## Problem 1 – Music Shop Management

The local music shop owner has asked you to write a system to help him keep track of what articles he is selling. The shop is going to be a really promising one and it will become a chain of stores very soon. This means you've got to make your system work for more than one store. He didn't have much time to explain because he was busy strumming his guitar but here are his basic requirements:

The system keeps track of **music shops** and what is sold in them.

**Music shops** have **name** and a set of **articles**. A music shop owner can **add** or **remove** articles or **view** the current items in the shop.

**Articles** have **make**, **model**, and **price**. An article must be unique (i. e. have unique **make** and **model**), but can exist in more than one shop at a time.

There are two main types of articles – **microphones** and **musical instruments**. Microphones can only **have cables** or be **wireless**. There is nothing more special about them.

**Instruments** are a different thing, though. Since the store is not so big, it only sells **guitars** and **drums** at the moment. Instruments have **color** and can be **electronic** or not. **Drums** have **size** – **width** and **height**, so the shop owners know how much they will need to pay in order to ship them. **Guitars** have two main parts – the **body** – the 'fat' part, and the **fingerboard** – the 'skinny' part. They can be made of different **types of wood**. Guitars also have **number of strings**.

Guitars can be one of the following types: **electric guitars**, **acoustic guitars**, or **bass guitars**. **Electric guitars** come pre-packaged with some adapters, so the shop owners need to keep track of how many **adapters** an electric guitar has. They also differ by their **number of frets** (the little metal things which cross the neck). **Acoustic guitars** can come with a **case** or not, and have strings made of some **material** such as steel or nylon. **Bass guitars** are like regular guitars except they produce a much lower sound and have **only four strings**. Acoustic and electric guitars have **six strings**.

**Electronic instruments** are **electric guitars** and **bass guitars**. **Acoustic guitars** and **drums** are not electronic.

### Design the Class Hierarchy

Your task is **to design an object-oriented class hierarchy** following the best practices in object-oriented programming (OOP) and object-oriented design (OOD). **Avoid duplicated code** through **abstraction**, **inheritance**, and **polymorphism**, and **encapsulate all fields** correctly.

All **music shops** should implement **IMusicShop**, all **articles** should implement **IArticle**. **Microphones** should implement **IMicrophone**, and **instruments** should implement **IInstrument**. **Drums** should implement **IDrums**, and **guitars** – **IGuitar.** **Electrc guitars** should implement **IElectricGuitar**, acoustic guitars – **IAcousticGuitar**, and **bass guitars** – **IBassGuitar**.

You are given the **interfaces** that you should **obligatorily** implement and use as the basis for your code:

|  |
| --- |
| public interface IMusicShop  {  string Name { get; }  IList<IArticle> Articles { get; }  void AddArticle(IArticle article);  void RemoveArticle(IArticle article);  string ListArticles();  }  public interface IArticle  {  string Make { get; }  string Model { get; }  decimal Price { get; }  }  public interface IMicrophone : IArticle  {  bool HasCable { get; }  }  public interface IInstrument : IArticle  {  string Color { get; }  bool IsElectronic { get; }  }  public interface IDrums : IInstrument  {  int Width { get; }  int Height { get; }  }  public interface IGuitar : IInstrument  {  string BodyWood { get; }  string FingerboardWood { get; }  int NumberOfStrings { get; }  }  public interface IElectricGuitar : IGuitar  {  int NumberOfAdapters { get; }  int NumberOfFrets { get; }  }  public interface IAcousticGuitar : IGuitar  {  bool CaseIncluded { get; }  StringMaterial StringMaterial { get; }  }  public interface IBassGuitar : IGuitar  {  } |

### Validate the Data

You should follow these validity rules very strictly in order to ensure the data integrity within the system:

#### Music shop validity rules:

* Name is **required** (cannot be missing or blank)

#### Article validity rules:

* Make and model are **required** (cannot be missing or blank)
* Price must be **positive**

#### Instrument validity rules:

* Color is **required** (cannot be missing or blank)

