Mahimna Pyakuryal

mpyakury

1014826

[mpyakury@uoguelph.ca](mailto:mpyakury@uoguelph.ca)

11/01/2019

Chamber Class:

Constructor:

No public instance variables are utilized in this class as all variables that need to be accessed outside of this class are done so through

The constructor without parameters was removed as it was not used

The constructor runs through every possible descriptions for the contents of the chamber and sets the appropriate attributes for them.

This would be tested by inputting preset ChamberShape and ChamberContents variables. I would then ensure that the returned values of the description for the contents are equal to the expected string output that would be hard coded into the junit test itself

setShape method:

This would be tested by defining a ChamberShape with preset values by inputting roll values rather than letter them be determined randomly. This would allow me to check the expected string result for the description, expected integer result for the area and the expected integer result for the number of exits with the values returned by the same preset ChamberShape passed through the method.

int getDoorNum method:

This can be tested by creating a ChamberShape object and storing the number of exits value into a variable. If the same value is passed through to the constructor of the class, then the expected result of getDoorNum (ie. the integer defined in the test) can be compared to the actual result (value returned by getDoorNum) to see if it working correctly.

ArrayList<Door> getDoors method:

this can be tested by defining any ChamberShape object in a test, then checking the return value of the ArrayList<Door> that defines the same object parameters to ensure that the values are equivalent

void addMonster method:

this can be tested by predefining a Monster object in a test case and setting the type to a specific desired outcome instead of randomly rolling. Then passing through an object with the same predefined values. Finally you can compare these two Monster class objects to ensure they are equal.

void Print

this can be tested by defining any attribute of a Chamber, including contents, shape and exits. you would then print the defined attribute and ensure it matches with the print statements printed in this method.

void addTreasure method:

this can be tested in the same way as addMonster. You can define a Treasure object in the dnd package and send an object with the same definition through to the addTreasure method. You would then compare the return result to the predefined value and ensure they are equivalent.

Door class:

Constructor:

this can be tested by passing through a predefined Door and comparing the attributes of the return value of the object defined under this class. If all the attributes are not null are equivalent to the correct data type expected, then the constructor is working.

setTrapped method

this can be tested by manually setting a trap in the test case. Then setting the same trap case by inputting a roll value manually. You can then compare the return value of the method with the manually set Door with attribute trapped and check if they are equivalent.

setOpen method

this can be tested by manually setting a true result of an open door i.e. setting a door description to open, then checking if the result of passing through true into the method is equivalent to the preset value.

isArchway method

this can be tested by presetting a door to an archway then checking the return value of this method to ensure all the attributes matchup (always open, unlocked, not trapped)

getTrapDescription method

this can be tested by presetting a Trap object to a desired test condition index, then setting a door with the same description. Finally you would get the description of the preset Trap and compare it to the return string of this method. If they are equivalent, the method is working.

* all door methods were left as public as they are all utilized outside of the class

Passage Class

Constructor:

this can be tested by presetting a specific description to a PassageSection. You can then send through the same description into the parameter and compare the results. If the results are equivalent the constructor is working.

getDoors method

this can be tested by defining any ArrayList of Door’s with the same size as the number of doors in the current passage. If the .size() of the array lists are equivalent values the method is working.

getDoor method

this can be tested by predefining a door and setting all the appropriate attributes for the door in the test method. Then you can pass through an empty door and set the same attributes by defining equivalent descriptions. If the return value and the preset value in the test are equal, then the method is working.

getMonster method

this can be tested by creating an array of Monster objects and passing through a pre-defined object with all attributes stored in various strings and integer. You can then pass through a Monster with the same predefined attributes and compare the return value of the method with the predefined Monster at a given index in the ArrayList

addPassageSection

this can be tested by defining an ArrayList of type PassageSection and adding a pre-defined value into a specific index. You can then define an empty ArrayList of type PassageSection to pass through to the function. To check if the method works compare the two ArrayLists at the index where the value was appended, if the values are equivalent it’s working.

SectionTable/SetTable method

this can be tested by pre-defining an integer that represents a specific description in the table provided in assignment 2. Then you can define the string that would be at the pre-defined index. Finally you can pass the index you predefined earlier into the SectionTable method and compare the string return value with the predefined string in the test method.

* getDoors method was switched to private due to it not being utilized outside of the function and other strategies implemented to get the data to various classes in the program.
* getMonster was switched to private due to it only needing to be set within the Passage and specified PassageSection (which is all done within the Passage class)
* addPassageSection was switched to private because this is a method used to define a PassageSection, which is only referred to in the Passage class.
* settable was switched to private being it is only required within the class as Descriptions are passed through other means (such as Boolean return methods)

PassageSection Class

Constructor

This can be tested by back checking any individual value associated with the description passed through. This includes direction, length, or any of the Boolean values being initialized in the if statements. This would be done by pre-defining a direction (or any other attribute of description), then passing through a description that should yield the same result of direction. You can then compare the direction returned with the direction pre-defined and if they are equivalent, it works.

getMonster method

this can be checked by pre-defining the flag checkMonster to true then passing pre-defining a specific monster given an attribute passed to the dnd Monster package. If the method with the same index returns an equivalent value, then the method is working.

getDescription method

this can be tested by pre-defining a PassageSection and the description of the passageSection in a string. Then pass the description to define that section into the constructor. You can then call the getDescription method and compare the description with the description of your pre-define passage section, and if they are equivalent the method works.

* all methods in this class left as public as they are all referred to in the Passage class.