1. Hackers Code

```
public class Main {

    public static char findFirstNonRepeating(String name) {
        HashMap<Character, Integer> charCount = new HashMap<>();


        for (char c : name.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }


        for (char c : name.toCharArray()) {
            if (charCount.get(c) == 1) {
                return c;
            }
        }


        return '\0';
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        System.out.println("Enter the first name");
        String firstName = scanner.nextLine();
        char firstNonRepeatingFirstName = 
findFirstNonRepeating(firstName);
        System.out.println("First non-repeating character of first 
name is: " );
        System.out.println(firstNonRepeatingFirstName);


        System.out.println("Enter the last name");
        String lastName = scanner.nextLine();
        char lastNonRepeatingLastName = findFirstNonRepeating(new 
StringBuilder(lastName).reverse().toString());
        System.out.println("Last non-repeating character of last 
name is: ");
        System.out.println(lastNonRepeatingLastName);


    }
}
```

2.Int to Roman

```
import java.util.*;

class HelloWorld {

 public static void main(String[] args) {

     Scanner sc=new Scanner(System.in);

    System.out.println("enter the number");

    int input=sc.nextInt();

    if (input < 1 || input > 3999)

    {

     System.out.println("Invalid Roman Number Value");

     return;

    }

 String s = "";

 while (input >= 1000) {

     s += "M";

     input -= 1000;        }

 while (input >= 900) {

     s += "CM";

     input -= 900;

 }

 while (input >= 500) {

     s += "D";

     input -= 500;

 }

 while (input >= 400) {
     s += "CD";
     input -= 400;
 }
 while (input >= 100) {
     s += "C";
     input -= 100;
 }
 while (input >= 90) {
     s += "XC";
     input -= 90;
 }
 while (input >= 50) {
     s += "L";
     input -= 50;
 }
 while (input >= 40) {
     s += "XL";
     input -= 40;
 }
 while (input >= 10) {
     s += "X";
     input -= 10;
 }
 while (input >= 9) {
    s += "IX";
    input -= 9;
 }
 while (input >= 5) {
    s += "V";
    input -= 5;
 }
 while (input >= 4) {
    s += "IV";
    input -= 4;
 }
 while (input >= 1) {
    s += "I";
    input -= 1;
 }
 System.out.println(s);
 }
}
```

3.OTP Generator

```
package com.user_id;
import java.util.Scanner;

public class UserInterface {
        public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the user id");
    String userId = scanner.nextLine();

    if (userId.length() != 14) {
        System.out.println(userId.length() + " is an invalid length");
```

```
            return;
        }

        if                              (!Character.isUpperCase(userId.charAt(0))                         ||
!Character.isUpperCase(userId.charAt(1))) {
            System.out.println(userId + " is an invalid user id");
            return;
        }

        String mobileNumber = userId.substring(2, 12);
        if (!mobileNumber.matches("[6-9][0-9]{9}")) {
            System.out.println(mobileNumber + " is an invalid mobile number");
            return;
        }

        int age;
        try {
            age = Integer.parseInt(userId.substring(12, 14));
        } catch (NumberFormatException e) {
            System.out.println(userId.substring(12, 14) + " is an invalid age");
            return;
        }

        if (age < 21 || age > 30) {
            System.out.println(age + " is an invalid age");
            return;
        }

        StringBuilder otp = new StringBuilder();
        for (int i = 1; i < mobileNumber.length(); i += 2) {
            otp.append(mobileNumber.charAt(i));
        }
        otp.append(age);

        System.out.println("OTP: " + otp.toString());
    }
}
```

4.Balancing

```
import java.util.*;

public class balancing {
	public static void main(String[] args) {
		Scanner sc= new Scanner(System.in);
		System.out.println("Enter the string");
		String n = sc.nextLine();
		if(!n.matches("[a-zA-Z]+")) {
			System.out.println(n+" is an invalid string");
```

```
                return;
            }
            char[] res = n.toCharArray();
            int sum1 = 0,sum2=0;
            int middle = n.length()/2;

            for(int i=0;i<middle;i++) {
                res[i] = (char)(res[i]+1);
                sum1+= res[i];
            }
            for(int i=middle+(n.length()%2);i<n.length();i++) {
                res[i] = (char)(res[i]+1);
                sum2+= res[i];
            }
            if(sum1 == sum2) {
                System.out.println(n+" is a balanced word");
            }else {
                System.out.println(n+" is not a blanced word");
            }
        }

}
```

