# PROG23199 - Assignment 1 [100 marks, weight - 10%]

**Due Date: See the Dropbox on SLATE for due date details.**

## 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.*
- *IMPORTANT: You must submit screenshot(s) demonstrating your work as instructed in the submission guideline. All screenshots MUST show **your name and Sheridan student number** (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(s) 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.*
- *You must include following information as comments at the beginning of each program you submit.*
    - *Assignment No.: 1*
    - *Course: PROG23199*
    - *Your Name:*
    - *Your Sheridan Student Number:*
    - *Submission date:*
    - *Instructor's name: Syed Tanbeer*
- *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 Course Plan document available on Slate.*

## Musical Instruments

Design and implement the interface and class hierarchy shown in Figure 1 to demonstrate the relationships and functions among various musical instruments in a typical musical instrument store.

As shown in the figure, *Repairable*, *PriceProvider*, and *Playable* are interfaces, each containing a single abstract method *how_to_repair()*, *get_price()* and *how_to_play()*, respectively. The dotted line from a class to an interface indicates the 'interface – class' inheritance (e.g., *StringFamily*, *PercussionFamily* classes implement *Repairable* and *Playable* interfaces – meaning either these classes or their subclasses must implement the abstract methods in those interfaces. Similarly, *WoodwindFamily* class implements only *Playable* interface, not the *Repairable* interface,

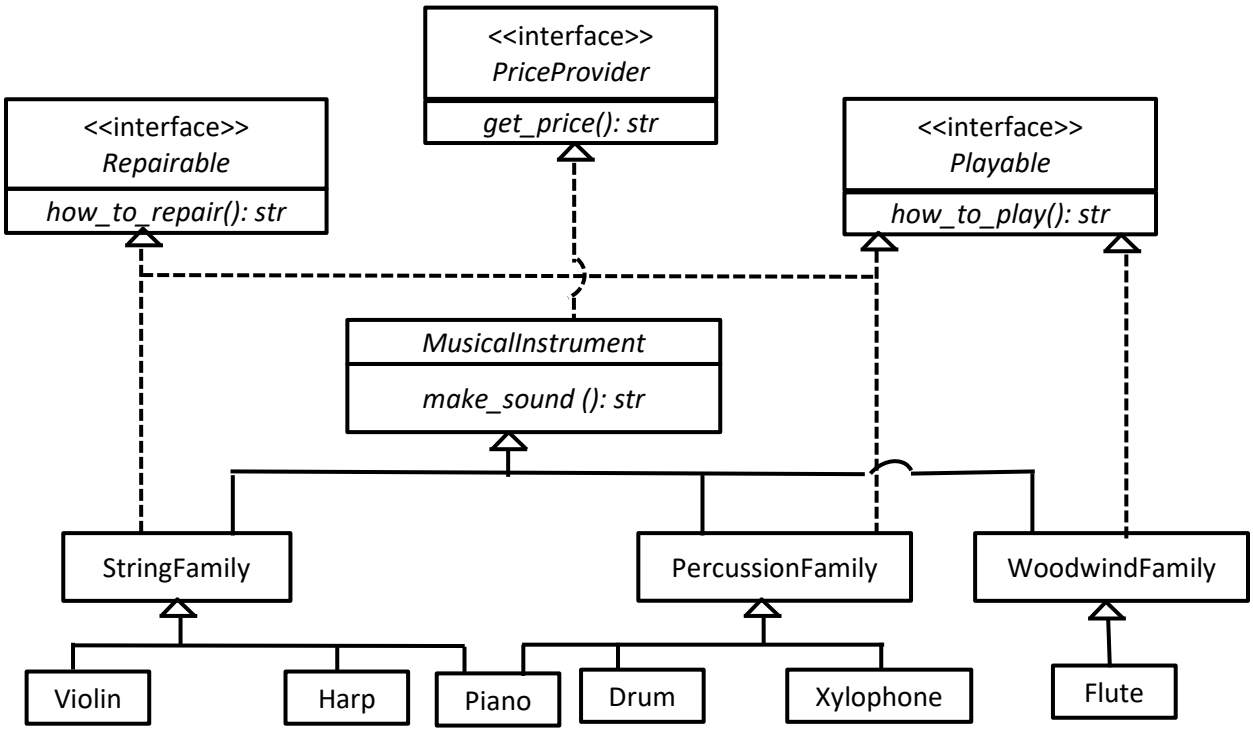


Figure 1: UML diagram showing class interface relationship of musical instruments.

assuming instruments in this category can not be repaired. Also, only class *MusicalInstrument* or its subclasses implement *PriceProvider* interface.)

The class *MusicalInstrument* is an abstract class containing an abstract method *make_sound()*, which must be implemented (overridden) by its concreted subclasses (i.e., *Violin*, *Harp*, *Piano*, *Drum*, *Xylophone* and *Flute*).

Most information handled and/or returned by interface and abstract class methods are presented in Table 1. In addition to that, some class may contain their specific attributes, which are discussed below. Use this table along with the description below for information to be returned/handled by the method(s) within respective classes.

