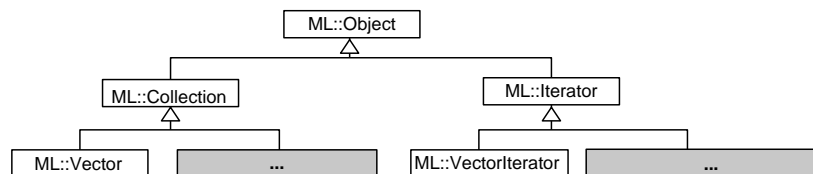


<input type="checkbox"/>	Gr. 1, DI Franz Gruber-Leitner	Name	<u>Roman Lumetsberger</u>	Aufwand in h	<u>7</u>
<input checked="" type="checkbox"/>	Gr. 2, Dr. Erik Pitzer	Punkte	_____	Kurzzeichen Tutor / Übungsleiter	_____/____

## MiniLib

Studieren Sie die abstrakten Basisklassen *ML::Collection* und *ML::Iterator*. Derzeit gibt es zwei davon abgeleitete konkrete Klassen *ML::Vector* und *ML::VectorIterator*, die im Wesentlichen ein dynamisches Feld realisieren. Die Klassenhierarchie hat derzeit somit folgendes Aussehen:



### 1. Objektmengen (sets)

(12 Punkte)

Realisieren Sie auf Basis der beiden Klassen *ML::Collection* und *ML::Iterator* zwei neue Klassen zur Repräsentation von Mengen (engl. *sets*): Entwerfen Sie eine Klasse *Set* und eine Klasse *SetIterator*. Die Klasse *Set* soll Mengen von Objekten verwalten können. Wie bei Mengen üblich, darf kein Objekt mehrmals in einem *Set* vorkommen. Überlegen Sie, welche Methoden notwendig sind, und vergessen Sie dabei nicht, auch Methoden zur Bildung der Schnittmenge, der Vereinigung und der Differenz zweier Mengen zu realisieren. Die Klasse *SetIterator* soll das "Iterieren" über die Elemente eines *Sets* ermöglichen.

Können Sie die Klassen implementieren ohne Änderungen an der MiniLib vorzunehmen? Wenn nein, welche Änderungen sind notwendig?

Dokumentieren Sie Ihren Entwurf, und zeigen Sie auf, welche Möglichkeiten sich Ihnen geboten haben, und warum Sie sich für die von Ihnen gewählte entschieden haben.

Implementieren Sie die entworfenen Klassen mit allen Methoden und testen Sie diese ausführlich. Wenn Sie Änderungen an der MiniLib vornehmen mussten, geben Sie bitte nur die geänderten Teile ab und heben Sie die Änderungen im Ausdruck hervor (unterstreichen, Leuchtmarker, ...).

### 2. Objektbehälter (bags)

(12 Punkte)

Entwerfen Sie eine neuerliche Erweiterung der MiniLib zur Repräsentation eines allgemeinen "Objektbehälters" (engl. *bag*). Eine neue Klasse *Bag* (gemeinsam mit einer neuen Iteratorklasse *BagIterator*) soll eine Sammlung von Objekten realisieren, auf welche die gleichen Operationen wie für *Sets* anwendbar sind. Objekte der Klasse *Bag* unterscheiden sich aber von Objekten der Klasse *Set* darin, dass in *Bags* Objekte mehrmals vorkommen dürfen. Implementieren Sie die von Ihnen entworfenen Klassen und achten Sie dabei auf eine möglichst platzsparende Speicherung von mehrfach vorkommenden Objekten. Vergessen Sie nicht, Ihre Entwurfsentscheidungen, eventuelle Probleme und Einschränkungen zu dokumentieren. Testen Sie Ihre Erweiterungen der Klassenbibliothek.

# 1 Aufgabe 1 - Objektmengen (sets)

## 1.1 Anmerkungen

Diese Aufgabe wird mit der *minilib* umgesetzt und dadurch werden alle Methoden mit einem großen Anfangsbuchstaben benannt, da dies in der *minilib* Konvention ist.

## 1.2 Lösungsidee

Ein *Set* ist im Grunde eine Liste, die keine mehrfachen Elemente enthalten darf. Darum wird die Klasse *Set* als Ableitung der Klasse *List*, die wir in der Übung erstellt haben, umgesetzt.

Es wird also eine doppelt verkettete Liste verwendet, die als Element die Klasse *Node* hat. Diese Klasse enthält dann das eigentliche Datenobjekt.

Es braucht dann nur die Methode *Add* überschrieben werden und um die Prüfung, ob ein Objekt bereits in der Liste vorhanden ist, erweitert werden. Diese Prüfung wird mit Hilfe der *minilib* Methode ***IsEqualTo*** realisiert.

### 1.2.1 Vereinigung - Union

Die Vereinigung zweier Mengen enthält alle Elemente der ersten und alle Elemente der zweiten Menge. Wichtig ist, dass jedes gleiche Element nur einmal in der Vereinigungsmenge vorkommt.

### 1.2.2 Schnittmenge - Intersect

Die Schnittmenge zweier Mengen enthält alle Elemente, die sowohl in der ersten, als auch in der zweiten Menge vorhanden sind.

### 1.2.3 Differenz - Difference

Die Differenz zweier Mengen enthält jene Elemente der ersten, aber nicht in der zweiten Menge vorkommen.

## 1.3 Entwurfsentscheidungen

Die Lösung dieser Aufgabe erfordert grundsätzlich keine Änderung der *minilib*, da beim Einfügen eines bereits im *Set* enthaltenen Objektes eine Fehlermeldung auf *cerr* ausgegeben wird. Zugegeben, das ist nicht die beste Lösung, doch in der kurzen Zeit von nur 1 Woche für die gesamte Übung, habe ich diese Variante gewählt.

