testDoor

testEnterItemOtherThanKey()

Description-> Tests that enter returns failure string when player attempts with item other than key
Method tested->enter()
Initial state->testPlayer in testRoom1, holding item other than key between testRoom1 and testRoom2
Input Data-> testPlayer
Output-> KEY-FAILURE string
Output State-> testPlayer did not enter testRoom2
Test covers-> enter() not allowing player through the door with item other than key
JUnit class-> testDoor

testEnterKeyFailure()

Description-> Tests that enter returns failure string when player attempts to enter without key
Method tested->enter()
Initial state-> testPlayer in testRoom1, not holding key to door between testRoom1 and testRoom2
Input Data-> testPlayer
Output-> KEY-FAILURE string
Output State-> testPlayer did not enter testRoom2
Test covers-> enter() not allowing player through the door without key
JUnit class-> testDoor

testEnterNeitherInNorOutsite()

Description-> Tests that enter returns failure string when player attempts to enter without key
Method tested->enter()
Initial state-> testPlayer in testRoom3, holding key to door between testRoom1 and testRoom2
Input Data-> testPlayer
Output-> KEY-FAILURE string
Output State-> testPlayer did not enter testRoom2
Test covers-> enter() not allowing player through the door when in incorrect room
JUnit class-> testDoor

testEnterSuccessInSite()

Description-> Tests that enter returns success string when player attempts to enter without key
Method tested->enter()
Initial state-> testPlayer in testRoom1, holding key to door between testRoom1 and testRoom2
Input Data-> testPlayer
Output-> KEY-Success string
Output State-> testPlayer entered testRoom2
Test covers-> enter() allowing player through the door with key

JUnit class-> testDoor

testEnterSuccessOutSite()
Description-> Tests that enter returns success string when player attempts to enter without key
Method tested->enter()
Initial state-> testPlayer in testRoom2, holding key to door between testRoom1 and testRoom2
Input Data-> testPlayer
Output-> KEY-Success string
Output State-> testPlayer entered testRoom1
Test covers-> enter() allowing player through the door with key
JUnit class-> testDoor

testPlayer
testDropLowerBound()
Description-> Tests that drop() handles lowerBound inputs
Method tested-> drop()
Initial state-> testRoom1 contains testPlayer and testPlayer contains testItem
Input Data-> int representing particular item in array
Output-> None, or a message saying Player does not contain such item
Output State-> None
Test covers-> lowerBound test for drop()
JUnit class-> testPlayer

testDropUpperBound()
Description-> Tests that drop() handles upperBound inputs
Method tested-> drop()
Initial state-> testRoom1 contains testPlayer and testPlayer contains testItem
Input Data-> int representing particular item in array
Output-> None, or a message saying Player does not contain such item
Output State-> None
Test covers-> upperBound test for drop()
JUnit class-> testPlayer

testDropValidFirstItem()
Description-> Tests that drop() works correctly for first item
Method tested-> drop()
Initial state-> testRoom1 contains testPlayer and testPlayer contains testItem1
Input Data-> int representing particular item in array
Output-> None
Output State-> Player no longer contains item, testRoom1 contains item, testPlayer's itemCount decremented by 1
Test covers-> test that drop() works correctly for single item

JUnit class-> testPlayer

testDropValidSecondItem()
Description-> Tests that drop() works correctly for first item
Method tested-> drop()
Initial state-> testRoom1 contains testPlayer and testPlayer contains testItem1 and testItem2
Input Data-> int representing particular item in array
Output-> None
Output State-> Player no longer contains testItem2, testRoom1 contains testItem2, testPlayer's itemCount decremented by 1
Test covers-> test that drop() works correctly for single item
JUnit class-> testPlayer

testDropWithoutItem()
Description-> Tests that drop() handles player attempting to drop item not contained by player
Method tested-> drop()
Initial state-> testRoom1 contains testPlayer
Input Data-> int representing particular item in array
Output-> None, or message saying testPlayer does not contain item
Output State-> None
Test covers-> test that drop() does not affect state when player doesn't have item
JUnit class-> testPlayer

testGoInValidDirectionNegative()
Description-> Tests that go() handles lowerBound inputs
Method tested-> go()
Initial state-> testRoom1 contains testPlayer
Input Data-> int representing particular direction
Output-> None, or a message saying Player cannot go in that direction
Output State-> IndexOutOfBoundsException
Test covers-> lowerBound test for drop()
JUnit class-> testPlayer

testGoInValidDirectionPositive()
Description-> Tests that go() handles upperBound inputs
Method tested-> go()
Initial state-> testRoom1 contains testPlayer
Input Data-> int representing particular direction
Output-> None, or a message saying Player cannot go in that direction
Output State-> IndexOutOfBoundsException
Test covers-> upperBound test for drop()
JUnit class-> testPlayer

testGoPlayerHasNoLoc()
Description-> Tests that go() handles null input
Method tested-> go()
Initial state-> testRoom1 connected to testRoom2 but testPlayer is in neither location
Input Data-> int representing particular direction
Output-> None, or a message saying Player has no location
Output State-> NullPointerException
Test covers-> go() handles null pointers
JUnit class-> testPlayer

testGoThroughDoorWithKey()
Description-> Tests that go() works correctly when moving through door
Method tested-> go()
Initial state-> testRoom1 connected to testRoom2 with door and testPlayer in testRoom1 containing correct key
Input Data-> int representing particular direction
Output-> None
Output State-> testPlayer in testRoom2
Test covers-> go() works correctly when door is between rooms
JUnit class-> testPlayer

testGoThroughDoorWithoutKey()
Description-> Tests that go() handles attempt of player to move through door without having key
Method tested-> go()
Initial state-> testRoom1 connected to testRoom2 with door and testPlayer in testRoom1 without the key
Input Data-> int representing particular direction
Output-> None, or message saying testPlayer doesn't contain correct key
Output State-> None
Test covers-> go() works correctly when door is between rooms and player doesn't have key
JUnit class-> testPlayer

testGoValidDirection()
Description-> Tests that go() works correctly for simplest case
Method tested-> go()
Initial state-> testRoom1 connected to testRoom2 testPlayer in testRoom1
Input Data-> int representing particular direction
Output-> None
Output State-> testPlayer in testRoom2
Test covers-> go() works correctly

