KEY: Orange Highlight = Test added during this milestone

Red Highlight	= Impossible Test Case
---------------	------------------------

Test	Equivalence class partitioning(Wect, Sect, Wrect, Srect)
PredicatedListTest .testDecorated	 Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list This is already done Wrect and srect are covered by testing of a null variable list This is already done
PredicatedListTest .testEquals	 Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list
PredicatedListTest .testHashCode	 Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list
PredicatedListTest .testGet	 Variables: List length L Index I Equivalence classes L1: L = 0 L2: L > 0 I1: I < 0 I2: I >= 0 && I < L I3: I >= L Wect
	ID L I

	WECT1	L1	l1
	WECT2	L2	12
	WECT3	L1	13
• Sect			
	ID	L	I
	SECT1	L1	l1
	SECT2	L2	I1
	SECT3	L1	12
	SECT4	L2	12
	SECT5	L1	13
	SECT6	L2	13
SECT2 • Wrect a	, SECT5, and SEC and srect are also co	Τ6	
• Equival	Object O lence classes L1: L = 0 L2: L > 0 O1: O is in list		
	ID	L	0
	WECT1	L1	01
	WECT2	L2	O2
• Sect			
	ID	L	I
1			
	Wrect a SECT2 Wrect a SECT2 Wrect a SEQuival Wect Sect	WECT2 WECT3 Sect ID SECT1 SECT2 SECT3 SECT4 SECT5 SECT6 Wrect and Srect would inve Wrect and srect are covere SECT2, SECT5, and SEC Wrect and srect are also or Variables:	WECT2 L2 WECT3 L1 • Sect ID L SECT1 L1 SECT2 L2 SECT3 L1 SECT4 L2 SECT5 L1 SECT6 L2 • Wrect and Srect would involve L < 0 but this is every management of the second se

		SECT2	L2	01
		SECT3	L1	O2
		SECT4	L2	O2
	 Wrect a 	and Srect would invented and srect are covered and srect are also countries.	ed by WECT1, SEC	T1
PredicatedListTest .tesetLastIndexOf	• Equiva •	es: List length L Object O lence classes L1: L = 0 L2: L > 0 O1: O is only instar O2: multiple instanc O3: O is not in list		
	Ü	ID	L	0
		WECT1	L1	O1
		WECT2	L2	O2
		WECT3	L1	О3
	• Sect			
		ID	L	I
		SECT1	L1	01
		SECT2	L2	01
		SECT3	L1	O2
		SECT4	L2	O2
		SECT5	L1	O3
		SECT6	L2	О3
	 Wrect a SECT3 	and Srect would invented and srect are covered and srect are also cand srect are also cand	ed by WECT1, WE	CT3, SECT1,

PredicatedListTest .testRemove	• Equiva • • • • • • • • • • • • • • • • • • •	es: List length L Index I lence classes L1: L = 0 L2: L > 0 I1: I < 0 I2: I >= 0 && I < L I3: I >= L		
		ID	L	I
		WECT1	L1	I1
		WECT2	L2	12
		WECT3	L1	13
	● Sect ○			
	_	ID	L	I
		SECT1	L1	l1
		SECT2	L2	l1
		SECT3	L1	12
		SECT4	L2	12
		SECT5	L1	13
		SECT6	L2	13
	 Wrect a SECT2 	and Srect would invented and srect are covered, SECT5, and SEC and srect are also county	ed by WECT1, WE0 T6	CT3, SECT1,
PredicatedListTest .testAdd	• Equiva • •	es: List length L Object O lence classes L1: L = 0 L2: L > 0 O1: O can be inser O2: O cannot be in		

			•	
		ID	L	0
		WECT1	L1	O1
		WECT2	L2	O2
	• Sect			
		ID	L	1
		SECT1	L1	O1
		SECT2	L2	O1
		SECT3	L1	O2
		SECT4	L2	O2
			olve L < 0 but this is ed by testing of a nu	
.testAddAll	 List length L List of objects O Equivalence classes L1: L = 0 L2: L > 0 O1: O can be fully inserted into list O2: O can be partially inserted into list O3: O cannot be inserted at all into list Wect 			
		ID	L	0
		WECT1	L1	O1
		WECT2	L2	O2
		WECT3	L1	О3
	• Sect			
		ID	L	1
		SECT1	L1	O1
		SECT2	L2	01
		SECT3	L1	O2

		SECT4	L2		O2
		SECT5	L1		O3
		SECT6	L2		O3
		and Srect would and srect are al			not possible a null variable list
PredicatedListTest .testListIterator	EquivalOThus, wa noneOWrect a	List length L lence classes L1: L = 0 L2: L > 0 vect and sect a	done d involve L < 0) but this is	
PredicatedListTest .testSet	• Equival	es: List length L Index I Object O lence classes L1: L = 0 L2: L > 0 I1: I < 0 I2: I >= 0 && I= I3: I >= L O1: O can be i O2: O cannot be	nserted		
	_	ID	L	I	0
		WECT1	L1	I1	01
		WECT2	L2	12	O2
		WECT3	L1	13	01
	• Sect				
		ID	L	I	0
		SECT1	L1	I1	01
		SECT2	L2	l1	O1

