TEST DESIGN

Black-box Testing

Problem:

- Statement: find first sad position given a start position

Specification

Data	feelings, startingPosition
Precondition	feelings list of int, startingPosition int

Results	position in the list or -1
Postcondition	(-1 and no sad feelings in the given list starting from the startingPosition) or (position in the list and sad feelings is in the list) or (-1 and preconditions are not met)

Equivalence classes:

Number EC	**Condition**	**Valid EC**	**Invalid EC**
1	startingPosition int and from [0, length of the list)	startingPosition int and valid value	
2			startingPosition < 0
3			startingPosition > length of the feelings list
4	feelings list of int	list non-empty and with -1, 0 and 1 values	
5			empty list
6			list contains other invalid values
7	the result is -1 or position	r = -1	
8		r = position	

Test cases based on the equivalence classes:

No TC	**EC**	**input data**		**output data**	
		feelings	**startPosition**	**expected**	**actual result**
1	1, 4, 8	[0, 0, -1, 0, 1]	1	2	2
2	2, 7	[0, 0, -1, 0, 1]	-1	-1	-1

3	3, 7	[0, 0, -1, 0, 1]	100	-1	-1
4	5, 7	[]	0	-1	-1
5	1, 4, 7	[0, 0 , 1, 1, 0]	0	-1	-1
6	5, 7	[2, 0, -1, 0, 1]	0	-1	-1

Boundary value analysis:

Number BVA	Condition	BVA Test Case
1, 2, 3	startingPosition int in [0, length of feelings array)	-1
		1
		100

Test cases based on BVA:

		input data		output data	
No TC	BVA	feelings	startingPosition	expected	actual
1	**1**		**-1**	**-1**	**-1**
2	**2**		**1**	**position**	**position**
3	**3**		**100**	**-1**	**-1**

All test cases:

FINAL			input data		output data	
No TC	TC from EC	TC from BVA	**feelings**	**startPosition**	expected	actual result
1	1	2	[0, 0, -1, 0, 1]	1	2	2
2	2	1	[0, 0, -1, 0, 1]	-1	-1	-1
3	3	3	[0, 0, -1, 0, 1]	100	-1	-1
4	4		[]	0	-1	-1
5	5		[0, 0 , 1, 1, 0]	0	-1	-1
6	6		[2, 0, -1, 0, 1]	0	-1	-1

White-box Testing

Problem:

- Check the neighbors of a given position

Specification

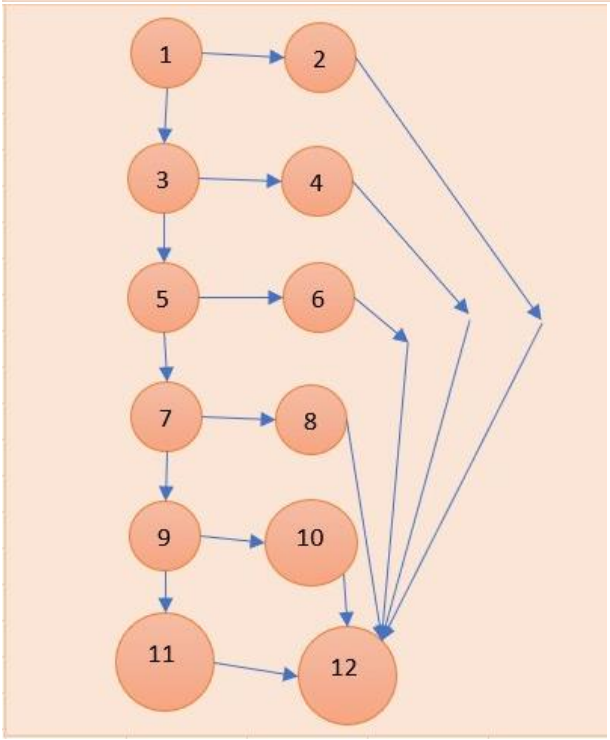
Data	feelings, position
Precondition	feelings array of int, position int

Results	true/false
Postcondition	(true and the neighbours of the given position are not 1) or (false and preconditions are not met) or (false and the neighbours are 1) or (false and the value on the position is not -1)

CFG, CC and paths:

```
public static Boolean checkNeighbours(List<Integer> feelings, Integer position) {
1       if (position < 0 || position >= feelings.size()) {
2               return false;
        }
3       if (feelings.get(position) != -1) {
4               return false;
        }
5       if (position == 0){
6               return true;
        }
7       if (position + 1 >= feelings.size()){
8               return true;
        }
9       if (feelings.get(position - 1) != 1 || feelings.get(position + 1) != 1){
10              return true;
        }
11      return false;
}

12
```



Cyclomatic complexity		
CC_1	regions	6
CC_2	E-N+2	6
CC_3	Predicate+1	6

Individual Paths	
1	1-2-12
2	1-3-4-12
3	1-3-5-6-12
4	1-3-5-7-8-12
5	1-3-5-7-9-10-12
6	1-3-5-7-9-11-12

Test cases:

						Condition/Decision coverage														Path coverage						Loop coverage						
	input data		output data		Statement	1				2		3		4		5				1	2	3	4	5	6							
						1		2								1		2														
No TC	feelings	position	expected	actual result	coverage	T	F	T	F	T	F	T	F	T	F	T	F	T	F							no	1	2	n-1	n	n+1	m<n
1	[0, 0, -1, 0, 1]	-1	FALSE	FALSE	1-2-12	x														x						x						
2	[0, 0, -1, 0, 1]	100	FALSE	FALSE	1-2-12		x	x												x						x						
3	[0, 0, 1, 0, -1]	2	FALSE	FALSE	1-3-4-12		x		x	x											x					x						
4	[-1, 0, -1, 0, 1]	0	TRUE	TRUE	1-3-5-6-12		x		x		x	x										x				x						
5	[0, 1, 0, 0, -1]	4	TRUE	TRUE	1-3-5-7-12		x		x		x		x	x									x			x						
6	[0, -1, 1, 0, 1]	1	TRUE	TRUE	1-3-5-7-9-10-12		x		x		x		x		x	x								x		x						
7	[1, -1, 0, 0, 1]	1	TRUE	TRUE	1-3-5-7-9-10-12		x		x		x		x		x		x	x						x		x						
8	[0, 1, -1, 1, 0]	2	FALSE	FALSE	1-3-5-7-9-11-12		x		x		x		x		x		x		x						x	x						

Integration Testing:

For integration testing we choose to use the Big Bang Integration approach, in which we test the entire system as a whole rather than individual functions. We execute a single test case that covers the complete functionality of the system. We execute the test case for the 'beHappy' method, which uses internally the other functions: 'findSadFeeling, checkNeighbours, insertHappyFeelings.

Input for the test case:

- It covers various cases in which the sad feeling can be found: at the beginning of the list, with no happy feelings as neighbors, with both neighbors as happy feelings, right neighbor is a happy feeling, left neighbor is a happy feeling
- Example: **[-1, -1, 0, 0, 1, 1, -1, 1, 0, -1, 1, 0, 1, 1, -1, 0, 1, 1]**

Expected result:

- **[1, -1, 1, -1, 1, 0, 0, 1, 1, -1, 1, 0, 1, -1, 1, 0, 1, 1, -1, 1, 0, 1, 1]**