Debugging question----------
Q. Fix the code to find sum of the factorial of all odd numbers

```
import java.io.*;
import java.util.*;
import java.lang.Math;


public class Solution {
    public static long factorial(int n){
        long fact = 1;
        for(int i=1; i<=n; i++){
            fact*=i;
        }
        return fact;
    }
    public static long buggySumOfOddFactorials(int n, List<Integer> arr) {
        // Fix the code here
        long sum = 0;
        for (int i = 0; i < arr.size(); i++) {
            if (arr.get(i) % 2 == 1) {
                sum += factorial(arr.get(i));
            }
        }
        return sum;
    }
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<Integer> arr = new ArrayList<>(n);
        for(int j=0; j<n; j++) {
            arr.add(Integer.parseInt(scan.nextLine().trim()));
        }

        long result = buggySumOfOddFactorials(n, arr);

        System.out.println(result);
    }
}
```

---
Javascript question
Q. find the final status of the package

```
function solve(statuses) {
    const statusList = statuses.split(';');
    return statusList[statusList.length-1];
    // Write your code here

}


const statuses = gets();

const result = solve(statuses);

print(result)
```

---
SQL Question
Q. Course Enrollment

```
-- Enter your query here
-- Note: MySQL queries are case-sensitive. To ensure correctness of the code, please follow the same standard.
SELECT c.course_name,
    COUNT(e.student_id) AS student_count
FROM courses c
LEFT JOIN
    enrollments e ON c.course_id=e.course_id
GROUP BY
    c.course_id,c.course_name
ORDER BY
    c.course_id;
```

---
Spring Security Question
Q. Student Management Microservice

StudentController.java

```
package com.student.api.controller;

import com.student.api.domain.Student;
import org.springframework.dao.EmptyResultDataAccessException;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
* REST controller for managing student system process. Use {@link StudentRowMapper} to
map database rows to Student entity object.
*/

@RestController
@RequestMapping("/api/v1")
public class StudentController {

    // use JdbcTemplate to query for students aganist database
    private final NamedParameterJdbcTemplate jdbcTemplate;

    public StudentController(NamedParameterJdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    /**
     * {@code GET  /students} : get all the Students.
     *
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and the list
     * of students in body.
     */
    @GetMapping("/students")
    public ResponseEntity<List<Student>> getAllStudents() {

        List<Student> students = jdbcTemplate.query("SELECT * FROM student", new
StudentRowMapper());
        return ResponseEntity.ok().body(students);
        //return ResponseEntity.ok().body(null);
    }

    /**
     * {@code GET  /students/:id} : get the "id" Student.
     *
     * @param id the id of the student to retrieve.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body
```

```
     * the student, or if does not exist, return with status "noContent".
     */
    // @GetMapping("/students/{id}")
    // public ResponseEntity<Student> getStudent(@PathVariable Long id) {
        /**
     * {@code GET  /students/:id} : get the "id" Student.
     *
     * @param id the id of the student to retrieve.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body
     * the student, or if does not exist, return with status "noContent".
     */

    @GetMapping("/students/{id}")
    public ResponseEntity<Student> getStudent(@PathVariable Long id) {
        try {
            Student student = jdbcTemplate.queryForObject("SELECT * FROM student WHERE id
= :id", new MapSqlParameterSource("id", id), new StudentRowMapper());
            return ResponseEntity.ok().body(student);
        } catch (EmptyResultDataAccessException e) {
            return ResponseEntity.noContent().build();
        }
        // return ResponseEntity.ok().body(null);
    }



    /**
     * {@code POST  /student} : Create a new student.
     *
     * @param student the student to create.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and with
     * body the new student
     */
    @PostMapping("/students")
    public ResponseEntity<Void> createStudent(@RequestBody Student student) {
        int rowsAffected = jdbcTemplate.update("INSERT INTO student (id, name) VALUES (:id,
:name)",
            new MapSqlParameterSource()
                .addValue("id", student.getId())
                .addValue("name", student.getName()));
        return rowsAffected > 0 ? ResponseEntity.ok().build() : ResponseEntity.noContent().build();
    }

    // @PostMapping("/students")
```

```
    // public ResponseEntity<Void> createStudent(@RequestBody Student student) {

    //     return ResponseEntity.ok().build();
    // }

    /**
     * {@code PUT  /student} : Updates an existing student.
     *
     * @param student the student to update.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body
     * the updated student.
     */



     @PutMapping("/students")
     public ResponseEntity<Void> updateStudent(@RequestBody Student student) {
        int rowsAffected = jdbcTemplate.update("UPDATE student SET name = :name WHERE id
= :id",
            new MapSqlParameterSource()
                .addValue("id", student.getId())
                .addValue("name", student.getName()));
        return rowsAffected > 0 ? ResponseEntity.ok().build() :
ResponseEntity.noContent().build();
     }

    /**
     * {@code DELETE  /student/:id} : delete the "id" student.
     *
     * @param id the id of the student to delete.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)}.
     */
    // @DeleteMapping("/students/{id}")
    // public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {

         /**
     * {@code DELETE  /student/:id} : delete the "id" student.
     *
     * @param id the id of the student to delete.
     * @return the {@link ResponseEntity} with status {@code 200 (OK)}.
     */

     @DeleteMapping("/students/{id}")
     public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {
```

```
        int rowsAffected = jdbcTemplate.update("DELETE FROM student WHERE id = :id", new
MapSqlParameterSource("id", id));
        return rowsAffected > 0 ? ResponseEntity.ok().build() :
ResponseEntity.noContent().build();
    }

}


        // jdbcTemplate.update("DELETE FROM student WHERE id =?", id);
```

