Objektorientierte Programmierung

FH-Prof. Dipl.-Ing. Dr. Marc Kurz
Alexander Palmanshofer, MSc
Bachelor Mobile Computing
Übung 05

General Information
Design Patterns

Die Klasse **Comparable** ist ein Interface, welche als einzige Methode die Vergleichsoperation **int compareTo(T other)** anbietet. Implementiert eine Klasse das Comparable-Interface, dann muss über diese Methode eine spezialisierte Vergleichsoperation bereitgestellt werden.

General Information
Design Patterns

Die Klasse **Comparable** ist ein Interface, welche als einzige Methode die Vergleichsoperation **int compareTo(T other)** anbietet. Implementiert eine Klasse das Comparable-Interface, dann muss über diese Methode eine spezialisierte Vergleichsoperation bereitgestellt werden.

Beispiel: Sie haben eine Klasse Car. Ein Car ist dann „größer" als ein anderes, wenn seine Länge größer ist. Daher würde die Implementierung etwa so aussehen:

```
public class Car implements Comparable<Car> {
	private double length; // relevant for comparison
	private double width; // not relevant for comp.
	private double engineDisplacement; // not relevant for comp.

	@Override
	public int compareTo(Car other) {
		if (this.length > other.length) {
			return 1;
		} else if (this.length == other.length)
			return 0;
		else
			return -1;
	}
}
```

General Information
Design Patterns

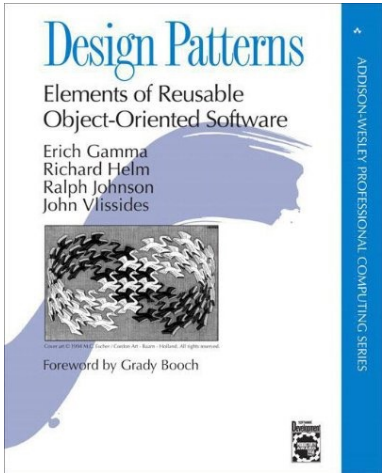
In der Implementierung der Klasse **BinarySearchTree** müssen Sie dann einerseits die Schnittstelle all jener Methoden anpassen, die vorher einen Integer als Wertparameter übernommen haben. Außerdem müssen sämtliche Vergleiche anpasst werden, da die Vergleichsoperatoren „<", „>", „==", etc. für Objekte nicht verfügbar sind.

Aus `if (_data > _tempRoot.data) {…}` wird dementsprechend

```
if (_data.compareTo(_tempRoot.data) > 0) {…}
```

General Information
Design Patterns

- **Creational Patterns (Erzeugende Muster):** Patterns for generating objects
- **Structural Patterns (Strukturmuster):** Structuring classes & objects
- **Behavioral Patterns (Verhaltensmuster):** how classes & objects interact with each other



Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Design Patterns. Elements of Reusable Object-Oriented Software**, Addison-Wesley Longman, Amsterdam, 395 Seiten, ISBN 0201633612, 1994.

Design Patterns is a modern classic in the literature of object-oriented development, offering timeless and elegant solutions to common problems in software design. It describes patterns for managing object creation, composing objects into larger structures, and coordinating control flow between objects. The book provides numerous examples where using composition rather than inheritance can improve the reusability and flexibility of code.

General Information
What is a design pattern?

Eine schematische Lösung
für ein häufig wiederkehrendes Problem

schematisch:
- kein Code
- kein Algorithmus
- sondern Skizzierung einer Lösung durch beteiligte Objekte und ihre Beziehungen

"Trickkiste" des erfahrenen Programmierers

Bewährtes Architekturwissen

Structural Patterns
Kompositum (Composite)

