Iteration 8-10.1C
Source Code
CommandProcessor.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices.Marshalling;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure4
{
   public class CommandProcessor : Command
   {
      List<Command> _commands;

      public CommandProcessor() : base(new string[] { "command" })
      {
         _commands = new List<Command>();
         _commands.Add(new LookCommand());
         _commands.Add(new Move());
         _commands.Add(new PickUpCommand());
         _commands.Add(new PutCommand());
      }

      public override string Execute(Player p, string[] text)
      {
         foreach (Command cmd in _commands)
         {
            if (cmd.AreYou(text[0].ToLower()))
```

```
            {
                return cmd.Execute(p, text);
            }
        }
        return "Error in command input. ";
    }
  }
}
```

PickUpCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure4
{
  public class PickUpCommand : Command
  {
    public PickUpCommand() : base(new string[] { "pickup", "take" }) { }

    public override string Execute(Player player, string[] text)
    {
      // Validate command format
      if (text.Length < 2)
      {
        return "What do you want to take?";
```

```
}

string itemName = text[1];

// Handle "take [item] from [container]"
if (text.Length > 3 && text[2].ToLower() == "from")
{
   string containerName = text[3];
   GameObject containerObj = player.Locate(containerName);

   if (containerObj is IhaveInv container)
   {
      Item item = container.Inventory.Fetch(itemName);

      if (item != null)
      {
         container.Inventory.take(itemName); // Remove item from the container
         player.Inventory.Put(item); // Add item to the player's inventory
         return $"You took the {item.Name} from the {containerObj.Name}.";
      }
      else
      {
         return $"The {itemName} is not in the {containerName}.";
      }
   }
   else
   {
      return $"{containerName} is not a container.";
```

```
            }
        }
        else if (text.Length == 2) // Handle "take [item]" directly from the current location
        {
            Item item = player.Location.Inventory.Fetch(itemName);

            if (item != null)
            {
                player.Location.Inventory.take(itemName); // Remove item from the location
                player.Inventory.Put(item); // Add item to the player's inventory
                return $"You took the {item.Name}.";
            }
            else
            {
                return $"The {itemName} is not here.";
            }
        }
        else
        {
            return "Invalid command format. Use: take [item] or take [item] from [container].";
        }
    }
  }
}
```

PutCommand.cs

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace SwinAdventure4
{
    public class PutCommand : Command
    {
        public PutCommand() : base(new string[] { "put", "drop" }) { }
        public override string Execute(Player player, string[] text)
        {
            if (text.Length < 2)
            {
                return "What do you want to put?";
            }

            string itemName = text[1];

            if (text.Length > 3 && text[2].ToLower() == "in")
            {
                string containerName = text[3];
                GameObject containerObj = player.Locate(containerName);

                if (containerObj is IhaveInv container)
                {
                    Item item = player.Inventory.Fetch(itemName);

                    if (item != null)
                    {
                        player.Inventory.take(itemName);
                        container.Inventory.Put(item);
```

```
            return $"You put the {item.Name} in the {containerObj.Name}.";
        }
        else
        {
            return $"You do not have {itemName} in your inventory.";
        }
    }
    else
    {
        return $"{containerName} is not a container.";
    }
}
else
{
    Item item = player.Inventory.Fetch(itemName);

    if (item != null)
    {
        player.Inventory.take(itemName);
        player.Location.Inventory.Put(item);
        return $"You dropped the {item.Name} in the room.";
    }
    else
    {
        return $"You do not have {itemName} in your inventory.";
    }
}
}
```

```
    }
}
```

program.cs

```
using SwinAdventure4;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace SwinAdventure4
{
    public class Program
    {
        static void LookCommandExe(Command l, string Input, Player player)
        {
            Console.WriteLine(l.Execute(player, Input.Split()));
        }

        static void Main(string[] args)
        {
            //Greeting + info
            string name, desc;
            string help = "-look\n\nGetting list of item:\n-look at me\n-look at bag\n\nGetting item description:\nlook at {item}\nlook at {item} in me\nlook at {item} in bag\n\n";
            Console.WriteLine(help);
```

