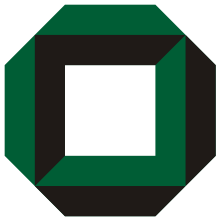
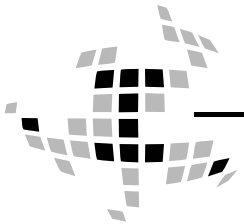


Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

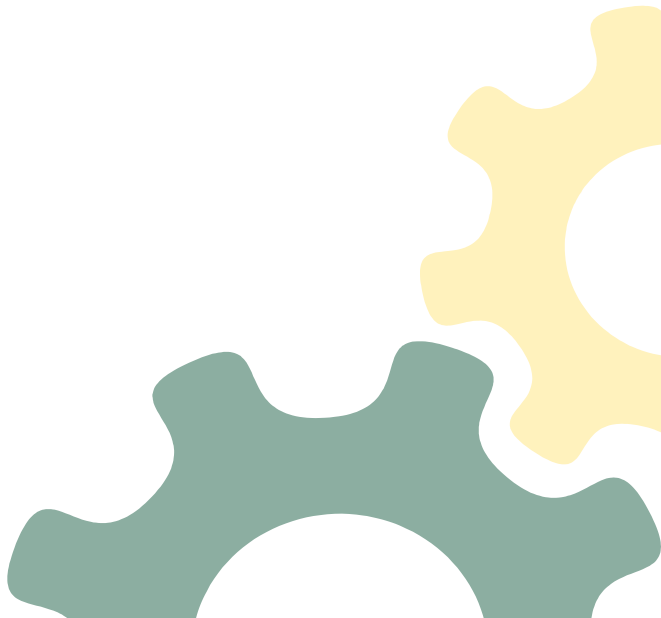
Kapitel 11
Klausurvorbereitung

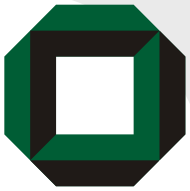
SWT I – Sommersemester 2009

Andreas Höfer
David J. Meder



Fakultät für **Informatik**
Lehrstuhl für Programmiersysteme





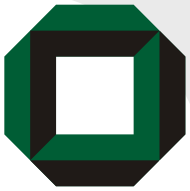
Kontrollflussorientierte Testverfahren

- Die Methode `toDecimal(…)` in die aus der VL bekannte Zwischensprache umwandeln.
- Aus der Zwischensprache den Kontrollfluss-graphen der Methode `toDecimal(…)` erstellen.
- Minimale Testfallmengen erstellen für:
  - Anweisungsüberdeckung
  - Zweigüberdeckung

Kontrollflussorientierte Testverfahren

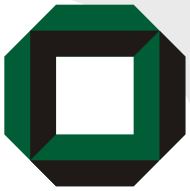
```
public int toDecimal(String roman) {
  int result = 0;
  if (roman == null) {
    result = -1;
  } else {
    String romans = "MDCLXVI";
    int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
    int prevIndex = Integer.MAX_VALUE;
    for (int i = roman.length() - 1; i >= 0; i--) {
      char c = roman.charAt(i);
      int index = romans.indexOf(c);
      if (index < 0) {
        result = -1;
        break;
      }
      if (index <= prevIndex) {
        result += decimals[index];
      } else {
        result -= decimals[index];
      }
      prevIndex = index;
    }
  }
  return result;
}
```

Wandeln Sie die obige Methode in die aus der Vorlesung bekannte Zwischensprache um.



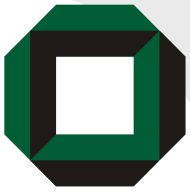
Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if (roman == null) {
030:   result = -1;
040: } else {
050:   String romans = "MDCLXVI";
060:   int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070:   int prevIndex = Integer.MAX_VALUE;
080:   for (int i = roman.length() - 1; i >= 0; i--) {
090:     char c = roman.charAt(i);
100:     int index = romans.indexOf(c);
110:     if (index < 0) {
120:       result = -1;
130:       break;
140:     }
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210:   }
220: }
230: return result;
```



Kontrollflussorientierte Testverfahren

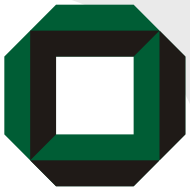
```
010: int result = 0;
020: if (roman == null) {
030:   result = -1;
040: } else {
050:   String romans = "MDCLXVI";
060:   int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070:   int prevIndex = Integer.MAX_VALUE;
080:   for (int i = roman.length() - 1; i >= 0; i--) {
090:     char c = roman.charAt(i);
100:     int index = romans.indexOf(c);
110:     if (index < 0) {
120:       result = -1;
130:       break;
140:     }
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210:   }
220: }
230: return result;
```



Kontrollflussorientierte Testverfahren

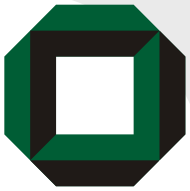
```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
080:   for (int i = roman.length() - 1; i >= 0; i--) {
090:     char c = roman.charAt(i);
100:     int index = romans.indexOf(c);
110:     if (index < 0) {
120:       result = -1;
130:       break;
140:     }
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210:   }
230: return result;
```

else → Sprung zur ersten Anweisung nach dem else-Block



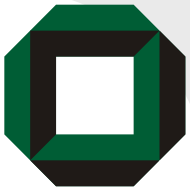
Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
080:   for (int i = roman.length() - 1; i >= 0; i--) {
090:     char c = roman.charAt(i);
100:     int index = romans.indexOf(c);
110:     if (index < 0) {
120:       result = -1;
130:       break;
140:     }
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210:   }
230: return result;
```



Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090:     char c = roman.charAt(i);
100:     int index = romans.indexOf(c);
110:     if (index < 0) {
120:       result = -1;
130:       break;
140:     }
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```



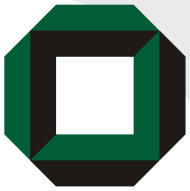
Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090:     char c = roman.charAt(i);
100:     int index = romans.indexOf(c);
110:     if (index < 0) {
120:       result = -1;
130:       break;
140:     }
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

Kontrollflussorientierte Testverfahren

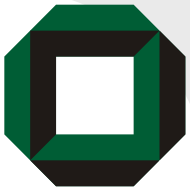
```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

break → Sprung zur ersten Anweisung nach der Schleife



Kontrollflussorientierte Testverfahren

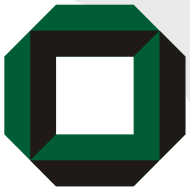
```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150:     if (index <= prevIndex) {
160:       result += decimals[index];
170:     } else {
180:        result -= decimals[index];
190:     }
200:     prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```



Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150: if not (index <= prevIndex) goto 180;
160: result += decimals[index];
170: goto 200;
180: result -= decimals[index];
200: prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

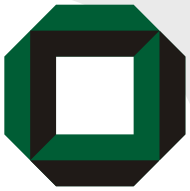
else → Sprung zur ersten Anweisung nach dem else-Block



Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150: if not (index <= prevIndex) goto 180;
160: result += decimals[index];
170: goto 200;
180: result -= decimals[index];
200: prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

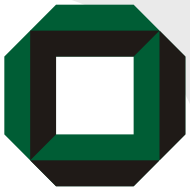
Erstellen Sie aus der Zwischensprache den Kontrollflussgraphen der Methode toDecimal(…). Wenden Sie dabei das in der Vorlesung vorgestellte Verfahren an.



Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150: if not (index <= prevIndex) goto 180;
160: result += decimals[index];
170: goto 200;
180: result -= decimals[index];
200: prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

1. Transformiere in Zwischensprache
2. Fasse alle Folgen, die mit einem Sprung enden, zu je einem Grundblock zusammen



Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150: if not (index <= prevIndex) goto 180;
160: result += decimals[index];
170: goto 200;
180: result -= decimals[index];
200: prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

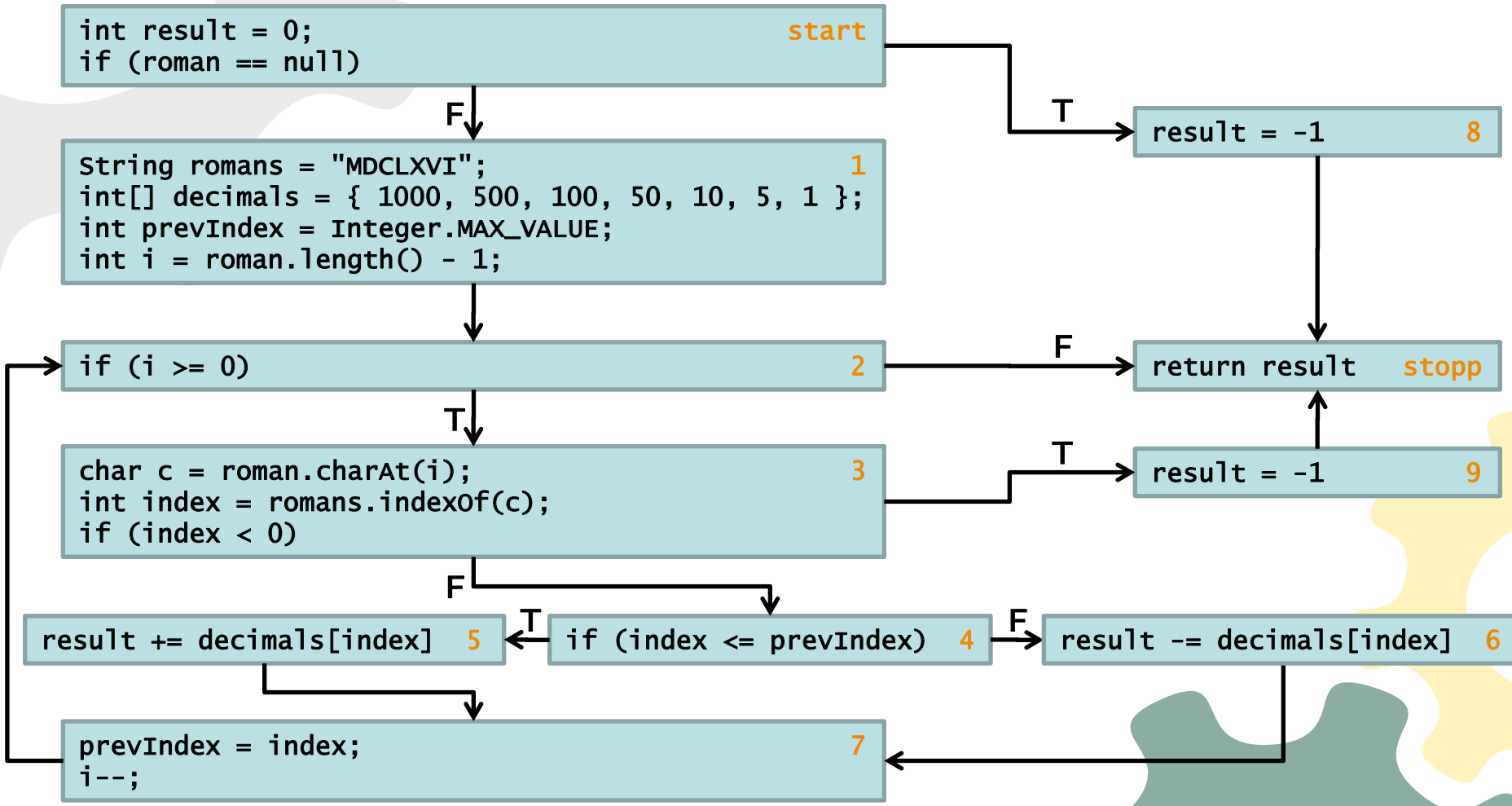
1. Transformiere in Zwischensprache
2. Fasse alle Folgen, die mit einem Sprung enden, zu je einem Grundblock zusammen