5.Bank Transaction

```
import java.util.*;

public class InvalidTransactionException extends Exception {
    public InvalidTransactionException(String message) {
        super(message);
    }

}


import java.util.*;
public class UserInterface {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.println("Enter the account details");
            String input = sc.nextLine();
            String[] details = input.split(":");
            if(details.length!=2) {
                throw new InvalidTransactionException("Invalid
inputs.");
            }
            String accountNumber = details[0];
            String transactionAmountStr = details[1];
            int transactionAmount =
Integer.parseInt(transactionAmountStr);
```

```
	if(accountNumber.length()!=4||accountNumber.charAt(0)!='9') {
				throw new InvalidTransactionException("Account 
number should start with 9.");
			}
			if(transactionAmount<1 || transactionAmount>99999) {
				throw new InvalidTransactionException("Invalid 
transaction Amount.");
			}
			double transactionFee = 
calculateTransactionFee(transactionAmount);
			if(transactionFee == 0.0) {
				System.out.println("No Transaction charges 
deducted.TRansaction completed.");
			}else {
				System.out.printf("Transaction fee Rs.%2f 
deducted.Transaction completed.%n",transactionFee);
			}
		}catch(InvalidTransactionException e) {
			System.out.println("Invalid inputs.");
		}finally {
			System.out.println("Thanks for using the 
application.");
			sc.close();
		}
	}
public static double calculateTransactionFee(int amount) {
	double feePercentage = 0.0;
	if(amount>=1 && amount <=10000) {
		feePercentage = 0.0;
	}else if(amount>=10001 && amount <=50000) {
		feePercentage = 0.1;
	}else if(amount>=50001 &&amount<=99999) {
		feePercentage = 0.3;
	}
	return(amount*feePercentage)/100;
}
}
```

6.EliteCareClinic

```
public class Patient {
	private String patientId;
    private String patientName;
    private int age;
    private String diseaseType;
    private String patientType;
```

```java
	public Patient(String patientId, String patientName, int age, 
String diseaseType, String patientType) {
		super();
		this.patientId = patientId;
		this.patientName = patientName;
		this.age = age;
		this.diseaseType = diseaseType;
		this.patientType = patientType;
	}

	public Patient() {
		super();
	}

	public String getPatientId() {
		return patientId;
	}

	public void setPatientId(String patientId) {
		this.patientId =patientId;
	}

	public String getPatientName() {
		return patientName;
	}

	public void setPatientName(String patientName) {
		this.patientName = patientName;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getDiseaseType() {
		return diseaseType;
	}

	public void setDiseaseType(String diseaseType) {
		this.diseaseType = diseaseType;
	}

	public String getPatientType() {
		return patientType;
	}

	public void setPatientType(String patientType) {
		this.patientType = patientType;
	}
	public String toString() {
		return patientId + " " + patientName + " " + age + " " + 
diseaseType + " " + patientType;
	}

}


import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class PatientUtil {
	public Stream<Patient> 
retrieveDetailsByDiseaseType(Stream<Patient> patientStream, String 
diseaseType) {

		 //Fill the code
		 return patientStream.filter(patient -> 
patient.getDiseaseType().equalsIgnoreCase(diseaseType));
	       }

	       public List<Patient> 
retrieveDetailsByPatientType(Stream<Patient> patientStream, String 
patientType) {

		  //Fill the code
		  return patientStream.filter(patient -> 
patient.getPatientType().equalsIgnoreCase(patientType)).collect(Coll
ectors.toList());
	       }


	       public Stream<String> 
retrievePatientIdsBasedOnAge(Stream<Patient> patientStream, int 
count) {
		  //Fill the code
		  return patientStream.sorted((p1, p2) -> 
Integer.compare(p2.getAge(), p1.getAge()))
				  .limit(count).map(Patient::getPatientId);
		 }

}
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class UserInterface {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter the total number of patients to
be added to the list");
		int totalPatients = Integer.parseInt(scanner.nextLine());

		List<Patient> patientList = new ArrayList<>();

		System.out.println("Enter the patient details");
		for (int i = 0; i < totalPatients; i++) {
		String[] details = scanner.nextLine().split(",");
		patientList.add(new Patient(details[0], details[1],
Integer.parseInt(details[2]), details[3], details[4]));
		}

		PatientUtil patientUtil = new PatientUtil();

		System.out.println("Enter the Disease Type");
		String diseaseType = scanner.nextLine();
		Stream<Patient> diseaseTypeStream =
patientUtil.retrieveDetailsByDiseaseType(patientList.stream(),
diseaseType);
		List<Patient> diseaseTypeList =
diseaseTypeStream.collect(Collectors.toList());
		if (diseaseTypeList.isEmpty()) {
		System.out.println("No patients with this disease type");
		} else {
		diseaseTypeList.forEach(patient ->
System.out.println(patient));
		}

		System.out.println("Enter the Patient Type
(Inpatient/Outpatient)");
		String patientType = scanner.nextLine();
		List<Patient> patientTypeList =
patientUtil.retrieveDetailsByPatientType(patientList.stream(),
patientType);
		if (patientTypeList.isEmpty()) {
		System.out.println("No patients with this patient type");
		} else {
```

```
            patientTypeList.forEach(patient -> 
System.out.println(patient));
            }

            System.out.println("Enter the number of patients to be 
retrieved from the list");
            int count = Integer.parseInt(scanner.nextLine());
            Stream<String> patientIdStream = 
patientUtil.retrievePatientIdsBasedOnAge(patientList.stream(), 
count);
            List<String> patientIds = 
patientIdStream.collect(Collectors.toList());
            System.out.println("Top " + count + " patients based on 
age");
            patientIds.forEach(System.out::println);

            scanner.close();
            }
}
```