Weiters kann der Verwender der Klasse mit *Contains* prüfen, ob das Element bereits vorhanden ist.

### 1.3.1 Weitere Lösungsmöglichkeiten

Folgende Lösungen wären auch möglich gewesen.

- Änderung des Rückgabeparameters der Methode *Collection::Add* von *void* auf *bool*, um den Aufrufer mitzuteilen, ob das Objekt eingefügt wurde oder nicht.
- Änderung des Rückgabeparameters der Methode *Collection::Add* von *void* auf *Object \**. Der Rückgabewert würde dann das bereits enthaltene Objekt, falls es bereits vorhanden war, oder eben das einzufügende Objekt liefern.  
Damit könnte der Aufrufer unterscheiden, ob es bereits vorhanden war oder nicht und bekäme auch noch eine Referenz auf das im *Set* befindliche Element.

### 1.3.2 Implementierung Union, Intersect, Difference

Da in der Angabe nicht konkret erwähnt wird, wie die Signatur der Methoden auszusehen haben, wurden sie so umgesetzt, dass sie als Rückgabewert ein neues *Set* liefern, das dann das Ergebnis der Operation beinhaltet.

Dies hat den Vorteil, dass das originale *Set* nicht verändert wird und somit weiterverwendet werden

kann.

**Dabei werden die Datenelemente aber nicht kopiert, d.h. der Aufrufer darf nicht *DeleteElements* auf das Original und Ergebnis anwenden.**

### 1.3.3 SetIterator

Laut Angabe wird eine eigene Klasse *SetIterator* verlangt. Dieser wurde **nicht implementiert**, da das Set von *List* abgeleitet wurde und der dort vorhandene *ListIterator* bereits alle Anforderungen erfüllt, um die Elemente zu durchlaufen.

## 2 Aufgabe 2 - Objektbehälter (bags)

### 2.1 Anmerkungen

Auch diese Aufgabe wird mit der *minilib* umgesetzt und dadurch werden alle Methoden mit einem großen Anfangsbuchstaben benannt, da dies in der *minilib* Konvention ist.

### 2.2 Lösungsidee

Ein Bag ist ein Set, indem die Elemente mehrfach vorkommen können.

Diese Klasse wird durch ableiten der Klasse *Set* realisiert. Dabei wird auch ein neuer Knotentyp *BagNode* von *Node* abgeleitet und durch eine weitere Datenkomponente *count* erweitert. Dies wurde so gemacht, da gefordert ist, dass die mehrfachen Elemente möglichst platzsparend gespeichert werden sollen.

Folgende Methoden müssen überschrieben werden, um die Klasse korrekt zu implementieren:

- Add: Mehrfache Elemente zulassen und inkrementieren der Datenkomponente *count*.
- Remove: Dekrementieren von *count* oder löschen des Knoten ( bei *count* = 0 ).

#### 2.2.1 BagIterator

Der *BagIterator* muss für diese Aufgabe implementiert werden, da dieser die mehrfachen Elemente auch mehrfach liefern muss. Der *ListIterator* würde nämlich jedes Objekt nur einmal liefern, da er nur die Knoten durchläuft.

#### 2.2.2 Vereinigung - Union

Die Vereinigung zweier *Bags* enthält alle Elemente der ersten und alle Elemente der zweiten Menge.

#### 2.2.3 Schnittmenge - Intersect

Die Schnittmenge zweier *Bags* enthält alle Elemente, die sowohl in der ersten, als auch in der zweiten Menge vorhanden sind. Sollte ein Element mehrfach vorkommen, dann wird das Minimum genommen.

#### 2.2.4 Differenz - Difference

Die Differenz zweier *Bags* wird als A ohne B implementiert. Dabei wird die Anzahl der Vorkommen in B von der Anzahl in A subtrahiert. Kommt das Element in B nicht vor, dann wird die Anzahl von A ins Ergebnis übernommen.

### 2.3 Entwurfsentscheidungen

Die Methoden *Union*, *Intersect*, *Difference* wurden in *Bag* mit der oben angegebenen Logik neu implementiert. Hier könnte man sich vielleicht überlegen, diese in *Collection* zu definieren, damit sie für alle *Collections* anwendbar sind.

## 2.4 Sourcecode

### List.h

```
1  /*****
2   List.h
3   Roman Lumetsberger
4
5   Header for class List, ListIterator
6   *****/
7  #ifndef LIST_H
8  #define LIST_H
9
10 #include <MLCollection.h>
11 #include "Node.h"
12
13 namespace ML {
14
15 class List : public Collection
16 {
17 protected:
18     Node *head;
19     int size;
20
21     virtual Node *Find(Object *o) const;
22     //Method used for creating the list node
23     virtual Node *CreateNode(Object *o) const;
24
25 public:
26     List();
27     virtual ~List();
28
29     virtual int Size() const;
30     virtual void Add(Object *o);
31     virtual Object *Remove(Object *o);
32     virtual bool Contains(Object *o) const;
33     //clears collection, does not delete elements
34     virtual void Clear();
35     virtual Iterator *NewIterator() const;
36
37 };
38
39 class ListIterator : public Iterator {
40     //allow NewIterator to call the private constructor
41     friend Iterator *List::NewIterator() const;
42
43 private:
44     Node *current;
45 private:
46     ListIterator(Node *head);
47
48 public:
49     virtual ~ListIterator();
50     Object *Next();
51 };
52
53 }
54 #endif // LIST_H
```