#### Drums validity rules:

* Width and height for a set of drums must be positive

#### Guitar validity rules:

* Body and fingerboard wood are required (cannot be missing or blank)
* Guitars generally have six strings (bass guitars have only four)

#### Electric guitar validity rules:

* Number of frets must be positive
* Number of adapters must be non-negative (an electric guitar can be sold with no adapter, because they can be bought separately)

#### Acoustic guitar validity rules:

* Strings can be made of one of the following materials: steel, brass, bronze, or nylon

Throw appropriate exception when data validation fails. When there is **an error in a parameter** (such as a missing required parameter), throw an **ArgumentException**, and when there is a **forbidden or meaningless method**, throw an **InvalidOperationException**.

The messages for the exceptions should be as follows:

* For required parameters: **The <parameter> is required.**
* For parameters with only positive values: **The <parameter> must be positive.**

**Always provide the lowest possible visibility** for properties and methods.

A programmer using the system **must not be able** to create instances of classes implementing **IArticle, IInstrument**, and **IGuitar directly.**

All music shops can be created only through **IMusicShopFactory** implemented by the class **MusicShopFactory**. All articles can be created only through **IArticleFactory** implemented by the class **ArticleFactory**. Both classes are located in the namespace **MusicShopManager.Engine.Factories**.

### Format The Output

The **music shop list of items** should return information in the following form:

|  |
| --- |
| **===== <name> =====**  **<articles>** |

**<articles>** is a list of articles, ordered by type, as follows

|  |
| --- |
| **----- Microphones -----**  **----- Drums -----**  **----- Electric guitars -----**  **----- Acoustic guitars -----**  **----- Bass guitars -----** |

**If there are no articles of a certain type, it must not be present in the menu.**

If the music shop is empty, the message should be:

|  |
| --- |
| **The shop is empty. Come back soon.** |

Within the categories, all articles should be **ordered alphabetically by name**. **Round all prices to 2 digits** after the decimal separator. Refer to the example to gain a deeper understanding how the article listing should work.

Microphones are presented in the following form:

|  |
| --- |
| **= <make> <model> =**  **Price: $<price>**  **Cable: <yes / no>** |

Drums are presented in the following form:

|  |
| --- |
| **= <make> <model> =**  **Price: $<price>**  **Color: <color>**  **Electronic: <yes / no>**  **Size: <width>cm x <height>cm** |

Electric guitars are presented in the following form:

|  |
| --- |
| **= <make> <model> =**  **Price: $<price>**  **Color: <color>**  **Electronic: <yes / no>**  **Strings: <strings>**  **Body wood: <body wood>**  **Fingerboard wood: <fingerboard wood>**  **Adapters: <adapters>**  **Frets: <frets>** |

Acoustic guitars are presented in the following form:

|  |
| --- |
| **= <make> <model> =**  **Price: $<price>**  **Color: <color>**  **Electronic: <yes / no>**  **Strings: <strings>**  **Body wood: <body wood>**  **Fingerboard wood: <fingerboard wood>**  **Case included: <yes / no>**  **String material: <string material>** |

Bass guitars are presented in the following form:

|  |
| --- |
| **= <make> <model> =**  **Price: $<price>**  **Color: <color>**  **Electronic: <yes / no>**  **Strings: <strings>**  **Body wood: <body wood>**  **Fingerboard wood: <fingerboard wood>** |

### Additional Notes

You are given a **command execution engine** to simplify your work. Please put all your classes in the **RestaurantManager.Models** namespace and all factories in the **RestaurantManager.Engine.Factories** namespace.

