



SU 3 – Design by Contract

Prof. Dr. Annika Wagner



Def. (Vertrag)

Verträge sind Einschränkungen auf einer Klasse, die es dem Klassenbenutzer, Klassenimplementierer und Klassenerweiterer ermöglichen, Annahmen über eine Klasse zu teilen.

[Meyer, 1997]

Vertrag umfasst Einschränkungen, die

- Klassenbenutzer einhalten muss
- Klassenimplementierer und Klassenerweiterer sicherstellen müssen

Arten von Einschränkungen:

• Invarianten

➡ Konsistenz der Werte von Attributen

• Vorbedingung

➡ stellt sicher, dass Operation korrekt funktioniert

• Nachbedingung

➡ garantiert Ergebnis der Operation



- Datum des letzten Kontaktes zu Kunden liegt nicht in der Zukunft

Einschränkung des Wertebereichs einer Instanzvariablen (über ihren Typ hinaus)

- Nur Kunden mit Status Neukunde können undef. Betreuer haben.

Abhängigkeit zwischen den Werten verschiedener Instanzvariablen

- Kunden mit Status Neukunde haben genau einen Geschäftsabschluss



- Kundenkontakt kann nur dokumentiert werden, wenn eine Kundenbetreuer zugewiesen ist

Einschränkung des erlaubten Wertebereichs von Instanzvariablen bei Methodenaufruf

- Kundenkontakt kann nur für heute oder einen Tag in der Vergangenheit dokumentiert werden

Einschränkung erlaubter Parameterwerte bei Methodenaufruf

- Dokumentation eines Kundenkontaktes kann nur abgeschlossen werden, wenn vorher die Aktualität der Kundendaten bestätigt wurde

Einschränkung der Reihenfolge von Methodenaufrufen



Beschreibt den veränderten Systemzustand oder Rückgabewerte

- Datum des letzten Kundenkontakts wurde auf den übergebenen Parameterwert gesetzt

Anpassung des Wertes einer Instanzvariablen (hier Attribut)

- Neukunde wurde angelegt

Anpassung des Wertes einer Instanzvariablen (hier Assoziation)

- Liefert das am nächsten in der Zukunft liegende Ablaufdatum eines Vertrages des Kunden zurück

Einschränkung des Rückgabewertes

Wie sollten Fehler mit Hilfe von Verträgen verhindert werden?



Nachbedingung muss vom Aufgerufenen garantiert werden, wenn ...

... der Aufrufer die Vorbedingung eingehalten hat

- Aufrufener **muss** mit allen
 - vertraglich erlaubten Parameterwerten rechnen
 - vertraglich erlaubten Reihenfolgen von Operationsaufrufen rechnen
- Aufrufener **darf**
 - bei Verletzung der Vorbedingung RuntimeException werfen

- Aufrufer **muss**
 - die Einhaltung der Vorbedingung sicherstellen
 - mit allen vertraglich erlaubten Rückgabewerten rechnen
- Aufrufer **darf** sich auf
 - die vertragliche Anpassung der Attributwerte des Aufgerufenen verlassen
 - die vertraglich vereinbarte Aufrufbarkeit von Operationen verlassen



- Unterscheidung bei Operationen zwischen Anfragen (Queries) und Anweisungen (Commands)
 - Anfragen liefern Werte zurück, aber verändern den Zustand (die Instanzvariablen) des Objektes nicht
 - Anweisungen liefern keine Werte zurück, sondern verändern nur den Zustand des Objektes
- Mischungen können neben den Reinformen existieren, aber nicht ausschließlich.



Methods

Modifier and Type		Method and Description
boolean	Anfrage	<code>empty()</code> Tests if this stack is empty.
E	Anfrage	<code>peek()</code> Looks at the object at the top of this stack without removing it from the stack.
E	Mischform	<code>pop()</code> Removes the object at the top of this stack and returns that object as the value of this function.
E	Anweisung	<code>push(E item)</code> Pushes an item onto the top of this stack.
int	Anfrage	<code>search(Object o)</code> Returns the 1-based position where an object is on this stack.

Prinzip 1: Anfragen in Verträgen



- NUR Anfragen in Verträgen, damit Überprüfung des Vertrags keine Seiteneffekte hat
- Anfragen müssen ausreichen, um die Effekte der Anweisungen auf die Objekte vollständig zu beschreiben



Vorbedingung: not empty() ✓

Nachbedingung:

- result = peek() ✓
- Aussehen des Stacks?

Methods

Modifier and Type	Method and Description
boolean Anfrage	empty() Tests if this stack is empty.
E Anfrage	peek() Looks at the object at the top of the stack without removing it from the stack.
E Mischform	pop() Removes the object at the top of this stack and returns that object as the value of this function.
✗ Anweisung	push(E item) Pushes an item onto the top of this stack.
int Anfrage	search(Object o) Returns the 1-based position where an object is on this stack.




Vorbedingung: keine

Nachbedingung:

- `empty() = false` ✓
- `peek() = item` ✓

Methods

Modifier and Type	Method and Description
boolean Anfrage	<code>empty()</code> Tests if this stack is empty.
E Anfrage	<code>peek()</code> Looks at the object at the top of this stack without removing it from the stack.
E Mischform	<code>pop()</code> Removes the object at the top of this stack and returns that object as the value of this function.
 Anweisung	<code>push(E item)</code> Pushes an item onto the top of this stack.
int Anfrage	<code>search(Object o)</code> Returns the 1-based position where an object is on this stack.



