# 1 Collections

| | Set | Bag | OrderedSet | Sequence |
|---|---|---|---|---|
| mehrfach gleiche Elemente | Nein | Ja | Nein | Ja |
| Reihenfolge | Nein | Nein | Ja | Ja |

# 2 Operationen

| def | Beschreibung |
|---|---|
| ->size() | Anzahl der Elemente in der Collection |
| ->isEmpty() | Ob die Collection leer ist |
| ->includes(obj) | True, wenn das Objekt in der Collection enthalten ist |
| ->excludes(obj) | True, wenn das Objekt nicht(!) in der Collection enthalten ist |
| ->count(obj) | Zählt wie häufig das Objekt in der Collection vorkommt |

Collections ohne Reihenfolge erlauben Mengenoperationen:

| def | Beschreibung |
|---|---|
| = | Checkt ob die beiden Collections gleich sind |
| ->including(obj) | Fügt das Element zur Collection hinzu |
| ->excluding(obj) | Entfernt das Objekt von der Collection |
| ->union(otherCollection) | Vereinigt 2 Collections |
| ->intersection(otherCollection) | Schnittmenge von 2 Collections |

# 3 Collections Umwandlung

Zwischen den Collections kann umgewandelt werden (gecasted). Dazu eignen sich folgende Befehle:

- ->asSet()
- ->asBag()
- ->asOrderedSet()

# 4 Filtern, Modifizieren, Reduzieren

OCL erlaubt es eine Collection zu filtern.
Set{10,20,30}->select(i:Integer| i>20) ≡ Set{30}
Set{10,20,30}->reject(i:Integer| i<=20) ≡ Set{30}

Außerdem können wir auch die Elemente in einer Collection modifizieren:
Set{10,20,30}->collect(i:Integer| i+1) ≡ Bag{11,21,31}

Und wir können auch über die Elemente iterieren und auf einen Wert reduzieren: Set{10,20,30}->iterate(x:Integer;acc:Integer = 0 | acc+x) ≡ 60

Das Prinzip von Filtern, Modifizieren und Reduzieren wird uns wieder in Haskell begegnen. Deswegen wenn ihr es hier verstanden habt, dann fällt es euch später einfacher. Das Prinzip ist nicht nur auf Haskell bezogen, ihr findet es in jeder Programmiersprache wieder.

# 5 Quantoren

- Set{10,20,30}->forAll(x:Integer|x>10) ≡ false
- Set{10,20,30}->exists(x:Integer|x>10) ≡ true

# 6 Closure

Wenn ihr Daten akkumulieren wollt, dann könnt ihr eine Closure verwenden. OCL ist dank der Closures übrigens Turing-Vollständig. Um alle Zahlen von 1-20 aufzulisten, könnt ihr folgenden Ausdruck verwenden:

Set{1}->closure(x|if x < 20 then x+1 else x endif)
Die Closure bricht dann ab, wenn sie 2x mal das selbe Objekt hinzufügt zum Set.Dies passiert bei der 20.

# 7 Typenüberprüfung

- obj.oclIsTypeOf(Type) => Überprüft den tatsächlichen Typen des Objekts
- obj.oclIsKindOf(Type) => Überprüft ob das Objekt dem Typ entspricht (Typ oder Subtyp)

# 8 Boolische Operatoren

implies, xor, and, or, not, =, <>