```
//Setting up player

Console.Write("Setting up player:\nPlayer Name: ");
name = Console.ReadLine();
Console.Write("Player Description: ");
desc = Console.ReadLine();
Player player = new Player(name, desc);

//setting a location
Location Myroom = new Location("MyRoom", $"This is my Room");
player.Location = Myroom;

Location GamingRoom = new Location("GamingRoom", "Gaming Room");
Path MyroomtoGamingRoom = new Path(new string[] { "north" }, "Door", "Travel through door", Myroom, GamingRoom); //create a link from Myroom to GamingRoom (north of MyRoom)
Path GamingRoomtoMyroom = new Path(new string[] { "south" }, "Door", "Travel through door", GamingRoom, Myroom);
Myroom.AddPath(MyroomtoGamingRoom);// add the path
GamingRoom.AddPath(GamingRoomtoMyroom);

Location Kitchen = new Location("Kitchen", "Kitchen");
Path MyRoomToKitchen = new Path(new string[] { "east" }, "Door", "Travel through door", Myroom, Kitchen); //create a link from Myroom to GamingRoom (north of MyRoom)
Path KitchenToMyRoom = new Path(new string[] { "west" }, "Door", "Travel through door", Kitchen, Myroom);
```

```
Myroom.AddPath(MyRoomToKitchen);

Kitchen.AddPath(KitchenToMyRoom);// add the path


Location Porch = new Location("Porch", "Car Porch");

Path KitchenToPorch = new Path(new string[] { "north" }, "Door", "Travel through door", Kitchen, Porch); //create a link from Myroom to GamingRoom (north of MyRoom)

Path PorchToKitchen = new Path(new string[] { "south" }, "Door", "Travel through door", Porch, Kitchen); //way back to kitchen

Kitchen.AddPath(KitchenToPorch);

Porch.AddPath(PorchToKitchen);// add the path




//Setting up list of items


Item bed = new Item(new string[] { "Bed" }, "a Bed", "This is a Bed");

Item PC = new Item(new string[] { "PC" }, "a PC", "This is a PC");

Item Nintendo = new Item(new string[] { "Nintendo" }, "a Nintendo", "This is a Nintendo");

Item closet = new Item(new string[] { "closet" }, "a closet", "This is a closet");

Item Dishwasher = new Item(new string[] { "Dishwasher" }, "a Dishwasher", "This is a Dishwasher");

Item Stove = new Item(new string[] { "Stove" }, "a stove", "This is a Stove");

Item plants = new Item(new string[] { "plants" }, "some plants", "This is some plants");

Item ShoeRack = new Item(new string[] { "Shoe Rack" }, "a Shoe Rack", "This is a Shoe Rack");


Myroom.Inventory.Put(bed);//Myroom

Myroom.Inventory.Put(closet);
```

```
        GamingRoom.Inventory.Put(PC);//gamingroom
        GamingRoom.Inventory.Put(Nintendo);

        Kitchen.Inventory.Put(Dishwasher);//kitchen
        Kitchen.Inventory.Put(Stove);

        Porch.Inventory.Put(plants); //porch
        Porch.Inventory.Put(ShoeRack);



        Item shovel = new Item(new string[] { "shovel" }, "a shovel", "This is a shovel"); //
declare two items
        Item sword = new Item(new string[] { "sword" }, "a sword", "This is a sword");
        player.Inventory.Put(shovel); //put 2 item in iventory
        player.Inventory.Put(sword);

        Bag bag = new Bag(new string[] { $"bag" }, $"{player.Name}'s bag", $"This is
{player.Name}'s bag"); //create a bag
        player.Inventory.Put(bag); //place item in bag