Zweck Zusammengesetztes Objekt, das wie ein Einzelobjekt behandelt werden kann

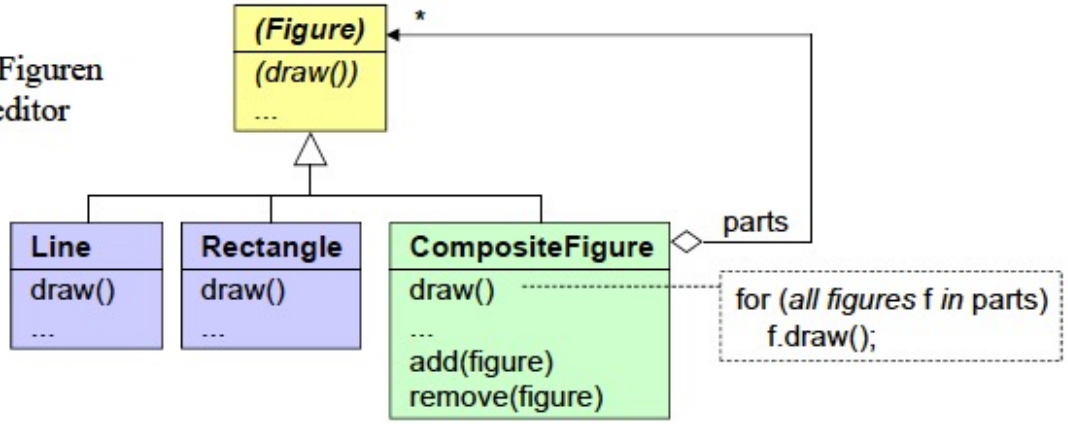
Lösung Zusammengesetztes Objekt ist Mitglied der Familie

Structural Patterns
Kompositum (Composite)

Zweck Zusammengesetztes Objekt, das wie ein Einzelobjekt behandelt werden kann

Lösung Zusammengesetztes Objekt ist Mitglied der Familie

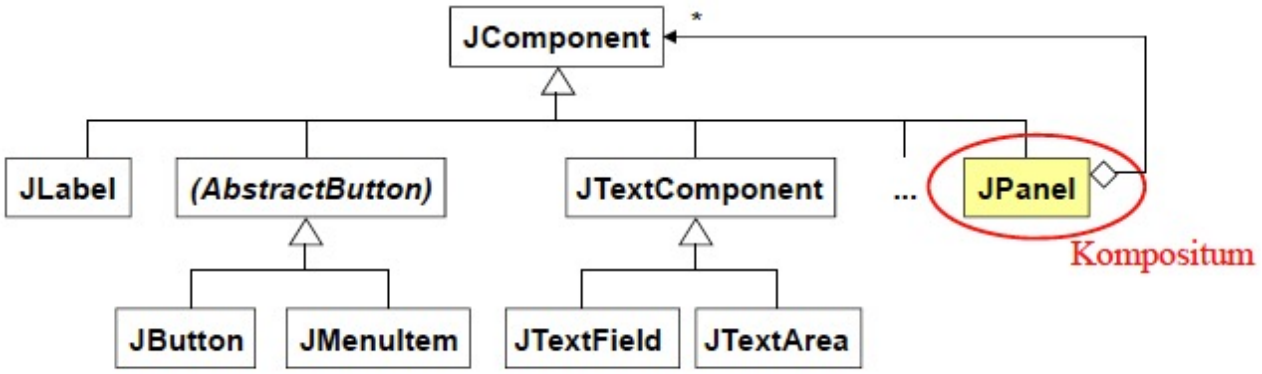
Beispiel
Gruppieren von Figuren
in einem Grafikeditor



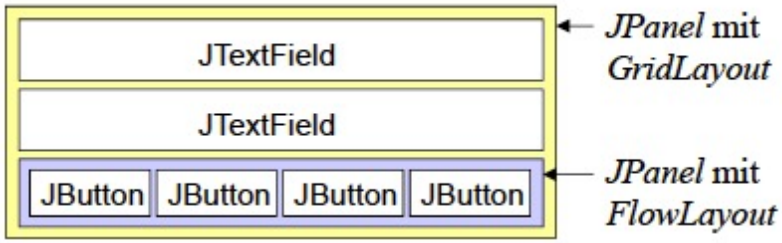
Alle *Figure*-Operationen sind auch auf Figurengruppen anwendbar

Oft ist auch ein *parent*-Zeiger von den Komponenten zum Kompositum nützlich

Structural Patterns
Beispiel: Panels in javax.swing



Taschenrechner-Beispiel aus dem Framework-Kapitel



Structural Patterns
Muster „Kompositum“