**QUESTION 6**

**CODE:**

/\*

1.Instance variables include coins (array), sum (target), result (number of ways), and comb (list of combinations).

The input method is synchronized to set coins and sum, followed by notifying the waiting thread.

2.The calculate method is synchronized and uses the recursive calculateWays function.

calculateWays explores all coin combinations to reach the target sum, returning ways and populating the comb list.

The printCombinations method prints combinations from the comb list.

3.In the main method, an instance of Multithread is created. User input for coin values and target sum is taken. Two threads (inputThread and calculateThread) handle input and calculation concurrently.

Thread execution involves inputThread prompting user input and calling input, while calculateThread calls calculate.

The algorithm employs recursion to explore all coin combinations, updating the comb list and returning the total number of ways. This multithreaded design enhances program efficiency,

with notify() and wait() ensuring proper synchronization between input and calculation threads.

\*/

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Multithread {

private int[] coins;

private int sum;

private int result;

private List<List<Integer>> comb;

public Multithread()

{

this.comb = new ArrayList<>();

}

public synchronized void input(int[] coins, int sum) {

this.coins = coins;

this.sum = sum;

notify();

}

public synchronized void calculate() {

try {

if (coins == null) {

wait();

}

} catch (InterruptedException e) {

System.out.println("Wait until input thread finishes");

}

result = calculateWays(coins, coins.length, sum, new ArrayList<>());

System.out.println("Number of ways: " + result);

printCombinations();

}

private int calculateWays(int coins[], int m, int n, List<Integer> currentCombination) {

if (n == 0) {

comb.add(new ArrayList<>(currentCombination));

return 1;

}

if (n < 0 || m <= 0) {

return 0;

}

int withoutCurrentCoin = calculateWays(coins, m - 1, n, currentCombination);

currentCombination.add(coins[m - 1]);

int withCurrentCoin = calculateWays(coins, m, n - coins[m - 1], currentCombination);

currentCombination.remove(currentCombination.size() - 1);

return withoutCurrentCoin + withCurrentCoin;

}

private void printCombinations() {

System.out.println("Possible Combinations:");

for (List<Integer> combination : comb) {

System.out.println(combination);

}

}

public static void main(String[] args) {

Multithread object = new Multithread();

Scanner scanner = new Scanner(System.in);

Thread inputThread = new Thread(new Runnable() {

@Override

public void run() {

System.out.print("Enter the number of coins: ");

int numberOfCoins = scanner.nextInt();

int[] coins = new int[numberOfCoins];

System.out.print("Enter the values of coins: ");

for (int i = 0; i < numberOfCoins; i++) {

coins[i] = scanner.nextInt();

}

System.out.print("Enter the target sum: ");

int targetSum = scanner.nextInt();

object.input(coins, targetSum);

}

});

Thread calculateThread = new Thread(new Runnable() {

@Override

public void run() {

object.calculate();

}

});

inputThread.start();

calculateThread.start();

}

}

OUTPUT:

