COS20007 Object Oriented Programming

Credit Task 10.1C: Case Study — Iteration 8: Command Processor

Show Wai Yan/105293041

PickUpCommand.cs

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

        public override string Execute(Player p, string[] text)
        {
            IHaveInventory container = null;
            string containerId = null;
            string itemId = null;
            switch (text.Length)
            {
                case 2:
                    itemId = text[1].ToLower();
                    container = p.CurrentLocation;
                    break;
                case 4:
                    if (text[2].ToLower() != "from")
                        return "Where do you want to pick up from?";
                    if (text[3].ToLower() == "room")
                        container = p.CurrentLocation;
                    else
                    {
                        containerId = text[3].ToLower();
                        container = FetchContainer(p.CurrentLocation, containerId);
                    }
                    itemId = text[1].ToLower();
                    break;
                default:
                    return "I don\'t know how to pick up like this.";
            }
            if (container == null)
                return $"We cannot find the {containerId}.";

            Item itm = GetItem(itemId, container);
            if (itm == null)
                return $"There is no such {itemId} to pick up.";

            p.Inventory.Put(itm);
            return $"You have taken {itm.ShortDescription.Split("(")[0].Trim()} from the {container.Name.ToLower()}";
        }

        private IHaveInventory FetchContainer(Location l, string containerId)
        {
            return l.Locate(containerId) as IHaveInventory;
        }

        private Item GetItem(string itemId, IHaveInventory container)
        {
            Item itm = container.Inventory.Take(itemId);
            return itm;
        }
    }
}
```

PutCommand.cs

```
namespace SwinAdventure
{
    public class PutCommand : Command
    {
        public PutCommand()
            : base(new string[] { "put", "drop" }) { }

        public override string Execute(Player p, string[] text)
        {
            IHaveInventory container = null;
            string containerId = null;
            string itemId = null;
            switch (text.Length)
            {
                case 2:
                    itemId = text[1].ToLower();
                    container = p.CurrentLocation;
                    break;
                case 4:
                    if (text[2].ToLower() != "in")
                        return "Where do you want to put into?";
                    if (text[3].ToLower() == "room")
                        container = p.CurrentLocation;
                    else
                    {
                        containerId = text[3].ToLower();
                        container = FetchContainer(p.CurrentLocation, containerId);
                    }
                    itemId = text[1].ToLower();
                    break;
                default:
                    return "I don\'t know how to pick up like this.";
            }

            if (container == null)
                return $"We cannot find the {containerId}.";

            Item itm = p.Inventory.Take(itemId);
            if (itm == null)
                return $"There is no such {itemId} to put in.";

            PutItem(itm, container);

            return $"You have put {itm.ShortDescription.Split("(")[0].Trim()} in the {container.Name.ToLower()}";
        }

        private IHaveInventory FetchContainer(Location l, string containerId)
        {
            return l.Locate(containerId) as IHaveInventory;
        }

        private Item PutItem(Item itm, IHaveInventory container)
        {
            container.Inventory.Put(itm);
            return itm;
        }
    }
}
```

CommandProcessor.cs

```
namespace SwinAdventure
{
  public class CommandProcessor
  {
    private List<Command> _commands;

    public CommandProcessor()
    {
      _commands = new List<Command>();
    }

    public void AddCommand(Command command)
    {
      _commands.Add(command);
    }

    public string Execute(Player p,string input)
    {
      string[] inputArr = input.Split(' ');
      foreach (Command command in _commands)
      {
        if (command.AreYou(inputArr[0])) return command.Execute(p,inputArr);
      }
      return $"I don\'t understand {inputArr[0]}";
    }
  }
}
```

IHaveInventory.cs

```
namespace SwinAdventure
{
    public interface IHaveInventory
    {
        public GameObject Locate(string id);
        public string Name { get; }
        public Inventory Inventory { get; }
    }
}
```

LookCommand.cs

```
namespace SwinAdventure
{
    public class LookCommand : Command
    {
        public LookCommand()
            : base(new string[] { "look", "inv", "inventory" }) { }

        public override string Execute(Player p, string[] text)
        {
            IHaveInventory container = null;
            string containerId = null;
            string itemId = null;
            switch (text.Length)
            {
                case 1:
                    if (text[0].ToLower() == "look")
                        return p.CurrentLocation.FullDescription;
                    else if (text[0].ToLower() == "inventory" || text[0].ToLower() == "inv")
                    {
                        container = p;
                        itemId = "me";
                    }
                    break;
                case 3:
                    if (text[1].ToLower() != "at")
                        return "What do you want to look at?";
                    container = p;
                    itemId = text[2].ToLower();
                    break;
                case 5:
                    if (text[3].ToLower() != "in")
                        return "What do you want to look in?";
                    containerId = text[4].ToLower();
                    itemId = text[2].ToLower();
                    container = FetchContainer(p, containerId);
                    break;
                default:
                    return "I don\'t know how to look like that";
            }
            if (container == null)
                return $"I cannot find the {containerId}";
            return LookAtIn(itemId, container);
        }

        private IHaveInventory FetchContainer(Player p, string containerId)
        {
            return p.Locate(containerId) as IHaveInventory;
        }

        private string LookAtIn(string thingId, IHaveInventory container)
        {

            if (container.Locate(thingId) == null)
                return $"I cannot find the {thingId} in the {container.Name}";

            return container.Locate(thingId).FullDescription;
        }
    }
}
```

Program.cs