---

---

StudentRowMapper.java

```
package com.student.api.controller;

import com.student.api.domain.Student;
import org.springframework.jdbc.core.RowMapper;

import java.sql.ResultSet;
import java.sql.SQLException;

public class StudentRowMapper implements RowMapper<Student> {
    /**
     *
     * @param rs Database ResultSet object. Get database data with the column names "ID" and
"NAME". Remember, "ID" column is Long data type and "NAME" column is String data type.
     * @param rowNum If you get data with column names as described above, you don't need to
use rowNum parameter
     * @return Student object with the mapped values from database
     */


    @Override
    public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
        Long id = rs.getLong("ID");
        String name = rs.getString("NAME");
        return new Student(id, name);
    }
}
```

---

REACT QUESTION
Q. Build a Sales Dashboard Application

Dashboard.jsx

```
import axios from "axios";
import React, { useEffect , useState} from "react";

import "./Dashboard.css";
import { calculateTotalSales, calculateTotalCashSale, calculateTotalCreditSale,
calculateBuyerWithMostSale} from './Reports';

function Dashboard(){
const App=()=>{
  const[data, setData]=useState([]);
  useEffect(()=>{
    (async()=>{
      const result=await axios.get('/sales.json');
      setData(result.data);
    })();
  },[])

  return (
    <div className="dashboard">
      <div className="card">
        <h2>Total Sales</h2>
        <p>{calculateTotalSales(data)}</p>
      </div>
      <div className="card">
        <h2>Total Cash Sales</h2>
        <p>{calculateTotalCashSale(data)}</p>
      </div>
      <div className="card">
        <h2>Total Credit Sales</h2>
        <p>{calculateTotalCreditSale(data)}</p>
      </div>
      <div className="card">
        <h2>Buyer with Most Sales</h2>
        <p>{calculateBuyerWithMostSale(data).buyerName}</p>
        <p>{calculateBuyerWithMostSale(data).saleTotal}</p>
      </div>
    </div>
```

```
  );
}
}
export default Dashboard;
```

---

Reports.js

```
import axios from "axios";

export const getSalesData = async () => {
  let { data } = await axios.get(`/sales.json`);
  return data;
};

export const calculateTotalSales = (sales) => {
 return sales.reduce((total, sale)=> total+sale.saleTotal, 0);
};

export const calculateTotalCashSale = (sales) => {
  return sales.filter(sale=>sale.creditCard===false)
  .reduce((total,sale)=>total+sale.saleTotal,0);

};

export const calculateTotalCreditSale = (sales) => {
  return sales.filter(sale=>sale.creditCard===true)
  .reduce((total, sale)=>total+sale.saleTotal,0);
};

export const calculateBuyerWithMostSale = (sales) => {
 const buyerMap={};
 for(const sale of sales){
  if(!buyerMap[sale.buyerName]){
    buyerMap[sale.buyerName]=0;
  }
  buyerMap[sale.buyerName]+=sale.saleTotal;
 }
 let maxBuyer=null;
 let maxTotal=0;
 for(const[buyer,total] of Object.entries(buyerMap)){
  if(total>maxTotal){
    maxBuyer=buyer;
```

```
    maxTotal=total;
  }
 }
 return {
  buyerName:maxBuyer,
  saleTotal:maxTotal
 };

};
```

