You can view this report online at : https://www.hackerrank.com/x/tests/1526474/candidates/49967548/report

Full Name:	Seva Nandu
Email:	mangali.sevanandu@wipro.com
Test Name:	**Back-End Developer (Spring Boot) Test Mangali Sevanandu**
Taken On:	7 Feb 2023 11:34:28 IST
Time Taken:	104 min 52 sec/ 105 min
Work Experience:	< 1 years
Linkedin:	https://www.linkedin.com/in/sevanandu-m-525018185
Invited by:	Kavitha
Invited on:	6 Feb 2023 21:55:43 IST
Skills Score:	Java (Basic) 50/50 Problem Solving (Basic) 50/50 REST API (Intermediate) 0/50 Spring Boot (Basic) 20/50
Tags Score:	Back-End Development 0/50 Easy 120/200 Filtering 20/50 Interviewer Guidelines 50/50 JSON 0/50 Java 70/100 Loops 50/50 OOPS 50/50 Operators 50/50 Problem Solving 50/100 REST API 0/50 Sorting 20/50 Spring Boot 20/50
Candidate Tags:	backend-springboot,backend-test

60%

120/200

scored in **Back-End Developer (Spring Boot) Test Mangali Sevanandu** in 104 min 52 sec on 7 Feb 2023 11:34:28 IST

Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review.

Question Description	Time Taken	Score	Status

Q1	Counting Closed Paths Coding	5 min 1 sec	50/ 50	
Q2	Spring Boot: Filter Microservice Back-end Developer	1 hour 17 min 54 sec	20/ 50	
Q3	Levels of Friendship Coding	15 min 53 sec	50/ 50	
Q4	REST API: Counting Movies Coding	6 min 40 sec	0/ 50	

QUESTION 1

Needs Review

Score 50

Counting Closed Paths Coding

Easy | Operators | Loops | Problem Solving | Interviewer Guidelines

QUESTION DESCRIPTION

Some numbers are formed with closed paths. The digits *0, 4, 6* and *9* each have *1* closed path, and *8* has *2*. None of the other numbers is formed with a closed path. Given a *number*, determine the total number of *closed paths* in all of its digits combined.

Example

number = 649578

The digits with closed paths are *6, 4, 9* and *8*. The total number of closed paths is *1 + 1 + 1 + 2 = 5*.

Function Description

Complete the function *closedPaths* in the editor below.

closedPaths has the following parameter(s):

int number: an integer

Returns:

int: the number of *closed paths* in *number*

Constraints

- $1 \leq number \leq 10^9$

Input Format For Custom Testing

Input from stdin will be processed as follows and passed to the function:

The only line of input contains a single integer, *number*.

Sample Case 0

Sample Input

```
STDIN       Function
-------     ----------
630     →   number = 630
```

Sample Output

```
2
```

Explanation

Sum the *closed paths* count for each digit, *6, 3* and *0*. Return *1 + 0 + 1 = 2*.

Sample Case 1

Sample Input

```
STDIN      Function
-------    ----------
1288   →   number = 1288
```

Sample Output

Explanation

Sum the *closed paths* count for each digit, *1, 2, 8, 8*. Return *0 + 0 + 2 + 2 = 4*.

INTERVIEWER GUIDELINES

▼ Solution

Skills: Loops, Problem Solving

Optimal Solution:

Iterate over all the digits of the number, adding 1 to the answer if the current digit is one of (0,4,6,9) or 2 if the current digit is 8.

```
def closedPaths(number):
    # Write your code here
    ans=0
    #Iterating over all digits of the number
    for i in map(int,str(number)):
        if(i in (0,4,6,9)):
            ans+=1
        if(i == 8):
            ans+=2
    return ans
```

▼ Complexity Analysis

Time Complexity - $O(\log_{10}(number))$

Space Complexity - O(1)

CANDIDATE ANSWER

Language used: **Java 15**

```
class Result {

    /*
     * Complete the 'closedPaths' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts INTEGER number as parameter.
     */

    public static int closedPaths(int number) {
        int temp=0,sum=0;
        while(number>0){
            temp=number%10;
            if(temp==0 || temp==4 || temp==6 || temp==9){
                sum=sum+1;
            }
            else if(temp==8){
                sum=sum+2;
            }
           number=number/10;
        }
        return sum;

    }