```
namespace SwinAdventure
{
    public class Program
    {
        public static void Main(string[] args)
        {
            // Configurations
            string helpCommand =
                $"Here is the List of command\n\t- look at me: Display what you are carrying in your inventory\n\t- look at <item>
[?in <container>]: Get description of that item,which inside in the container\n\t- look: Display location's information\n\t- move
<direction>: Player travel to that location\n\t- pickup/take <item> [?from <container>]: Pick up the item and put into your
inventory\n\t- put/drop <item> [?in <container>]: Drop the item from your inentory\n\t- quit/exit: Halt the program\n";

            // Getting Player's Name and Description
            string PlayerName = "";
            string PlayerDescription = "";
            Console.WriteLine("Write Your Name, Traveller!");
            Console.Write("NAME -> ");
            PlayerName = Console.ReadLine();
            Console.WriteLine("How about Your description, Traveller!");
            Console.Write("Description -> ");
            PlayerDescription = Console.ReadLine();

            // LOCATIONS
            Location shire = new Location(
                new[] { "shire" },
                "The Shire",
                "A peaceful land of Hobbits, green and quiet."
            );
            Location bree = new Location(
                new[] { "bree" },
                "Bree",
                "A small town with The Prancing Pony inn."
            );
            Location rivendell = new Location(
                new[] { "rivendell" },
                "Rivendell",
                "An Elven sanctuary full of ancient magic."
            );
            Location moria = new Location(
                new[] { "moria" },
                "Moria",
                "A dark underground Dwarven city, full of echo and danger."
            );
            Location mountDoom = new Location(
                new[] { "mount doom", "doom" },
                "Mount Doom",
                "A fiery mountain in the heart of Mordor."
            );
            Location escapeTunnel = new Location(
                new[] { "tunnel", "escape tunnel" },
                "Secret Escape Tunnel",
                "A hidden tunnel beneath Mount Doom, dimly lit by glowing stones."
            );

            // Location items
            // SHIRE
            shire.Inventory.Put(
                new Item(
                    new[] { "pipeweed", "pouch" },
                    "Pipeweed Pouch",
                    "A small pouch of fine pipeweed."
                )
            );
            shire.Inventory.Put(
                new Item(
                    new[] { "hat", "farmer's hat" },
                    "Farmer's Hat",
                    "A straw hat once worn by a hobbit farmer."
                )
            );
            shire.Inventory.Put(
                new Item(
                    new[] { "loaf", "bread" },
                    "Hobbit Loaf",
                    "Freshly baked bread from the Shire."
                )
            );

            // BREE
            bree.Inventory.Put(
                new Item(
                    new[] { "mug", "ale" },
                    "Mug of Ale",
                    "A frothy mug from The Prancing Pony."
                )
            );
            bree.Inventory.Put(
                new Item(
                    new[] { "dagger", "rusty dagger" },
                    "Rusty Dagger",
                    "Old and blunt, but still dangerous."
                )
            );
            bree.Inventory.Put(
                new Item(
                    new[] { "cloak", "travel cloak" },
                    "Travel Cloak",
                    "A heavy cloak for cold nights."
                )
            );

            // RIVENDELL
            rivendell.Inventory.Put(
                new Item(
                    new[] { "bread", "elven bread", "lembas" },
                    "Elven Bread",
                    "One bite is enough for a full day's journey."
                )
            );
            rivendell.Inventory.Put(
                new Item(
                    new[] { "pendant", "silver pendant" },
                    "Silver Pendant",
                    "An Elven trinket that shimmers faintly."
                )
            );
            rivendell.Inventory.Put(
                new Item(
                    new[] { "book", "ancient book" },
                    "Ancient Book",
                    "Filled with forgotten lore and legends."
                )
            );

            // MORIA
            moria.Inventory.Put(
                new Item(
                    new[] { "pickaxe", "broken pickaxe" },
                    "Broken Pickaxe",
                    "Snapped at the handle."
                )
            );
            moria.Inventory.Put(new Item(new[] { "torch" }, "Torch", "Still usable if relit."));
            moria.Inventory.Put(
                new Item(
                    new[] { "gauntlets", "dwarven gauntlets" },
                    "Dwarven Gauntlets",
                    "Heavy gloves forged in the mountains."
                )
            );

            // MOUNT DOOM
            mountDoom.Inventory.Put(
                new Item(
                    new[] { "ring shard", "shard" },
                    "Black Ring Shard",
                    "A broken piece of something ancient and cursed."
                )
            );
            mountDoom.Inventory.Put(
                new Item(new[] { "lava", "rock" }, "Lava Rock", "Still warm to the touch.")
            );
            mountDoom.Inventory.Put(
                new Item(
                    new[] { "journal", "burned journal" },
                    "Burned Journal",
                    "Most pages are unreadable, but a few notes remain."
                )
            );

            // ESCAPE TUNNEL
            escapeTunnel.Inventory.Put(
                new Item(
                    new[] { "silk", "spider silk" },
                    "Spider Silk",
                    "Sticky and unnaturally strong."
                )
            );
            escapeTunnel.Inventory.Put(
                new Item(
                    new[] { "crystal", "shard" },
                    "Crystal Shard",
                    "Glows faintly with magical energy."
                )
            );
            escapeTunnel.Inventory.Put(
                new Item(
                    new[] { "torch", "elven torch" },
                    "Elven Torch",
                    "Lights automatically in the darkness."
                )
            );

