Data Structures (Proposal)

This document proposes some data classes representing various types of collections as inspired by standard libraries available for languages such as C++ (STL library), Java and OCL. This proposal covers *Array2D*, *Bag*, *Set*, *Sequence*, *Queue*, *Stack* and *Map*, together with relevant iterators.

Two-Dimensional Array

# Array2D

This class extends *Object* and represents a two-dimensional Array.

***at(R, C: Integer) : Object***

Returns the object located at row R and column C in case R and C are within the index bounds of the *Array2D*. Otherwise, an error is generated.

***printString : String***

Returns a *String* representation of the content of the receiver.

***putAll(O: Object) : Matrix***

Replaces all elements in the receiver by O.

***putAt(R, C: Integer, O: Object) : Matrix***

Replaces the object at row R and column C with O in case R and C are within the index bounds of the *Array2D*. Otherwise, an error is generated.

***resize(R, C: Integer) : Matrix***

Changes the dimensions of the receiver where R is the number of rows and C is the number of columns. In case R or C is smaller than the current dimensions, elements are removed. In case R or C is larger than the current dimensions, elements valued **nil** are added. If R and C are larger than the size of the Array2D, an error is generated.

***rows : Integer***

Returns the number of rows of the receiver.

***columns : Integer***

Returns the number of columns of the receiver.

***size : Integer***

Returns the number of elements of the receiver (i.e., rows \* columns).

Collections

The following collection types are proposed: *Bag*, *Set* and *Sequence*. The next table lists some properties for the collection types (conform Java), where uniqueness of elements is guaranteed for all modifiers of *Sets* (as opposed to Java).

|  |  |  |
| --- | --- | --- |
| Collection Type | Unique | Ordered |
| Bag |  |  |
| Set | X |  |
| Sequence |  | X |

All proposed collection classes (*Bag*, *Set,* *Sequence*) inherit from class *Collection*. Each of the *Collections* is accommodated with the possibility of using *Iterators*. Variables denoting iterators can be declared as of type *Iterator*. *CollectionIterators* are a subclass of *Iterator* and are directly used for *Sets*. *Bags* and *Sequences* use subclasses *BagIterator* and *SequenceIterator* of *CollectionIterator*. A brief overview of the resulting *Iterator* functions is given below.

|  |  |  |  |
| --- | --- | --- | --- |
| Iterator Functionality | Advance | Remove | Reverse / First / Last |
| Bag | X | X |  |
| Set | X | X |  |
| Sequence | X | X | X |

More details of the *Iterators* are discussed in the appropriate subsections.

# Collection

This class extends *Object*. It introduces a super class for *Bag*, *Set* and *Sequence*.

***clear : Collection***

Clears the content of the receiver.

***isEmpty : Boolean***

Returns **true** in case the *Collection* does not contain any elements. Otherwise, it returns **false**.

***size : Integer***

Returns the number of elements in the *Collection*.

***excludes(O: Object) : Boolean***

Returns **true** in case element O is excluded in the receiver. Otherwise, it returns **false**.

***excludesAll(C: Collection) : Boolean***

Returns **true** in case all elements of C are excluded in the receiver. Otherwise, it returns **false**. In case C is not a *Collection*, an error is generated.

***includes(O: Object) : Boolean***

Returns **true** in case element O is included in the receiver. Otherwise, it returns **false**.

***includesAll(C: Collection) : Boolean***

Returns **true** in case all elements of C are included in the receiver. Otherwise, it returns **false**. In case C is not a *Collection*, an error is generated.

***toArray :Array***

Returns an *Array* that contains the elements in the *Collection*. In case the receiver is a *Sequence*, the ordering of the elements (and indices of the elements) is preserved.

***toBag : Bag***

Returns a *Bag* that contains the elements in the *Collection*.

***toSet : Set***

Returns a *Set* that contains the elements in the *Collection* (without duplicates).

***toSequence : Sequence***

Returns a *Sequence* that contains the elements in the *Collection*. In case the receiver is a *Sequence*, the ordering of the elements (and indices of the elements) is preserved.

