



OCP

The open-closed Principle

Inhalte

- Einführung
- The Shape Abstraction
- Strategic Closure
- Heuristics & Conventions
- Fallbeispiel: UML-Klassendiagramm

Einführung

Heuristik ist mit OOD verbunden -> OCP liegt dem zugrunde

- Systeme verändern sich im Lebenszyklus: dies muss bei der Software-Entwicklung beachtet werden
- Hauptmerkmale des OCP:
 - Zur Erweiterung offen
 - Zur Änderung geschlossen



Abstraktion ist hier der Schlüssel zur Lösung

The Shape Abstraction

Ausgangssproblematik:

- Funktion **DrawAllShapes** wird für die Zeichnung von Kreisen und Quadraten in einer bestimmten Reihenfolge implementiert.



Programm muss die Liste mit der entsprechenden Reihenfolge durchlaufen und die jeweilige Zeichnung ausführen

The Shape Abstraction

Problem:

- Wenn man die Funktion **DrawAllShapes** so erweitern möchte, dass auch Dreiecke gezeichnet werden können, muss man die Funktion, und damit bestehenden Code, ändern.

Lösung mithilfe des **Open-Close-Prinzip**:

- Verhalten des Systems ändern, ohne Code zu ändern: Abstraktionen und damit neuen Code hinzufügen.




Entwickler sollten daher mit Abstraktionen in Form von abstrakten Klassen oder Interfaces arbeiten.

Strategic Closure

Kein Programm kann zu 100% geschlossen für Modifikationen sein

- Beispiel: Die Formen sollen nun in einer bestimmten Reihenfolge gezeichnet werden. (Erst Kreise, dann Quadrate)
 - Funktion `DrawAllShapes` ist für diese Änderung nicht geschlossen
 - Es gibt immer bestimmte Änderungen, für die Module nicht geschlossen sind



Entwickler sollten darauf achten, dass das open-closed Prinzip für die Wahrscheinlichsten Änderungen eingehalten wird.

Strategic Closure: Ansatz Abstraktion

Wie könnten wir die `DrawAllShapes` Funktion aufgrund von Änderungen in der Zeichnungsreihenfolge schließen?

Wir benötigen eine „Ordnungsabstraktion“:

- Ansatz: Mit 2 Objekten soll festgelegt werden welches zuerst gespeichert wird.
- Lösung: Funktion in der Shape-Klasse, welches 2 Shape-Objekte miteinander vergleicht und prüft, welches beim Zeichnen priorisiert wird.

Strategic Closure: Ansatz Abstraktion

Problem:

Jedes mal, wenn
Eine neue Shape-Klasse
Hinzukommt muss
die Precedes Methode
bearbeitet werden.

Listing 3

Shape with ordering methods.

```
class Shape
{
    public:
        virtual void Draw() const = 0;
        virtual bool Precedes(const Shape&) const = 0;

        bool operator<(const Shape& s) {return Precedes(s);}
};
```

Listing 4

DrawAllShapes with Ordering

```
void DrawAllShapes(Set<Shape*>& list)
{
    // copy elements into OrderedSet and then sort.
    OrderedSet<Shape*> orderedList = list;
    orderedList.Sort();

    for (Iterator<Shape*> i(orderedList); i; i++)
        (*i)->Draw();
}
```

Listing 5

Ordering a Circle

```
bool Circle::Precedes(const Shape& s) const
{
    if (dynamic_cast<Square*>(s))
        return true;
    else
        return false;
}
```

Boolcher Operator "<" wird
überschrieben mit
Funktion Precedes

Funktion Precedes prüft,
welches Shape-Objekt
Vorrang hat

Strategic Closure

Lösung: Verwenden eines datengesteuerten Ansatzes zum Schließen

- mithilfe eines tabellarischen Ansatzes kann verhindert werden, dass es zu Änderungen in jeder geerbten Klasse kommt

Heuristics & Conventions

Make All Variables Private

- Klassenvariablen sollen privat sein
 - *Private* Variable
 - Kapselung
- Klassenvariablen beeinflussen die abhängige Funktion

No Global Variables - Ever

- Die Nutzung von globale Variablen können die abhängige Module beeinflussen

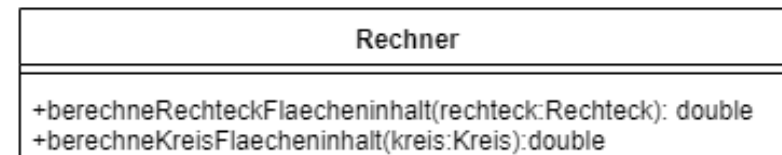
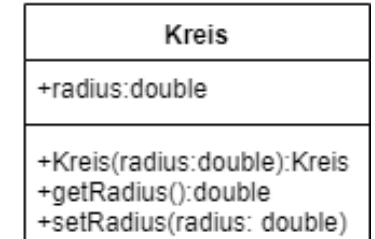
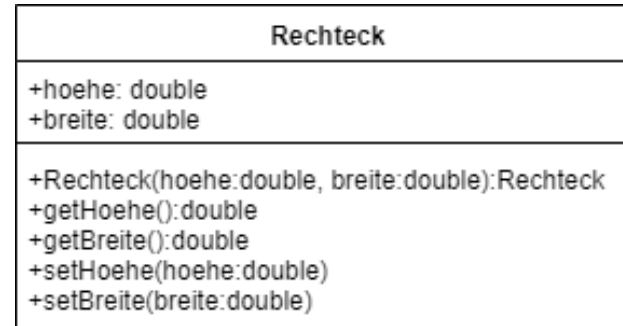
RTTI is Dangerous!

