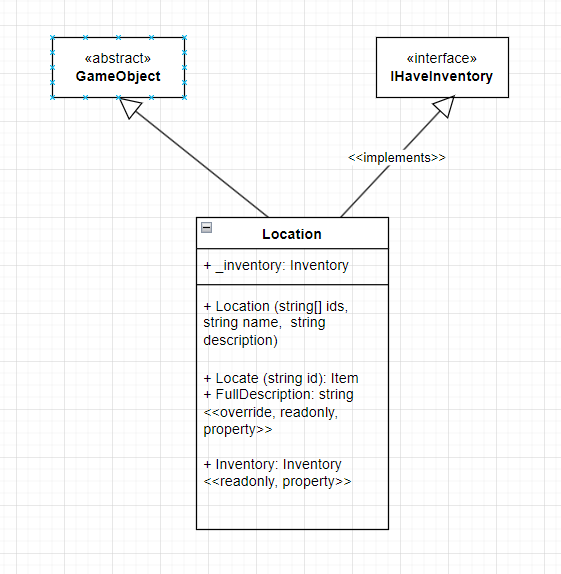
7.2: Location Implementation

UML



Player Locate Sequence Diagram

A diagram of a company

Description automatically generated

Tests Passing

A screenshot of a computer

Description automatically generated

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();  public Location(string[] ids, string name, string description) : base(ids, name, description)  {  }  public Inventory Inventory  {  get  {  return \_inventory;  }  }  public GameObject? Locate(string id)  {  if (AreYou(id))  {  return this;  }  return \_inventory.Fetch(id);  }  public override string FullDescription  {  get  {  return "You are in a small" + Name + "\n" + Description + "\n" + "In this room you can see:\n" + Inventory.ItemList;  }  }  GameObject? IHaveInventory.Locate(string id)  {  return Locate(id);  }  string IHaveInventory.Name  {  get  {  return Name;  }  }  }  } |

Test file

|  |
| --- |
| using SwinAdventure;  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")));  }  }  } |

Look Command

|  |
| --- |
| 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 != 3 && text.Length != 5)  {  return "I don't know how to look like that";  }  else if (text[0] != "look")  {  return "Error in look input";  }  else if (text[1] != "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] != "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;  }  }  public interface IHaveInventory  {  GameObject? Locate(string id);  string Name { get; }  }  }  } |

Command

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public abstract class Command : IdentifiableObject  {  public Command(string[] ids) : base(ids)  {  }  public abstract string Execute(Player p, string[] text);    }  } |

Program

|  |
| --- |
| namespace SwinAdventure  {  internal class Program  {  static void Main(string[] args)  {  Console.WriteLine("Enter the player's name:");  string? playerName = Console.ReadLine();  Console.WriteLine("Enter the player's description:");  string? playerDescription = Console.ReadLine();  Player player = new Player(playerName, playerDescription);    Item shiba = new Item(new string[] { "shiba", "dog" }, "Shiba", "A cute shiba inu");  Item nitendo = new Item(new string[] { "switch", "nitendo" }, "Nitendo Switch", "A gaming console");  player.Inventory.Put(shiba);  player.Inventory.Put(nitendo);  Bag container = new Bag(new string[] { "bag", "container" }, "Bag", "A metal container");  player.Inventory.Put(container);  Item staff = new Item(new string[] { "staff", "stick" }, "Staff", "A wooden (magical?) staff");  Item glasses = new Item(new string[] { "glasses", "spectacles" }, "Glasses", "A pair of glasses");  container.Inventory.Put(staff);  container.Inventory.Put(glasses);  while (true)  {  Console.WriteLine("Enter a command:");  string? command = Console.ReadLine();  // Split the command into an array of words contained within the command  string[] convertedCommand = command.Split(' ');  LookCommand lookCommand = new LookCommand();  Console.WriteLine(lookCommand.Execute(player, convertedCommand));  }  }  }  } |

Bag

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using static SwinAdventure.LookCommand;  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

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class Inventory  {  private List<Item> \_items = new List<Item>();  public Inventory()  {  }  public string ItemList  {  get  {  string list = "";  foreach (Item item in \_items)  {  list += "\t " + item.ShortDescription + "\n";  }  return list;  }  }  public bool HasItem(string id)  {  foreach (Item item in \_items)  {  if (item.AreYou(id))  {  return true;  }  }  return false;  }  public void Put(Item itm)  {  \_items.Add(itm);  }  public Item? Take(string id)  {  foreach (Item item in \_items)  {  if (item.AreYou(id))  {  \_items.Remove(item);  return item;  }  }  return null;  }  public Item? Fetch(string id)  {  foreach (Item item in \_items)  {  if (item.AreYou(id))  {  return item;  }  }  return null;  }  }  } |

GameObject

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public class GameObject : IdentifiableObject  {  private string \_name;  private string \_description;  public GameObject(string[] idents, string name, string desc) : base(idents)  {  \_name = name;  \_description = desc;  }  public string Name  {  get  {  return \_name;  }  }  public string Description  {  get  {  return \_description;  }  }  public string ShortDescription  {  get  {  return Description + " (" + FirstId + ")";  }  }  public virtual string FullDescription  {  get  {  return Description;  }  }  }  } |

Player

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using static SwinAdventure.LookCommand;  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 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;  }  }  }  } |

Item

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure;  public class Item : GameObject  {  public Item(string[] idents, string name, string desc) : base(idents, name, desc)  {  }    } |

Identifiable Object

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SwinAdventure  {  public abstract class IdentifiableObject  {  private List<string> \_identifiers = new List<string>();  public IdentifiableObject(string[] idents)  {  foreach (string id in idents)  {  AddIdentifier(id.ToLower());  }  }  public bool AreYou(string id)  {  return \_identifiers.Contains(id.ToLower());  }  public string FirstId  {  get  {  if (\_identifiers.Count > 0)  {  return \_identifiers[0];  }  else  {  return "";  }  }  }  public void AddIdentifier(string id)  {  \_identifiers.Add(id.ToLower());  }  }  } |