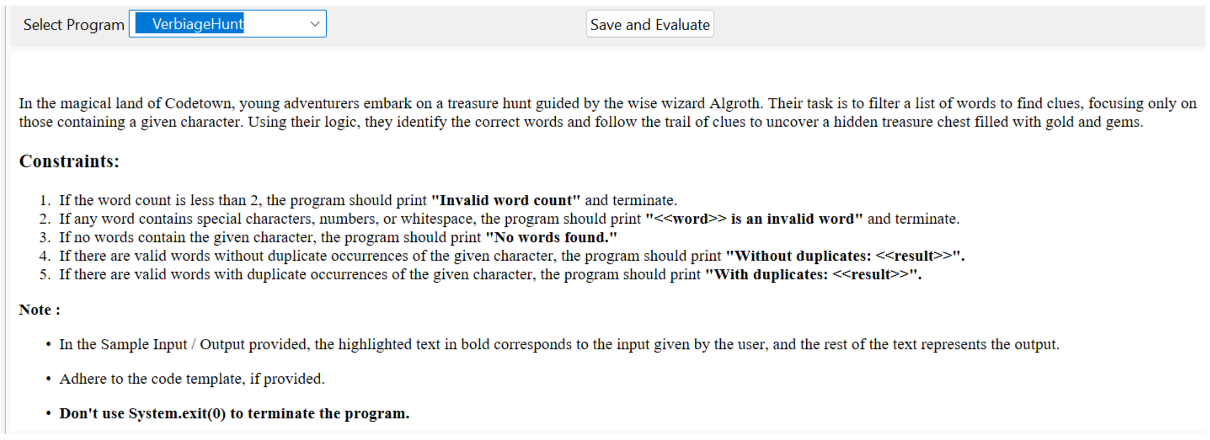


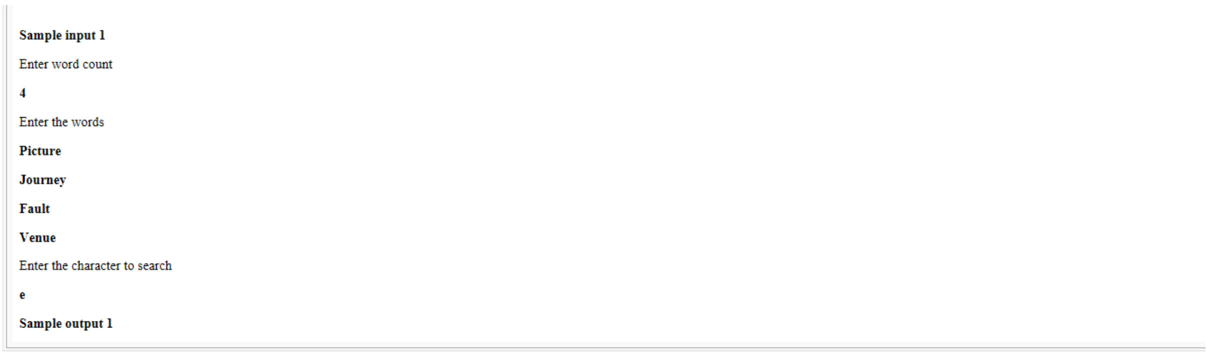
In the magical land of Codetown, young adventurers embark on a treasure hunt guided by the wise wizard Algroth. Their task is to filter a list of words to find clues, focusing only on those containing a given character. Using their logic, they identify the correct words and follow the trail of clues to uncover a hidden treasure chest filled with gold and gems.

Constraints:

1. If the word count is less than 2, the program should print **"Invalid word count"** and terminate.
2. If any word contains special characters, numbers, or whitespace, the program should print **"<<word>> is an invalid word"** and terminate.
3. If no words contain the given character, the program should print **"No words found."**
4. If there are valid words without duplicate occurrences of the given character, the program should print **"Without duplicates: <<result>>".**
5. If there are valid words with duplicate occurrences of the given character, the program should print **"With duplicates: <<result>>".**

Note :

- In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.
- Adhere to the code template, if provided.
- **Don't use System.exit(0) to terminate the program.**



Sample input 1

Enter word count

4

Enter the words

Picture

Journey

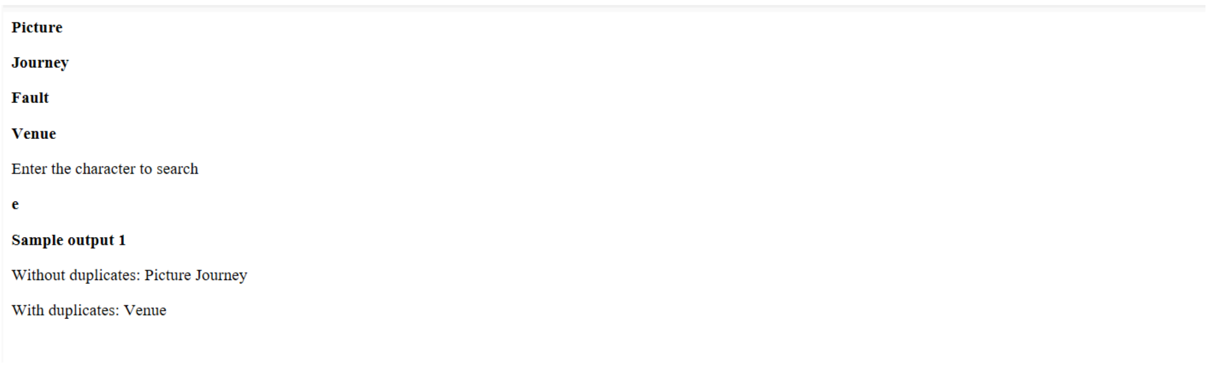
Fault

Venue

Enter the character to search

e

Sample output 1



Picture

Journey

Fault

Venue

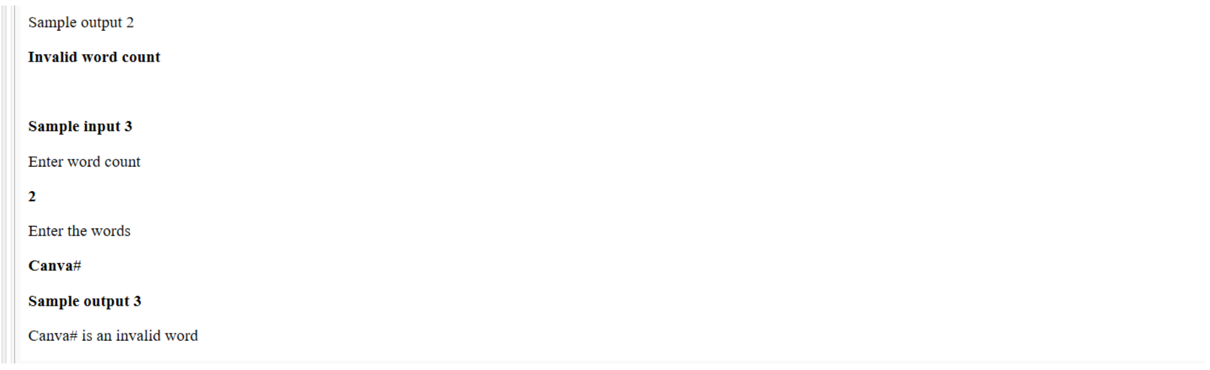
Enter the character to search

e

Sample output 1

Without duplicates: Picture Journey

With duplicates: Venue



Sample output 2

Invalid word count

Sample input 3

Enter word count

2

Enter the words

Canva#

Sample output 3

Canva# is an invalid word

Global Plants Ranking is a popular website for reviewing plants and their medical benefits. They plan to use an online platform to provide plant information and medical benefits based on the ratings. The company hires a software developer to help with their process. You, being the software developer, develop a Java program based on the requirement.

Component Specification: PlantsInfo Class

Type (Class)	Attributes	Methods
PlantsInfo	Map<String, Float> plantDetailsMap	Getter and setter methods for the attribute are included in the code skeleton.

Note: *key: Plant Botanical Name value: Rating for the plantDetailsMap attribute*

Requirement 1: Add the plantName and the rating to the Map.

Type (Class)	Methods	Responsibilities
PlantsInfo	public void addMoviesDetails(String plantName, float rating)	This method should **add** the plantName and rating passed as arguments into the plantDetailsMap.

Requirement 2: Filter the plant record based on the plant's botanical name.

Type (Class)	Methods	Responsibilities
PlantsInfo	public float **findPlantRating**(String plantName)	This method accepts plantName as an argument. If the plant's name is found on the Map, return the plant's rating. Else return -1. ***Condition***: *plantName is case insensitive.*

Requirement 3: Filter the plants based on the highest rating.