SECT3						
SECT5			SECT3	L1	12	01
SECT6 L2 I3 O1			SECT4	L2	12	01
SECT7			SECT5	L1	13	01
SECT8 L2 I1 O2 SECTIO L1 I2 O2 SECTIO L2 I2 O2 SECTIO L2 I3 O2 SECTIO L2 I3 O2 SECTIO L2 I3 O2 • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 • Wrect and srect are also covered by testing of a null variable list • Variables: • List length L • Equivalence classes • L1: L = 0 • L2: L > 0 • Thus, wect and sect are both covered by using an empty list and a nonempty • This is already done • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by testing of a null variable list • Variables: • List length L • Object O • Equivalence classes • L1: L = 0 • L2: L > 0 • O1: O does not already exist in list • O2: O already exists in list • Wect			SECT6	L2	13	O1
SeCT10 L2 I2 O2 SECT11 L1 I3 O2 SECT12 L2 I3 O2 • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 • Wrect and srect are also covered by testing of a null variable list SetUniqueListTest Equivalence classes • L1: L = 0 • L2: L > 0 • Thus, wect and sect are both covered by using an empty list and a nonempty • This is already done • Wrect and Srect would involve L < 0 but this is not possible • Wrect and Srect would involve L < 0 but this is not possible • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by testing of a null variable list SetUniqueListTest List length L • Object O • Equivalence classes • L1: L = 0 • L2: L > 0 • O1: O does not already exist in list • O2: O already exists in list • Wect			SECT7	L1	I1	O2
SECT10 L2 I2 O2 SECT11 L1 I3 O2 SECT12 L2 I3 O2 • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 • Wrect and srect are also covered by testing of a null variable list SetUniqueListTest .testAsSet • Variables: • List length L • Equivalence classes • L1: L = 0 • L2: L > 0 • Thus, wect and sect are both covered by using an empty list and a nonempty • This is already done • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by testing of a null variable list SetUniqueListTest .testAdd • Variables: • List length L • Object O • Equivalence classes • L1: L = 0 • L2: L > 0 • O1: O does not already exist in list • Wect			SECT8	L2	I1	O2
SECT11 L1 I3 O2 SECT12 L2 I3 O2 • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 • Wrect and srect are also covered by testing of a null variable list • Variables:			SECT9	L1	12	O2
SECT12 L2 I3 O2 • Wrect and Srect would involve L < 0 but this is not possible • Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 • Wrect and srect are also covered by testing of a null variable list SetUniqueListTest .testAsSet • Variables:			SECT10	L2	12	O2
Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 Wrect and srect are also covered by testing of a null variable list Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list SetUniqueListTest LestAdd Variables: List length L Object O Equivalence classes L1: L = 0 L2: L > 0 OL2: L > 0 OL			SECT11	L1	13	O2
Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT3, SECT5, SECT6, SECT7, SECT9, SECT10, SECT11, SECT12 Wrect and srect are also covered by testing of a null variable list Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list SetUniqueListTest LestAdd Variables: List length L Object O Equivalence classes L1: L = 0 L2: L > 0 C1: C does not already exist in list O2: O already exists in list Wect Wect			SECT12	L2	13	O2
 ∴testAdd ○ List length L ○ Object O ● Equivalence classes ○ L1: L = 0 ○ L2: L > 0 ○ O1: O does not already exist in list ○ O2: O already exists in list ● Wect 	.testAsSet	SECT2 SECT1 • Wrect a • Variable • Equival • Thus, v a none • Wrect a • Wrect a	es: List length L lence classes L1: L = 0 L2: L > 0 vect and sect a mpty This is already and srect are c	T5, SECT6, S Iso covered by are both covered done d involve L < 0	ed by using but this is	CT9, SECT10, a null variable list g an empty list and not possible
		• Equival	List length L Object O lence classes L1: L = 0 L2: L > 0 O1: O does no O2: O already	exists in list	t in list	0
			טו	L		U

		WECT1	L1	01
		WECT2	L2	O2
	• Sect			
		ID	L	I
		SECT1	L1	01
		SECT2	L2	O1
		SECT3	L1	O2
		SECT4	L2	O2
		and Srect would inv		
.testAddAll	 List length L List of objects O Equivalence classes L1: L = 0 L2: L > 0 O1: O can be fully inserted into list without duplicates O2: O can be partially inserted into list without duplicates O3: O cannot be inserted at all into list without duplicate Wect 			
		ID	L	0
		WECT1	L1	01
		WECT2	L2	O2
		WECT3	L1	O3
	• Sect			
		ID	L	I
		SECT1	L1	O1
		SECT2	L2	O1
		SECT3	L1	O2
		SECT4	L2	O2