# Bag

This class extends *Collection*. It provides an unordered *Collection* that may contain duplicates.

***clear : Bag***

Clears the content of the receiver.

***iterator : BagIterator***

Returns an iterator for the receiver.

***fromArray(A: Array) : Bag***

Modifies the content of the receiver by clearing its contents and subsequently adding all elements of *Array* A.

***=(B: Object) : Boolean***

Returns **true** in case B is a Bag and the receiver and B contain an equal number of the same objects. Otherwise, **false** is returned.

***!=(B: Object) : Boolean***

Returns!(**self** = B).

***isUnique : Boolean***

Returns **true** in case the *Bag* does not contain any duplicate elements. Otherwise, **false** is returned.

***count(O: Object) : Integer***

Returns the number of duplicates of O in the *Bag*.

***add(O: Object) : Bag***

Adds object O to the *Bag* and returns the receiver.

***add(O: Object, N: Integer) : Bag***  *// Currently named: addN*

Adds N duplicates of object O to the *Bag* and returns the receiver in case N is a non-negative integer. Otherwise, an error is generated.

***remove(O: Object) : Bag***

Removes one instance of O from the receiver (if such instance was included) and returns the receiver.

***remove(O: Object, N: Integer) : Bag*** *// Currently named: removeN*

Removes N instance of O from the receiver (or if such instance was included) or all instances of O in case N is larger than the number of instances in the *Bag*. It returns the receiver.

***removeDuplicates : Bag***

Removes all duplicate elements from the *Bag* and returns the receiver.

***+(B: Bag) : Bag***

Returns a new *Bag* that contains those elements that contained in both the receiver and B. The number of duplicate elements becomes the sum of the number of the duplicate elements in the receiver and B. In case B is not a *Bag*, an error is generated.

***union(B: Bag) : Bag***

Modifies the receiver to also contain those elements in B. The number of duplicate elements becomes the sum of the number of the duplicate elements in the receiver and B. In case B is not a *Bag*, an error is generated.

***-(B: Bag) : Bag***

Returns a new *Bag* that only contains those elements of the receiver that are not included in B. The number of duplicate elements becomes the number of the duplicate elements in the receiver minus the number of duplicate elements in B (if larger than 0). In case B is not a *Bag*, an error is generated.

***subtract(B: Bag) : Bag***

Modifies the receiver to contain only those elements that are not included in B. The number of duplicate elements becomes number of the duplicate elements in the receiver minus the number of duplicate elements in B (if larger than 0). In case B is not a *Bag*, an error is generated.

***difference(B: Bag) : Bag***

Modifies the receiver to become the symmetric difference of the receiver and B. This means that those elements of the receiver that are also in B are removed (such that a minimum of duplicates remains, if positive) and those elements in B that are not in the receiver are added (including duplicates). In case B is not a *Bag*, an error is generated.

***intersection(B: Bag) : Bag***

Modifies the receiver to contain only those elements that are also contained in the receiver and B. The number of duplicate elements becomes the minimum of the number of the duplicate elements included in the receiver and B. In case B is not a *Bag*, an error is generated.

***printString : String***

Returns a pretty print of the contents of the receiver.

***>(B: Bag) : Boolean***

Returns **true** in case the receiver is a strict super bag of B, where the number of duplicates in the receiver must be strictly larger than the number of duplicates in B. Otherwise, **false** is returned. In case B is not a *Bag*, an error is generated.

***>=(B: Bag) : Boolean***

Returns **true** in case the receiver is a super bag of B, where the number of duplicates in the receiver must be larger than or equal to the number of duplicates in B. Otherwise, **false** is returned. In case B is not a *Bag*, an error is generated.

***<(B: Bag) : Boolean***

Returns **true** in case the receiver is a strict sub bag of B, where the number of duplicates in the receiver must be strictly smaller than the number of duplicates in B. Otherwise, **false** is returned. In case B is not a *Bag*, an error is generated.