7.Employee Processing Details

```
public class Employee {
      private String name;
    private String jobTitle;
    private String department;
    private double salary;


      public Employee() {
            super();
      }
      public Employee(String name, String jobTitle, String 
department, double salary) {
            super();
            this.name = name;
            this.jobTitle = jobTitle;
            this.department = department;
            this.salary = salary;
      }
      public String getName() {
            return name;
      }
      public void setName(String name) {
            this.name = name;
      }
      public String getJobTitle() {
            return jobTitle;
      }
```

```
	public void setJobTitle(String jobTitle) {
		this.jobTitle = jobTitle;
	}
	public String getDepartment() {
		return department;
	}
	public void setDepartment(String department) {
		this.department = department;
	}
	public double getSalary() {
		return salary;
	}
	public void setSalary(double salary) {
		this.salary = salary;
	}

}
import java.util.List;
import java.util.stream.Stream;
import java.util.stream.Collectors;
import java.util.Comparator;
public class EmployeeUtility {
	public                                                     List<Employee>
retrieveEmployeesByJobTitle(Stream<Employee>  employeeStream,  String
jobTitle) {
        //Fill the code here
              return employeeStream
                     .filter(e                                       ->
e.getJobTitle().equalsIgnoreCase(jobTitle))
                     .collect(Collectors.toList());
          }


    public                                                         List<Employee>
retrieveEmployeesByDepartment(Stream<Employee>       employeeStream,
String department) {
        //Fill the code here

             return employeeStream
                    .filter(e                                        ->
e.getDepartment().equalsIgnoreCase(department))
                    .collect(Collectors.toList());
         }

        //Fill the code here
         public                                                List<Employee>
sortEmployeesBySalaryDescending(Stream<Employee> employeeStream) {
             return employeeStream
```

```
.sorted(Comparator.comparingDouble(Employee::getSalary).reversed())
                        .collect(Collectors.toList());
          }


}
mport java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Main {
      public static void main(String[] args) {
         Scanner scanner = new Scanner(System.in);
         List<Employee> employees = new ArrayList<>();
         System.out.println("Enter the total number of employees to
add:");
         int totalEmployees = scanner.nextInt();
         scanner.nextLine();

         System.out.println("Enter              the             employee
details(format:name,jobTitle,department,salary):");

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

             String[] employeeDetails = scanner.nextLine().split(",");
             employees.add(new            Employee(employeeDetails[0],
employeeDetails[1],                         employeeDetails[2],
Double.parseDouble(employeeDetails[3])));
         }

        EmployeeUtility employeeUtility = new EmployeeUtility();
            // Fill the code here to -Retrieve employees by jobTitle


                 boolean exit = false;
                 while (!exit) {
                     System.out.println("\nMenu:");
                     System.out.println("1. Retrieve employees by job
title");

                     System.out.println("2.  Retrieve  employees  by
department");

                     System.out.println("3. Sort employees by salary
(descending)");

                     System.out.println("4. Exit");
                     System.out.print("Enter your choice: ");

                     int choice = scanner.nextInt();
                     scanner.nextLine(); // Consume newline character
```

```
                        switch (choice) {
                            case 1:
                                System.out.print("Enter the job title to
retrieve employees: ");
                                String              jobTitle             =
scanner.nextLine().trim();

                                List<Employee>    employeesByJobTitle    =
employeeUtility.retrieveEmployeesByJobTitle(employees.stream(),
jobTitle);

                                if (employeesByJobTitle.isEmpty()) {
                                    System.out.println("No employee found
for the given job title");
                                } else {
                                    System.out.println("Employees    with
job title \"" + jobTitle + "\":");
                                    employeesByJobTitle.forEach(e       ->
System.out.println(e.getName()  +  "/"  +  e.getJobTitle()  +  "/"  +
e.getDepartment()));
                                }
                                break;

                            case 2:
                                System.out.print("Enter the department to
retrieve employees: ");
                                String             department            =
scanner.nextLine().trim();

                                List<Employee>   employeesByDepartment   =
employeeUtility.retrieveEmployeesByDepartment(employees.stream(),
department);

                                if (employeesByDepartment.isEmpty()) {
                                    System.out.println("No employee found
for the given department");
                                } else {
                                    System.out.println("Employees      in
department \"" + department + "\":");
                                    employeesByDepartment.forEach(e     ->
System.out.println(e.getName()  +  "/"  +  e.getJobTitle()  +  "/"  +
e.getDepartment()));
                                }
                                break;

                            case 3:
                                List<Employee>     sortedEmployees       =
employeeUtility.sortEmployeesBySalaryDescending(employees.stream());
```

```
                        System.out.println("Sorting  employees  by
salary (descending):");
                        sortedEmployees.forEach(e               ->
System.out.println(e.getName()  +  "/"  +  e.getJobTitle()  +  "/"  +
e.getSalary()));
                        break;

                    case 4:
                        exit = true;
                        break;

                    default:
                        System.out.println("Invalid       choice.
Please enter a number from 1 to 4.");
                }
            }

            scanner.close();
        }

}
```

