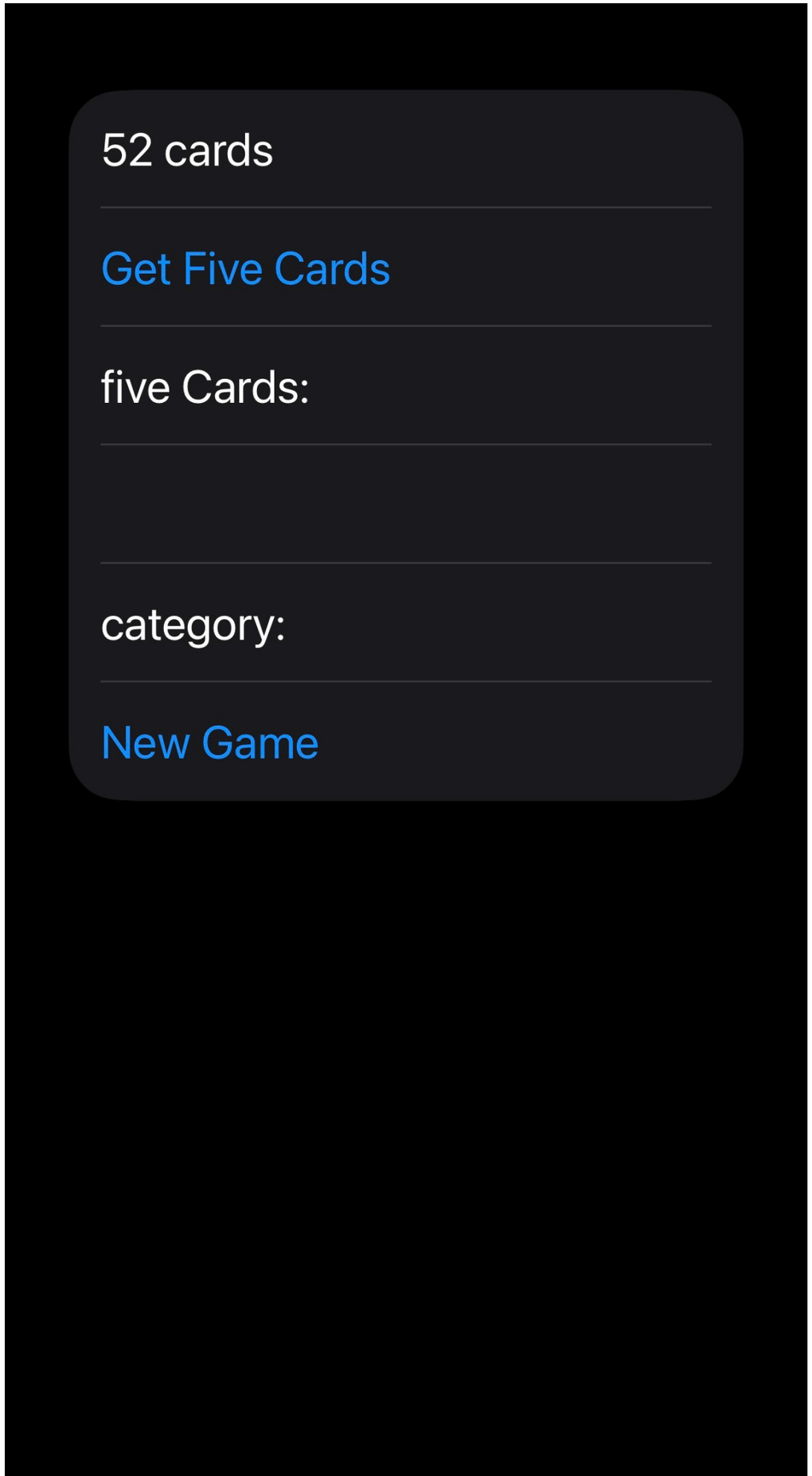
Object Oriented Programming


52 cards
Get Five Cards
five Cards:
category:
New Game

47 cards

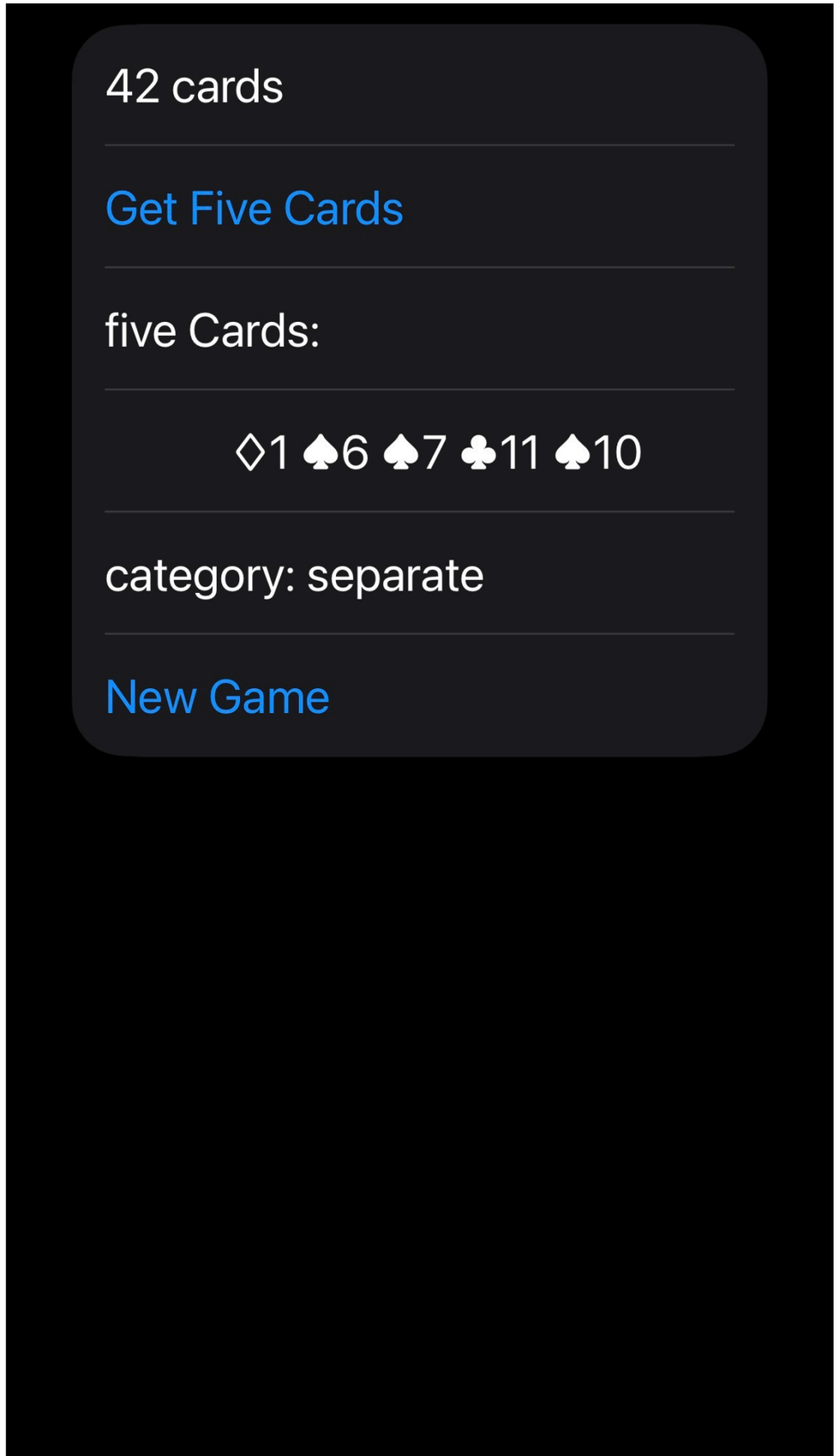
Get Five Cards

five Cards:

♢13 ♡1 ♣3 ♣5 ♠3

category: onePair

New Game


42 cards
Get Five Cards
five Cards:
♢1 ♠6 ♠7 ♣11 ♠10
category: separate
New Game

27 cards

Get Five Cards

five Cards:

♠1 ♣10 ♢10 ♡5 ♡12

category: onePair

New Game

17 cards

Get Five Cards

five Cards:

♣12 ♡10 ♠2 ♢8 ♡8

category: onePair

New Game

1. (10 points) Declare enum Suit

```
enum Suit : Character {
    case spades = "♠",
}
```

