```python
from abc import ABC, abstractmethod
import json
import re

class Catalog:
    """
    A class that simulates a library catalog and its functionality to track available items in the library.
    """

    def __init__(self, dataPath):
        """
        Initializes a Catalog object.

        Parameters:
            dataPath: The path to a json file containing information about library items
        """
        self.__dataPath = dataPath
        with open(dataPath) as file:
            self.__data = json.load(file)

        self.__items = []
        self.__update()

    @staticmethod
    def convert(data):
        """
        Returns a formatted string containing information of all items in list of data dictionary.

        Parameters:
            data (list): List of items whose information is in dictionaries

        Returns:
            res (str): Formatted string of data
        """
        res = []
        for d in data:
            r = ""
            for i in d:
                r += f"{i}: {d[i]}\n"
            res.append(r)
        res = "\n".join(res)
        return res

    def search(self, keyword, field=None):
        """
        Finds all items that matches with the provided keyword.

        Parameters:
            keyword (str): The keyword to search for
            field (str): The field in which the function searches. If None, the function will search all fields

        Returns:
            results (list): List of all library items that matches the keyword
        """
        if keyword == "":
            raise ValueError("Invalid Value.")

        results = []

        for item in self.__items:
            information = item.locate()
            foundDF = False
            if field == "Contributor":
                for t in item.getContribTypes():
                    if re.search(keyword, information[t], flags=re.IGNORECASE) is not None:
                        information[t] = re.sub(keyword, self.__f, information[t], flags=re.IGNORECASE)
                        foundDF = True
            elif field is not None:
                if re.search(keyword, information[field], flags=re.IGNORECASE) is not None:
                    information[field] = re.sub(keyword, self.__f, information[field], flags=re.IGNORECASE)
                    foundDF = True
            else:
                for i in information:
                    if re.search(keyword, information[i], flags=re.IGNORECASE) is not None:
                        information[i] = re.sub(keyword, self.__f, information[i], flags=re.IGNORECASE)
                        foundDF = True

            if foundDF:
                results.append(information)

        if len(results) != 0:
            results.sort(key=lambda x: (x["Type"], x["Title"]))

        return results

    def getItems(self):
        """
        Returns all library items sorted by type.

        Returns:
            results (list): A list containing all library items sorted by type. In which:
                results[0]: A list of Book items
                results[1]: A list of CD items
                results[2]: A list of DVD items
                results[3]: A list of Magazine items
        """
        results = [[], [], [], []]

        for item in self.__items:
            res = item.locate()
            if res["Type"] == "Book":
                del res["Type"]
                results[0].append(res)
            elif res["Type"] == "CD":
                del res["Type"]
                results[1].append(res)
            elif res["Type"] == "DVD":
                del res["Type"]
                results[2].append(res)
            elif res["Type"] == "Magazine":
                del res["Type"]
                results[3].append(res)

        for r in results:
            r.sort(key = lambda x: x["Title"])
        return results

    def addItem(self, data):
        """
        Adds new item to library json file.

        Parameters:
            data (dict): A dictionary containing information of the item
        """

        self.__data.extend(data)
        updated_json = json.dumps(self.__data, indent=4)

        with open(self.__dataPath, 'w') as file:
            file.write(updated_json)

        self.__update()

    def deleteItems(self, keyword):
        """
        Deletes items that match with keyword by title.

        Parameters:
            keyword (str): keyword to search for
        """
        newData = []
        for i in range(len(self.__data)):
            if keyword not in self.__data[i]["Title"]:
                newData.append(self.__data[i])

        self.__data = newData

        updated_json = json.dumps(self.__data, indent=4)

        with open(self.__dataPath, 'w') as file:
            file.write(updated_json)

        self.__update()


    def __update(self):
        """
        Updates items in the library.
        """
        items = []
        for d in self.__data:
            if d["Type"] == "Book":
                item = Book(d)
            elif d["Type"] == "CD":
                item = CD(d)
            elif d["Type"] == "DVD":
                item = DVD(d)
            else:
                item = Magazine(d)
            items.append(item)
        self.__items = items

    def __f(self, match):
        """
        Wraps matched strings with green background.
        """
        return "\x1b[6;30;42m" + match.group(0)[0] + match.group(0)[1:] + "\x1b[0m"


class LibraryItem(ABC):

    """
    An abstract class that contains information about an item in the library.
    The item can be a Book, a CD, a DVD, or a Magazine.