```

```
26
27
28
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.0868 sec	31.3 KB
TestCase 1	Easy	Sample case	Success	1	0.0774 sec	31.6 KB
TestCase 2	Easy	Sample case	Success	1	0.0962 sec	31.2 KB
TestCase 3	Easy	Sample case	Success	6	0.0526 sec	31.5 KB
TestCase 4	Easy	Sample case	Success	3	0.0909 sec	31.4 KB
TestCase 5	Easy	Hidden case	Success	3	0.0585 sec	31.1 KB
TestCase 6	Easy	Hidden case	Success	7	0.0804 sec	31.3 KB
TestCase 7	Easy	Hidden case	Success	7	0.0538 sec	31.3 KB
TestCase 8	Easy	Hidden case	Success	7	0.0791 sec	31.2 KB
TestCase 9	Easy	Hidden case	Success	7	0.071 sec	31.4 KB
TestCase 10	Easy	Hidden case	Success	7	0.0504 sec	31.4 KB

No Comments

QUESTION 2

Correct Answer

Score 20

Spring Boot: Filter Microservice

Back-end Developer | Spring Boot | Java | Filtering | Sorting | Easy

QUESTION DESCRIPTION

Implement REST APIs to perform filter and sort operations on a collection of Products.

Each event is a JSON entry with the following keys:

- `barcode` : the unique id of the product (String)
- `price` : the price of the product (Integer)
- `discount` : the discount % available on the product(Integer)
- `available` : the availability status of the product (0 or 1)

Here is an example of a product JSON object:

```
[
    {
      "barcode": "74001755",
      "item": "Ball Gown",
      "category": "Full Body Outfits",
      "price": 3548,
      "discount": 7,
      "available": 1
    },
    {
      "barcode": "74002423",
      "item": "Shawl",
      "category": "Accessories",
      "price": 758,
      "discount": 12,
      "available": 1
    }
]
```

You are provided with the implementation of the models required for all the APIs The task is to implement

a set of REST services that exposes the endpoints and allows for filtering and sorting the collection of product records in the following ways:

`GET` request to `/filter/price/{initial_range}/{final_range}`:

- returns a collection of all products whose price is between the initial and the final range supplied
- The response code is 200, and the response body is an array of products in the price range provided.
- In case there are no such products return status code 400.

`GET` request to `/sort/price`:

- returns a collection of all products sorted by their pricing
- The response code is 200 and the response body is an array of the product names sorted in ascending order of price.

Complete the given project so that it passes all the test cases when running the provided unit tests.

▼ Example requests and responses

`GET` **request to** `/filter/price/{initial_range}/{final_range}`

The response code is 200, and when converted to JSON, the response body is as follows for filter/750/900:

```
[
  {
    "barCode": "74002423"
  }
]
```

`GET` **request to** `/sort/price`

The response code is 200 and the response body, when converted to JSON, is as follows:

```
[
  {
    "barCode": "74002423"
  },
  {
    "barCode": "74001755"
  }
]
```

CANDIDATE ANSWER

controller/SampleController.java

```
package com.hackerrank.sample.controller;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
```

```
  -                       -

@RestController
public class SampleController {


           final String uri =
"https://jsonmock.hackerrank.com/api/inventory";
           RestTemplate restTemplate = new RestTemplate();
           String result = restTemplate.getForObject(uri, String.class);
           JSONObject root = new JSONObject(result);

           JSONArray data = root.getJSONArray("data");



                @CrossOrigin

@GetMapping("/filter/price/{initial_price}/{final_price}")
                private ResponseEntity< ArrayList<FilteredProducts> >
filtered_books(@PathVariable("initial_price") int init_price ,
@PathVariable("final_price") int final_price)
                {

                        try {

                                            ArrayList<FilteredProducts> books
= new ArrayList<FilteredProducts>();

                                List<JSONObject> list = new ArrayList<>
();
                                for (int i = 0; i < data.length(); i++) {
                                                FilteredProducts
filteredProduct = new FilteredProducts(data.get(i).getString("barcode"));

books.add(filteredProduct);
                                            }
                                }