Type (Class)	Methods	Responsibilities
PlantsInfo	public List<String> **findPlantsWithHighestRating**()	This method filters the plants and returns the list of plant names that satisfies the below mentioned condition. ***Condition :*** *All the plant records whose rating is equal to 5 are considered to have the highest rating and need to be added to the list.*

- In the **UserInterface** class, the user enters the number of plant details they want to add, then collects the plant details from the user in the form of a string input as per the sample input. Split the string by delimiter and invoke the **addPlantsDetails()** method to add the plant details.
- Then invoke the **findPlantRating()** method to return the rating of the specified plantName.
- If the rating of the specified plantName is not available, then print **"<plantName> is not available in the given plant details"**.
- Invoke the **findPlantsWithHighestRating()** method to retrieve the list of **plantName** with a rating equal to 5.
- If there is no plant available with a rating equal to 5, then print **"No plants were found with the highest rating"**.

Enter number of plant details to be added:

3

Enter the plant details (Plant Name : Rating):

Eclipta prostrata:3

Crataeva religiosa:4

Acalypha indica:5

Enter the plant name needs to be searched:

Eclipta prostrata

3.0

The names of the plants with the highest rating are:

Acalypha indica

Sample Input/Output 2:

Enter number of plant details to be added:

2

Enter the plant details (Plant Name : Rating):

Eclipta prostrata:3

Crataeva religiosa:2

Enter the plant name needs to be searched:

Acalypha indica

Acalypha indica is not available in the given plant details

No plants were found with the highest rating

Spice Sorter, a renowned enterprise in the spice market, is aiming to streamline its spice inventory management using Java Streams. Assist them in automating this process to meet their organisational needs effectively.

Requirements

1. Retrieve spice details for the specified origin.
2. Retrieve spice details in ascending order by shelf life.
3. Retrieve unique spice details.

Component Specification: Spice (POJO class)

Type(Class)	Attributes	Methods
Spice	String spiceName String origin String color int shelfLife	Getters and setters, no argument, and four-argument constructors are given in the code skeleton. The code skeleton also includes hashcode, equals and toString methods.

Component Specification: SpiceUtility

Type (Class)	Methods	Responsibilities
SpiceUtility	public Stream<Spice> **retrieveSpicesByOrigin**(List<Spice> spiceList, String origin)	This method takes a list of spices and a specific origin as input parameters. It filters the spices in the list based on their origin and returns a stream containing only the spices from the specified origin. ***Condition*: *origin*** is case-sensitive
SpiceUtility	public Stream<Spice> **retrieveSpicesInAscendingOrderByShelfLife**(List<Spice> spiceList)	This method takes a list of spices as input, sorts them in ascending order based on their shelf life, and returns a stream of the sorted spices.
SpiceUtility	public Stream<Spice> **retrieveUniqueSpices**(List<Spice> spiceList)	This method takes a list of spices as input and returns a stream containing only the unique spices, removing any duplicates.

The main method in the UserInterface class gets the total number of spices, and their details from the user are provided as a part of the code skeleton. Create an object for the Spice class, set the values to the object, and store all the objects in a list.

The main method in the UserInterface class gets the total number of spices, and their details from the user are provided as a part of the code skeleton. Create an object for the Spice class, set the values to the object, and store all the objects in a list.

- Get the *origin* from the user. Invoke the **retrieveSpicesByOrigin** method to filter the spice by origin. If the spices are available for the given origin, then print the available spices using ***toString()*** method. Otherwise, print ***"No spices found for the given origin <origin>".***
- Invoke the **retrieveSpicesInAscendingOrderByShelfLife** method to sort the spices in ascending order by shelfLife, then print the result using **toString()** method as shown in sample input / output.
- Invoke the **retrieveUniqueSpices** method to get the unique spices, then print the result using **toString()** method as shown in sample input / output.

Note:

- In the sample input / output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.
- Ensure to follow the object-oriented specifications provided in the question description.
- Ensure to provide the names for classes, attributes, and methods as specified in the question description.
- Adhere to the code template, if provided.
- Assume that the number of spices needed to be entered into the list is always a valid positive number.
- Assume that the number of spices to be retrieved from the list is always a valid positive number.
- Assume that the spices details are always valid.
- **Do not use System.exit(x) to terminate the code.**