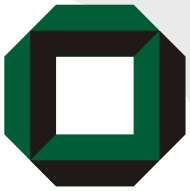
Kontrollflussorientierte Testverfahren

```
010: int result = 0;
020: if not (roman == null) goto 050;
030: result = -1;
040: goto 230;
050: String romans = "MDCLXVI";
060: int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
070: int prevIndex = Integer.MAX_VALUE;
075: int i = roman.length() - 1;
080: if not (i >= 0) goto 230;
090: char c = roman.charAt(i);
100: int index = romans.indexOf(c);
110: if not (index < 0) goto 150;
120: result = -1;
130: goto 230;
150: if not (index <= prevIndex) goto 180;
160: result += decimals[index];
170: goto 200;
180: result -= decimals[index];
200: prevIndex = index;
210: i--;
215: goto 080;
230: return result;
```

- ✓ 3. Prüfe, ob Eintritt nur am Anfang
- ✓ 4. Teile ggf. auf

Kontrollflussorientierte Testverfahren

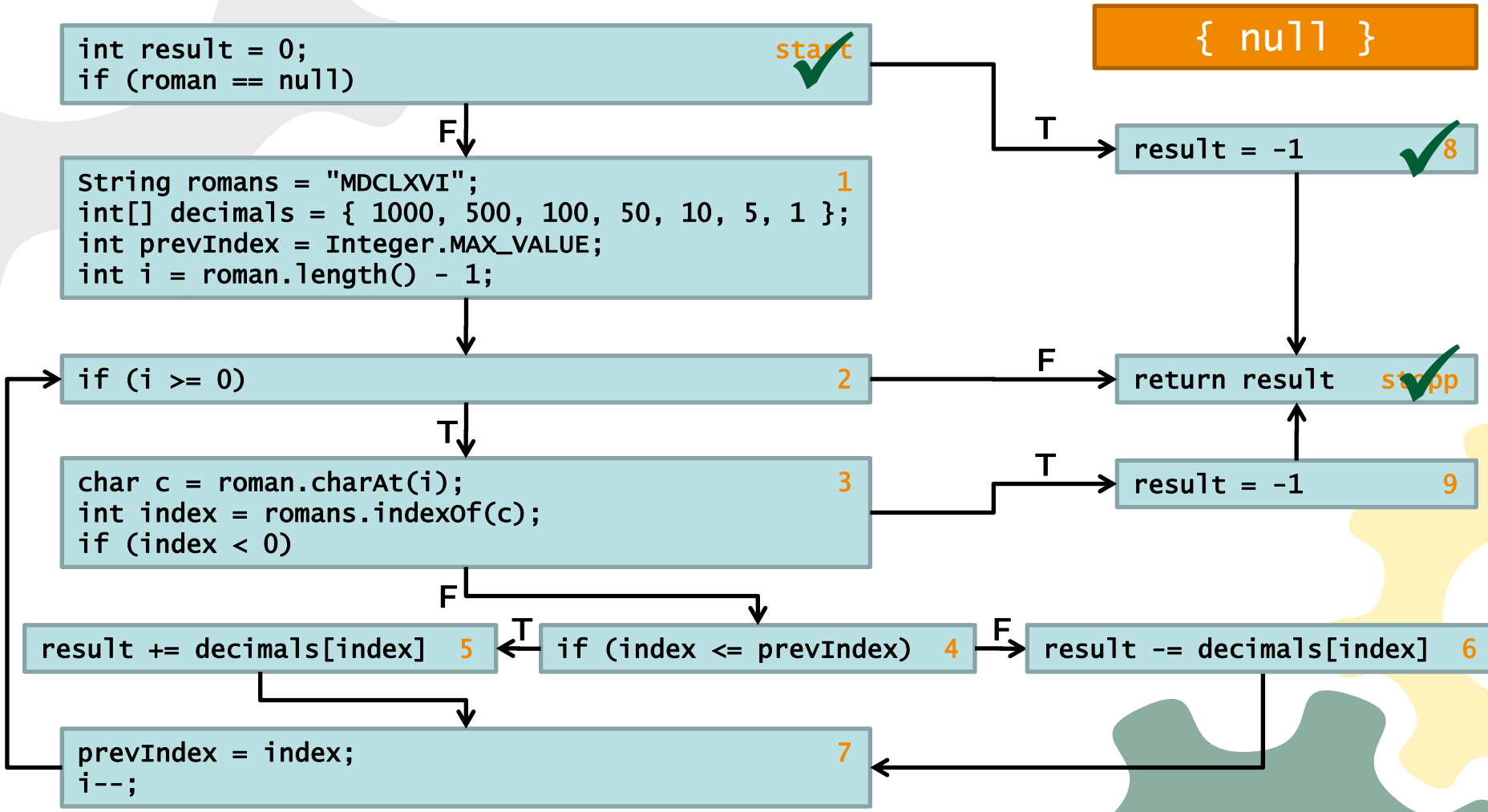




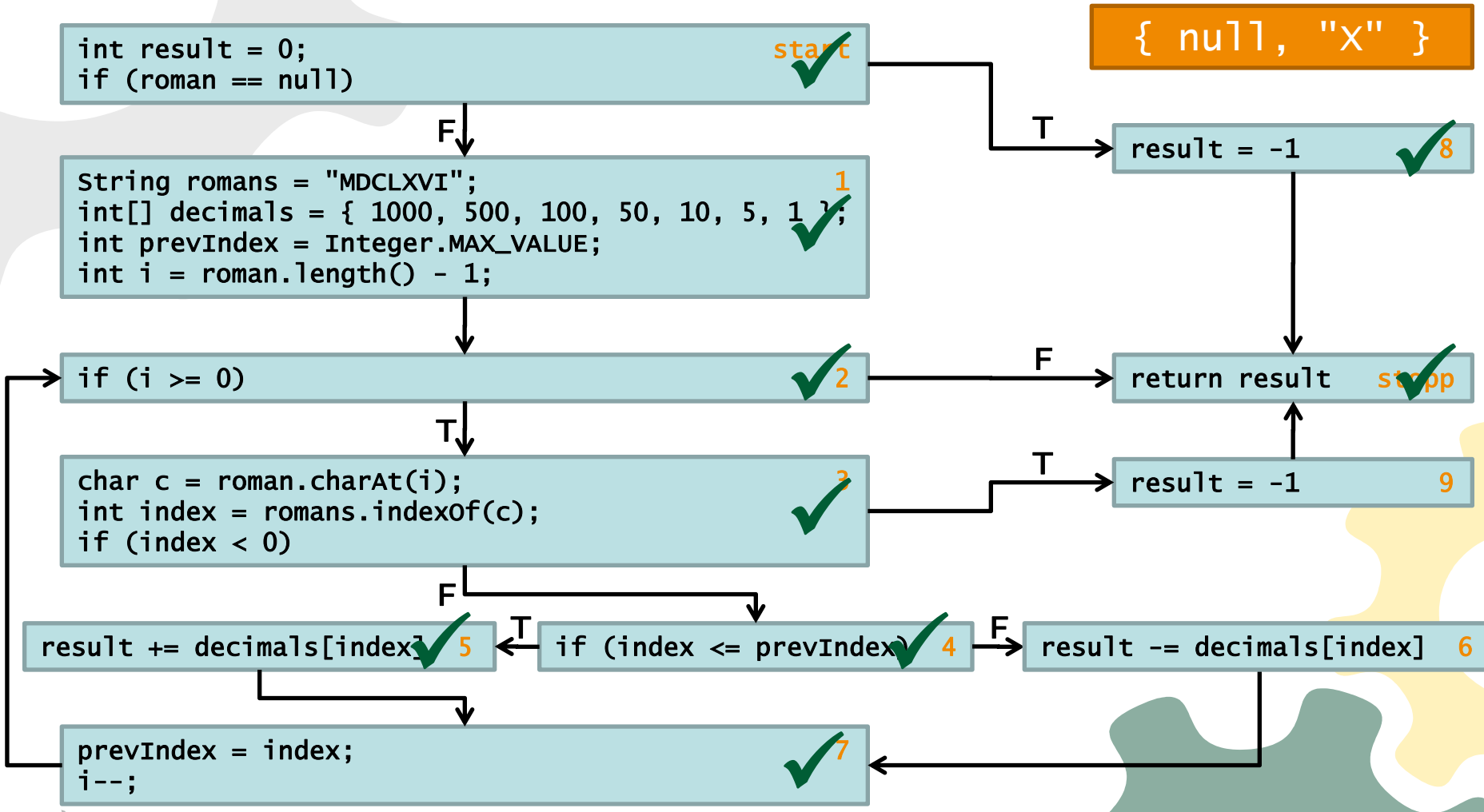
Kontrollflussorientierte Testverfahren

- Gesucht: minimale Testfallmengen für die
  - Anweisungsüberdeckung
  - Zweigüberdeckung
- Was bedeutet „minimale Testfallmenge“?
  1. Die Menge soll möglichst wenige Elemente enthalten.
  2. Jedes einzelne Element soll so kurz wie möglich sein.

