JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Arbeitsprotokoll

A16 Music Player

Softwareetwicklung
4CHIT 2016/17

Martin Wölfer

Version 0.2
Begonnen am 24. April 2017
Beendet am 30. April 2017

Inhaltsverzeichnis

1 Einführung

Schreibe ein Programm, welches einen Music Player einerseits simuliert und andererseits implementiert!

1.1 Grundanforderungen

- Verwende die vorgegebenen Klassen:
  - main: Erzeugt entweder eine Mockup-Fabrik oder eine echte Fabrik
  - Musikstueck: Abstrakte Produkt-Klasse
  - MusikdatenbankFabrik: Abstrakte Fabrik mit Fabrik-Methode `lade_musik`
- Eine eigene Mockup-Klasse mockt die abspielen-Methode mit einer simplen Ausgabe (Mockup-Produkt), z.B. Sie hören den Titel XXX von YYY aus dem Album ZZZ
- Eine eigene Mockup-Klasse erzeugt einige beliebige Mockup-Produkte (Mockup-Fabrik)
- Eine eigene Klasse spielt die Musik ab (File-Produkt)
- Eine eigene Klasse erzeugt die echten Produkte, indem sie nach mp3-Files sucht und für jedes mp3-File ein Produkt erzeugt (File-Fabrik)
- Kommentare und Sphinx-Dokumentation
- Protokoll

Tipp: Verwende zum Abspielen die Library pyglet. Achtung: pyglet.app.run() endet nicht automatisch, sondern muss über ein Callback nach dem Song über pyglet.app.exit() wieder beendet werden!

1.2 Erweiterungen

- Recherchiere, wie aus Musikdateien automatisch Titel, Album und Interpret ausgelesen werden können
- Füge die Funktionalität deiner echten Produktfamilie hinzu, sodass die Informationen ebenfalls angezeigt werden können
- Baue eine (sehr simple) GUI für deinen Music Player

2 Ergebnisse

2.1 Vorgehensweise

2.1.1 MockUp

Zuerst wurden die Mockup-Klassen erstellt. Eine MockUp Klasse dient dazu, eine bestimmte Funktionalität vorzugeben um somit die Komposition des Programmes zu testen. Beispielweise, wenn eine Klasse normalerweise ein File abspielen würde, macht eine MockUp Klasse nichts anderes als so zu tun als würde es das File abspielen indem es in der Konsole ausgibt `File wird abgespielt`.

2.1.2 MusikStueckFabrik

Nachdem die simplen MockUp Klassen erstellt wurden, konnte man sich gleich an das tatsächliche Programm antasten. Der erste Schritt war, anstatt per Hand Lieder zu erstellen, ein gewünschtes Directory durchzugehen und dort alle Musik-Lieder heraus zu suchen.

Umgesetzt wurde dies, klassisch wie man es meistens in Python macht, mit `os.walk()` und einer Rekursion. Bei jedem File wird anschließlich nachgeschaut, ob die extension einem Musik-File gleicht, dann werden Meta-Daten ausgelesen und schließlich das MusikStueck erstellt.

Falls keine passenden Lieder gefunden wurden, wird eine Fehlernachricht ausgegeben

```
def lade_musik(self):
    """
    Fuegt der Playlistt MockupMusikstuecke hinzu welche automatisch ausgelesen werden

    :return: None
    """
    # iterate through given directory
    for dirname, subdirs, files in os.walk(self.dir):
        for filename in files:
            # get the file extension of each file
            extension = os.path.splitext(filename)[1][1:]
            # check if the extension is a music file
            if extension.lower() in ["mp3","wma", "wav","ra", "ram", "rm", "mid", "flac", "ogg"]:
                # get each mp3 file in this directory
                file = pyglet.media.load(dirname + os.path.sep + filename)

                # get each specific info needed
                songlaenge = file.duration
                songtitel = file.info.title.decode()
                songinterpret = file.info.author.decode()
                songalbum = file.info.album.decode()
                # append each song with the informations to the playlist and the gui object
                self.playlist.append(MusikstueckFile(file, songlaenge, songtitel, songinterpret,
                    songalbum, self.updatefunction))
            else:
                print("%s is not a music file!" % filename)
    if len(self.playlist) == 0:
        # if no music file was found, print out error message and exit program
        print("No suitable Files found. Please restart Program and choose another Folder")
        sys.exit(0)
```