## Set.h

---

Seite 5

```
19     virtual ~Set();
20     virtual void Add(Object *o) override;
21
22     virtual Set *Union(Set *other) const;
23     virtual Set *Intersect(Set *other) const;
24     virtual Set *Difference(Set *other) const;
25 };
26
27 }
28 #endif // SET_H
```

---

## BagNode.h

```
1  /*****
2   BagNode.h
3   Roman Lumetsberger
4
5   Header for class BagNode
6   *****/
7  #ifndef BAGNODE_H
8  #define BAGNODE_H
9
10 #include "Node.h"
11
12 namespace ML {
13
14 class BagNode : public Node
15 {
16     protected:
17
18     public:
19         int count; //public, because only used in bag
20
21         BagNode(Object *value = nullptr,
22                 BagNode *prev = nullptr,
23                 BagNode *next = nullptr);
24         virtual ~BagNode();
25
26         std::string AsString() const override;
27
28 };
29 }
30 #endif // BAGNODE_H
```

---

## Bag.h

```
1  /*****
2   Bag.h
3   Roman Lumetsberger
4
5   Header for class Bag, BagIterator
6   *****/
7  #ifndef BAG_H
8  #define BAG_H
9
10 #include "BagNode.h"
```

```
11 #include "Set.h"
12
13 namespace ML {
14
15 class Bag : public Set
16 {
17     protected:
18         virtual BagNode *Find(Object *o) const override;
19         virtual BagNode *CreateNode(Object *o) const override;
20     public:
21         Bag();
22         virtual ~Bag();
23
24         virtual void Add(Object *o) override;
25         virtual Object *Remove(Object *o) override;
26         //Needed, because Collection would delete objects twice
27         virtual void DeleteElements() override;
28
29         virtual Bag *Union(Bag *other) const;
30         virtual Bag *Intersect(Bag *other) const;
31         virtual Bag *Difference(Bag *other) const;
32
33         virtual Iterator *NewIterator() const;
34 };
35
36 class BagIterator : public Iterator {
37     //allow NewIterator to call the private constructor
38     friend Iterator *Bag::NewIterator() const;
39
40     private:
41         BagNode *current;
42         int currentNodeCount;
43
44     private:
45         BagIterator(BagNode *head);
46
47     public:
48         virtual ~BagIterator();
49         Object *Next();
50 };
51
52
53 }
54 #endif // BAG_H
```

---

## List.cpp

```
1  /*****
2   List.cpp
3   Roman Lumetsberger
4
5   Implementation for class List, ListIterator
6   *****/
7  #include "List.h"
8  #include <cassert>
9  namespace ML {
10
11  List::List() : head(nullptr), size(0) {
```

```
12 Register("List", "Collection");
13 }
14
15 List::~List() {
16     Clear();
17 }
18
19 int List::Size() const {
20     return size;
21 }
22
23 Node *List::Find(Object *o) const {
24     assert(o != nullptr);
25     Node *cur = head;
26     while(cur != nullptr &&
27           !o->IsEqualTo(cur->value)) {
28         cur = cur->next;
29     }
30     return cur;
31 }
32
33 Node *List::CreateNode(Object *o) const {
34     return new Node(o);
35 }
36
37 Object *List::Remove(Object* o) {
38     assert(o != nullptr);
39     Node *n = Find(o);
40
41     if(n == nullptr) return nullptr;
42     if(n == head) {
43         head = head->next;
44     }
45
46     if(n->prev != nullptr) n->prev->next = n->next;
47     if(n->next != nullptr) n->next->prev = n->prev;
48
49     size--;
50     Object *value = n->value;
51     delete n;
52     return value;
53 }
54
55 bool List::Contains(Object* o) const {
56     return Find(o) != nullptr;
57 }
58
59 void List::Clear() {
60     Node * current = head;
61     while(current != nullptr) {
62         Node *tmp = current;
63         current = current->next;
64         delete tmp;
65     }
66     head = nullptr;
67     size = 0;
68 }
69
```



```
70 Iterator *List::NewIterator() const {
71     return new ListIterator(head);
72 }
73
74 void List::Add(Object* o) {
75     assert(o != nullptr);
76     Node *n = CreateNode(o);
77     size++;
78
79     if( head == nullptr) {
80         head = n;
81     }
82     else {
83         Node *last = head;
84         while(last->next != nullptr) {
85             last = last->next;
86         }
87
88         last->next = n;
89         n->prev = last;
90     }
91 }
92
93 ListIterator::ListIterator(Node *head) :current(head) {
94     Register("ListIterator","Iterator");
95 }
96
97 ListIterator::~ListIterator() {}
98 Object *ListIterator::Next() {
99     if(current == nullptr) return nullptr;
100     Object *o=current->value;
101     current = current->next;
102     return o;
103 }
104
105
106 }
```

---

## Node.cpp

---

```
1  /*****
2   Node.cpp
3   Roman Lumetsberger
4
5   Implementation for class Node
6   *****/
7  #include "Node.h"
8
9  using namespace std;
10
11 namespace ML {
12
13 Node::Node(Object * value, Node *prev, Node* next)
14 : value(value), prev(prev), next(next) {
15     Register("Node","Object");
16 }
17
18 Node::~Node(){
```

```
19  /* nothing todo */
20  }
21
22  std::string Node::AsString() const {
23      if( value == nullptr)
24          return "<nullptr>";
25
26      return value->AsString();
27  }
28
29  }
```