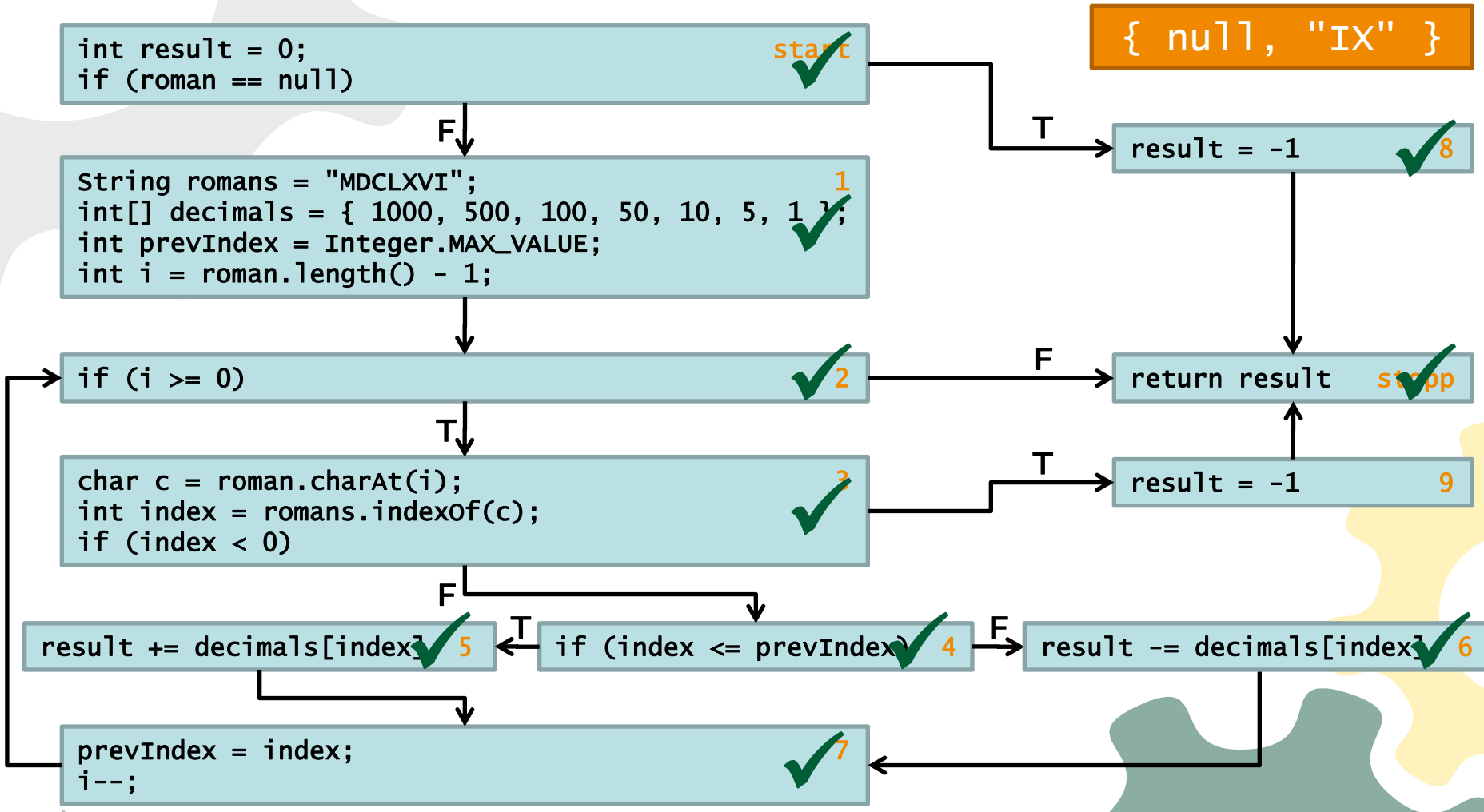
Kontrollflussorientierte Testverfahren



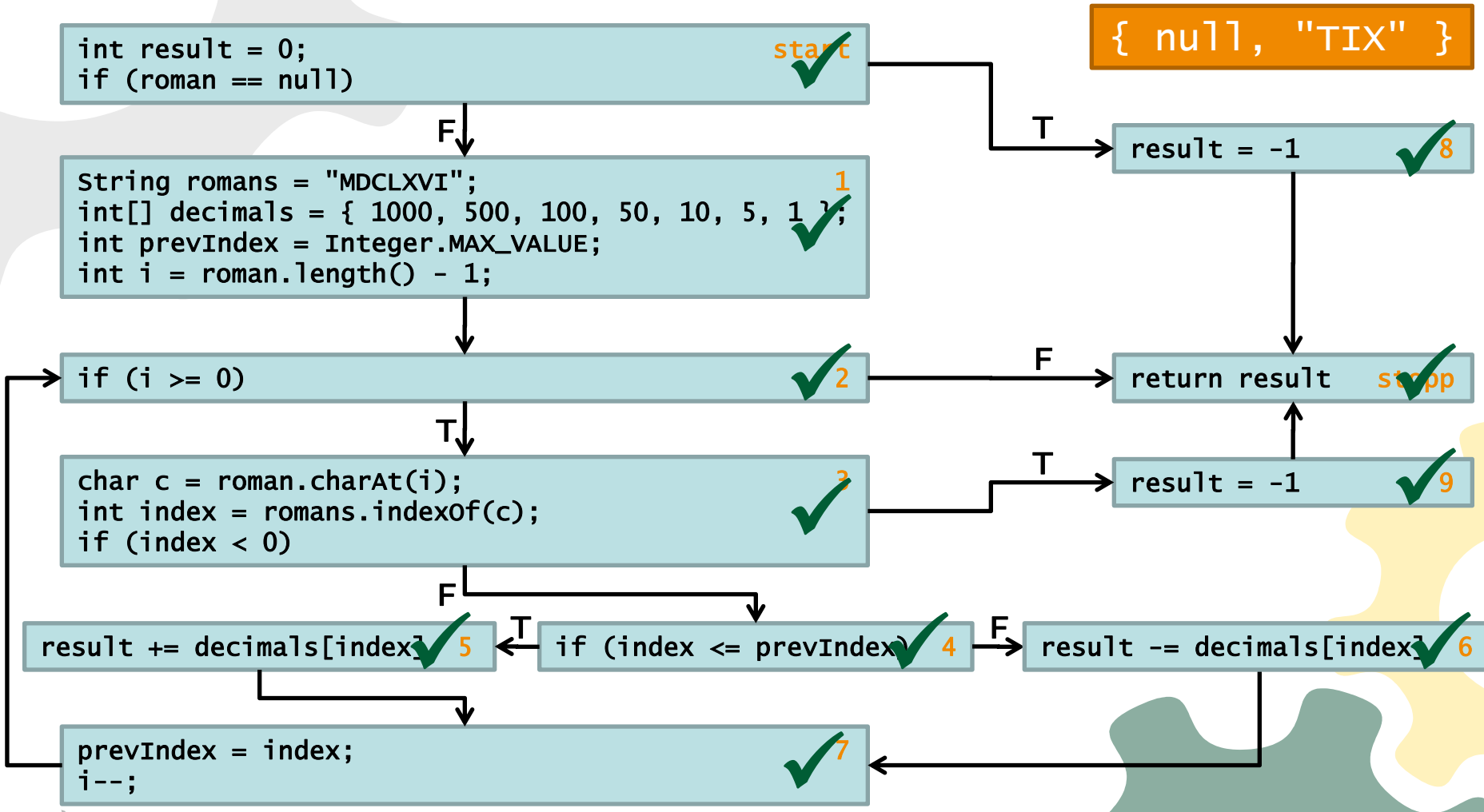
Kontrollflussorientierte Testverfahren



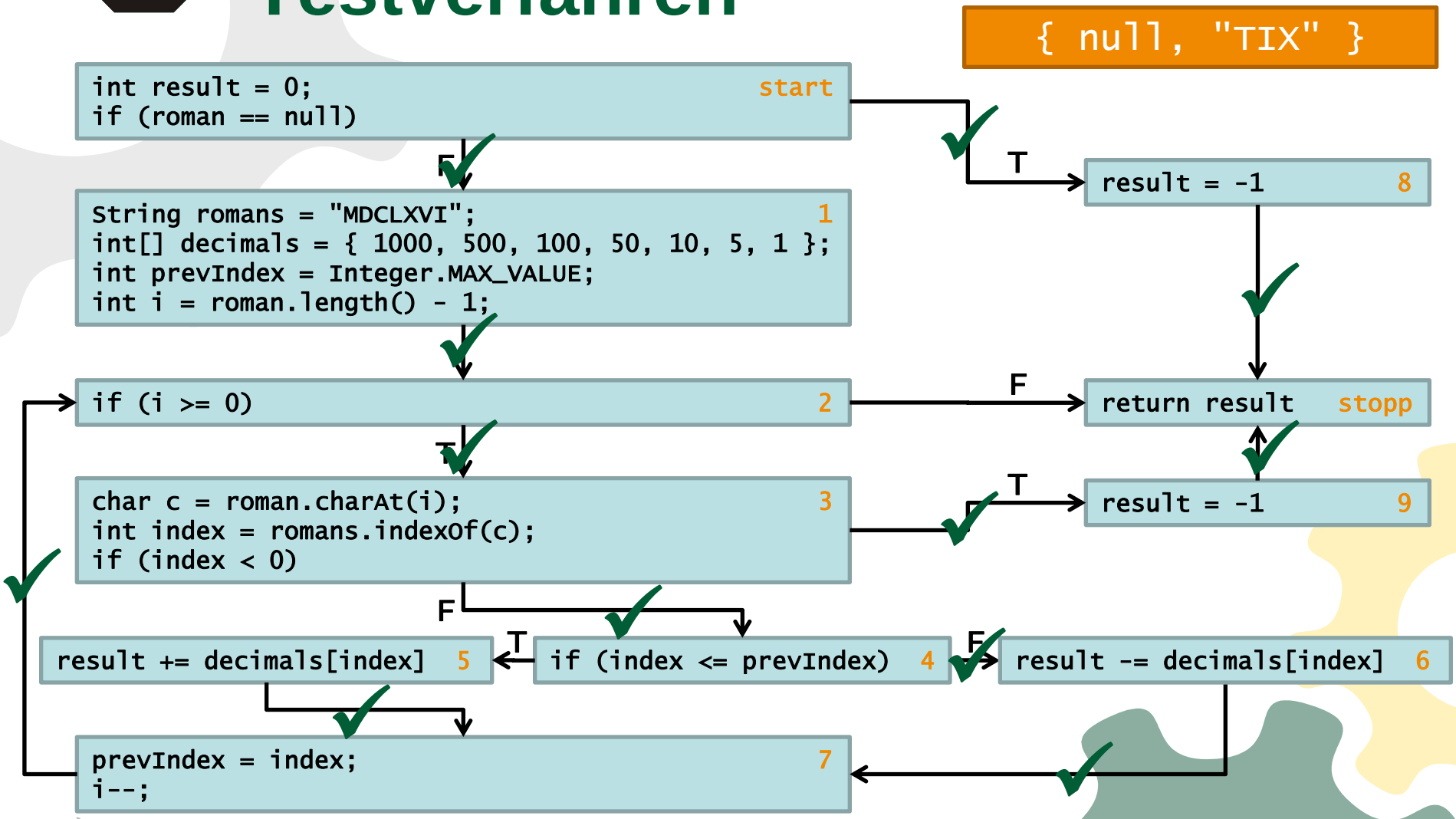
Kontrollflussorientierte Testverfahren



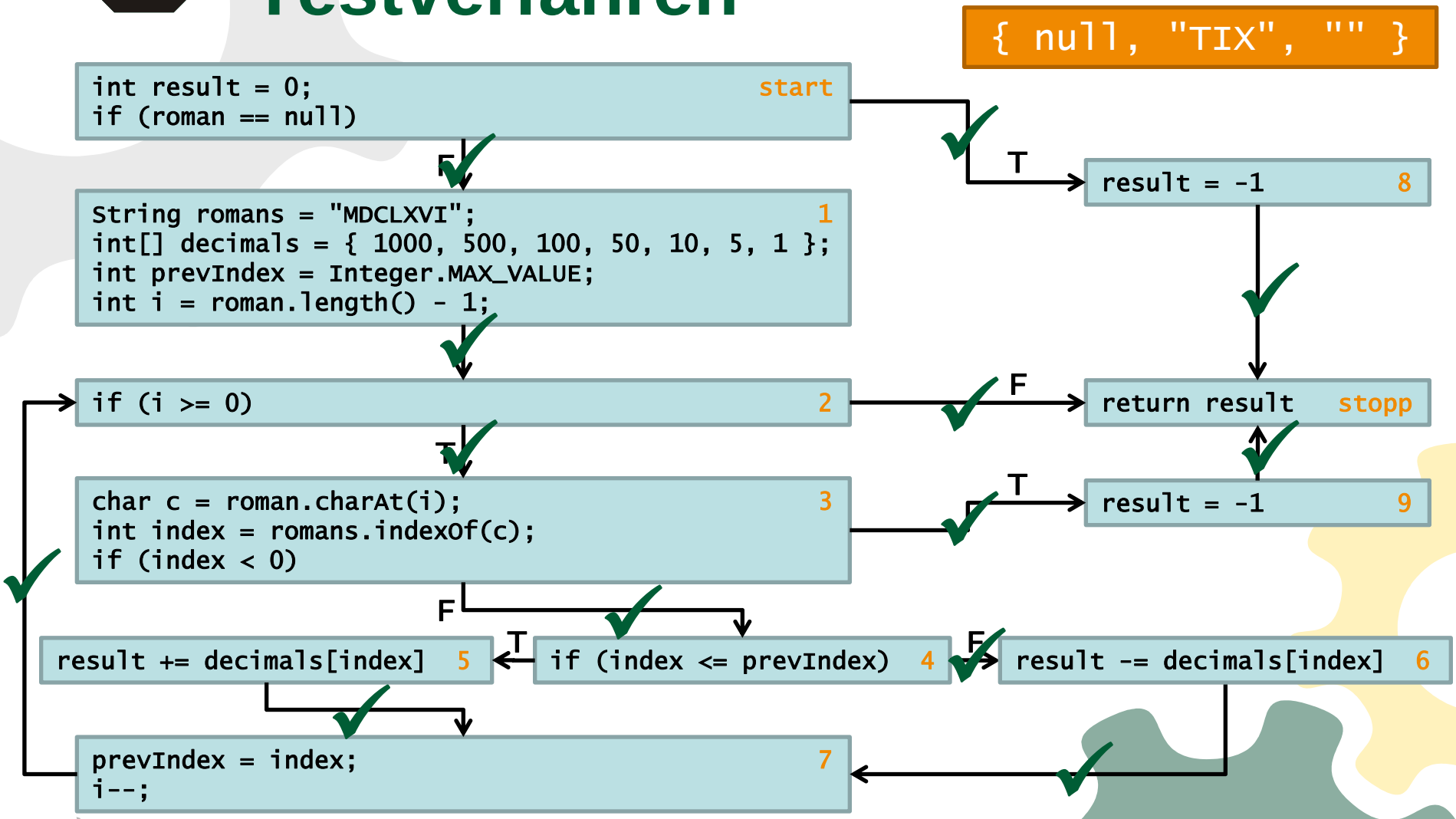
Kontrollflussorientierte Testverfahren



Kontrollflussorientierte Testverfahren



Kontrollflussorientierte Testverfahren

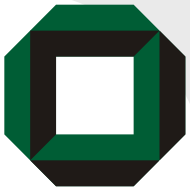


Kontrollflussorientierte Testverfahren

```
public int toDecimal(String roman) {
  int result = 0;
  if (roman == null) {
    result = -1;
  } else {
    String romans = "MDCLXVI";
    int[] decimals = { 1000, 500, 100, 50, 10, 5, 1 };
    int prevIndex = Integer.MAX_VALUE;