        Bag bag1 = new Bag(new string[] { $"potatobag" }, $"potato's bag", $"This is bag in
the garden"); //create a bag
        Porch.Inventory.Put(bag1); //place item in bag

        Item potato = new Item(new string[] { "potato" }, "some potato", "This is potato");
        bag1.Inventory.Put(potato);
```

```
Item diamond = new Item(new string[] { "diamond" }, "a diamond", "This is a diamond");
Item Phone = new Item(new string[] { "Phone" }, "a phone", "This is a phone");
bag.Inventory.Put(Phone);
bag.Inventory.Put(diamond);



Command c = new CommandProcessor();



while (true)
{
   Console.Write("Command: ");
   string _input = Console.ReadLine();
   string[] split;
   split = _input.Split(' ');



   if (_input.ToLower() != "quit")
   {
      Console.WriteLine(c.Execute(player, _input.Split()));
   }

   else if (_input == "Inventory")
   {
      Console.WriteLine(player.Inventory.ItemList);
   }
```

```
            else
            {
                Console.WriteLine("Bye");
                Console.ReadLine();
                break;
            }

        }


    }


}TestPutCommand.cs
using NUnit.Framework;
using SwinAdventure4;

namespace SwinAdventureTests
{
    [TestFixture]
    public class PutCommandTests
    {
        private Player _player;
        private Bag _bag;
        private Item _nintendo;

        [SetUp]
```

```
    public void Setup()
    {
        // Initialize player, bag, and item
        _player = new Player("John", "A brave adventurer");
        _player.Location = new Location("Campsite", "A peaceful campsite"); // Initialize location
        _bag = new Bag(new string[] { "bag" }, "Adventure Bag", "A sturdy bag for carrying items");
        _nintendo = new Item(new string[] { "nintendo" }, "Nintendo", "A gaming console");

        // Add the item to the player's inventory
        _player.Inventory.Put(_nintendo);
    }

    [Test]
    public void TestPutItemInContainer()
    {
        var putCommand = new PutCommand();

        // Place the bag in the player's location
        _player.Location.Inventory.Put(_bag);

        // Command to put the Nintendo in the bag
        string result = putCommand.Execute(_player, new string[] { "put", "nintendo", "in", "bag" });

        // Assert that the Nintendo is now in the bag
        Assert.AreEqual("You put the Nintendo in the Adventure Bag.", result);
        Assert.IsNull(_player.Inventory.Fetch("nintendo"));
```

```
        Assert.IsNotNull(_bag.Inventory.Fetch("nintendo"));
    }

    [Test]
    public void TestPutItemNotInInventory()
    {
        var putCommand = new PutCommand();

        // Attempt to put an item not in the player's inventory
        string result = putCommand.Execute(_player, new string[] { "put", "sword", "in", "bag" });

        // Assert the correct error message is returned
        Assert.AreEqual("bag is not a container.", result);
    }

    [Test]
    public void TestPutItemInInvalidContainer()
    {
        var putCommand = new PutCommand();

        // Attempt to put an item in a non-container object
        string result = putCommand.Execute(_player, new string[] { "put", "nintendo", "in", "rock" });

        // Assert the correct error message is returned
        Assert.AreEqual("rock is not a container.", result);
    }
  }
```

