Laura Barreto
CS203

API - data2

Package: data2

Interface Summary:
FiniteBag<D extends java.lang.Comparable>
Randomness<D extends java.lang.Comparable>
Sequence<D>
Sequenced<D extends java.lang.Comparable>

Note: { } represent FiniteBag, [ ] represent Sequences

Interface FiniteBag<D extends java.lang.Comparable>

Type Parameters:
D - Generic object type

All Superinterfaces:
Sequenced<D>

All Known Implementing Classes:
SBBT_MT: Empty Bag
SBBT_ST: Bag with objects

public interface FiniteBag<D extends java.lang.Comparable>
extends Sequenced<D>

Method Summary

Methods	
Modifier and Type	Method and Description
FiniteBag<D>	add(D elt) Example: {a, c}.add(b) = {a, b, c}
FiniteBag<D>	add(D elt, int n) Example: {a, c}.add(b,2} = {a, b, b, c}
FiniteBag<D>	balance()
FiniteBag<D>	blacken()
int	cardinality() Example: {a, a, a, b, c}.cardinality() = 5
FiniteBag<D>	diff(FiniteBag u) Example: {a, b, c}.diff{a, b, c, d} = {d}
boolean	equal(FiniteBag u) Example: {a, b}.equal{a, b} = true
int	getCount(D elt) Example: {a, a, a, b, c}.getCount(a) = 3
FiniteBag<D>	inter(FiniteBag u) Example: {a, b, c}.inter{a, b, c, d} = {a, b, c}
boolean	isBlackHuh()
boolean	isEmptyHuh() Example: {}.isEmptyHuh() = true; Example: {a, b, c, d}.isEmptyHuh() = false
boolean	member(D elt) Example: {a, b, c}.member(a) = true; Example: {a, b, c}.member(d) = false
FiniteBag<D>	remove(D elt) Example: {a, b, c}.remove(a) = {b, c}
FiniteBag<D>	remove(D elt, int n) Example: {a, a, a, a, b, c}.remove(a, 2) = {a, a, b, c}
FiniteBag<D>	removeAll(D elt) Example: {a, a, a, a, b, c}.removeAll(a) = {b, c}
Sequence<D>	seq() Example: {a, b, c, d}.seq() = [a, b, c, d]
String	stringIt()
String	stringItS(Sequence<D> s)
boolean	subset(FiniteBag u) Example: {a, b, c}.subset{a, b, c, d} = true

Methods	
Modifier and Type	Method and Description
int	sumIt()
int	sumItS(Sequence<D> s)
FiniteBag<D>	union(FiniteBag u) Example: {a, b, c}.union{a, b, c, d} = {a, a, b, b, c, c, d}

Method Detail

getCount
int getCount(D elt)
Example: {a, a, a, b, c}.getCount(a) = 3;
Example: {a, a, a, b}.getCount(b) = 1;

Parameters:

elt - is a generic object type D

Returns:

Int that represents the amount of elts in the FiniteBag

cardinality
int cardinality()
Example: {a, a, a, b, c}.cardinality() = 5;
Example: {}.cardinality() = 0;
Example: {1500000000}.cardinality() = 1;

Returns:

Int that represents the size of the set or the total amount of elements

isEmptyHuh
boolean isEmptyHuh()
Example: {}.isEmptyHuh() = true;
Example: {a, b, c, d}.isEmptyHuh() = false;

Returns:

Boolean that represents whether the set

member
boolean member(D elt)
Example: {a, b, c}.member(a) = true;
Example: {a, b, c}.member(d) = false;

Parameters:

elt - is a generic object type D

Returns:

Boolean that represents whether the elt is in the FiniteBag

remove
FiniteBag<D> remove(D elt)
Example: {a, b, c}.remove(a) = {b, c}

Parameters:

elt - is a generic object type D

Returns:

FiniteBag with one less count of elt.

remove
FiniteBag<D> remove(D elt, int n)
Example: {a, a, a, a, b, c}.remove(a, 2) = {a, a, b, c}

Parameters:

elt - is a generic object type D
n - is an int that indicates the number of elements to be removed

Returns:

FiniteBag with n less counts of elt.

removeAll
FiniteBag<D> removeAll(D elt)
Example: {a, a, a, a, b, c}.removeAll(a) = {b, c}

Parameters:

elt - is a generic object type D

Returns:

FiniteBag with all instances of elt removed.

add
FiniteBag<D> add(D elt)
Example: {a, c}.add(b) = {a, b, c}

Parameters:

elt - is a generic object type D

Returns:

FiniteBag that is balanced with an inserted elt

add
FiniteBag<D> add(D elt,
int n)
Example: {a, c}.add(b,2} = {a, b, b, c}

Parameters:

elt - is a generic object type D
n - is an int that indicates the number of elements to be added

Returns:

FiniteBag that is balanced with n inserted elts

equal
boolean equal(FiniteBag u)
Example: {a}.equal{b} = false;
Example: {a, b}.equal{a, b} = true;

Parameters:

u - is a FiniteBag containing generic object types D

Returns:

Boolean that indicates whether FiniteBags are equal

subset
boolean subset(FiniteBag u)
Example: {a, b, c}.subset{a, b, c, d} = true;
Example: {a, b, c}.subset{a, b, c} = true;
Example: {a, b, c}.subset{a, b} = false;

Parameters:

u - is a FiniteBag containing generic object types D

Returns:

Boolean that indicates whether this is a subset of u

union
FiniteBag<D> union(FiniteBag u)
Example: {a, b, c}.union{a, b, c, d} = {a, a, b, b, c, c, d};
Example: {a, b, c}.union{d, e, f} = {a, b, c, d, e, f};
Example: {a, b, c}.union{} = {a, b, c};

Parameters:

u - is a FiniteBag containing generic object types D

Returns:

FiniteBag containing everything in this or u

inter
FiniteBag<D> inter(FiniteBag u)
Example: {a, b, c}.inter{a, b, c, d} = {a, b, c};
Example: {a, b, c}.inter{d, e, f} = {};
Example: {a, b, c}.inter{} = {};

Parameters:

u - is a FiniteBag containing generic object types D

Returns:

FiniteBag containing everything in this and u

diff
FiniteBag<D> diff(FiniteBag u)
Example: {a, b, c}.diff{a, b, c, d} = {d};
Example: {a, b, c}.diff{d, e, f} = {d, e, f};
Example: {a, b, c}.diff{} = {};

Parameters:

u - is a FiniteBag containing generic object types D

Returns:

FiniteBag containing everything in u that is not in this

seq

Sequence<D> seq()

Description copied from interface: Sequenced

Example: {a, b, c, d}.seq() = [a, b, c, d]

Specified by:

seq in interface Sequenced<D extends java.lang.Comparable>

Returns:

A Sequence that contains objects of generic type D

sumIt

int sumIt()

Returns:

Int that represents the number of elements in this sequence

sumItS

int sumItS(Sequence<D> s)

Parameters:

s - is a sequence that contains objects of generic type D

Returns:

Int that represents the number of elements in the s sequence

stringIt

java.lang.String stringIt()

Returns:

String containing all items in this sequence

stringItS

java.lang.String stringItS(Sequence<D> s)

Parameters:

s - is a sequence that contains objects of generic type D

Returns:

String containing all items in the s sequence

blacken

FiniteBag<D> blacken()

Returns:

Black parent node

isBlackHuh

boolean isBlackHuh()

Returns:

Boolean that returns true if root in FiniteBag is black

balance
FiniteBag<D> balance()
Returns:
Balanced FiniteBag

Interface Sequenced<D extends java.lang.Comparable>

Type Parameters:
D - Generic object type

All Superinterfaces:
FiniteBag<D>

All Known Implementing Classes:
SBBT_MT: Empty Bag
SBBT_ST: Bag with objects
SequenceST

public interface Sequenced<D extends java.lang.Comparable>

Method Summary

Methods	
Modifier and Type	Method and Description
Sequence<D>	seq() Example: {a, b, c, d}.seq() = [a, b, c, d]

Method Detail
seq
Sequence<D> seq()
Returns:
A Sequence containing objects of generic type D in this