Table 1: Information about musical instruments

| Instrument | Make sound | Price | How to play | How to repair |
|---|---|---|---|---|
| **Drum** | Vibrating stretched membrane | $349.50 | By hitting the membrane with sticks | Replace the membrane |
| **Flute** | Guiding a stream of air | $74.99 | By blowing into the flute | *<No implementation: Flute can't be repaired>* |

| | | | | |
|---|---|---|---|---|
| **Harp** | Vibrating strings | $255.00 | By strumming the strings and peddling to adjust the string lengths | Replace the broken strings |
| **Piano** | Vibrating the soundboard | $725.00 | By hitting the keys that trigger hammers to hit the strings | Replace the broken strings or keys |
| **Violin** | Vibrating strings | $350.00 | By resting the violin on shoulder, plucking the strings bow and picking notes with fingers | Replace the broken bridge |
| **Xylophone** | Through resonators | $49.00 | By hitting bars with two mallets | Replace the broken keys |

Details about the classes and interfaces are below:

**Class *StringFamily*** [10 marks]: All *StringFamily* instruments e.g., *Harp*, *Violin* and *Piano* should have a number of strings. Use appropriate attribute inheritance to deal with this attribute between this superclass and its subclasses.

**Class *PercussionFamily*** [10 marks]: Assume all *PerecussionFamily* instruments e.g., *Xylophone*, *Drum* and *Piano* should have a number of keys. Use appropriate attribute inheritance to deal with this attribute between this superclass and its subclasses. In addition to the inherited attributes, class *Drum* should also have an attribute called 'diameter_in_inch' indicating the diameter of the drum surface in inch.

Note: As shown in the figure, class *Piano* inherits both *StringFamily* and *PercussionFamily* since pianos have both strings and keys.

**Class *WoodwindFamily*** [10 marks]: Every *WoodwindFamily* instrument has an attribute called 'made of' which will return 'made of wood', assuming all instrument in this family are made of wood. Also, instruments in this family, e.g., *Flute*, can not be fixed when broken or damaged, hence this class or its subclasses do not implement *Repairable* interface.

**Class *Order*** [7 marks] In addition to the classes shown in the UML in Figure 1, develop another class called *Order* to keep track of orders for musical instruments by different customers. Along with required accessors and mutators, the *Order* class should have following attributes at a minimum:

*o_id*: an id of the order.

*customer_name*: the name of the customer making the order.

*orders*: a list of musical instruments ordered by the customer.

*add_instrument(self, instrument)*: an instance method to be used to add an instrument (passed by the argument) to the *orders* list.

**Testing the implementation** [8 marks]: Create a tester class called *Tester_<Your First Name>* to test your implementation. In your tester class, create at least one object of each instrument as below:

- *drum = Drum(0, 20)* # this drum has 0 keys and the size of its diameter is 20. (Assume drum does not have any key.)
- *flute = Flute()* # this flute does not have any specific attribute, except the inherited ones.
- *harp = Harp(40)* # this harp has 40 strings
- *piano = Piano(88, 230)* # this piano has 88 keys and 230 strings
- *violin = Violin(4)* # this violin has 4 strings
- *xylophone = Xylophone(25)* # this xylophone has 25 keys

and provide the following outputs. [Do not hard code any data for any output, it should be obtained through invoking the respective method.]

**Outputs 1 – 6** [20 marks]**:**

Programmatically display the sound making, play method, repair method and the price for each instrument (except no repair method for flute) as shown in Outputs 1 to 6 below.

```
----------------------OUTPUT 1 ----------------------------
Details of Drum
Makes sound: vibrating stretched membrane
Play method: by hitting the membrane with sticks
Repair method: replace the membrane
The price: 349.5
----------------------OUTPUT 2 ----------------------------
Details of Fluet
Makes sound: guiding a stream of air
Play method: by blowing into the flute
The price: 74.99
----------------------OUTPUT 3 ----------------------------
Details of Harp
Makes sound: vibrating the string
Play method: by strumming the strings and peddling to adjust the string lengths
Repair method: replace the broken strings
The price: 255.99
----------------------OUTPUT 4 ----------------------------
Details of Piano
Makes sound: vibrating the string
Play method: hitting the keys that trigger hammers to hit the strings
Repair method: replace the broken strings or keys
The price: 725.0
```

```
----------------------OUTPUT 5 ----------------------------
Details of Violin
Makes sound: vibrating the string
Play method: resting the violin on shoulder, plucking the strings bow and picking notes with fingers
Repair method: replace the broken bridge
The price: 350.0
----------------------OUTPUT 6 ----------------------------
Details of Xylophone
Makes sound: through resonators
Play method: hitting barss with two mallets
Repair method: replace the broken keys
The price: 89.99
```

**Output 7** [10 marks]**:**

Create a list of instruments and add all the above instruments in it. Then for each instrument in the list, display only its special feature (e.g., number of keys for percussion family instruments, number of strings for string family instruments, diameter of a drum, and made of for woodwind family instruments.). See the output below. Again, do not hard code any output data, you must obtain the values programmatically through invoking appropriate method offered in the class.

```
----------------------OUTPUT 7 ----------------------------
SPECIAL FEATURES
Key count of Xylophone is: 25
Key count of Drum is: 0
The diameter of Drum is: 20 inch
String count of Harp is: 40
String count of Piano is: 88
Key count of Piano is: 230
Fluet is: Made of wood
String count of Violin is: 4
```