					·
		SECT5	L1		О3
		SECT6	L2		О3
		and Srect would and srect are al			s not possible f a null variable list
SetUniqueListTest .testSet	• Equiva • • • • • • • • • • • • • • • • • • •	es: List length L Index I Object O lence classes L1: L = 0 L2: L > 0 I1: I < 0 I2: I >= 0 && I = 0 I3: I >= L O1: O is not a condition of the	duplicate		
		ID	L	1	0
		WECT1	L1	I1	01
		WECT2	L2	12	O2
		WECT3	L1	13	O1
	• Sect				
		ID	L	I	0
		SECT1	L1	I1	O1
		SECT2	L2	I1	O1
		SECT3	L1	12	O1
		SECT4	L2	12	O1
		SECT5	L1	13	01
		SECT6	L2	13	01
		SECT7	L1	I1	O2
		SECT8	L2	I1	O2

SECT9	L1	12	O2
SECT10	L2	12	O2
SECT11	L1	13	O2
SECT12	L2	13	O2

0

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by testing of a null variable list

SetUniqueListTest .testRemove

- Variables:
 - o List length L
 - o Index I
- Equivalence classes
 - o L1: L = 0
 - o L2: L > 0
 - o I1: I < 0
 - I2: I >= 0 && I < L
 - I3: I >= L
- Wect

0

ID	L	I
WECT1	L1	I1
WECT2	L2	12
WECT3	L1	13

Sect

C

ID	L	_
SECT1	L1	I1
SECT2	L2	I 1
SECT3	L1	12
SECT4	L2	12
SECT5	L1	13
SECT6	L2	13

 \circ

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT5, and SECT6

	Wrect a	and srect are also c	overed by testing o	f a null variable list	
SetUniqueListTest .testRemovelf					
SetUniqueListTest .testRemoveAll	 Variables: List length L List of objects O Equivalence classes L1: L = 0 L2: L > 0 O1: O can be fully removed from list O2: O can be partially removed from list O3: No elements of O exist in list Wect Wect 				
		ID	L	0	
		WECT1	L1	O1	
		WECT2	L2	O2	
		WECT3	L1	O3	
	• Sect			_	
		ID	L	ı	
		SECT1	L1	O1	
		SECT2	L2	01	
		SECT3	L1	O2	
		SECT4	L2	O2	
		SECT5	L1	O3	
		SECT6	L2	O3	
		and Srect would inv and srect are also c			
SetUniqueListTest .testRetainAll		es: List length L List of objects O lence classes L1: L = 0 L2: L > 0 O1: All elements of	O exist in list		

	• Wect	O2: Some elements of				
		ID	L	0		
		WECT1	L1	O1		
		WECT2	L2	O2		
		WECT3	L1	О3		
	• Sect		,			
		ID	L	I		
		SECT1	L1	O1		
		SECT2	L2	O1		
		SECT3	L1	O2		
		SECT4	L2	O2		
		SECT5	L1	O3		
		SECT6	L2	O3		
		and Srect would inve and srect are also c				
SetUniqueListTest .testClear	 Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list This is already done Wrect and srect are covered by testing of a null variable list This is already done Wrect and srect are covered by testing of a null variable list This is already done					
SetUniqueListTest .testContains	0	es: List length L Object O lence classes L1: L = 0 L2: L > 0 O1: O is in list O2: O is not in list				

	• Wect				
		ID	L	0	
		WECT1	L1	O1	
		WECT2	L2	O2	
	• Sect				
		ID	L	I	
		SECT1	L1	01	
		SECT2	L2	01	
		SECT3	L1	O2	
		SECT4	L2	O2	
SetUniqueListTest .testContainsAll					
		ID	L	0	
		WECT1	L1	01	
		WECT2	L2	O2	
		WECT3	L1	O3	
	• Sect				
		ID	L	I	
		SECT1	L1	O1	
		SECT2	L2	01	

				-				
		SECT3	L1	O2				
		SECT4	L2	O2				
		 Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are also covered by testing of a null variable list 						
SetUniqueListTest .testIterator	 Variables: List length L Equivalence classes L1: L = 0 L2: L > 0 Thus, wect and sect are both covered by using an empty list and a nonempty This is already done Wrect and Srect would involve L < 0 but this is not possible Wrect and srect are covered by testing of a null variable list 							
SetUniqueListTest .testListIterator	EquivalOThus, wan noneOWrect a	List length L lence classes L1: L = 0 L2: L > 0 vect and sect are bo	e olve L < 0 but this is	s not possible				

Test	Boundary Valu	e Analysis			
PredicatedListTest .testGet	• Test Se	 Index I Test Set BVL: {nom: L = 0, nom: L > 0} BVI: {min-: I = -1, min: I = 0, min+: I = 1, nom: 1 < I < L, max-: I = L - 2, max: I = L - 1, max+: I = L} BVA (basic) 			
		ID	L	1	
		BVA (basic) 1	L > 0	0	
		BVA (basic) 2	L > 0	1	

