34장

진급 프로젝트:카드 게임

34장 진급 프로젝트:카드 게임

34 프로젝트 문제

34.1 기존 클래스 사용하기

34.1 기존 클래스 사용하기

» 카드 게임에 사용하려고 하는 유용한 클래스는 random이다. 난수를 만들어주는 이 클래스 정의를 다음과 같이 불러올 수 있다.

```
import random
```

```
r = random.random()
```

0(포함)부터 1(포함하지 않음)까지 범위에 속하는 난수를 만든다

```
r = random.randint(a, b)
```

a와 b 사이의 난수(a, b를 모두 포함함)를 만든다

```
r = random.choice(L)
```

리스트 L의 원소 중 하나를 임의로 선택한다.

34.2 워 게임 상세 규칙

34.2 워 게임 상세 규칙

- 단순화를 위해 덱에는 네 가지 모양(suite)이 있고, 각 모양 마다 2부터 9까지 숫자가 적힌 카드 8장이 있다고 가정하자. 카드를 가리킬 때는 '2H'로 하트 2를, '4D'로 다이아몬드 4를,'7S'로 스페이드 7을, '9C'로 클럽 9를 표현하는 식으로 모양과 숫자를 붙여서 표현한다.
- 플레이어는 이름(문자열)이 있고, 카드 무더기(리스트)를 손에 들고 있다.
- 게임을 시작하면 두 사용자에게 각자의 이름을 물어보고 각 사용자를 설정한다.
- 라운드마다 플레이어의 손에 카드를 하나씩 추가한다
- 플레이어끼리 방금 들어온 카드를 비교한다. 처음에는 숫자를 비교하고(숫자가 크면 더 센카드), 숫자가 같으면 스페이드 〉 하트 〉 다이아몬드 〉 클럽 순서로 세기를 정한다.
- 더 센 카드를 가진 사람은 더 작은 카드를 가진 사람에게 주고, 카드를 받은 사람은 손에 있는 카드 무더기에 카드를 추가한다.
- 덱의 카드가 다 없어지면, 각 플레이어가 들고 있는 카드 개수를 비교한다. 카드가 더 적은 사람이 승리한다.

34.3 Player 클래스 정의하기

34.3 Player 클래스 정의하기

각 플레이어를 그 사람의 이름과 손에 들고 있는 카드로 정의할 수 있다.

- 이름은 문자열이고, 손에 들고 있는 카드는 각 카드를 표현하는 문자열로 이뤄진 리스트다.
- Player 객체를 정의할 때 이름을 생성자의 인자로 전달하고, 처음에는 손에 카드가 없다고 가정한다.

```
class Player(object):
    """ 플레이어 """

    def __init__(self, name):
        """ 이름을 설정하고, 손에는 아무 카드도 들고 있지 않다 """
        self.hand = [] ---- 빈 리스트로 손에 든 카드를 설정
        self.name = name ---- Player 객체를 만들 때 전달받은 문자열로 이름을 설정

    def get_name(self):
        """ 플레이어 이름을 반환한다 """ ---- 플레이어의 이름을 반환하는 메서드
        return self.name
```

코드 34-1
Player 클래스 정의

34.3 Player 클래스 정의하기

추가된 카드가 None이 아닌지 검사하여 카드가 올바른 카드인지 체크한다.

- 플레이어가 손에 들고있는 카드의 개수를 체크하여 승자를 결정하려면 플레이어가 손에 들고 있는 카드의 수를 반환하는 메서드가 필요하다. 다음은 이 세 가지 메서드를 구현한 코드다.

코드 34-2
Player 클래스 정의

```
class Player(object):
    """ 플레이어 """

    # 34-1에 나왔던 메서드들 생략
    def add_card_to_hand(self, card):
        """ card, 문자열
            플레이어 손에 정상적인 카드를 추가한다 """
        if card != None:
            self.hand.append(card)
    def remove_card_from_hand(self, card):
        """ card, 문자열
            플레이어 손에서 카드를 제거한다 """
        self.hand.remove(card)
    def hand_size(self):
        """ 플레이어 손에 있는 카드 개수를 반환한다 """
        return len(self.hand)
```

손에 카드를 추가하면 리스트에 카드를 추가함.
다만 카드가 올바른 카드인 경우에만 추가함

손에서 카드를 제거하려면 리스트에서
그 카드를 찾아서 제거

손에 들고 있는 카드의 개수는 리스트의 원소 개수

34.4 CardDeck 클래스 정의하기

34.4 CardDeck 클래스 정의하기

CardDeck은 카드 덱을 표현한다. 덱의 카드는 32장이고, 각 카드는 2부터 9 사이의 숫자와 스페이드, 하트, 다이아몬드, 클럽 중 한 모양이 있다.