            // PATHS (Bidirectional and One-Way)

            // Shire ↔ Bree
            Path shireToBree = new Path(
                new[] { "east", "e" },
                "east",
                "A path to Bree, lined with fields.",
                bree
            );
            Path breeToShire = new Path(
                new[] { "west", "w" },
                "west",
                "A path back to the Shire.",
                shire
            );

            // Bree ↔ Rivendell
            Path breeToRivendell = new Path(
                new[] { "north", "n" },
                "north",
                "The path to Rivendell through forested slopes.",
                rivendell
            );
            Path rivendellToBree = new Path(
                new[] { "south", "s" },
                "south",
                "A path back down to Bree.",
                bree
            );

            // Shire ↔ Rivendell (shortcut)
            Path shireToRivendell = new Path(
                new[] { "northeast", "ne" },
                "northeast",
                "An old Elven path to Rivendell.",
                rivendell
            );
            Path rivendellToShire = new Path(
                new[] { "southwest", "sw" },
                "southwest",
                "A trail through hills back to the Shire.",
                shire
            );

            // Bree ↔ Moria
            Path breeToMoria = new Path(
                new[] { "east", "e" },
                "east",
                "The eastern road to the mines of Moria.",
                moria
            );
            Path moriaToBree = new Path(
                new[] { "west", "w" },
                "west",
                "A narrow road back to Bree.",
                bree
            );

            // Moria → Mount Doom (one-way)
```

```
Path moriaToDoom = new Path(
    new[] { "south", "s" },
    "south",
    "A dark, narrow path leads to Mount Doom.",
    mountDoom
);

// Mount Doom → Escape Tunnel (one-way)
Path doomToTunnel = new Path(
    new[] { "down", "d" },
    "Escape Tunnel",
    "A rocky slope leads to a hidden escape tunnel.",
    escapeTunnel
);

// Escape Tunnel → Moria (return path)
Path tunnelToMoria = new Path(
    new[] { "up", "u" },
    "Moria",
    "You follow the tunnel upward back into Moria's depths.",
    moria
);

// ADD PATHS TO LOCATIONS

shire.AddPath(shireToBree);
shire.AddPath(shireToRivendell);

bree.AddPath(breeToShire);
bree.AddPath(breeToRivendell);
bree.AddPath(breeToMoria);

rivendell.AddPath(rivendellToBree);
rivendell.AddPath(rivendellToShire);

moria.AddPath(moriaToBree);
moria.AddPath(moriaToDoom); // No return from Doom to Moria

mountDoom.AddPath(doomToTunnel); // No path back to Moria

escapeTunnel.AddPath(tunnelToMoria); // Secret return

// Player
Player me = new Player(PlayerName, PlayerDescription, shire);

// Player Items
Item sword = new Item(
    new[] { "sword", "steel sword" },
    "Steel Sword",
    "A well-balanced sword of polished steel."
);
Item shield = new Item(
    new[] { "shield", "leather shield" },
    "Leather Shield",
    "A round shield made of hardened leather."
);
Bag starterBag = new Bag(
    new[] { "bag", "satchel" },
    "Adventurer's Bag",
    "A worn leather bag with room for essentials."
);

// Items inside the bag
Item healingPotion = new Item(
    new[] { "potion", "healing potion" },
    "Healing Potion",
    "Restores health when consumed."
);
Item mapFragment = new Item(
    new[] { "map", "fragment" },
    "Map Fragment",
    "A torn piece of an ancient map leading somewhere..."
);

// Add items to bag
starterBag.Inventory.Put(healingPotion);
starterBag.Inventory.Put(mapFragment);

// Add everything to player
me.Inventory.Put(sword);
me.Inventory.Put(shield);
me.Inventory.Put(starterBag);

// Command Configuration
CommandProcessor commandProcessor = new CommandProcessor();
commandProcessor.AddCommand(new LookCommand());
commandProcessor.AddCommand(new MoveCommand());
commandProcessor.AddCommand(new PickUpCommand());
commandProcessor.AddCommand(new PutCommand());

// Game Loop
Console.WriteLine("Write '-h' for helper");

Console.WriteLine(me.Arrive());
while (true)
{
    string command = "";
    Console.Write("Command -> ");
    command = Console.ReadLine().ToLower();
    Console.WriteLine(); // to make clear after input line for presented looking

    if (command == "exit" || command == "quit")
    {
        Console.WriteLine("Take the rest, Traveller!");
```

```
                    return;
                }
                else if (command == "-h")
                {
                    Console.WriteLine(helpCommand);
                }
                else
                    Console.WriteLine(commandProcessor.Execute(me, command));
            }
        }
    }
}
```

TestPickUpCommand.cs

```
using NUnit.Framework;
using NUnit.Framework.Legacy;
using SwinAdventure;

namespace UnitTests
{
    [TestFixture]
    public class TestPickUpCommand
    {
        private PickUpCommand pickUp;
        private Player player;
        private Location l1;
        private Bag bag;
        private Item gem = new Item(new string[] { "gem" }, "Gem", "This is a gem");
        private Item shovel = new Item(new string[] { "shovel" }, "Shovel", "This is a shovel");
        private Item diamond = new Item(new string[] { "diamond" }, "Diamond", "This is a diamond");