		BVA (basic) 3	3	L > 0		nor	n
		BVA (basic) 4	ļ	L > 0		L - :	2
		BVA (basic) 5 L > 0 L			L-	1	
	● BVA (rc	bust)					
	C	ID		L		I	
		BVA (robust)	1	L > 0		-1	
		BVA (robust)	2	L > 0		0	
		BVA (robust)	3	L > 0		1	
		BVA (robust)	4	L > 0		nor	n
		BVA (robust)	5	L > 0		L - :	2
		BVA (robust)	6	L > 0		L-	1
		BVA (robust)	7	L > 0		L	
	•					-	
PredicatedListTest .testSet	 Variables: List length L Index I Object O Test Set BVL: {nom: L = 0, nom: L > 0} BVI: {min-: I = -1, min: I = 0, min+: I = 1, nom: 1 < I < L, max-: I = L - 2, max: I = L - 1, max+: I = L} BVO: {nom: O can be inserted, nom: O cannot be inserted} BVA (basic) 						
	0	ID	L		I		0
		BVA (basic)	L > (0	0		O can be inserted
		BVA (basic) 2	L > (0	1		O can be inserted
		BVA (basic)	L > (0	nom		O can be inserted

		BVA (basic) 4	L > 0	L - 2	O can be inserted
		BVA (basic) 5	L > 0	L - 1	O can be inserted
	BVA (ro	bust)			
		ID	L	I	О
		BVA (robust) 1	L > 0	-1	O can be inserted
		BVA (robust) 2	L > 0	0	O can be inserted
		BVA (robust) 3	L > 0	1	O can be inserted
		BVA (robust) 4	L > 0	nom	O can be inserted
		BVA (robust) 5	L > 0	L - 2	O can be inserted
		BVA (robust) 6	L > 0	L - 1	O can be inserted
		BVA (robust) 7	L > 0	L	O can be inserted
	•				
SetUniqueListTest .testSet	• Test Se	List length L Index I Object O t BVL: {nom: L = BVI: {min-: I = max-: I = L - 2, BVO: {nom: O	-1, min: I = 0, , max: I = L - 1	min+: I = 1, no	
		ID	L	I	0
		BVA (basic) 1	L > 0	0	O is unique

	BVA (basic) 2	L > 0	1	O is unique
	BVA (basic)	L > 0	nom	O is unique
	BVA (basic) 4	L > 0	L - 2	O is unique
	BVA (basic) 5	L > 0	L - 1	O is unique
BVA (ro	bust)			
	ID	L	I	0
	BVA (robust) 1	L > 0	-1	O is unique
	BVA (robust) 2	L > 0	0	O is unique
	BVA (robust) 3	L > 0	1	O is unique
	BVA (robust) 4	L > 0	nom	O is unique
	BVA (robust) 5	L > 0	L - 2	O is unique
	BVA (robust) 6	L > 0	L - 1	O is unique
	BVA (robust) 7	L > 0	L	O is unique
0				

Test	Decision Table
PredicatedListTest .testEquals	 Conditions C1: L <= 0? C2: Objects are equal? Actions A1: Objects are found to be equal Table O

		Test Case ID	C1	C2	Expecte d Output
		TC1	Y/T	-	-
		TC2	N/F	Y/T	True
		TC3	N/F	N/F	False
	0				<u> </u>
PredicatedListTest .testGet	○ • Actions	C1: L <= 0? C2: Index in	range?		
	O	Test Case ID	C1	C2	Expecte d Output
		TC1	Y/T	-	-
		TC2	N/F	Y/T	Object at index
		TC3	N/F	N/F	Error
	0				
PredicatedListTest .testIndexOf	○ • Actions	C1: L <= 0? C2: Object i	in list?		
		Test Case ID	C1	C2	Expecte d Output
		TC1	Y/T	-	-
		TC2	N/F	Y/T	Index of object
		TC3	N/F	N/F	-1
	0				
PredicatedListTest .tesetLastIndexOf	• Conditi	ons C1: L <= 0?			

	Actions	C2: Object	in list?			
	0	A1: Index is	gotten			
	• Table			_		
		Test Case ID	C1	C2	Expecte d Output	
		TC1	Y/T	-	-	
		TC2	N/F	Y/T	Index of object	
		тс3	N/F	N/F	-1	
	0			•	•	
PredicatedListTest .testRemove	Actions	C1: L <= 0? C2: Object	in list?			
		Test Case ID	C1	C2	Expecte d Output	
		TC1	Y/T	-	-	
		TC2	N/F	Y/T	Object removed	
		TC3	N/F	N/F	No change	
	0			•		
PredicatedListTest .testAdd	 Conditions C1: L <= 0? C2: Object can be inserted? Actions A1: Object is inserted Table O 					
		Test Case ID	C1	C2	Expecte d Output	
		TC1	Y/T	-	-	
		TC2	N/F	Y/T	Object	