2. (10 points) Declare enum Rank

```
enum Rank : Int {
    case 

}
```

3. (25 points) Declare class Card.

```
class Card {
    private let rank: Rank
    private let suit: Suit
    init(_ rank: Rank, of suit: Suit){

    }
    func getRank() -> Rank {

    }
    func getSuit() -> Suit{

    }
    func compareSuit(with otherCard: Card) -> Bool{

    }
    func compareRank(with otherCard: Card) -> Bool{

    }
}
```

A. Complete init()
B. Complete getRank()
C. Complete getSuit()
D. Complete compareRank()
E. Complete compareSuit()

4. (20 points) Let a be a constant that represents an ace of spades. Let q be a constant that represents a queen of hearts.
    A. Declare constant a.

```
let a = Card(
let q = Card(
print(a.
print(q.getRank(), q.
print(q.
print(a.
print(a.compareSuit(
```

    B. Declare constant q.
    C. Print the rank and suit of a.
    D. Print the rank and suit of q.
    E. Print the raw value of the rank and suit of q.
    F. Print the result of comparing equality of the rank of q and a.
    G. Print the result of comparing equality of the suit of q and a.

5. (10 points) Declare two constant lists

```
let suits = [Suit.spades,                                    ]
let ranks = [Rank.ace,                                      e,
    Rank.six, Rank.seven, Rank.eight, Rank.nine, Rank.ten,
                                                            ]
```

A. A list, suits, that collects all cases of enum Suit.

B. A list, ranks, that collects all cases of enum Rank.

```
let suits = [Suit.clubs, Suit.diamonds, Suit.hearts,
Suit.spades]
let ranks = [Rank.ace, Rank.two, Rank.three, Rank.four,
Rank.five, Rank.six, Rank.seven, Rank.eight, Rank.nine,
Rank.ten ,Rank.jack, Rank.queen, Rank.king]
```

6. (15 points) Declare a stuct RandomDeckOfCards such that constant cards denotes a list collecting a random shuffle of all cards.

```
struct RandomDeckOfCards {
    let cards: 
    init(){
        var allCards = 
        for suit in {
            for rank in {
                let card = 
                allCards.append(card)
            }
        }
        self.cards = allCards.()
    }
}
```

```
struct RandomDeckOfCards{
    let cards: [Card]
    init(){
        var allCards = [Card]()
        for suit in suits {
            for rank in ranks {
                let card = Card(rank, of: suit)
                allCards.append(card)
            }
        }
        self.cards = allCards.shuffled()
    }
}
```

7. (20 points) Declare class GameCards. The constant deck belongs to the class of RandomDeckOfCards. The private var top denotes a pointer to the card for just dealing.
    A. Complete init()
    B. Complete remainingNumOfCards(). The value returned by this method represents the number of remaining cards for further dealing.
    C. (10 points) Complete getFiveCards(). This method returns an empty array if the number of remaining cards for further dealing is less than the constant getNum. Otherwise its returns five cards with indices ranging from top to top + 4.

```
class GameCards {
    let deck : RandomDeckOfCards
    private var top = 0
    init(){
        [redacted].deck = [redacted]
    }
    func remainingNumOfCards() -> Int{
        return 52 - [redacted]
    }
    func getFiveCards() -> [Card] {
        let getNum = 5
        var fiveCards = [redacted]
        if self.remainingNumOfCards() >= getNum{
            for i in top...[redacted]-1{
                let card = deck.[redacted]
                fiveCards.[redacted]
            }
            top [redacted]
        }
        return [redacted]
    }
}
```

```swift
class GameCards{
    let deck : RandomDeckOfCards
    private var top = 0
    init(){
        self.deck = RandomDeckOfCards()
    }
    func remainingNumOfCards() -> Int{
        return deck.cards.count - top
    }
    func getFiveCards() -> [Card]{
        let getNum = 5
        var fiveCards = [Card]()
        if top + getNum  <= deck.cards.count {
            for card in deck.cards[top...top+getNum-1]{
                fiveCards.append(card)
            }
        }
        top += getNum
        return fiveCards
    }

}
```

