6. Class - Answer

Initial Commit: 2020.08.27
Private lesson (OOP Basics with Python, August 2020)
본 답안들은 제시된 문제를 푸는 방법 중 하나일 뿐입니다.

(1-1) 디지털미디어고등학교의 교내 도서관은 문학 도서를 1번 서재에, 사회 도서를 2번 서재에, 과학 도서를 3번 서재에 보관하고, 장르에 관계없이 페이지 수가 1000이 넘어가는 책은 4번 서재에 배치합니다. 그런데 이러한 규칙을 잘 모르는 학생들이 반납한 책을 아무 곳에나 꽂아두는 바람에 도서 정리에 어려움이 발생하고 있습니다. 이 문제를 해결하기 위해 학생들의 용이한 도서 정리를 위한 프로그램을 만들고자 합니다.

아래의 지시를 따라서 프로토타입을 구현해주세요.

[Book 클래스 필드]
name - 도서명
genre - 장르
page - 페이지 수
isBorrowed - 대출 여부. 이 속성은 생성자에서 파라미터로 초기화되지 않고 기본값 False를 가집니다.

[Book 클래스 메서드]
getPosition() - 해당 도서가 배치되어야 하는 서재 번호를 return합니다.
borrow() - 대출 여부를 True로 설정하고 적절한 메시지를 출력합니다. (실행 예시 참고)
handIn() - 대출 여부를 False로 설정하고 해당 도서가 배치되어야 하는 서재를 안내합니다. (실행 예시 참고)

[프로그램 테스트]
다음의 세 인스턴스 Demian (Literacy/390p), The Third Wave (Social/366p), University Physics (Science/1562p) 를 생성합니다.
세 도서를 대출한 후 반납하여 적절한 서재를 안내하는지 테스트하세요.
테스트 결과는 실행 예시와 같아야 합니다.

```
class Book():
 def __init__(self, name, genre, page):
  self.name = name
  self.genre = genre
  self.page = page
  self.isBorrowed = False

 def getPosition(self):
```

```
  if self.page >= 1000:
   return 4
  elif self.genre == 'Literacy':
   return 1
  elif self.genre == 'Social':
   return 2
  elif self.genre == 'Science':
   return 3
  return 0

 def borrow(self):
  self.isBorrowed = True
  print('You borrowed', self.name + '. Enjoy your book!')

 def handIn(self):
  self.isBorrowed = False
  print('You returned', self.name + '. Please put it in the shelf #' +
str(self.getPosition()) + '.')

bookA = Book('Demian', 'Literacy', 390)
bookB = Book('The Third Wave', 'Social', 366)
bookC = Book('University Physics', 'Science', 1562)

bookA.borrow()
bookB.borrow()
bookC.borrow()

bookA.handIn()
bookB.handIn()
bookC.handIn()
```

(1-2) 1-1번 프로그램에 반납일 및 연체 여부를 안내하는 기능을 추가하고자 합니다. 도서관 정책 상으로는 대출한 도서는 7일 이내에 반납해야 합니다.
아래의 수정 사항을 따라서 프로그램을 보완해주세요. (2월의 마지막 날은 항상 28일이라고 가정하며, 모든 날짜는 정수로 표현하는 것으로 합니다; 예를 들어 5월 7일은 507, 12월 31일은 1231입니다.)

[수정 사항: Book 클래스 필드]
borrowDate - 대출 일자. 이 속성은 생성자에서 파라미터로 초기화되지 않고 기본값 0을 가집니다.

[수정 사항: Book 클래스 메서드]

borrow(date) - 대출 일자를 date 파라미터로 받고, 반납 기한을 안내합니다. (실행 예시 참고)

handIn(date) - 반납 일자를 date 파라미터로 받아 연체 여부를 표시한 후, 해당 도서가 배치되어야 하는 서재를 안내합니다. (실행 예시 참고)

[프로그램 테스트]