					inserted					
		TC3	N/F	N/F	No change					
	0									
PredicatedListTest .testAddAll	o o • Actions	 C1: L <= 0? C2: Objects can be inserted? Actions A1: Objects are inserted Table 								
		Test Case ID	C1	C2	Expecte d Output					
		TC1	Y/T	-	-					
		TC2	N/F	Y/T	Objects inserted					
		тС3	N/F	N/F	No change					
	0									
PredicatedListTest .testSet	o o • Actions	C1: L <= 0? C2: Index ir C3: Object	n range? can be inse							
		Test Case ID	C1	C2	C3	Expecte d Output				
		TC1	Y/T	-	-	-				
		TC2	N/F	Y/T	Y/T	Object at index is set				
		TC3	N/F	Y/T	N/F	Error				
		TC4	N/F	N/F	Y/T	Error				
		TC5	N/F	N/F	N/F	Error				
	0									

PredicatedListTest .testSublist	 Conditions C1: L <= 0? C2: Start index in range? C3: End index in range? Actions A1: Sublist is made Table Conditions C3: End index in range? 								
		Test Case ID	C1	C2	C3	Expecte d Output			
		TC1	Y/T	-	-	-			
		TC2	N/F	Y/T	Y/T	Sublist			
		тс3	N/F	Y/T	N/F	Error			
		TC4	N/F	N/F	Y/T	Error			
		TC5	N/F	N/F	N/F	Error			
.testAdd	Actions	C1: L <= 0? C2: Object i	-	rithout remo	oval				
	O	Test Case ID	C1	C2	Expecte d Output				
		TC1	Y/T	-	-				
		TC2	N/F	Y/T	Object inserted without removal				
		TC3	N/F	N/F	Object inserted with removal				
	0								
SetUniqueListTest .testAddAll	 Conditions C1: L <= 0? C2: Objects are unique? Actions 								

		A 4 . Obi 4-	:	-1				
	• Table	A1: Objects	are inserted	a without re	movais			
		Test Case ID	C1	C2	Expecte d Output			
		TC1	Y/T	-	-			
		TC2	N/F	Y/T	Objects inserted without removal			
		TC3	N/F	N/F	Objects inserted with removal			
	0							
	 C1: L <= 0? C2: Index in range? C3: Object is unique Actions A1: Object at index is set without removal Table 							
		Test Case ID	C1	C2	C3	Expecte d Output		
		TC1	Y/T	-	-	-		
		TC2	N/F	Y/T	Y/T	Object at index is set without removal		
		TC3	N/F	Y/T	N/F	Object at index is set with removal		
		TC4	N/F	N/F	-	Error		
	0							
SetUniqueListTest .testRemove	ConditionsC1: L <= 0?C2: Object in list?							

	Actions	: A1: Object i	removed					
	Table	711. 00,000	omovou.					
	0	Test Case ID	C1	C2	Expecte d Output			
		TC1	Y/T	-	-			
		TC2	N/F	Y/T	Object removed			
		TC3	N/F	N/F	No change			
	0				<u>. </u>			
SetUniqueListTest .testRemoveIf								
		Test Case ID	C1	C2	Expecte d Output			
		TC1	Y/T	-	-			
		TC2	N/F	Y/T	Objects removed			
		TC3	N/F	N/F	No change			
	0							
SetUniqueListTest .testRemoveAll	 Conditions C1: L <= 0? C2: Objects exist to be removed? Actions A1: Objects removed Table 							
	0	Test Case ID	C1	C2	Expecte d Output			
		TC1	Y/T	-	-			
		TC2	N/F	Y/T	Objects			

					removed		
		TC3	N/F	N/F	No change		
	0						
SetUniqueListTest .testRetainAll	Actions	C1: L <= 0? C2: Objects	exist to be	retained?			
		Test Case ID	C1	C2	Expecte d Output		
		TC1	Y/T	-	-		
		TC2	N/F	Y/T	Objects retained		
		TC3	N/F	N/F	Empty List		
	0						
SetUniqueListTest .testContains	Actions	C1: L <= 0? C2: Object i	in list?				
		Test Case ID	C1	C2	Expecte d Output		
		TC1	Y/T	-	-		
		TC2	N/F	Y/T	True		
	0	TC3	N/F	N/F	False		
SetUniqueListTest .testContainsAll	 Conditions C1: L <= 0? C2: Objects in list? Actions A1: Objects found in list Table 						

Expecte d Output Test C1 C2 Case ID Y/T TC1 TC2 N/F Y/T True TC3 N/F N/F False 0

SetUniqueListTest .testSublist

Conditions

o C1: L <= 0?

C2: Start index in range?C3: End index in range?