2.1.3 MusikStueck

Der nächste Schritt war es das MusikStueckFile zu verändern, also jene Klasse die tatsächlich das File abspielt. Diese Klasse erbt von MusikStueck, und eigentlich muss nur die abstrakte Methode `abspielen()` überschrieben werden, aber weil noch zusätzliche Informationen benötigt werden, wurde dem Konstruktur noch 3 Parameter hinzugefügt:

```
def __init__(self, file, laenge, titel, interpret, album, update_function):
    """
    Adds 2 additional parameters to the super constructor

    :param file: Thile file object which gets played

    :param laenge: The length of the mp3 file in order to pause the playlist for this amount

    :param titel: Title of the song, read out of the mp3 file

    :param interpret: Artist of the song, read out of the mp3 file

    :param album: Album of the song, read out of the mp3 file

    :param updatefunction: function of the gui object to update the data
    """
    # call super constructor
    Musikstueck.__init__(self, titel, interpret, album)
    # set the 3 additional parameters to class attributes
    self.file = file
    self.laenge = laenge
    self.update_function = update_function
```

In der `abspielen()` methode wird lediglich das pyglet.media File abgespielt, die Methode aufgerufen welche im Gui die Daten ändert und gewartet bis das Lied zu Ende ist.

```
def abspielen(self):
    """
    Play the file object given and update information

    In order for the playlist not to play every song at the same time, everytime a song is played, time.sleep for the length of the song

    :return: None
    """
    self.file.play()
    self.update_function(str(self.interpret), str(self.titel), str(self.album))
    time.sleep(self.laenge)
```

2.2 Probleme

2.2.1 avbin.dll

pyglet verwendet um MusikFiles abzuspielen, abgesehen von .wma, eine externe `dynamic link library` die `avbin.dll` heißt. Diese kann man im Internet überall finden, es war jedoch eine Herausforderung herauzufinden welche .dll man herunterladen muss und wo sie hingehört oder wie man sie in das System einfügt.

Wie es für mich funktioniert hat, war indem ich eine 32bit Version heruntergeladen habe, namens `avbin.dll` (ohne 64) und diese in `C:\Windows\System` verschoben habe. Somit konnte pyglet die Library verwenden.

2.2.2 Gui Freezing

Wie erstmals das Gui hinzugefügt wurde, hat das Gui nicht reagiert und hat immer zu einem Absturz des Programmes geführt. Problem war folgendes:

In meiner `MusikStueckFile` Klasse, wo das tatsächliche File abgespielt wird, ist ein `time.sleep()` eingebaut, damit während einem Song gewartet wird damit nicht alle Songs in der Playlist gleichzeitig abgespielt werden. Dies hat dazu geführt dass der `Main Thread` immer in einem Wartezustand ist, und somit das Gui auch wartet und somit nicht reagiert.

Lösung zu diesem Problem war, in der Main Klasse eine Thread-Klasse zu erzeugen, welche die Factory initialisiert und startet. Dies führt dazu dass die GUI und der Factory-Prozess, in welchem gewartet wird, in verschieden Threads realisiert ist und so unabhängig voneinander agieren können.

2.2.3 Programm Beenden

Wenn die GUI geschlossen wurde, ist im Hintergrund trotzdem das Programm weitergelaufen. Dies war ein Problem infolge der Lösung des anderen Problems, weil ja diese Prozesse nun in verschiedenen Threads ablaufen.

Dieses Problem konnte mithilfe von `Pierre Rieger` leicht behoben werden, indem im `FactoryThread` das Klassen-Attribut `self.daemon` auf True gesetzt wurde. Erklärung wie es im Source-Code steht:

```
1  class FactoryThread(Thread):
       # factorythread for starting the factory process aka playing the music
3      # has to implemented as thread and can't be written plain in the main thread because
       # else the gui freezes
5      def __init__(self,dir,update_function):
           """
7          Calls super constructor and initializes class attributes

9          if self.deamon is set to True, the thread automatically becomes a 'Deamon-Thread'
           this basically means that if no other Threads are left, this Deamon Threads also automatically
11         closes itself, which is incredibly helpful because the gui Thread gets terminated by closing the
               window
           but the factory_thread will always keep on running unless its a daemon thread
13
           :param dir: directory with which the factory gets initialized
15
           :param update_function:
17         """
           Thread.__init__(self)
19         self.dir = dir
           self.update_function = update_function
21         self.daemon = True
```