**Question 1 :-**

**Write a function to return the longest even length word in a sentence**

Sample input: Be not afraid of greatness, some are born great, some achieve greatness, and some have greatness thrust upon them.

Output: afraid

**package** com;

**public** **class** TestNew {

**public** **static** **void** main(String[] args) {

String str = "Be not afraid of greatness, some are born great, some achieve greatness, and some have greatness thrust upon them";

String[] words = str.replaceAll("[,]", "").split(" ");

**int** maxLength = 0;

String evenWord = "";

**for** (String word : words) {

**if** (word.length() % 2 == 0) {

**if** (word.length() > maxLength) {

maxLength = word.length();

evenWord = word;

}

}

}

System.***out***.println(evenWord);

}

}

Question 2

Given a number, find the next smallest palindrome larger than the number. Give an

optimized solution rather than a brute force algorithm.

Few examples

Input Number Output - The next smallest palindrome

125 131

250 252

123 131

397 404

Write the below function

int nextPalindrome(int input)

solution :-

**public** **class** TestNew {

**public** **static** **void** main(String[] args) {

Scanner s1 = **new** Scanner(System.***in***);

**int** input = s1.nextInt();

**int** nextPailandrome = *nextPalindrome*(input);

System.***out***.println(nextPailandrome);

}

**private** **static** **int** nextPalindrome(**int** input) {

input++;

**while** (!*checkPalindrome*(input)) {

input++;

}

**return** input;

}

**private** **static** **boolean** checkPalindrome(**int** input) {

String str = Integer.*toString*(input);

**int** left = 0;

**int** right = str.length() - 1;

**while** (left < right) {

**if** (str.charAt(left) != str.charAt(right)) {

**return** **false**;

}

left++;

right--;

}

**return** **true**;

}

}

Question 3:-

ATM Dispenser Simulator

Design a function that simulates an ATM machine dispensing money. The ATM contains

various denominations of bills, and a customer requests a specific withdrawal amount. the

function should determine whether the ATM can fulfil the request based on the available

notes and their quantities.

Input:

withdrawal\_amount (positive integer): the amount of money the customer wants to

withdraw.

atm\_notes (data structure): A data structure representing the available denominations (keys)

and their corresponding quantities (values) in the ATM. This data structure can he a

HashMap (Java)

programming language.

Example:

withdrawal amount = 270

atm\_notes =

100: 5, # 5 notes of $100 denomination

50:3,#3 notes of $50 denomination

20:2,#2 notes of $20 denomination

10:1 #1 note of $10 denomination

}

**Output:**

• Return a data structure indicating the number of notes dispensed from each

denomination (if successful). This data structure should mirror the format used for

atm\_notes.

• For 270 withdrawal ATM can dispense below notes.