**You are not allowed to change anything else in the project**.

The engine accepts the following commands:

* CreateMusicShop[name:<name>]
* CreateMicrophone[make:<make>;model:<model>;price:<price>;cable:<yes / no>]
* CreateDrums[make:<make>;model:<model>;price:<price>;color:<color>;width:<width>;height:<height>]
* CreateElectricGuitar[make:<make>;model:<model>;price:<price>;color:<color>;body:<body>;fingerboard:<fingerboard>;adapters:<adapters>;frets:<frets>]
* CreateAcousticGuitar[make:<make>;model:<model>;price:<price>;color:<color>;body:<body>;fingerboard:<fingerboard>;case:<yes / no>;strings:<type of strings>]
* CreateBassGuitar[make:<make>;model:<model>;price:<price>;color:<color>;body:<body>;fingerboard:<fingerboard>]
* AddArticleToShop[name:<name>;make:<make>;model:<model>]
* RemoveArticleFromShop[name:<name>;make:<make>;model:<model>]
* ListArticles[name:<name>]

The parameters may be provided in any sequence. The engine returns appropriate messages for each command. Duplicate music shop and article names are not allowed. The engine skips blank lines and lines which start with "//". You may refer to the sample input and output for more details.

### Sample Input

|  |
| --- |
| ListArticles[name:My Music Shop]  CreateMusicShop[name:My Music Shop]  ListArticles[name:My Music Shop]  CreateMicrophone[make:JTS;model:TM989;price:20.00;cable:yes]  AddArticleToShop[model:TM989;name:My Music Shop;make:JTS]  ListArticles[name:My Music Shop]  CreateDrums[model:DP-101;price:569.99;make:Legion;color:black;width:56;height:40]  AddArticleToShop[name:My Music Shop;make:Legion;model:DP-101]  CreateElectricGuitar[model:Chicago Flame;body:Mahogany;fingerboard:Ebony;price:1159.989;make:DEAN V;color:Classic Black;adapters:2;frets:22]  AddArticleToShop[name:My Music Shop;make:DEAN V;model:Chicago Flame]  CreateAcousticGuitar[body:Mahogany;model:AD-20;fingerboard:Mahogany;price:320;make:Aria;color:Natural;case:no;strings:Steel]  AddArticleToShop[name:My Music Shop;make:Aria;model:AD-20]  CreateElectricGuitar[model:Chicago Flame;body:Mahogany;fingerboard:Ebony;price:1159.989;make:DEAN V;color:Classic Black;adapters:2;frets:22]  AddArticleToShop[name:My Music Shop;make:DEAN V;model:Chicago Flame]  ListArticles[name:My Music Shop]  CreateMusicShop[name:Other Shop]  AddArticleToShop[name:Other Shop;make:DEAN V;model:Chicago Flame] CreateBassGuitar[make:Dean;model:Hillsboro 09 PJ;price:380.99;color:Classic Black;body:Basswood;fingerboard:Maple]  AddArticleToShop[name:My Music Shop;make:Dean;model:Hillsboro 09 PJ]  ListArticles[name:My Music Shop]  ListArticles[name:Other Shop]  AddArticleToShop[name:My Music Shop;make:Does Not;model:Exist]  CreateElectricGuitar[model:Rocketeer Deluxe;body:Basswood;frets:22;fingerboard:Rosewood;price:473.45;make:Rogue;adapters:2;color:Brown / White]  CreateElectricGuitar[model:DDF-1340TV / VSB;body:Basswood;fingerboard:Maple;price:1159.989;make:Apollo;color:Sunburst;adapters:3;frets:22]  CreateMicrophone[make:JTS;model:IN264/IN264TH;price:1136;cable:no]  AddArticleToShop[model:IN264/IN264TH;name:My Music Shop;make:JTS]  AddArticleToShop[name:My Music Shop;make:Rogue;model:Rocketeer Deluxe]  AddArticleToShop[name:My Music Shop;make:Apollo;model:DDF-1340TV / VSB]  ListArticles[name:My Music Shop]  RemoveArticleFromShop[model:IN264/IN264TH;name:My Music Shop;make:JTS]  RemoveArticleFromShop[model:TM989;name:My Music Shop;make:JTS]  ListArticles[name:My Music Shop]  RemoveArticleFromShop[model:TM989;name:My Music Shop;make:JTS]  End |