    """
    def __init__(self, data):
        """
        A customized constructor for derived class.

        Parameters:
            data (dict): A dictionary containing information of the item
        """
        self._title = data["Title"]
        self._UPC = data["UPC"]
        self._contributors = []
        for d in data["Contributor"]:
            self._contributors.append(ContributorWithType(d, data["Contributor"][d]))

    @abstractmethod
    def locate(self) -> dict:
        """
        An abstract method that locates the item in the library inventory.

        Returns:
            A dictionary containing information of the item.
        """

    def getContribTypes(self):
        """
        Returns a list of contributors' types.
        """
        return [c.getType() for c in self._contributors]

class ContributorWithType:
    """
    A class containing name and type of contributors of a library item.
    """
    def __init__(self, type, contributor):
        """
        Initializes a ContributorWithType object.

        Parameters:
            type (str): Type of contributor (Author, Director, Actor, ...)
            contributor (str): Name of contributors, seperated by ", "
        """
        self.__type = type
        self.__contributor = [Contributor(c) for c in contributor.split(", ")]

    def getContributors(self):
        """
        Returns a list containing name of all contributors in the class.
        """
        return [c.getName() for c in self.__contributor]

    def getType(self):
        return self.__type


class Contributor:
    """
    A class containing name of the contributor.
    """
    def __init__(self, name):
        """
        Initializes a Contributor object.

        Parameters:
            name (str): Name of the contributor
        """
        self.__name = name

    def getName(self):
        return self.__name

class Book(LibraryItem):
    """
    A class containing information of book-type item in library.
    """
    def __init__(self, data):
        """
        Initializes a Book object.

        Parameters:
            data (dict): Dictionary of information of the book
        """
        super().__init__(data)
        self.__subject = data["Subject"]
        self.__ISBN = data["ISBN"]
        self.__DDS = data["DDS"]
    def locate(self):
        """
        Overwrites locate method in LibraryItem class.
        """
        contribs = {}
        for c in self._contributors:
            contribs.update({c.getType(): ", ".join(c.getContributors())})

        res = {
            "Title": self._title,
            "Type": "Book"
        }
        res.update(contribs)
        res.update({
            "Subject": self.__subject,
            "ISBN": self.__ISBN,
            "DDS": self.__DDS,
            "UPC": self._UPC
        })

        return res

class CD(LibraryItem):
    """
    A class containing information of CD-type item in library.
    """
    def __init__(self, data):
        """
        Initializes a CD object

        Parameters:
            data (dict): Dictionary of information of the CD
        """
        super().__init__(data)
        self.__genre = data["Genre"]
        self.__ASIN = data["ASIN"]

    def locate(self):
        """
        Overwrites Locate method in LibraryItem class.
        """
        contribs = {}
        for c in self._contributors:
            contribs.update({c.getType(): ", ".join(c.getContributors())})

        res = {
            "Title": self._title,
            "Type": "CD"
        }
        res.update(contribs)
        res.update({
            "Genre": self.__genre,
            "ASIN": self.__ASIN,
            "UPC": self._UPC
        })

        return res

class DVD(LibraryItem):
    """
    A class containing information of DVD-type item in library.
    """
    def __init__(self, data):
        """
        Initializes a DVD object

        Parameters:
            data (dict): Dictionary of information of the DVD
        """
        super().__init__(data)
        self.__genre = data["Genre"]
        self.__ASIN = data["ASIN"]

    def locate(self):
        """
        Overwrites Locate method in LibraryItem class.
        """
        contribs = {}
        for c in self._contributors:
            contribs.update({c.getType(): ", ".join(c.getContributors())})

        res = {
            "Title": self._title,
            "Type": "DVD"
        }
        res.update(contribs)
        res.update({
            "Genre": self.__genre,
            "ASIN": self.__ASIN,
            "UPC": self._UPC
        })

        return res

class Magazine(LibraryItem):
    """
    A class containing information of magazine-type item in library.
    """
    def __init__(self, data):
        """
        Initializes a Magazine object

        Parameters:
            data (dict): Dictionary of information of the magazine
        """
        super().__init__(data)
        self.__volume = data["Volume"]
        self.__issue = data["Issue"]

    def locate(self):
        """
        Overwrites Locate method in LibraryItem class.