---

HTML/CSS/JS Question
Q. Bank Management System Form

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Transaction Filter</title>
 <style>
  body {
   background-color: gray;
  }
 </style>
</head>
<body>
 <div>
  <select id="transactionType">
   <option value="all">All</option>
   <option value="deposit">Deposits</option>
   <option value="withdrawal">Withdrawals</option>
  </select>
  <ul id="transactionList"></ul>
 </div>
</body>

</html>
```

---

index.css

```
body {
```

```
  background-color: gray;
}
```

---

index.js

```
const transactions = [
 { type: "deposit", amount: 100 },
 { type: "withdrawal", amount: 50 },
 { type: "deposit", amount: 200 },
 { type: "withdrawal", amount: 30 },
 { type: "deposit", amount: 150 }
];

function filterTransactions(type, container) {
 container.innerHTML = ""; // Clear previous entries

 const filtered = type === "all"
  ? transactions
  : transactions.filter(txn => txn.type === type);

 filtered.forEach(txn => {
  const li = document.createElement("li");
  li.textContent = `${txn.type.toUpperCase()}: $${txn.amount}`;
  container.appendChild(li);
 });

}
module.exports = filterTransactions;
```

---

Reactjss

React 1

React 1 : Property

AddProperty.js:

```
import React, { useState } from 'react';
import PropertyService from './PropertyService';
```

```
import './App.css';

const AddProperty = () => {
  const [property, setProperty] = useState({
    _id: '',
    type: '',
    location: '',
    price: '',
    rooms: '',
    size: ''
  });

  const [error, setError] = useState(null);
  const [message, setMessage] = useState('');

  const handleChange = (e) => {
    setProperty({ ...property, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);
    try {
      await PropertyService.addProperty(property);
      setMessage('Property added successfully!');
    } catch (err) {
      setError(err.message);
    }
  };

  return (
<div className="add-property-container">
<h2>Add New Property</h2>
      {error && <p className="error">{error}</p>}
      {message && <p className="success">{message}</p>}
<form onSubmit={handleSubmit}>
        {['type', 'location', 'price', 'rooms', 'size', '_id'].map((field) => (
<div key={field}>
<label>{field}:</label>
<input
              name={field}
              value={property[field]}
              onChange={handleChange}
              required
```

```
          />
</div>
       ))}
<button type="submit">Add Property</button>
</form>
</div>
  );
};

export default AddProperty;
```

PropertyList.js :

```
import React, { useState, useEffect } from 'react';
import PropertyService from './PropertyService';
import { Link } from 'react-router-dom';
import './App.css';

const PropertyList = () => {
  const [properties, setProperties] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchProperties = async () => {
      try {
        const data = await PropertyService.getAllProperties();
        setProperties(data);
        setLoading(false);
      } catch (err) {
        setError(err.message);
        setLoading(false);
      }
    };

    fetchProperties();
  }, []);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;
```

```
  return (
<div className="property-list-container">
<h2 className="property-list-header">Properties List</h2>
<ul className="property-list">
        {properties.map((property) => (
<li key={property._id}>
<Link to={`/properties/${property._id}`}>
              {property.location} - {property.type}
</Link>
</li>
        ))}
</ul>
</div>
  );
};

export default PropertyList;
```

PropertyDetail.js :

```
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import PropertyService from './PropertyService';
import './App.css';

const PropertyDetail = () => {
  const { propertyID } = useParams();
  const [property, setProperty] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchProperty = async () => {
      try {
        const data = await PropertyService.getPropertyByID(propertyID);
        setProperty(data[0]); // use first element
        setLoading(false);
      } catch (err) {
        setError(err.message);
        setLoading(false);
```

```
    }
   };

   fetchProperty();
  }, [propertyID]);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
<div className="property-detail-container">
<h2>Property Details</h2>
<p>Type: {property.type}</p>
<p>Location: {property.location}</p>
<p>Price: {property.price}</p>
<p>Rooms: {property.rooms}</p>
<p>Size: {property.size}</p>
</div>
  );
};

export default PropertyDetail;
```

PropertyService.js :

```
const API_URL = `http://localhost:3000/properties`;

const PropertyService = {
  getAllProperties: async () => {
    const response = await fetch(API_URL);
    if (!response.ok) {
      throw new Error('Failed to fetch properties');
    }
    return response.json();
  },

  getPropertyByID: async (propertyID) => {
    const response = await fetch(`${API_URL}?_id=${propertyID}`);
    if (!response.ok) {
      throw new Error('Failed to fetch property details');
```