### Sample Output

|  |
| --- |
| The music shop My Music Shop does not exist  Music shop My Music Shop created  ===== My Music Shop =====  The shop is empty. Come back soon.  Microphone JTS TM989 created  Article JTS TM989 successfully added to music shop My Music Shop  ===== My Music Shop =====  ----- Microphones -----  = JTS TM989 =  Price: $20.00  Cable: yes  Drums Legion DP-101 created  Article Legion DP-101 successfully added to music shop My Music Shop  Electric guitar DEAN V Chicago Flame created  Article DEAN V Chicago Flame successfully added to music shop My Music Shop  Acoustic guitar Aria AD-20 created  Article Aria AD-20 successfully added to music shop My Music Shop  The article DEAN V Chicago Flame already exists  The article DEAN V Chicago Flame already exists in shop My Music Shop  ===== My Music Shop =====  ----- Microphones -----  = JTS TM989 =  Price: $20.00  Cable: yes  ----- Drums -----  = Legion DP-101 =  Price: $569.99  Color: black  Electronic: no  Size: 56cm x 40cm  ----- Electric guitars -----  = DEAN V Chicago Flame =  Price: $1159.99  Color: Classic Black  Electronic: yes  Strings: 6  Body wood: Mahogany  Fingerboard wood: Ebony  Adapters: 2  Frets: 22  ----- Acoustic guitars -----  = Aria AD-20 =  Price: $320.00  Color: Natural  Electronic: no  Strings: 6  Body wood: Mahogany  Fingerboard wood: Mahogany  Case included: no  String material: Steel  Music shop Other Shop created  Article DEAN V Chicago Flame successfully added to music shop Other Shop  Bass guitar Dean Hillsboro 09 PJ created  Article Dean Hillsboro 09 PJ successfully added to music shop My Music Shop  ===== My Music Shop =====  ----- Microphones -----  = JTS TM989 =  Price: $20.00  Cable: yes  ----- Drums -----  = Legion DP-101 =  Price: $569.99  Color: black  Electronic: no  Size: 56cm x 40cm  ----- Electric guitars -----  = DEAN V Chicago Flame =  Price: $1159.99  Color: Classic Black  Electronic: yes  Strings: 6  Body wood: Mahogany  Fingerboard wood: Ebony  Adapters: 2  Frets: 22  ----- Acoustic guitars -----  = Aria AD-20 =  Price: $320.00  Color: Natural  Electronic: no  Strings: 6  Body wood: Mahogany  Fingerboard wood: Mahogany  Case included: no  String material: Steel  ----- Bass guitars -----  = Dean Hillsboro 09 PJ =  Price: $380.99  Color: Classic Black  Electronic: yes  Strings: 4  Body wood: Basswood  Fingerboard wood: Maple  ===== Other Shop =====  ----- Electric guitars -----  = DEAN V Chicago Flame =  Price: $1159.99  Color: Classic Black  Electronic: yes  Strings: 6  Body wood: Mahogany  Fingerboard wood: Ebony  Adapters: 2  Frets: 22  The article Does Not Exist does not exist  Electric guitar Rogue Rocketeer Deluxe created  Electric guitar Apollo DDF-1340TV / VSB created  Microphone JTS IN264/IN264TH created  Article JTS IN264/IN264TH successfully added to music shop My Music Shop  Article Rogue Rocketeer Deluxe successfully added to music shop My Music Shop  Article Apollo DDF-1340TV / VSB successfully added to music shop My Music Shop  ===== My Music Shop =====  ----- Microphones -----  = JTS IN264/IN264TH =  Price: $1136.00  Cable: no  = JTS TM989 =  Price: $20.00  Cable: yes  ----- Drums -----  = Legion DP-101 =  Price: $569.99  Color: black  Electronic: no  Size: 56cm x 40cm  ----- Electric guitars -----  = Apollo DDF-1340TV / VSB =  Price: $1159.99  Color: Sunburst  Electronic: yes  Strings: 6  Body wood: Basswood  Fingerboard wood: Maple  Adapters: 3  Frets: 22  = DEAN V Chicago Flame =  Price: $1159.99  Color: Classic Black  Electronic: yes  Strings: 6  Body wood: Mahogany  Fingerboard wood: Ebony  Adapters: 2  Frets: 22  = Rogue Rocketeer Deluxe =  Price: $473.45  Color: Brown / White  Electronic: yes  Strings: 6  Body wood: Basswood  Fingerboard wood: Rosewood  Adapters: 2  Frets: 22  ----- Acoustic guitars -----  = Aria AD-20 =  Price: $320.00  Color: Natural  Electronic: no  Strings: 6  Body wood: Mahogany  Fingerboard wood: Mahogany  Case included: no  String material: Steel  ----- Bass guitars -----  = Dean Hillsboro 09 PJ =  Price: $380.99  Color: Classic Black  Electronic: yes  Strings: 4  Body wood: Basswood  Fingerboard wood: Maple  Article JTS IN264/IN264TH successfully removed from music shop My Music Shop  Article JTS TM989 successfully removed from music shop My Music Shop  ===== My Music Shop =====  ----- Drums -----  = Legion DP-101 =  Price: $569.99  Color: black  Electronic: no  Size: 56cm x 40cm  ----- Electric guitars -----  = Apollo DDF-1340TV / VSB =  Price: $1159.99  Color: Sunburst  Electronic: yes  Strings: 6  Body wood: Basswood  Fingerboard wood: Maple  Adapters: 3  Frets: 22  = DEAN V Chicago Flame =  Price: $1159.99  Color: Classic Black  Electronic: yes  Strings: 6  Body wood: Mahogany  Fingerboard wood: Ebony  Adapters: 2  Frets: 22  = Rogue Rocketeer Deluxe =  Price: $473.45  Color: Brown / White  Electronic: yes  Strings: 6  Body wood: Basswood  Fingerboard wood: Rosewood  Adapters: 2  Frets: 22  ----- Acoustic guitars -----  = Aria AD-20 =  Price: $320.00  Color: Natural  Electronic: no  Strings: 6  Body wood: Mahogany  Fingerboard wood: Mahogany  Case included: no  String material: Steel  ----- Bass guitars -----  = Dean Hillsboro 09 PJ =  Price: $380.99  Color: Classic Black  Electronic: yes  Strings: 4  Body wood: Basswood  Fingerboard wood: Maple  The article JTS TM989 does not exist in shop My Music Shop |

