

```

1 #####
2 # DCS 229 -- Unfair Solitaire
3 # This is the Card.py file, which defines the Card class
  used in the Solitaire game. The Card class represents a
  playing card with a suit and value, and includes
  methods to retrieve the suit and value, as well as a
  string representation of the card.
4 # Date: 04/01/2026
5 # Name: Benjamin Adovasio
6 # Resources Used: Fran
7 #####
8 class Card:
9     """
10     Represents a playing card with a suit and value.
11     """
12     def __init__(self, suit: str, value: str):
13         self._suit = suit
14         self._value = value
15         #Creates a Card object with the specified suit and
  value.
16
17     def get_suit(self) -> str:
18         return self._suit
19
20     #Returns the suit of the card.
21
22     def get_value(self) -> str:
23         return self._value
24     #Returns the value of the card.
25
26     def __repr__(self) -> str:
27         return f"{self._value} of {self._suit}"
28
29     #Returns a string representation of the card in the
  format "Value of Suit".
30
31 if __name__ == "__main__":
32     card1 = Card("Hearts", "Ace")

```

```
33     card2 = Card("Spades", "10")
34
35     assert card1.get_suit() == "Hearts"
36     assert card1.get_value() == "Ace"
37     assert card2.get_suit() == "Spades"
38     assert card2.get_value() == "10"
39
40     assert repr(card1) == "Ace of Hearts"
41     assert repr(card2) == "10 of Spades"
42
43     print("All Card tests passed.")
```

```

1 #####
2 # DCS 229 -- Unfair Solitaire
3 # This is the Deck.py file, which defines the Deck class
  used in the Solitaire game. The Deck class represents a
  standard 52-card deck, and includes methods to shuffle
  the deck, deal cards, and check the number of remaining
  cards.
4 # Date: 04/01/2026
5 # Name: Benjamin Adovasio
6 # Resources Used: Fran
7 #####
8
9 import random
10 from Card import Card
11
12 """
13 Defines the deck of cards used in the Solitaire game.
  Contains methods to shuffle the deck, deal cards, and
  check the number of remaining cards.
14
15 Important Note: The prints in this file only run when
  this file is ececuted directly. This file should not be
  executed directly normally.
16 """
17 class Deck:
18     def __init__(self):
19         self._cards= []
20         self._next_card = 0
21
22         suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades'
23 ] #the possible suits
24         values = ['2', '3', '4', '5', '6', '7', '8', '9'
25 , '10', 'Jack', 'Queen', 'King', 'Ace'] #the possible
  values
26
27         for suit in suits:
28             for value in values:
29                 self._cards.append(Card(suit, value)) #

```

```

27 creates a standard 52-card deck
28
29     def shuffle(self):
30         """
31         Shuffles the deck of cards using the Fisher-
32         Yates algorithm. This algorithm iterates through the
33         deck and swaps each card with a randomly selected card
34         from the remaining unshuffled portion of the deck. After
35         shuffling, it resets the next card index to 0.
36         """
37         for i in range(len(self._cards)): #iterates
38             through each card in the deck
39                 j = random.randrange(i, len(self._cards)) #
40                 selects a random index from i to the end of the deck
41                 self._cards[i], self._cards[j] = self._cards
42                 [j], self._cards[i] #Shuffles the deck
43                 self._next_card = 0      #Resets the next card
44                 index after shuffling.
45
46     def deal(self):
47         """
48         Deals the next card from the deck. Returns None
49         if there are no cards left to deal.
50         """
51         if self._next_card >= len(self._cards):
52             return None #Returns None if there are no
53             cards left to deal.
54             card = self._cards[self._next_card]
55             self._next_card += 1 #Advances the next card
56             index.
57             return card
58
59     def number_of_cards(self) -> int:
60         return len(self._cards) - self._next_card #
61         Returns the number of remaining cards in the deck.
62
63     def __repr__(self) -> str:

```

```
53         """
54         Returns a string representation of the entire
    deck,
55         with one card per line.
56         """
57         return "\n".join(str(card) for card in self.
    _cards)
58
59 if __name__ == "__main__":
60     deck = Deck()
61     deck.shuffle() #Calls the shuffle method to
    randomize the deck.
62     for _ in range(5):
63         print(deck.deal())
64     print("Cards left:", deck.number_of_cards()) #Prints
    the number of cards left in the deck after dealing five
    cards.
```

```
1 #####
2 # DCS 229 -- Unfair Solitaire
3 # This is the main.py file, which contains the main
  function to run the simulation.
4 # Date: 04/01/2026
5 # Name: Benjamin Adovasio
6 # Resources Used: Fran
7 #####
8 from Solitaire import Solitaire
9
10
11 def main():
12     wins = 0
13
14     for games in range(1, 10001):
15         game = Solitaire()
16         if game.playGame():
17             wins += 1
18
19         if games % 1000 == 0:
20             percent = (wins / games) * 100
21             print(f"{wins}/{games} games won = {percent:
22 .2f}%")
23
24 if __name__ == "__main__":
25     main()
26
```

```

1 #####
2 # DCS 229 -- Unfair Solitaire
3 # Defines the Solitaire game logic. Contains methods to
  play the game by dealing cards, removing cards based on
  game rules, and checking for a win condition.
4 # Name: Benjamin Adovasio
5 # Resources Used: Fran
6 #####
7
8 from Deck import Deck
9
10
11 class Solitaire:
12     def __init__(self):
13         self.deck = Deck()
14         self.face_up = [] #List to hold face-up cards.
15
16         '''
17         deals cards until there are four face-up cards or
the deck is empty.
18         '''
19         def deal_until_four(self):
20             while len(self.face_up) < 4 and self.deck.
number_of_cards() > 0:
21                 self.face_up.append(self.deck.deal()) #Deals
cards until there are four face-up cards.
22
23             '''
24             removes the last four cards if they are all the same
suit. Returns True if cards were removed, False
otherwise.
25             '''
26             def remove_four_same_suit(self) -> bool:
27                 if len(self.face_up) < 4:
28                     return False #Not enough cards to remove
four of the same suit.
29
30                 last_four = self.face_up[-4:]

```

```

31         suit = last_four[0].get_suit()
32
33         if all(card.get_suit() == suit for card in
last_four):
34             del self.face_up[-4:]
35             return True #Removed four cards of the same
suit.
36
37         return False
38
39     '''
40     removes the inner two cards if the first and last
card of the last four cards are the same suit. Returns
True if cards were removed, False otherwise.
41     '''
42     def remove_inner_two(self) -> bool:
43         if len(self.face_up) < 4:
44             return False #Not enough cards to remove
inner two cards.
45
46         last_four = self.face_up[-4:]
47         if last_four[0].get_suit() == last_four[3].
get_suit():
48             del self.face_up[-3:-1]
49             return True #Removed inner two cards.
50
51         return False
52
53
54     '''
55     Attempts to remove cards based on game rules.
56     '''
57     def remove_all_possible(self):
58         removed = True
59         while removed and len(self.face_up) >= 4:
60             removed = self.remove_four_same_suit()
61             if not removed:
62                 removed = self.remove_inner_two() #

```

```
62 Attempts to remove cards based on game rules.
63
64     '''
65     plays the game by shuffling the deck, dealing cards
    , removing cards based on game rules, and checking for a
    win condition.
66     '''
67     def playGame(self) -> bool:
68         self.deck.shuffle()
69         self.face_up = []
70
71         self.deal_until_four() #Deals cards until there
    are four face-up cards.
72
73         while self.deck.number_of_cards() > 0:
74             self.remove_all_possible()
75             self.face_up.append(self.deck.deal())
76
77             self.remove_all_possible()
78         return len(self.face_up) == 0 # Checks for a
    win condition.
79
80 if __name__ == "__main__":
81     game = Solitaire()
82     print("win?", game.playGame()) #Prints whether the
    game was won or not. Not normally executed.
```