COS20007: Object Oriented Programming

Credit Task 7.2: Case Study — Iteration 6: Locations

Show Wai Yan/105293041

# Location.cs

namespace SwinAdventure

{

public class Location : GameObject, IHaveInventory

{

// Fields

private readonly Inventory \_inventory;

private readonly string \_arrivalJourney; // to indicate how player enter this location

private Dictionary<string, Location> \_exists; // to indicate 10 directions

// Constructor

public Location(string[] ids, string name, string description, string arrivalJourney)

: base(ids.Concat(new string[] { "location", "place" }).ToArray(), name, description)

{

\_inventory = new Inventory();

\_arrivalJourney = arrivalJourney;

}

// Properties

public Inventory Inventory

{

get { return \_inventory; } // Readonly properties

}

public string ArrivalJourney

{

get { return \_arrivalJourney; }

}

public override string ShortDescription

{

// need to make base's properties as virtual to make specific for location

get { return FirstId; }

}

public override string FullDescription

{

get

{

return $"{base.FullDescription}\n{FindExists()}\nIn this room you can see\n{Inventory.ItemList}";

}

}

public Dictionary<string, Location> Exists

{

get { return \_exists; }

set { \_exists = value; }

}

// Methods

public GameObject Locate(string id)

{

if (AreYou(id))

return this;

return \_inventory.Fetch(id);

}

public string FindExists()

{

// Currently placeholder fortesting

// will be implemented in itertaion 7

return "There are exits to the south.";

}

}

}

# Player.cs

namespace SwinAdventure

{

public class Player : GameObject, IHaveInventory

{

// Field

private Inventory \_inventory = new Inventory();

private Location \_currentLocation;

// Constructor

public Player(string name, string desc, Location spawnLocatoin)

: base(new string[] { "me", "inventory" }, name, desc)

{

\_currentLocation = spawnLocatoin;

}

// Properties

public override string FullDescription

{

get

{

return $"You are {Name} {base.FullDescription}\nYou are carrying\n {Inventory.ItemList}";

}

}

public Inventory Inventory

{

get { return \_inventory; }

}

public Location CurrentLocation {

get {return \_currentLocation;}

}

// Methods

public GameObject Locate(string id)

{

if (AreYou(id))

return this;

GameObject obj = Inventory.Fetch(id);

if (obj != null)

return obj;

obj = CurrentLocation.Locate(id);

if (obj != null)

return obj;

return null;

}

public string WhereAmI()

{

return $"You are in {CurrentLocation.ShortDescription}";

}

public string Arrive()

{

return $"You have arrived in {CurrentLocation.ShortDescription}";

}

// For Iteration 7

// public string Exit()

// {

// return "";

// }

// public string Travel()

// {

// return "";

// }

}

}

# LookCommand.cs

namespace SwinAdventure

{

public class LookCommand : Command

{

public LookCommand()

: base(new string[] { "look","inventory","inv" }) { }

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";

}

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 == null)

return "I cannot find the bag";

if (container.Locate(thingId) == null)

return $"I cannot find the {thingId} in the {container.Name}";

return container.Locate(thingId).FullDescription;

}

}

}

# GameObject.cs

namespace SwinAdventure

{

public abstract class GameObject : IndentifiableObject

{

// Fields

private string \_description;

private string \_name;

// Constructor

public GameObject(string[] ids, string name, string desc) : base(ids)

{

\_name = name;

\_description = desc;

}

// Properties

public string Name

{

get { return \_name; }

}

public virtual string ShortDescription

{

get

{

char firstChar = char.ToLower(Name[0]);

string article = (firstChar == 'a' || firstChar == 'e' || firstChar == 'i' ||

firstChar == 'o' || firstChar == 'u') ? "an" : "a";

return $"{article} {Name.ToLower()} ({FirstId})";

}

}

public virtual string FullDescription

{

get { return \_description; }

}

}

}

# 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- 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();

// Object Configurations

Location hallWay = new Location(

new string[] { "the Hallway", "Hallway" },

"Hallway",

"This is a long well lit Hallway.",

"walk down into the Hallway"

);

Player me = new Player(PlayerName, PlayerDescription, hallWay); // Create Player

// Create two items and put these to player's inventory

Item sword = new Item(

new string[] { "sword", "bronze sword" },

"Bronze Sword",

"A shiny bronze sword"