8.FineCinemas

```
import java.util.List;

import java.util.stream.*;

public class MovieUtil {
     public List<Movie> retrieveMovieDetailsByGenre(Stream<Movie>
movieStream, String genre) {
        return movieStream.filter(movie ->
movie.getGenre().equals(genre)).collect(Collectors.toList());
    }

    public Stream<Movie>
retrieveMovieDetailsByReleaseYear(Stream<Movie> movieStream, int
year) {
        return movieStream.filter(movie -> movie.getReleaseYear() ==
year);
    }

    public Stream<String>
retrieveMovieNameByAverageRating(Stream<Movie> movieStream, int
count) {
        return movieStream.sorted((m1, m2) ->
Double.compare(m2.getAverageRating(),
m1.getAverageRating())).limit(count).map(Movie::getMovieName);
    }
```

```java
}


import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.stream.*;


public class UserInterface {
	public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Movie> movieList = new ArrayList<>();

        System.out.println("Enter the total number of movies to be
added to the list");
        int totalMovies = Integer.parseInt(scanner.nextLine());

        System.out.println("Enter the movie details");
        for (int i = 0; i < totalMovies; i++) {
            String[] details = scanner.nextLine().split(",");
            String movieName = details[0];
            String genre = details[1];
            int releaseYear = Integer.parseInt(details[2]);
            double averageRating = Double.parseDouble(details[3]);
            movieList.add(new Movie(movieName, genre, releaseYear,
averageRating));
        }

        MovieUtil movieUtil = new MovieUtil();

        System.out.println("Enter the Genre");
        String genre = scanner.nextLine();
        List<Movie> moviesByGenre =
movieUtil.retrieveMovieDetailsByGenre(movieList.stream(), genre);
        if (moviesByGenre.isEmpty()) {
            System.out.println("No movies are present in this
Genre");
        } else {
            moviesByGenre.forEach(System.out::println);
        }

        System.out.println("Enter the year");
        int year = Integer.parseInt(scanner.nextLine());
        Stream<Movie> moviesByYear =
movieUtil.retrieveMovieDetailsByReleaseYear(movieList.stream(),
year);
        List<Movie> moviesByYearList =
moviesByYear.collect(Collectors.toList());
```

```
        if (moviesByYearList.isEmpty()) {
            System.out.println("No movies are released in this 
year");
        } else {
            moviesByYearList.forEach(System.out::println);
        }

        System.out.println("Enter the number of movies to be 
retrieved from the list");
        int count = Integer.parseInt(scanner.nextLine());
        Stream<String> topMoviesByRating = 
movieUtil.retrieveMovieNameByAverageRating(movieList.stream(), 
count);
        System.out.println("Top " + count + " movies based on 
average rating");
        topMoviesByRating.forEach(System.out::println);

        scanner.close();
    }
}
```

9.FlightTrackingSystem

```
import java.util.regex.*;
public class FlightTrackingSystem {

           public static void validateUserDetails(String username, 
String password, String role) throws InvalidFlightDetailsException {
           // Validate username
           if (!Pattern.matches("^[a-zA-Z0-9]+@[a-zA-Z0-9]+\\.[a-zA-
Z0-9]+$", username)) {
           throw new InvalidFlightDetailsException("Invalid 
Username. Must be in the format username@domain.ext");
           }

           // Validate password
           if (!Pattern.matches("^[A-Z]{7}\\d{4}$", password)) {
           throw new InvalidFlightDetailsException("Invalid 
Password. Must be exactly 7 uppercase letters followed by 4 
digits.");
           }

           // Validate role
           if (!(role.equalsIgnoreCase("manager") || 
role.equalsIgnoreCase("admin"))) {
           throw new InvalidFlightDetailsException("Invalid Role. 
Must be either 'manager' or 'admin'.");
           }
           }
```

```
        public static String getFlightStatus() {
        return "Bon Voyage";
        }
        }
    import java.util.Scanner;

public class UserInterface {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
        System.out.println("Enter Username:");
        String username = scanner.nextLine();

        System.out.println("Enter Password:");
        String password = scanner.nextLine();

        System.out.println("Enter Role:");
        String role = scanner.nextLine();

        FlightTrackingSystem.validateUserDetails(username,
password, role);

    System.out.println(FlightTrackingSystem.getFlightStatus());
        } catch (InvalidFlightDetailsException e) {
        System.out.println(e.getMessage());
        }
        }

}
```

10.Shipment

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;


public class UserInterface {
    public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter the Shipment Details");
      String input = scanner.nextLine();
      scanner.close();

      try {
          processShipmentDetails(input);
```

```
        } catch (InvalidShipmentException e) {
            System.out.println(e.getMessage());
        } finally {
            System.out.println("Thank you for using the
application");
        }
    }

    private static void processShipmentDetails(String input) throws
InvalidShipmentException {
        String[] details = input.split(":");

        if (details.length != 4) {
            throw new InvalidShipmentException("Invalid input
format");
        }