8. (95 points) Declare struct Hand.
   A. (5) Accomplish init()
   B. Define func flush. Let variable setSuit represent a set of suits of the five cards. If there is only one element in setSuit, the five cards are categorized as a flush.

```
    func flush() -> Bool{
        var setSuit = Set<Suit>()
        for [redacted] {
            let suit = card.[redacted]()
            setSuit.[redacted]
        }
        return setSuit.count [redacted]
    }
```

```
struct Hand {
    let fiveCards : [Card]
    init(from game: GameCards){
        self.[redacted] = game.[redacted]
    }
    func straightFlush() -> Bool {
    }
    func flush() -> Bool{
    }
    func straight() -> Bool{
    }
    func fourOfaKind() -> Bool{
    }
    func fullHouse() -> Bool{
    }
    func twoPair() -> Bool {
    }
    func threeOfaKind() -> Bool{
    }
    func onePair() -> Bool{
    }
    func separate() -> Bool{
    }
}
```

```
    func flush() -> Bool{
        var setSuit = Set<Suit>()
        for card in fiveCards {
            let suit = card.getSuit()
            setSuit.insert(suit)
        }
        return setSuit.count == 1
    }
```

```
struct Hand {
    let fiveCards : [Card]
    init(from game: GameCards){
        self.fiveCards = game.getFiveCards()
    }
    func straightFlush()-> Bool {
        return self.straight() && self.flush()
    }
```

```
    func straight() -> Bool{
        var fiveRank = [Int]()
        for card in fiveCards {
            let r = card.getRank().rawValue
            fiveRank.append(r)
        }
        fiveRank.sort()
        if fiveRank[0] == 1 && fiveRank[1] == 10 &&
fiveRank[2] == 11 && fiveRank[3] == 12 && fiveRank[4] == 13 {
            return true
        }
        for i in 0...fiveRank.count-2{
            let d = fiveRank[i+1] - fiveRank[i]
            if d != 1{
                return false
            }
        }
        return true
    }
```

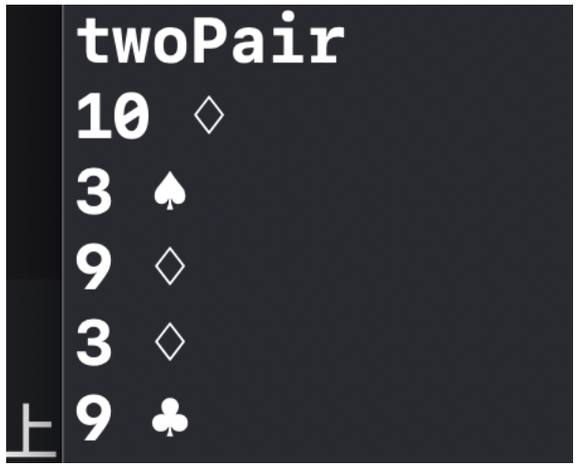
```
    func fourOfaKind() -> Bool{
        var setRank = Set<Rank>()
        var set2Rank = Set<Rank>()
        for card in fiveCards {
            let rank = card.getRank()
            if setRank.contains(rank){
                set2Rank.insert(rank)
            } else {
                setRank.insert(rank)
            }
        }
        if setRank.count == 2 && set2Rank.count == 1 {
            return true
        }
        return false
    }
```

A rank in setRank occurs once at least

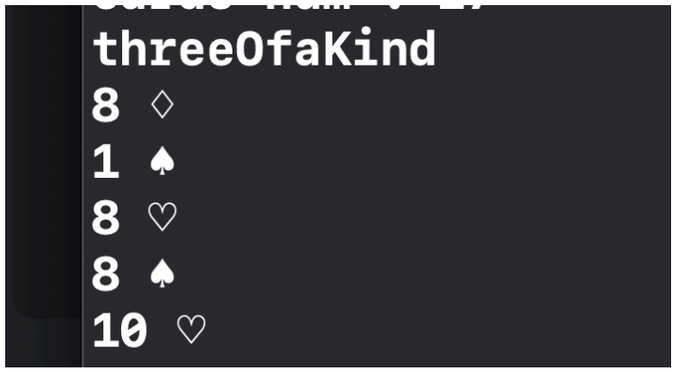
A rank in set2Rank occurs twice at least

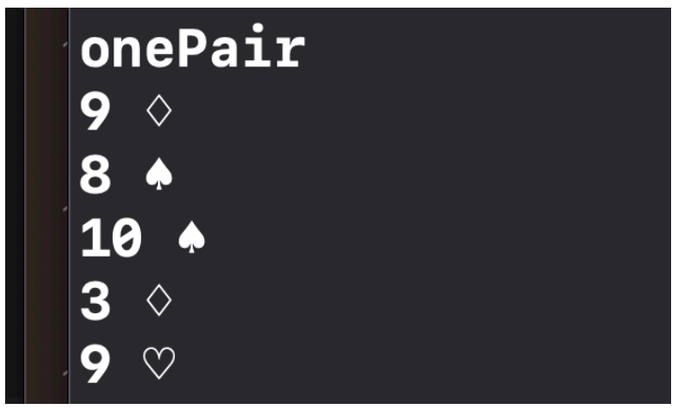
A rank in setRank occurs once at least

	Number of elements in setRank	Number of elements in set2Rank
fourOfaKind	2	1
fullHouse	2	2
twoPair	3	2
threeOfaKind	3	1
onePair	4	1

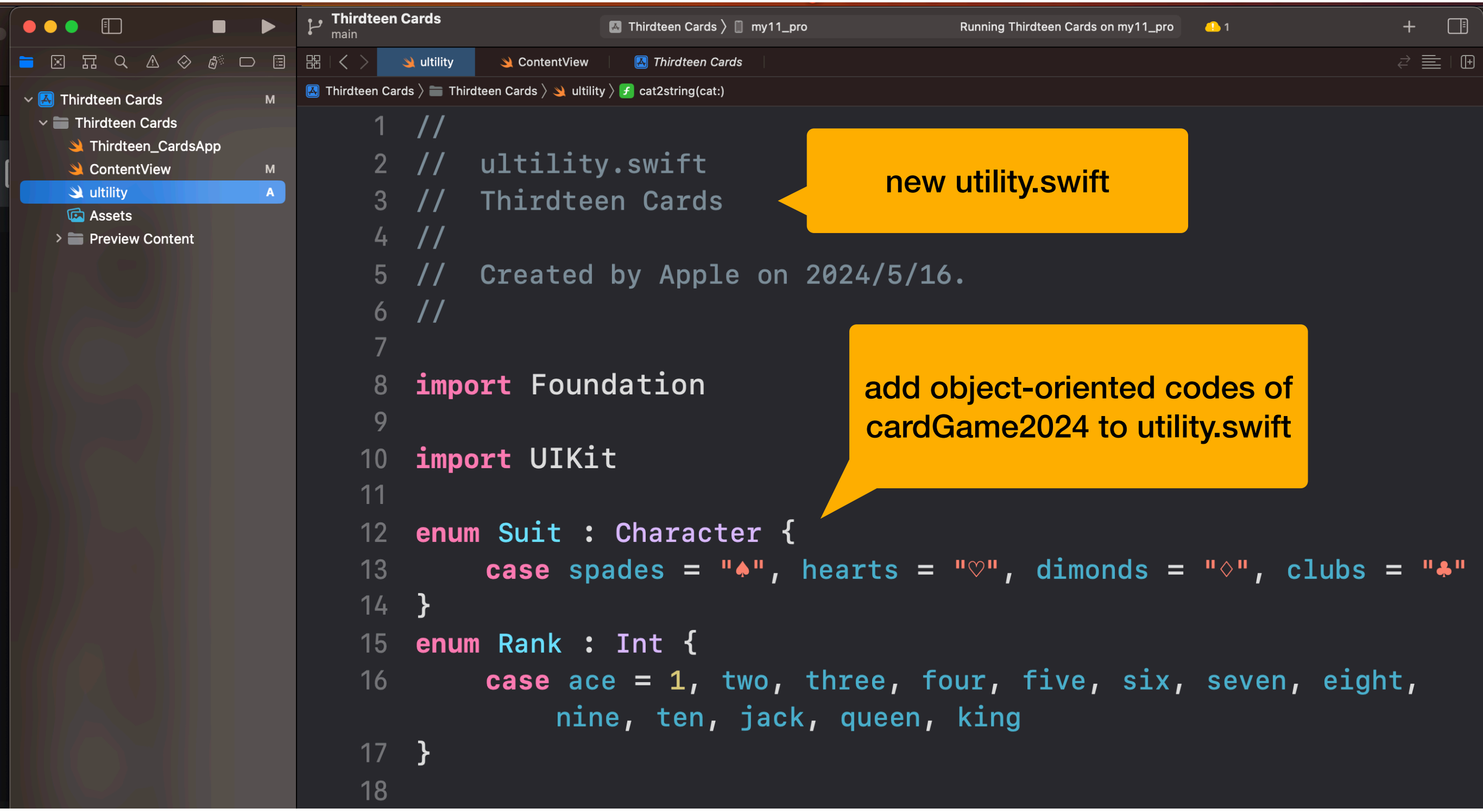


A rank in set2Rank occurs twice at least

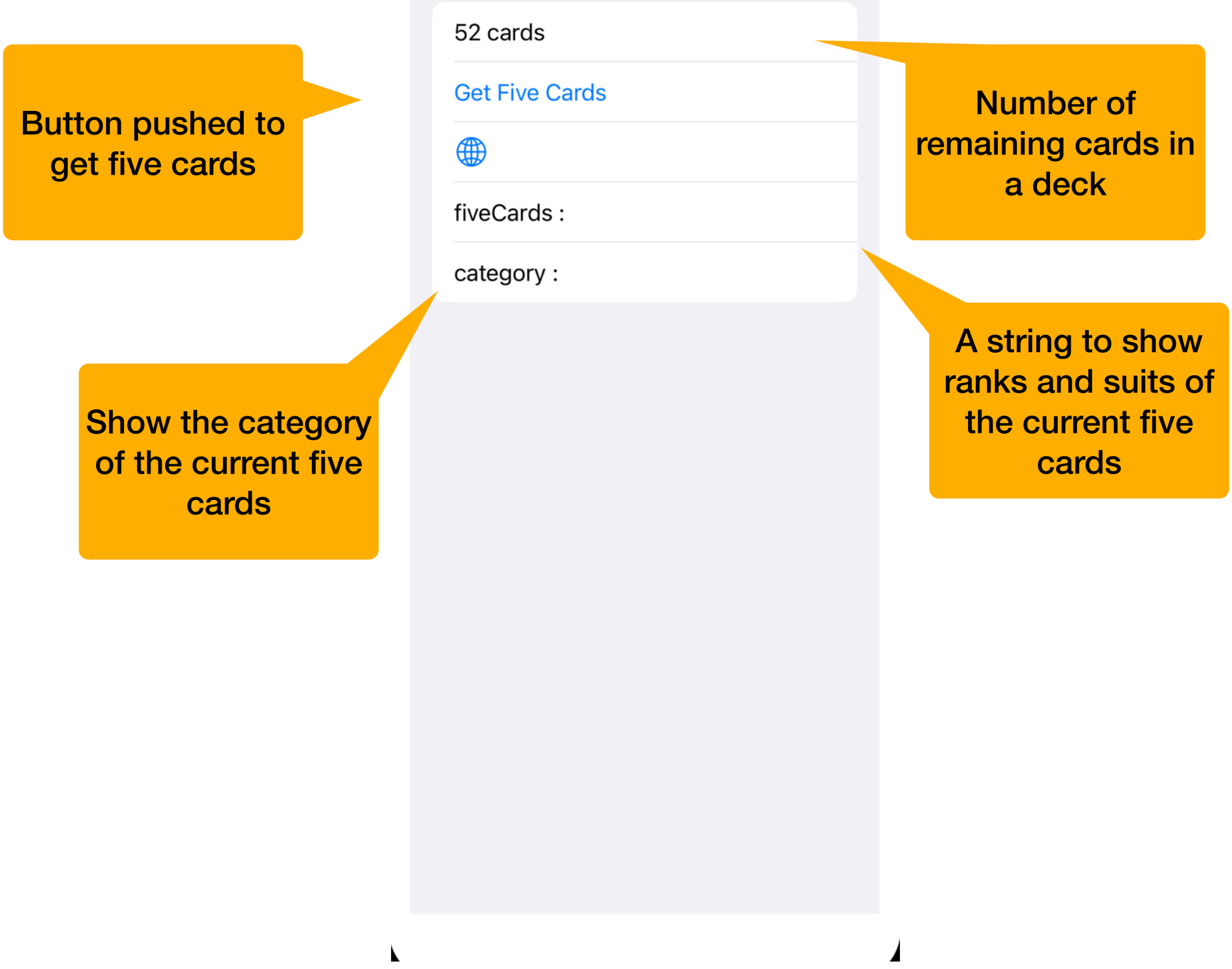




CardGameApp: one player


Thirdteen Cards
main
Thirdteen Cards 〉 my11_pro
Running Thirdteen Cards on my11_pro
1
ultility
ContentView
Thirdteen Cards
Thirdteen Cards 〉 Thirdteen Cards 〉 ultility 〉 cat2string(cat:)
Thirdteen Cards M
Thirdteen Cards
Thirdteen_CardsApp
ContentView M
ultility A
Assets
Preview Content
1 //
2 // ultility.swift
3 // Thirdteen Cards
4 //
5 // Created by Apple on 2024/5/16.
6 //
7
8 import Foundation
9
10 import UIKit
11
12 enum Suit : Character {
13 case spades = "♠", hearts = "♡", dimonds = "♢", clubs = "♣"
14 }
15 enum Rank : Int {
16 case ace = 1, two, three, four, five, six, seven, eight,
nine, ten, jack, queen, king
17 }
18
new utility.swift
add object-oriented codes of cardGame2024 to utility.swift

```swift
func cat2string(cat: Category) -> String{
    switch cat.rawValue {
    case 1:
        return "straightFlush"
    case 2:
        return "fourOfaKind"
    case 3:
        return "fullHouse"
    case 4:
        return "flush"
    case 5:
        return "straight"
    case 6:
        return "threeOfaKind"
    case 7:
        return "twoPair"
    case 8:
        return "onePair"
    default:
        return "separate"
    }
}
```


52 cards
Get Five Cards
fiveCards :
category :
Button pushed to get five cards
Number of remaining cards in a deck
Show the category of the current five cards
A string to show ranks and suits of the current five cards