        """
        contribs = {}
        for c in self._contributors:
            contribs.update({c.getType(): ", ".join(c.getContributors())})

        res = {
            "Title": self._title,
            "Type": "Magazine",
        }
        res.update(contribs)
        res.update({
            "Volume": self.__volume,
            "Issue": self.__issue,
            "UPC": self._UPC
        })

        return res
```

```python
import re
from os import name, system

class Menu:
    """
    A class that creates interactive beautiful menus.
    """
    __ranges = [
        {"from": "\u3300", "to": "\u33ff"},         # compatibility ideographs
        {"from": "\ufe30", "to": "\ufe4f"},         # compatibility ideographs
        {"from": "\uf900", "to": "\ufaff"},         # compatibility ideographs
        {"from": "\U0002F800", "to": "\U0002fa1f"}, # compatibility ideographs
        {'from': "\u3040", 'to': "\u309f"},         # Japanese Hiragana
        {"from": "\u30a0", "to": "\u30ff"},         # Japanese Katakana
        {"from": "\u2e80", "to": "\u2eff"},         # cjk radicals supplement
        {"from": "\u4e00", "to": "\u9fff"},
        {"from": "\u3400", "to": "\u4dbf"},
        {"from": "\U00020000", "to": "\U0002a6df"},
        {"from": "\U0002a700", "to": "\U0002b73f"},
        {"from": "\U0002b740", "to": "\U0002b81f"},
        {"from": "\U0002b820", "to": "\U0002ceaf"}  # included as of Unicode 8.0
    ]

    def __init__(self, maxLength, title="", sep=" "):
        """
        Initializes a Menu object.

        Parameters:
            maxLength (int): the number of maximum characters in one line
            title (str): title of the menu
            sep (str): a character used to seperate the title from its content
        """
        self.__maxLength = maxLength
        self.__title = title
        self.__sep = "|" + sep*(maxLength-2) + "|"
        self.__lines = []
        self.__counter = 0
        self.__border = "+" + "-"*(maxLength-2) + "+"
        self.__checkFormat(title)
        self.__content = []

    def addLines(self, text=""):
        """
        Add one or more lines to the menu.

        Parameters:
            text (str): line(s) to add
        """
        map(self.__checkFormat,text.split("\n"))
        self.__lines.extend([line for line in text.split("\n")])

    def addList(self, text):
        """
        Add a numbered list formatted line to the menu.

        Parameters:
            text (str): content of the list
        """
        self.__counter += 1

        line = f"{self.__counter}. {text}"
        self.__checkFormat(line)
        self.__lines.append(line)

    def setTitle(self, title):
        """
        Changes the title of the menu.

        Parameters:
            title (str): new title of the menu.
        """
        self.__title = title

    def show(self):
        """
        Print the menu to the screen.
        """
        Menu.clear()
        self.__format()
        print(self.__border)
        for c in self.__content:
            print(c)
        print(self.__border)

    @staticmethod
    def getChoice(choices):
        """
        Returns an integer of user's input if it is in a given range, else -1.

        Parameters:
            choices (int): max value in range. The range will be [1, choices]
        """
        key = input(f"Enter a number (1-{choices}): ")
        return int(key) if key in [*map(str, range(1,choices+1))] else -1

    @staticmethod
    def getKeyLog(text, empty=True):
        """
        Returns user's input.

        Parameters:
            text (str): the message indicating that the program is waiting for user's input
            empty (bool): whether the program accepts empty input or not. Default=True
        """
        inp = input(f"{text}: ")
        if not empty and inp=="":
            raise ValueError("Cannot enter empty string.")
        return inp
    @staticmethod
    def clear():
        """
        A function that clears the console screen.
        """
        if name == "nt":
            _ = system("cls")
        else:
            _ = system("clear")

    @staticmethod
    def logError(error):
        """
        Logs error to screen.

        Parameters:
            error (str): Error message to be logged
        """
        Menu.clear()
        print(f"Error: {error}")
        input()

    @staticmethod
    def __calcLen(line):
        """
        Calculate the length of the line. This function covers the situations in which the line contains special characters.
        """
        pattern = []
        for r in Menu.__ranges:
            pattern.append("["+r["from"]+"-"+r["to"]+"]")
        pattern = re.compile("|".join(pattern))
        res = re.findall(pattern, line)
        return len(line) - line.count("\x1b[6;30;42m")*14 + len(res)

    def __format(self):
        """
        Formats the whole menu content.