```
    }
    return response.json(); // returns an array
  },

  addProperty: async (newProperty) => {
    const response = await fetch(API_URL, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(newProperty),
    });
    if (!response.ok) {
      throw new Error('Failed to add property');
    }
    return response.json();
  },
};

export default PropertyService;
```

---

React 2

React Project 2 : Patient Information

PatientInformation.js :

```
import React, { useState, useEffect } from 'react';
import { getPatients } from './PatientService';
import './App.css';

export const PatientInformation = ({ patientID }) => {
  const [patient, setPatient] = useState(null);

  useEffect(() => {
    const fetchPatient = async () => {
      const patients = await getPatients();
      const found = patients.find(p => p.patientID === patientID);
      setPatient(found || null);
    };

    if (patientID) {
      fetchPatient();
    }
  }, [patientID]);
```

```
  return (
    <div className="patient-info-container">
      {patient ? (
        <div className="patient-card">
          <h3>Patient Details</h3>
          <p>Patient ID: {patient.patientID}</p>
          <p>Name: {patient.name}</p>
          <p>Age: {patient.age}</p>
          <p>Gender: {patient.gender}</p>
          <p>Condition: {patient.condition}</p>
          <p>Last Visit: {patient.lastVisit}</p>
        </div>
      ) : (
        <p>No patient found for ID: {patientID}</p>
      )}
    </div>
  );
};
```

PatientRegistrationForm.js :

```
import React, { useState } from 'react';
import { addPatient } from './PatientService';
import './App.css';

const PatientRegistrationForm = ({ onRegister }) => {
  const [errors, setErrors] = useState({});
  const [formData, setFormData] = useState({
    name: '',
    age: '',
    gender: '',
    condition: '',
    lastVisit: '',
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };
```

```
  const isValidDate = (dateString) => {
    const regex = /^\d{4}-\d{2}-\d{2}$/;
    return regex.test(dateString);
  };

  const validateForm = () => {
    const errs = {};
    if (!formData.name.trim()) errs.name = 'Name is required';
    if (!formData.age) errs.age = 'Age is required';
    else if (isNaN(formData.age) || formData.age <= 0) errs.age = 'Age must be a positive
number';
    if (!formData.gender) errs.gender = 'Gender is required';
    if (!formData.condition.trim()) errs.condition = 'Condition is required';
    if (!formData.lastVisit.trim()) errs.lastVisit = 'Last Visit is required';
    else if (!isValidDate(formData.lastVisit)) errs.lastVisit = 'Invalid date format (YYYY-MM-DD)';

    setErrors(errs);
    return Object.keys(errs).length === 0;
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!validateForm()) return;

    const newPatient = {
      ...formData,
      patientID: `P${Date.now().toString().slice(-4)}`
    };

    await addPatient(newPatient);
    if (onRegister) {
      onRegister(formData); // matches test expectation
    }
    setFormData({ name: '', age: '', gender: '', condition: '', lastVisit: '' });
    setErrors({});
  };

  return (
    <form className="patient-form" onSubmit={handleSubmit}>
      <h3>Register New Patient</h3>
      <input name="name" placeholder="Name" value={formData.name}
onChange={handleChange} />
      {errors.name && <div className="error">{errors.name}</div>}
```

```
      <input name="age" placeholder="Age" value={formData.age} onChange={handleChange} />
      {errors.age && <div className="error">{errors.age}</div>}

      <select name="gender" value={formData.gender} onChange={handleChange}>
        <option value="">Select Gender</option>
        <option>Male</option>
        <option>Female</option>
        <option>Other</option>
      </select>
      {errors.gender && <div className="error">{errors.gender}</div>}

      <input name="condition" placeholder="Condition" value={formData.condition}
onChange={handleChange} />
      {errors.condition && <div className="error">{errors.condition}</div>}

      <input name="lastVisit" placeholder="Last Visit (YYYY-MM-DD)" value={formData.lastVisit}
onChange={handleChange} />
      {errors.lastVisit && <div className="error">{errors.lastVisit}</div>}

      <button type="submit">Register Patient</button>
    </form>
  );
};

export default PatientRegistrationForm;
```

PatientService.js :

```
import environment from "./environments/environment.ts"
const API_URL = environment.apiUrl;

export const getPatients = async () => {
    const response = await fetch(`${API_URL}/patients`);
    if (!response.ok) throw new Error("Failed to fetch patients");
    return await response.json();
};

export const addPatient = async (newPatient) => {
    const response = await fetch(`${API_URL}/patients`, {
        method: 'POST',
```

```
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(newPatient)
    });

    if (!response.ok) throw new Error("Failed to add patient");
    return await response.json();
};
```

---

HTML--- QQQQ