        String shipmentId = details[0];
        String shipmentDateStr = details[1];
        String shipmentType = details[2];
        double packageWeight = Double.parseDouble(details[3]);

        validateShipmentId(shipmentId);
        validateShipmentDate(shipmentDateStr);
        validateShipmentType(shipmentType);

        double shippingCost = calculateShippingCost(shipmentType,
packageWeight);
        System.out.printf("Shipment Cost Rs. %.2f%n", shippingCost);
    }

    private static void validateShipmentId(String shipmentId) throws
InvalidShipmentException {
        if (!shipmentId.matches("SHP\\d{5}")) {
            throw new InvalidShipmentException("Invalid Shipment
Id");
        }
    }

    private static void validateShipmentDate(String shipmentDateStr)
throws InvalidShipmentException {
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        sdf.setLenient(false);
        try {
            Date shipmentDate = sdf.parse(shipmentDateStr);
        } catch (ParseException e) {
            throw new InvalidShipmentException("Invalid date
format");
        }
    }

    private static void validateShipmentType(String shipmentType)
throws InvalidShipmentException {
        if (!(shipmentType.equals("Air") ||
shipmentType.equals("Ground") || shipmentType.equals("Express"))) {
            throw new InvalidShipmentException("Invalid Shipment
Type");
        }
    }

    private static double calculateShippingCost(String shipmentType,
double packageWeight) {
        double percentage = 0.0;

        switch (shipmentType) {
            case "Air":
                percentage = 0.85;
                break;
            case "Ground":
                percentage = 0.45;
                break;
            case "Express":
                percentage = 0.75;
                break;
        }

        return packageWeight * percentage;
    }
}
```

11.SV Cinemas

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class TicketBooking {
     private Map<String, String> ticketMap = new HashMap<>();

    // Getter and Setter for ticketMap
    public Map<String, String> getTicketMap() {
        return ticketMap;
    }

    public void setTicketMap(Map<String, String> ticketMap) {
        this.ticketMap = ticketMap;
```

```java
    }

    // Method to add ticket booking details
    public void addTicketBookingDetails(String seatNumber, String
ticketType) {
        ticketMap.put(seatNumber, ticketType);
    }

    // Method to find the total count of seat numbers based on
ticket type
    public int findTotalCountOfSeatNumbersBasedOnTicketType(String
ticketType) {
        int count = 0;
        for (String type : ticketMap.values()) {
            if (type.equalsIgnoreCase(ticketType)) {
                count++;
            }
        }
        return count > 0 ? count : -1;
    }

    // Method to search seat numbers by ticket type
    public List<String> searchSeatNumbersByTicketType(String
ticketType) {
        List<String> seatNumbers = new ArrayList<>();
        for (Map.Entry<String, String> entry : ticketMap.entrySet())
{
            if (entry.getValue().equalsIgnoreCase(ticketType)) {
                seatNumbers.add(entry.getKey());
            }
        }
        return seatNumbers;
    }
}
import java.util.*;
import java.util.List;

public class UserInterface {
     public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        TicketBooking ticketBooking = new TicketBooking();

        System.out.println("Enter the number of tickets to be
booked");
        int numberOfTickets = Integer.parseInt(scanner.nextLine());

        System.out.println("Enter the ticket details (Seat Number:
Ticket Type)");
        for (int i = 0; i < numberOfTickets; i++) {
            String[] ticketDetails = scanner.nextLine().split(":");
            String seatNumber = ticketDetails[0].trim();
            String ticketType = ticketDetails[1].trim();
            ticketBooking.addTicketBookingDetails(seatNumber, 
ticketType);
        }

        System.out.println("Enter the ticket type to count the seats 
filled");
        String typeToCount = scanner.nextLine().trim();
        int count = 
ticketBooking.findTotalCountOfSeatNumbersBasedOnTicketType(typeToCou
nt);
        if (count == -1) {
            System.out.println("No tickets were booked in ticket 
type " + typeToCount);
        } else {
            System.out.println("Number of seats filled in ticket 
type " + typeToCount + " is " + count);
        }

        System.out.println("Enter the ticket type to find seat 
numbers");
        String typeToFind = scanner.nextLine().trim();
        List<String> seatNumbers = 
ticketBooking.searchSeatNumbersByTicketType(typeToFind);
        if (seatNumbers.isEmpty()) {
            System.out.println("No tickets were booked in ticket 
type " + typeToFind);
        } else {
            System.out.println("Seat numbers in ticket type " + 
typeToFind + " are");
            for (String seatNumber : seatNumbers) {
                System.out.println(seatNumber);
            }
        }

        scanner.close();
    }

}
```