Interface Sequence<D>

Type Parameters:
D - Generic object type

All Known Implementing Classes:
SequenceCat: Concatenated Sequences
SequenceMT: Empty Sequences
SequenceST: Sequences with objects

Method Summary

Methods	
Modifier and Type	Method and Description
boolean	hasNext() Example: [a, b, c, d].hasNext() = true
D	here() Example: [a, b, c, d].here() = a
Sequence<D>	next() Example: [a, b, c, d].next() = [b, c, d]
String	seqToString() Example: [a, b, c, d].seqToString() = “a b c d”

Method Detail

here
D here()
Example: [a, b, c, d].here() = a

Returns:

Generic object D at the current location in sequence being looked at

hasNext
boolean hasNext()
Example: [a, b, c, d].hasNext() = true;

Returns:

Boolean that represents if end of sequence has not been reached

next
Sequence<D> next()
Example: [a, b, c, d].next() = [b, c, d];

Returns:

Sequence containing generic objects D that follow the current object

seqToString
java.lang.String seqToString()
Example: [a, b, c, d].seqToString() = "a b c d"

Returns:

String of the Sequence

Interface Randomness<D extends java.lang.Comparable>

Type Parameters:
D - Generic object type

All Known Implementing Classes:
randInt: creates random integer
randString: creates random String

public interface Randomness<D extends java.lang.Comparable>

Method Summary

Methods	
Modifier and Type	Method and Description
D	createRand()

Method Detail

createRand
D createRand()
Returns:
Random object of type D which could either be a random int or random string

---

Properties and Performance Characteristics of Methods in Interfaces

Properties Between FiniteBag<D> Methods:
A FiniteBag represents a Polymorphic FiniteSet which is self-balancing and is able to contain multiple instances of the same element.

r —> FiniteBag
s —> FiniteBag
t —> FiniteBag
e —> Random Element
n —> int
mt —> Empty FiniteBag

The following properties of multisets hold:

1. (isEmptyHuh t) = true <—> t = empty()
2. (union s mt) = s <—> mt = empty()

3. (inter s mt) = mt <—> mt = empty()
4. (diff s mt) = mt <—> mt = empty()
5. (diff mt s) = s <—> mt = empty()
6. (cardinality t) = 0 <—> t = empty()
7. (cardinality t) < (cardinality (add t e))
8. (cardinality t) <= (cardinality(remove t e))
9. (member e (union r s)) <—> (member e r) v (member e s)
10. (member e (intersection r s)) <—> (member e r) ^ (member e s)
11. (getCount e (union r s)) = (getCount e r) + (getCount e s)
12. (subset r (union r s)) = true
13. (equal r s) = true <—> (subset r (inter r s)) ^ (subset s (inter r s))
14. (sumIt t) == (cardinality t)
15. !(isEmptyHuh (diff r s)) —> (member e s) == true ^ (member e r) == false
16. if (add e t) —> then (member e t) == true

Performance Characteristics of FiniteBag<D>:
Because the FiniteBags are balanced, the speed of searching through the Bag would be constant at log(n). The speed of inserting elements in a Bag would be constant at log(n). The speed of removing elements in a Bag would be constant at log(n). All other methods run at a linear speed

———

Properties of Sequenced method seq():
Gives an instance of a sequence in order to show that any Sequenced object can have its elements organized into a sequence in order to apply appropriate methods to any element in the Sequenced object.

Performance Characteristics of Sequenced:
Because each node and Bag become a part of a Sequence, the seq() runs linear time.

———

Properties Between Sequence Methods:
Here, hasNext, and next are used as abstract iterators to iterate through the FiniteBag multisets and apply any method to the elements in the set. If SeqToString method is called on the empty set, the SeqToString should not print out anything.

Performance Characteristics of Sequence:
Iteration of the set looks at one element at a time thus the sequence methods run on linear time.

Properties Between Randomness Method createRand():
CreateRand() either creates a random integer element or a random string element. The purpose of this is to be able to test random elements.

Performance Characteristics of Sequence:
CreateRand() works on constant time because whether user is generating a randInt or a randString, the createRand method works efficiently.