```python
class Card:
    """
    Represents a playing card with a suit and value.
    """
    def __init__(self, suit: str, value: str):
        self._suit = suit
        self._value = value
    #Creates a Card object with the specified suit and value.

    def get_suit(self) -> str:
        return self._suit

    #Returns the suit of the card.

    def get_value(self) -> str:
        return self._value
    #Returns the value of the card.

    def __repr__(self) -> str:
        return f"{self._value} of {self._suit}"

    #Returns a string representation of the card in the format "Value of Suit".
```

```
 1 import random
 2 from Card import Card
 3
 4 """
 5 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.
 6
 7 Important Note: The prints in this file only run when
    this file is ececuted directly. This file should not
    be executed directly normally.
 8 """
 9 class Deck:
10     def __init__(self):
11         self._cards= []
12         self._next_card = 0
13
14         suits = ['Hearts', 'Diamonds', 'Clubs', '
   Spades'] #the possible suits
15         values = ['2', '3', '4', '5', '6', '7', '8',
   '9', '10', 'Jack', 'Queen', 'King', 'Ace'] #the
   possible values
16
17         for suit in suits:
18             for value in values:
19                 self._cards.append(Card(suit, value
   )) #creates a standard 52-card deck
20
21     def shuffle(self):
22         for i in range(len(self._cards)): #iterates
   through each card in the deck
23             j = random.randrange(i, len(self._cards
   )) #selects a random index from i to the end of the
   deck
24             self._cards[i], self._cards[j] = self.
   _cards[j], self._cards[i] #Shuffles the deck
25         self._next_card = 0    #Resets the next card
   index after shuffling.
```

```
26
27      def deal(self):
28          if self._next_card >= len(self._cards):
29              return None #Returns None if there are no
     cards left to deal.
30          card = self._cards[self._next_card]
31          self._next_card += 1 #Advances the next card
    index.
32          return card
33
34      def number_of_cards(self) -> int:
35          return len(self._cards) - self._next_card #
    Returns the number of remaining cards in the deck.
36
37  if __name__ == "__main__":
38      deck = Deck()
39      deck.shuffle() #Calls the shuffle method to
    randomize the deck.
40      for _ in range(5):
41          print(deck.deal())
42      print("Cards left:", deck.number_of_cards()) #
    Prints the number of cards left in the deck after
    dealing five cards.
```

```python
"""
This is the main file that should be called to run the simulation.
"""
from Solitaire import Solitaire


def main():
    wins = 0

    for games in range(1, 10001):
        game = Solitaire()
        if game.playGame():
            wins += 1

        if games % 1000 == 0:
            percent = (wins / games) * 100
            print(f"{wins}/{games} games won = {percent:.2f}%")


if __name__ == "__main__":
    main()

```

```
 1 from Deck import Deck
 2 """
 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
 5 Important Note: The prints in this file only run when
    this file is ececuted directly. This file should not
    be executed directly normally.
 6 """
 7
 8 class Solitaire:
 9     def __init__(self):
10         self.deck = Deck()
11         self.face_up = [] #List to hold face-up cards
   .
12
13     def deal_until_four(self):
14         while len(self.face_up) < 4 and self.deck.
   number_of_cards() > 0:
15             self.face_up.append(self.deck.deal()) #
   Deals cards until there are four face-up cards.
16
17     def remove_four_same_suit(self) -> bool:
18         if len(self.face_up) < 4:
19             return False #Not enough cards to remove
   four of the same suit.
20
21         last_four = self.face_up[-4:]
22         suit = last_four[0].get_suit()
23
24         if all(card.get_suit() == suit for card in
   last_four):
25             del self.face_up[-4:]
26             return True #Removed four cards of the
   same suit.
27
28         return False
```

```
29
30      def remove_inner_two(self) -> bool:
31          if len(self.face_up) < 4:
32              return False #Not enough cards to remove
    inner two cards.
33
34          last_four = self.face_up[-4:]
35          if last_four[0].get_suit() == last_four[3].
    get_suit():
36              del self.face_up[-3:-1]
37              return True #Removed inner two cards.
38
39          return False
40
41      def remove_all_possible(self):
42          removed = True
43          while removed and len(self.face_up) >= 4:
44              removed = self.remove_four_same_suit()
45              if not removed:
46                  removed = self.remove_inner_two() #
    Attempts to remove cards based on game rules.
47
48      def playGame(self) -> bool:
49          self.deck.shuffle()
50          self.face_up = []
51
52          self.deal_until_four() #Deals cards until
    there are four face-up cards.
53
54          while self.deck.number_of_cards() > 0:
55              self.remove_all_possible()
56              self.face_up.append(self.deck.deal())
57
58          self.remove_all_possible()
59          return len(self.face_up) == 0  #   Checks for
     a win condition.
60
61  if __name__ == "__main__":
```

```
62      game = Solitaire()
63      print("win?", game.playGame()) #Prints whether
    the game was won or not. Not normally executed.
```