---

## Set.cpp

---

```
1  *****
2  Set.cpp
3  Roman Lumetsberger
4
5  Implementation for class Set
6  *****/
7  #include "Set.h"
8  #include "List.h"
9  #include <cassert>
10 #include <iostream>
11
12 using namespace std;
13
14 namespace ML {
15
16 Set::Set(): List() {
17     Register("Set","List");
18 }
19
20 Set::~Set() {
21     /* nothing todo */
22 }
23 void Set::Add(Object* o) {
24     if(!List::Contains(o)) {
25         List::Add(o);
26     }
27     else {
28         cerr << o->AsString() << " already in this set!" << endl;
29     }
30 }
31
32 Set *Set::Union(Set* other) const {
33     assert(other != nullptr);
34     Set *result = new Set();
35
36     //add elements from first set
37     Iterator *it = NewIterator();
38     for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
39         if(!result->Contains(o)) {
40             result->Add(o);
41         }
42     }
43     delete it;
44 }
```

```
45 //add elements from second set;
46 it = other->NewIterator();
47 for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
48     if(!result->Contains(o)) {
49         result->Add(o);
50     }
51 }
52 delete it;
53 return result;
54 }
55
56 Set *Set::Intersect(Set* other) const {
57     assert(other != nullptr);
58     Set *result = new Set();
59
60     Iterator *it = NewIterator();
61     for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
62         if(other->Contains(o)) {
63             result->Add(o);
64         }
65     }
66     delete it;
67     return result;
68 }
69
70 Set *Set::Difference(Set* other) const {
71     assert(other != nullptr);
72     Set *result = new Set();
73
74     Iterator *it = NewIterator();
75     for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
76         if(!other->Contains(o)) {
77             result->Add(o);
78         }
79     }
80     delete it;
81     return result;
82 }
83 }
84
85 }
```

---

## BagNode.cpp

---

```
1  /*****
2   BagNode.cpp
3   Roman Lumetsberger
4
5   Implementation for class BagNode
6   *****/
7  #include "BagNode.h"
8  #include <MLObject.h>
9  #include "Node.h"
10 #include <sstream>
11
12 using namespace std;
13
14 namespace ML {
```

```
15
16 BagNode::BagNode(Object *value,
17     BagNode *prev,
18     BagNode *next) : Node(value, prev, next), count(1) {
19     Register("BagNode", "Node");
20 }
21
22 BagNode::~~BagNode() { /* nothing todo */ }
23
24 std::string BagNode::AsString() const {
25     if( value == nullptr)
26         return "<nullptr>";
27     stringstream ss;
28     ss << value->AsString() << "(" << count << ")";
29     return ss.str();
30 }
31
32
33 }
```

---

## Bag.cpp

---

```
1  /*****
2   Bag.cpp
3   Roman Lumetsberger
4
5   Implementation for class Bag, BagIterator
6   *****/
7  #include "Set.h"
8  #include "Bag.h"
9  #include <cassert>
10 #include <iostream>
11
12 using namespace std;
13
14 namespace ML {
15
16 Bag::Bag() :Set() {
17     Register("Bag", "Set");
18 }
19
20 Bag::~~Bag(){
21     Clear();
22 }
23
24 BagNode *Bag::Find(Object* o) const {
25     Node *parentNode = Set::Find(o);
26     if(parentNode == nullptr ) return nullptr;
27
28     BagNode *bagNode = dynamic_cast<BagNode *>(parentNode);
29     assert(bagNode != nullptr);
30     return bagNode;
31 }
32
33 BagNode *Bag::CreateNode(Object *o) const {
34     return new BagNode(o);
35 }
36 }
```

```
37 void Bag::Add(Object* o) {
38     BagNode *node = Find(o);
39     if(node != nullptr) {
40         node->count++;
41         size++;
42     }
43     else {
44         Set::Add(o);
45     }
46 }
47
48 Object *Bag::Remove(Object* o) {
49     BagNode *node = Find(o);
50     if(node != nullptr) {
51         if(node->count > 1) {
52             node->count--;
53             size--;
54         }
55         else {
56             Set::Remove(o);
57         }
58
59         return node->value;
60     }
61
62     return nullptr;
63 }
64
65 void Bag::DeleteElements() {
66     Node * current = head;
67     while(current != nullptr) {
68         Node *tmp = current;
69         current = current->next;
70         delete tmp->value;
71         delete tmp;
72     }
73     head = nullptr;
74     size = 0;
75 }
76
77 Iterator *Bag::NewIterator() const {
78     BagNode *bagHead = nullptr;
79     if(head != nullptr) {
80         bagHead = dynamic_cast<BagNode*>(head);
81         assert(bagHead != nullptr);
82     }
83     return new BagIterator(bagHead);
84 }
85
86 Bag *Bag::Union(Bag* other) const {
87     assert(other != nullptr);
88     Bag *result = new Bag();
89
90     //add elements from first Bag
91     Iterator *it = NewIterator();
92     for(Object *o = it->Next(); o != nullptr; o = it->Next()) {
93         result->Add(o);
94     }
```

```
95     delete it;
96
97     //add elements from second Bag;
98     it = other->NewIterator();
99     for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
100         result->Add(o);
101     }
102     delete it;
103     return result;
104 }
105
106 Bag *Bag::Intersect(Bag* other) const {
107     assert(other != nullptr);
108     Bag *result = new Bag();
109
110     Iterator *it = NewIterator();
111     for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
112         if(other->Contains(o)) { // other bag contains o
113             //Object not already in result bag
114             if(!result->Contains(o)) {
115                 BagNode *thisBagNode = Find(o);
116                 BagNode *otherBagNode = other->Find(o);
117
118                 int count = min(thisBagNode->count, otherBagNode->count);
119                 for(int i = 0; i < count; i++){
120                     result->Add(o);
121                 }
122             }
123         }
124     }
125     delete it;
126     return result;
127 }
128
129 Bag *Bag::Difference(Bag* other) const {
130     assert(other != nullptr);
131     Bag *result = new Bag();
132
133     Iterator *it = NewIterator();
134     for(Object *o = it->Next(); o != nullptr;o = it->Next()) {
135         //Object in not already in result bag
136         if(!result->Contains(o)) {
137             BagNode *thisBagNode = Find(o);
138             BagNode *otherBagNode = other->Find(o);
139
140             int count = thisBagNode->count;
141             if(otherBagNode != nullptr)
142                 count -= otherBagNode->count;
143
144             for(int i = 0; i < count; i++){
145                 result->Add(o);
146             }
147         }
148     }
149     delete it;
150     return result;
151 }
152 }
```

```
153
154
155
156
157 BagIterator::BagIterator(BagNode *head) : current(head), currentNodeCount(0) {
158     Register("BagIterator", "Iterator");
159     if(head != nullptr) {
160         currentNodeCount = head->count;
161     }
162 }
163
164 BagIterator::~BagIterator() { /*nothing todo */}
165
166 Object *BagIterator::Next() {
167     if(current == nullptr) return nullptr;
168     Object *o=current->value;
169     currentNodeCount--;
170     if(currentNodeCount == 0) {
171         if(current->next != nullptr) {
172             current = dynamic_cast<BagNode*>(current->next);
173             assert(current != nullptr);
174             currentNodeCount = current->count;
175         }
176         else {
177             current = nullptr;
178         }
179     }
180     return o;
181 }
182
183 }
```