- Techniken zur Laufzeitbestimmung sind gefährlich
 - `dynamic_cast`

UML-Diagramm ohne OCP

- Klassenvariablen sind öffentlich
- Es werden keine Interfaces verwendet
- Die Klasse Rechner muss bei einer neuen Form um die benötigten Methoden erweitert werden

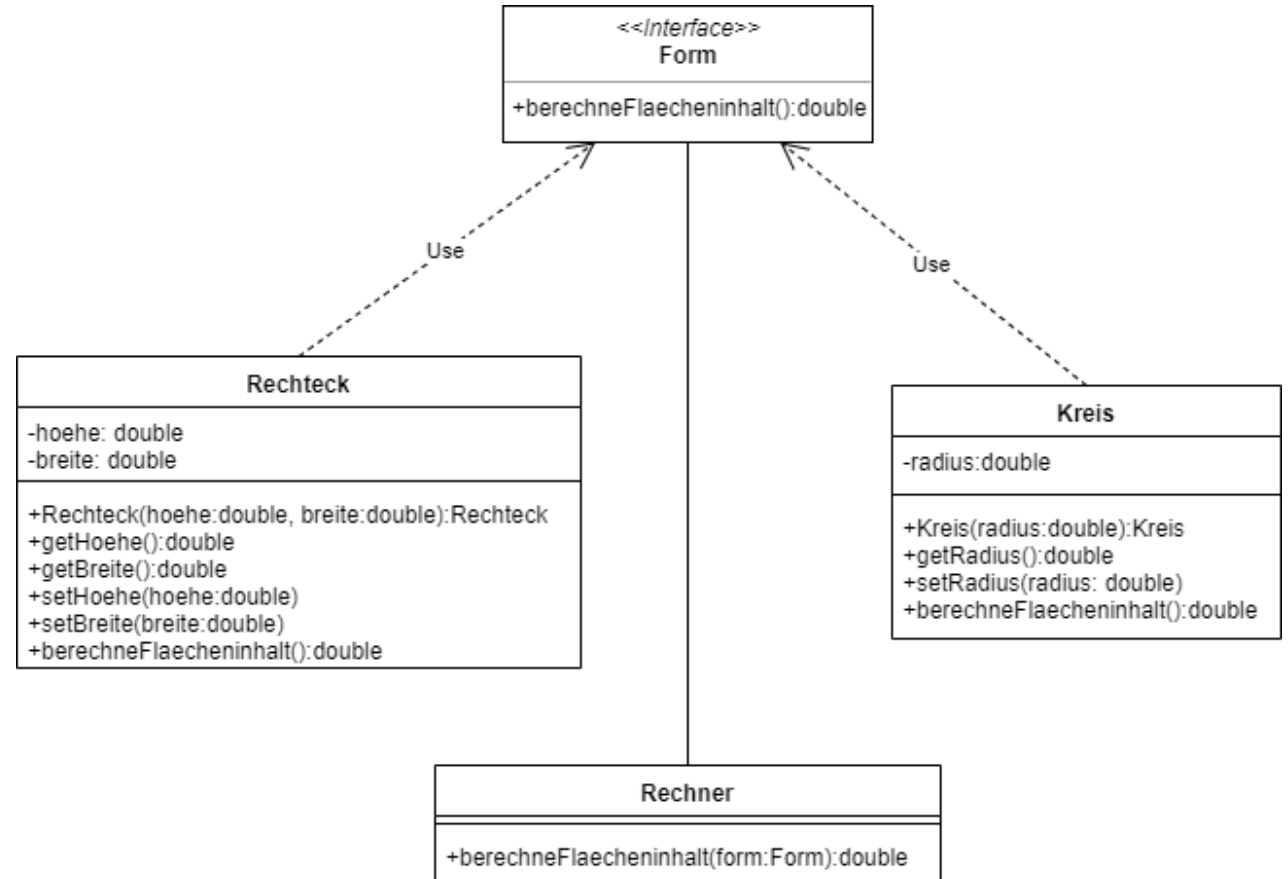
~~Clean Code~~



UML- Klassendiagramm mit OCP

- Klassenvariablen sind privat
- Es werden Interfaces verwendet
- Die Klasse Rechner muss bei einer neuen Form **NICHT** um die benötigten Methoden erweitert werden
- Die Klassen Rechteck und Kreis implementieren ihre eigenen Methoden (vorgegeben durch das Interface Form)

Clean Code



Quellen

- <https://weekly-geekly-es.imtqy.com/articles/de472186/index.html>
- https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#SOLID-Prinzipien
- https://www.youtube.com/watch?v=QKX9VJAvGWM&ab_channel=ProgrammierenStarten
- <http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod>