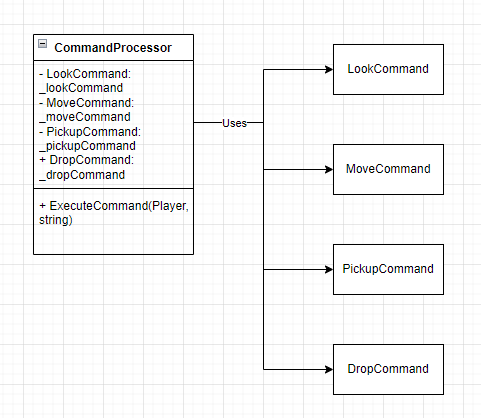
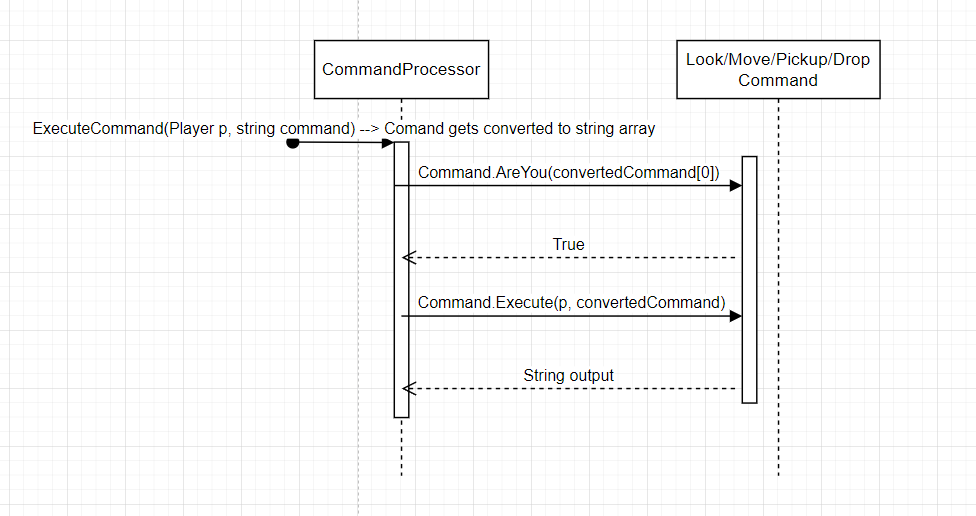
Iteration 8: Command Processor

UML



Sequence Diagram



Unit Test results

A screenshot of a computer

Description automatically generated

Test file

|  |
| --- |
| using SwinAdventure;  using Path = SwinAdventure.Path;  namespace TestQueue  {  public class Tests  {  Item item1 = new Item(new string[] { "sword" }, "sword", "a sword");  Item item2 = new Item(new string[] { "shield" }, "shield", "a shield");  Item item3 = new Item(new string[] { "shiba" }, "shiba", "a shiba");  Item item4 = new Item(new string[] { "gem" }, "gem", "a gem");  [SetUp]  public void Setup()  {  }  // Test the Item class  [Test]  public void ItemIdentifiable()  {  Assert.IsTrue(item1.AreYou("sword"));  }  [Test]  public void ShortDescription()  {  Assert.That(item1.ShortDescription, Is.EqualTo("a sword (sword)"));  }  [Test]  public void FullDescription()  {  Assert.That(item1.FullDescription, Is.EqualTo("a sword"));  }  // Test the Inventory class  [Test]  public void FindItem()  {  Inventory inventory = new Inventory();  inventory.Put(item1);  Assert.IsTrue(inventory.HasItem("sword"));  }  [Test]  public void NoItem()  {  Inventory inventory = new Inventory();  Assert.IsFalse(inventory.HasItem("sword"));  }  [Test]  public void FetchItem()  {  Inventory inventory = new Inventory();  inventory.Put(item1);  Assert.That(item1, Is.EqualTo(inventory.Fetch("sword")));  Assert.IsTrue(inventory.HasItem("sword"));  }  [Test]  public void TakeItem()  {  Inventory inventory = new Inventory();  inventory.Put(item1);  Assert.That(item1, Is.EqualTo(inventory.Take("sword")));  Assert.IsFalse(inventory.HasItem("sword"));  }  [Test]  public void ItemList()  {  Inventory inventory = new Inventory();  inventory.Put(item1);  inventory.Put(item2);  //the list string below is the expected output, consisting of every item in the following format: name ( first id)  Assert.That(inventory.ItemList, Is.EqualTo("\t a sword (sword)\n\t a shield (shield)\n"));  }  // Test the Player class  [Test]  public void PlayerIdentifiable()  {  Player player = new Player("Tan", "A player");  Assert.IsTrue(player.AreYou("me"));  Assert.IsTrue(player.AreYou("inventory"));  }  [Test]  public void PlayerLocate()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  Assert.That(item1, Is.EqualTo(player.Locate("sword")));  }  [Test]  public void PlayerLocateItself()  {  Player player = new Player("Tan", "A player");  Assert.That(player, Is.EqualTo(player.Locate("me")));  Assert.That(player, Is.EqualTo(player.Locate("inventory")));  }  [Test]  public void PlayerLocateNothing()  {  Player player = new Player("Tan", "A player");  Assert.That(player.Locate("sword"), Is.Null);  }  [Test]  public void PlayerFullDescription()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  player.Inventory.Put(item2);  //the list string below is the expected output, consisting of every item in the following format: name ( first id)  Assert.That(player.FullDescription, Is.EqualTo("You are Tan A player\nYou are carrying:\n\t a sword (sword)\n\t a shield (shield)\n"));  }  //Test the Bag class  [Test]  public void BagLocate()  {  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  backpack.Inventory.Put(item1);  backpack.Inventory.Put(item2);  backpack.Inventory.Put(item3);  //ask to return item and item stays in backpack  Assert.That(item3, Is.EqualTo(backpack.Locate("shiba")));  Assert.IsTrue(backpack.Inventory.HasItem("shiba"));  }  [Test]  public void BagLocatesItself()  {  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  Assert.That(backpack, Is.EqualTo(backpack.Locate("backpack")));  }  [Test]  public void BagLocateNothing()  {  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  Assert.That(backpack.Locate("sword"), Is.Null);  }  [Test]  public void BagFullDescription()  {  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "A backpack");  backpack.Inventory.Put(item1);  backpack.Inventory.Put(item2);  backpack.Inventory.Put(item3);  //the list string below is the expected output, consisting of every item in the following format: name ( first id)  Assert.That(backpack.FullDescription, Is.EqualTo("A backpack\nYou look in the backpack and see:\n\t a sword (sword)\n\t a shield (shield)\n\t a shiba (shiba)\n"));  }  [Test]  public void BagInBag()  {  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  Bag satchel = new Bag(new string[] { "satchel" }, "satchel", "a satchel");  backpack.Inventory.Put(satchel);  Assert.That(satchel, Is.EqualTo(backpack.Locate("satchel")));  }  //Test for the LookCommand class  [Test]  public void LookAtMe()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  player.Inventory.Put(item2);  LookCommand LookCommand = new LookCommand();  string expectedDescription = "You are Tan A player\nYou are carrying:\n\t a sword (sword)\n\t a shield (shield)\n";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "me" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookAtGem()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item4);  LookCommand LookCommand = new LookCommand();  string expectedDescription = "a gem";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "gem" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookAtUnk()  {  Player player = new Player("Tan", "A player");  LookCommand LookCommand = new LookCommand();  string expectedDescription = "I can't find the gem in the Tan";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "gem" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookAtGemInBag()  {  Player player = new Player("Tan", "A player");  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  player.Inventory.Put(backpack);  backpack.Inventory.Put(item4);  LookCommand LookCommand = new LookCommand();  string expectedDescription = "a gem";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "gem", "in", "backpack" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookAtBag()  {  Player player = new Player("Tan", "A player");  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "A backpack");  backpack.Inventory.Put(item1);  backpack.Inventory.Put(item2);  player.Inventory.Put(backpack);  LookCommand LookCommand = new LookCommand();  string expectedDescription = "A backpack\nYou look in the backpack and see:\n\t a sword (sword)\n\t a shield (shield)\n";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "backpack" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookAtGemInNoBag()  {  Player player = new Player("Tan", "A player");  LookCommand LookCommand = new LookCommand();  string expectedDescription = "I can't find the backpack";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "gem", "in", "backpack" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookAtNoGemInBag()  {  Player player = new Player("Tan", "A player");  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  player.Inventory.Put(backpack);  LookCommand LookCommand = new LookCommand();  string expectedDescription = "I can't find the gem in the backpack";  string testDescription = LookCommand.Execute(player, new string[] { "look", "at", "gem", "in", "backpack" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void InvalidLookCommand()  {  Player player = new Player("Tan", "A player");  LookCommand LookCommand = new LookCommand();  string expectedDescription = "I don't know how to look like that";  //only 2 arguments  string testDescription = LookCommand.Execute(player, new string[] { "look", "at" });  Assert.That(testDescription, Is.EqualTo(expectedDescription));  //4 arguments  string testDescription2 = LookCommand.Execute(player, new string[] { "look", "at", "gem", "in" });  Assert.That(testDescription2, Is.EqualTo(expectedDescription));  //5 arguments but the 4th argument is not "in"  string testDescription3 = LookCommand.Execute(player, new string[] { "look", "at", "a", "at", "b" });  string expectedDescription2 = "What do you want to look in?";  Assert.That(testDescription3, Is.EqualTo(expectedDescription2));  //5 arguments but the 2nd argument is not "at"  string testDescription4 = LookCommand.Execute(player, new string[] { "look", "in", "a", "in", "b" });  string expectedDescription3 = "What do you want to look at?";  Assert.That(testDescription4, Is.EqualTo(expectedDescription3));  }  //Test for Location  [Test]  public void LookInPlayerLocationForItem()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item1);  LookCommand LookCommand = new LookCommand();  string textDescription = LookCommand.Execute(player, new string[] { "look", "at", "sword" });  string expectedDescription = "a sword";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookInPlayerLocationForBag()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Bag backpack = new Bag(new string[] { "backpack" }, "backpack", "a backpack");  backpack.Inventory.Put(item1);  player.Location.Inventory.Put(backpack);  LookCommand LookCommand = new LookCommand();  string textDescription = LookCommand.Execute(player, new string[] { "look", "at", "sword", "in", "backpack" });  string expectedDescription = "a sword";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LookInPlayerLocationForPlayerLocationWhichHasItem()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item1);  LookCommand LookCommand = new LookCommand();  string textDescription = LookCommand.Execute(player, new string[] { "look", "at", "sword", "in", "Garden" });  string expectedDescription = "a sword";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void LocationIdentifyItself()  {  Location location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Assert.IsTrue(location.AreYou("Garden"));  }  [Test]  public void LocateItemInPlayerLocation()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item1);  Assert.That(item1, Is.EqualTo(player.Location.Locate("sword")));  }  //Test for MoveCommand  [Test]  public void MoveToLocation()  {  Player player = new Player("Tan", "A player");  Location Garden = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Location Forest = new Location(new string[] { "Forest" }, "Forest", "A forest filled with trees");  player.Location = Garden;  Path path = new Path(Direction.North, "north", "You go through a door", Forest);  player.Location.AddPath(path);  MoveCommand moveCommand = new MoveCommand();  string textDescription = moveCommand.Execute(player, new string[] { "move", "north" });  string expectedDescription = "You head North\nYou go through a door\nYou have arrived in a small Forest";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void GetPathFromLocation()  {  Location Garden = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Location Forest = new Location(new string[] { "Forest" }, "Forest", "A forest filled with trees");  Path path = new Path(Direction.NorthEast, "northeast", "You go through a door", Forest);  Garden.AddPath(path);  Assert.That(path, Is.EqualTo(Garden.GetPath(Direction.NorthEast)));  }  [Test]  public void PathMovePlayer()  {  Player player = new Player("Tan", "A player");  Location Garden = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Location Forest = new Location(new string[] { "Forest" }, "Forest", "A forest filled with trees");  player.Location = Garden;  Path path = new Path(Direction.West, "west", "You go through a door", Forest);    path.Move(player);  Assert.That(player.Location, Is.EqualTo(Forest));  }  [Test]  public void PlayerLocationDontChange()  {  Player player = new Player("Tan", "A player");  Location Garden = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Location Forest = new Location(new string[] { "Forest" }, "Forest", "A forest filled with trees");  player.Location = Garden;  Path path = new Path(Direction.North, "north", "You go through a door", Forest);  player.Location.AddPath(path);  MoveCommand moveCommand = new MoveCommand();  string textDescription = moveCommand.Execute(player, new string[] { "move", "south" });  Assert.That(player.Location, Is.EqualTo(Garden));  }  //Test for PickupCommand  [Test]  public void ErrorCommand1()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item1);  PickupCommand pickupCommand = new PickupCommand();  string textDescription = pickupCommand.Execute(player, new string[] { "pickup", "sword", "in" });  string expectedDescription = "I don't know how to pickup like that";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  public void ErrorCommand2()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item1);  PickupCommand pickupCommand = new PickupCommand();  string textDescription = pickupCommand.Execute(player, new string[] { "equip", "sword" });  string expectedDescription = "Error in pickup input";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void PickupItemFromPlayerLocation()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item3);  PickupCommand pickupCommand = new PickupCommand();  string textDescription = pickupCommand.Execute(player, new string[] { "pickup", "shiba" });  string expectedDescription = "You have picked up the shiba";    //Check string output and if item is in player inventory  Assert.That(textDescription, Is.EqualTo(expectedDescription));  Assert.That(item3, Is.EqualTo(player.Inventory.Fetch("shiba")));  }  [Test]  public void PickupItemFromBagInPlayerLocation()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Bag chest = new Bag(new string[] { "chest" }, "chest", "a chest");  chest.Inventory.Put(item3);  player.Location.Inventory.Put(chest);  PickupCommand pickupCommand = new PickupCommand();  string textDescription = pickupCommand.Execute(player, new string[] { "pickup", "shiba", "from", "chest" });  string expectedDescription = "You have picked up the shiba from the chest";  //Check string output and if item is in player inventory  Assert.That(textDescription, Is.EqualTo(expectedDescription));  Assert.That(item3, Is.EqualTo(player.Inventory.Fetch("shiba")));  }  //Test for Command Processor  [Test]  public void ExecuteLookCommand()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  player.Inventory.Put(item2);  CommandProcessor commandProcessor = new CommandProcessor();  string textDescription = commandProcessor.ExecuteCommand(player, "look at me");  string expectedDescription = "You are Tan A player\nYou are carrying:\n\t a sword (sword)\n\t a shield (shield)\n";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void ExecuteMoveCommand()  {  Player player = new Player("Tan", "A player");  Location Garden = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  Location Forest = new Location(new string[] { "Forest" }, "Forest", "A forest filled with trees");  player.Location = Garden;  Path path = new Path(Direction.North, "north", "You go through a door", Forest);  player.Location.AddPath(path);  CommandProcessor commandProcessor = new CommandProcessor();  string textDescription = commandProcessor.ExecuteCommand(player, "move north");  string expectedDescription = "You head North\nYou go through a door\nYou have arrived in a small Forest";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void ExecutePickupCommand()  {  Player player = new Player("Tan", "A player");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(item3);  CommandProcessor commandProcessor = new CommandProcessor();  string textDescription = commandProcessor.ExecuteCommand(player, "pickup shiba");  string expectedDescription = "You have picked up the shiba";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  [Test]  public void ExecuteDropCommand()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  CommandProcessor commandProcessor = new CommandProcessor();  string textDescription = commandProcessor.ExecuteCommand(player, "drop sword");  string expectedDescription = "You have dropped the sword";  Assert.That(textDescription, Is.EqualTo(expectedDescription));  }  //Test for DropCommand  [Test]  public void DropItemFromPlayerInventoryInPlayerLocation()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  DropCommand dropCommand = new DropCommand();  string textDescription = dropCommand.Execute(player, new string[] { "drop", "sword" });  string expectedDescription = "You have dropped the sword";  //Check string output, if item is in player inventory  Assert.That(textDescription, Is.EqualTo(expectedDescription));  Assert.IsFalse(player.Inventory.HasItem("sword"));  Assert.IsTrue(player.Location.Inventory.HasItem("sword"));  }  [Test]  public void DropItemInBagInLocation()  {  Player player = new Player("Tan", "A player");  player.Inventory.Put(item1);  Bag chest = new Bag(new string[] { "chest" }, "chest", "a chest");  player.Location = new Location(new string[] { "Garden" }, "Garden", "A garden filled with butterflies");  player.Location.Inventory.Put(chest);  DropCommand dropCommand = new DropCommand();  string textDescription = dropCommand.Execute(player, new string[] { "put", "sword", "in", "chest" });  string expectedDescription = "You have dropped the sword in the chest";  //Check string output and if item is in player inventory  Assert.That(textDescription, Is.EqualTo(expectedDescription));  Assert.IsFalse(player.Inventory.HasItem("sword"));  Assert.IsTrue(chest.Inventory.HasItem("sword"));  }  }  } |

Command Processor

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class CommandProcessor  {  LookCommand \_lookCommand = new LookCommand();  MoveCommand \_moveCommand = new MoveCommand();  PickupCommand \_pickupCommand = new PickupCommand();  DropCommand \_dropCommand = new DropCommand();  public CommandProcessor()  {  }  public string ExecuteCommand(Player p, string command)  {  // Trim trailing spaces from the command  command = command.TrimEnd();  // Split the command into an array of words contained within the command  string[] convertedCommand = command.Split(' ');  if (\_lookCommand.AreYou(convertedCommand[0]))  {  return \_lookCommand.Execute(p, convertedCommand);  }  else if (\_moveCommand.AreYou(convertedCommand[0]))  {  return \_moveCommand.Execute(p, convertedCommand);  }  else if (\_pickupCommand.AreYou(convertedCommand[0]))  {  return \_pickupCommand.Execute(p, convertedCommand);  }  else if (\_dropCommand.AreYou(convertedCommand[0]))  {  return \_dropCommand.Execute(p, convertedCommand);  }  else  {  return "I don't know how to " + convertedCommand[0];  }  }  }  } |

IHaveInventory

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public interface IHaveInventory  {  GameObject? Locate(string id);  string Name { get; }  Inventory Inventory { get; }  }  } |

Program

|  |
| --- |
| namespace SwinAdventure  {  internal class Program  {  static string GetNonEmptyInput(string prompt)  {  string? input;  do  {  Console.WriteLine(prompt);  input = Console.ReadLine();  // Trim the input to remove any leading or trailing white spaces  input = input?.TrimEnd();  if (string.IsNullOrWhiteSpace(input))  {  Console.WriteLine("Input cannot be empty. Please enter a valid value.");  }  } while (string.IsNullOrWhiteSpace(input));  return input;  }  static void Main(string[] args)  {  //SET UP PLAYER  string playerName = GetNonEmptyInput("Enter the player's name:");  string playerDescription = GetNonEmptyInput("Enter the player's description:");  Player player = new Player(playerName, playerDescription);  //SET UP EACH LOCATIONS    //Home - Starting Location of the player  Location home = new Location(new string[] { "home" }, "Home", "Your cozy home.");  player.Location = home;    Item shiba = new Item(new string[] { "shiba", "dog" }, "Shiba", "Your cute companion");  Item nitendo = new Item(new string[] { "switch", "nitendo" }, "Nitendo Switch", "A gaming console");  home.Inventory.Put(shiba);  home.Inventory.Put(nitendo);    //Park - North of Home  Location park = new Location(new string[] { "park" }, "Park", "A beautiful park");  home.AddPath(new Path(Direction.North, "North", "A path to the park", park));  park.AddPath(new Path(Direction.South, "South", "A path to home", home));  Item pinkPlant = new Item(new string[] { "plant", "pink" }, "Pink Plant", "A pink plant that seems to be poisonous");  Item shovel = new Item(new string[] { "shovel" }, "Shovel", "A rusted shovel");  park.Inventory.Put(pinkPlant);  park.Inventory.Put(shovel);    //Cave - East of Park  Location dungeon = new Location(new string[] { "dungeon" }, "Dungeon", "A dark and scary dungeon");  park.AddPath(new Path(Direction.East, "East", "A path to the dungeon", dungeon));  dungeon.AddPath(new Path(Direction.West, "West", "A path to the park", park));  Item sword = new Item(new string[] { "sword" }, "Sword", "A shiny sword");  Item staff = new Item(new string[] { "staff", "stick" }, "Staff", "A wooden (magical?) staff");  Bag chest = new Bag(new string[] { "chest" }, "Chest", "A wooden chest");  chest.Inventory.Put(sword);  chest.Inventory.Put(staff);  dungeon.Inventory.Put(chest);    //PROGRAM LOOP  while (true)  {  Console.WriteLine("Enter a command:");  string? command = Console.ReadLine();    CommandProcessor commandProcessor = new CommandProcessor();  if (string.IsNullOrWhiteSpace(command))  {  Console.WriteLine("Please enter a command");  }  else if (command.ToLower() == "exit")  {  break;  }  else if (command.ToLower() == "inv" || command.ToLower() == "inventory")  {  Console.WriteLine("\n" + player.FullDescription + "\n");  }  else  {  Console.WriteLine("\n" + commandProcessor.ExecuteCommand(player, command) + "\n");  }  }  }  }  } |

