Strategie

(Strategy / Policy)

Ein objektbasiertes Verhaltensmuster

Stephan Munkelt, Stefan Salzmann - 03IN

Gliederung

1. Zweck
2. Motivation
3. Anwendbarkeit
4. Struktur
5. Teilnehmer
6. Interaktionen
7. Konsequenzen
8. Implementierung
9. Beispiele
10. Bekannte Verwendungen

1. Zweck

→ Definiere Familie von Algorithmen

→ Kapsele jeden einzelnen und mache Sie austauschbar

→ Variationen des Algorithmus unabhängig von ihn nutzenden Klienten möglich

2. Motivation

Kapselung unterschiedlicher Algorithmen, welche gleiche bzw. ähnliche Aufgaben erfüllen, ist aus folg. Gründen vorteilhaft:

→ Übersichtlicher

→ Einfachere Wartung

→ Code einfacher zu erweitern und zu verändern

→ Nur wirklich benötigte Algorithmen werden bereitgestellt

→ Zu verwendender Algorithmus ergibt sich aus Kontext

Einen solcherart gekapselten Algorithmus nennt man Strategie.

3. Anwendbarkeit

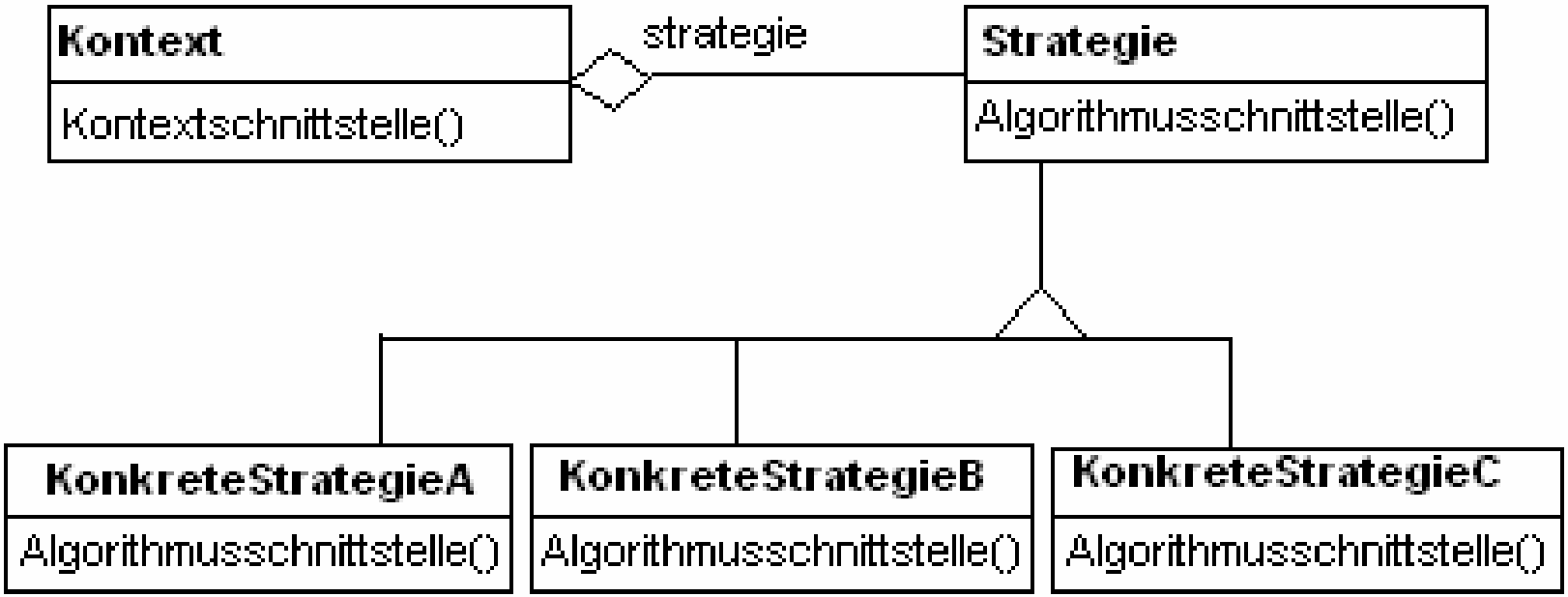
→ bei unterschiedlichen Verhaltensmustern verwandter Klassen