Sample Input / Output 1

Enter the number of spices

4

Enter the spice details

Anise:Turkey:Black:5

Turmeric:India:Yellow:3

Anise:Turkey:Black:5

Jalapeno:Mexico:Green:4

Enter the origin

India

Spices in the given origin

Spices in the given origin

Turmeric, India, Yellow, 3

Spices in ascending order of shelf life

Turmeric, India, Yellow, 3

Jalapeno, Mexico, Green, 4

Anise, Turkey, Black, 5

Anise, Turkey, Black, 5

Unique spices are

Anise, Turkey, Black, 5

Turmeric, India, Yellow, 3

Jalapeno, Mexico, Green, 4

Sample Input/Output 2

Enter the number of spices

3

Enter the spice details

Chilli:India:Red:2

Cinnamon:Vietnam:Brown:4

Cinnamon:Vietnam:Brown:4

Enter the origin

Mexico

No spices found for the given origin Mexico

Spices in ascending order of shelf life

Chilli, India, Red, 2

Cinnamon, Vietnam, Brown, 4

Enter the origin

Mexico

No spices found for the given origin Mexico

Spices in ascending order of shelf life

Chilli, India, Red, 2

Cinnamon, Vietnam, Brown, 4

Cinnamon, Vietnam, Brown, 4

Unique spices are

Chilli, India, Red, 2

Cinnamon, Vietnam, Brown, 4

In the realm of library management, EliteLibrary provides a solution to automate the validation of membership details and the calculation of membership fees based on user inputs. Exception handling is incorporated to manage invalid inputs effectively.

Consider the following scenario:

You receive input that includes the user's name, library card number, membership type, and password in the format of a string in the **UserInterface** class.

Component Specification: LibraryManagementSystem

Type (Class)	**Method**	**Parameters**	**Responsibilities**
LibraryManagementSystem	**validateMembershipDetails**	String libraryCardNumber, String membershipType, String password	This method validates the membership details according to the specified validation rules. If all details are valid, it returns the string "**Validation successful**". If any one of the details are not valid, throw an **InvalidLibraryException** with the appropriate error message.
LibraryManagementSystem	**calculateMembershipFee**	String membershipType	This method calculates and returns the membership fee based on the type of membership. **Guidelines to calculate the membership fee based on the membershipType:** **membershipType** | **fee** Regular | 100 Premium | 250 VIP | 300 **Note**: • **membershipType is case-sensitive.**

Component Specification: InvalidLibraryException (This class inherits the Exception Class)

Type (Class)	Responsibilities
InvalidLibraryException	Provided with a single-argument constructor: public **InvalidLibraryException(String message)**. It is thrown when the libraryCardNumber, membershipType , or password does not follow the validation rules.

Validation Rules:

1. Library Card Number: Must be exactly six characters. The first two characters must be uppercase alphabets followed by four digits.

2. Membership Type: Must be either "**Regular**", "**Premium**", or "**VIP**" (case-sensitive).

3. Password: Must be exactly 8 characters. The first five characters must be uppercase alphabets, followed by three digits.

Note:

- Propagate the exceptions that occur in the **LibraryManagementSystem** class and handle them in the main method.
- If the library card number is invalid, display the message "**Invalid library card number**".
- If the membership type is invalid, display the message "**Invalid membership type**".
- If the password is invalid, display the message "**Invalid password**".

Note:

- Propagate the exceptions that occur in the **LibraryManagementSystem** class and handle them in the main method.
- If the library card number is invalid, display the message "**Invalid library card number**".
- If the membership type is invalid, display the message "**Invalid membership type**".
- If the password is invalid, display the message "**Invalid password**".
- In the sample input and output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.
- Ensure to follow the object-oriented specifications provided in the question description.
- Ensure to provide the names for classes, attributes, and methods as specified in the question description.
- Adhere to the code template, if provided.
- **Please don't use System.exit(0) to terminate the program.**

Sample Input / Output 1:

Enter the name

John

Enter the library card number

AB1243

Sample Input / Output 1:

Enter the name

John

Enter the library card number

AB1243

Enter the membership type

Regular

Enter the password

ABCDE123

Membership Fee 100.0

Sample Input / Output 2:

Enter the name

Smith

Enter the library card number

CDF124556

Enter the membership type

Premium

Enter the password

MNBVC456

Invalid library card number

Sample Input / Output 3:

Enter the name

Johnson

Enter the library card number

RR9876

Enter the membership type

Gold

Enter the password

SDCDE145

Invalid membership type

Sample Input / Output 4:

Enter the name

Jack

Enter the library card number

ZX9087

Enter the membership type

VIP

Enter the password

SDFG45

Invalid password