***<=(B: Bag) : Boolean***

Returns **true** in case the receiver is a sub bag of B, where the number of duplicates in the receiver must be smaller than or equal to the number of duplicates in B. Otherwise, **false** is returned. In case B is not a *Bag*, an error is generated.

# Bag Iterators

*Iterators* for *Bags* manifest through class *BagIterator* and support the following functions. Variables denoting iterators can be declared as of type *Iterator*.

***isDone : Boolean***

Returns **true** in case all elements in the *Bag* (including all duplicates) have been referred to. Otherwise, **false** is returned.

***element : Object***

Returns the element to which the iterator is pointing in the *Bag* in case the *Bag* is not empty. Otherwise, **nil** is returned.

***advance : BagIterator***

Modifies the receiver to refer to the next element in the *Bag* if not all elements have been referred to yet.

***remove : BagIterator***

Modifies the *Bag* by removing the element to which the receiver is pointing.

***printString : String***

Returns a pretty print of the contents of the receiver.

Notice that the advance and remove methods can be combined, as shown in this example:

|B: Bag, I: Iterator, P: String|

B := **new**(Bag) add(1) add(4) add(5) add(4) add(3);

P := ""; I := B iterator;

**while** !(I isDone) **do**

P := P concat(I element printString);

I remove advance // Empties B aka B clear

**od**;

// P = "14453"

# Set

This class extends *Collection*. It provides an unordered *Collection* that cannot contain duplicates.

***iterator : CollectionIterator***

Returns an iterator for the receiver.

***fromArray(A: Array) : Set***

Modifies the content of the receiver by clearing its contents and subsequently adding all elements of *Array* A.

***=(S: Object) : Boolean***

Returns **true** in case S is a Set and the receiver and S contain the same objects. Otherwise, **false** is returned.

***!=(S: Object) : Boolean***

Returns!(**self** = S).

***count(O: Object) : Integer***

Returns 1 in case O is included in the *Set*. Otherwise, 0 is returned.

***add(O: Object) : Set***

Adds O to the receiver (if it was not already included).

***remove(O: Object) : Set***

Removes O from the receiver (if it was included).

***+(S: Set) : Set***

Returns a new *Set* containing all elements of both the receiver and S (without duplicates). In case S is not a *Set*, an error is generated.

***union(S: Set) : Set***

Modifies the receiver by adding all elements of S to it (without duplicates). In case S is not a *Set*, an error is generated.

***-(S: Set) : Set***

Returns a new *Set* that contains only elements that contained in the receiver but not in S. In case S is not a *Set*, an error is generated.

***subtract(S: Set) : Set***

Modifies the receiver to no longer include the elements in S. In case S is not a *Set*, an error is generated.

***difference(S: Set) : Set***

Modifies the receiver to become the symmetric difference of the receiver and S. This means that those elements of the receiver that are also in S are removed and those elements in S that are not in the receiver are added. In case S is not a *Set*, an error is generated.

***intersection(S: Set) : Set***

Modifies the receiver to contain only those elements that are also contained in the receiver and S. In case s is not a *Set*, an error is generated.

***printString : String***

Returns a pretty print of the contents of the receiver.

***>(S: Set) : Boolean***

Returns **true** in case the receiver is a strict super set of S. Otherwise, **false** is returned. In case S is not a *Set*, an error is generated.

***>=(S: Set) : Boolean***

Returns **true** in case the receiver is a super set of S. Otherwise, **false** is returned. In case S is not a *Set*, an error is generated.

***<(S: Set) : Boolean***

Returns **true** in case the receiver is a strict sub set of S. Otherwise, **false** is returned. In case S is not a *Set*, an error is generated.

***<=(S: Set) : Boolean***

Returns **true** in case the receiver is a sub set of S. Otherwise, **false** is returned. In case S is not a *Set*, an error is generated.

# Set Iterators

*Iterators* for *Sets* manifest through class *CollectionIterator* and support the following functions. Variables denoting iterators can be declared as of type *Iterator*.