47 cards

Get Five Cards

🌐

fiveCards : ♢13 ♢7 ♢4 ♡7 ♢8

category : onePair

47 cards

Get Five Cards

🌐

fiveCards : ♣12 ♣4 ♡4 ♡12 ♠1

category : twoPair

42 cards

Get Five Cards

fiveCards : ♢1 ♢12 ♠3 ♣10 ♡3

category : onePair

27 cards

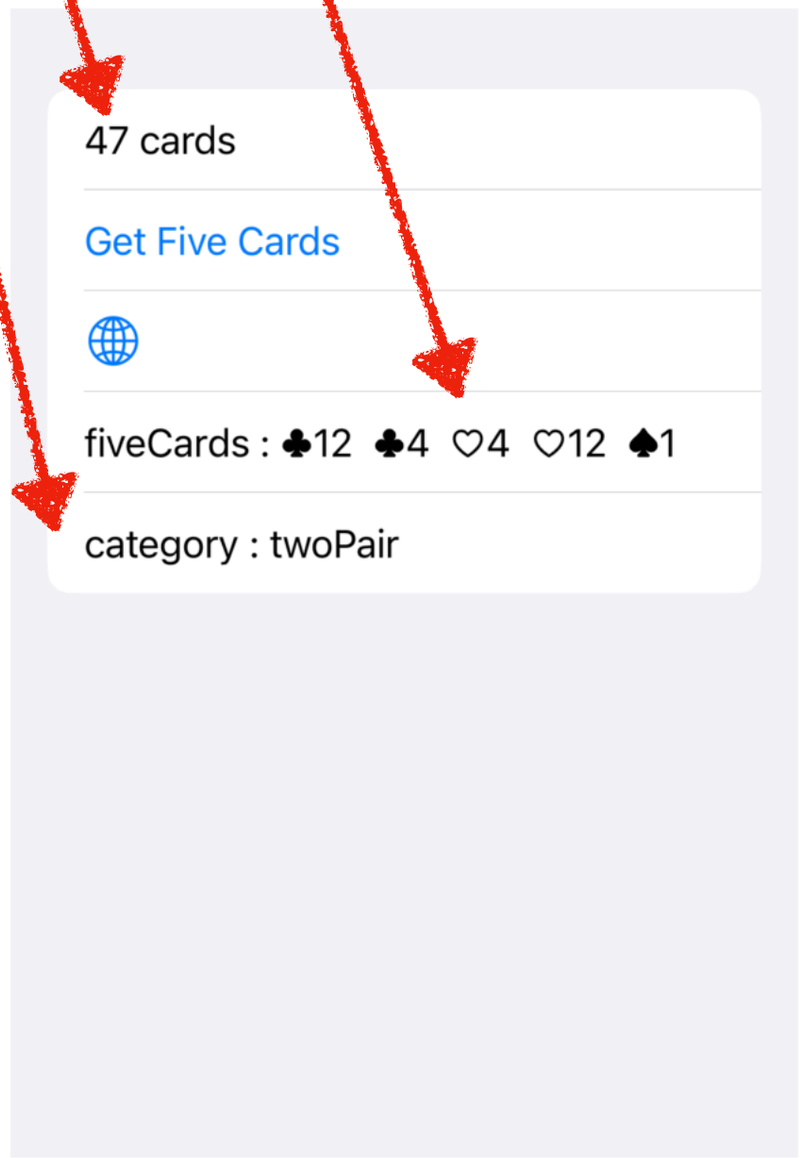
Get Five Cards

fiveCards : ♣2 ♣7 ♠2 ♡4 ♡2

category : threeOfaKind

Content view

```swift
struct ContentView: View {
    @State private var catString = ""
    @State private var fiveCards = ""
    @State private var remainingNum : Int = 52
    var game = GameCards()
```



```
    var body: some View {
        Form {
            Text("\(remainingNum) cards" )
            Button("Get Five Cards"){
                if game.remainingNumOfCards() > 10 {
                    let hand = Hand(from: game)
                    let cat = hand.setCategory()
                    remainingNum = game.remainingNumOfCards()
                    catString = cat2string(cat: cat)
                    fiveCards = ""
                    for card in hand.fiveCards {
                        let rr = String(card.getSuit().rawValue)
                        fiveCards += rr