                                if (books.isEmpty()){
                                        throw new Exception();
                                }
                                    return new
ResponseEntity<ArrayList<FilteredProducts>>(books, HttpStatus.OK);



                        }catch(Exception E)
                                {
                System.out.println("Error encountered :
"+E.getMessage());
             return new ResponseEntity<ArrayList<FilteredProducts>>
(HttpStatus.NOT_FOUND);
                                }

                }


                @CrossOrigin
                @GetMapping("/sort/price")
                private ResponseEntity<SortedProducts[]> sorted_books()
                {

                        try {
                                 List<JSONObject> list = new ArrayList<>
();
                                 for (int i = 0; i < data.length(); i++){
                                        list.add(data.get(i));
                                 }
```

```
                                return new ResponseEntity<SortedProducts[]>
(ans, HttpStatus.OK);

                            }catch(Exception E)
                                    {
                    System.out.println("Error encountered :
"+E.getMessage());
                return new ResponseEntity<SortedProducts[]>
(HttpStatus.NOT_FOUND);
                                    }

                    }


}
```

CANDIDATE SUBMISSION

This question did not define reportable data.

We could not score the candidate's solution because of the following reason:

Scoring command failed to generate test result files

View candidate code

Review logs: output log

No Comments

QUESTION 3

Needs Review

Score 50

Levels of Friendship Coding

QUESTION DESCRIPTION

This challenge implements levels of friendship in Java classes.

Acquaintance: A person one knows slightly, but who is not a close friend.
Friend: A person with whom one has a close bond.
Best Friend: A person's closest friend.

As the levels of friendship increase, you get to know more about the person.

Java | OOPS | Easy

Implement these levels in terms of 3 Java classes :

Class *Acquaintance*

- Has an attribute: “name” (variable of type String)
- Constructor: Acquaintance(String name)
- Has a method public void getStatus which prints “[name] is just an acquaintance.\n”

Class *Friend*

- Class Friend inherits class Acquaintance
- Constructor: Friend(String name, String homeTown)
- Has attribute “homeTown” (variable of type String)
- Has a method public void getStatus which prints “[name] is a friend and he is from [homeTown].\n”

Class *BestFriend*

- Class *BestFriend* inherits class *Friend*
- Constructor: *BestFriend*(String *name*, String *homeTown*, String *favoriteSong*)
- Has attribute “*favoriteSong*” (variable of type String)
- Has a method public void getStatus which prints *“[name]* is my best friend. He is from *[homeTown]* and his favorite song is *[favoriteSong]*."

Note: You do not have to worry about input handling, code stub does that

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of friends
Each line *i* of the *n* subsequent lines (where *0 < i < n*) contains data for a friend in the format:
[Acquaintance|Friend|BestFriend] [FriendName] {HomeTown} {FavouriteSong}

▼ Sample Case 0

Sample Input

```
4
Acquaintance Jaden
Friend Jake Florida
BestFriend Ryan Utah Dangerous
Friend David Texas
```

Sample Output

```
Jaden is just an acquaintance.
Jake is a friend and he is from Florida.
Ryan is my best friend. He is from Utah and his favorite song is
Dangerous.
David is a friend and he is from Texas.
```

▼ Sample Case 1

Sample Input

```
5
Acquaintance Roger
BestFriend Carson Boston Believer
Friend Oren Atlanta
BestFriend Ramon Miami Radioactive
Friend Tyson Denver
```

Sample Output

```
Roger is just an acquaintance.
Carson is my best friend. He is from Boston and his favorite song is
Believer.
Oren is a friend and he is from Atlanta.
Ramon is my best friend  He is from Miami and his favorite song is
```

```
Radioactive.
Tyson is a friend and he is from Denver.
```

CANDIDATE ANSWER

Language used: **Java 7**