테스트 대상 인스턴스는 위 1-1번 문제와 같습니다.
Demian 도서를 8월 12일에 대출하고 8월 16일에 반납합니다.
The Third Wave 도서를 9월 27일에 대출하고 10월 8일에 반납합니다.
University Physics 도서를 10월 25일에 대출하고 11월 1일에 반납힙니다.
테스트 결과는 실행 예시와 같아야 합니다.

```
lastDay = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

class Book():
 def __init__(self, name, genre, page):
  self.name = name
  self.genre = genre
  self.page = page
  self.isBorrowed = False
  self.borrowDate = 0

 def getPosition(self):
  if self.page >= 1000:
   return 4
  elif self.genre == 'Literacy':
   return 1
  elif self.genre == 'Social':
   return 2
  elif self.genre == 'Science':
   return 3
  return 0

 def borrow(self, date):
  self.isBorrowed = True
  self.borrowDate = date
  due = date + 7
  if due % 100 > lastDay[due // 100 - 1]:
   due = due + 100 - lastDay[due // 100 - 1]
```

```
    if due >= 1300:
     due = 100 + (due % 100)
   print('You borrowed', self.name + '. Due date:', str(due // 100) + '.' +
str(due % 100))

 def handIn(self, date):
  self.isBorrowed = False
  due = self.borrowDate + 7
  if due % 100 > lastDay[due // 100 - 1]:
   due = due + 100 - lastDay[due // 100 - 1]
  if date <= due:
   print('You returned', self.name, 'on time!', 'Please put it in the shelf #'
+ str(self.getPosition()) + '.')
  else:
   print('You returned', self.name, 'after the due date...', 'Please put it in
the shelf #' + str(self.getPosition()) + '.')

bookA = Book('Demian', 'Literacy', 390)
bookB = Book('The Third Wave', 'Social', 366)
bookC = Book('University Physics', 'Science', 1562)

bookA.borrow(812)
bookA.handIn(816)

bookB.borrow(927)
bookB.handIn(1008)

bookC.borrow(1025)
bookC.handIn(1101)
```

(2-1) 어떤 국내 영어학원의 수업반은 National, International, Universal의 3개 반으로 구성되어있고, 이 중 Universal 수강생을 제외한 모든 학생들은 매달 말에 Monthly Test를 봐서 90점 이상의 고득점을 받은 경우 더 높은 반을 수강할 수 있게 됩니다. 단, 14세 미만의 학생은 International반을 수강할 수 없고, 16세 미만의 학생은 Universal반을 수강할 수 없습니다.

규칙이 크게 어렵진 않지만, 시험 주기가 짧기도 하고 학원을 다니는 학생들이 많아지면서 정보 관리에 실수가 나타나는 등 어려운 점이 많습니다. 이러한 문제를 해결하기 위해 학생 정보를 프로그램으로 관리하고자 합니다.

아래의 지시를 따라서 프로토타입을 구현해주세요.

[Student 클래스 필드]
name - 학생 이름
age - 학생 나이
grade - 수강반 (National / International / Universal)
recentScore - 시험 점수. 이 속성은 생성자에서 파라미터로 초기화되지 않고 기본값 0을 가집니다.

[Student 클래스 메서드]
goNextGrade() - 수강반을 한 단계 올리고 적절한 메시지를 출력합니다. (실행 예시 참고)
takeTest(score) - 시험 점수를 score 파라미터로 받아 저장하고 적절한 메시지를 출력합니다. 더 높은 반을 수강할 수 있는지 체크합니다. (실행 예시 참고)

[프로그램 테스트]
다음의 세 인스턴스 Charlie(16/International), Brice(16/Universal), Natalie(13/National)를 생성합니다. 세 학생의 시험 점수를 96, 91, 92점으로 입력하고 프로그램이 규칙에 맞게 작동하는지 테스트합니다. 테스트 결과는 실행 예시와 같아야 합니다.

```
class Student():
 def __init__(self, name, age, grade):
  self.name = name
  self.age = age
  self.grade = grade
  self.recentScore = 0

 def goNextGrade(self):
  if self.grade == 'National':
   self.grade = 'International'
  elif self.grade == 'International':
   self.grade = 'Universal'
  print('Now', self.name, 'is in', self.grade, 'class.')

 def takeTest(self, score):
  self.recentScore = score
  print(self.name, 'got', score, 'in this test!')
  if score >= 90:
   if self.grade == 'National' and self.age >= 14:
    self.goNextGrade()
   elif self.grade == 'International' and self.age >= 16:
    self.goNextGrade()

studentA = Student('Charlie', 16, 'International')
```

```
studentB = Student('Brice', 16, 'Universal')
studentC = Student('Natalie', 13, 'National')

studentA.takeTest(96)
studentB.takeTest(91)
studentC.takeTest(92)
```

(2-2) 2-1번 프로그램에서 구현한 Student 클래스를 이용해 학생 정보 데이터베이스를 만들고자 합니다. 아래의 지시를 따라서 데이터베이스를 구현해주세요.

