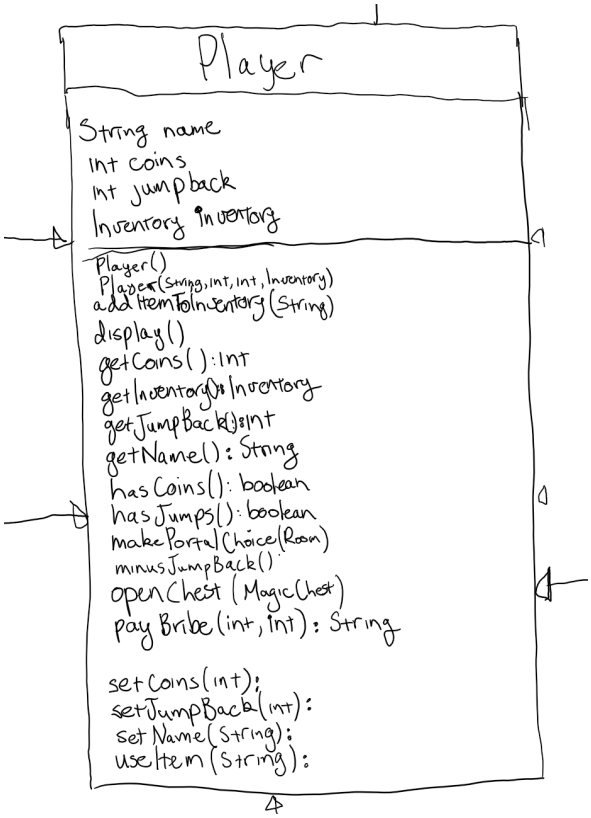

TEST PLAN

Player()

1.1Create a player object using the default constructor.

Expected:

Name: Player 1, coins: 10, jumpback: 3, inventory: empty,empty,empty

Actual:

```
public static void main(String[] args)
{
    Player player = new Player();
    player.display();
}
```

/home/Player.java 103:26 Spaces: 4 (Auto) All changes saved

Terminal Submit

```
[user@sahara ~]$ java Player
Player 1        10 coins        3 Get out of jail jumps
Inventory: 1.Empty 2.Empty 3.Empty
```

Reset

Player(name, coins, jumpback, inventory)

Positive test

2.1 Create a player object using the non-default constructor with valid inputs.

Input:

```
Item[] itemList = new Item[3];
itemList[0] = new Item("Cloak");
itemList[1] = new Item();
itemList[2] = new Item();
Inventory inv = new Inventory(itemList);
Player player = new Player("Mike",10,12, inv);
player.display();
```

Expected:

Name: Mike, coins: 10, jumpback: 12, inventory: Cloak, empty,empty

Actual:

```
[user@sahara ~]$ java Player
Mike    10 coins        12 Get out of jail jumps
Inventory: 1.Cloak 2.Empty 3.Empty
```

2.2 Create a player object using the non-default constructor with edge case inputs.

Input:

```
Item[] itemList = new Item[3];
itemList[0] = new Item("Cloak");
itemList[1] = new Item();
itemList[2] = new Item();
Inventory inv = new Inventory(itemList);
Player player = new Player("     ",0,0, inv);
player.display();
```

Expected:

Name: Player 1, coins: 0, jumpback: 0, inventory: Cloak, empty,empty

Actual:

```
[user@sahara ~]$ java Player
Player 1        0 coins 0 Get out of jail jumps
Inventory: 1.Cloak 2.Empty 3.Empty
```

Negative test

2.3 Create a player object using the non-default constructor with invalid inputs.

Input:

```
Item[] itemList = new Item[3];
itemList[0] = new Item("Cloak");
itemList[1] = new Item();
itemList[2] = new Item();
Inventory inv = new Inventory(itemList);
Player player = new Player("ok",-10,-2, inv);
player.display();
```

Expected:

Name: Player 1, coins: 0, jumpback: 3, inventory: Cloak, empty,empty

Actual:

```
[user@sahara ~]$ java Player
Player 1        0 coins 3 Get out of jail jumps
Inventory: 1.Cloak 2.Empty 3.Empty
```

addItemToInventory(String item)