**atm\_notes =**

100: 2, # 2 notes of $100 denomination

50: 1, # 1 note of $50 denomination

20: 1, # 1 note of $20 denomination

**Solution**

**package com;**

**import java.util.HashMap;**

**import java.util.Map;**

**public class TestNew {**

**public static void main(String[] args) {**

**int withdrawalAmount = 270;**

**Map<Integer, Integer> atmNotes = new HashMap<>();**

**atmNotes.put(100, 5);**

**atmNotes.put(50, 3);**

**atmNotes.put(20, 2);**

**atmNotes.put(10, 1);**

**Map<Integer, Integer> dispensedNotes = withdrawFromATM(withdrawalAmount, atmNotes);**

**if (dispensedNotes != null) {**

**System.out.println("ATM dispensed notes:");**

**for (Map.Entry<Integer, Integer> entry : dispensedNotes.entrySet()) {**

**System.out.println(entry.getKey() + ": " + entry.getValue());**

**}**

**} else {**

**System.out.println("ATM cannot dispense the requested amount.");**

**}**

**}**

**public static Map<Integer, Integer> withdrawFromATM(int amount, Map<Integer, Integer> atmNotes) {**

**Map<Integer, Integer> dispensedNotes = new HashMap<>();**

**int[] denominations = { 100, 50, 20, 10 };**

**for (int denomination : denominations) {**

**if (atmNotes.containsKey(denomination)) {**

**int count = Math.min(amount / denomination, atmNotes.get(denomination));**

**if (count > 0) {**

**dispensedNotes.put(denomination, count);**

**amount =amount- count \* denomination;**

**}**

**}**

**}**

**if (amount == 0) {**

**for (Map.Entry<Integer, Integer> entry : dispensedNotes.entrySet()) {**

**atmNotes.put(entry.getKey(), atmNotes.get(entry.getKey()) - entry.getValue());**

**}**

**return dispensedNotes;**

**} else {**

**return null;**

**}**

**}**

**}**

**Question 4**

**You have to distribute N coconuts equally among some elephants according to the following**

**conditions:**

**• You can choose the number of elephants that receive coconuts.**

**• Each elephant should get more than one coconut.**

**• One elephant cannot receive all the coconuts.**

**• All the N coconuts must be distributed.**

**• Each elephant can only receive full coconuts. You can't distribute half**

**coconut.**

**Write a program to determine whether the coconuts can be equally distributed among the**

**elephants. There will be bonus marks for optimizing the solution.**

**Output**

**Input**

**N coconuts to be distributed. Return true or false depending upon the result.**

**Examples**

**N = 2**

**2 coconuts cannot be distributed among group of any size. suppose we take a group of size 1**

**then one elephant takes all the coconuts. If we take a group of size 2 each elephant will only**

**1 coconut which violates the rule of distribution**

**N 15 =**

**15 coconut can be equally distributed among 3 elephants each one getting 5 coconuts.**

**Solution :**

Sure, let's break down the solution step-by-step with a focus on understanding the logic:

**Problem Requirements**

You need to distribute N coconuts among some elephants with the following conditions:

1. **Each elephant should get more than one coconut**.
2. **One elephant cannot receive all the coconuts**.
3. **All the coconuts must be distributed**.
4. **Each elephant can only receive whole coconuts** (no fractional coconuts).

**Strategy**

To solve this, we need to:

1. **Identify the possible number of elephants** that can receive coconuts.
2. **Check if the total number of coconuts can be evenly divided** by the number of elephants.
3. **Ensure that each elephant gets more than one coconut**.

**Steps in the Code**

1. **Edge Case Handling**:
   * If N is less than 2, it's immediately impossible to distribute coconuts among multiple elephants as per the rules.
2. **Iterate Over Possible Numbers of Elephants**:
   * We loop through possible numbers of elephants from 2 to N / 2. We start from 2 because one elephant cannot receive all the coconuts, and we go up to N / 2 because any number of elephants more than N / 2 would mean each elephant gets less than 2 coconuts.
3. **Check Divisibility and Distribution Conditions**:
   * For each number of elephants, we check:
     + If N can be divided evenly by this number (N % elephants == 0).
     + If each elephant receives more than one coconut (N / elephants > 1).

**Detailed Code Explanation**

****

**package** com;

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** TestNew {

**public** **static** **void** main(String[] args) {

**int**[] testCases = { 2, 15, 18, 20 };

**for** (**int** N : testCases) {

System.***out***.println("N = " + N + " -> " + *canDistributeCoconuts*(N));

}

}

**public** **static** **boolean** canDistributeCoconuts(**int** N) {

**if** (N < 2)

**return** **false**; // Not enough coconuts for any meaningful distribution

// Loop from 2 to N/2 to check all possible numbers of elephants

**for** (**int** elephants = 2; elephants <= N / 2; elephants++) {

// Check if N can be evenly divided by this number of elephants

// and if each elephant gets more than one coconut

**if** (N % elephants == 0 && N / elephants > 1) {

**return** **true**;

}

}

**return** **false**; // No valid distribution found

}

}

### Example Walkthroughs

#### Example 1: N = 2

* Check possible numbers of elephants from 2 to N / 2 (which is 1 in this case).
* No valid number of elephants as 2 coconuts can't be divided among 2 elephants (each would get 1 coconut, violating the rule).
* **Output: false**

#### Example 2: N = 15

* Check possible numbers of elephants from 2 to 15 / 2 (which is 7).
* For 3 elephants: 15 % 3 == 0 and 15 / 3 > 1 (each gets 5 coconuts).
* **Output: true**

#### Example 3: N = 18

* Check possible numbers of elephants from 2 to 18 / 2 (which is 9).
* For 2 elephants: 18 % 2 == 0 and 18 / 2 > 1 (each gets 9 coconuts).
* **Output: true**