***isDone : Boolean***

Returns **true** in case all elements in the *Set* have been referred to. Otherwise, **false** is returned.

***element : Object***

Returns the element to which the iterator is pointing in the *Set* in case the *Set* is not empty. Otherwise, **nil** is returned.

***advance : CollectionIterator***

Modifies the receiver to refer to the next element in the *Set* if not all elements have been referred to yet.

***remove : CollectionIterator***

Modifies the *Set* by removing the element to which the receiver is pointing.

***printString : String***

Returns a pretty print of the contents of the receiver.

Notice that the advance and remove methods can be combined, as shown in this example:

|S: Set, I: Iterator|

S := **new**(Set) add(1) add(4) add(5) add(3);

P := ""; I := S iterator;

**while** !(I isDone) **do**

P := P concat(I element printString);

I remove advance // Empties S aka S clear

**od**;

// P = "1453"

# Sequence

This class extends *Collection*. It provides an ordered *Collection* that may contain duplicates.

***iterator : SeqeunceIterator***

Returns a (forward directed) iterator for the receiver.

***fromArray(A: Array) : Sequence***

Modifies the content of the receiver by clearing its contents and subsequently appending all elements of *Array* A (i.e., the ordering is preserved).

***isUnique : Boolean***

Returns **true** in case the *Sequence* does not contain any duplicate elements. Otherwise, **false** is returned.

***count(O: Object) : Integer***

Returns the number of duplicates of O in the *Seqeunce*.

***append(O: Object) : Sequence***

Adds O to the receiver at the last position.

***prepend(O: Object) : Sequence***

Adds O to the receiver at the first position.

***putAt(I : Integer, O: Object) : Sequence***

Modifies the receiver by replacing the element at index I with O if I is not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated.

***insertAt(I : Integer, O: Object) : Sequence***

Modifies the receiver by inserting element O at index I if I is not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated.

***at(I : Integer) : Object***

Returns the element at index I if I is not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated.

***first : Object***

Returns the first element in the *Sequence* (or **nil** if it is empty).

***last : Object***

Returns the first element in the *Sequence* (or **nil** if it is empty).

***removeAt(I : Integer) : Sequence***

Modifies the receiver by removing the element at index I if I is not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated.

***removeFirst : Sequence***

Returns the receiver after removing the first element in the *Sequence*.

***removeLast : Sequence***

Returns the receiver after removing the last element in the *Sequence*.

***+(S: Sequence) : Sequence***

Returns a new *Sequence* containing all elements in the receiver succeeded by all elements of S. In case S is not a *Seqeunce*, an error is generated.

***concat(S: Sequence) : Sequence***

Modifies the receiver by appending all elements of S. In case S is not a *Seqeunce*, an error is generated.

***reverse : Sequence***

Modifies the receiver by reversing the order of the elements.

***splice(I: Integer, S: Sequence) : Sequence***

Modifies the receiver by inserting all elements of S from index I onwards if I is not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated. Also in case S is not a *Seqeunce*, an error is generated. An error is also generated in case the receiver is empty.

***swap(I, J: Integer) : Sequence***

Modifies the receiver by swapping the elements at index I and J if both I and J are not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated. An error is also generated in case the receiver is empty or in case I != J when the receiver only contains one element.

***subSequence(I, L: Integer) : Sequence***

Returns a new *Sequence* that contains the elements of the receiver from and including index I until and including index I + L if I is not smaller than 1, L is not negative and I + L is not greater than the size of the *Sequence* + 1. Otherwise, an index out of bounds error is generated. An error is also generated in case the receiver is empty.

***find(I: Integer, O: Object) : Integer***

Returns the index at which object O is located, starting from index I in case I is not smaller than 1 and not greater than the size of the *Sequence.* When O is not found from index I onwards, nil is returned. In case I does not satisfy the indicated conditions, an index out of bounds error is generated. An error is also generated in case the receiver is empty.