```
<!DOCTYPE html>
<html>
  <head>
    <title>Online Banking: Account Transactions Viewer</title>
    <style>
      body {
        background-color: #f0f0f0;
      }

      form {
        display: flex;
        flex-direction: column;
        width: 50%;
        justify-content: center;
        align-items: center;
        border: 1px solid #fff;
        margin: 0 auto;
        padding: 10px;
      }

      div {
        width: 50%;
        display: flex;
        justify-content: center;
        margin: 4rem auto;
      }

      label {
        width: 20%;
        font-size: 1.2rem;
```

```
    }

    select {
      width: 20%;
    }

    table {
      font-family: arial, sans-serif;
      border-collapse: collapse;
      width: 100%;
    }

    td,
    th {
      border: 1px solid #dddddd;
      text-align: left;
      padding: 8px;
    }

    tr.deposit {
      background-color: #d4edda;
      color: #155724;
    }

    tr.withdrawl {
      background-color: #f8d7da;
      color: #721c24;
    }

    a:hover {
      color: orange;
    }
  </style>
</head>
<body>
  <h2>Online Banking: Account Transactions Viewer</h2>

  <div>
    <label for="type">Transaction Type</label>
    <select id="type">
      <option value="">All</option>
      <option value="DEPOSIT">DEPOSIT</option>
      <option value="WITHDRAWL">WITHDRAWL</option>
    </select>
```

```
  <button id="search-btn">Search</button>
</div>

<div>
  <table>
    <thead>
      <tr>
        <th>Description</th>
        <th>Amount</th>
        <th>Type</th>
      </tr>
    </thead>
    <tbody id="transactionTableBody"></tbody>
  </table>
</div>

<script type="text/javascript">
  // Do not change these hardcoded transactions
  const transactions = [
    {
      description: "Transfer to Mr A",
      amount: 1000,
      type: "WITHDRAWL",
    },
    {
      description: "Salary March 2022",
      amount: 50000,
      type: "DEPOSIT",
    },
    {
      description: "House Rent",
      amount: 4000,
      type: "WITHDRAWL",
    },
    {
      description: "Receive from Mr B",
      amount: 2000,
      type: "DEPOSIT",
    },
  ];

  const transactionTableBody = document.getElementById("transactionTableBody");
  const searchBtn = document.getElementById("search-btn");
  const dropdown = document.getElementById("type");
```

```
    // Populate transactions based on selected type
    searchBtn.addEventListener("click", (e) => {
      e.preventDefault();
      const selectedType = dropdown.value;
      populateTransactions(selectedType);
    });

    function populateTransactions(selectedType = "") {
      transactionTableBody.innerHTML = "";

      const filteredTransactions = getTransactions(selectedType);

      filteredTransactions.forEach((transaction) => {
        const row = document.createElement("tr");
        row.className = transaction.type.toLowerCase();

        row.innerHTML = `
          <td>${transaction.description}</td>
          <td>${transaction.amount}</td>
          <td>${transaction.type}</td>
        `;
        transactionTableBody.appendChild(row);
      });
    }

    function getTransactions(selectedType) {
      if (selectedType === "") {
        return transactions;
      }
      return transactions.filter((transaction) => transaction.type === selectedType);
    }