- 데이터 속성은 덱에 들어갈 수 있는 모든 카드가 들어 있는 리스트, 단 하나뿐이다. 각 카드를 문자열로 표현한다. 예를 들어 "3H"는 하트 3이다

```
class CardDeck(object):
    """ 스페이드, 하트, 다이아몬드, 클럽의 2부터 9까지로 구성된 덱 """
    def __init__(self):
        """ 가능한 모든 카드가 들어 있는 덱
            카드를 "2H"(하트 2)와 같은 문자열로 표현한다 """
        hearts = "2H,3H,4H,5H,6H,7H,8H,9H"
        diamonds = "2D,3D,4D,5D,6D,7D,8D,9D"
        spades = "2S,3S,4S,5S,6S,7S,8S,9S"
        clubs = "2C,3C,4C,5C,6C,7C,8C,9C"

        self.deck = hearts.split(',')+diamonds.split(',') + \
                    spades.split(',')+clubs.split(',')
```

덱 안에 들어갈 수 있는 모든 카드를 표현하는 문자열을 만듦

콤마 기준으로 긴 문자열을 분할해서 각 카드를 구하고, 모든 카드(문자열)를 리스트에 넣음

코드 34-3 CardDeck 클래스 초기화

34.4 CardDeck 클래스 정의하기

카드 덱을 덱 안에 들어 있는 모든 카드를 포함하는 문자열로 표현하기로 결정했으므로, 이 클래스에 들어갈 메서드를 구현할 수 있다.

- 이 클래스는 random 클래스를 사용해 플레이어가 사용할 카드를 임의로 고른다.
- 한 메서드는 덱에서 임의로 카드를 하나 선택해 반환하고, 다른 메서드는 두 카드를 비교해서 어떤 카드가 더 센 카드인지 알려준다.

코드 34-4
CardDeck 클래스의 메서드들

```
import random

class CardDeck(object):
    """ 스페이드, 하트, 다이아몬드, 클럽의 2부터 9까지로 구성된 덱 """
    def __init__(self):
        """ 가능한 모든 카드가 들어 있는 덱
            카드를 "2H"(하트 2)와 같은 문자열로 표현한다 """
        hearts = "2H,3H,4H,5H,6H,7H,8H,9H"
        diamonds = "2D,3D,4D,5D,6D,7D,8D,9D"
        spades = "2S,3S,4S,5S,6S,7S,8S,9S"
        clubs = "2C,3C,4C,5C,6C,7C,8C,9C"

        self.deck = hearts.split(',')+diamonds.split(',') + \
                    spades.split(',')+clubs.split(',')
```

34.4 CardDeck 클래스 정의하기

```
def get_card(self):
    """ 임의의 카드(문자열)를 하나 반환한다.
    덱에 카드가 없는 경우에는 `None`을 반환한다  """
    if len(self.deck) < 1:
        return None
    card = random.choice(self.deck)
    self.deck.remove(card)
    return card

def compare_cards(self, card1, card2):
    """ 다음 규칙에 따라 더 센 카드를 반환한다.
    (1) 숫자가 더 큰 카드가 더 세다. 숫자가 같다면
    (2) 스페이드 > 하트 > 다이아몬드 > 클럽 순으로 더 세다 """
    if card1[0] > card2[0]:
        return card1
    elif card1[0] < card2[0]:
        return card2
    elif card1[1] > card2[1]:
        return card1
    else:
        return card2
```

deck에 카드가 없으면 None을 반환

deck 리스트에서 임의의 카드를 하나 고름

deck 리스트에서 카드를 제거

카드 값(문자열)을 반환

카드의 숫자를 비교해서 첫 번째 카드가 더 크면 첫 번째 카드를 반환

카드의 숫자를 비교해서 두 번째 카드가 더 크면 두 번째 카드를 반환

카드의 숫자가 같으면 모양을 비교

34.5 카드 게임 시뮬레이션하기

34.5.1 객체 초기화하기

» 게임을 시뮬레이션하기 위한 첫 단계는 Player 객체를 두 개, CardDeck 객체를 하나 만들어서 게임을 초기화하는 것이다.

- 두 플레이어의 이름을 물어본 후, Player 객체를 만들면서 이름을 설정한 다음, 덱을 만든다.

```
name1 = input("이름을 입력해 주세요. 플레이어 1: ") ---- 플레이어 1의 이름을 사용자에게 입력 받음
player1 = Player(name1) ---- 새 Player 객체를 만듦
name2 = input("이름을 입력해 주세요. 플레이어 2: ")
player2 = Player(name2)
deck = CardDeck() ---- 새 CardDeck 객체를 만듦
```