Methods

Modifier and Type		Method and Description
boolean	Anfrage	<code>empty()</code> Tests if this stack is empty.
E	Anfrage	<code>peek()</code> Looks at the object at the top of this stack without removing it from the stack.
E	Mischform	<code>pop()</code> Removes the object at the top of this stack and returns that object as the value of this function.
E	Anweisung	<code>push(E item)</code> Pushes an item onto the top of this stack.
int	Anfrage	<code>search(Object o)</code> Returns the 1-based position where an object is on this stack.



Methods inherited from class java.util.Vector

add, add, addAll, addAll, addElement, capacity, clear, clone,
contains, containsAll, copyInto, **elementAt** elements,
ensureCapacity, equals, firstElement, get, hashCode, indexOf,
indexOf, insertElementAt, isEmpty, iterator, lastElement,
lastIndexOf, lastIndexOf, listIterator, listIterator, remove,
remove, removeAll, removeAllElement, removeElement,
removeElementAt, removeRange, retainAll, set, setElementAt,
setSize, **size** subList, toArray, toArray, toString, trimToSize

Erlaubt Nachbedingungen für
Anfragen empty() und peek()
zu formulieren!

Prinzip 1 (ergänzt): Anfragen in Verträgen



- **NUR** Anfragen in Verträgen, damit Überprüfung des Vertrags keine Seiteneffekte hat
- **Basisanfragen** müssen ausreichen, um die Effekte der Anweisungen auf die Objekte vollständig zu beschreiben
- Unterscheidung von Basisanfragen und **abgeleiteten Anfragen**, die sich durch die Nutzung von Basisanfragen implementieren lassen
 - ➔ Nachbedingungen der erweiterten Anfragen basieren auf Basisanfragen

Prinzip1 (ergänzt) bei Stack<E>



Methods

Modifier and Type

Method and Description

boolean

Basisanfrage

`empty()`

Tests if this stack is empty.

E

Basisanfrage

`peek()`

Looks at the object at the top of this stack without removing it from the stack.

E

Mischform

`pop()`

Removes the object at the top of this stack and returns that object as the value of this function.

~~X~~

Anweisung

`push(E item)`

Pushes an item onto the top of this stack.

~~int~~

~~Anfrage~~

~~`search(Object o)`~~

~~Returns the 1-based position where an object is on this stack.~~

Setzt korrekte Funktionsweise voraus!

Prinzip1 (ergänzt) bei Stack<E>



Methods

Erlaubt Vertrag für
korrekte Funktionsweise!

Modifier and Type	Method and Description
-------------------	------------------------

boolean	empty () Tests if this stack is empty.
---------	--

Abgeleitete Anfrage

E	peek () Looks at the object at the top of this stack without removing it from the stack.
----------	--

Abgeleitete Anfrage

Methods inherited from class java.util.Vector

add, add, addAll, addAll, addElement, capacity, clear, clone, contains, contain **Basisanfrage** **elementAt** elements, ensureCapacity, equals, insertElement, get, hashCode, indexOf, indexOf, insertElementAt, isEmpty, iterator, lastElement, lastIndexOf, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, removeAllElements, removeElement, removeElementAt, retainAll, set, setElementAt, setSize, **size** **Basisanfrage** ay, toArray, toString, trimToSize



Basisanfragen
genügen, damit
Klassenbenutzer
vollständiges
Verständnis von
möglichen Zuständen
des Objektes erhält

10
15
4

size() = 3

elementAt(2) = 10

elementAt(1) = 15

elementAt(0) = 4

Prinzip 2: Nachbedingungen



- Abgeleitete Anfragen haben Nachbedingungen, in denen ihre Rückgabe durch Basisanfragen spezifiziert wird
- Anweisungen haben Nachbedingungen, in denen ihr Effekt auf den Zustand des Objektes mit Hilfe von Basisanfragen spezifiziert wird

- Sowohl für Anfragen als auch für Anweisungen sinnvoll
- Genaue Ausprägung hängt von Nutzung der Klasse ab
 - Spezielle, genau bekannte Anwendung
 - ➔ stärkere Bedingung, die es dem Klassenimplementierer leichter macht
 - Framework
 - ➔ schwächere Bedingung, die eine weitere Nutzung erlaubt

Prinzip1 bei Stack<E>

Hochschule Fulda



Vorbedingung: keine
weil Teil eines Frameworks
-> erfordert flexible Erweiterung
der Kapazität

Einfachere Implementierung
würde feste Kapazität vorsehen

Methods		
Modifier and Type	Method and Description	
boolean	empty ()	Tests if this stack is empty.
	Anfrage	
E	peek ()	Looks at the object at the top of this stack without removing it from the stack.
	Anfrage	
E	pop ()	Removes the object at the top of this stack and returns that object as the value of this function.
	Mischform	
 E	push (E item)	Pushes an item onto the top of this stack.
	Anweisung	
int	search (Object o)	Returns the 1-based position where an object is on this stack.
	Anfrage	