DropCommand

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class DropCommand : Command  {  public DropCommand() : base(new string[] { "put", "drop" })  {  }  public override string Execute(Player p, string[] text)  {  if (text.Length != 2 && text.Length != 4)  {  return "I don't know how to drop like that";  }  else if (text[0].ToLower() != "drop" && text[0].ToLower() != "put")  {  return "Error in drop input";  }  else if (text.Length == 2)  {  Item? thing = DropIn(text[1], p);  if (thing == null)  {  return "I can't find the " + text[1];  }  else  {  p.Inventory.Take((Item)thing);  p.Location.Inventory.Put(thing);  return "You have dropped the " + thing.Name;  }  }  else if (text.Length == 4)  {  if (text[2].ToLower() != "in")  {  return "Where do you want to drop this item?";  }  else if (text[3].ToLower() == "room")  {  Item? thing = DropIn(text[1], p);  if (thing == null)  {  return "I can't find the " + text[1] + "to drop";  }  else  {  p.Inventory.Take((Item)thing);  p.Location.Inventory.Put(thing);  return "You have dropped the " + thing.Name + " in the room";    }  }  else  {  IHaveInventory? container = FetchContainer(p, text[3]) as IHaveInventory;  // Check if container is null after the cast  if (container == null)  {  return "I can't find the " + text[4];  }  else  {  Item? thing = DropIn(text[1], p);  if (thing == null)  {  return "I can't find the " + text[1] + "to drop";  }  else  {  p.Inventory.Take((Item)thing);  container.Inventory.Put(thing);  return "You have dropped the " + thing.Name + " in the " + container.Name;  }  }  }  }  //default return  return "I don't know how to drop like that";  }  public IHaveInventory? FetchContainer(Player p, string containerId)  {  if (p.Locate(containerId) != null)  {  return p.Locate(containerId) as IHaveInventory;  }  else  {  return null;  }  }  public Item? DropIn(string thingId, IHaveInventory containerId)  {  Item? thing = (Item?)containerId.Locate(thingId);  if (thing == null)  {  return null;  }  else  {  return thing;  }  }  }  } |

PickupCommand

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class PickupCommand : Command  {  public PickupCommand() : base(new string[] { "pickup", "take" })  {  }  public override string Execute(Player p, string[] text)  {  if (text.Length != 2 && text.Length != 4)  {  return "I don't know how to pickup like that";  }  else if (text[0].ToLower() != "pickup" && text[0].ToLower() != "take")  {  return "Error in pickup input";  }  else if (text.Length == 2)  {  Item? thing = PickUpIn(text[1], p);  if (thing == null)  {  return "I can't find the " + text[1];  }  else if (p.Inventory.HasItem(thing.Name))  {  return "You already have the " + thing.Name;  } else  {  p.Inventory.Put((Item)thing);  p.Location.Inventory.Take((Item)thing);  return "You have picked up the " + thing.Name;  }  }  else if (text.Length == 4)  {  if (text[2].ToLower() != "from")  {  return "Where is this item?";  } else if (text[3].ToLower() == "room")  {  IHaveInventory room = p.Location;  Item? thing = PickUpIn(text[1], room);  if (thing == null)  {  return "I can't find the " + text[1] + " in the room";  }  else if (p.Inventory.HasItem(thing.Name))  {  return "You already have the " + thing.Name;  }  else  {  p.Inventory.Put((Item)thing);  p.Location.Inventory.Take((Item)thing);  return "You have picked up the " + thing.Name + " from the room";  }  } else  {  IHaveInventory? container = FetchContainer(p, text[3]) as IHaveInventory;  // Check if container is null after the cast  if (container == null)  {  return "I can't find the " + text[4];  }  else  {  // Look at the thing in the container  Item? thing = PickUpIn(text[1], container);  if (thing == null)  {  return "I can't find the " + text[1] + " in the " + container.Name;  }  else if (p.Inventory.HasItem(thing.Name))  {  return "You already have the " + thing.Name;  }  else  {  p.Inventory.Put((Item)thing);  container.Inventory.Take((Item)thing);  return "You have picked up the " + thing.Name + " from the " + container.Name;  }  }  }  }  //default return  return "I don't know how to pickup like that";  }  public IHaveInventory? FetchContainer(Player p, string containerId)  {  if (p.Locate(containerId) != null)  {  return p.Locate(containerId) as IHaveInventory;  }  else  {  return null;  }  }  public Item? PickUpIn(string thingId, IHaveInventory containerId)  {  Item? thing = (Item?)containerId.Locate(thingId);  if (thing == null)  {  return null;  }  else  {  return thing;  }  }  }  } |