        """
        self.__content = []
        anchor = (self.__maxLength-2)//2 - len(self.__title)//2
        self.__content.append("|" + " "*anchor + self.__title + " "*(self.__maxLength-2-anchor-len(self.__title)) + "|")
        if len(self.__lines) != 0:
            self.__content.append(self.__sep)
        for i in range(len(self.__lines)):
            self.__content.append("|" + " " + self.__lines[i] + " "*(self.__maxLength-3-Menu.__calcLen(self.__lines[i])) + "|")

    def __checkFormat(self, line):
        """
        Checks if the length of the line surpasses provided max length. If fails, raise error.

        Parameters:
            line (str): the line to be checked
        """
        if Menu.__calcLen(line) > self.__maxLength-2:
            raise ValueError("Line too long")
```

```python
import os
import json
from sys import exit
from catalog import Catalog
from menu import Menu

def addFunc():
    results = []
    res = {}
    count = 0
    while True:
        addMenu = Menu(100, "Add item", "-")
        addMenu.show()
        type = Menu.getKeyLog("Enter item type(Book, CD, DVD, Magazine) or !q to exit")
        if type == "!q":
            break
        if type not in ["Book", "CD", "DVD", "Magazine"]:
            Menu.logError("Wrong type")
            continue
        addMenu.addLines(f"Type: {type}")
        addMenu.show()
        res = {}
        inp = Menu.getKeyLog("Enter title", False)
        res.update({"Title": inp})
        res.update({"Type": type})
        addMenu.addLines(f"Title: {inp}")
        addMenu.show()
        addMenu.addLines("Contributor:")
        contribs = {}
        while True:
            addMenu.show()
            contribType = Menu.getKeyLog("Enter contributor's type (press !q to continue)", False)
            if contribType == "!q":
                if len(contribs)==0:
                    Menu.logError("Must have at least one contributor")
                    continue
                break
            inp = Menu.getKeyLog("Enter contributors' names (seperated by ', ')", False)
            contribs.update({contribType: inp})
            addMenu.addLines(f"    {contribType}: {inp}")
            addMenu.show()
        res.update({"Contributor": contribs})
        if type == "Book":
            inp = Menu.getKeyLog("Enter subject", False)
            res.update({"Subject": inp})
            addMenu.addLines(f"Subject: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter ISBN", False)
            res.update({"ISBN": inp})
            addMenu.addLines(f"ISBN: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter DDS", False)
            res.update({"DDS": inp})
            addMenu.addLines(f"ISBN: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter UPC", False)
            res.update({"UPC": inp})
            addMenu.addLines(f"UPC: {inp}")
            addMenu.show()
        elif type == "CD" or type == "DVD":
            inp = Menu.getKeyLog("Enter genre", False)
            res.update({"Genre": inp})
            addMenu.addLines(f"Genre: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter ASIN", False)
            res.update({"ASIN": inp})
            addMenu.addLines(f"ASIN: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter UPC", False)
            res.update({"UPC": inp})
            addMenu.addLines(f"UPC: {inp}")
            addMenu.show()
        else:
            inp = Menu.getKeyLog("Enter volume", False)
            res.update({"Volume": inp})
            addMenu.addLines(f"Volume: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter issue", False)
            res.update({"Issue": inp})
            addMenu.addLines(f"Issue: {inp}")
            addMenu.show()
            inp = Menu.getKeyLog("Enter UPC", False)
            res.update({"UPC": inp})
            addMenu.addLines(f"UPC: {inp}")
            addMenu.show()
        count += 1
    if res != {}:
        results.append(res)
    addMenu.addLines()
    addMenu.addLines(f"Added {count} items to library.")
    addMenu.show()
    Menu.getKeyLog("Press Enter to continue")
    return results

def main():
    createMenu = Menu(100, "Library Catalog (ver 1.0)")
    createMenu.addLines("A catalog that helps you find what you need in the library.")
    createMenu.addLines()
    createMenu.addList("Create a new Catalog")
    createMenu.addList("Import Catalog from json file")
    createMenu.addList("Quit")

    fileMenu = Menu(100, "Create a new Catalog")

    mainMenu = Menu(100, "Welcome to Library Catalog", "-")
    mainMenu.addList("Search for items with a keyword")
    mainMenu.addList("List all items in the library")
    mainMenu.addList("Add items")
    mainMenu.addList("Delete items")
    mainMenu.addList("Quit")

    searchMenu = Menu(100, "Search for items with a keyword", "-")
    searchMenu.addList("Search by title")
    searchMenu.addList("Search by contributors")
    searchMenu.addList("Search by UPC")
    searchMenu.addList("Quit")

    searchByMenu = Menu(100, sep="-")



    fileName = ""
    while True:
        createMenu.show()
        choice = Menu.getChoice(3)
        if choice == -1:
            Menu.logError("Invalid value.")