        [SetUp]
        public void Setup()
        {
            pickUp = new PickUpCommand();

            l1 = new Location(
                new string[] { "a small tent", "tent" },
                "Small Tent",
                "This is a resting place for travelers."
            );

            player = new Player("Show", "The Programmer", l1);

            bag = new Bag(
                new string[] { "bag", "backpack" },
                "Leather Bag",
                "A sturdy leather bag to carry items"
            );

            bag.Inventory.Put(gem);
            l1.Inventory.Put(bag);
            l1.Inventory.Put(shovel);
        }

        [Test]
        public void TestPickUpFromCurrentLocation()
        {
            string excepted = "You have taken a shovel from the small tent";
            string result = pickUp.Execute(player, new string[] { "pickup", "shovel" });
            ClassicAssert.False(l1.Inventory.HasItem("shovel"));
            ClassicAssert.True(player.Inventory.HasItem("shovel"));
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestPickUpNothingFromCurrentLocation()
        {
            string excepted = "There is no such sword to pick up.";
            string result = pickUp.Execute(player, new string[] { "pickup", "sword" });
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestPickUpFromNothing()
        {
            string excepted = "We cannot find the box.";
            string result = pickUp.Execute(
                player,
                new string[] { "pickup", "shovel", "from", "box" }
            );
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestPickUpSomethingFromSomething()
        {
            string excepted = "You have taken a gem from the leather bag";
            string result = pickUp.Execute(player, new string[] { "pickup", "gem", "from", "bag" });
            ClassicAssert.False(bag.Inventory.HasItem("gem"));
            ClassicAssert.True(player.Inventory.HasItem("gem"));
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestFromValidation()
        {
            string excepted = "Where do you want to pick up from?";
            ClassicAssert.That(
                pickUp.Execute(player, new string[] { "pickup", "paper", "in", "room" }),
                Is.EqualTo(excepted)
            );
        }
    }
}
```

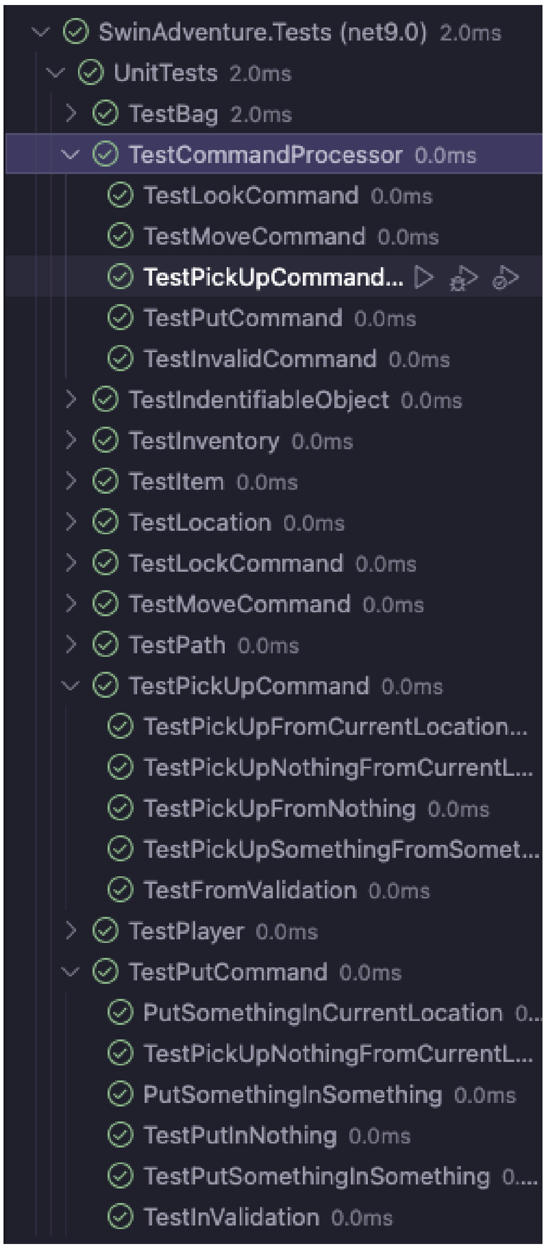
TestPutCommand.cs

```
using NUnit.Framework;
using NUnit.Framework.Legacy;
using SwinAdventure;

namespace UnitTests
{
    [TestFixture]
    public class TestPutCommand
    {
        private PutCommand put;
        private Player player;
        private Location l1;
        private Bag bag;
        private Item gem = new Item(new string[] { "gem" }, "Gem", "This is a gem");
        private Item shovel = new Item(new string[] { "shovel" }, "Shovel", "This is a shovel");
        private Item diamond = new Item(new string[] { "diamond" }, "Diamond", "This is a diamond");

        [SetUp]
        public void Setup()
        {
            put = new PutCommand();

            l1 = new Location(
                new string[] { "a small tent", "tent" },
                "Small Tent",
                "This is a resting place for travelers."
            );

            player = new Player("Show", "The Programmer", l1);

            bag = new Bag(
                new string[] { "bag", "backpack" },
                "Leather Bag",
                "A sturdy leather bag to carry items"
            );

            bag.Inventory.Put(gem);
            player.Inventory.Put(diamond);
            l1.Inventory.Put(bag);
            l1.Inventory.Put(shovel);
        }

