**Assignment IV – Word Ladder**

Hari Kosuru – Kevin Yee

1. **Test Requirements – BlackBox and WhiteBox Testing**

Our test plan follows the two conventional styles of testing: black box testing and white box testing.

**BlackBox:**

Our black box testing approach is derived from pre-emptively knowing the results to certain inputs:

From the given solutions, we know that the words “heads” to “tails” has a possible word ladder. Thus, we tested the existence of such ladder and used a method known as “validateResult()” to confirm that the first and last entries of the list array contains “heads” and “tails” respectively. Furthermore, we confirmed that all the results between the first and last entry only differed by one character.

We also know that the words “angels” and “devils” have no word ladder between them. The method in which the solution was derived is no interest to us in blackbox testing. We simply confirmed that the list returned was empty.

**WhiteBox:**

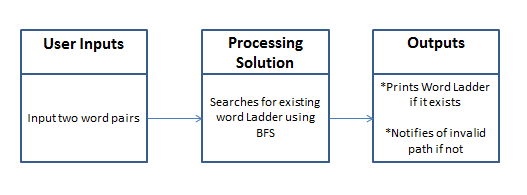
Our white box testing approach is derived from a programmer’s perspective and knowing the constraints.

We tested the wordLadder using special cases such as words that are note equal to five letters. Words with such properties were returned with a message prompting the user that there exist no paths between those two words.

Furthermore, other knowledge and test cases we were able to derive is the input of null strings. The input of null strings returns an empty list as well.

To structure our whitebox testing further, every correct input will print a message indicating that the result is indeed correct using our “validateResult()” method.

1. **System IPO**



1. **Use-Case Diagram**

|  |
| --- |
| 1. Groceries |
| Protected:  <Inherited fields>  Boolean perishable |
| CalculatePrice():  Uses: price, weight, q quantity, I perishable |

|  |
| --- |
| Item |
| Protected:  String name  Float price  Long quantity  Long weight  Boolean shipping |
| CalculatePrice():  Uses: price, weight, q quantity |

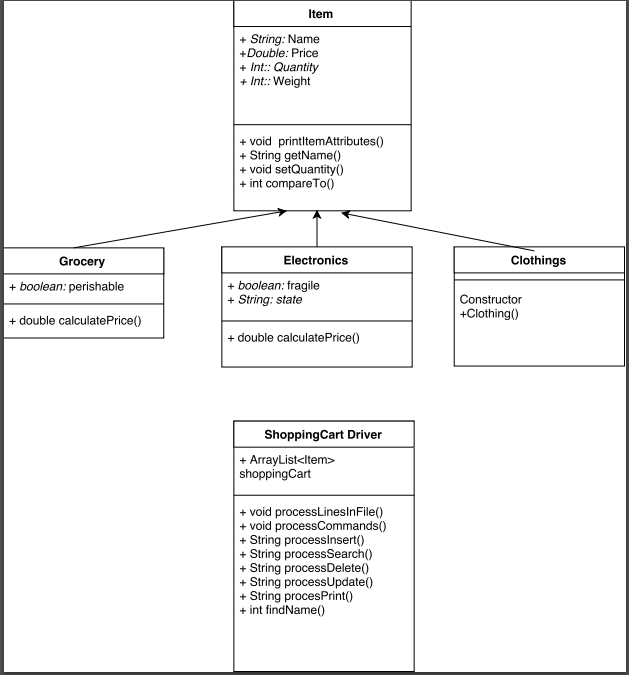
|  |
| --- |
| Clothing |
| Protected:  <Inherited fields> |
| CalculatePrice():  Uses: price, weight, q quantity |

|  |
| --- |
| Electronics |
| Protected:  <Inherited fields>  Boolean fragile  String state |
| CalculatePrice():  Uses: price, weight, q quantity, I fragile, state |

|  |
| --- |
| A3Driver / Main |
| Public:  String filename  ShoppingCart cart  String[] args  String[] userInput |
| ProcessInput():  Uses: userInput, args, cart, filename  isInputValid():  Uses: userInput  insertElectronic():  Uses: userInput, cart  insertGroceries():  Uses: userInput, cart  insertClothes():  Uses: userInput, cart  Insert(), search(), delete(), update();  Uses: cart  validState(), recoverLongAmount(), recoverFloatAmount();  Uses: userInput  print():  Uses: Compare() |

|  |
| --- |
| nameComparator |
| //No fields |
| Compare(); |

|  |
| --- |
| ShoppingCart |
| Public:  ArrayList<Item> |
| //No methods |

1. **ADT Level Description**
2. 

**Shopping Cart Diver:**

Parses String Inputs dependent on commands:

* Insert
* Search
* Delete
* Update
* Print

**Items:**

Super Class for Groceries, Clothing, Electronics

Stores Name of Items, Price, Quantity, and Weight

**Groceries:**

Extends Items

Determines if Items are perishable/non-perishable

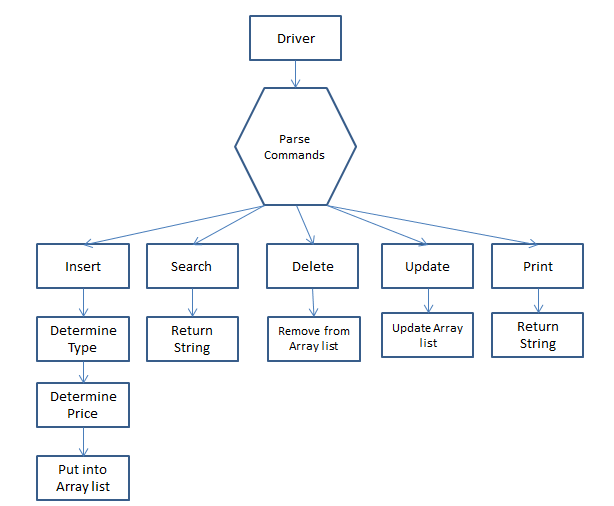
**Electronics:**

Extends Items

Determines if Items are Fragile/Non-Fraigle

Determines which States it must be shipped to.

1. **Functional Block Diagram**



1. **Algorithm**

While readline.next()

Commands[] = readlines.next();

If Commands[0] equals “insert” parseInsert()  
 If Commands[0] equals “delete” parseDelete()

If Commands[0] equals “search” parseSearch()

If Commands[0] equals “update” parseUpdate()

If Commands[0] equals “print” parsePrint()

Return

parseInsert()

GenerateItem(Commands);

Return

parseDelete()

Find item from Arraylist

Remove Index

Return

parseSearch()

Find item from Arraylist

Return Index

parseUpdate()

Find item from Arraylist

Update Values

Return

parsePrint()

Print Arraylist

Return