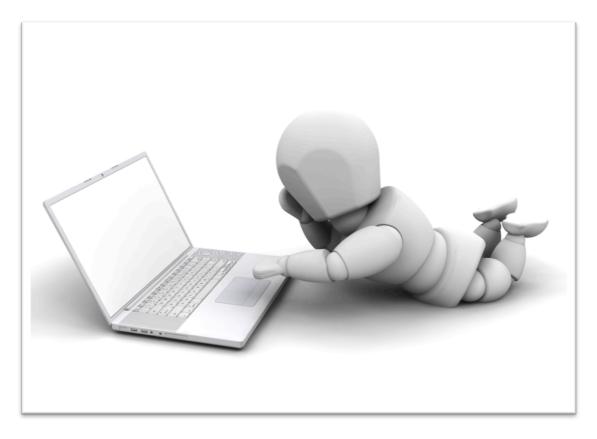
Christa Fox

Basic Linked Collection

Test Case Plan



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

Test Set and Get

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

Add methods (tests getDistance methods as well)

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

Subtraction methods (tests getDistance methods as well)

Operation	Purpose	Object State	Expected Result

Test equals method and compare method

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

Test hash

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

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.