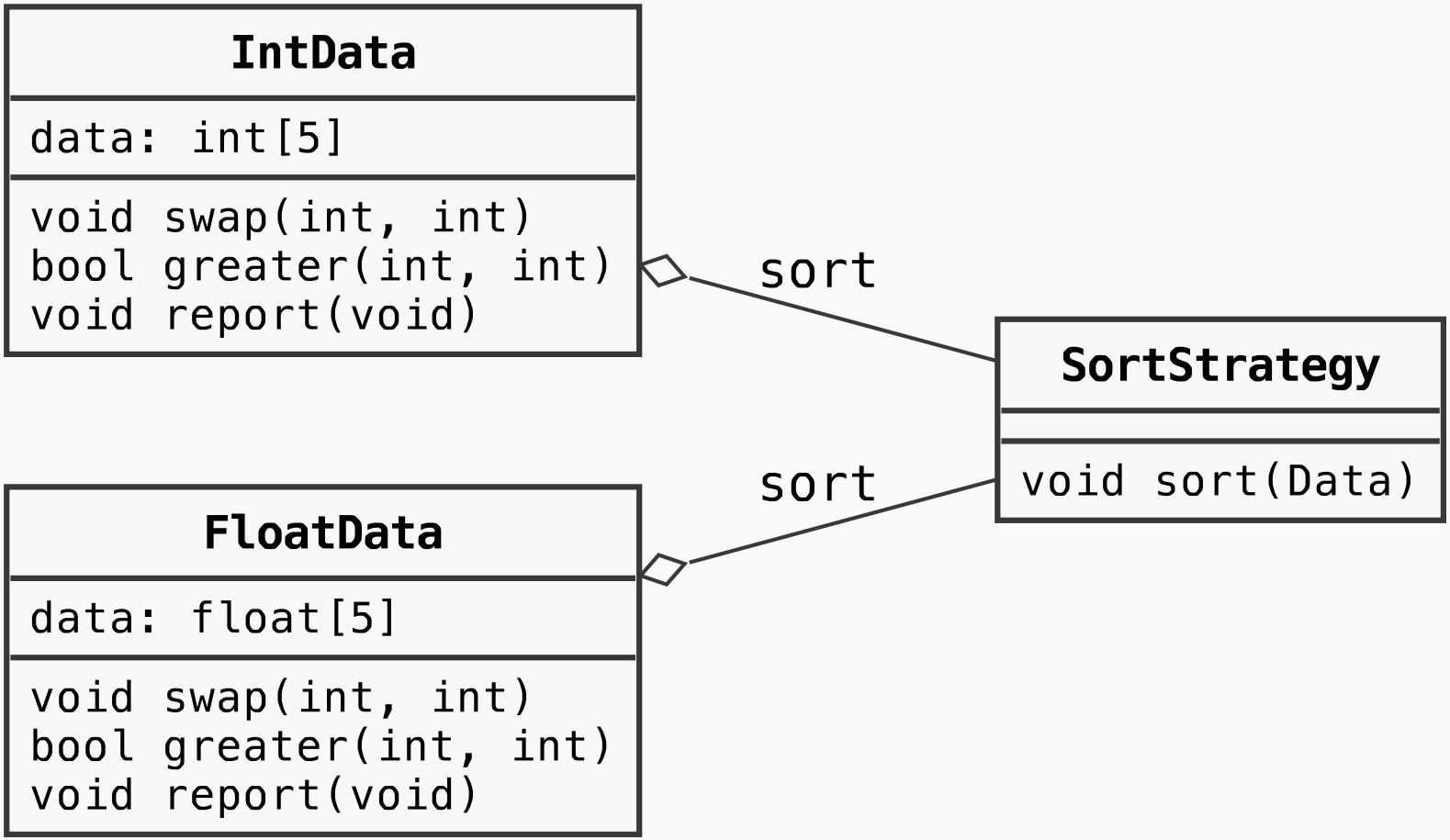
$\longrightarrow$ Duplizierung von Code

Wir nehmen an, dass der Sortieralgorithmus die grösste Komplexität hat.