Actions

o A1: Sublist is made

Table

0

Test Case ID	C1	C2	C3	Expecte d Output
TC1	Y/T	-	-	-
TC2	N/F	Y/T	Y/T	Sublist
ТС3	N/F	Y/T	N/F	Error
TC4	N/F	N/F	Y/T	Error
TC5	N/F	N/F	N/F	Error

Test	Data Flow Coverage								
PredicatedListTest .testEquals	•	Criterion	object	Program Paths	Test Inputs				
		All-def	<99, 100>	<99, 100>	(object = {2, 4, 6, 8, 10})				
		All-use	<99, 100>	<99, 100>	(object = {2, 4, 6, 8, 10})				
		All-P-user-so	<99, 100>	<99, 100>	(object = {2,				

		Γ	I			
		me-C-uses			4, 6, 8, 10})	
		All-C-user-so me-P-uses	<99, 100>	<99, 100>	(object = {2, 4, 6, 8, 10})	
		All DU paths	<99, 100>	<99, 100>	(object = {2, 4, 6, 8, 10})	
PredicatedListTest	•		ı	1		
.testGet		Criterion	index	Program Paths	Test Inputs	
		All-def	<110, 111>	<110, 111>	(index = 0)	
		All-use	<110, 111>	<110, 111>	(index = 0)	
		All-P-user-so me-C-uses	<110, 111>	<110, 111>	(index = 0)	
		All-C-user-so me-P-uses	<110, 111>	<110, 111>	(index = 0)	
		All DU paths	<110, 111>	<110, 111>	(index = 0)	
				•		
PredicatedListTest	•		Ι	1		
.testIndexOf		Criterion	object	Program Paths	Test Inputs	
		All-def	<115, 116>	<115, 116>	(object = 2)	
		All-use	<115, 116>	<115, 116>	(object = 2)	
		All-P-user-so me-C-uses	<115, 116>	<115, 116>	(object = 2)	
		All-C-user-so me-P-uses	<115, 116>	<115, 116>	(object = 2)	
		All DU paths	<115, 116>	<115, 116>	(object = 2)	
		•	!		·	
PredicatedListTest	•		ı	i		
.tesetLastIndexOf		Criterion	object	Program Paths	Test Inputs	
		All-def	<120, 121>	<120, 121>	(object = 2)	

		All-use		<120, 12	:1>	<120	, 121>	(0	bject = 2)	
		All-P-user-s me-C-uses	0	<120, 12	<120, 121>		<120, 121>		(object = 2)	
		All-C-user-so me-P-uses		<120, 121>		<120, 121>		(0	(object = 2)	
		All DU paths	3	<120, 12	:1>	<120	, 121>	(C	object = 2)	
PredicatedListTest	•									
.testRemove		Criterion		index		Progr Paths		Te	est Inputs	
		All-def		<125, 12	:6>	<125	, 126>	(0	object = 2)	
		All-use		<125, 12	:6>	<125	, 126>	(0	object = 2)	
		All-P-user-seme-C-uses	0	<125, 12	:125, 126> <12		125, 126>		(object = 2)	
		All-C-user-so me-P-uses		<125, 126>		<125, 126>		(0	(object = 2)	
		All DU paths	3	<125, 126>		<125, 126>		(C	(object = 2)	
PredicatedListTest	•									
.testAdd	_	Criterion	ind	dex	objec	t	Program Paths		Test Inputs	
		All-def		31, 3>	<131, 132>		<131, 13 133>	2,	(object = 2)	
		All-use		31, 33>	<131, 132> <131, 133>		<131, 132, 133>		(object = 2)	
		All-P-user- some-C-u ses		31, 33>	<131, 132>		<131, 13 133>	2,	(object = 2)	
		All-C-user- some-P-us es		31, 33>	<131, 132> <131, 133>		<131, 13 133>	2,	(object = 2)	

	All DU paths		<131, 132> <131, 132, 133>	<131, 132, 133>	(object = 2)
PredicatedListTest .testAddAll	Criter	ion index	coll	Program Paths	Test Inputs
	All-de	f <137, {138, 139}*, 138, 141	<137, 138>	<137, {138, 139}*, 138, 141>	(coll = {12, 14, 16})
	All-us	e <137, {138, 139}*, 138, 141	<137, {138, 139}*, > 138> <137, {138, 139}*, 138, 139> <137, {138, 139}*, 138, 139, 141>	<137, {138, 139}*, 138, 141>	(coll = {12, 14, 16})
	All-Psome ses	· · · · · · · · · · · · · · · · · · ·	<137, {138, 139}*, > 138>	<137, {138, 139}*, 138, 141>	(coll = {12, 14, 16})
	All-C- some es		<137, {138, 139}*, > 138> <137, {138, 139}*, 138, 139> <137, {138, 139}*, 138, 139, 141>	<137, {138, 139}*, 138, 141>	(coll = {12, 14, 16})
	All DU	J <137,	<137,	<137,	(coll = {12,