[수정 사항: Student 클래스 메서드]
toString() - 학생 정보를 적절한 문자열로 바꾸어 return합니다. (실행 예시 참고)

[StudentDB 클래스 필드]
students - 학생 정보를 담는 리스트. 이 속성은 생성자에서 파라미터로 초기화되지 않고 기본값으로 빈 리스트 []를 가집니다.

[StudentDB 클래스 메서드]
insert(student) - 학생 정보를 student 파라미터로 입력받아 students 리스트에 추가합니다.
searchByName(name) - 검색 메시지를 출력합니다. 그 후 학생 이름을 name 파라미터로 입력받아 해당 이름을 가진 학생들의 정보를 리스트로 return합니다. 만약 return하는 리스트가 비어있다면 오류 메시지를 출력합니다. (실행 예시 참고)
searchByAge(age) - 검색 메시지를 출력합니다. 그 후 나이를 age 파라미터로 입력받아 해당 나이인 학생들의 정보를 리스트로 return합니다. 만약 return하는 리스트가 비어있다면 오류 메시지를 출력합니다. (실행 예시 참고)
searchByGrade(grade) - 검색 메시지를 출력합니다. 그 후 수강반을 grade 파라미터로 입력받아 해당 반을 수강하는 학생들의 정보를 리스트로 return합니다. 만약 return하는 리스트가 비어있다면 오류 메시지를 출력합니다. (실행 예시 참고)

[프로그램 테스트]
StudentDB 클래스로부터 인스턴스를 생성합니다.
다음의 다섯 학생 정보 Charlie(16/International), Brice(16/Universal), Natalie(13/National), Henry(15/International), Ray(15/International)를 하나씩 데이터베이스에 저장합니다.
이름이 Bill인 학생을 검색하여 해당 학생(들)의 정보를 모두 출력합니다.
나이가 13인 학생을 검색하여 해당 학생(들)의 정보를 모두 출력합니다.
수강반이 International인 학생을 검색하여 해당 학생(들)의 정보를 모두 출력합니다.
테스트 결과는 실행 예시와 같아야 합니다.

```
class Student():
 def __init__(self, name, age, grade):
  self.name = name
```

```
  self.age = age
  self.grade = grade
  self.recentScore = 0

 def goNextGrade(self):
  if self.grade == 'National':
   self.grade = 'International'
  elif self.grade == 'International':
   self.grade = 'Universal'
  print('Now', self.name, 'is in', self.grade, 'class.')

 def takeTest(self, score):
  self.recentScore = score
  print(self.name, 'got', score, 'in this test!')
  if score >= 90:
   if self.grade == 'National' and self.age >= 14:
    self.goNextGrade()
   elif self.grade == 'International' and self.age >= 16:
    self.goNextGrade()

 def toString(self):
  return self.name + ' (' + str(self.age) + '), ' + self.grade

class StudentDB():
 def __init__(self):
  self.students = []

 def insert(self, student):
  self.students.append(student)

 def searchByName(self, name):
  print('Searching students by name ', name, '...', sep='')
  result = []
  for st in self.students:
   if st.name == name:
    result.append(st)
  if len(result) == 0:
   print('No data found.')
  return result
```

```
 def searchByAge(self, age):
  print('Searching students by age ', age, '...', sep='')
  result = []
  for st in self.students:
   if st.age == age:
    result.append(st)
  if len(result) == 0:
   print('No data found.')
  return result

 def searchByGrade(self, grade):
  print('Searching students by grade ', grade, '...', sep='')
  result = []
  for st in self.students:
   if st.grade == grade:
    result.append(st)
  if len(result) == 0:
   print('No data found.')
  return result

db = StudentDB()
db.insert(Student('Charlie', 16, 'International'))
db.insert(Student('Brice', 16, 'Universal'))
db.insert(Student('Natalie', 13, 'National'))
db.insert(Student('Henry', 15, 'International'))
db.insert(Student('Ray', 15, 'International'))

for st in db.searchByName('Bill'):
 print(st.toString())

for st in db.searchByAge(13):
 print(st.toString())

for st in db.searchByGrade('International'):
 print(st.toString())
```

(3) 일산의 한 카페에서는 코로나19 확산에 의해 사회적 거리두기가 강화되면서 테이블 간의 거리를 기존의 3배로 재배치했습니다. 매장의 수용 인원이 줄어든만큼 판매량이 대폭 감소할 것을 우려해, 테이

크아웃을 장려하기 위한 이벤트를 시작했습니다. 자세한 사항은 아래와 같습니다.

