```
import java.util.*;


public class OkeyGame {


   // This function checks whether the given 14 stones are in the "ends" position    public static boolean isComplete(List<Integer> board) {
      if (board.size() != 14) {
         return false; // If the number of stones is not 14, the game is not over
      }


      // 7 pair control: 7 pairs of stones with the same number
      if (checkPairs(board)) {
         return true; // If there are 7 pairs, it is considered finished
      }


      // Block control: The same number of stones or consecutive stones
      return checkBlocks(board);
   }


   // Double control (7 double control)
   private static boolean checkPairs(List<Integer> board) {
      Map<Integer, Integer> countMap = new HashMap<>();


      // Calculate the frequency of stones
      for (int card : board) {
         countMap.put(card, countMap.getOrDefault(card, 0) + 1);
      }
```

```
    // we need to find 7 pairs
    int pairs = 0;
    for (int count : countMap.values()) {
        if (count == 2) {
            pairs++;
        }
    }

    return pairs >= 7; // If there are 7 or more pairs, the game is considered completed
}

// Block control: Stones with the same number or consecutive stones
private static boolean checkBlocks(List<Integer> board) {
    Collections.sort(board); // We sort the stones

    // Check blocks: Same numbers or consecutive numbers
    for (int i = 0; i < board.size() - 1; i++) {
        // Aynı sayıya sahip taşlar (setler)
        if (board.get(i) == board.get(i + 1)) {
            continue; // Aynı sayı, set oluşturur
        }
        // Consecutive numbers
        if (board.get(i) + 1 == board.get(i + 1)) {
            continue; // Consecutive numbers form blocks
        }
        // If it is not consecutive or not the same number, it is not a valid block            return false;
    }
```

```
        return true; // Eğer bloklar geçerli ise
    }

    // Test function
    public static void main(String[] args) {
        // Test 1: 7 pairs and some consecutive stones
        List<Integer> board1 = Arrays.asList(1, 1, 2, 2, 3, 3, 4, 4, 5, 6, 6, 7, 8, 9);
        System.out.println("Board 1 is complete: " + isComplete(board1)); // true

        // Test 2: 7 pairs and the same numbers
        List<Integer> board2 = Arrays.asList(1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7);
        System.out.println("Board 2 is complete: " + isComplete(board2)); // true

        // Test 3: Consecutive stones and sets       List<Integer> board3 = Arrays.asList(10, 11, 12, 13, 14, 15, 6, 6, 6, 7, 7, 8, 8, 8);
        System.out.println("Board 3 is complete: " + isComplete(board3)); // true

        // Test 4: Eksik çiftler, geçerli blok yok
        List<Integer> board4 = Arrays.asList(1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 11);
        System.out.println("Board 4 is complete: " + isComplete(board4)); // false
    }
}
```

Açıklamalar:

- **`isComplete`Fonksiyonu** : This function takes a player's pieces and after checking that the pieces are 14 pieces, it checks whether there are 7 pairs. if there are 7 pairs oyun it

is considered finished. if there are no 7 pairs, the blocks (consecutive stones or stones with the same number) are checked.

- **`checkPairs`Fonksiyonu** : This function is 7 pairs between the stones and Decays.
- **`checkBlocks`Fonksiyonu** : This function sorts the stones and creates consecutive stones or aytru

1. *** Test Scenarios**
2. **Board 1** : There will be 7 pairs and batru Decouple between the stones, because the stones are of biting length
3. **Board 2** : there are 7 pairs of stones, and therefore the root of the root.
4. **Board 3** : Successive Stones vtruechrook.
5. **Board 4** : There are not enough pairs here, and the current BFAL is the root.

Test Çıktısı:

Board 1 is complete: true

Board 2 is complete: true

Board 3 is complete: true

Board 4 is complete: false

Of course, I have written a function below that determines whether the pieces in the Okey game are in the "end position" or not. This function will check a valid block structure or 7 pairs by checking the stones.

