Test Strategy – Player Class

**Test plan**

1. Create a Player object with the default constructor.
2. Create a Player object with the non-default constructor:
   * with valid field values
   * with invalid field values
3. Test all get methods:
   * Test getBackJumps()
   * Test getCoins()
   * Test getInventory()
   * Test getName()
   * Test getSizeOfInventory()
4. Test all set methods:
   * Test setBackJumps()
     + with valid values
     + with invalid values
   * Test setCoins()
     + with valid values
     + with invalid values
   * Test setInventory()
     + with valid values
   * Test setName()
     + with valid values
     + with invalid values
5. Test all other methods:
   * Test addToInventory()
   * Test collectCoins()
   * Test generatePoliceBribe()
   * Test jumpBack()
     + With valid values (tested in game play)
     + With invalid values
   * Test jumpForward()
     + With valid values (tested in game play)
     + With invalid values
   * Test removeCloak()

**Test 1**

Create a Player object with the default constructor.

Test data:

* name: “unknown”
* backJumps: -1
* coins: -1
* inventory: arraylist item size 1

Expected results:

* name: “unknown”
* backJumps: -1
* coins: -1
* cloaks: 0

Actual results:

* name: “unknown”
* backJumps: -1
* coins: -1
* cloaks: 0

Passed

**Test 2.1**

Create a Player object with the non-default constructor with valid field values

Test data:

* name: “Bob”
* backJumps: 3
* coins: 10
* inventory: arraylist item size 1

Expected results:

* name: “Bob”
* backJumps: 3
* coins: 10
* cloaks: 1

Actual results:

* name: “Bob”
* backJumps: 3
* coins: 10
* cloaks: 1

Passed

**Test 2.2**

Create a Player object with the non-default constructor with **invalid** field values.

Test data:

* name: “B”
* backJumps: -1
* coins: -1
* inventory: arraylist item size 1

Expected results:

* name: “B”
* backJumps: -1
* coins: -1
* cloaks: 1

Actual results:

* name: “B”
* backJumps: -1
* coins: -1
* cloaks: 1

Failed – but gameplay checks player fields are never of invalid value

**Test 3.1**

Test getBackJumps().

Test data:

* backJumps: 3

Expected results:

* backJumps: 3

Actual results:

* backJumps: 3

**Test 3.2**

Test getCoins ().

Test data:

* coins: 10

Expected results:

* coins: 10

Actual results: 10

Passed

**Test 3.3**

Test getInventory ().

Test data:

* inventory: arraylist of Item of size 1

Expected results:

* inventory: arraylist of Item

Actual results:

Arraylist of item (hash code)

Passed

**Test 3.4**

Test getName ().

Test data:

* name: “Bob”

Expected results:

* name: “Bob”

Actual results:

“Bob”

Passed

**Test 3.5**

Test getSizeOfInventory().

Test data:

* inventory: arraylist item size 1

Expected results:

* 1

Actual results: 1

Passed

**Test 4.1.1**

Test setBackJumps() with valid values

Test data:

* backJumps: 3

Expected results:

* backJumps: 3

Actual results:

3

Passed

**Test 4.1.2**

Test setBackJumps() with **invalid** values

Test data:

* backJumps: -1

Expected results:

* backJumps: -1

Actual results:

-1

Failed – but game play checks they are never below 0

**Test 4.2.1**

Test setCoins() with valid values

Test data:

* coins: 10

Expected results:

* coins: 10

Actual results:

10

Passed

**Test 4.2.2**

Test setCoins() with **invalid** values

Test data:

* coins: -1

Expected results:

* coins: -1

Actual results:

-1

Failed – but gameplay checks they never go below 0

**Test 4.3.1**

Test setInventory() with valid values

Test data:

* inventory: arraylist item size 2

Expected results:

* cloaks: 2

Actual results:

2

Passed

**Test 4.4.1**

Test setName() with valid values

Test data:

* name: “Vee”

Expected results:

* name: “Vee”

Actual results:

Vee

Passed

**Test 4.4.2**

Test setName() with **invalid** values

Test data:

* name: “V”

Expected results:

* name: “V”

Actual results:

V

Failed – but player.name is validated in another part of the game play where it is passed

**Test 5.1**

Test addToInventory()

Test data:

* item: default item

Expected results:

* cloaks: 3

Actual results:

3

Passed

**Test 5.2**

Test collectCoins()

Test data:

* player.coins: 0
* coins: 15

Expected results:

* player.coins: 15

Actual results:

15

Passed

**Test 5.3**

Test generatePoliceBribe()

Test data:

* player.coins: 10

Expected results:

* bribe (coins): x between [5, 15]

Actual results:

5

Passed

**Test 5.4.1**

Test jumpBack() with valid values – Tested in game play

Test data:

* backJumps: 3
* room: non-default room
* direction: “North”
* room.isPolice: true or false

Expected results:

* backJumps: 2
* room display: portals are not null, valid, non-default and have been reset
* room.isPolice: true or false

Actual results:

* backJumps: 2
* room display: portals are not null, valid, non-default and have been reset
* room.isPolice: true or false

Passed

**Test 5.4.2**

Test jumpBack() with invalid values

Test data:

* backJumps: 3
* room: default room
* direction: “Southeast”
* room.isPolice: true or false

Expected results:

* backJumps: 2
* room display: portals are null
* room.isPolice: true or false

Actual results:

* backJumps: 2
* room display: portals are not null in directions North and South but null in West and East
* room.isPolice: false
* user input validations (isBlank, isValid) successful

Failed

**Test 5.5.1**

Test jumpForward() with valid values – Tested in game play

Test data:

* validDirections: {N, W, E, S}
* room: non-default room with valid non-default exits

Expected results:

* direction: North, West, East or South

Actual results:

Direction: North West East or South

Passed

**Test 5.5.2**

Test jumpForward () with invalid values

Test data:

* validDirections: {N, S}
* room: default room with null exits

Expected results:

* direction: empty string

Actual results:

Empty string

Passed

**Test 5.6**

Test removeCloak()

Test data:

* inventory: arraylist item size 3

Expected results:

* cloaks: 2

Actual results:

2

Passed