- 음료를 테이크아웃할 때마다 쿠폰이 하나씩 증정되며, 쿠폰 3개를 모으면 매장 주문/테이크아웃에 관계없이 4000원 미만의 메뉴를 무료로 구매할 수 있습니다.
- 카페에서 주문할 수 있는 메뉴에는 식사류와 음료가 있으며, 식사류를 테이크아웃하는 경우 쿠폰이 증정되지 않습니다.

해당 이벤트는 카페를 이용하는 고객들이 알기 쉽도록 소프트웨어로 그 내용을 제공하고자 합니다. 지시에 따라 프로그램을 구현해주세요.

[Menu 클래스 필드]
name - 메뉴의 이름입니다.
price - 메뉴의 가격입니다.

[Menu 클래스 메서드]
orderStore() - 매장 주문입니다. 무료가 아닌 경우 가격을 계산하여 구매 내용을 안내하고, 돈이 부족한 경우에는 오류 메시지를 출력하세요. (실행 예시 참고)
takeOut() - 테이크아웃 주문입니다. 무료가 아닌 경우 가격을 계산하여 구매 내용을 안내하고, 돈이 부족한 경우에는 오류 메시지를 출력하세요. (실행 예시 참고)

[Meal 클래스 메서드]
Menu 클래스의 자식으로, 식사류 메뉴에 해당합니다. 필요에 따라 오버라이딩을 수행하세요.

[Beverage 클래스 메서드]
Menu 클래스의 자식으로, 음료 메뉴에 해당합니다. 필요에 따라 오버라이딩을 수행하세요.

[프로그램 테스트]
식사류 메뉴 인스턴스 EggToast(3500won), FrenchOmelet(6500won)와 음료 메뉴 인스턴스 Americano(2500won), CafeLatte(3000won), Frappuccino(5000won)을 생성합니다.
초기 소지금은 15000으로 하고, 쿠폰을 쓸 수 있으면 무조건 사용하는 것으로 합니다.
Americano 메뉴를 테이크아웃으로 구입합니다.
CafeLatte 메뉴를 매장 주문으로 구입합니다.
CafeLatte 메뉴를 테이크아웃으로 구입합니다.
EggToast 메뉴를 테이크아웃으로 구입합니다.
Americano 메뉴를 테이크아웃으로 구입합니다.
Frappuccino 메뉴를 테이크아웃으로 구입합니다.
CafeLatte 메뉴를 매장 주문으로 구입합니다.

테스트 결과는 실행 예시와 같아야 합니다.

```
class Menu():
 def __init__(self, name, price):
```

```
  self.name = name
  self.price = price

 def orderStore(self):
  global money
  global coupon
  if self.price < 4000 and coupon >= 3:
   print("You've ordered", self.name, 'with coupons.')
   coupon = coupon - 3
  elif self.price > money:
   print("You're running out of money!")
  else:
   money = money - self.price
   print("You've ordered", self.name, '-', str(money) + 'won left.')

 def takeOut(self):
  self.orderStore()

class Meal(Menu):
 def takeOut(self):
  self.orderStore()

class Beverage(Menu):
 def takeOut(self):
  global money
  global coupon
  if self.price < 4000 and coupon >= 3:
   print("You've ordered", self.name, 'with coupons.')
   coupon = coupon - 3
  elif self.price > money:
   print("You're running out of money!")
  else:
   money = money - self.price
   print("You've ordered", self.name, '-', str(money) + 'won left.')
   coupon = coupon + 1
   print('Here is a coupon!')

money = 15000
coupon = 0
```

```
et = Meal('EggToast', 3500)
fo = Meal('FrenchOmelet', 6500)
am = Beverage('Americano', 2500)
cl = Beverage('CafeLatte', 3000)
fr = Beverage('Frappuccino', 5000)

am.takeOut()
cl.orderStore()
cl.takeOut()
et.takeOut()
am.takeOut()
fr.takeOut()
cl.orderStore()
```