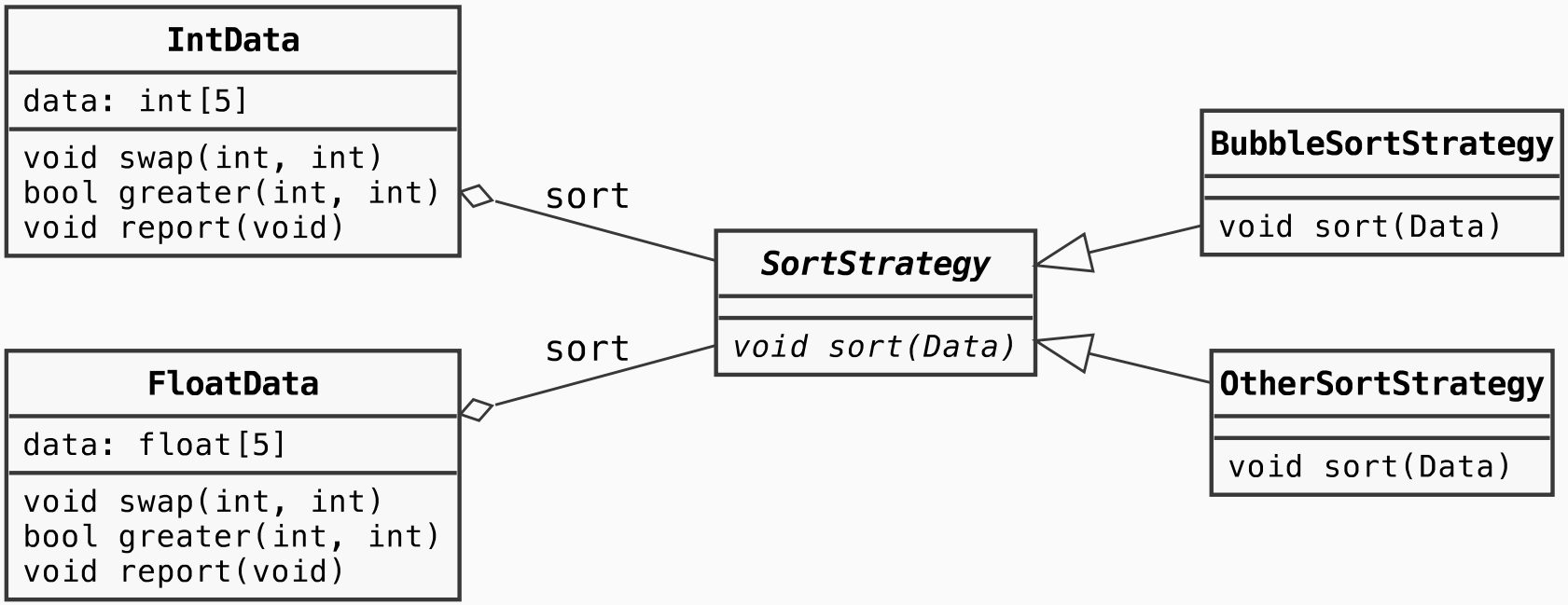
Entwurfmuster 1: Template Muster (Template Pattern)



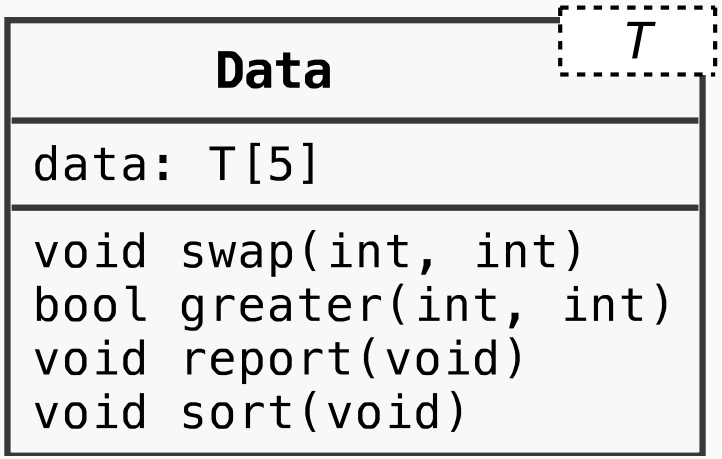
Entwurfmuster 2: Strategie Muster (Strategy Pattern)