        [Test]
        public void PutSomethingInCurrentLocation()
        {
            string excepted = "You have put a diamond in the small tent";
            string result = put.Execute(player, new string[] { "put", "diamond" });
            ClassicAssert.True(l1.Inventory.HasItem("diamond"));
            ClassicAssert.False(player.Inventory.HasItem("diamond"));
            ClassicAssert.That(result,Is.EqualTo(excepted));
        }

        [Test]
        public void TestPickUpNothingFromCurrentLocation()
        {
            string excepted = "There is no such sword to put in.";
            string result = put.Execute(player, new string[] { "put", "sword" });
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void PutSomethingInSomething()
        {
            string excepted = "You have put a diamond in the leather bag";
            string result = put.Execute(player, new string[] { "put", "diamond", "in", "bag" });
            ClassicAssert.False(l1.Inventory.HasItem("diamond"));
            ClassicAssert.True(bag.Inventory.HasItem("diamond"));
            ClassicAssert.False(player.Inventory.HasItem("diamond"));
            ClassicAssert.That(result,Is.EqualTo(excepted));
        }

        [Test]
        public void TestPutInNothing()
        {
            string excepted = "We cannot find the box.";
            string result = put.Execute(
                player,
                new string[] { "put", "diamond", "in", "box" }
            );
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }


        [Test]
        public void TestPutSomethingInSomething()
        {
            string excepted = "You have put a diamond in the leather bag";
            string result = put.Execute(player, new string[] { "put", "diamond", "in", "bag" });
            ClassicAssert.True(bag.Inventory.HasItem("diamond"));
            ClassicAssert.False(player.Inventory.HasItem("diamond"));
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }
        [Test]
        public void TestInValidation()
        {
            string excepted = "Where do you want to put into?";
            ClassicAssert.That(
                put.Execute(player, new string[] { "put", "paper", "from", "room" }),
                Is.EqualTo(excepted)
            );
        }
    }
}
```

TestCommandProcessor.cs

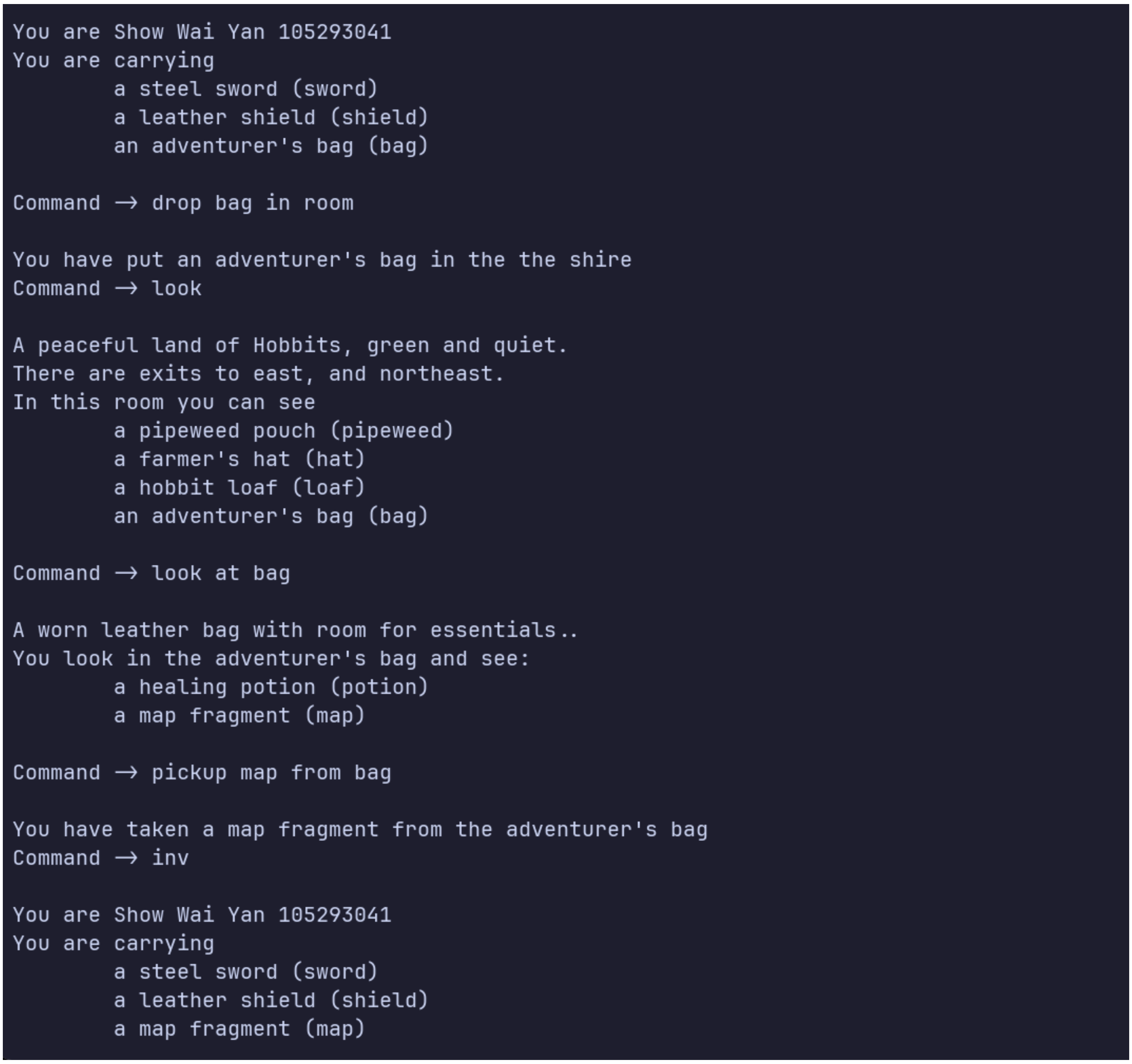
```
using NUnit.Framework;
using NUnit.Framework.Legacy;
using SwinAdventure;