    // Populate all transactions initially
    populateTransactions();
  </script>
 </body>
</html>
```

---

Another HTML Q: Student ID Employee ID in cancelled exam

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Course Registration</title>
<style>

body{font-family:'Arial',sans-serif;background-color:#f5f5f5;margin:0;padding:0;display:flex;justif
y-content:center;align-items:center;min-height:100vh}
    .registration-form{background-color:white;padding:30px;border-radius:8px;box-shadow:0 4px
6px rgba(0,0,0,0.1);width:100%;max-width:500px}
    .form-group{margin-bottom:20px}
    label{display:block;margin-bottom:8px;font-weight:600;color:#333}
    input{width:100%;padding:12px;border:1px solid
#ddd;border-radius:4px;font-size:16px;box-sizing:border-box}
    input:focus{outline:none;border-color:#4a90e2}
    .error{color:#e74c3c;font-size:14px;margin-top:5px}
    .error-input{border-color:#e74c3c!important}
    .submit-button{background:linear-gradient(to
right,#4a90e2,#3a7bd5);color:white;border:none;padding:12px
20px;border-radius:4px;cursor:pointer;font-size:16px;width:100%;transition:background 0.3s
ease}
    .submit-button:hover{background:linear-gradient(to right,#3a7bd5,#2a5bb5)}

.success-message{color:#2ecc71;background-color:rgba(46,204,113,0.1);padding:15px;border-r
adius:4px;margin-bottom:20px;text-align:center;font-weight:600;border:1px solid
#2ecc71;display:none}
</style>
</head>
<body>
<div class="registration-form">
<form id="registration-form">
<div id="success-message" class="success-message" style="display:none">Registration
successful!</div>
<div class="form-group">
<label for="studentID">Student ID</label>
<input type="text" id="studentID" placeholder="Enter your student ID">
<div id="studentID-error" class="error"></div>
</div>
<div class="form-group">
<label for="email">Email</label>
<input type="email" id="email" placeholder="Enter your email">
<div id="email-error" class="error"></div>
</div>
<div class="form-group">
```

```
<label for="password">Password</label>
<input type="password" id="password" placeholder="Enter your password">
<div id="password-error" class="error"></div>
</div>
<button type="submit" class="submit-button">Submit</button>
</form>
</div>

 <script>
  const registrationForm=document.getElementById("registration-form"),
       studentID=document.getElementById("studentID"),
       email=document.getElementById("email"),
       password=document.getElementById("password"),
       studentIDError=document.getElementById("studentID-error"),
       emailError=document.getElementById("email-error"),
       passwordError=document.getElementById("password-error"),
       successMessage=document.getElementById("success-message");

  registrationForm.addEventListener("submit",function(e){
   e.preventDefault();
   let isValid=true;
   // Reset error messages and styles
   studentIDError.textContent=emailError.textContent=passwordError.textContent="";
   successMessage.style.display="none";
   studentID.classList.remove("error-input");
   email.classList.remove("error-input");
   password.classList.remove("error-input");

   // Validate Student ID
   const studentIDValue=studentID.value.trim(),
        studentIDRegex=/^[a-zA-Z0-9]*$/;
   if(!studentIDValue){
    studentIDError.textContent="Student ID is required.";
    studentID.classList.add("error-input");
    isValid=false;
   }else if(!studentIDRegex.test(studentIDValue)){
    studentIDError.textContent="Student ID can only contain letters and numbers.";
    studentID.classList.add("error-input");
    isValid=false;
   }

   // Validate Email
   const emailValue=email.value.trim(),
        emailRegex=/^[^\s@]+@[^\s@]+\.[^\s@]+$/;
```

```
    if(!emailValue){
      emailError.textContent="Email is required.";
      email.classList.add("error-input");
      isValid=false;
    }else if(!emailRegex.test(emailValue)){
      emailError.textContent="Please enter a valid email address.";
      email.classList.add("error-input");
      isValid=false;
    }

    // Validate Password
    const passwordValue=password.value,

passwordRegex=/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$/;
    if(!passwordValue){
      passwordError.textContent="Password is required.";
      password.classList.add("error-input");
      isValid=false;
    }else if(!passwordRegex.test(passwordValue)){
      passwordError.textContent="Password must be at least 8 characters long and contain at
least one uppercase letter, one lowercase letter, one number, and one special character.";
      password.classList.add("error-input");
      isValid=false;
    }

    // If valid, show success
    if(isValid){
      successMessage.style.display="block";
      registrationForm.reset();
    }
  });
</script>
</body>
</html>
```

---

Spring mock 6 questions

---

1. Writing a transaction service in java
//TransactionService.java

```
package com.tasks.problem;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.Reader;
import java.util.List;
import com.opencsv.bean.CsvToBean;
import com.opencsv.bean.CsvToBeanBuilder;
public class TransactionService {

	List<Transaction> transactions;


	public List<Transaction> getAllTransactions() {
		try {
			Reader reader = new BufferedReader(new
FileReader("transactions.csv"));
			//@todo You can initialize CsvBean<T> class and then call the parse
method to get the list



			// TODO Auto-generated catch block

	}

	public Double getTotalTransactionAmount() {
		if(transactions == null) {
			getAllTransactions();
		}

		double tot = 0d;
		//@todo Write code to assign total transaction amount to amt variable
		return null;

	}


	public Transaction getTransactionWithHighestAmount() {
		if(transactions == null) {
			getAllTransactions();
		}

		double tot = 0d;
```

```
                Transaction highestTransaction = null;
        //@todo Write code to get the transaction object with highest amount


           return null;
           }

           public Transaction getTransactionWithLowestmount() {
                if(transactions == null) {
                        getAllTransactions();
                }

                double tot = 0d;
                Transaction lowestTransaction = null;
        //@todo Write code to figure out transaction with lowest amount
                                }

                        }

           return null;
           }

           public Double getAverageTransactionAmount() {
                if(transactions == null) {
                        transactions = getAllTransactions();
                }

        //@todo Write code to get the average of transaction amounts & return it.
        return null;
           }

}
```