---

### main.cpp

---

```
1 #include <iostream>
2 #include "Set.h"
3 #include "Bag.h"
4 #include <MLString.h>
5 #include <cassert>
6
7 using namespace std;
8 using namespace ML;
9
10 int main(int argc ,char** argv)
11 {
12     if(argc != 2) {
13         cerr << "Wrong parameter count" << endl;
14         cerr << "Usage: " << argv[0] << " testcase";
15         return 0;
16     }
17
18     Set *testSet = new Set();
19     Bag *testBag = new Bag();
20     String *testEntry = new String("entry1");
21     String *testEntry2 = new String("entry2");
22     String *testEntry3 = new String("entry3");
23
24     int testcase = atoi(argv[1]);
```

```
25  switch(testcase) {
26      case 1:
27          {
28              cout << "Testcase Set - Operations:" << endl;
29              cout << "-----" << endl;
30              testSet->Add(new String("entry1"));
31              testSet->Add(new String("entry2"));
32              testSet->Add(testEntry);
33
34              cout << endl;
35              cout << "Add Method: " << *testSet << endl;
36
37              Object *removedEntry = testSet->Remove(testEntry);
38              assert(removedEntry->IsEqualTo(testEntry));
39              cout << "Removed 'entry1': " << *testSet << endl << endl;
40              delete removedEntry;
41
42              cout << "Contains 'entry1': " << boolalpha << testSet->Contains(testEntry) << endl;
43              testSet->Add(new String("entry1"));
44              cout << "Contains 'entry1' after adding again': " << boolalpha << testSet->Contains(testEntry) << endl;
45          }
46      break;
47
48
49      case 2:
50          {
51              cout << "Testcase Set - Clear:" << endl;
52              cout << "-----" << endl;
53
54              testSet->Add(testEntry);
55              testSet->Add(testEntry2);
56              testSet->Add(testEntry3);
57              testSet->Clear();
58              cout << "Nodes deleted but Strings not" << endl;
59              WriteMetaInfo();
60              cout << endl;
61              break;
62          }
63      case 3:
64          {
65              cout << "Testcase Set - DeleteElements:" << endl;
66              cout << "-----" << endl;
67
68              testSet->Add(testEntry);
69              testSet->Add(testEntry2);
70              testSet->Add(testEntry3);
71              testSet->DeleteElements();
72              cout << "Nodes and Strings deleted" << endl;
73              WriteMetaInfo();
74              testEntry = nullptr;
75              testEntry2 = nullptr;
76              testEntry3 = nullptr;
77              cout << endl;
78              break;
79          }
80      case 4:
81          {
82              cout << "Testcase Set - Union:" << endl;
```



```
83     cout << "-----" << endl;
84
85     testSet->Add(testEntry);
86     testSet->Add(testEntry2);
87
88     Set *secondSet = new Set();
89     secondSet->Add(testEntry3);
90
91     Set *result = testSet->Union(secondSet);
92     cout << "Set1: " << *testSet << endl;
93     cout << "Set2: " << *secondSet << endl;
94     cout << "Union Result: " << *result << endl;
95     delete secondSet;
96     delete result;
97     testSet->Clear();
98
99     break;
100 }
101 case 5:
102 {
103     cout << "Testcase Set - Difference:" << endl;
104     cout << "-----" << endl;
105
106     testSet->Add(testEntry);
107     testSet->Add(testEntry2);
108     testSet->Add(testEntry3);
109
110     Set *secondSet = new Set();
111     secondSet->Add(testEntry3);
112
113     Set *result = testSet->Difference(secondSet);
114
115     cout << "Set1: " << *testSet << endl;
116     cout << "Set2: " << *secondSet << endl;
117     cout << "Difference Result: " << *result << endl;
118
119     delete secondSet;
120     delete result;
121     testSet->DeleteElements();
122     testEntry = nullptr;
123     testEntry2 = nullptr;
124     testEntry3 = nullptr;
125
126     break;
127 }
128 case 6:
129 {
130     cout << "Testcase Set - Intersect:" << endl;
131     cout << "-----" << endl;
132
133     testSet->Add(testEntry);
134     testSet->Add(testEntry2);
135     testSet->Add(testEntry3);
136
137     Set *secondSet = new Set();
138     secondSet->Add(testEntry3);
139
140     Set *result = testSet->Intersect(secondSet);
```

```
141
142     cout << "Set1: " << *testSet << endl;
143     cout << "Set2: " << *secondSet << endl;
144     cout << "Intersect Result: " << *result << endl;
145
146     delete secondSet;
147     delete result;
148     testSet->DeleteElements();
149     testEntry = nullptr;
150     testEntry2 = nullptr;
151     testEntry3 = nullptr;
152     break;
153 }
154
155 case 7:
156 {
157     cout << "Testcase Bag - Operations:" << endl;
158     cout << "-----" << endl;
159     testBag->Add(new String("entry1"));
160     testBag->Add(new String("entry2"));
161     testBag->Add(testEntry);
162
163     cout << endl;
164     cout << "Add Method: " << *testBag << endl;
165
166     Object *removedEntry = testBag->Remove(testEntry);
167     assert(removedEntry->IsEqualTo(testEntry));
168     cout << "Removed 'entry1': " << *testBag << endl << endl;
169     removedEntry = testBag->Remove(testEntry);
170     cout << "Removed 'entry1': " << *testBag << endl << endl;
171
172     delete removedEntry;
173     cout << "Contains 'entry1': " << boolalpha << testBag->Contains(testEntry) << endl;
174     testBag->Add(new String("entry1"));
175     cout << "Contains 'entry1' after adding again': " << boolalpha << testBag->Contains(testEntry) << endl;
176     break;
177 }
178 case 8:
179 {
180     cout << "Testcase Bag - Clear:" << endl;
181     cout << "-----" << endl;
182
183     testBag->Add(testEntry);
184     testBag->Add(testEntry2);
185     testBag->Add(testEntry3);
186     testBag->Clear();
187     cout << "Nodes deleted but Strings not" << endl;
188     WriteMetaInfo();
189     cout << endl;
190     break;
191 }
192 case 9:
193 {
194     cout << "Testcase Bag - DeleteElements:" << endl;
195     cout << "-----" << endl;
196
197     testBag->Add(testEntry);
198     testBag->Add(testEntry2);
```

```
199     testBag->Add(testEntry3);
200     testBag->Add(testEntry3);
201     testBag->DeleteElements();
202     cout << "Nodes and Strings deleted" << endl;
203     WriteMetaInfo();
204     testEntry = nullptr;
205     testEntry2 = nullptr;
206     testEntry3 = nullptr;
207     cout << endl;
208     break;
209 }
210 case 10:
211 {
212     cout << "Testcase Bag - Union:" << endl;
213     cout << "-----" << endl;
214
215     testBag->Add(testEntry);
216     testBag->Add(testEntry2);
217
218     Bag *secondBag = new Bag();
219     secondBag->Add(testEntry);
220     secondBag->Add(testEntry3);
221     secondBag->Add(testEntry3);
222
223     Bag *result = testBag->Union(secondBag);
224     cout << "Bag1: " << *testBag << endl;
225     cout << "Bag2: " << *secondBag << endl;
226     cout << "Union Result: " << *result << endl;
227     delete secondBag;
228     delete result;
229     testBag->Clear();
230
231     break;
232 }
233 case 11:
234 {
235     cout << "Testcase Bag - Difference:" << endl;
236     cout << "-----" << endl;
237
238     testBag->Add(testEntry);
239     testBag->Add(testEntry2);
240     testBag->Add(testEntry2);
241     testBag->Add(testEntry3);
242     testBag->Add(testEntry3);
243     testBag->Add(testEntry3);
244
245     Bag *secondBag = new Bag();
246     secondBag->Add(testEntry3);
247     secondBag->Add(testEntry2);
248     secondBag->Add(testEntry);
249     secondBag->Add(testEntry);
250
251     Bag *result = testBag->Difference(secondBag);
252
253     cout << "Bag1: " << *testBag << endl;
254     cout << "Bag2: " << *secondBag << endl;
255     cout << "Difference Result: " << *result << endl;
256
```

```
257     delete secondBag;
258     delete result;
259     testBag->DeleteElements();
260     testEntry = nullptr;
261     testEntry2 = nullptr;
262     testEntry3 = nullptr;
263
264     break;
265 }
266 case 12:
267 {
268     cout << "Testcase Bag - Intersect:" << endl;
269     cout << "-----" << endl;
270
271     testBag->Add(testEntry);
272     testBag->Add(testEntry);
273     testBag->Add(testEntry2);
274     testBag->Add(testEntry3);
275     testBag->Add(testEntry3);
276     testBag->Add(testEntry3);
277     testBag->Add(testEntry3);
278
279     Bag *second = new Bag();
280
281     second->Add(testEntry2);
282     second->Add(testEntry2);
283     second->Add(testEntry3);
284     second->Add(testEntry3);
285
286
287     Bag *result = testBag->Intersect(second);
288
289     cout << "Bag1: " << *testBag << endl;
290     cout << "Bag2: " << *second << endl;
291     cout << "Intersect Result: " << *result << endl;
292
293     delete second;
294     delete result;
295     testBag->DeleteElements();
296     testEntry = nullptr;
297     testEntry2 = nullptr;
298     testEntry3 = nullptr;
299     break;
300 }
301
302 }
303 testSet->DeleteElements();
304 testBag->DeleteElements();
305 delete testSet;
306 delete testBag;
307 delete testEntry;
308 delete testEntry2;
309 delete testEntry3;
310 WriteMetaInfo();
311
312 }
```

---

## 2.5 Testfälle

### 2.5.1 Testfall 1 - Set Operationen

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 1
Testcase Set - Operations:
-----
entry1 already in this set!

