BERGISCHE UNIVERSITÄT
WUPPERTAL
GAUSS-STRASSE 20
42119 WUPPERTAL
TELEFAX (0202) 439-2901
TELEFON (0202) 439-0
WWW www.uni-wuppertal.de

Fachbereich C

MATHEMATIK UND NATURWISSENSCHAFTEN
Fachgruppe Mathematik und Informatik

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

E-Mail: Hans-Juergen.Buhl@math.uni-wuppertal.de
WWW: http://www.math.uni-wuppertal.de/∼buhl

Formale Methoden

WS 2007/2008 – Übungsblatt 2

6. November 2007
Ausgabe: 9. November 2007

Aufgabe 1. *Ein Contract*

Erläutern Sie die Prinzipien des DbC/PbC 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{

  vector<KEY>* keys;
  vector<VALUE>* values;
  unsigned int count;

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);
  ENSURE(" get_count() == # { k in KEY | has(k) } ");
  ...
};
...
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>
bool mydictionary<KEY, VALUE>::invariant() const{
  ...
};
...
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(count == s.count);
    ENSURE(A(int i=1, i<=count, i++, keys->at(i-1) == s.keys->at(i-1)));
    ENSURE(A(int i=1, i<=count, i++, values->at(i-1) == s.values->at(i-1)))
    ...
    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());
  ID(set<KEY> old_keys(keys->begin(), keys->begin()+count_old));
  ...
  ENSURE(/* count incremented */    get_count() == count_old + 1);
  ENSURE(/* key in dict. */         has(k) );
  ENSURE(/* correct value */        value_for(k) == v);
  ID(set<KEY> new_keys(keys->begin(),keys->begin()+count));
  ENSURE(old_keys + k == new_keys);
  ENSURE("Framebedingung: fuer alle k in old_keys:  value_for(k) == ...");
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());
  ID(set<KEY> old_keys(keys->begin(), keys->begin()+count_old));
  ...
  ENSURE(/* count decremented */  get_count() == count_old - 1);
  ENSURE(/* key not in dict. */   ! has(k));
  ...
  ID(set<KEY> new_keys(keys->begin(),keys->begin()+count));
  ENSURE(old_keys - k == new_keys);
  ENSURE("Framebedingung");
END;
...
int main(){
  ...
}
```

Aufgabe 2. *Subcontracting*

Erläutern Sie, warum die Nachbedingung eines Modifikators einer „is-a“ Unterklasse im Falle der Gültigkeit der Vaterklassenvorbedingung nicht schwächer sein darf als die Vaterklassennachbedingung, jedoch andernfalls „beliebig“ sein darf:

```
------------------------------ Fussgaengerbruecke
   QUERIES
      MaxLast : REAL
      AktLast : REAL
   INVARIANTS
      MaxLast >= 7500
      AktLast <= MaxLast
   ACTIONS
      ueberquereBruecke( IN gew : REAL,
                         OUT Guthaben : INTEGER )
         PRE
            gew  + AktLast <= MaxLast
            gew <= 200
            Guthaben >= 2
         POST
            AktLast = OLD(AktLast) + gew
            Guthaben = OLD(Guthaben) - 2
      verlasseBruecke( IN gew : REAL )
         ...
```

sowie ein Subcontract:

```
------------------------------ Autobruecke
   QUERIES
```

```
   MaxLast : REAL
   AktLast : REAL
INVARIANTS
   MaxLast >= 800000
   AktLast <= MaxLast
ACTIONS
   ueberquereBruecke( IN gew : REAL,
                      OUT Guthaben : INTEGER )
      PRE
         gew  + AktLast <= MaxLast
         gew <= 20000
         Guthaben >= 20
      POST
         AktLast = OLD(AktLast) + gew
         OLD(gew) <= 200     IMPLIES Guthaben = OLD(Guthaben) - 2
         NOT OLD(gew) <= 200 IMPLIES Guthaben = OLD(Guthaben) - 20
   verlasseBruecke( IN gew : REAL )
      ...
```

Aufgabe 3. *Redesign Sparbuch*

Designen Sie das Sparbuch aus Übungsblatt 1 neu nach den Prinzipien des DbC/PbC.

Aufgabe 4. *Ein UML-Modell*

Konzipieren und konstruieren Sie ein Klassenmodell im Umfeld Bestellung/Lieferschein/Rechnung.