---

2. Writing a product service in java

```
//ProductService.java

package com.tasks.problem;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.Reader;
import java.util.ArrayList;
```

```
import java.util.List;

import com.opencsv.bean.CsvToBean;
import com.opencsv.bean.CsvToBeanBuilder;

public class ProductService {

	List<Product> products;


	public List<Product> getAllProducts() {
		try {
			//@todo write code using OpenCSV to read the file products.csv and marshalling it to list of products.




			return products;
		} catch (Exception e) {
			// TODO Auto-generated catch block
			throw new RuntimeException("FAILURE_TO_PROCESS_CSV");
		}

	}

	public List<Product> getProductsWithPriceGreaterThan(Double price){
		if(this.products == null) {
			getAllProducts();
		}
		//@todo Write code to filter products having price greater than the passed price agrument


		return null;
	}

	public Double groupByCategoryAndAggregateValue(String category){
		if(this.products == null) {
			getAllProducts();
		}
		//@todo Write code to group by category argument apssed as method parameter and then retutn the aggregated price of products belonging to the category.
```

```
		double tot = 0d;

		return null;
	}

	public Double calculateAverageOfAllProducts(){
		if(this.products == null) {
			getAllProducts();
		}
		//@todo Write code to evaluate the average of prices of all products

		double tot = 0d;

		return null;
	}

	public Product findProductWithHighestPrice(){
		if(this.products == null) {
			getAllProducts();
		}
		//@todo Write code to fetch the product with highest price

		double tot = 0d;
			}
		*/
		return null;
	}

}
```

---

3. Java LRu Cache

```
//LRUCache.java
package com.tasks.problem;

import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

public class LRUCache<K,V> {

   private  ConcurrentLinkedQueue<K> concurrentLinkedQueue = new
ConcurrentLinkedQueue<K>();
```

```
private  ConcurrentHashMap<K,V> concurrentHashMap = new ConcurrentHashMap<K, V>();

private ReadWriteLock readWriteLock = new ReentrantReadWriteLock();



int maxSize=0;

public LRUCache(final int MAX_SIZE){
   this.maxSize=MAX_SIZE;
}

public V get(K key){


   readWriteLock.readLock().lock();
   try {
   V v=null;
     if(concurrentHashMap.containsKey(key)){
        concurrentLinkedQueue.remove(key);
        v= concurrentHashMap.get(key);
          concurrentLinkedQueue.add(key);
     }


   //@todo return the appropriate object
   return null;
   }finally{
      readWriteLock.readLock().unlock();
   }
}

public int size() {
      readWriteLock.readLock().lock();

      try {
               return concurrentHashMap.size();
      }
      finally{
      readWriteLock.readLock().unlock();
   }
}
```

```
    public void put(K key,V value){

          readWriteLock.writeLock().lock();
        try {
        if(concurrentHashMap.containsKey(key)){
             concurrentLinkedQueue.remove(key);
        }
        while(concurrentLinkedQueue.size() >=maxSize){
          //@todo Get the least used key and delete it
        }
        //@todo Add the key

        //return value;
        } finally{
          readWriteLock.writeLock().unlock();
        }
    }
}
```

