.NET Technologies Using C# (PROG32356)
Assignment 1
Due Date: See Slate

INSTRUCTIONS

- *This assignment must be completed individually without any outside collaboration. All work must be your own. Copying or reproducing the work done by others (in part or in full) or letting others to copy or reproduce your own work is subject to significant grade reduction or getting no grade at all and/or being treated as academic dishonesty under the College's Academic Dishonesty Policy.*
- *This is an out of class assignment and you are required to complete this assignment on your own time, unless otherwise specified by your instructor.*
- ***IMPORTANT**: You must submit screenshot(s) demonstrating your work as instructed in the assignment. All screenshots MUST show your name and student Id (you can place a small text window containing your name and id on top of your screen, not covering any content.) All screenshots must be readable in 100% zoom size. Your submission may not be marked or may receive minimum 30% marks penalty, if one or more required screenshot is missing/unreadable/not following the guideline.*
- *Your application must compile and run upon download to receive any mark. Your supplied outputs (e.g., screenshots) may not be marked, if the corresponding program fails to compile and run.*
- *To submit the assignments, please follow the Submission Guideline provided at the end of this assignment.*
- *You must submit the assignment by the due date. Late submissions policy is specified in the Academic Procedures for Evaluations document available through the class plan on Slate.*
- *Total mark = 50 (weight = 10% of the final grade).*

Musical Instruments

Design and implement the following interface and class hierarchy (shown in Figure 1) in C# to demonstrate the relationships and functions among various musical instruments in a typical musical instrument shop.

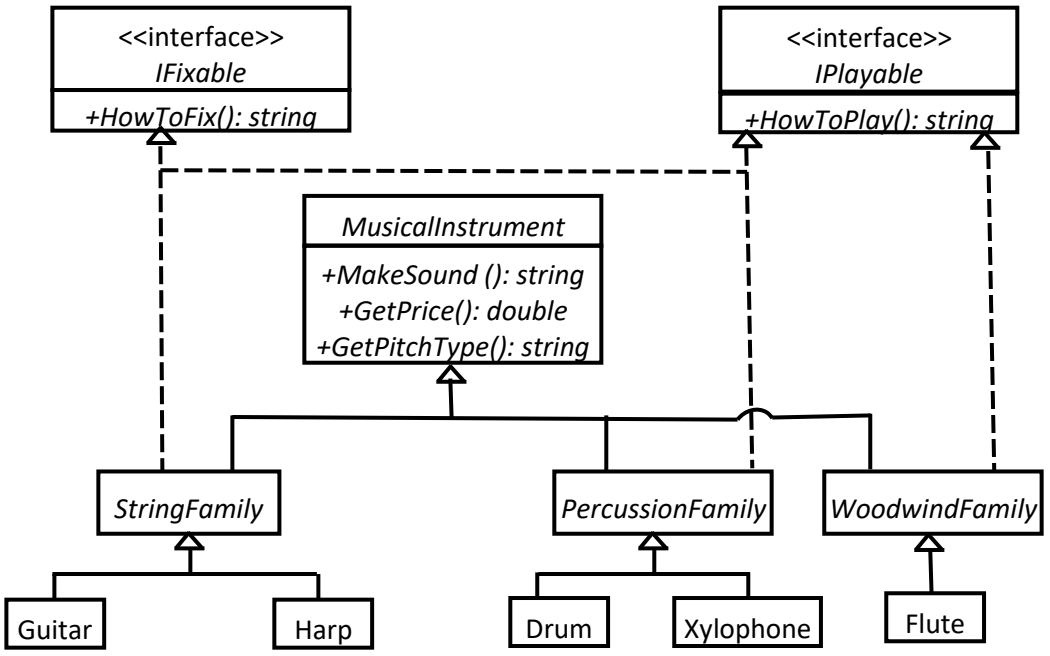

Figure 1: `IFixable` and `IPlayable` are two interfaces. `MusicalInstrument`, `StringFamily`, `PercussionFamily`, and `WoodwindFamily` are all abstract classes. Guitar, Harp, Drum, Xylophone, and Flute are concrete classes.

Most information handled and/or returned by interface and abstract class methods are presented in Table 1. Use this table for information to be returned by your method implementations within respective class.