→ bei unterschiedlichen Varianten eines Algorithmus

→ wenn Algorithmus Daten verwendet, die Klienten nur eingeschränkt bekannt sein sollen

→ wenn eine Klasse unterschiedliche Verhaltensweisen definiert und diese als mehrfache Bedingungsanweisungen in ihren Operationen erscheinen
(Auslagerung von Bedingungszweigen in eigene Strategieklassen)

4. Struktur


Kontext
Kontextschnittstelle()
strategie
Strategie
Algorithmusschnittstelle()
KonkreteStrategieA
Algorithmusschnittstelle()
KonkreteStrategieB
Algorithmusschnittstelle()
KonkreteStrategieC
Algorithmusschnittstelle()

5. Teilnehmer

Kontext: - wird mit KonkreteStrategie-Objekt konfiguriert
- verwaltet Referenz auf Strategieobjekt
- kann Schnittstelle für Strategieobjekte definieren

Strategie: - deklariert eine von allen unterstützten Algorithmen angebotene Schnittstelle (um auf Algorithmus der KonkreteStrategie zugreifen zu können)

KonkreteStrategie: - stellt über Strategieschnittstelle den zu implementierenden Algorithmus bereit

6. Interaktionen

- Kontextmenu übergibt benötigte Daten (evtl. sich selbst) an Strategieobjekt

- Klient erzeugt, von ihm ausgewähltes, KonkreteStategie-Objekt und kann über Kontext auf Strategie zugreifen

7. Konsequenzen

→ Familie von verwandten Algorithmen

→ Stellt Alternative zur Unterklassenbildung

→ Strategien entfernen Bedingungsanweisungen

→ Auswahlmöglichkeit für Implementierungen (Wiederverwendung)

→ Klienten müssen Strategien kennen

→ Kommunikationsaufwand zwischen Strategie und Kontext

→ Erhöhte Anzahl von Objekten

8. Implementierung

- Definition der Strategie- und Kontextschnittstelle

  Kontext übergibt benötigte Daten als Parameter oder Strategie erfragt Daten explizit und direkt vom Kontext (über Referenz)

- Strategien als Template-Paramenter in C++, wenn:

  → Strategie zur Übersetzungszeit auswählbar

  → Strategie während Laufzeit beibehalten wird

- Strategieobjekte optional machen

  → Kontext benötigt nicht zwingend Strategieobjekt, wenn vordefiniertes Standardverhalten ausreichend

9. Beispiel (I)

Interface *Stepper*:

(Strategie)

```
package uebKW12;

public interface Stepper {
        public abstract String form();

        public abstract void step();
}
```

Klasse *Texter*:

(KonkreteStrategieA)

```
package uebKW12;

public class Texter implements Stepper {
        String foobar = "";
        public String form() {
                return foobar;
        }

        public void step() {
                foobar = foobar + "x";
        }
}
```

9. Beispiel (II)

Klasse *Counter*:

(KonkreteStrategieB)

```
package uebKW12;

public class Counter implements Stepper {
        int state = 0;

        public String form () {
                return Integer.toString(state);
        }

        public int getState() {
                return state;
        }

        public void setState(int state) {
                this.state = state;
        }

        public void step () {
                setState(1 + getState());
        }
}
```

9. Beispiel (III)

Klasse *Display*:

(Kontext)

```
package uebKW12;
import java...

public class Display extends Applet {
        Button b = new Button ("foobar-Schalter");
        Label  l = new Label  ("foobar-Label");

        public void init() {
                final Stepper c  = new Counter ();
                add (b);
                add (l);

                b.addActionListener (new ActionListener () {
                    public void actionPerformed(ActionEvent e) {
                        c.step ();
                        l.setText(c.form());
                    }
                });
        }
}
```

10. Bekannte Verwendung

- Mechanismen zur Registerbelegung, Verwendung von Befehlssätzen

  $\rightarrow$ erhöhte Flexibilität bei Anpassung an unterschiedliche Systemarchitekturen

  (RTL-System zur Codeoptimierung von Compilern)

- Berechnungsmaschinen für unterschiedliche Finanzmodelle

  (ET++SwapsManagerFramework)

- Strategien für Dialogboxen zur Eingabevalidierung

  (Borland ObjectWindows-Framework)

ENDE