		paths	{138, 139}*, 138, 14	1>	{138, 139}*, 138> <137, {138, 139}*, 138, 139> <137, {138, 139}*, 138, 139, 141>	{13 139 138		14, 16})
PredicatedListTest .testListIterator	•							
lesicistiterator		Criterion					Progran	n Paths
		All-def	All-def					
		All-use						
		All-P-user-some-C- uses						
		All-C-user-some-P- uses						
		All DU paths	3					
PredicatedListTest	•							
.testSet		Criterion	index		object	Pro Pat	gram hs	Test Inputs
		All-def	<155, 1 157>	56,	<155, 156>	<15 157	55, 156, '>	(index = 0, object = 0)
		All-use	<155, 1 157>	56,	<155, 156> <155, 156, 157>	<15 157	55, 156, '>	(index = 0, object = 0)
		All-P-user- some-C-u ses	<155, 1 157>	56,	<155, 156>	<15 157	55, 156, '>	(index = 0, object = 0)
		All-C-user- some-P-us	<155, 1 157>	56,	<155, 156>	<15 157	55, 156, '>	(index = 0, object = 0)

						•			
		es		<155, 157>	156,				
		All DU paths	<155, 156, 157>	<155, 156> <155, 157>		<155, 15 157>	6,	(index = 0, object = 0)	
SetUniqueListTest	•								
.testAdd		Criterion	object	sizeB	efore	Program Paths		Test Inputs	
		1		<127, 133>	130,	<125, 12 130, 133		(object = 1)	
		All-use	<125, 127, 130>	<127, 133>	130,	<125, 12 130, 133		(object = 1)	
		some-C-u 130> 133> ses		<127, 133>	130, <125, 127 130, 133			(object = 1)	
				<127, 133>	130,	<125, 127, 130, 133>		(object = 1)	
		All DU paths	<125, 127, 130>	<127, 130, 133>		<125, 127, 130, 133>		(object = 1)	
SetUniqueListTest	•		1		T		_		
.testAddAll		Criterion	coll		Program Paths		Te	Test Inputs	
		All-def	<170, 17	71>	<170	, 171>		coll = {1, 2, , 4, 5, 1, 2, })	
		All-use <170		71> <170		3		coll = {1, 2, , 4, 5, 1, 2, })	
		All-P-user-some-C-uses	o <170, 17	71>	> <170, 171>		3,	(coll = {1, 2, 3, 4, 5, 1, 2, 3})	
		All-C-user-s	o <170, 17	71>	<170	, 171>	(0	coll = {1, 2,	

			_									
		me-P-uses All DU paths							3, 4, 5, 1, 2, 3})			
				<170, 171> <1		<170, 171>		(coll = {1, 2, 3, 4, 5, 1, 2, 3})				
SetUniqueListTest												
.testSet		Criteri on	index	object	pos		remov ed	m	ogra aths	Test Inputs		
		All-def	<213, 214, 215>	<213, 214>	<214 215, {217 220} 217	, , ,*,	<215, {217, 220}*, 217, 223>		{217, 220}*, 217, 223>		213, 4, 5, 17, 20)*, 7, 23,	(index = 1, object = 5)
		All-use	<213, 214, 215> <213, 214, 215, {217, 220}*, 217>	<213, 214> <213, 214, 215> <213, 214, 215, {217, 220}*, 217, 223, 224>	<214 215, {217 220} 217; <214 215, {217 220}	, , , , ,	<215, {217, 220}*, 217, 223> <215, {217, 220}*, 217, 223, 224, 226>	21 {2 22 21 22 22	213, 4, 5, 17, 20}*, 7, 23, 24,	(index = 1, object = 5)		
		All-P-u ser-so me-C- uses	<213, 214, 215, {217, 220}*, 217>	<213, 214>	<21 ⁴ 215, {217 220} 217	, , *,	<215, {217, 220}*, 217, 223>	21 {2 22 21 22 22	213, 4, 5, 17, 20}*, 7, 23,	(index = 1, object = 5)		
		All-C-u ser-so	<213, 214,	<213, 214>	<21 ⁴ 215,		<215, {217,		213, 4,	(index = 1,		

me-P- uses	215>	<213, 214, 215> <213, 214, 215, {217, 220}*, 217, 223, 224>	{217, 220}*>	220}*, 217, 223> <215, {217, 220}*, 217, 223, 224, 226>	215, {217, 220}*, 217, 223, 224, 226>	object = 5)
All DU paths	<213, 214, 215> <213, 214, 215, {217, 220}*, 217>	<213, 214> <213, 214, 215> <213, 214, 215, {217, 220}*, 217, 223, 224>	<214, 215, {217, 220}*, 217> <214, 215, {217, 220}*>	<215, {217, 220}*, 217, 223> <215, {217, 220}*, 217, 223, 224, 226>	<213, 214, 215, {217, 220}*, 217, 223, 224, 226>	(index = 1, object = 5)

SetUniqueListTest .testRemove

.