);

Item shield = new Item(

new string[] { "shield", "wooden shield" },

"Wooden Shield",

"A tough wooden shield"

);

me.Inventory.Put(sword);

me.Inventory.Put(shield);

// Create a bag and put it to player's inventory

Bag myBag = new Bag(

new string[] { "bag", "backpack" },

"Leather Bag",

"A sturdy leather bag to carry items"

);

me.Inventory.Put(myBag);

// Create another item and add it to the bag

Item potion = new Item(

new string[] { "potion", "health potion" },

"Health Potion",

"A magical red potion that restores health"

);

myBag.Inventory.Put(potion);

// Create three object and placed in the Hallway

Item bow = new Item(

new string[] { "bow", "longbow" },

"Longbow",

"A finely crafted bow with great range"

);

Item helmet = new Item(

new string[] { "helmet", "iron helmet" },

"Iron Helmet",

"A sturdy iron helmet for head protection"

);

Item ring = new Item(

new string[] { "ring", "magic ring" },

"Magic Ring",

"A mysterious ring that glows faintly with magical energy"

);

hallWay.Inventory.Put(bow);

hallWay.Inventory.Put(helmet);

hallWay.Inventory.Put(ring);

// Command Configuration

LookCommand lookCommand = new LookCommand();

// 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(lookCommand.Execute(me, command.Split(' ')));

}

}

}

}

}

# TestPlayer.cs

using NUnit.Framework;

using NUnit.Framework.Legacy;

using SwinAdventure;

namespace UnitTests

{

[TestFixture]

public class TestPlayer

{

private Player testPlayer;

private Location testLocation;

private Item sword = new Item(

new string[] { "sword", "bronze sword" },

"Bronze Sword",

"A shiny bronze sword"

);

private Item shield = new Item(

new string[] { "shield", "wooden shield" },

"Wooden Shield",

"A tough wooden shield"

);

private Item potion = new Item(

new string[] { "potion", "health potion" },

"Health Potion",

"A magical red potion that restores health"

);

[SetUp]

public void Setup()

{

testLocation = new Location(

new string[] { "a small tant", "tant" },

"Small Tant",

"This a rest place for traveller.",

"walk by the road and see the tank and come in."

);

testPlayer = new Player("Show", "The Programmer", testLocation);

testPlayer.Inventory.Put(sword);

testPlayer.Inventory.Put(shield);

testPlayer.Inventory.Put(potion);

}

[Test]

public void TestPlayerIsIdentifiable()

{

ClassicAssert.True(testPlayer.AreYou("me"));

ClassicAssert.True(testPlayer.AreYou("inventory"));

}

[Test]

public void TestPlayerLocateItems()

{

ClassicAssert.That(sword, Is.EqualTo(testPlayer.Locate("sword")));

ClassicAssert.True(testPlayer.Inventory.HasItem("sword"));

ClassicAssert.That(shield, Is.EqualTo(testPlayer.Locate("wooden shield")));

ClassicAssert.True(testPlayer.Inventory.HasItem("wooden shield"));

}

[Test]

public void TestPlayerLocateItself()

{

ClassicAssert.That(testPlayer, Is.EqualTo(testPlayer.Locate("me")));

ClassicAssert.That(testPlayer, Is.EqualTo(testPlayer.Locate("inventory")));

}

[Test]

public void TestPlayerLocateNothing()

{

ClassicAssert.That(testPlayer.Locate("gun"), Is.EqualTo(null));

}

[Test]

public void TestPlayerFullDescription()

{

string testDescription =

$"You are Show The Programmer\nYou are carrying\n \t{sword.ShortDescription}\n\t{shield.ShortDescription}\n\t{potion.ShortDescription}\n";

ClassicAssert.That(testPlayer.FullDescription, Is.EqualTo(testDescription));

}

}

}

# TestLookCommand.cs

using NUnit.Framework;

using NUnit.Framework.Legacy;

using SwinAdventure;

namespace UnitTests

{

[TestFixture]

public class TestLockCommand

{

private LookCommand look;

private Location testLocation;

private Player testPlayer;

private Bag bag;

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

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

private Item diamond = new Item(

new string[] { "diamond" },

"a diamond",

"This is a diamond"

);

[SetUp]

public void Setup()

{

look = new LookCommand();

testLocation = new Location(

new string[] { "a small tant", "tant" },

"Small Tant",

"This a rest place for traveller.",

"walk by the road and see the tank and come in."