***iteratorAt(I : Integer) : SequenceIterator***

Returns a (forward directed) *Iterator* to the element at index I of the *Sequence* if I is not smaller than 1 and not greater than the size of the *Sequence*. Otherwise, an index out of bounds error is generated. An error is also generated in case the receiver is empty.

***printString : String***

Returns a pretty print of the contents of the receiver.

# Sequence Iterators

*Iterators* for *Sequences* manifest through class *SequenceIterator* and support the following functions. Variables denoting iterators can be declared as of type *Iterator*.

***isDone : Boolean***

Returns **true** in case all elements in the *Sequence* have been referred to. Otherwise, **false** is returned.

***element : Object***

Returns the element to which the iterator is pointing in the *Sequence* in case the *Sequence* is not empty. Otherwise, **nil** is returned.

***first : SequenceIterator***

Modifies the receiver to point to the first element in the *Sequence*.

***last : SequenceIterator***

Modifies the receiver to point to the last element in the *Sequence*.

***reverse : SequenceIterator***

Modifies the receiver to advance in reverse direction.

***advance : SequenceIterator***

Modifies the receiver to refer to the next element (for forward directed iterators) or previous element (for reversely directed iterators) in the *Sequence* if not all elements have been referred to yet.

***remove : SequenceIterator***

Modifies the *Sequence* by removing the element to which the receiver is pointing.

***printString : String***

Returns a pretty print of the contents of the receiver.

Notice that the advance and remove methods can be combined, as shown in this example:

|S: Sequence, I: Iterator, P: String|

S := **new**(Sequence) append(1) append(4) append(5) append(4) append(3);

P := ""; I := S iterator last reverse; // Reverse from last element

**while** !(I isDone) **do**

P := P concat(I element printString);

I remove advance // Empties S aka S clear

**od**;

// P = "34541"

Queue & Stack

# Queue

This class extends *Object*. It provides an ordered *Collection* with FIFO ordering of elements. At initialization, a *Queue* has unbounded capacity.

***clear : Queue***

Clears the content of the receiver.

***resize(I: Integer) : Queue***

Modifies the capacity of the *Queue*. If I is smaller than the current capacity, elements at the tail of the *Queue* are removed. If I equals **nil** then the capacity is set to unbounded. If I is smaller than 1, an error is generated.

***size: Integer***

Returns the capacity of the *Queue* or **nil** if unbounded.

***occuption : Integer***

Returns the number of elements in the *Queue*.

***isEmpty : Boolean***

Returns **true** in case the *Queue* does not contain any elements. Otherwise, it returns **false**.

***isFull : Boolean***

Returns **true** in case the occupancy of the *Queue* equals its capacity. Otherwise, it returns **false** (also if the capacity is unbounded).

***excludes(O: Object) : Boolean***

Returns **true** incase there are no instances of O included in the *Queue*. Otherwise, it returns **false**.

***includes(O: Object) : Boolean***

Returns **true** incase at least one instance of O is included in the *Queue*. Otherwise, it returns **false**.

***count(O: Object) : Integer***

Returnsthenumber of instances of O in the *Queue*.

***add(O: Object) : Queue***

Adds object O to the end (tail) of the receiver in case it is not full.

***inspect : Object***

Returns the first element (head) in the receiver if it is not empty. Otherwise, it returns **nil**. It does not modify the receiver.

***remove : Object***

Returns the first element (head) in thereceiver and removes it in case the receiver is not empty. Otherwise, it returns **nil**.

***printString : String***

Returns a pretty print of the contents of the receiver.

# Stack

This class extends *Object*. It provides an ordered *Collection* with LIFO ordering of elements. At initialization, a *Stack* has unbounded capacity.

***clear : Stack***

Clears the content of the receiver.

***resize(I: Integer) : Stack***

Modifies the capacity of the *Stack*. If I is smaller than the current capacity, elements at the tail of the *Stack* are removed. If I equals **nil** then the capacity is set to unbounded. If I is smaller than 1, an error is generated.

***size: Integer***

Returns the capacity of the *Stack* or **nil** if unbounded.

***occuption : Integer***

Returns the number of elements in the *Stack*.