**ENGINE**

namespace MusicShopManager.Engine

{

using System;

using System.Collections.Generic;

using System.Linq;

using MusicShopManager.Engine.Factories;

using MusicShopManager.Interfaces;

using MusicShopManager.Interfaces.Engine;

using MusicShopManager.Models;

public sealed class MusicShopEngine : IMusicShopEngine

{

private static IMusicShopEngine instance;

private readonly IMusicShopFactory musicShopFactory;

private readonly IArticleFactory articleFactory;

private readonly IDictionary<string, IMusicShop> musicShops;

private readonly IDictionary<string, IArticle> articles;

private readonly IUserInterface userInterface;

private MusicShopEngine()

{

this.musicShopFactory = new MusicShopFactory();

this.articleFactory = new ArticleFactory();

this.musicShops = new Dictionary<string, IMusicShop>();

this.articles = new Dictionary<string, IArticle>();

this.userInterface = new ConsoleInterface();

}

public static IMusicShopEngine Instance

{

get

{

if (instance == null)

{

instance = new MusicShopEngine();

}

return instance;

}

}

public void Start()

{

var commands = this.ReadCommands();

var commandResults = this.ProcessCommands(commands);

this.userInterface.Output(commandResults);

}

private ICollection<ICommand> ReadCommands()

{

var commands = new List<ICommand>();

foreach (var line in this.userInterface.Input())

{

commands.Add(Command.Parse(line));

}

return commands;

}

private IEnumerable<string> ProcessCommands(ICollection<ICommand> commands)

{

var commandResults = new List<string>();

foreach (var command in commands)

