BERGISCHE
UNIVERSITÄT
WUPPERTAL

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik/Numerik

Fachbereich C
Mathematik und Naturwissenschaften,
Mathematik und Informatik

E-MAIL buhl@math.uni-wuppertal.de
WWW www.math.uni-wuppertal.de/~buhl

DATUM 8. Mai 2013

Formale Methoden

SS 2013 – Übungsblatt 4
Ausgabe: 10. Mai 2013
Abgabe bis 16. Mai 2013 an: *dsavvidi+fm@studs.math.uni-wuppertal.de*

Aufgabe 1. *ParameterDirectionKind, subsets ownedRule*

Welche C++-Typmodifizierer werden (vermutlich) bei der Abbildung von Modellen auf Code-Eigenschaften für die UML-`ParameterDirectionKinds` (vgl. http://msdn.microsoft.com/de-de/library/microsoft.visualstudio.uml.classes.parameterdirectionkind.aspx) genutzt?

Lesen Sie in Abschnitt 7 (Classes) der UML 2.4.1 Superstructure Specification das Diagramm 7.11. Erläutern Sie die `subsets`-Eigenschaft der Rollen `precondition` und `postcondition`.

Warum ist es wichtig, für (fast) alle Methoden (Operationen) Vor- und Nachbedingungen explizit anzugeben?

Aufgabe 2. *Redesign Sparbuch*

Designen Sie das Sparbuch aus Übungsblatt 2 neu nach den Prinzipien objektorientierter Programmierung (abstrakte Klasse `Waehrung`, abstrakte Klasse `Konto`, ...) und der SdV (welche grundlegenden Observatoren sind nötig, welche Modifikatoren, ...). Konstruieren Sie in `papyrus` die nötigen Diagramme. Vergessen Sie nicht entsprechende `subsets`-Eigenschaften der verschiedenen Buchungstypen explizit anzugeben.

Konzipieren Sie einen «`datatype`» Datum und eine Methode Zinstage, die die Anzahl der wirksamen Zinstage jedes Datums nach der kaufmännischen Zinsrechnung (12 * 30 Tage) bestimmt. Wie kann das Ergebnis dieser Methode in OCL spezifiziert werden? Welche Invariante sollte `Datum` besitzen?

Aufgabe 3. *OclVoid und null; Real*

Informieren Sie sich im OCL2.3.1-Manual über den Datentyp `OclVoid`. Wozu kann er benutzt werden?
Welche Operationen gibt es für den `primitive type` Real in der OCL-Spezifikation? Erläutern Sie die Nachbedingung der Operation `round()`. Warum wird in dieser die Zeichenfolge `"abs()"` so unüblich benutzt?

Aufgabe 4. *Ein Code-Contract*

Erläutern Sie die Prinzipien der SdV

Spezifikation durch Vertrag - eine Basistechnologie ...

an folgendem Beispiel:

```
...
template <class T>
set<T> operator+(const set<T>& s, const T& e){
    ...
};
...
template <class T>
set<T> operator-(const set<T>& s, const T& e){
    ...
};
...
class mydictionary{

public:
  //////////////// basic queries:

  unsigned int get_count() const;      // number of key/value-pairs in dict.

  bool has(const KEY& k) const;                // key in dictionary?

  VALUE value_for(const KEY& k) const;       // lookup value for key

  //////////////// class invariant:
private:
  virtual bool invariant() const;
public:

  //////////////// derived queries:
  // not yet necessary

  //////////////// constructors and destructors:

  mydictionary();

  ~mydictionary();

  //////////////// copy constructor

  mydictionary(const mydictionary<KEY, VALUE>& s);


  //////////////// deactivate operator=
```

```
private:
  mydictionary& operator=(const mydictionary<KEY, VALUE>& s);
public:

  //////////////  (pure) modificators
  void put(const KEY& k, const VALUE& v);
                                  // put key/value-pair in dict.

  void remove(const KEY& k);
                                 // remove key/value-pair

};
...
template<class KEY, class VALUE>
unsigned int mydictionary<KEY, VALUE>::get_count() const{
  REQUIRE( invariant() );
  ...
};
...
template<class KEY, class VALUE>
bool mydictionary<KEY, VALUE>::has(const KEY& k) const {
  REQUIRE( invariant() );
  ...
  ENSURE( /* consistent with count */  (get_count() != 0) || ! result);
  ...
};
...
template<class KEY, class VALUE>
VALUE mydictionary<KEY, VALUE>::value_for(const KEY& k) const{
  REQUIRE( invariant() );
  REQUIRE(/* key in dict. */     has(k));
  ...
};
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::mydictionary(){
  ...
  ENSURE(invariant());
  ENSURE(count == 0);
};
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::~mydictionary(){
  REQUIRE( invariant() );
  ...
};
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::mydictionary(const
                            mydictionary<KEY, VALUE>& s){
    ...
    ENSURE(get_count() == s.get_count());
    ...
    ENSURE(invariant());
};
```

```
...
template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::put(const KEY& k, const VALUE& v)
DO
  REQUIRE(/* key not in dict. */   ! has(k));
  ID(unsigned int count_old=get_count());
  ...
  ENSURE(/* count incremented */    get_count() == count_old + 1);
  ENSURE(/* key in dict. */         has(k) );
  ENSURE(/* correct value */        value_for(k) == v);
END;
...
template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::remove(const KEY& k)
DO
  REQUIRE(/* key in dict. */    has(k));
  ID(unsigned int count_old = get_count());
  ...
  ENSURE(/* count decremented */  get_count() == count_old - 1);
  ENSURE(/* key not in dict. */   ! has(k));
  ...
END;
...
int main(){
  ...
}
```

Die Abfrage `has(k)` reicht aus, genau spezifizieren zu können, wann `put(k,v)` und `remove(k)` aufgerufen werden dürfen und welchen Haupteffekt ihre Aufrufe jeweils haben.

Sie reicht jedoch nicht aus, um die vollständige Wirkung von `put()` beziehungsweise `remove()` spezifizieren zu können (Framebedingung). Wie müssten die grundlegenden Observatoren geändert werden, um auch dies zu erlauben?

Aufgabe 5. *Design by contract*

Der Artikel

Design by Contract (insbesondere Seite 9)

beschreibt ebenfalls die Methodik der Codeverträge.

Welche Programmiersprachen unterstützen diese Methodik von Haus aus, welche erst mit Hilfe zusätzlicher externer Tools?