);

testPlayer = new Player("Show", "The Programmer", testLocation);

bag = new Bag(

new string[] { "bag", "backpack", "leather bag" },

"Leather Bag",

"A sturdy leather bag to carry items"

);

testPlayer.Inventory.Put(bag);

}

[Test]

public void TestLookAtMe()

{

string excepted = testPlayer.FullDescription;

string testOutPut = look.Execute(

testPlayer,

new string[] { "look", "at", "Inventory" }

);

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void TestLookAtGem()

{

string excepted = gem.FullDescription;

testPlayer.Inventory.Put(gem);

string testOutPut = look.Execute(testPlayer, new string[] { "look", "at", "Gem" });

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void TestLookAtUnk()

{

string excepted = $"I cannot find the gem in the {testPlayer.Name}";

string testOutPut = look.Execute(testPlayer, new string[] { "look", "at", "Gem" });

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void TestLookAtGemInMe()

{

string excepted = gem.FullDescription;

testPlayer.Inventory.Put(gem);

string testOutPut = look.Execute(

testPlayer,

new string[] { "look", "at", "Gem", "in", "me" }

);

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void TestLookAtGemInBag()

{

string excepted = gem.FullDescription;

bag.Inventory.Put(gem);

string testOutPut = look.Execute(

testPlayer,

new string[] { "look", "at", "Gem", "in", "bag" }

);

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void TestLookAtGemInNoBag()

{

string excepted = "I cannot find the bag";

Player noBagPlayer = new Player("Ricky", "I have No Bag bro", testLocation);

string testOutPut = look.Execute(

noBagPlayer,

new string[] { "look", "at", "Gem", "in", "bag" }

);

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void TestLookAtNoGemInBag()

{

string excepted = $"I cannot find the gem in the {bag.Name}";

string testOutPut = look.Execute(

testPlayer,

new string[] { "look", "at", "Gem", "in", "bag" }

);

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

[Test]

public void InvalidLook()

{

string excepted = "I don\'t know how to look like that";

string testOutPut = look.Execute(testPlayer, new string[] { "look", "around" });

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

excepted = "Error in look input";

testOutPut = look.Execute(testPlayer, new string[] { "hello", "105293041" });

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

excepted = $"I cannot find the show wai yan in the {testPlayer.Name}";

testOutPut = look.Execute(testPlayer, new string[] { "look", "at", "Show Wai Yan" });

ClassicAssert.That(testOutPut, Is.EqualTo(excepted));

}

}

}

# TestLocation.cs

using NUnit.Framework;

using NUnit.Framework.Legacy;

using SwinAdventure;

namespace UnitTests

{

[TestFixture]

public class TestLocation

{

private Location testLocation;

private Bag bag;

private Player player;

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

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

private Item diamond = new Item(

new string[] { "diamond" },

"a diamond",

"This is a diamond"

);

[SetUp]

public void Setup()

{

testLocation = new Location(

new string[] { "a small tant", "tant" },

"Small Tant",

"This a rest place for traveller.",

"walk by the road and see the tank and come in."

);

bag = new Bag(

new string[] { "bag", "backpack", "leather bag" },

"Leather Bag",

"A sturdy leather bag to carry items"

);

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

bag.Inventory.Put(gem);

bag.Inventory.Put(diamond);

testLocation.Inventory.Put(shovel);

testLocation.Inventory.Put(bag);

}

[Test]

public void TestLocationIsIdentifiable()

{

ClassicAssert.True(testLocation.AreYou("location"));

ClassicAssert.True(testLocation.AreYou("place"));

}

[Test]

public void TestLocationCanLocateItem()

{

string bagId = bag.FirstId;

ClassicAssert.That(bag, Is.EqualTo(testLocation.Locate(bagId)));

string shovelId = shovel.FirstId;

ClassicAssert.That(shovel, Is.EqualTo(testLocation.Locate(shovelId)));

}

[Test]

public void TestPlayerCanLocateItemInLocation()

{

string bagId = bag.FirstId;

ClassicAssert.That(bag, Is.EqualTo(player.Locate(bagId)));

string shovelId = shovel.FirstId;