    for (int i = roman.length() - 1; i >= 0; i--) {
      char c = roman.charAt(i);
      int index = romans.indexOf(c);
      if (index < 0) {
        result = -1;
        break;
      }
      if (index <= prevIndex) {
        result += decimals[index];
      } else {
        result -= decimals[index];
      }
      prevIndex = index;
    }
  }
  return result;
}
```

So wie hier angegeben, reagiert die Methode nicht auf alle inkorrekten Eingaben mit dem Rückgabewert -1. Geben Sie einen Testfall an, der dieses Verhalten offenlegt.

```
{"VX"}
oder {"IXI"}
```

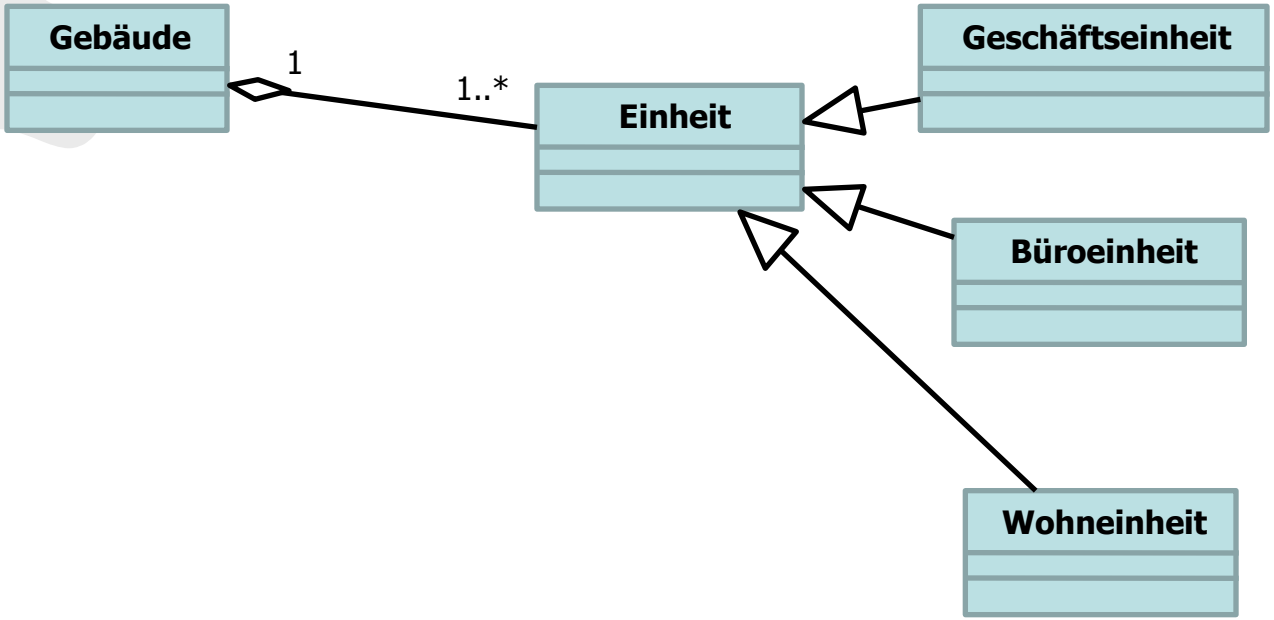


Objektorientierte Analyse und Modellieren

Modellieren Sie folgendes Szenario möglichst genau als UML-Klassendiagramm.