MoveCommand

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class MoveCommand : Command  {  public MoveCommand() : base(new string[] { "move" , "go", "head", "leave" })  {  }  public override string Execute(Player p, string[] text)  {  if (text.Length != 2)  {  return "I don't know how to move like that";  }  else if (text[0].ToLower() != "move" && text[0].ToLower() != "go" && text[0].ToLower() != "head" && text[0].ToLower() != "leave")  {  return "Error in move input";  }  else if (text[1].ToLower() != "north" && text[1].ToLower() != "south" && text[1].ToLower() != "east" && text[1].ToLower() != "west")  {  return "I don't know how to move like that";  }  else  {  if (text[1].ToLower() == "north" || text[1].ToLower() == "n")  {  return p.Move(Direction.North);  }  else if (text[1].ToLower() == "south" || text[1].ToLower() == "s")  {  return p.Move(Direction.South);  }  else if (text[1].ToLower() == "east" || text[1].ToLower() == "e")  {  return p.Move(Direction.East);  }  else if (text[1].ToLower() == "west" || text[1].ToLower() == "w")  {  return p.Move(Direction.West);  } else if (text[1].ToLower() == "northeast" || text[1].ToLower() == "ne")  {  return p.Move(Direction.NorthEast);  } else if (text[1].ToLower() == "northwest" || text[1].ToLower() == "nw")  {  return p.Move(Direction.NorthWest);  } else if (text[1].ToLower() == "southeast" || text[1].ToLower() == "se")  {  return p.Move(Direction.SouthEast);  } else if (text[1].ToLower() == "southwest" || text[1].ToLower() == "sw")  {  return p.Move(Direction.SouthWest);  } else if (text[1].ToLower() == "up")  {  return p.Move(Direction.Up);  } else if (text[1].ToLower() == "down")  {  return p.Move(Direction.Down);  } else  {  return "I don't know how to move like that";  }  }  }  }  } |

LookCommand

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class LookCommand : Command  {  public LookCommand() : base(new string[] { "look" })  {  }  public override string Execute(Player p, string[] text)  {  if (text.Length == 1 && text[0].ToLower() == "look")  {  return p.Location.FullDescription;  }  if (text.Length != 3 && text.Length != 5)  {  return "I don't know how to look like that";  }  else if (text[0].ToLower() != "look")  {  return "Error in look input";  }  else if (text[1].ToLower() != "at")  {  return "What do you want to look at?";  }  else if (text.Length == 3)  {  return LookAtIn(text[2], p);  }  else if (text.Length == 5)  {  if (text[3].ToLower() != "in")  {  return "What do you want to look in?";  }  else  {  IHaveInventory? container = FetchContainer(p, text[4]) as IHaveInventory;  // Check if container is null after the cast  if (container == null)  {  return "I can't find the " + text[4];  }  else  {  // Look at the thing in the container  return LookAtIn(text[2], container);  }  }  }  //default return  return "I don't know how to look like that";  }  public IHaveInventory? FetchContainer(Player p, string containerId)  {  if (p.Locate(containerId) != null)  {  return p.Locate(containerId) as IHaveInventory;  }  else  {  return null;  }  }  public string LookAtIn(string thingId, IHaveInventory containerId)  {  GameObject? thing = containerId.Locate(thingId);  if (thing == null)  {  return "I can't find the " + thingId + " in the " + containerId.Name;  }  else  {  return thing.FullDescription;  }  }  }  } |

