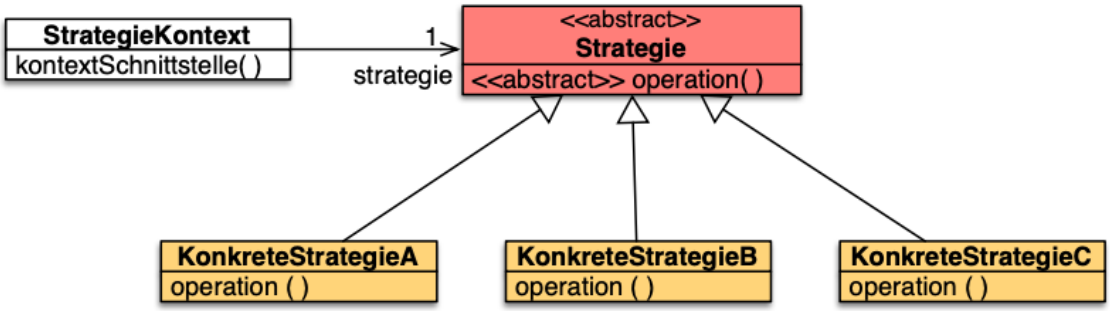
Software Engineering 1 Übungsblatt 10 - Pattern / Implementierung

Aufgabe 10.1: Strategie-Pattern – Pflichtaufgabe
Das Strategie-Pattern dient der Austauschbarkeit und Kapselung von Algorithmen.

- **Problem**:
  Verwandte Klassen unterscheiden sich häufig nur dadurch, dass sie gleiche Aufgaben teilweise durch verschiedene Algorithmen lösen. Diese Algorithmen sollen während der Laufzeit ausgetauscht werden können.
- **Lösung**:
  Eine Klasse (`StrategieKontext`) beschreibt den Standardablauf zur Problemlösung, in dem unterschiedliche Algorithmen verwendet werden können. Eine abstrakte Klasse (`Strategie`) definiert die gemeinsame Schnittstelle für alle Algorithmus-Varianten. Von dieser Klasse wird für jede Implementierungsalternative eine konkrete Unterklasse (`KonkreteStrategie`) abgeleitet. Die Klasse `StrategieKontext` benutzt `KonkreteStrategie`-Objekte, um die unterschiedlich implementierten Operationen per Delegation auszuführen.

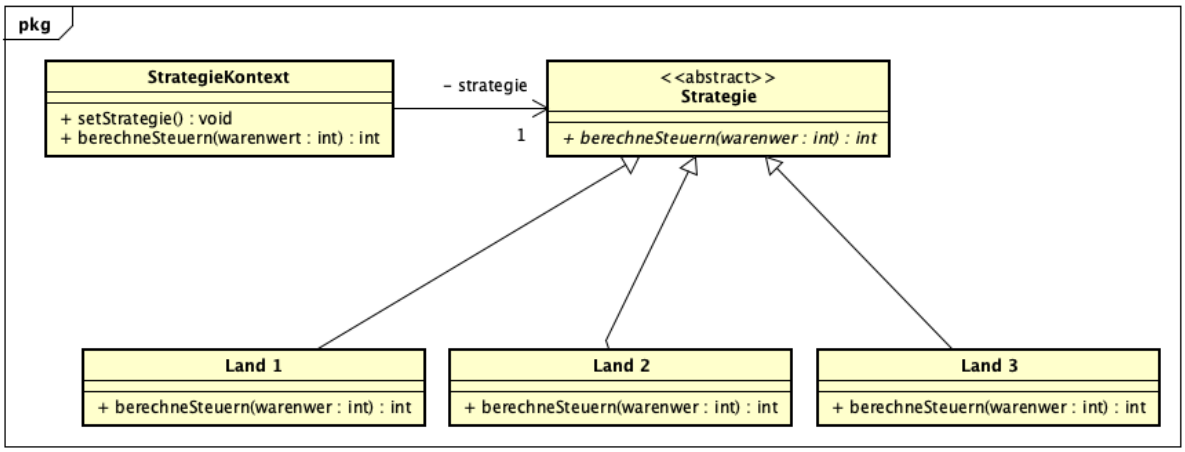


Lösen Sie mit Hilfe des Strategie-Patterns das folgende Problem:
Eine Firma verkauft ihre Waren in verschiedene Länder mit unterschiedlichen Steuervorschriften. Bei der Abwicklung von Kundenaufträgen müssen die länderspezifischen Steuerberechnungsvorschriften beachtet werden.

- Es soll für jede Steuerberechnungsvorschrift eine Methode entwickelt werden
  `berechneSteuer(int warenwert) : int`

Die Firma soll dynamisch, d.h. während der Laufzeit, ihre Steuerberechnungsvorschriften ändern können (und zwar mit einer Methode `setStrategie(Strategie strategie))`.

a) Entwickeln Sie ein Klassendiagramm, mit dem sich das beschriebene Szenario umsetzen lässt.
b) Implementieren Sie Ihre Lösung.
c) Wie könnte man die Aufgabe mit Vererbung implementieren (von Oberklasse `Strategie-Kontext`)? Was sind die Nachteile dieses Ansatzes im Vergleich zum Strategie-Pattern?



```java
abstract class Strategie {
    public abstract double berechneSteuern(double i);
}
public class StrategieKontext {

    Strategie strategie;

    public void setStrategie(Strategie s) {
        strategie = s;
    }

    public double berechneSteuern(double d) {
        if( this.strategie != null)
            return this.strategie.berechneSteuern(d);
        return 0;
    }


}
```

```java
public class a_LandA extends Strategie{
    @Override
    public double berechneSteuern(double i) {
        return i * 0.7;
    }
}
```

```java
public class a_LandB extends Strategie{
    @Override
    public double berechneSteuern(double i) {
        return i * 0.11;
    }
}
```

```java
public class a_LandC extends Strategie{
    @Override
    public double berechneSteuern(double i) {
        return i * 0.19;
    }
}
```

Aufgabe 10.2: Integration

1. Stellen Sie in einer Tabelle die in der Vorlesung vorgestellten unterschiedlichen Integrationsstrategien vergleichend gegenüber (z.B. Bewertung +, o, -). Welche Kriterien können für den Vergleich verwendet werden?
2. Welche Integrationsstrategie ist für den Unified Process am besten geeignet?

1.

	Fachliches Risiko	**Technisches Risiko**
Big-Bang-Integration	-	-
Top-Down-Integration	+	-
Bottom-Up-Integration	-	+
Build-Integration	o	o

- Build-Integration ist richtiges/kritisches Use Case (vertikaler Ausschnitt) wichtig. Manchmal ist die Teilmenge von Use Cases nicht ausreichend um Risiko zu minimieren. Build-Integration häufig größter Aufwand, aber verringert Risiko am meisten

2. UP versucht technisches und fachliches Risiko zu minieren, also ist der **Build-Integration** am besten geeignet