namespace UnitTests
{
    public class TestCommandProcessor
    {
        private CommandProcessor cp;
        private LookCommand look;
        private MoveCommand move;
        private PickUpCommand pickup;
        private PutCommand put;
        private Player player;
        private Location l1;
        private Location l2;
        private SwinAdventure.Path p1;
        private Bag bag;
        private Item gem;
        private Item shovel;
        private Item diamond;

        [SetUp]
        public void Setup()
        {
            cp = new CommandProcessor();
            look = new LookCommand();
            move = new MoveCommand();
            pickup = new PickUpCommand();
            put = new PutCommand();

            cp.AddCommand(look);
            cp.AddCommand(move);
            cp.AddCommand(pickup);
            cp.AddCommand(put);

            l1 = new Location(
                new string[] { "a small tent", "tent" },
                "Small Tent",
                "This is a resting place for travelers."
            );
            l2 = new Location(
                new string[] { "a dark cave", "cave" },
                "Dark Cave",
                "A damp, echoing cave stretches into the darkness."
            );
            p1 = new SwinAdventure.Path(
                new string[] { "north", "n" },
                "forest",
                "You are entering a dense forest from the north.",
                l2
            );
            l1.AddPath(p1);

            player = new Player("Show", "The Programmer", l1);

            bag = new Bag(
                new string[] { "bag", "backpack" },
                "Leather Bag",
                "A sturdy leather bag to carry items"
            );

            gem = new Item(new string[] { "gem" }, "Gem", "This is a gem");
            shovel = new Item(new string[] { "shovel" }, "Shovel", "This is a shovel");
            diamond = new Item(new string[] { "diamond" }, "Diamond", "This is a diamond");

            player.Inventory.Put(diamond);
            l1.Inventory.Put(bag);
            l1.Inventory.Put(shovel);
            bag.Inventory.Put(gem);
        }

        [Test]
        public void TestLookCommand()
        {
            string excepted = gem.FullDescription;
            string result = cp.Execute(player, "look at gem in bag");
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestMoveCommand()
        {
            string excepted =
                $"{player.Exit(p1.FirstId)}\n{player.Travel(p1)}\nYou have arrived in {l2.ShortDescription}";
            ClassicAssert.That(cp.Execute(player, "move north"), Is.EqualTo(excepted));
        }

        [Test]
        public void TestPickUpCommand()
        {
            string excepted = "You have taken a shovel from the small tent";
            string result = cp.Execute(player, "pickup shovel");
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestPutCommand()
        {
            string excepted = "You have put a diamond in the small tent";
            string result = cp.Execute(player, "put diamond");
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }

        [Test]
        public void TestInvalidCommand()
        {
            string excepted = "I don\'t understand sleep";
            string result = cp.Execute(player, "sleep");
            ClassicAssert.That(result, Is.EqualTo(excepted));
        }
    }
}
```

Screenshot of unit test passing



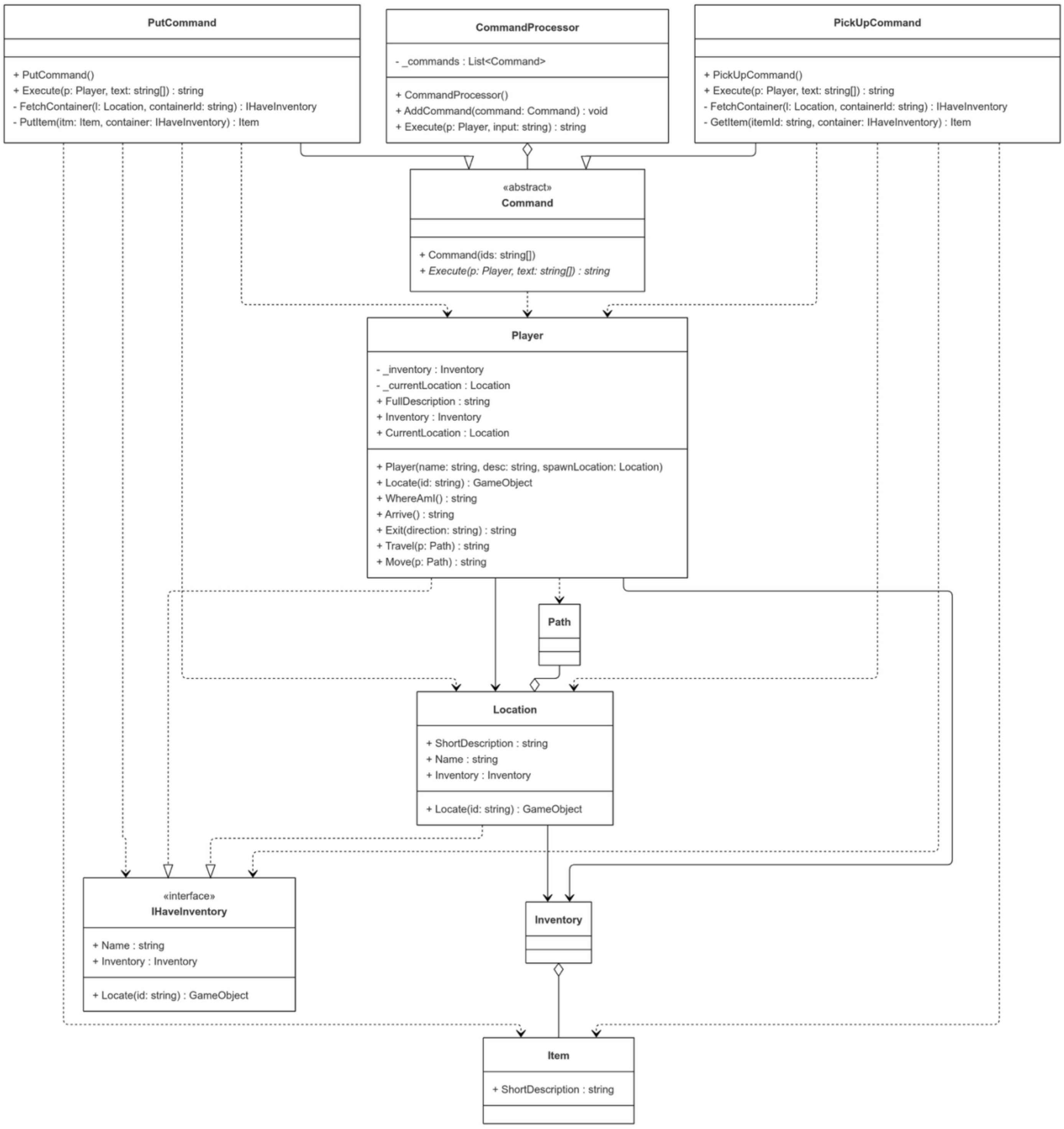
```
SwinAdventure.Tests (net9.0) 2.0ms
  UnitTests 2.0ms
    TestBag 2.0ms
    TestCommandProcessor 0.0ms
      TestLookCommand 0.0ms
      TestMoveCommand 0.0ms
      TestPickUpCommand...
      TestPutCommand 0.0ms
      TestInvalidCommand 0.0ms
    TestIndentifiableObject 0.0ms
    TestInventory 0.0ms
    TestItem 0.0ms
    TestLocation 0.0ms
    TestLockCommand 0.0ms
    TestMoveCommand 0.0ms
    TestPath 0.0ms
    TestPickUpCommand 0.0ms
      TestPickUpFromCurrentLocation...
      TestPickUpNothingFromCurrentL...
      TestPickUpFromNothing 0.0ms
      TestPickUpSomethingFromSomet...
      TestFromValidation 0.0ms
    TestPlayer 0.0ms
    TestPutCommand 0.0ms
      PutSomethingInCurrentLocation 0...
      TestPickUpNothingFromCurrentL...
      PutSomethingInSomething 0.0ms
      TestPutInNothing 0.0ms
      TestPutSomethingInSomething 0....
      TestInValidation 0.0ms
```

Screenshot of program running showing new commands related to locations

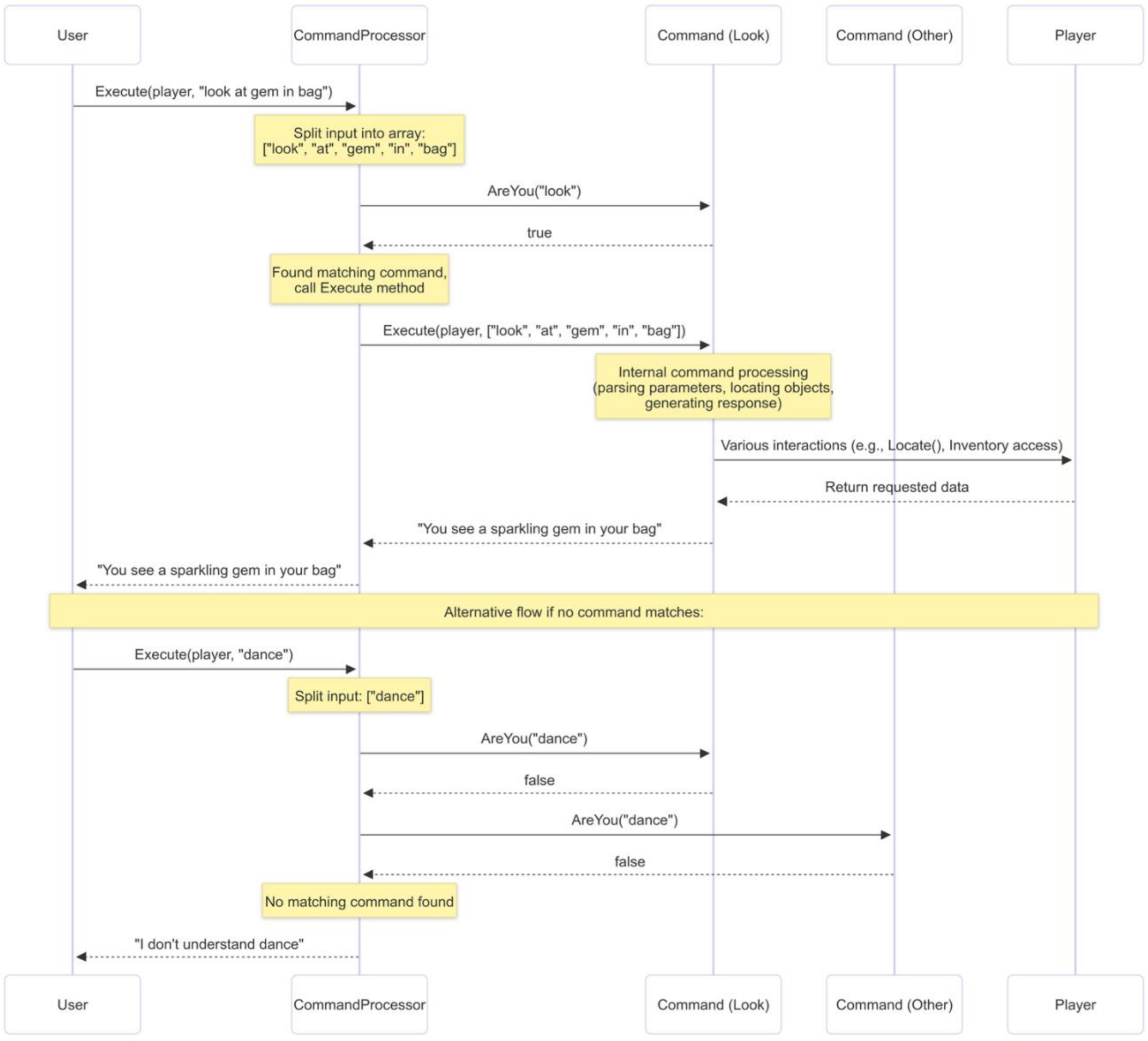