코드 34-5
게임 변수와 객체 초기화하기

34.5.2 게임의 각 라운드 시뮬레이션하기

- **라운드마다 플레이어는 카드를 한 장씩 받는다. 따라서 덱의 get_card 메서드를 플레이어당 1번씩, 라운드 당 총 2번 호출한다. 그 후 각 플레이어 객체의 add_card_to_hand를 호출해 덱에서 받아온 카드를 플레이어의 손에 있는 카드에 추가한다.**
  - 덱이 비어 있으면 게임이 끝난다.
  - 덱에 카드가 남아 있으면, 두 플레이어는 카드를 비교해 누가 누구에게 카드를 줄지 결정한다.
- **게임이 끝나면 두 플레이어가 손에 든 카드의 수를 hand_size로 가져와 비교한다. 더 카드를 많이 가진 플레이어가 패배하고, 게임 루프는 종료한다.**
- **게임이 끝나지 않았다면 덱 객체의 comapre_cards를 호출해서 더 센 카드를 가지고 있는 플레이어를 결정한다.**
  - compare_cards가 반환하는 값은 숫자가 더 높은 카드다. 만약 숫자가 같으면 정해진 규칙에 따라 모양을 고려해 더 센 카드를 결정한다.
  - 더 센 카드가 player1의 카드와 같으면,player1이 자신의 카드를 player2에게 준다.
  - 코드 상에서 이런 동작은 player1의 remove_card_from_hand를 호출하고, player2의 add_card_to_hand를 호출하는 것으로 구현된다.
  - 더 센 카드가player2의 카드와 같은 경우 비슷한 방식으로(대신 카드가 움직이는 방향은 반대) 처리한다.

34.5.2 게임의 각 라운드 시뮬레이션하기


```
name1 = input("이름을 입력해 주세요. 플레이어 1: ")
player1 = Player(name1)
name2 = input("이름을 입력해 주세요. 플레이어 2: ")
player2 = Player(name2)
deck = CardDeck()

while True:
    player1_card = deck.get_card()
    player2_card = deck.get_card()
    player1.add_card_to_hand(player1_card)
    player2.add_card_to_hand(player2_card)
    if player1_card == None or player2_card == None:
        print("게임 오버. 덱에 카드가 없습니다.")
        print(name1, ": 최종 카드 수 ", player1.hand_size())
        print(name2, ": 최종 카드 수 ", player2.hand_size())
        print("승자는?")
```

---- 두 플레이어 중 한 사람이라도 카드가 없으면 게임을 끝냄

34.5.2 게임의 각 라운드 시뮬레이션하기


```
        if player1.hand_size() > player2.hand_size():
            print(name2, " 승리!")
        elif player1.hand_size() < player2.hand_size():
            print(name1, " 승리!")
        else:
            print("비겼습니다!")
        break
    else:
        print(name1, ": ", player1_card)
        print(name2, ": ", player2_card)
        if deck.compare_cards(player1_card,player2_card)==player1_card:
            player1.remove_card_from_hand(player1_card)
            player2.add_card_to_hand(player1_card)
        else:
            player2.remove_card_from_hand(player2_card)
            player1.add_card_to_hand(player2_card)
```

손에 든 카드의 개수를 비교해서, 더 적은 카드를 가진 player2가 승리

손에 든 카드의 개수를 비교해서, 더 적은 카드를 가진 player1이 승리

두 사람의 카드 수가 같으므로 비김

둘 중 한 사람이 승리하거나, 비긴 경우 break로 while 루프를 나감

손에 비교할 카드가 있으므로 게임을 계속함

두 플레이어의 카드를 비교 더 높은 카드가 반환됨

player1이 더 센 카드를 가지고 있으므로 player2에게 그 카드를 넘김

player1이 더 센 카드를 가지고 있으므로 player1의 손에서 그 카드를 제거

코드 34-6
게임의 라운드들을 시뮬레이션하는 루프

34.6 클래스를 사용한 모듈화와 추상화

34.6. 클래스를 사용한 모듈화와 추상화

문제를 더 작은 단위로 나누지 않으면 게임을 코딩하는 과정과 결과가 지저분해지기 쉽다.

- 객체를 사용하고 객체 지향 프로그래밍을 활용하여 프로그램을 더 잘 모듈화할 수 있었다.
- 우리는 코드를 여러 객체로 나누고, 각 객체에게 데이터 속성과 메서드를 부여했다.
- 객체 지향 프로그래밍을 사용하면 코드를 조직하기 위해 클래스를 만드는 과정과 그 클래스를 사용해 게임을 플레이하는 코드를 구현하는 과정을 분리할 수 있다.
- 게임 플레이를 시뮬레이션하는 동안 같은 타입의 객체를 일관성 있게 사용할 수 있고, 그 결과 더 깔끔하기 읽기 쉬운 코드를 만들 수 있다.
- 이때 객체 타입이나 메서드가 어떻게 구현되었는지에 대한 자세한 내용은 분리해 추상화하면 그 객체를 사용하는 쪽에서 신경쓰지 않아도 된다.
- 그리고 객체를 사용할 때는 메서드 독스트링을 사용해서 시뮬레이션에 적합한 메서드가 어떤 것인지 결정한다.

34.7 요약

34.7 요약

- 클래스를 정의하는 코드는 단 한 번만 작성하면 된다.
- 클래스에는 여러분의 객체가 가질 모든 특성과 여러분이 각 객체에 대해 수행할 수 있는 연산을 기술한다.
- 이 구조는 어떤 객체의 특징이나 객체가 할 수 있는 일을 다른 사람에게 알려주는 코드와 그 객체를 사용해 여러 가지 목적을 달성하는 코드를 분리한다.
- 클래스 정의와 그 클래스에 속한 객체 사용을 분리함으로써, 게임 플레이를 구현하기 위해 꼭 필요한 사항을 제외한 나머지 자세한 세부구현 사항을 감출 수 있다.