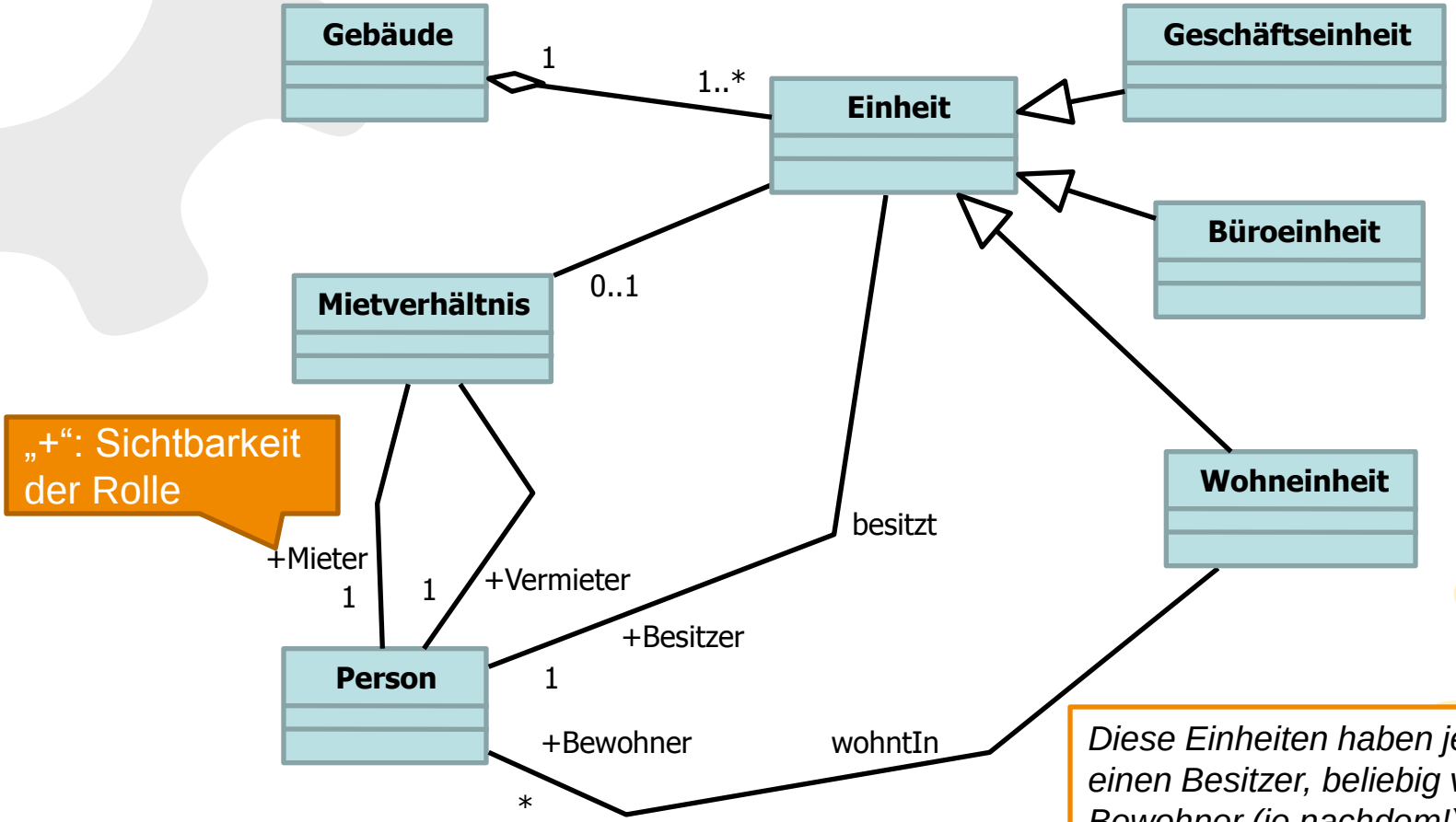
Ein Gebäude besteht aus einer oder mehreren Wohn-, Geschäfts- und/oder Büroeinheiten. Diese Einheiten haben jeweils einen Besitzer, beliebig viele Bewohner (je nachdem!) und ggf. einen Mieter und einen Vermieter. Wenn es einen Mieter gibt, muss es natürlich auch einen Vermieter geben und umgekehrt. Jede Person soll jederzeit ein Miet-, Besitz- oder Bewohn-Verhältnis eingehen oder beenden können. Ein Wohnungseigentümer muss seine eigenen Wohnungen bewohnen können oder fremde Wohnungen mieten können. Jede Person wohnt in mindestens einer Wohneinheit, genau eine dieser Wohneinheiten ist jedoch ihr Hauptwohnsitz.

Objektorientierte Analyse und Modellieren



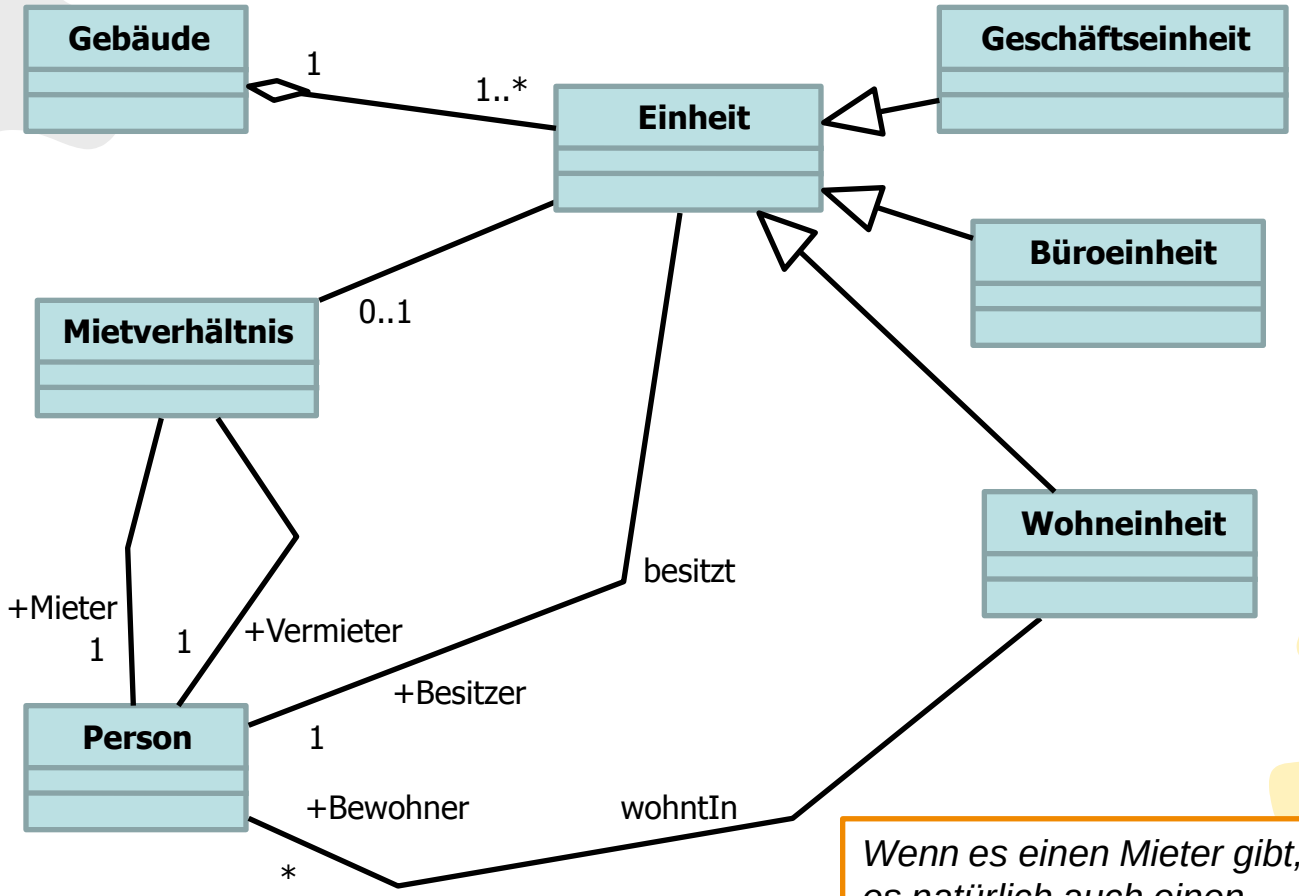
Ein Gebäude besteht aus einer oder mehreren Wohn-, Geschäfts- und/oder Büroeinheiten.