Strategie Muster erlaubt verschiedene Strategien zu benutzen

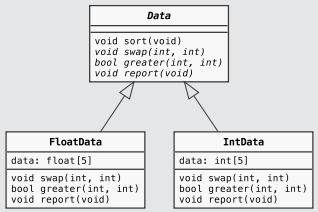


Alternative 3: Template Klasse (C++)



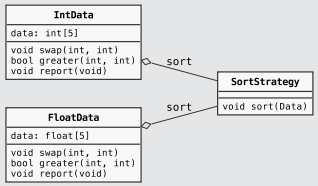
Zusammenfassung Sortierproblem

Templatemuster



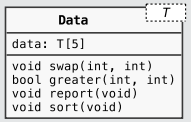
Generischer code in Basisklasse (`sort`), abgeleitete Klassen für spezifische Details

Strategiemuster



Benutze eine Klasse für generischen Sortiercode (die Strategie). Datenobjekte werden der Sortierklasse als Argument übergeben. Kann Strategie zur Laufzeit ändern.

Templateklasse



Benutze C++ Templateklasse für dieses besondere Problem.

Entwurfsmuster (Design Patterns)

Tabelle Design Patterns

	Erzeugende (Creational)	*Strukturelle (Structural)*	*Verhalten (Behavioral)*
Klassen-muster	Factory Method	Adapter (class)	Interpreter Template Method
Objekt-muster	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Tabelle 1.1 von GoF Buch

Gute deutsch-englisch Übersetzung:
`https://de.wikipedia.org/wiki/Entwurfsmuster_(Buch)`

Überblick Design Patterns

Eingeteilt in

- Erzeugungsmuster (creational patterns)
  - zB "Singleton": Eine Klasse für die nur eine einzige Instanz erzeugt wird (zB Logging, Datenbank, Hardware)
- Strukturmuster (structural patterns)
  - zB "Fassade": Eine Fassadeklasse agiert als vereinfachtes Interface zu komplexem Set von Klassen und Methoden
- Verhaltensmuster (behavioral patterns)
  - zB: "Mediator": Eine Vermittlerklasse agiert als Mittelsmann zwischen interagierenden Klassen, und erlaubt lose Kopplung zwischen den Klassen

Orthogonale Einteilung in

- Klassenmuster (Übersetzungszeit)
- Objektmuster (Laufzeit)

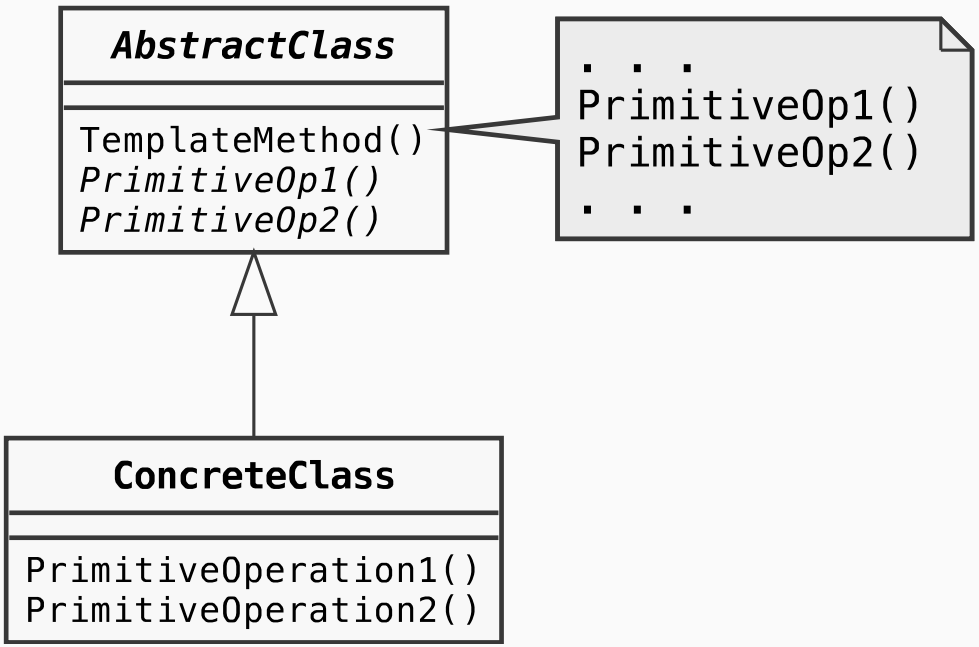
Kernbestandteile eines Designpatterns

Design Pattern enthalten viel Information. Typische Beschreibung durch:

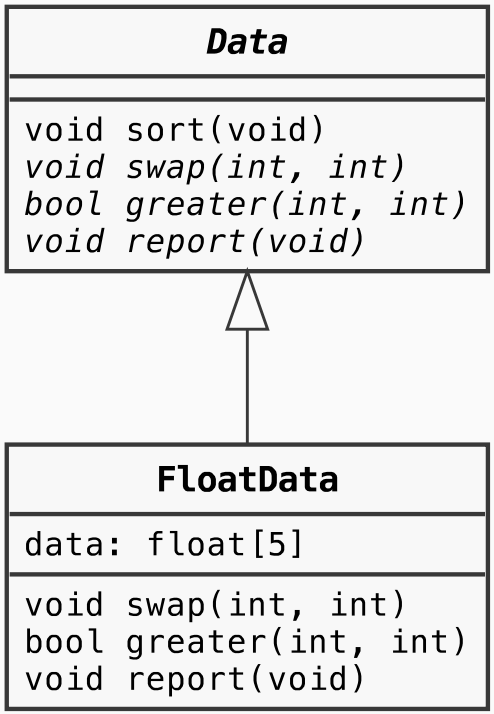
- Name des Pattern: möglichst beschreibend
- Struktur: UML Klassendiagramm
- Intention: Zusammenfassung
- Anwendbarkeit: beschreibt Situationen, in denen das Pattern verwendet werden kann
- Teilnehmer: beschreibt das Design und Zusammenwirken der Element des Patterns
- Konsequenzen: Vor- und Nachteile dieses Design Patterns
- Implementierung: (ggfs sprachspezifische) Hinweise zur Implementierung
- bekannte Anwendungsbeispiele

Template Muster 1/3

Allgemeine Struktur (GoF)



Unser Beispiel



Template Muster 2/3

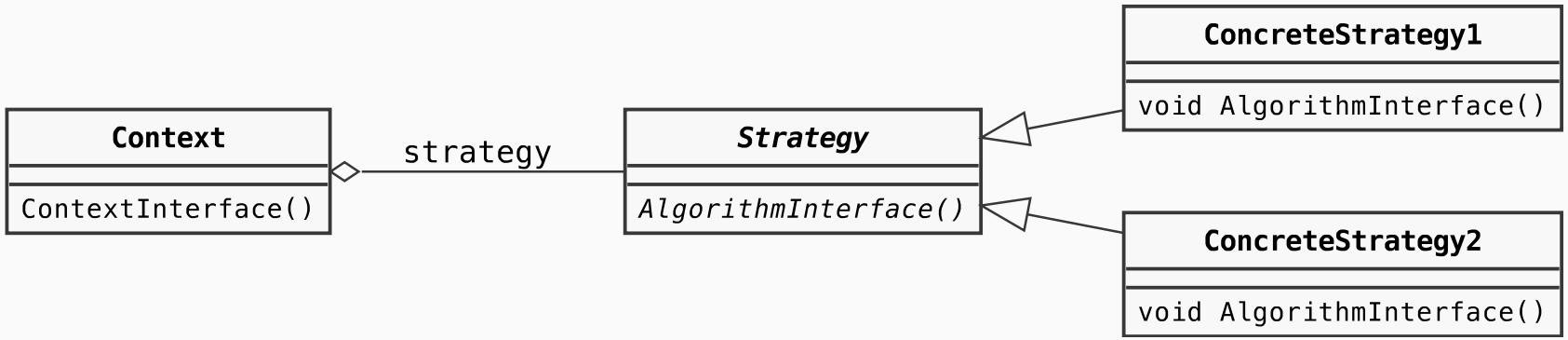
- Intention: Definiert das Skelett eines Algorithmus; lässt einige Schritte von Unterklassen ausführen.
- Anwendbarkeit:
  - implementiere invarienten Teil eines Algorithmus nur einmal, und Unterklassen implementieren abweichendes Verhalten
  - gemeinsamer Code in Subklassen soll reduziert werden ("refactor to generalize")