{

string commandResult;

switch (command.Name)

{

case EngineConstants.CreateMusicShopCommand:

commandResult = this.CreateMusicShop(command.Parameters["name"]);

break;

case EngineConstants.CreateMicrophoneCommand:

commandResult = this.CreateMicrophone(

command.Parameters["make"],

command.Parameters["model"],

decimal.Parse(command.Parameters["price"]),

this.ParseBoolean(command.Parameters["cable"]));

break;

case EngineConstants.CreateDrumsCommand:

commandResult = this.CreateDrums(

command.Parameters["make"],

command.Parameters["model"],

decimal.Parse(command.Parameters["price"]),

command.Parameters["color"],

int.Parse(command.Parameters["width"]),

int.Parse(command.Parameters["height"]));

break;

case EngineConstants.CreateElectricGuitarCommand:

commandResult = this.CreateElectricGuitar(

command.Parameters["make"],

command.Parameters["model"],

decimal.Parse(command.Parameters["price"]),

command.Parameters["color"],

command.Parameters["body"],

command.Parameters["fingerboard"],

int.Parse(command.Parameters["adapters"]),

int.Parse(command.Parameters["frets"]));

break;

case EngineConstants.CreateAcousticGuitarCommand:

commandResult = this.CreateAcousticGuitar(

command.Parameters["make"],

command.Parameters["model"],

decimal.Parse(command.Parameters["price"]),

command.Parameters["color"],

command.Parameters["body"],

command.Parameters["fingerboard"],

this.ParseBoolean(command.Parameters["case"]),

command.Parameters["strings"]);

break;

case EngineConstants.CreateBassGuitarCommand:

commandResult = this.CreateBassGuitar(

command.Parameters["make"],

command.Parameters["model"],

decimal.Parse(command.Parameters["price"]),

command.Parameters["color"],

command.Parameters["body"],

command.Parameters["fingerboard"]);

break;

case EngineConstants.AddArticleToShopCommand:

commandResult = this.AddArticleToShop(

command.Parameters["name"],

command.Parameters["make"],

command.Parameters["model"]);

break;

case EngineConstants.RemoveArticleFromShopCommand:

commandResult = this.RemoveArticleFromShop(

command.Parameters["name"],

command.Parameters["make"],

command.Parameters["model"]);

break;

case EngineConstants.ListArticlesCommand:

commandResult = this.ListArticles(

command.Parameters["name"]);

break;

default:

commandResult = string.Format(EngineConstants.InvalidCommandMessage, command.Name);

break;

}

commandResults.Add(commandResult);

}

return commandResults;

}

private bool ParseBoolean(string boolValue)

{

if (boolValue == "yes")

{

return true;

}

else if (boolValue == "no")

{

return false;

}

else

{

throw new ArgumentException("Invalid boolean value provided: " + boolValue);

}

}

private string CreateMusicShop(string name)

{

if (this.musicShops.ContainsKey(name))

{

return string.Format(EngineConstants.MusicShopExistsMessage, name);

}

var musicShop = this.musicShopFactory.CreateMusicShop(name);

this.musicShops.Add(name, musicShop);

return string.Format(EngineConstants.MusicShopCreatedMessage, name);

}

private string CreateMicrophone(string make, string model, decimal price, bool hasCable)

{

string name = make + " " + model;

try

{

this.EnsureUniqueArticle(make, model);

}

catch (ArgumentException)

{

return string.Format(EngineConstants.ArticleExistsMessage, name);

}

var microphone = this.articleFactory.CreateMirophone(make, model, price, hasCable);

this.articles.Add(name, microphone);

return string.Format(EngineConstants.ArticleCreatedMessage, "Microphone", name);

}

private string CreateDrums(string make, string model, decimal price, string color, int width, int height)

{

string name = make + " " + model;

try

{

this.EnsureUniqueArticle(make, model);

}

catch (ArgumentException)

{

return string.Format(EngineConstants.ArticleExistsMessage, name);