Player

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class Player : GameObject, IHaveInventory  {  private Inventory \_inventory = new Inventory();  private Location \_location;  public Player(string name, string desc) : base(new string[] { "me", "inventory" }, name, desc)  {  }  public string Move (Direction direction)  {  Path? path = Location.GetPath(direction);  if (path == null)  {  return "There is no path in that direction";  }  else  {  Location = path.DestinationLocation;  return "You head " + path.PathDirection.ToString() + "\n" + path.FullDescription + "\nYou have arrived in a small " + path.DestinationLocation.Name;  }  }  public Location Location  {  get  {  return \_location;  }  set  {  \_location = value;  }  }  public Inventory Inventory  {  get  {  return \_inventory;  }  }  public GameObject? Locate(string id)  {  if (AreYou(id))  {  return this;  }  else if (Inventory.HasItem(id))  {  return Inventory.Fetch(id);  }  else if (Location != null)  {  return Location.Locate(id);  }  else  {  return null;  }  }  public override string FullDescription  {  get  {  return "You are " + Name + " " + Description + "\nYou are carrying:\n" + \_inventory.ItemList;  }  }  GameObject? IHaveInventory.Locate(string id)  {  return Locate(id);  }  string IHaveInventory.Name  {  get  {  return Name;  }  }  Inventory IHaveInventory.Inventory  {  get  {  return Inventory;  }  }  }  } |

Bag

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using static SwinAdventure.LookCommand;  using static SwinAdventure.PickupCommand;  namespace SwinAdventure  {  public class Bag : Item, IHaveInventory  {  private Inventory \_inventory = new Inventory();  public Bag(string[] idents, string name, string desc) : base(idents, name, desc)  {  }  public Item? Locate(string id)  {  if (AreYou(id))  {  return this;  }  else  return \_inventory.Fetch(id);  }  public override string FullDescription  {  get  {  if (Inventory.ItemList == "")  {  return Description + "The " + Name + " is empty.";  }  else  return Description + "\nYou look in the " + Name + " and see:\n" + \_inventory.ItemList;  }  }  public Inventory Inventory  {  get  {  return \_inventory;  }  }  GameObject? IHaveInventory.Locate(string id)  {  return Locate(id);  }  string IHaveInventory.Name  {  get  {  return Name;  }  }  Inventory IHaveInventory.Inventory  {  get  {  return Inventory;  }  }  }  } |

Location

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Xml.Linq;  using static SwinAdventure.LookCommand;  namespace SwinAdventure  {  public class Location : GameObject, IHaveInventory  {  private Inventory \_inventory = new Inventory();  private List<Path> \_paths = new List<Path>();  public Location(string[] ids, string name, string description) : base(ids, name, description)  {  }    public List<Path> Paths  {  get  {  return \_paths;  }  }  public void AddPath(Path path)  {  \_paths.Add(path);  }  public Path? GetPath (Direction Direction)  {  foreach (Path path in Paths)  {  if (path.AreYou(Direction.ToString()))  {  return path;  }  }  return null;  }  public Inventory Inventory  {  get  {  return \_inventory;  }  }  public GameObject? Locate(string id)  {  if (AreYou(id))  {  return this;  }  return \_inventory.Fetch(id);  }  public string GetExits()  {  if (\_paths.Count == 0)  {  return "There are no exits.";  }  StringBuilder exits = new StringBuilder();  foreach (Path path in \_paths)  {  exits.Append(path.PathDirection.ToString() + ", ");  }  // Remove the trailing ", "  if (exits.Length > 2)  {  exits.Length -= 2;  }  return "There are exits to the " + exits.ToString() + ".";  }  public override string FullDescription  {  get  {  return "You are in a small " + Name + "\n" + Description + "\n" + GetExits() + "\n\n" + "In this room you can see:\n" + Inventory.ItemList;  }  }  GameObject? IHaveInventory.Locate(string id)  {  return Locate(id);  }  string IHaveInventory.Name  {  get  {  return Name;  }  }  Inventory IHaveInventory.Inventory  {  get  {  return Inventory;  }  }  }  } |