Criterion	index	result	Program Paths	Test Inputs
All-def	<239, 240>	<240, 241>	<239, 240, 241, 242>	(index = 2)
All-use	<239, 240>	<240, 241> <240, 241, 242>	<239, 240, 241, 242>	(index = 2)
All-P-user- some-C-u ses	<239, 240>	<240, 241>	<239, 240, 241, 242>	(index = 2)
All-C-user- some-P-us es	<239, 240>	<240, 241> <240, 241, 242>	<239, 240, 241, 242>	(index = 2)
All DU	<239,	<240,	<239, 240,	(index = 2)

					I		1		_	
		paths	240>		241> <240, 241, 242>		241, 242>			
SetUniqueListTest	•		1		ı		<u> </u>			
.testRemoveIf		Criterion	filter		result	result		Program Paths		est Inputs
		All-def	<249, 250>		<250 252>	, 251,		19, 250, I, 252>	(l e	filter = ist lement %) == 0)
		All-use	<249, 250> <249, 2 251>	50,	<250 252>	, 251,		19, 250, I, 252>	(l e	filter = ist lement %) == 0)
		All-P-user- some-C-u ses			<250 252>	, 251,		19, 250, I, 252>	(l e	filter = ist lement %) == 0)
		All-C-user- some-P-us es		50,	<250 252>	, 251,		19, 250, I, 252>	(l e	filter = ist lement %) == 0)
		All DU paths	<249, 250> <249, 250, 251>		<250, 252>			19, 250, I, 252>	(l e	filter = ist lement %) == 0)
								_		
SetUniqueListTest	•	г						1		
.testRemoveAll		Criterion	coll	res	ult	name)	Prograi Paths	m	Test Inputs
		All-def	<256, 257, {258, 259}*, 258>	257, {25 258, 259 259}*, 258		<258 {259, 258}* 259>		<256, 257, {258, 259}*, 258, 261>		(coll = {2, 3, 4, 6})
		All-use	<256, 257,	<25 {25	· ·		· · · · ·			(coll = {2, 3, 4,

			{258, 259}*, 258>	259}*, 258, 259> <257, {258, 259}*, 258, 261>	258}*, 259>	{258, 259}*, 258, 261>	6})
		All-P-us er-some -C-uses	<256, 257, {258, 259}*, 258>	<257, {258, 259}*, 258, 259>	<258, {259, 258}*, 259>	<256, 257, {258, 259}*, 258, 261>	(coll = {2, 3, 4, 6})
		All-C-us er-some -P-uses	<256, 257, {258, 259}*, 258>	<257, {258, 259}*, 258, 259> <257, {258, 259}*, 258, 261>	<258, {259, 258}*, 259>	<256, 257, {258, 259}*, 258, 261>	(coll = {2, 3, 4, 6})
		All DU paths	<256, 257, {258, 259}*, 258>	<257, {258, 259}*, 258, 259> <257, {258, 259}*, 258, 261>	<258, {259, 258}*, 259>	<256, 257, {258, 259}*, 258, 261>	(coll = {2, 3, 4, 6})
IniqueListTest RetainAll	•			<u> </u>			

SetUniqueListTest .testRetainAll

Criterion	coll	result	Program Paths	Test Inputs
All-def	<274, 275>	<275, 276>	<274, 275, 276, 279, 281, 285>	(coll = {1, 5, 7})
All-use	<274, 275>	<275, 276>	<274, 275, 276, 279,	(coll = {1, 5, 7})

					<275, 279, 2 285>		281, 285	>		
		some-C-u 275 ses		?74, '5>	<275, 276> <275, 276, 279, 281, 285>		<274, 275, 276, 279, 281, 285>		(coll = {1, 5, 7})	
				274, '5>			<274, 275, 276, 279, 281, 285>		(coll = {1, 5, 7})	
		All DU paths		274, '5>	<275, 276> <275, 279, 2 285>	276,	<274, 27 276, 279 281, 285	,	(coll = {1, 5, 7})	
SetUniqueListTest										
.testContains	•	Criterion	object		Prog Path:			Test Inputs		
		All-def		<295, 296>		<295, 296>		(object = 1)		
		All-use		<295, 296>		<295, 296>		(0	bject = 1)	
		All-P-user-s me-C-uses	0	<295, 29	6>	<295, 296>		(0	bject = 1)	
		All-C-user-s me-P-uses	0	<295, 29	6>	<295	95, 296>		bject = 1)	
		All DU paths	3	<295, 29	6>	<295, 296>			(object = 1)	
SetUniqueListTest	•									
.testContainsAll	•	Criterion		coll		Program Paths		Test Inputs		
		All-def		<300, 30	1>	<300	0, 301>		oll = {1, 5})	
		All-use		<300, 30	1>	<300	, 301>	(coll = {1, 5})		
		All-P-user-s me-C-uses	0	<300, 30	1>	<300	, 301>	(C	oll = {1, 5})	
		All-C-user-s	0	<300, 30	1>	<300	, 301>	(c	oll = {1, 5})	

me-P-us	ses			
All DU p	aths	<300, 301>	<300, 301>	(coll = {1, 5})