Objektorientierte Analyse und Modellieren



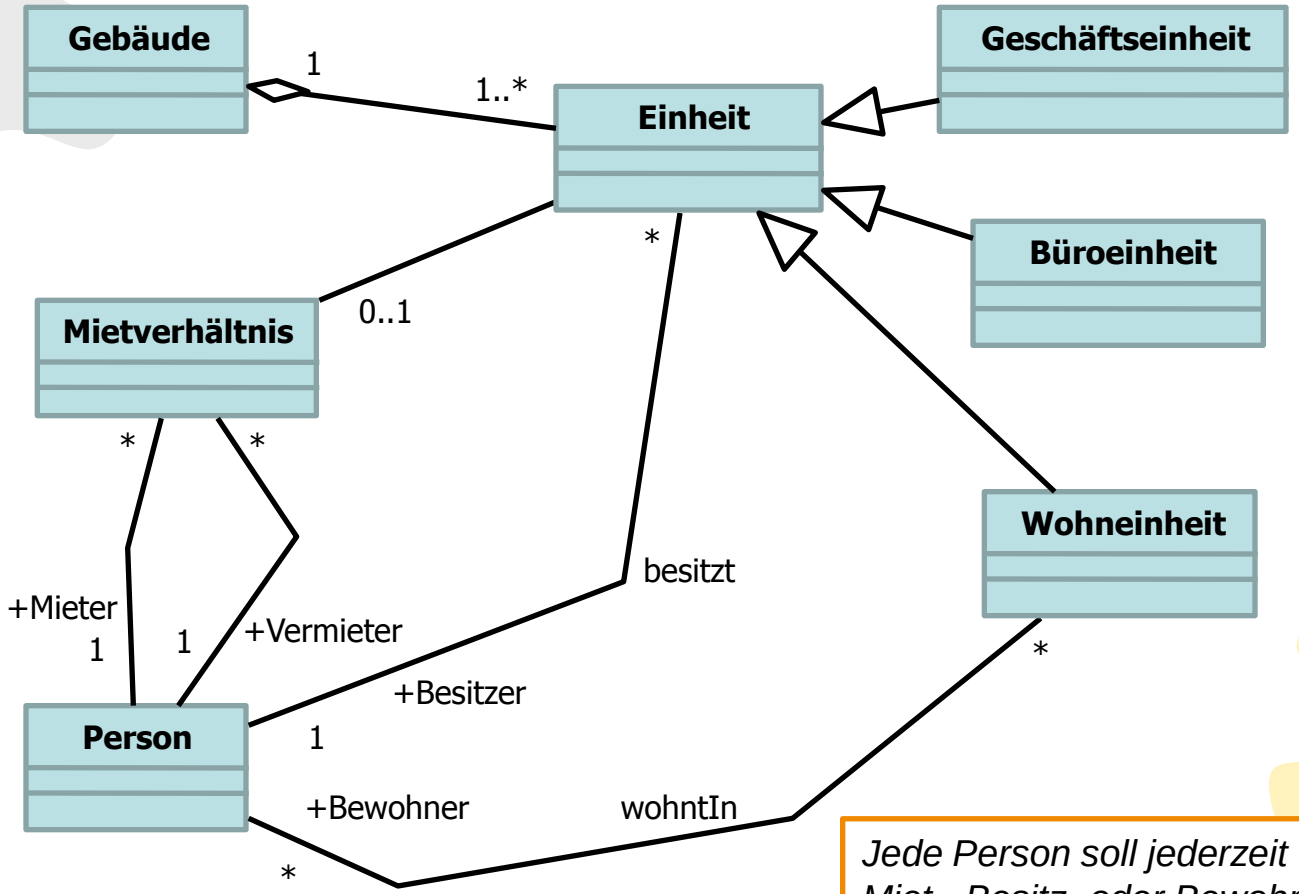
Diese Einheiten haben jeweils einen Besitzer, beliebig viele Bewohner (je nachdem!) und ggf. einen Mieter und einen Vermieter.

Objektorientierte Analyse und Modellieren



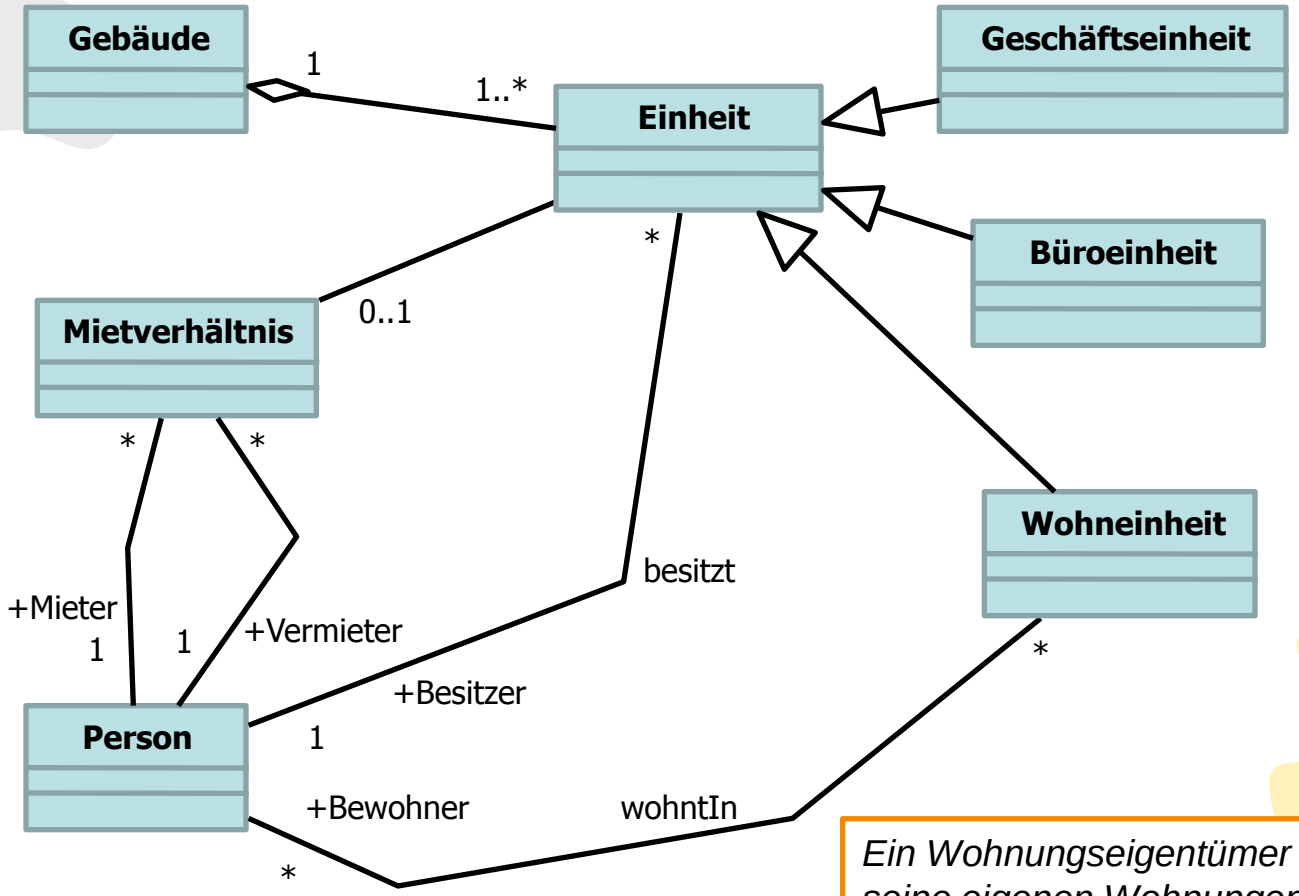
Wenn es einen Mieter gibt, muss es natürlich auch einen Vermieter geben und umgekehrt.

Objektorientierte Analyse und Modellieren



Jede Person soll jederzeit ein Miet-, Besitz- oder Bewohn-Verhältnis eingehen oder beenden können.

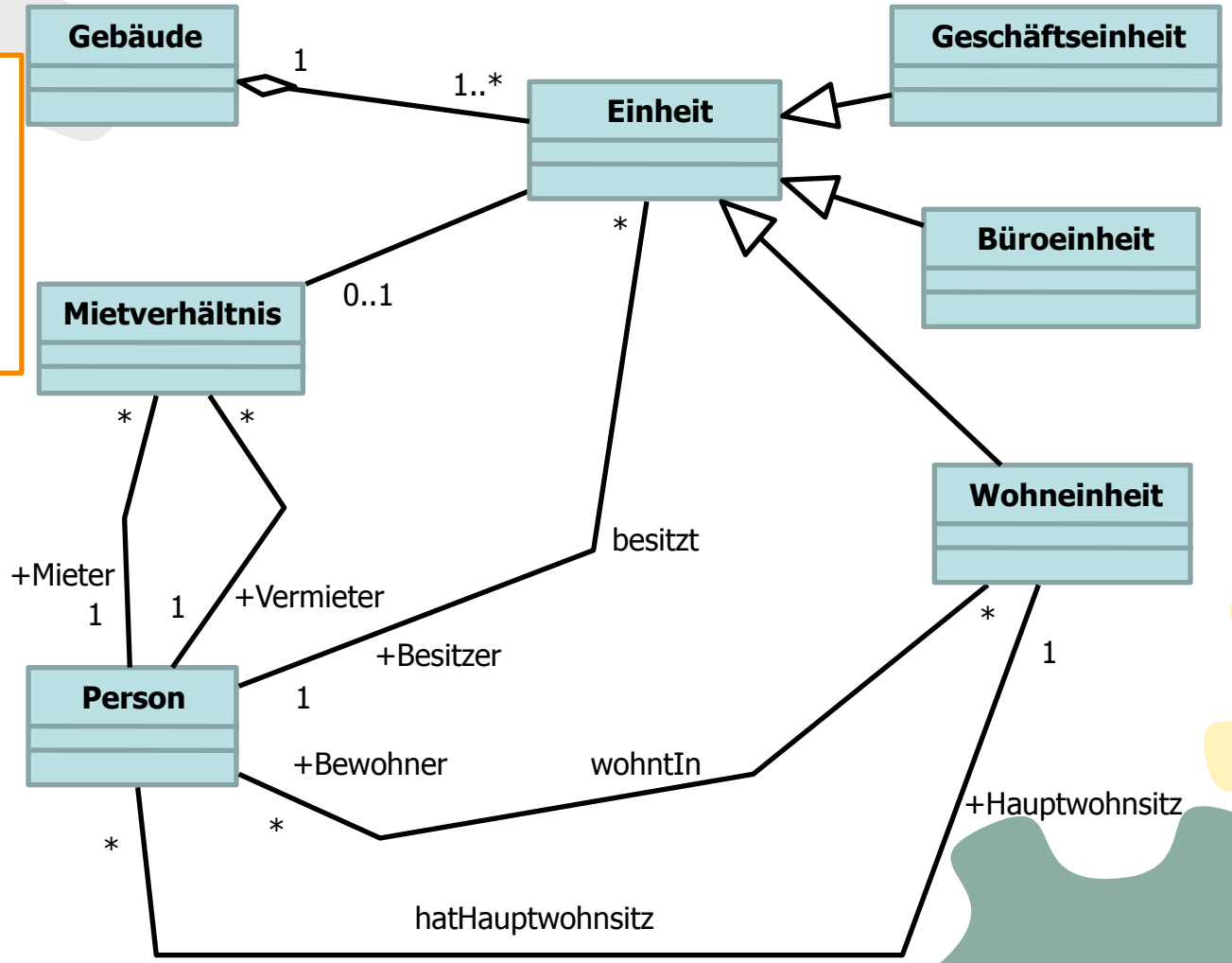
Objektorientierte Analyse und Modellieren