Template Muster 3/3

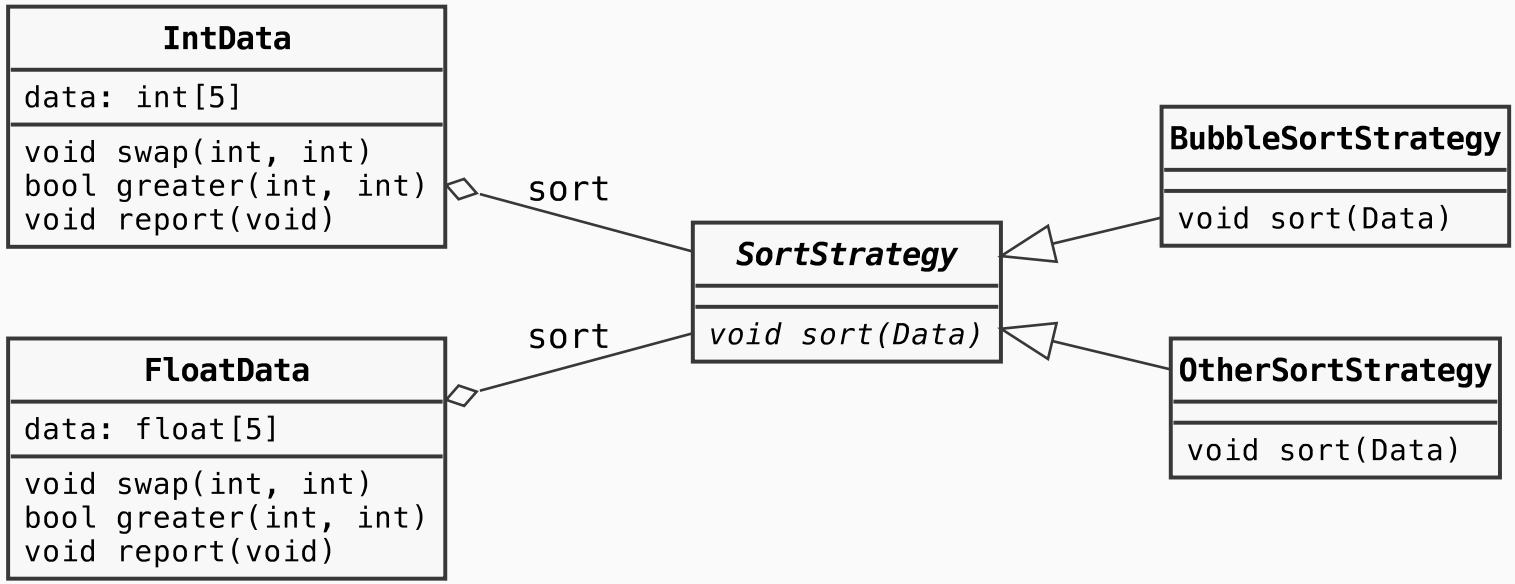
- Teilnehmer:
  - Abstrakte Klasse definiert abstrakte primitive Operationen, die von konkreten Unterklassen implementiert werden
  - Abstrakte Klasse definiert Templatemethode, die die primitiven Operation benutzt.
  - Konkrete Klasse: implementiert die primitiven Operationen die spezifisch für die Unterklasse sind
- Konsequenzen und Implementierung
  - Fundamentale Methode zur Wiederverwendung von Code
  - Unterklassen muss klar sein, welche Methoden überschrieben werden *können* und *müssen*.