1. **Steps:**
2. **2. Ends position: In order for a player to be considered complete, he must either have 7 pairs (7 stones with the same number), or the stones must form certain blocks (consecutive stones or stones with the same number).**
3. **Test scenarios: We will do tests with some sample inputs to make sure that the code works.**
4. **Kod:**

```
import java.util.*;

public class OkeyGame {

    // Bu fonksiyon, verilen 14 taşın "biter" pozisyonunda olup olmadığını
kontrol eder
    public static boolean isComplete(List<Integer> board) {
```

```
        if (board.size() != 14) {
            return false; // Eğer taş sayısı 14 değilse, oyun bitmemiştir
        }

        // 7 çift kontrolü: Aynı sayıya sahip taşlardan 7 çift
        if (checkPairs(board)) {
            return true; // Eğer 7 çift varsa, bitmiş sayılır
        }

        // Blok kontrolü: Aynı sayıda taşlar veya ardışık taşlar
        return checkBlocks(board);
    }

    // Çift kontrolü (7 çift kontrolü)
    private static boolean checkPairs(List<Integer> board) {
        Map<Integer, Integer> countMap = new HashMap<>();

        // Taşların sıklığını hesapla
        for (int card : board) {
            countMap.put(card, countMap.getOrDefault(card, 0) + 1);
        }

        // 7 çift bulmalıyız
        int pairs = 0;
        for (int count : countMap.values()) {
            if (count == 2) {
                pairs++;
            }
        }

        return pairs >= 7; // Eğer 7 veya daha fazla çift varsa, oyun
tamamlanmış kabul edilir
    }

    // Blok kontrolü: Aynı sayıya sahip taşlar veya ardışık taşlar
    private static boolean checkBlocks(List<Integer> board) {
        Collections.sort(board); // Taşları sıralıyoruz

        // Blokları kontrol et: Aynı sayılar veya ardışık sayılar
        for (int i = 0; i < board.size() - 1; i++) {
            // Aynı sayıya sahip taşlar (setler)
            if (board.get(i) == board.get(i + 1)) {
                continue; // Aynı sayı, set oluşturur
            }
            // Ardışık sayılar
            if (board.get(i) + 1 == board.get(i + 1)) {
                continue; // Ardışık sayılar, blok oluşturur
            }
            // Eğer ardışık değilse veya aynı sayı değilse, geçerli bir
blok değil
            return false;
        }
        return true; // Eğer bloklar geçerli ise
    }

    // Test fonksiyonu
    public static void main(String[] args) {
        // Test 1: 7 çift ve bazı ardışık taşlar
        List<Integer> board1 = Arrays.asList(1, 1, 2, 2, 3, 3, 4, 4, 5, 6,
6, 7, 8, 9);
```

```
        System.out.println("Board 1 is complete: " + isComplete(board1));
// true

        // Test 2: 7 çift ve aynı sayılar
        List<Integer> board2 = Arrays.asList(1, 1, 2, 2, 3, 3, 4, 4, 5, 5,
6, 6, 7, 7);
        System.out.println("Board 2 is complete: " + isComplete(board2));
// true

        // Test 3: Ardışık taşlar ve setler
        List<Integer> board3 = Arrays.asList(10, 11, 12, 13, 14, 15, 6, 6,
6, 7, 7, 8, 8, 8);
        System.out.println("Board 3 is complete: " + isComplete(board3));
// true

        // Test 4: Eksik çiftler, geçerli blok yok
        List<Integer> board4 = Arrays.asList(1, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 10, 10, 11);
        System.out.println("Board 4 is complete: " + isComplete(board4));
// false
    }
}
```

Açıklamalar:

- * isComplete Function: This function takes a player's pieces and checks if there are 7 pairs, after checking that the pieces are 14 pieces. If there are 7 pairs, the game is considered finished. If there are no 7 pairs, the blocks (consecutive stones or stones with the same number) are checked.
- * checkPairs Function: This function checks whether there are 7 pairs between the stones.Dec. * checkPairs Function: This function checks whether there are 7 pairs between the stones. If there are 7 or more pairs, the function returns true.checkBlocks

*** Function: This function sorts the stones and checks consecutive stones or stones with the same number. If there is a valid block (consecutive stones or stones with the same number), the function returns true.**

1. **Test Scenarios:**
2. Board 1: There are 7 pairs of stones and some consecutive stones between the stones.Dec. In this case, the function will return true, because the stones are in the end position.
3. **Board 2**: there are 7 pairs of stones, and that's why the game is almost over. The function will return true.
4. **Board 3**: There are consecutive stones and some sets. This is also a valid completed game position. The function will return true.
5. **Board 4**: There are not enough pairs here, and there are also no valid blocks. Therefore, the game is incomplete and the function will return false.
6. **Çıktı:**

```
Board 1 is complete: true
Board 2 is complete: true
Board 3 is complete: true
Board 4 is complete: false
```

This code provides a simple control function to check whether the pieces in the Okey game have been completed.