            continue
        if choice == 1:
            fileMenu.show()
            fileName = fileMenu.getKeyLog("Enter file name")
            fileName = fileName if ".json" in fileName else fileName + ".json"
            if os.path.isfile(fileName):
                choice = Menu.getKeyLog("File exists. Doing this will delete all content of the file. Continue? (y/n)")
                if choice.lower() != "y":
                    Menu.clear()
                    Menu.getKeyLog("Aborted. Press Enter to continue")
                    continue
            res = []
            while True:
                res = addFunc()
                if len(res) == 0:
                    Menu.logError("You have to add at least 1 item to the library.")
                    continue
                break
            data = json.dumps(res, indent=4)
            with open(fileName, "w") as newFile:
                newFile.write(data)
            ctl = Catalog(fileName)
        elif choice == 2:
            fileMenu.setTitle("Import Catalog from json file")
            fileMenu.show()
            fileName = fileMenu.getKeyLog("Enter file name")
            fileName = fileName if ".json" in fileName else fileName + ".json"
            if not os.path.isfile(fileName):
                Menu.logError("File not found.")
                continue
        elif choice == 3:
            Menu.clear()
            exit()
        break

    while True:
        ctl = Catalog(fileName)
        mainMenu.show()
        choice = Menu.getChoice(5)
        if choice == -1:
            Menu.logError("Invalid value")
        if choice == 1:
            while True:
                resultMenu = Menu(100, sep="-")
                searchMenu.show()
                choice = Menu.getChoice(4)
                if choice == -1:
                    Menu.logError("Invalid value")
                    continue
                if choice == 1:
                    searchByMenu.setTitle("Search by title")
                    searchByMenu.show()
                    res = ctl.search(Menu.getKeyLog("Enter a keyword"), "Title")
                    resultMenu.setTitle("Found " + str(len(res)) + " items")
                    resultMenu.addLines(Catalog.convert(res))
                    resultMenu.show()
                    Menu.getKeyLog("Press Enter to continue")
                elif choice == 2:
                    searchByMenu.setTitle("Search by contributors")
                    searchByMenu.show()
                    res = ctl.search(Menu.getKeyLog("Enter a keyword"), "Contributor")
                    resultMenu.setTitle("Found " + str(len(res)) + " items")
                    resultMenu.addLines(Catalog.convert(res))
                    resultMenu.show()
                    Menu.getKeyLog("Press Enter to continue")
                elif choice == 3:
                    searchByMenu.setTitle("Search by UPC")
                    searchByMenu.show()
                    res = ctl.search(Menu.getKeyLog("Enter a keyword"), "UPC")
                    resultMenu.setTitle("Found " + str(len(res)) + " items")
                    resultMenu.addLines(Catalog.convert(res))
                    resultMenu.show()
                    Menu.getKeyLog("Press Enter to continue")
                else:
                    break

        elif choice == 2:
            res = ctl.getItems()
            menus = [Menu(100, "Book: "+str(len(res[0]))+" items", "-"), Menu(100, "CD: "+str(len(res[1]))+" items", "-"), Menu
            for i, menu in enumerate(menus):
                menu.addLines(Catalog.convert(res[i]))
                menu.show()
                Menu.getKeyLog("Press Enter to continue")
        elif choice == 3:
            res = addFunc()
            ctl.addItem(res)
        elif choice == 4:
            deleteMenu = Menu(100, "Delete item", "-")
            deleteMenu.show()
            keyword = Menu.getKeyLog("Enter the title of the item you want to delete")
            matches = ctl.search(keyword, "Title")
            delItems = len(matches)
            matches = Catalog.convert(matches)
            if matches == "":
                deleteMenu.addLines("Found 0 items...")
                deleteMenu.show()
                Menu.getKeyLog("Press Enter to continue")
                continue
            deleteMenu.addLines(matches)
            deleteMenu.show()
            choice = Menu.getKeyLog("Are you sure you want to delete these items? (y/n)")
            if choice.lower() == "y":
                ctl.deleteItems(keyword)
                deleteMenu.addLines(f"Delete {delItems} items.")
                deleteMenu.show()
                Menu.getKeyLog("Press Enter to continue")
            else:
                Menu.getKeyLog("Aborted. Press Enter to continue")
        elif choice == 5:
            Menu.clear()
            exit()

if __name__ == "__main__":
    main()
```