**Output 8** [10 marks]:

Use list's *sort()* method to sort the instruments in the instrument list in price ascending order. [**Hint**: you may need to implement *__ge__()* and *__lt__()* magic methods in an appropriate class to facilitate the sorting on the instrument list.] Display the name of the instrument and its price by walking through the sorted instrument list, as below.

```
----------------------OUTPUT 8 ----------------------------
All instuments in price ascending order:
Fluet price: 74.99
Xylophone price: 89.99
Harp price: 255.99
Drum price: 349.5
Violin price: 350.0
Piano price: 725.0
```

**Output 9** [15 marks]:

Create following two orders by two customers Bob and Alice, respectively:

*order_Bob = Order(111, 'Bob')* # id of the order = 111 and customer name = Bob

*order_Alice = Order(222, 'Alice')* # id of the order = 222 and customer name = Alice

Have *order_Bob* order for one of each percussion family instrument and *order_Alice* order for one of each string family instrument. [**Hint**: Walkthrough the instrument list and add percussion family instruments in the *orders* list of *order_Bob*, and string family instruments in the *orders* list of *order_Alice*. Make use of the *add_instrument(self, instrument)* method.]

Once the order list of Bob and Alice is populated, output the total price of all the instruments in each list, see the output below.

Create another order instance for a custom order where the order id should be the **last 3 digits of your Sheridan student number** and use **your first name** as the customer's name. Then, ask the user to enter a set of instruments that you want to order. Please see the output below as a guide to obtain user input for instrument choices. For example, in the output below, user ordered for a flute, a drum, a xylophone and a piano by typing the string *fdxp*. The program then extracted each letter from the input string to learn about the instruments ordered and create the order list. This is a guideline only, you can use your own way to obtain user choice for instruments.

Once the order list is created, output the total price of all the instruments ordered along with the names of instruments ordered (see the output below).

```
----------------------OUTPUT 9 -----------------------------
Order total for Bob (111): 1164.49
Order total for Alice (222): 1330.99
----------------------CUSTOM ORDER -----------------------------
Please enter your order by typing the letter for each instrument without any space:
d = Drum, f = Fluet, h = Harp, p = Piano, v = Violin, and X = Xylophone
fdxp
Syed ordered for ['Drum', 'Fluet', 'Piano', 'Xylophone']
Order total for Syed (999): 1239.48
```

```
----------------------CUSTOM ORDER -----------------------------
Please enter your order by typing the letter for each instrument without any space:
d = Drum, f = Fluet, h = Harp, p = Piano, v = Violin, and X = Xylophone
hdv
Syed ordered for ['Drum', 'Harp', 'Violin']
Order total for Syed (999): 955.49
```

**Important requirements:**

- Demonstrate industry standard design e.g., applying generalization or specialization of attributes when possible.
- All interfaces should be placed in a separate module named: *Interfaces_<Your First Name>*.
- The *MusicalInstrument* class should be implemented in a separate module named *Instruments_<Your First Name>.*
- Classes *StringFamily, Harp* and *Violin* should be implemented in a separate module named *StringFamily_<Your First Name>*.
- Classes *PercussionFamily*, *Drum* and *Xylophone* should be implemented in a separate module named *PercussionFamily_<Your First Name>*.

- Classes *WoodwindFamily* and *Fluet* should be implemented in a separate module named *WoodwindFamily_<Your First Name>*.
- Class *Piano* should be implemented in a separate module named *Piano_<Your First Name>*.
- Class *Order* should be implemented in a separate module named *OrderList_<Your First Name>*.
- Use a separate class called *Tester_<Your First Name>* to test the functionalities and operations (including input and output).
- Access or modification to any class attribute from outside of the class must be done through proper accessor or mutator.
- Add your signature (e.g., name, Sheridan student number) in each source code file as commented.
- Overall, follow industry standard design principle and Python code writing conventions and naming conventions for classes, interfaces, files, objects, etc. as specified in this assignment. *[At least 10% marks deduction for failing to properly follow these standard and conventions].*

## Submission Guideline:

- Create a document file with name <YourFirstName-YourID-A1>. The cover page of the document must include following information:
    - Assignment #: 1
    - Course: PROG23199
    - Your full name:
    - Your Sheridan Student Number:
    - Instructor's name: Syed Tanbeer
- Run your program and take screenshot for each of the above input/output scenarios (Outputs 1 - 9). Two separate custom order outputs for order of two different set of instruments. Screenshots must demonstrate all the requirements of the assignment and be readable in 100% zoom size (unreadable screenshots will not be marked). Paste the screenshots in the document file in sequence with reference number.
- Submit following files as a single zip file to the ''Submit Assignment 1" Dropbox available at Slate.

i. All the source code (.py) files.

ii. The document file (Word or pdf) containing screenshots showing outputs from sample runs of your simulation.

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

---: End of Assignment 1 : --