```
You are Show Wai Yan 105293041
You are carrying
        a steel sword (sword)
        a leather shield (shield)
        an adventurer's bag (bag)

Command → drop bag in room

You have put an adventurer's bag in the the shire
Command → look

A peaceful land of Hobbits, green and quiet.
There are exits to east, and northeast.
In this room you can see
        a pipeweed pouch (pipeweed)
        a farmer's hat (hat)
        a hobbit loaf (loaf)
        an adventurer's bag (bag)

Command → look at bag

A worn leather bag with room for essentials..
You look in the adventurer's bag and see:
        a healing potion (potion)
        a map fragment (map)

Command → pickup map from bag

You have taken a map fragment from the adventurer's bag
Command → inv

You are Show Wai Yan 105293041
You are carrying
        a steel sword (sword)
        a leather shield (shield)
        a map fragment (map)
```

UML Class diagram showing what needs to be added


PutCommand
+ PutCommand()
+ Execute(p: Player, text: string[]) : string
- FetchContainer(l: Location, containerId: string) : IHaveInventory
- PutItem(itm: Item, container: IHaveInventory) : Item
CommandProcessor
- _commands : List<Command>
+ CommandProcessor()
+ AddCommand(command: Command) : void
+ Execute(p: Player, input: string) : string
PickUpCommand
+ PickUpCommand()
+ Execute(p: Player, text: string[]) : string
- FetchContainer(l: Location, containerId: string) : IHaveInventory
- GetItem(itemId: string, container: IHaveInventory) : Item
«abstract»
Command
+ Command(ids: string[])
+ Execute(p: Player, text: string[]) : string
Player
- _inventory : Inventory
- _currentLocation : Location
+ FullDescription : string
+ Inventory : Inventory
+ CurrentLocation : Location
+ Player(name: string, desc: string, spawnLocation: Location)
+ Locate(id: string) : GameObject
+ WhereAmI() : string
+ Arrive() : string
+ Exit(direction: string) : string
+ Travel(p: Path) : string
+ Move(p: Path) : string
Path
Location
+ ShortDescription : string
+ Name : string
+ Inventory : Inventory
+ Locate(id: string) : GameObject
«interface»
IHaveInventory
+ Name : string
+ Inventory : Inventory
+ Locate(id: string) : GameObject
Inventory
Item
+ ShortDescription : string

UML sequence diagram to explore how executing a command works


User
CommandProcessor
Command (Look)
Command (Other)
Player
Execute(player, "look at gem in bag")
Split input into array: ["look", "at", "gem", "in", "bag"]
AreYou("look")
true
Found matching command, call Execute method
Execute(player, ["look", "at", "gem", "in", "bag"])
Internal command processing (parsing parameters, locating objects, generating response)
Various interactions (e.g., Locate(), Inventory access)
Return requested data
"You see a sparkling gem in your bag"
"You see a sparkling gem in your bag"
Alternative flow if no command matches:
Execute(player, "dance")
Split input: ["dance"]
AreYou("dance")
false
AreYou("dance")
false
No matching command found
"I don't understand dance"
User
CommandProcessor
Command (Look)
Command (Other)
Player