Positive test

3.1 Add a cloak to a partially full inventory.

Input:

```
Item[] itemList = new Item[3];
itemList[0] = new Item("Cloak");
itemList[1] = new Item();
itemList[2] = new Item();
Inventory inv = new Inventory(itemList);
Player player = new Player("",0,0, inv);
player.addItemToInventory("Cloak");
player.display();
```

Expected:

Name: Player 1, coins: 0, jumpback: 0, inventory: Cloak, Cloak,empty

Actual:

```
[user@sahara ~]$ java Player
Player 1        0 coins 0 Get out of jail jumps
 Inventory: 1.Cloak 2.Cloak 3.Empty
```

3.2 Add a cloak to a full inventory.

Input:

```
Item[] itemList = new Item[3];
itemList[0] = new Item("Cloak");
itemList[1] = new Item("Cloak");
itemList[2] = new Item("Cloak");
Inventory inv = new Inventory(itemList);
Player player = new Player("",0,0, inv);
player.addItemToInventory("Cloak");
player.display();
```

Expected:

Name: Player 1, coins: 0, jumpback: 0, inventory: Cloak, Cloak, Cloak

Actual:

```
[user@sahara ~]$ java Player
Player 1        0 coins 0 Get out of jail jumps          Inventory: 1.Cloak 2.Cloak 3.Cloak
```

Negative test

3.3 Add an item with a blank string for type.

Input:

```
Item[] itemList = new Item[3];
itemList[0] = new Item("Cloak");
itemList[1] = new Item("Cloak");
itemList[2] = new Item();
Inventory inv = new Inventory(itemList);
Player player = new Player("",0,0, inv);
player.addItemToInventory(" ");
player.display();
```

Expected:

"Item type cannot be blank."

Actual:

```
Item type cannot be blank.
Player 1        0 coins 0 Get out of jail jumps          Inventory: 1.Cloak 2.Cloak 3.Empty
```

inputPlayerName()

Positive test

4.1 Enter a valid name

Expected
“Welcome NAME to javalice”
Actual

```
[user@sahara ~]$ java Player
        Please enter your name:
NAME
                        Welcome, NAME, to Javalice
```

4.2 Enter a valid name with leading blank space

Expected:
Strip the white space and print welcome message
Actual:

```
        Please enter your name:
    mki
                        Welcome, mki, to Javalice
```

4.3 Enter a valid name with trailing blank space

Expected:
Strip the white space and print welcome message
Actual:

```
        Please enter your name:
mki
                        Welcome, mki, to Javalice
```

Negative test

4.4 Enter a name less than three characters

Expected:
Error please try again
Actual:

```
Please enter your name:
mk
Your name must be between 3 and 12 characters
Please enter your name:
```

4.5 Enter a blank name

Expected:

Error please try again

Actual:

```
Please enter your name:

Your name must be between 3 and 12 characters
Please enter your name:
```

4.6 Enter an invalid name with leading blank space

Expected:

Error please try again

Actual:

```
Please enter your name:
m
Your name must be between 3 and 12 characters
Please enter your name:
```

4.7 Enter an invalid name with trailing blank space

Expected:

Error please try again

Actual:

```
Please enter your name:
m
Your name must be between 3 and 12 characters
```

4.8 Enter a name more than 25 characters

Expected:

Error please try again

Actual:

```
Please enter your name:
qwertyuiopasdfghjkl;zxcvbnm,.
Your name must be between 3 and 12 characters
Please enter your name:
```

makePortalChoice(Room room)

Positive test

5.1 Given a default room, pick a portal that is not open, then pick an open portal (west)

Input:

```
Player player = new Player();
Room room = new Room();
int outcome = player.makePortalChoice(room);
System.out.println(outcome);
```

Expected: (my input in red)

```
Open Portals: [list of open portals]
        Choose your portals

(character corresponding to closed portal)
Please enter an appropriate character
w
1
```

actual:

```
Open Portals: W E S
        Choose your portal
n

Please enter an appropriate character
Open Portals: W E S
        Choose your portal
w
1
```

