

Christa Fox

# Basic Linked Collection

Test Case Plan



## Explanation

After or before writing code (dependent on your software development style), test cases are written to ensure that all types of entries have been accounted for before the end user receives your code. This minimizes bugs, and unstable programs. Constructors and all methods should be tested for valid input, and invalid input. Specific to this assignment where an integer is requested for a distance, negative integers would not make logic sense to be valid. The following is my test case plan for the distance class.

The operation is the actual code. The purpose is the pseudo code for what the test case is supposed to achieve. The object state is how the fields are affected, which is usually only applicable to constructors. The expected result is what the end result should be.

## Test Case Plan

### *Distance constructors (tests set methods as well)*

Operation	Purpose	Object State	Expected Result
Distance d1 = new Distance(0, 0);	Create a new distance object of 0 feet, 0 inches.	Feet = 0 Inches = 0	A new distance object was created with the listed values.
Distance d2 = new Distance( );	Create a distance object with the default values.	Feet = 1 Inches = 1	A new distance object with the default values
Distance d3 = new Distance(3, 2);	Create a new distance object of 3 feet, 2 inches.	Feet = 3 Inches = 2	A new distance object was created with the listed values.
Distance d4 = new Distance(-1, 1)	Create a distance object with the default values when invalid input is received.	Feet = 1 Inches = 1	A new distance object was created with the default values.
Distance d5 = new Distance(Integer.MAX_VALUE, 1)	Create a distance object of many feet and 1 inch.	Feet = 40,000 Inches = 1	A new distance object was created with the listed values.
Distance d6 = new Distance(Integer.MIN_VALUE, 1)	Create a distance object with the default values when invalid input is received.	Feet = 1 Inches = 1	A new distance object was created with the default values.

*Test Set and Get*

Operation	Purpose	Object State	Expected Result
d1.setFeet(3)	The amount of feet of d1 will change.		Feet = 3 Inches = 0
d2.setFeet(-3)	The amount of feet is invalid and should throw an exception.		InvalidInputException
d1.setInches(1)	The amount of inches in d1 will change.		Feet = 3 Inches = 1
d2.setInches(-10)	An invalid number of inches should throw an InvalidInputException.		InvalidInputException
d1.getFeet()	Returns the amount of feet in the distance object.		Feet = 3
d1.getInches()	Returns the amount of inches in the distance object.		Inches = 1

*Add methods (tests getDistance methods as well)*

Operation	Purpose	Object State	Expected Result
addDistance(d1, d2)	Add 2 distance methods. It converts them to inches, and then back to inches and feet in the getDistance method.		Feet = 1 Inches = 1
addDistance(d3, d4);	Add 2 distance methods it converts them to inches, and then back to inches and feet in the getDistance method.		Feet = 4 Inches = 3
addDistance(d5, d6);	Add 2 distance methods it converts them to inches, and then back to inches and feet in the getDistance method.		Feet = 41,000 Inches = 2

*Subtraction methods (tests getDistance methods as well)*

Operation	Purpose	Object State	Expected Result
-----------	---------	--------------	-----------------

subDistance(d1, d2)	Subtract 2 distance methods. It converts them to inches, and then back to inches and feet in the getDistance method.	Feet = 1 Inches = 1
subDistance(d3, d4);	Subtract 2 distance methods it converts them to inches, and then back to inches and feet in the getDistance method.	Feet = 2 Inches = 1
subDistance(d5, d6);	Subtract 2 distance methods it converts them to inches, and then back to inches and feet in the getDistance method.	Feet = 39,999 Inches = 0

#### *Test equals method and compare method*

Operation	Purpose	Object State	Expected Result
d1.equals(d2)	Check that 2 methods that are not equal are defined as such.		false
d2.equals(d4)	Check that 2 methods that are equal are defined as such.		true

#### *Test hash*

Operation	Purpose	Object State	Expected Result
assertFalse( d1.hashCode()==d2.hashCode() );	Check that 2 methods that are not equal are defined as such.		false
assertTrue( d2.hashCode()==d4.hashCode() );	Check that 2 methods that are equal are defined as such.		true

## Reflection

The main purpose of technology is to simplify tasks and make things efficient. Software development is no exception. If anything people depend on the use of a running and correct product more so, than with other aspects of technology. The reason for this is that many businesses depend on certain aspects or functionality that it provides.

Robustness is the ability for the program to accept invalid input. If one user of 200 employees could input something wrong and this shut the entire system down, the company would lose profits for the dead time.

JUnit was effective in testing for foreseeable problems. It runs through the code much faster than stepping through in debug. You are able to quickly identify the mismatched output and see what other methods are failing. I had called the add and subtract methods twice within each code block, and had not noticed until the test failed. I see why testing is an important part of programming.