---

4. Java Word Analyzer using collections

```
//WordAnalyzerService.java
package com.tasks.problem;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;

public class WordAnalyzerService {
    String fileData;

    String[] words;

    Set<String> set = new HashSet<>();

    Map<String, Long> map = new HashMap<>();

    /**
     * @return number of words present in the file words.txt
     */
    public long readFileAndReturnNoOfWords() {
        StringBuilder sb = new StringBuilder();
```

```
        try (BufferedReader reader = new BufferedReader(new FileReader("words.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                sb.append(line).append(" ");
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
            return 0;
        }
        fileData = sb.toString().trim();
        if (!fileData.isEmpty()) {
            words = fileData.split("\\s+");
            return words.length;
        }
        return 0;
    }

    /*
     * @return the unique words present in the file. These words should be populated in the set
variable declared above.
     */
    public long createSetOfUniqueWordsAndReturnUniqueCount() {
        if (words == null) {
            readFileAndReturnNoOfWords();
        }
        set.clear();
        Collections.addAll(set, words);
        return set.size();
    }

    /**
     * Populate the map variable with key-value mapping of word-count, count representing how
many times the word appeared in the file.
     */
    public void createMapOfWord_Count() {
        if (words == null) {
            readFileAndReturnNoOfWords();
        }
        map.clear();
        for (String word : words) {
            map.put(word, map.getOrDefault(word, 0L) + 1);
        }
    }
```

```
/**
 * @param word - input word
 * @return the number of times the input word appeared in the file
 */
public long getOccurrencesOf(String word) {
    if (map.isEmpty()) {
        createMapOfWord_Count();
    }
    return map.getOrDefault(word, 0L);
}

/**
 * @return top 3 words sorted (desc) by number of occurrences in the file
 */
public List<String> findThreeMostCommonWords() {
    if (map.isEmpty()) {
        createMapOfWord_Count();
    }

    List<Map.Entry<String, Long>> entries = new ArrayList<>(map.entrySet());

    // Sort by frequency descending, then lexicographically ascending
    entries.sort((e1, e2) -> {
        int cmp = Long.compare(e2.getValue(), e1.getValue());
        if (cmp == 0) {
            return e1.getKey().compareTo(e2.getKey());
        }
        return cmp;
    });

    List<String> topWords = new ArrayList<>();
    for (int i = 0; i < Math.min(3, entries.size()); i++) {
        topWords.add(entries.get(i).getKey());
    }
    return topWords;
}

/**
 * Sort the map keys based on key value with most commonly used word at the top.
 * @param hm
 * @return sorted map by value descending
 */
private static Map<String, Long> sortByValue(Map<String, Long> hm) {
    List<Map.Entry<String, Long>> list = new LinkedList<>(hm.entrySet());
```

```
        // Sort by value descending
        list.sort((e1, e2) -> Long.compare(e2.getValue(), e1.getValue()));

        Map<String, Long> sortedMap = new LinkedHashMap<>();
        for (Map.Entry<String, Long> entry : list) {
            sortedMap.put(entry.getKey(), entry.getValue());
        }
        return sortedMap;
    }
}
```

---

5. Java Binary Tree Serialization

```
//SerializeDeserializeBinaryTree.java
package com.tasks.problem;
        if (data == null || data.equals("null")) {
            return null;
        }
        String[] nodes = data.split(",");
        t = 0;
        return helper(nodes);
    }

    private static int t;

    private static TreeNode helper(String[] nodes) {
        if (t >= nodes.length || nodes[t].equals("null")) {
            t++;
            return null;
        }

        // Create node with the current value
        TreeNode root = new TreeNode(Integer.parseInt(nodes[t]));
        t++;
        // Recur for left and right children
        root.left = helper(nodes);
        root.right = helper(nodes);

        return root;
    }

    private static void printPreOrderTraversal(TreeNode node) {
        if (node != null) {
            System.out.print(node.val + " ");
```

```
            printPreOrderTraversal(node.left);
            printPreOrderTraversal(node.right);
        }
    }

    // Testing the implementation
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);

        String serialized = serialize(root);
        System.out.println("Serialized: " + serialized);

        TreeNode deserialized = deserialize(serialized);
        System.out.print("Deserialized: ");
        printPreOrderTraversal(deserialized); // Should print 1 2 4 5 3
    }
}
```

---

6. Working with substrings in java

```
//LongestSubstring.java
package com.tasks.problem;

import java.util.HashMap;

public class LongestSubstring {

    public static int lengthOfLongestSubstring(String str) {
        if (str == null || str.length() == 0) {
            return 0;
        }

        // Map to store the last index of each character seen
        HashMap<Character, Integer> map = new HashMap<>();

        int maxLength = 0;
        int start = 0; // Left boundary of current window
```

```
        for (int end = 0; end < str.length(); end++) {
            char currentChar = str.charAt(end);

            // If character is already seen, move the start pointer
            if (map.containsKey(currentChar)) {
                start = Math.max(start, map.get(currentChar) + 1);
            }

            // Update last seen index of current character
            map.put(currentChar, end);

            // Update maxLength
            maxLength = Math.max(maxLength, end - start + 1);
        }

        return maxLength;
    }

    // Optional main method for local testing
    public static void main(String[] args) {
        String input1 = "ABDEFGABEF";
        System.out.println("Longest substring length: " + lengthOfLongestSubstring(input1)); // 6

        String input2 = "BBBB";
        System.out.println("Longest substring length: " + lengthOfLongestSubstring(input2)); // 1

        String input3 = "ABCDE";
        System.out.println("Longest substring length: " + lengthOfLongestSubstring(input3)); // 5
    }
}
```