JUnit class-> testPlayer

testPickUpItemNotInRoom()
Description-> Tests that pickUp() handles attempt to pick up item not in room
Method tested-> pickUp()
Initial state-> testRoom1 contains testPlayer and testItem2 but not testItem1
Input Data-> Item player is attempting to pick up
Output-> None, or message saying item is not in room
Output State-> None
Test covers-> pickUp() works correctly when item is not in room
JUnit class-> testPlayer

testPickUpMoreThanCanCarry()
Description-> Tests that pickUp() handles attempt to pick up item when player can't hold any more items
Method tested-> pickUp()
Initial state-> testRoom1 contains testPlayer and testItem3, testPlayer contains testItem1 and testItem2
Input Data-> Item player is attempting to pick up
Output-> None, or message saying Player's hands are full
Output State-> None
Test covers-> pickUp() handles attempt to pick up item player cannot hold
JUnit class-> testPlayer

testPickUpValid()
Description-> Tests that pickUp() works correctly for simplest case
Method tested-> pickUp()
Initial state-> testRoom1 contains testPlayer and testItem1
Input Data-> Item player is attempting to pick up
Output-> None
Output State-> Player contains testItem1
Test covers-> pickUp() works for simplest case
JUnit class-> testPlayer

testRoom
testAddItemValid()
Description-> Tests that addItem() works correctly for simplest case
Method tested-> addItem()
Initial state-> testRoom1 does not contain testItem1
Input Data-> testItem1
Output-> None
Output State-> testRoom1 contains testItem1

Test covers-> addItem() works for simplest case
JUnit class-> testRoom

testEnterNullPlayer()
Description-> Tests that enter() handles case where player is null
Method tested-> enter()
Initial state-> testRoom1 does not contain testPlayer and testPlayer is set to null
Input Data-> testPlayer
Output-> None, or message that player is null
Output State-> NullPointerException
Test covers-> enter() handles case of null player
JUnit class-> testRoom

testEnterNullRoom()
Description-> Tests that enter() handles case where room is null
Method tested-> enter()
Initial state-> testRoom1 is set to null
Input Data-> testPlayer
Output-> None, or message that Room is null
Output State-> NullPointerException
Test covers-> enter() handles case of null player
JUnit class-> testRoom

testEnterSameLocation()
Description-> Tests that enter() handles case where player is trying to enter room player is alread in
Method tested-> enter()
Initial state-> testRoom1 does contains testPlayer Input Data-> testPlayer
Output-> None, or message that player is already in the room
Output State-> None
Test covers-> enter() does not move the player
JUnit class-> testRoom

testEnterValid()
Description-> Tests that enter() works correctly for simplest case
Method tested-> enter()
Initial state-> testRoom1 does not contain testPlayer
Input Data-> testPlayer
Output-> ROOM-ENTER message
Output State-> testRoom1 contains testPlayer
Test covers-> enter() works for simplest case
JUnit class-> testRoom

testExitInValidDirection()
Description-> Tests that enter() handles case of invalid direction
Method tested-> enter()
Initial state-> testRoom1 contains testPlayer and is connected to testRoom2
Input Data-> int representing particular direction
Output-> None, or message that invalid direction was chosen
Output State-> None
Test covers-> enter() handles case of invalid direction
JUnit class-> testRoom

testExitLowerBound()
Description-> Tests that enter() handles case of direction where input is a lowerBound of direction array
Method tested-> enter()
Initial state-> testRoom1 contains testPlayer and is connected to testRoom2
Input Data-> int representing particular direction
Output-> None, or message that invalid direction was chosen
Output State-> ArrayOutofBoundsException
Test covers-> enter() works for lowerBound value
JUnit class-> testRoom

testExitUpperBound()
testExitLowerBound()
Description-> Tests that enter() handles case of direction where input is an upperBound of direction array
Method tested-> enter()
Initial state-> testRoom1 contains testPlayer and is connected to testRoom2
Input Data-> int representing particular direction
Output-> None, or message that invalid direction was chosen
Output State-> ArrayOutofBoundsException
Test covers-> enter() works for upperBound value
JUnit class-> testRoom

testExitValid()
Description-> Tests that exit() works correctly for simplest case
Method tested-> exit()
Initial state-> testRoom1 contains testPlayer and is connected to testRoom2
Input Data-> an int representing a direction and testPlayer
Output-> ROOM-ENTER message
Output State-> testRoom2 contains testPlayer

Test covers-> exit() works for simplest case
JUnit class-> testRoom

testRemoveItemNotInRoom()
Description-> Tests that removeItem() handles case where item is not in room
Method tested-> removeItem()
Initial state-> testRoom1 contains testPlayer and testItem2
Input Data-> testItem1
Output-> None, or message that item is not in room
Output State-> testItem1 is still in testRoom1 and player still contains testItem2
Test covers-> removeItem() handles case where item is not in room
JUnit class-> testRoom

testRemoveItemValid()
Description-> Tests that removeItem() works correctly for simplest case
Method tested-> removeItem()
Initial state-> testRoom1 contains testPlayer and testItem1
Input Data-> testItem1
Output-> None
Output State-> testItem1 is no longer in testRoom1
Test covers-> removeItem() works for simplest case
JUnit class-> testRoom