Add Method: Set with 2 Elements: { entry1, entry2 }
Removed 'entry1': Set with 1 Elements: { entry2 }

Contains 'entry1': false
Contains 'entry1' after adding again': true

```

```

=====
Meta information for MiniLib application
-----
Class hierarchy      | Number of dynamic objects
                    +-----+-----+-----+
                    | created | deleted | still alive
-----+-----+-----+-----+
Object              |      0 |      0 |      0
Collection          |      0 |      0 |      0
List                |      0 |      0 |      0
Set                 |      1 |      1 |      0
Bag                 |      1 |      1 |      0
String              |      6 |      6 |      0
Node                |      3 |      3 |      0
Iterator            |      0 |      0 |      0
ListIterator        |      3 |      3 |      0
-----+-----+-----+-----+
Number of classes:  9 | Summary: all objects deleted
=====

```

```
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █
```

### 2.5.2 Testfall 2 - Set Clear

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 2
Testcase Set - Clear:
-----
Nodes deleted but Strings not

```

```

=====
Meta information for MiniLib application
-----
Class hierarchy      | Number of dynamic objects
                    +-----+-----+-----+
                    | created | deleted | still alive
-----+-----+-----+-----+
Object              |      0 |      0 |      0
Collection          |      0 |      0 |      0
List                |      0 |      0 |      0
Set                 |      1 |      0 |      1
Bag                 |      1 |      0 |      1
String              |      3 |      0 |      3
Node                |      3 |      3 |      0
-----+-----+-----+-----+
Number of classes:  7 | Summary: 5 object(s) still alive
=====

```

```

-----+-----+-----+-----+
Number of classes:  9 | Summary: all objects deleted
=====

```

### 2.5.3 Testfall 3 - Set DeleteElements

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
clearclear: command not found
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ clear

romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 3
Testcase Set - DeleteElements:
-----
Nodes and Strings deleted

```

```

=====
Meta information for MiniLib application
-----
Class hierarchy      | Number of dynamic objects
                     |-----|
                     | created | deleted | still alive
-----|-----|-----|
Object              |      0 |      0 |      0
Collection           |      0 |      0 |      0
List                |      0 |      0 |      0
Set                 |      1 |      0 |      1
Bag                 |      1 |      0 |      1
String              |      3 |      3 |      0
Node                |      3 |      3 |      0
Iterator            |      0 |      0 |      0
ListIterator        |      1 |      1 |      0
-----|-----|-----|
Number of classes:  9 | Summary: 2 object(s) still alive
=====
-----|-----|-----|
Number of classes:  9 | Summary: all objects deleted
-----|-----|-----|
=====

```

### 2.5.4 Testfall 4 - Set Union

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 4
Testcase Set - Union:
-----
Set1: Set with 2 Elements: { entry1, entry2 }
Set2: Set with 1 Elements: { entry3 }
Union Result: Set with 3 Elements: { entry1, entry2, entry3 }

```

```

=====
Meta information for MiniLib application
-----
Class hierarchy      | Number of dynamic objects
                     |-----|
                     | created | deleted | still alive
-----|-----|-----|
Object              |      0 |      0 |      0
Collection           |      0 |      0 |      0
List                |      0 |      0 |      0
Set                 |      3 |      3 |      0
Bag                 |      1 |      1 |      0
String              |      3 |      3 |      0
Node                |      6 |      6 |      0
Iterator            |      0 |      0 |      0
ListIterator        |      6 |      6 |      0
-----|-----|-----|
Number of classes:  9 | Summary: all objects deleted
=====

```

```

romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █

```

## 2.5.5 Testfall 5 - Set Difference

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 5
Testcase Set - Difference:
-----
Set1: Set with 3 Elements: { entry1, entry2, entry3 }
Set2: Set with 1 Elements: { entry3 }
Difference Result: Set with 2 Elements: { entry1, entry2 }

```

```

=====
Meta information for MiniLib application
-----

```

Class hierarchy	Number of dynamic objects		
	created	deleted	still alive
Object	0	0	0
Collection	0	0	0
List	0	0	0
Set	3	3	0
Bag	1	1	0
String	3	3	0
Node	6	6	0
Iterator	0	0	0
ListIterator	6	6	0

```

-----
Number of classes: 9 | Summary: all objects deleted
=====

```

```
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █
```

## 2.5.6 Testfall 6 - Set Intersect

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 6
Testcase Set - Intersect:
-----
Set1: Set with 3 Elements: { entry1, entry2, entry3 }
Set2: Set with 1 Elements: { entry3 }
Intersect Result: Set with 1 Elements: { entry3 }

```

```

=====
Meta information for MiniLib application
-----

```

Class hierarchy	Number of dynamic objects		
	created	deleted	still alive
Object	0	0	0
Collection	0	0	0
List	0	0	0
Set	3	3	0
Bag	1	1	0
String	3	3	0
Node	5	5	0
Iterator	0	0	0
ListIterator	6	6	0

```

-----
Number of classes: 9 | Summary: all objects deleted
=====

```

```
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █
```

## 2.5.7 Testfall 7 - Bag Operationen

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 7
Testcase Bag - Operations:
-----

Add Method: Bag with 3 Elements: { entry1, entry1, entry2 }
Removed 'entry1': Bag with 2 Elements: { entry1, entry2 }