}

var drums = this.articleFactory.CreateDrums(make, model, price, color, width, height);

this.articles.Add(name, drums);

return string.Format(EngineConstants.ArticleCreatedMessage, "Drums", name);

}

private string CreateElectricGuitar(string make, string model, decimal price, string color,

string bodyWood, string fingerboardWood, int numberOfAdapters, int numberOfFrets)

{

string name = make + " " + model;

try

{

this.EnsureUniqueArticle(make, model);

}

catch (ArgumentException)

{

return string.Format(EngineConstants.ArticleExistsMessage, name);

}

var electricGuitar = this.articleFactory.CreateElectricGuitar(make, model, price, color, bodyWood, fingerboardWood, numberOfAdapters, numberOfFrets);

this.articles.Add(name, electricGuitar);

return string.Format(EngineConstants.ArticleCreatedMessage, "Electric guitar", name);

}

private string CreateAcousticGuitar(string make, string model, decimal price, string color,

string bodyWood, string fingerboardWood, bool caseIncluded, string stringMaterial)

{

string name = make + " " + model;

try

{

this.EnsureUniqueArticle(make, model);

}

catch (ArgumentException)

{

return string.Format(EngineConstants.ArticleExistsMessage, name);

}

var acousticGuitar = this.articleFactory.CreateAcousticGuitar(make, model, price, color,

bodyWood, fingerboardWood, caseIncluded, (StringMaterial)Enum.Parse(typeof(StringMaterial), stringMaterial));

this.articles.Add(name, acousticGuitar);

return string.Format(EngineConstants.ArticleCreatedMessage, "Acoustic guitar", name);

}

private string CreateBassGuitar(string make, string model, decimal price, string color, string bodyWood, string fingerboardWood)

{

string name = make + " " + model;

try

{

this.EnsureUniqueArticle(make, model);

}

catch (ArgumentException)

{

return string.Format(EngineConstants.ArticleExistsMessage, name);

}

var bassGuitar = this.articleFactory.CreateBassGuitar(make, model, price, color, bodyWood, fingerboardWood);

this.articles.Add(name, bassGuitar);

return string.Format(EngineConstants.ArticleCreatedMessage, "Bass guitar", name);

}

private void EnsureUniqueArticle(string make, string model)

{

string name = make + " " + model;

if (this.articles.ContainsKey(name))

{

throw new ArgumentException(EngineConstants.ArticleExistsMessage, name);

}

}

private string AddArticleToShop(string shopName, string make, string model)

{

if (!this.musicShops.ContainsKey(shopName))

{

return string.Format(EngineConstants.MusicShopDoesNotExistMessage, shopName);

}

string articleName = make + " " + model;

if (!this.articles.ContainsKey(articleName))

{

return string.Format(EngineConstants.ArticleDoesNotExistMessage, articleName);

}

if (this.musicShops[shopName].Articles.Select(a => a.Make + " " + a.Model).Contains(articleName))

{

return string.Format(EngineConstants.ArticleExistsInShopMessage, articleName, shopName);

}

this.musicShops[shopName].AddArticle(this.articles[articleName]);

return string.Format(EngineConstants.ArticleAddedMessage, articleName, shopName);

}

private string RemoveArticleFromShop(string shopName, string make, string model)

{

if (!this.musicShops.ContainsKey(shopName))

{

return string.Format(EngineConstants.MusicShopDoesNotExistMessage, shopName);

}

string articleName = make + " " + model;

if (!this.articles.ContainsKey(articleName))

{

return string.Format(EngineConstants.ArticleDoesNotExistMessage, articleName);

}

if (!this.musicShops[shopName].Articles.Select(a => a.Make + " " + a.Model).Contains(articleName))

{

return string.Format(EngineConstants.ArticleDoesNotExistInShopMessage, articleName, shopName);

}

this.musicShops[shopName].RemoveArticle(this.articles[articleName]);

return string.Format(EngineConstants.ArticleRemovedMessage, articleName, shopName);

}

private string ListArticles(string shopName)

