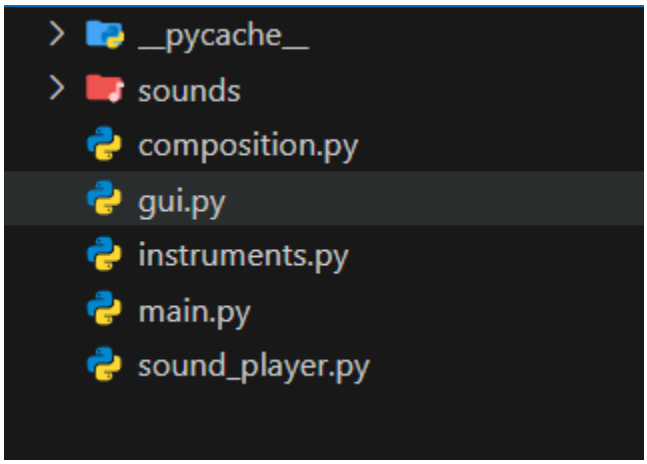
Balajadia, Kent Russel G.
3-BSCS-1
Intelligent System

Multiple Inheritance




```
import pygame

class Composition:
    def __init__(self, title):
        self.title = title
        self.notes = []  # Store notes and their corresponding
instruments

    def add_note(self, note, instrument):
        """Adds a note to the composition and plays it."""
        self.notes.append((note, instrument))  # Store note with
instrument object
        return instrument.play_note(note)  # Play using the correct
instrument

    def play_composition(self, bpm=120):
        """Plays the full composition at a fixed BPM."""
        note_duration = (60 / bpm)  # Convert BPM to seconds

        for note, instrument in self.notes:
```

```
            instrument.play_note(note)  # Play using stored instrument
            pygame.time.wait(int(note_duration * 1000))  # Wait for note
duration

    def save_to_file(self, filename):
        with open(filename, "w") as file:
            for note, instrument in self.notes:
                file.write(f"{note} - {instrument.name}\n")  # Store
instrument name

    def load_from_file(self, filename):
        with open(filename, "r") as file:
            self.notes = [line.strip().split(" - ") for line in file]
```

```
import tkinter as tk
from tkinter import ttk
from instruments import Piano, Guitar, Violin, Flute, Trumpet
from composition import Composition

# Initialize Tkinter
root = tk.Tk()
root.title("Music Composition Assistant")
root.geometry("400x500")

# Available notes and instruments
notes = ["C4", "C#4", "D4", "D#4", "E4", "F4", "F#4", "G4", "G#4", "A4",
"A#4", "B4", "C5"]
instruments = {
    "Piano": Piano(),
    "Guitar": Guitar(),
    "Violin": Violin(),
    "Flute": Flute(),
    "Trumpet": Trumpet()
}

# Note selection
tk.Label(root, text="Select Note:").pack()
note_var = tk.StringVar()
```

```
note_dropdown = ttk.Combobox(root, textvariable=note_var, values=notes,
state="readonly")
note_dropdown.pack()

# Instrument selection
tk.Label(root, text="Select Instrument:").pack()
instrument_var = tk.StringVar()
instrument_var.set("Piano")  # Default selection
instrument_dropdown = ttk.Combobox(root, textvariable=instrument_var,
values=list(instruments.keys()), state="readonly")
instrument_dropdown.pack()

# Tempo selection
tk.Label(root, text="Select Tempo (BPM):").pack()
tempo_var = tk.IntVar()
tempo_var.set(120)  # Default tempo
tempo_dropdown = ttk.Combobox(root, textvariable=tempo_var, values=[60,
90, 120, 150, 180], state="readonly")
tempo_dropdown.pack()

# Listbox for composition
composition_listbox = tk.Listbox(root)
composition_listbox.pack(fill=tk.BOTH, expand=True)

# Indicator for currently playing note
current_note_label = tk.Label(root, text="Now Playing: None",
font=("Arial", 12, "bold"), fg="red")
current_note_label.pack()

# Composition object
composition = Composition("My Song")

# Function to add note
def add_note():
    note = note_var.get()
    instrument_name = instrument_var.get()

    if note and instrument_name:
        instrument = instruments[instrument_name]  # Get instrument
```

```
object
        composition.add_note(note, instrument)  # Store instrument
object
        composition_listbox.insert(tk.END, f"{note} -
{instrument_name}")  # Update GUI

# Function to highlight the currently playing note
def highlight_note(index):
    composition_listbox.selection_clear(0, tk.END)  # Clear previous
selection
    composition_listbox.selection_set(index)  # Select the current note
    composition_listbox.activate(index)  # Highlight it
    composition_listbox.see(index)  # Ensure it's visible

# Function to reset highlight after playing
def reset_highlight(index):
    composition_listbox.selection_clear(index)

# Function to play entire composition at fixed tempo
def play_composition(index=0):
    if index < len(composition.notes):
        bpm = tempo_var.get()  # Get selected tempo
        note_duration = int((60 / bpm) * 1000)  # Convert BPM to
milliseconds

        note, instrument = composition.notes[index]

        # Update current playing note label
        current_note_label.config(text=f"Now Playing: {note} on
{instrument.name}")

        # Highlight the currently playing note
        highlight_note(index)

        # Play the note
        instrument.play_note(note)

        # Schedule the next note and reset highlight
        root.after(note_duration, lambda: reset_highlight(index))
```

```
        root.after(note_duration, play_composition, index + 1)
    else:
        current_note_label.config(text="Now Playing: None")  # Reset
after playing

# Function to remove selected note
def remove_selected():
    selected_index = composition_listbox.curselection()
    if selected_index:
        index = selected_index[0]
        composition_listbox.delete(index)  # Remove from GUI list
        del composition.notes[index]  # Remove from composition data

# Function to delete all notes
def delete_all():
    composition_listbox.delete(0, tk.END)  # Clear the listbox
    composition.notes.clear()  # Clear the composition data

# Frame for buttons (to arrange them horizontally)
button_frame = tk.Frame(root)
button_frame.pack(pady=10)

# Buttons inside the frame
add_note_button = tk.Button(button_frame, text="Add Note",
command=add_note)
add_note_button.pack(side=tk.LEFT, padx=5)

play_button = tk.Button(button_frame, text="Play Composition",
command=lambda: play_composition(0))
play_button.pack(side=tk.LEFT, padx=5)

remove_selected_button = tk.Button(button_frame, text="Remove",
command=remove_selected)
remove_selected_button.pack(side=tk.LEFT, padx=5)

delete_all_button = tk.Button(button_frame, text="Clear All",
command=delete_all)
delete_all_button.pack(side=tk.LEFT, padx=5)
```

```
root.mainloop()
```

```
from sound_player import SoundPlayer

class Instrument:
    def __init__(self, name, midi_program, sound_file):
        self.name = name
        self.midi_program = midi_program
        self.sound_player = SoundPlayer(sound_file)  # Pass the sound
file to SoundPlayer

    def play_note(self, note):
        print(f"{self.name} is playing {note}")
        self.sound_player.play_note(note)


class StringInstrument:
    def strum(self):
        return f"{self.name} is strumming chords!"

    def bow(self):
        return f"{self.name} is playing with a bow!"


class WindInstrument:
    def breathe_control(self):
        return f"{self.name} uses breath control!"


class Piano(Instrument):
    def __init__(self):
        super().__init__("Piano", 0, "sounds/piano_C4.wav")  # Specify
piano sound file


class Guitar(Instrument, StringInstrument):
    def __init__(self):
```

```
        super().__init__("Guitar", 24, "sounds/guitar_C4.wav")  #
Specify guitar sound file


class Violin(Instrument, StringInstrument):
    def __init__(self):
        super().__init__("Violin", 40, "sounds/violin_C4.wav")  #
Specify violin sound file


class Flute(Instrument, WindInstrument):
    def __init__(self):
        super().__init__("Flute", 73, "sounds/flute_C4.wav")  # Specify
flute sound file


class Trumpet(Instrument, WindInstrument):
    def __init__(self):
        super().__init__("Trumpet", 56, "sounds/trumpet_C4.wav")  #
Specify trumpet sound file
```

```
import pygame
import numpy as np

class SoundPlayer:
    NOTE_RATIOS = {
        "C4": 1.0, "C#4": 1.059, "D4": 1.122, "D#4": 1.189, "E4": 1.26,
        "F4": 1.335, "F#4": 1.414, "G4": 1.498, "G#4": 1.587, "A4":
1.682,
        "A#4": 1.782, "B4": 1.888, "C5": 2.0  # Octave higher
    }

    def __init__(self, sound_file):
        pygame.mixer.init()
        self.original_sound = pygame.mixer.Sound(sound_file)  # Load the
specified sound file

    def play_note(self, note):
```

```
        """Plays a note by resampling the original sound."""
        if note in self.NOTE_RATIOS:
            ratio = self.NOTE_RATIOS[note]
            original_freq = 44100
            new_freq = int(original_freq * ratio)

            sound = pygame.sndarray.array(self.original_sound)  # Get
sound array
            resampled_sound =
pygame.sndarray.make_sound(self.resample(sound, ratio))  # Resample
            resampled_sound.play()
        else:
            print(f"Note {note} not found")

    def resample(self, sound_array, ratio):
        """Resamples the sound array to match the new note frequency."""
        indices = np.round(np.arange(0, len(sound_array),
ratio)).astype(int)
        indices = indices[indices < len(sound_array)]  # Avoid
out-of-bounds error
        return sound_array[indices]
```

```
from gui import MusicApp
import tkinter as tk

if __name__ == "__main__":
    root = tk.Tk()
    app = MusicApp(root)
    root.mainloop()
```