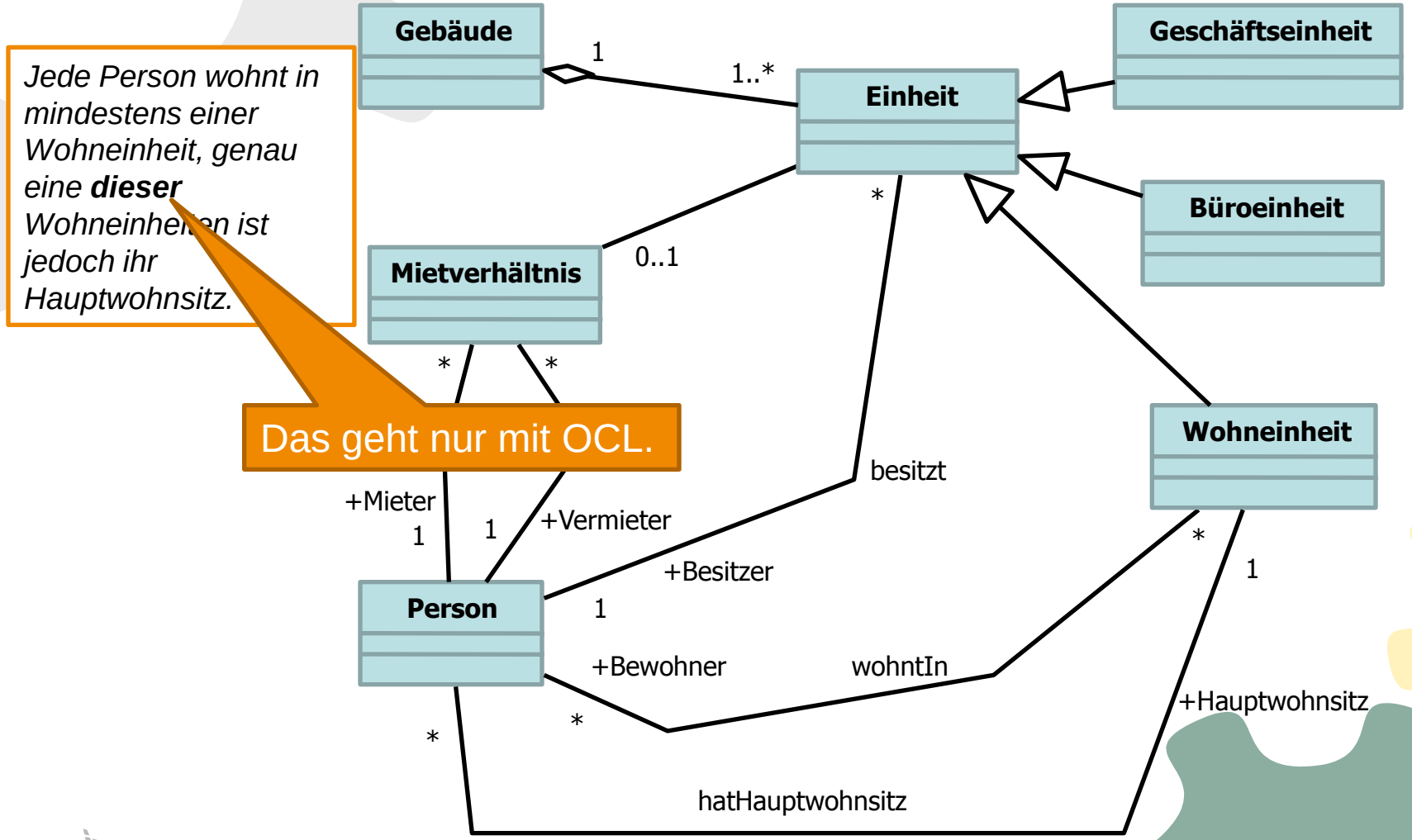
Ein Wohnungseigentümer muss seine eigenen Wohnungen bewohnen können oder fremde Wohnungen mieten können.

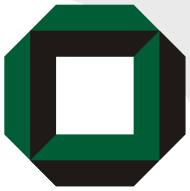
Objektorientierte Analyse und Modellieren

Jede Person wohnt in mindestens einer Wohneinheit, genau eine dieser Wohneinheiten ist jedoch ihr Hauptwohnsitz.



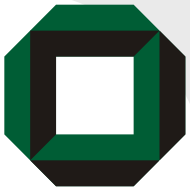
Objektorientierte Analyse und Modellieren





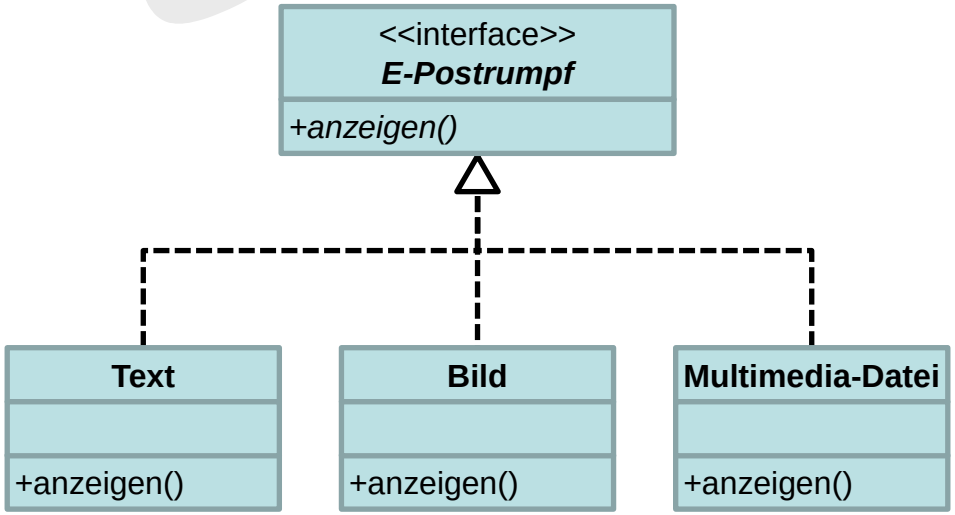
Entwurfsmuster

Gegeben ist folgendes Szenario:
Ein E-Postrumpf kann ein Text, ein Bild oder eine Multimedia-Datei sein, die alle die gemeinsame Schnittstelle E-Postrumpf implementieren. Außerdem kann ein E-Postrumpf noch aus mehreren der genannten Bestandteile bestehen. Das E-Mail-Programm soll den E-Postrumpf und seine Bestandteile einheitlich behandeln. Benutzen Sie bei Ihrem Entwurf ein Entwurfsmuster und benennen Sie dieses. Tragen Sie in Ihr Klassendiagramm die notwendigen Klassen, öffentlichen Methoden, Assoziationen und Vererbungsbeziehungen ein.



Entwurfsmuster

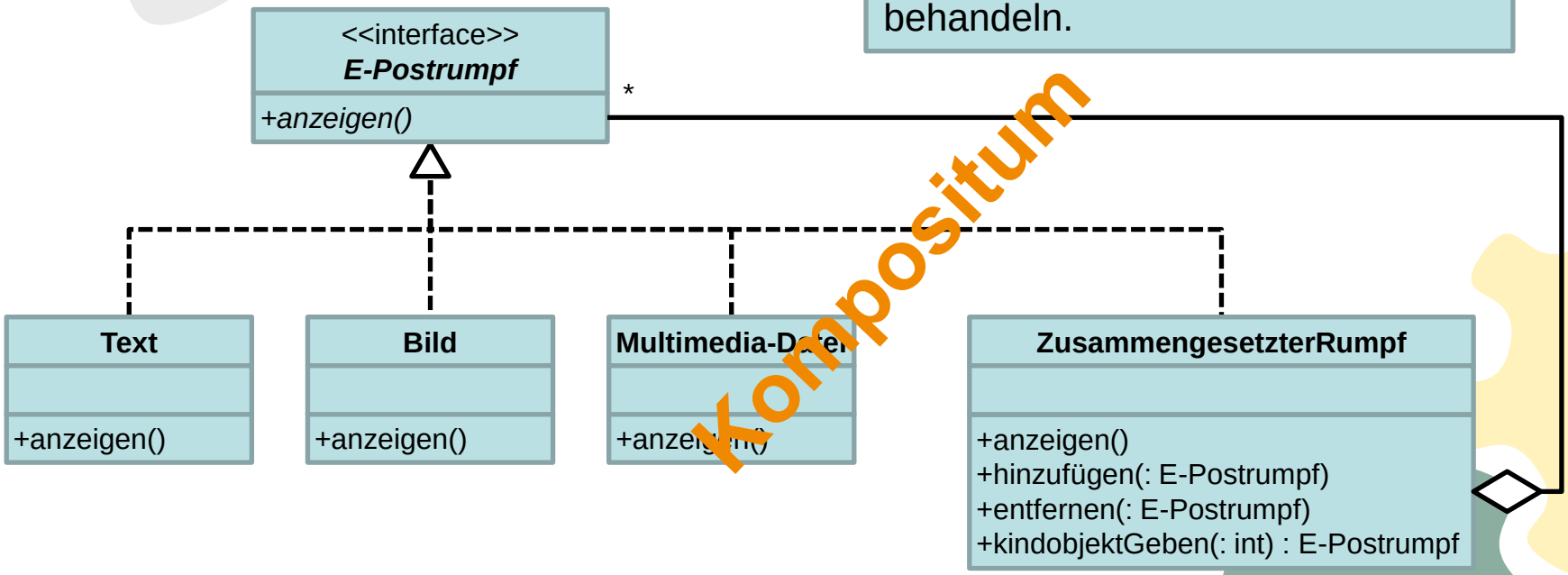
Ein E-Postrumpf kann ein Text, ein Bild oder eine Multimedia-Datei sein, die alle die gemeinsame Schnittstelle E-Postrumpf implementieren.