```
class Acquaintance{
    public String name;
    public Acquaintance(String name){
        this.name=name;
    }
    public String getName(){
         return name;
    }
    public void setName(String name){
        this.name=name;
    }
    public void getStatus(){
        System.out.println(name+" is just an acquaintance.");
    }
}
class Friend extends Acquaintance{
    public String hometown;
    public Friend(String name,String hometown){
        super(name);
        this.hometown=hometown;
    }
    public void getStatus(){
        System.out.println(name+" is a friend and he is from "+hometown+".");
    }
}
class BestFriend extends Friend{
    private String fs;
    public BestFriend(String name,String hometown,String fs){
        super(name,hometown);

        this.fs=fs;
    }
    public void getStatus(){
        System.out.println(name+" is my best friend. He is from "+hometown+"
and his favorite song is "+fs+".");
    }
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	1	0.0824 sec	23.8 KB
Testcase 1	Easy	Sample case	Success	1	0.0873 sec	23.9 KB
Testcase 2	Easy	Hidden case	Success	14	0.0715 sec	24.6 KB
Testcase 3	Easy	Hidden case	Success	10	0.1377 sec	23.8 KB
Testcase 4	Easy	Hidden case	Success	10	0.0976 sec	24.4 KB
Testcase 5	Easy	Hidden case	Success	14	0.1478 sec	24.4 KB

No Comments

Not Submitted

Score 0

Problem Solving

QUESTION DESCRIPTION

Write an *HTTP GET* method to retrieve information from a movie database concerning how many movies have a particular string in their title. Given a search term, query *https://jsonmock.hackerrank.com/api/moviesdata/search/?Title=[substr].* The query response will be a JSON object with the following five fields:

- *page*: The current page.
- *per_page*: The maximum number of results per page.
- *total*: The total number of movies having the substring *substr* in their title.
- *total_pages*: The total number of pages which must be queried to get all the results.
  data: An array of JSON objects containing movie information where the *Title* field denotes the title of the movie.

The function will return the integer value found in the *total* field in the returned JSON object.

Function Description

Complete the function *getNumberOfMovies* in the editor below.

getNumberOfMovies has the following parameter(s):

str substr: the string to search for in the movie database

Returns

int: the value of the total field in the returned JSON object

Constraints

0 < |substr| < 20

▼

Input from stdin will be processed as follows and passed to the function.

The only line contains the string *substr*.

▼

Sample Input 0

```
STDIN       Function

maze    →   substr = 'maze'
```

Sample Output 0

```
37
```

Explanation 0

The value of *substr* is *maze*, so our query is *https://jsonmock.hackerrank.com/api/moviesdata/search/?Title=maze* and the response is:

```
{
  "page": 1,
  "per_page": 10,
  "total": 37,
  "total_pages": 4,
  "data": [
    {
```

"Title": "The Maze Runner",

```
    },
    {
      "Title": "Maze Runner: The Scorch Trials",
      "Year": 2015,
      "imdbID": "tt4046784"
    },
    {
      "Title": "Into the Grizzly Maze",
      "Year": 2015,
      "imdbID": "tt1694021"
    },
    {
      "Title": "Hercules in the Maze of the Minotaur",
      "Year": 1994,
      "imdbID": "tt0110018"
    },
    {
      "Title": "The Crystal Maze",
      "Year": 1990,
      "imdbID": "tt0098774"
    },
    {
      "Title": "The Maze",
      "Year": 2010,
      "imdbID": "tt1675758"
    },
    {
      "Title": "Maze",
      "Year": 2000,
      "imdbID": "tt0246072"
    },
    {
      "Title": "Iron Maze",
      "Year": 1991,
      "imdbID": "tt0102128"
    },
    {
      "Title": "The Maze",
      "Year": 1953,
      "imdbID": "tt0046057"
    },
    {
      "Title": "Maze Runner: The Burn Trials",
      "Year": 2015,
      "imdbID": "tt4844320"
    }
  ]
}
```


Return the value of the *total* field, 37, as the answer.

CANDIDATE ANSWER

No answer was submitted for this question. Showing compiled/saved versions.

Language used: **Java 8**

```
public class Solution {
    /*
     * Complete the function below.
     */
    static int getNumberOfMovies(String substr) {
        /*
         * Endpoint: "https://jsonmock.hackerrank.com/api/moviesdata/search/?Title=substr"
*/
```

```
10
    }
```

No Comments

PDF generated at: 7 Feb 2023 18:02:53 UTC