***isEmpty : Boolean***

Returns **true** in case the *Stack* does not contain any elements. Otherwise, it returns **false**.

***isFull : Boolean***

Returns **true** in case the occupancy of the *Stack* equals its capacity. Otherwise, it returns **false** (also if the capacity is unbounded).

***excludes(O: Object) : Boolean***

Returns **true** incase there are no instances of O included in the *Stack*. Otherwise, it returns **false**.

***includes(O: Object) : Boolean***

Returns **true** incase at least one instance of O is included in the *Stack*. Otherwise, it returns **false**.

***count(O: Object) : Integer***

Returnsthenumber of instances of O in the *Stack*.

***push(O: Object) : Queue***

Adds object O to the begin (head) of the receiver in case it is not full.

***inspect : Object***

Returns the first element (head) in the receiver if it is not empty. Otherwise, it returns **nil**. It does not modify the receiver.

***pop : Object***

Returns the first element (head) in thereceiver and removes it in case the receiver is not empty. Otherwise, it returns **nil**.

***printString : String***

Returns a pretty print of the contents of the receiver.

Map

A *Map* represents a key-value paired dictionary. Iterators for a *Map* manifest through class *MapIterator* and support advancing and removing, see the appropriate subsection for details.

# Map

This class extends *Object* and provides a key-value dictionary.

***clear : Map***

Clears the content of the receiver.

***isEmpty : Boolean***

Returns **true** in case the *Map* does not contain any key-value pairs. Otherwise, it returns **false**.

***size : Integer***

Returns the number of key-value pairs in the *Map*.

***iterator : MapIterator***

Returns an iterator for the receiver.

***includesKey(K: Object) : Boolean***

Returns **true** in case key K is included in the receiver. Otherwise, it returns **false**.

***includesValue(V: Object) : Boolean***

Returns **true** in case at least one instance of value V is included in the receiver. Otherwise, it returns **false**.

***keys : Set***

Returns a *Set* with all keys in the receiver.

***values : Bag***

Returns a *Bag* with all values corresponding to all keys (hence, it may contain duplicates).

***=(M: Object) : Boolean***

Returns **true** in case the receiver and M contain the same key-value pairs. Otherwise, it returns **false**.

***!=(M: Object) : Boolean***

Returns !(**self** = M).

***putAt(K, V: Object) : Map***

Modifies the receiver to store value V at key K. In case key K was not yet included, it is added.

***at(K: Object) : Object***

Returns the value at key K if such key is included. Otherwise, it returns **nil**.

***removeAt(K: Object) : Map***

Modifies the receiver by removing the key-value pair with key K (if it was included).

***printString : String***

Returns a pretty print of the contents of the receiver.

# Map Iterators

*Iterators* for *Maps* manifest through class *MapIterator* and support the following functions. Variables denoting iterators can be declared as of type *Iterator*.

***isDone : Boolean***

Returns **true** in case all key-value pairs in the *Map* have been referred to. Otherwise, **false** is returned.

***key : Object***

Returns the key of the key-value pair to which the iterator is pointing in the *Map* in case the *Map* is not empty. Otherwise, **nil** is returned.

***value : Object***

Returns the value of the key-value pair to which the iterator is pointing in the *Map* in case the *Map* is not empty. Otherwise, **nil** is returned.

***advance : MapIterator***

Modifies the receiver to refer to the next key-value pair in the *Map* if not all elements have been referred to yet.

***remove : MapIterator***

Modifies the *Map* by removing the key-value pair to which the receiver is pointing.

***printString : String***

Returns a pretty print of the contents of the receiver.

Notice that the advance and remove methods can be combined, as shown in this example:

|M: Map, I: Iterator, P: String|

M := **new**(Map) putAt(1,1) putAt(2,2) putAt(3,3) putAt(4,4);

P := ""; I := M iterator;

**while** !(I isDone) **do**

P := P concat(I key printString) concat(I value printString);

I remove advance // Empties M aka M clear

**od**;

// P = "11223344"