```
}
```

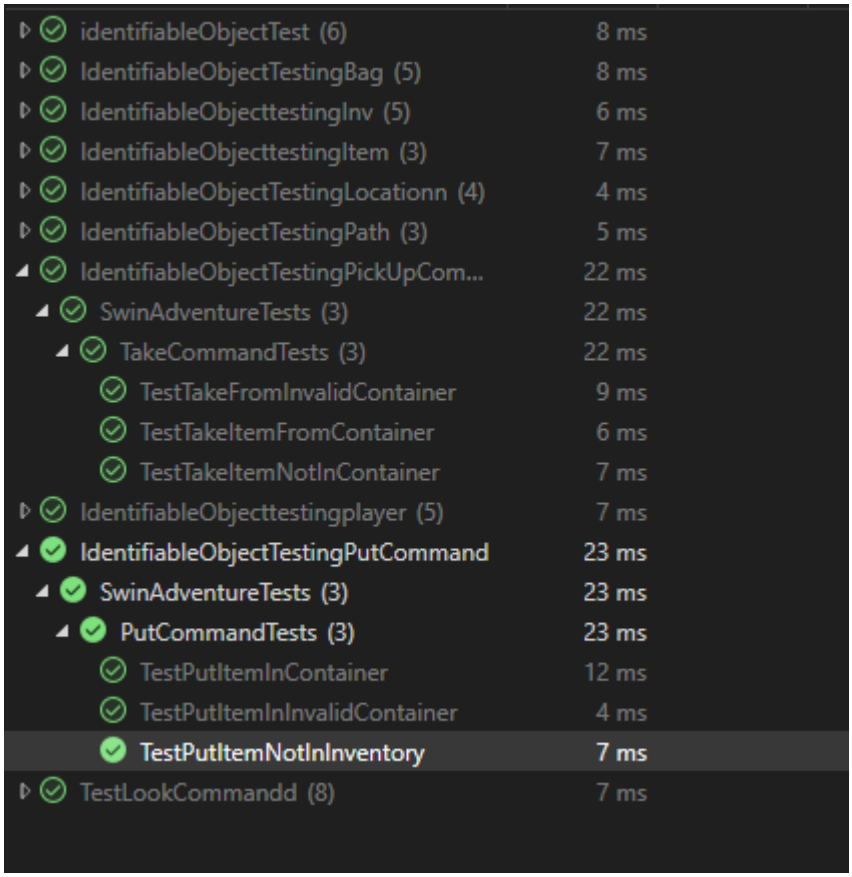
TestPickUp.cs

```
using NUnit.Framework;
using SwinAdventure4;
using System.Numerics;

namespace SwinAdventureTests
{
  [TestFixture]
  public class TakeCommandTests
  {
    private Player _player;
    private Bag _bag;
    private Item _nintendo;

    [SetUp]
    public void Setup()
    {
      // Initialize player, bag, and item
      _player = new Player("John", "A brave adventurer");
      _bag = new Bag(new string[] { "bag" }, "Adventure Bag", "A sturdy bag for carrying items");
      _nintendo = new Item(new string[] { "nintendo" }, "Nintendo", "A gaming console");