Negative test

5.2 Given a default room, where all portals are closed.

Input:

```
Player player = new Player();
Room room = new Room();
int outcome = player.makePortalChoice(room);
System.out.println(outcome);
```

Expected: (my input in red)

Open Portals:
Choose your portals

(will accept any character)
-1

Actual:

```
        Open Portals:
                Choose your portal

-1
```

openChest(MagicChest chest)

Positive test

6.1 Choose to open a chest

Expected:
y
[String describing item in chest]

Actual:

```
[user@sahara ~]$ java Player
         Would you like to open the chest? y/n
y
Cloak
```

6.2 Choose not to open a chest

Expected:
N
You did not open the chest

Actual:

```
[user@sahara ~]$ java Player
         Would you like to open the chest? y/n
n
         You did not open the chest.
```

Negative test

6.3 Input something other than y or n

Expected:
[input]
Please select either Y or N
Would you like to open the chest? y/n

Actual:

```
[user@sahara ~]$ java Player
         Would you like to open the chest? y/n
k
Please select either Y or N
         Would you like to open the chest? y/n
```

payBribe(int coins, int bribe)

Positive test

7.1 Case where coins is greater than bribe and paying

Expected:

You must pay x coins, would you like to pay?
Y

Actual:

```
[user@sahara ~]$ java Player
        You must pay 8coins
                Would you like to pay the bribe? Y/N
y
```

Returns “y”

7.2 Case where coins is greater than bribe and not paying

Expected:
You must pay x coins, would you like to pay?
N
Well then how are you going to escape?

Actual:

```
[user@sahara ~]$ java Player
        You must pay 8coins
                Would you like to pay the bribe? Y/N
n
        Ok well, how are you going to escape?
```

Returns “y”

7.3 Case where coins is less than bribe

Expected:

You must pay x coins.
You cannot pay the bribe

Actual:

```
[user@sahara ~]$ java Player
        You must pay 12coins
        You cannot pay this bribe
```

Returns “n”

Negative test

7.4 Case where coins is negative

Expected:

Both coins and bribe must be positive

Actual

```
[user@sahara ~]$ java Player
Both bribe and coin values must be greater than 0
```

7.5 Case where bribe is negative

Expected:

Both coins and bribe must be positive

Actual:

```
[user@sahara ~]$ java Player
Both bribe and coin values must be greater than 0
```

7.6 Case where something other than y/n is entered

Expected:

Please try again

Actual:

```
[user@sahara ~]$ java Player
        You must pay 12coins
                Would you like to pay the bribe? Y/N
k
        please select either y or n
                Would you like to pay the bribe? Y/N

        please select either y or n
                Would you like to pay the bribe? Y/N
```

respondToPolice(String choices)

Positive test

8.1 Case where user selects invalid input then valid input

Expected

User will be prompted to keep choosing until a valid input is selected

Actual

```
Please choose from J,B,C

a
Please select one of J,B,C

b
```

8.2 Jail Case

Expected

Function will return “Jail”

Actual

```
Jail
```

Negative test

8.3 Case where blank choices are passed

Expected

Function will print an error

Actual

```
Invalid choices provided
```

useItem(String item)

Positive test

Case where item is in inventory

Expected:

Item specified will be removed from inventory

Actual:

```
[user@sahara ~]$ java Player
         Inventory: 1.Empty 2.Empty 3.sword
         You have used 1 sword
Player 1        10 coins        3 Get out of jail jumps          Inventory: 1.Empty 2.Empty 3.Empty
```

Case where item is not in inventory

Expected:

Do nothing

Actual:

```
[user@sahara ~]$ java Player
         Inventory: 1.Empty 2.Empty 3.Empty
Player 1        10 coins        3 Get out of jail jumps          Inventory: 1.Empty 2.Empty 3.Empty
```

Negative test

Passing a blank string

Expected:

Do nothing

Actual

```
^[[A[user@sahara ~]$ java Player
         Inventory: 1.Empty 2.Empty 3.Empty
Player 1        10 coins        3 Get out of jail jumps          Inventory: 1.Empty 2.Empty 3.Empty
```