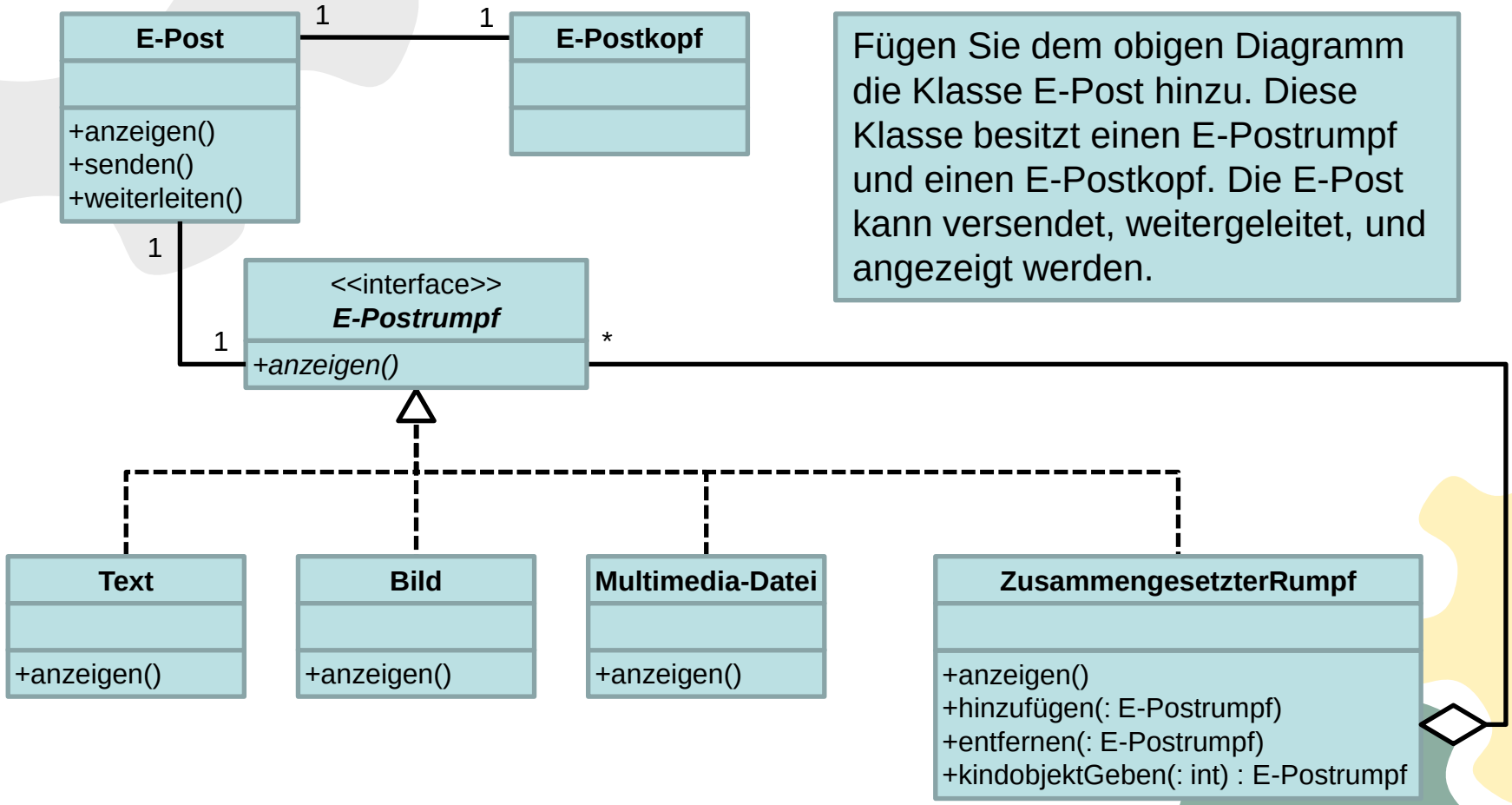
Entwurfsmuster

Wie heißt das Entwurfsmuster?

Außerdem kann ein E-Postrumpf noch aus mehreren der genannten Bestandteile bestehen. Das E-Mail-Programm soll den E-Postrumpf und seine Bestandteile einheitlich behandeln.

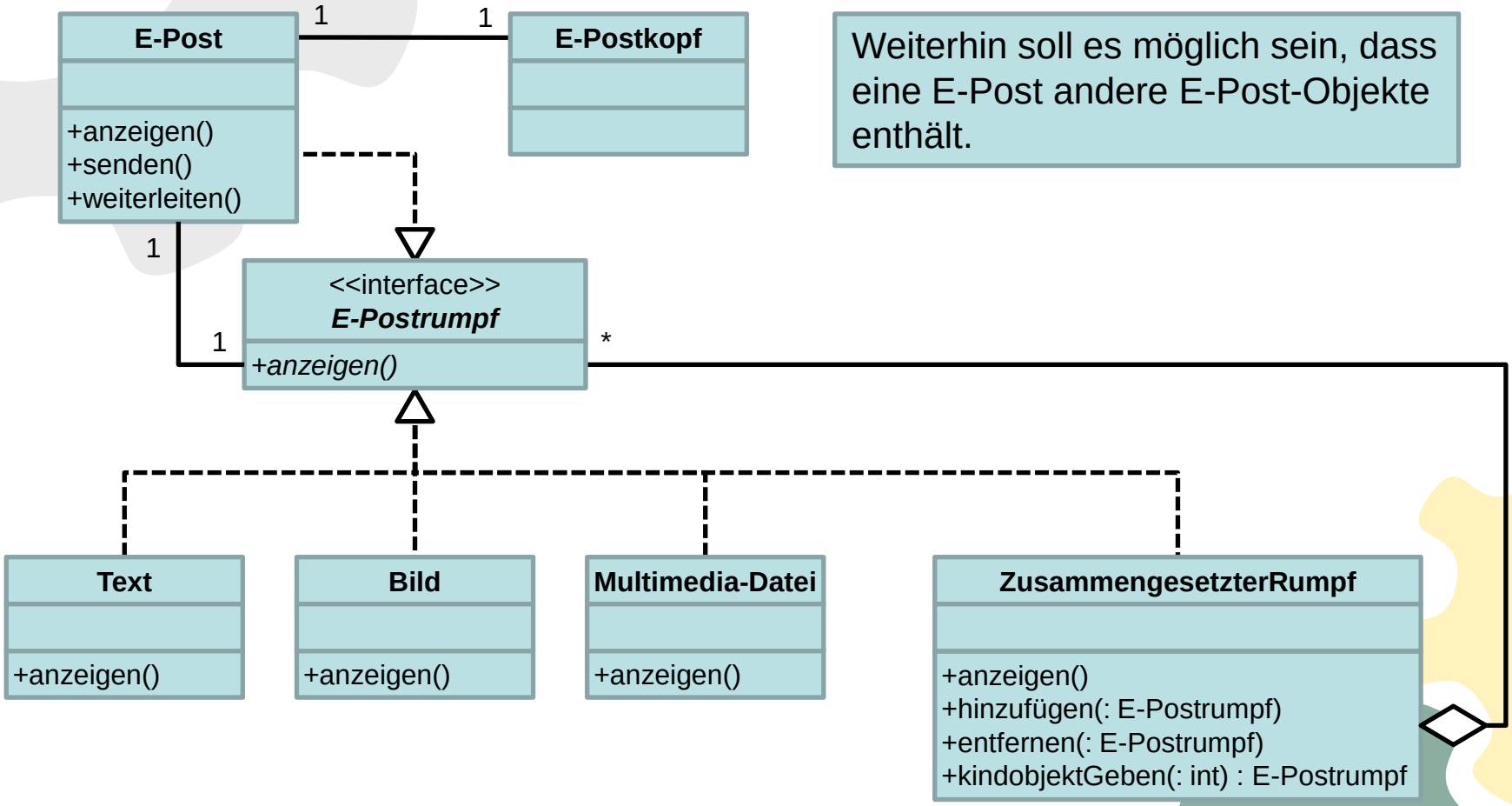


Entwurfsmuster

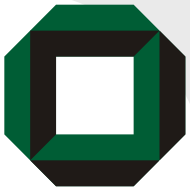


Fügen Sie dem obigen Diagramm die Klasse E-Post hinzu. Diese Klasse besitzt einen E-Postrumpf und einen E-Postkopf. Die E-Post kann versendet, weitergeleitet, und angezeigt werden.

Entwurfsmuster

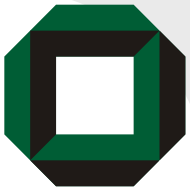


Weiterhin soll es möglich sein, dass eine E-Post andere E-Post-Objekte enthält.



Allgemeine Informationen

- Klausurtermin: 6.8.2009 14.00 Uhr; Dauer: 60 min.
- Letzte Chance für An- oder Abmeldung: 2.8.2009
- Hörsaaleinteilung wird nach Anmeldeschluss bekannt geben.
  - SWT 1-Mailingliste
  - SWT 1-Webseite
  - Aushang gegenüber Sekretariat Tichy
- Wörterbücher müssen bis spätestens am 30.7.2009 im Sekretariat zur Durchsicht abgeben werden. Sonst dürfen sie nicht verwendet werden.
- Essen und Trinken ist während der Klausur nach Möglichkeit zu unterlassen.



Allgemeine Informationen

- Lösen Sie die SWT-Klausuren der letzten Jahre (zu finden auf dem SVN-Server).
- Fragen zum Stoff der Vorlesung bitte an die SWT1-Mailingliste.