{

if (!this.musicShops.ContainsKey(shopName))

{

return string.Format(EngineConstants.MusicShopDoesNotExistMessage, shopName);

}

return this.musicShops[shopName].ListArticles();

}

}

}

**INPUT & OUTPUT**

namespace MusicShopManager.Engine

{

using System;

using System.Collections.Generic;

using System.Text;

using MusicShopManager.Interfaces.Engine;

public class ConsoleInterface : IUserInterface

{

public IEnumerable<string> Input()

{

string currentLine = Console.ReadLine();

while (currentLine != "End")

{

if (currentLine == string.Empty || currentLine.StartsWith("//"))

{

currentLine = Console.ReadLine();

continue;

}

yield return currentLine;

currentLine = Console.ReadLine();

}

}

public void Output(IEnumerable<string> output)

{

var result = new StringBuilder();

foreach (string line in output)

{

result.AppendLine(line.Trim());

}

Console.WriteLine(result.ToString().Trim());

}

}

}

**COMMAND**

namespace MusicShopManager.Engine

{

using System;

using System.Collections.Generic;

using MusicShopManager.Interfaces.Engine;

public class Command : ICommand

{

private const char CommandNameSeparator = '[';

private const char CommandParameterSeparator = ';';

private const char CommandValueSeparator = ':';

private string name;

private IDictionary<string, string> parameters = new Dictionary<string, string>();

public Command(string input)

{

this.TranslateInput(input);

}

public string Name

{

get

{

return this.name;

}

private set

{

if (string.IsNullOrEmpty(value))

{

throw new ArgumentNullException("The command name is required.");

}

this.name = value;

}

}

public IDictionary<string, string> Parameters

{

get

{

return this.parameters;

}

private set

{

if (value == null)

{

throw new ArgumentNullException("The command parameters are required.");

}

this.parameters = value;

}

}

public static Command Parse(string input)

{

return new Command(input);

}

private void TranslateInput(string input)

{

int parametersBeginning = input.IndexOf(CommandNameSeparator);

this.Name = input.Substring(0, parametersBeginning);

var parametersKeysAndValues = input.Substring(parametersBeginning + 1, input.Length - parametersBeginning - 2)

.Split(new[] { CommandParameterSeparator }, StringSplitOptions.RemoveEmptyEntries);

foreach (var parameter in parametersKeysAndValues)

{

var split = parameter.Split(new[] { CommandValueSeparator }, StringSplitOptions.RemoveEmptyEntries);

this.Parameters.Add(split[0], split[1]);

}

}

}

}

**ARTICLE FACTORY**

using MusicShop.Models;

namespace MusicShopManager.Engine.Factories

{

using System;

using MusicShopManager.Interfaces;

using MusicShopManager.Interfaces.Engine;

using MusicShopManager.Models;

public class ArticleFactory : IArticleFactory

{

public IMicrophone CreateMirophone(string make, string model, decimal price, bool hasCable)

{

return new Microphone(make, model, price, hasCable);

}

public IDrums CreateDrums(string make, string model, decimal price, string color, int width, int height)

{

return new Drum(make, model, price, color, width, height);

}

public IElectricGuitar CreateElectricGuitar(string make, string model, decimal price, string color,

string bodyWood, string fingerboardWood, int numberOfAdapters, int numberOfFrets)

{

return new ElectricGuitar(make, model, price, color, bodyWood, fingerboardWood, numberOfAdapters, numberOfFrets);

}

public IAcousticGuitar CreateAcousticGuitar(string make, string model, decimal price, string color,

string bodyWood, string fingerboardWood, bool caseIncluded, StringMaterial stringMaterial)

{

return new AcousticGuitar(make, model, price, color, bodyWood, fingerboardWood, caseIncluded, stringMaterial);

}

public IBassGuitar CreateBassGuitar(string make, string model, decimal price, string color, string bodyWood, string fingerboardWood)

{

return new BassGuitars(make, model, price, color, bodyWood, fingerboardWood);

}

}

}