Strategiemuster 1/3

Allgemeine Struktur Stragiemuster (GoF)



Unser Beispiel



Strategiemuster 2/3

- Intention: Definiere eine Familie von Algorithmen, verberge jeden in einer Klasse, und mache sie austauschbar: das Strategiemuster erlaubt, den Algorithmus unabhängig vom Klienten, der ihn benutzt, zu ändern.
- Anwendbarkeit:
  - wenn viele Klassen sich nur in ihrem Verhalten unterscheiden. Das Strategiemuster erlaubt eine Klasse mit vielen unterschiedlichen Verhaltensweisen zu konfigurieren
  - wenn unterschiedliche Varianten eines Algorithmus gebraucht werden.
  - wenn eine Klasse vielfache Verhaltensweisen definiert, und diese als wiederholte Fallentscheidungen auftauchen. Zusammenhängende Codefragmente für ein Verhalten formen dann eine Strategieklasse.

Strategiemuster 3/3

- Teilnehmer: (i) Strategieklasse (abstrakt) definiert Interface, (ii) konkrete Strategieklassen implementieren Algorithmus, (iii) Klientobjekt wird mit konkretem Strategieobjekt konfiguriert und stellt Interface für die Strategieklasse zur Verfügung.
- Konsequenzen:
  - Stragiemuster ist eine alternative zur Verberbung, und ist flexibler
  - Stragiemuster reduziert Fallunterscheidungen
  - Stragiemuster erlaubt verschiedene Implementierungen für das gleiche Verhalten (zB verschiedene Zeit- und Speicherbedarf)
  - Familien ähnlicher Algorithmen können hierarchisch geordnet werden
  - Klienten müssen Strategie wählen (und Strategien kennen)

Design Patterns stellen gemeinsames Vokabular zur Verfügung

- Natürliche Sprache gibt uns Namen für häufig benötigte Dinge und Konzepte: Haus, Siedlung, Geld, Steuererklärung, Fallstudie, ...
- Design Patterns geben uns ein Vokabular für gängige Entwurfsmuster in der Softwaretechnologie
- Sehr hilfreich für Kommunikation, Dokumentation und Entwurfdiskussionen:
  - *Könnten wir ein Strategiepattern verwenden, um die Anzahl der Fallentscheidungen zu reduzieren?*

Design Patterns unterstützen Wiederverwendung von Code

- Ziel ist robuster, flexibler und wiederverwendbarer Code
- Unser Design muss Anforderungensänderungen in der Zukunft erlauben:
- Nur Code, der sich weiter entwickeln kann, kann wiederverwendet werden
- Monolithischer, unflexibler Code ist ein Risiko: erfordert möglicherweise grundlegende Änderungen / Neuentwicklung in der Zukunft
- Design Pattern unterstützen Flexibilität

"Design for change"

Sind Design Patterns 'language smells'?

- Design Patterns sind nützliche Muster, die im Allgemeinen nicht direkt von Programmiersprache unterstützt werden:
  - ausgehend von prozeduralen Sprachen würden wir über Muster wie Vererbung, Encapsulation and Polymorphismus reden.
  - Design Patterns in GoF Buch diskutieren Smalltalk/C++-level language (von 1995)
- Einige der Muster sind direkt verfügbar in weniger weit verbreiteten OO Sprachen
- Sprachen ändern sich, und integrieren Design Patterns im Laufe der Zeit
- Neue domänenspezifische Pattern entstehen (Web Design, Concurrency, Security, ...)

Zusammenfassung

Zusammenfassung

- Einführung Design Patterns
- Überblick
- Template und Strategie Muster
- Design Patterns repräsentieren nützlichen Konzepte und Erfahrungen
- Design Patterns geben uns eine gemeinsam Sprache für diese Konzepte

Die nächsten Schritte

In der nächsten Übung

- Sie bekommen Code und den Auftrag, diesen zu verbessern, ohne die Funktionalität zu ändern.
- Wir empfehlen Designpattern, die Sie in Erwägung ziehen und verwenden können.

Literatur

Literatur

Design Patterns – Elements of Reusable OO Software

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:

Addison-Wesley Professional Computing Series, 1994 [Use for Reference]

Head First Design Patterns

Eric Freeman and Elisabeth Freeman

OReilley Publications

Videos

Derek Banas, Design Patterns Video Tutorials

`https://youtu.be/vNHpsC5ng_E`

Vorlesungsmaterialien

`http://github.com/uni-wuppertal/SE-designpattern`