12.Task Hub

```
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class TaskUtility {
      public List<Task> retrieveTasksByAssignee(Stream<Task>
taskStream, String assignee) {
        return taskStream
                .filter(task ->
task.getAssignee().equalsIgnoreCase(assignee))
                .collect(Collectors.toList());
    }

    public List<Task> retrieveTasksByProject(Stream<Task>
taskStream, String project) {
        return taskStream
                .filter(task ->
task.getProject().equalsIgnoreCase(project))
                .collect(Collectors.toList());
    }

    public List<Task> findHighPriorityTasks(Stream<Task> taskStream,
int priorityLimit) {
        return taskStream
                .sorted((task1, task2) ->
Integer.compare(task1.getPriority(), task2.getPriority()))
                .limit(priorityLimit)
                .collect(Collectors.toList());
    }
}
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.stream.Stream;

public class UserInterface {
      public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Task> taskList = new ArrayList<>();

        System.out.println("Enter the total number of tasks needed
to add in the list");
        int totalTasks = Integer.parseInt(scanner.nextLine());

        System.out.println("Enter the task details");
        for (int i = 0; i < totalTasks; i++) {
            String[] details = scanner.nextLine().split(",");
            taskList.add(new Task(details[0], details[1],
details[2], Integer.parseInt(details[3]), details[4]));
        }

        TaskUtility taskUtility = new TaskUtility();

        System.out.println("Enter the assignee name");
        String assignee = scanner.nextLine();
        List<Task> tasksByAssignee =
taskUtility.retrieveTasksByAssignee(taskList.stream(), assignee);
        if (tasksByAssignee.isEmpty()) {
            System.out.println("No tasks found for the given
assignee");
        } else {
            System.out.println("Tasks assigned to " + assignee +
":");
            tasksByAssignee.forEach(task ->
System.out.println("Project:" + task.getProject() + "/" +
task.getTaskName() + "-" + task.getDescription()));
        }

        System.out.println("Enter the project name");
        String project = scanner.nextLine();
        List<Task> tasksByProject =
taskUtility.retrieveTasksByProject(taskList.stream(), project);
        if (tasksByProject.isEmpty()) {
            System.out.println("No tasks found for the given
project");
        } else {
            System.out.println("Tasks in project " + project + ":");
            tasksByProject.forEach(task ->
System.out.println(task.getTaskName() + "-" +
task.getDescription()));
        }

        System.out.println("Enter the priority threshold for high
priority tasks");
        int priorityLimit = Integer.parseInt(scanner.nextLine());
        List<Task> highPriorityTasks =
taskUtility.findHighPriorityTasks(taskList.stream(), priorityLimit);
        if (highPriorityTasks.isEmpty()) {
            System.out.println("No high priority tasks found");
        } else {
            System.out.println("High priority tasks:");
            highPriorityTasks.forEach(task ->
System.out.println("Project:" + task.getProject() + "/Task:" +
task.getTaskName() + "/Assignee:" + task.getAssignee()));
        }

        scanner.close();
    }
}
```