Table 1: Information on musical instruments

Instrument	Make sound	Price	How to play	How to fix	Pitch Type
Drum	vibrating stretched membrane	$349.50	by hitting the membrane	replace the membrane	Sonic pitch
Flute	guiding a stream of air	$74.90	by blowing into the flute	*N/A: it cannot be fixed*	Fundamental pitch is middle C
Guitar	vibrating strings	$199.00	by strumming the strings	replace the strings	Low to high pitch
Harp	vibrating strings	$255.00	with the thumb and first three fingers	replace the strings	Has seven levels of pitch

Xylophone	through resonators	$49.00	with two mallets	replace bars	Each bar produces different pitch

Your implementation should:

i) **[*Requirement 1*]** receive the price value for each instrument from the user to create an object of that instrument (other information for that instrument object e.g., make sound, how to play, how to fix, and pitch type can be hard coded as per Table 1).

ii) **[*Requirement 2*]** display *How to Play, How to Fix, Pitch type* and *Price* information for the most expensive instrument through comparing objects of all instruments.

iii) **[*Requirement 3*]** display the names of all instruments (Drum, Flute, Guitar, Harp, Xylophone) in descending order of price. You must override the ToString() method to display the name of an instrument.

iv) **[*Requirement 4*]** receive an instrument family name (e.g., String family, Percussion family or Woodwind family) from the user, and display how each instrument of that given family makes sound.

Your implementation must also satisfy following requirements:

v) **[*Requirement 5*]** You must demonstrate the use of `IComparable` interface and implement its `CompareTo()` method (or `IComparer` interface and implement its `Compare()` method) to find the instrument object with the highest price and sort them.

vi) **[*Requirement 6*]** You must demonstrate the use of one or more Collection (e.g., a List of type `MusicalInstrument` to hold objects of each concrete type of musical instrument). Use the `CompareTo()` method that you may have overridden in `MusicalInstrument` class (or `Compare()` method that you might have implemented in a separate helper class) to find the instrument object with the highest price and sort them in the collection. No operation on the instruments should be hard coded – meaning all operations must be done programmatically.

vii) **[*Requirement 7*]** Your program must demonstrate proper modularization of code, C# industry standard with proper comments, indentations, and naming convention.

viii) **[*Requirement 8*]** You must include the following information as commented at the beginning of your main class file.

- Assignment: 1
- Name: <Your Full Name>
- Id: <Your Sheridan Id>

ix) **[*Requirement 9*]** You must test your application for multiple runs with different price values for all instruments. Change the price value for each instrument to show that your application works even if the price of instruments is updated. Take screenshots for at least two runs showing inputs/outputs for Requirements 1 - 4.

x) **[*Requirement 10*]** In addition, you **MUST** submit all the screenshots as specified in Submission Guideline.

Sample Output: On the above information of musical instruments in Table 1, the output of a test run should be as follows:

```
--: Requirement 1 :--
Enter the price for Drum: <349.5>
Enter the price for Flute: <74.9>
Enter the price for Guitar: <199>
Enter the price for Harp: <254>
Enter the price for Xylophone: <49>

--: Requirement 2 :--
The most expensive instrument is: Drum
Drum's cost is: $349.50
Drum is played: by hitting the membrane
Drum fixing: replace the membrane
Drum pitch type: Sonic pitch

--: Requirement 3 :--
Instruments in price descending order:
[Drum, Harp, Guitar, Flute, Xylophone]

--: Requirement 4 :--
Enter an instrument family: <Percussion>
Xylophone makes sound through resonators.
Drum makes sound vibrating stretched membrane.
```

Submission Guideline:

Step 1: Create a document file named <YourFirstName-YourID-A1>. The cover page of this document must include following information:

Assignment #: 1
Course: PROG32356
Your full name:
Your student Id:

Step 2: Paste the screenshots as per Requirement 9 above for at least two runs of your application.

Step 3: Take screenshots for your block of code showing that you have satisfied Requirements 5 and 6. Paste them in the document file.

Note: All screenshots MUST follow the screenshot guideline mentioned in INSTRUCTIONS.

Step 4: Submit following files to the ''Submit Assignment 1" Dropbox available on Slate.

- The C# project as a single zip file (e.g., .zip)

- The document file (Word or pdf) containing screenshots showing outputs.

[Multiple submissions are allowed, but only the last submission will be marked. Please make sure your last submission is complete i.e., includes all required files.]

[Note: Your program will not be marked, if any required file in your project or the additional document file is missing, corrupted, or it fails to open. Please double check whether your zip conversion is successful, by unzipping it and opening all files at least once, before submitting.]

Grading Rubric

Requirements	**Description**	**Marks**
Requirement 1	Missing/incorrect process/input handling	12
Requirement 2	Missing/incorrect design/process/output	30
Requirement 3	Missing/incorrect process/output, use of ToString() method	14
Requirement 4	Missing/incorrect process/design/input/output	14
Requirement 5	Missing/incorrect use of IComparable or IComparer interface	10
Requirement 6	Missing/incorrect use of collection	10
Requirement 7	Modularization and C# industry standard	6
Requirement 8	Missing/incorrect student information	4
Requirements 9 and 10	Missing/incorrect screenshot(s)	-30%
Overall	Missing/incorrect submission file(s) or application files to run	-100%
Total		**100**