                        let ss = String(card.getRank().rawValue)
                        fiveCards += ss + "   "
                    }

                }
            }
    }
```

Requirement 1: Correct Category.

42 cards

Get Five Cards

fiveCards : ♢1 ♢12 ♠3 ♣10 ♡3

category : onePair

27 cards

Get Five Cards

fiveCards : ♣2 ♣7 ♠2 ♡4 ♡2

category : threeOfaKind

Requirement 1: New a Game.
Add a button for a new game.
All cards are shuffled and the number of remaining cards equals 52

52 cards

Get Five Cards

five Cards:

category:

New Game

Requirement 3 : Add Card Images

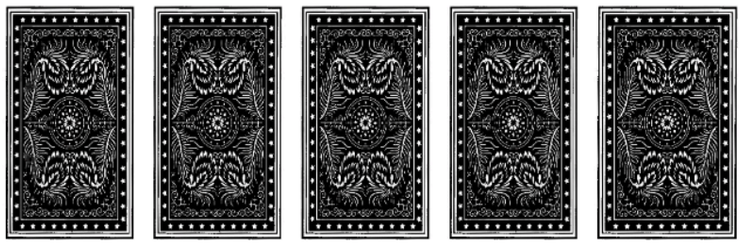
9:41

52 cards

Get Five Cards

fiveCards :

category :



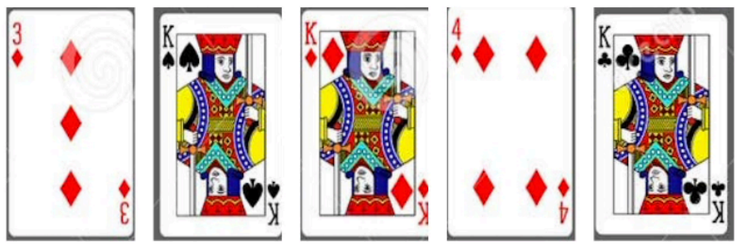
9:41

47 cards

Get Five Cards

fiveCards : ♢3 ♠13 ♢13 ♢4 ♣13

category : threeOfaKind



Requirement 4 :
Add a button to get 13 cards.

Automatically divide 13 cards to three groups respectively containing 3, 5 and 5 cards.

Display the category of each group.

(option) Try to develop an algorithm for 3-group dividing and state its advantages for winning the 13-card game when the number of players increases to 4.