13.Ticket Master

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
```

```java
import java.time.format.DateTimeParseException;

public class TicketBookingSystem {
     public int validateNumberOfTickets(String tickets) throws
InvalidDataException {
        try {
            int numberOfTickets = Integer.parseInt(tickets);
            if (numberOfTickets < 1 || numberOfTickets > 100) {
                throw new InvalidDataException("Number of tickets
should be between 1 and 100");
            }
            return numberOfTickets;
        } catch (NumberFormatException e) {
            throw new InvalidDataException("Invalid number of
tickets / price");
        }
    }

    public LocalDate validateEventDate(String eventDate) throws
InvalidDataException {
        try {
            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd/MM/yyyy");
            return LocalDate.parse(eventDate, formatter);
        } catch (DateTimeParseException e) {
            throw new InvalidDataException("Invalid event date");
        }
    }

    public String bookTickets(String ticketDetails) throws
InvalidDataException {
        String[] details = ticketDetails.split(":");
        if (details.length != 4) {
            throw new InvalidDataException("Invalid input format");
        }

        try {
            String eventName = details[0];
            LocalDate eventDate = validateEventDate(details[1]);
            int numberOfTickets =
validateNumberOfTickets(details[2]);
            double ticketPrice = Double.parseDouble(details[3]);

            double totalCost = numberOfTickets * ticketPrice;
            return String.format("Total Ticket Cost: %.1f",
totalCost);
        } catch (NumberFormatException e) {
            throw new InvalidDataException("Invalid number of
tickets / price");
```

```
        }
    }

}
import java.util.Scanner;

public class UserInterface {
      public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        TicketBookingSystem bookingSystem = new
TicketBookingSystem();

        System.out.println("Enter event details:");
        String input = scanner.nextLine();

        try {
            String result = bookingSystem.bookTickets(input);
            System.out.println(result);
        } catch (InvalidDataException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("Thank you for booking with
TicketMaster");
        scanner.close();
    }

}
```

14.ZBook

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.Map;
import java.util.Collections;
import java.util.HashMap;

public class BooksInfo {
      private Map<String, Float> bookDetailsMap = new HashMap<>();

    public BooksInfo() {
        bookDetailsMap = new HashMap<>();
    }

    public void addBooksDetails(String bookName, float rating) {
        bookDetailsMap.put(bookName.toLowerCase(), rating);
    }
    public float findBookRating(String bookName) {
        return bookDetailsMap.getOrDefault(bookName.toLowerCase(), -
1f);
    }

    public List<String> findBooksWithHighestRating() {
        List<String> highestRatedBooks = new ArrayList<>();
        for (Map.Entry<String, Float> entry :
bookDetailsMap.entrySet()) {
            if (entry.getValue() == 5f) {
                highestRatedBooks.add(entry.getKey());
            }
        }
        return highestRatedBooks;
    }

    // Getter and setter for bookDetailsMap
    public Map<String, Float> getBookDetailsMap() {
        return bookDetailsMap;
    }

    public void setBookDetailsMap(Map<String, Float> bookDetailsMap)
{
        this.bookDetailsMap = bookDetailsMap;
    }

}


import java.util.List;
import java.util.Scanner;
public class UserInterface {
      public static void main(String args[]) {
            // Fill the code here
               Scanner sc = new Scanner(System.in);
                   System.out.print("Enter number of book details to
be added: ");
                   int numBooks = sc.nextInt();
                   sc.nextLine(); // Consume newline
                   BooksInfo booksInfo = new BooksInfo();
                   for (int i = 0; i < numBooks; i++) {
                       System.out.print("Enter the book details
(bookName : rating): ");
                       String input = sc.nextLine();
                       String[] parts = input.split(":");
                       if (parts.length == 2) {
                           String bookName = parts[0].trim();
```

```
                            float rating =
Float.parseFloat(parts[1].trim());
                            booksInfo.addBooksDetails(bookName,
rating);
                        }
                    }
                    System.out.print("Enter the book name needs to be
searched: ");
                    String searchBook = sc.nextLine();
                    float rating =
booksInfo.findBookRating(searchBook);
                    if (rating != -1) {
                        System.out.println(rating);
                    } else {
                        System.out.println(searchBook + " is not
available in the given book details");
                    }
                    List<String> highestRatedBooks =
booksInfo.findBooksWithHighestRating();
                    if (!highestRatedBooks.isEmpty()) {
                        System.out.println("The names of the books
with the highest rating are:");
                        for (String book : highestRatedBooks) {
                            System.out.println(book);
                        }
                    } else {
                        System.out.println("No books were found with
the highest rating");
                    }
                }

}
```