ClassicAssert.That(shovel, Is.EqualTo(player.Locate(shovelId)));

}

}

}

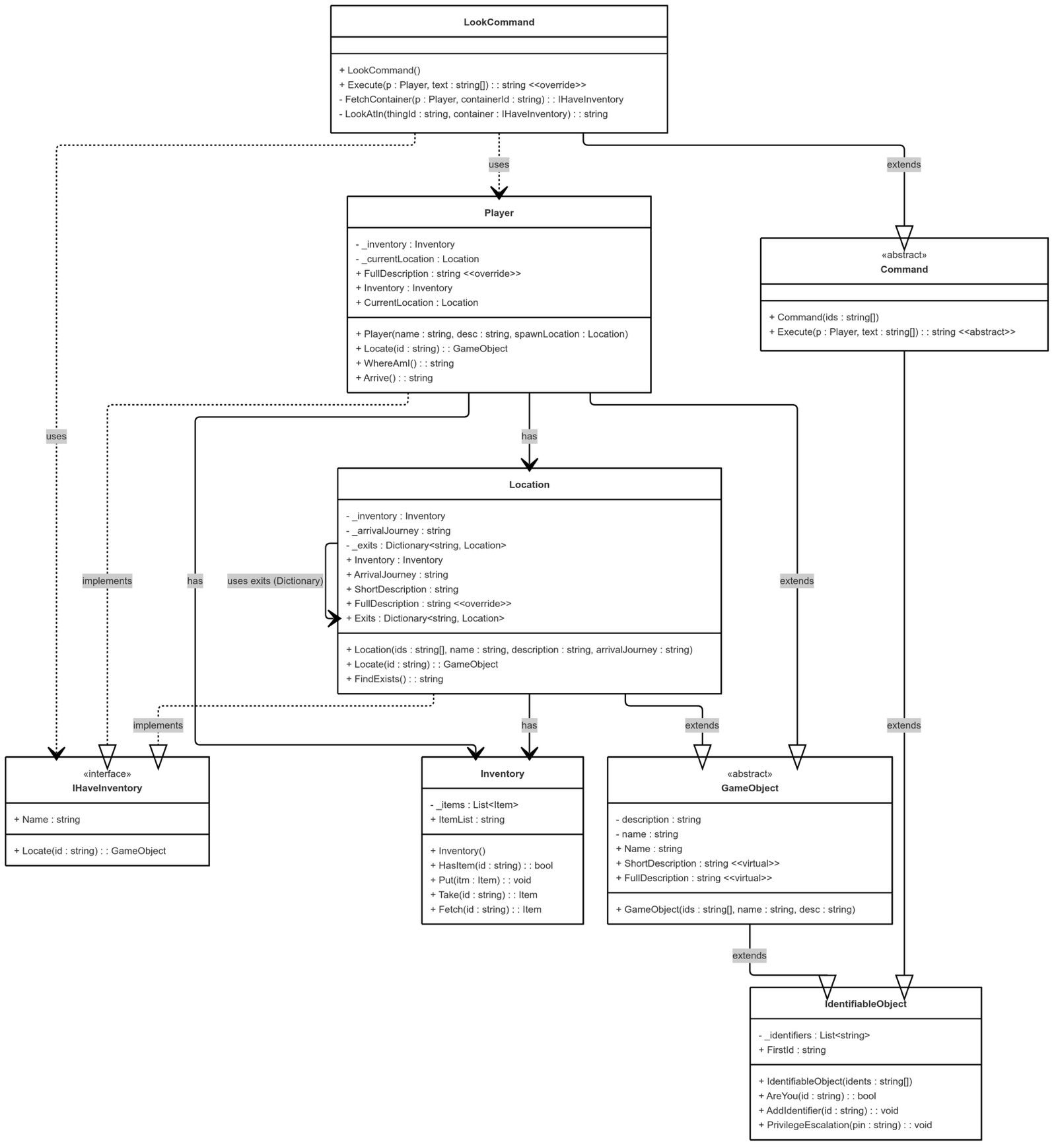
# Screenshot of unit test passing

# Screenshot of program running showing new commands related to locations

A screenshot of a computer program

AI-generated content may be incorrect.

# UML Class diagram showing what needs to be added



# UML Sequence diagram to explain how Locate works in the Player

A screenshot of a computer program

AI-generated content may be incorrect.