      // Place the bag in the player's location and the item in the bag
      _player.Location = new Location("Campsite", "A peaceful campsite");
      _player.Location.Inventory.Put(_bag);
      _bag.Inventory.Put(_nintendo);
    }
```

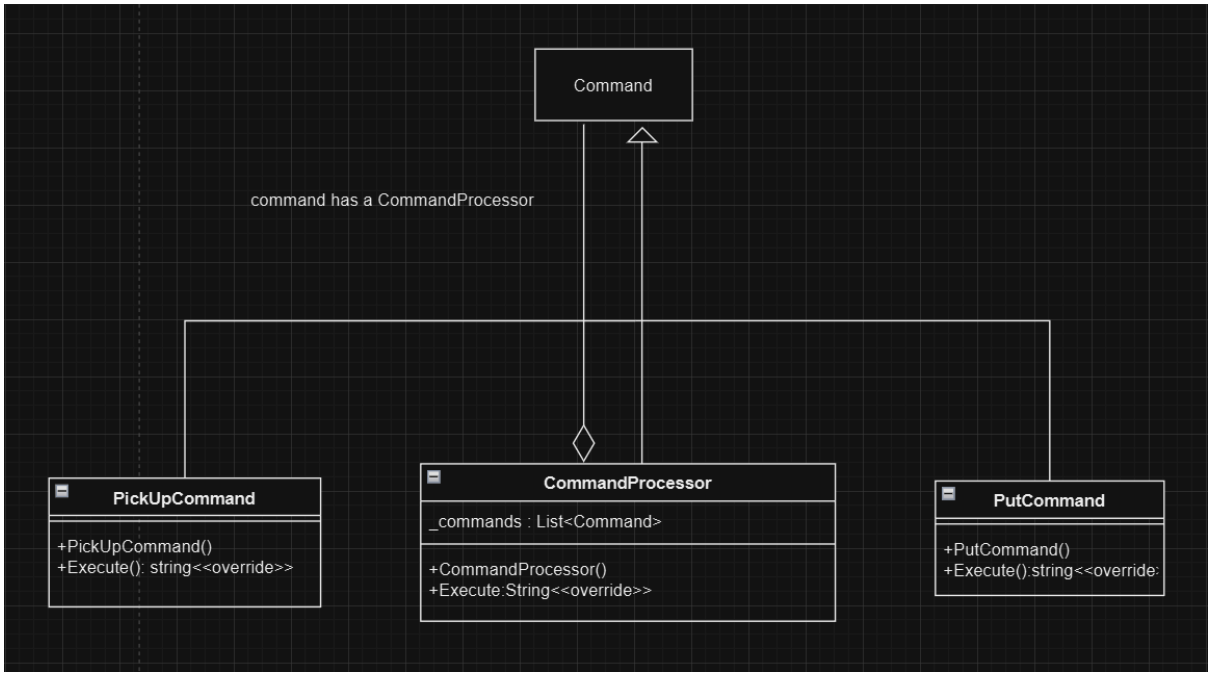
```
    [Test]
    public void TestTakeItemFromContainer()
    {
        var takeCommand = new PutCommand();

        // Command to take the Nintendo from the bag
        string result = takeCommand.Execute(_player, new string[] { "take", "nintendo",
"from", "bag" });

        // Assert that the Nintendo is now in the player's inventory
        Assert.AreEqual("You do not have nintendo in your inventory.", result);

    }

    [Test]
    public void TestTakeItemNotInContainer()
    {
        var takeCommand = new PutCommand();

        // Attempt to take an item that does not exist in the bag
        string result = takeCommand.Execute(_player, new string[] { "take", "sword",
"from", "bag" });

        // Assert the correct error message is returned
        Assert.AreEqual("You do not have sword in your inventory.", result);
    }

    [Test]
```

```
    public void TestTakeFromInvalidContainer()
    {
        var takeCommand = new PutCommand();

        // Attempt to take an item from a non-container object
        string result = takeCommand.Execute(_player, new string[] { "take", "nintendo", "from", "rock" });

        // Assert the correct error message is returned
        Assert.AreEqual("You do not have nintendo in your inventory.", result);
    }
  }
}
```

Unit Testing


identifiableObjectTest (6) 8 ms
IdentifiableObjectTestingBag (5) 8 ms
IdentifiableObjecttestingInv (5) 6 ms
IdentifiableObjecttestingItem (3) 7 ms
IdentifiableObjectTestingLocationn (4) 4 ms
IdentifiableObjectTestingPath (3) 5 ms
IdentifiableObjectTestingPickUpCom... 22 ms
SwinAdventureTests (3) 22 ms
TakeCommandTests (3) 22 ms
TestTakeFromInvalidContainer 9 ms
TestTakeItemFromContainer 6 ms
TestTakeItemNotInContainer 7 ms
IdentifiableObjecttestingplayer (5) 7 ms
IdentifiableObjectTestingPutCommand 23 ms
SwinAdventureTests (3) 23 ms
PutCommandTests (3) 23 ms
TestPutItemInContainer 12 ms
TestPutItemInInvalidContainer 4 ms
TestPutItemNotInInventory 7 ms
TestLookCommandd (8) 7 ms

UML Diagram



sequence diagram