Removed 'entry1': Bag with 1 Elements: { entry2 }

Contains 'entry1': false
Contains 'entry1' after adding again': true

```

```

=====
Meta information for Minilib application
-----
Class hierarchy      | Number of dynamic objects
+-----+-----+-----+
| created | deleted | still alive
+-----+-----+-----+
Object              |      0 |      0 |      0
Collection           |      0 |      0 |      0
List                 |      0 |      0 |      0
Set                  |      1 |      1 |      0
Bag                  |      1 |      1 |      0
String               |      6 |      6 |      0
Node                 |      0 |      0 |      0
BagNode              |      3 |      3 |      0
Iterator             |      0 |      0 |      0
BagIterator          |      3 |      3 |      0
ListIterator         |      1 |      1 |      0
+-----+-----+-----+
Number of classes: 11 | Summary: all objects deleted
=====

```

## 2.5.8 Testfall 8 - Bag Clear

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 8
Testcase Bag - Clear:
-----

Nodes deleted but Strings not

```

```

=====
Meta information for Minilib application
-----
Class hierarchy      | Number of dynamic objects
+-----+-----+-----+
| created | deleted | still alive
+-----+-----+-----+
Object              |      0 |      0 |      0
Collection           |      0 |      0 |      0
List                 |      0 |      0 |      0
Set                  |      1 |      0 |      1
Bag                  |      1 |      0 |      1
String               |      3 |      0 |      3
Node                 |      0 |      0 |      0
BagNode              |      3 |      3 |      0
+-----+-----+-----+
Number of classes:  8 | Summary: 5 object(s) still alive
=====

-----+-----+-----+-----
Number of classes: 10 | Summary: all objects deleted
=====

```



## 2.5.9 Testfall 9 - Bag DeleteElements

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
-----
Number of classes: 10 | Summary: all objects deleted
=====

romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ clear

romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 9
Testcase Bag - DeleteElements:
-----
Nodes and Strings deleted

=====
Meta information for Minilib application
-----
Class hierarchy | Number of dynamic objects
-----
| created | deleted | still alive
-----
Object          | 0 | 0 | 0
Collection      | 0 | 0 | 0
List            | 0 | 0 | 0
Set             | 1 | 0 | 1
Bag             | 1 | 0 | 1
String          | 3 | 3 | 0
Node            | 0 | 0 | 0
BagNode         | 3 | 3 | 0
-----
Number of classes: 8 | Summary: 2 object(s) still alive
=====

-----
Number of classes: 10 | Summary: all objects deleted
=====

```

## 2.5.10 Testfall 10 - Bag Union

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 10
Testcase Bag - Union:
-----
Bag1: Bag with 2 Elements: { entry1, entry2 }
Bag2: Bag with 3 Elements: { entry1, entry3, entry3 }
Union Result: Bag with 5 Elements: { entry1, entry1, entry2, entry3, entry3 }

=====
Meta information for Minilib application
-----
Class hierarchy | Number of dynamic objects
-----
| created | deleted | still alive
-----
Object          | 0 | 0 | 0
Collection      | 0 | 0 | 0
List            | 0 | 0 | 0
Set             | 1 | 1 | 0
Bag             | 3 | 3 | 0
String          | 3 | 3 | 0
Node            | 0 | 0 | 0
BagNode         | 7 | 7 | 0
Iterator        | 0 | 0 | 0
BagIterator      | 5 | 5 | 0
ListIterator     | 1 | 1 | 0
-----
Number of classes: 11 | Summary: all objects deleted
=====

romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █

```

## 2.5.11 Testfall 11 - Bag Difference

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 11
Testcase Bag - Difference:
-----
Bag1: Bag with 6 Elements: { entry1, entry2, entry2, entry3, entry3, entry3 }
Bag2: Bag with 4 Elements: { entry3, entry2, entry1, entry1 }
Difference Result: Bag with 3 Elements: { entry2, entry3, entry3 }

```

```

=====
Meta information for MiniLib application
-----
Class hierarchy      | Number of dynamic objects
                     |-----|
                     | created | deleted | still alive
-----|-----|-----|-----
Object              |        0 |        0 |          0
Collection           |        0 |        0 |          0
List                 |        0 |        0 |          0
Set                  |        1 |        1 |          0
Bag                  |        3 |        3 |          0
String               |        3 |        3 |          0
Node                 |        0 |        0 |          0
BagNode              |        8 |        8 |          0
Iterator             |        0 |        0 |          0
BagIterator          |        4 |        4 |          0
ListIterator         |        1 |        1 |          0
-----|-----|-----|-----
Number of classes: 11 | Summary: all objects deleted
=====

```

```
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █
```

## 2.5.12 Testfall 12 - Bag Intersect

```

romanlum@ubuntu: ~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ ./minilibCollection 12
Testcase Bag - Intersect:
-----
Bag1: Bag with 7 Elements: { entry1, entry1, entry2, entry3, entry3, entry3, entry3 }
Bag2: Bag with 4 Elements: { entry2, entry2, entry3, entry3 }
Intersect Result: Bag with 3 Elements: { entry2, entry3, entry3 }

```

```

=====
Meta information for MiniLib application
-----
Class hierarchy      | Number of dynamic objects
                     |-----|
                     | created | deleted | still alive
-----|-----|-----|-----
Object              |        0 |        0 |          0
Collection           |        0 |        0 |          0
List                 |        0 |        0 |          0
Set                  |        1 |        1 |          0
Bag                  |        3 |        3 |          0
String               |        3 |        3 |          0
Node                 |        0 |        0 |          0
BagNode              |        7 |        7 |          0
Iterator             |        0 |        0 |          0
BagIterator          |        4 |        4 |          0
ListIterator         |        1 |        1 |          0
-----|-----|-----|-----
Number of classes: 11 | Summary: all objects deleted
=====

```

```
romanlum@ubuntu:~/swo3/UebungMoodle7/Beispiel/minilibCollection/bin/Debug$ █
```