QUESTION 1:
Appliance Rental System using ArrayList

Sally runs an appliance rental store and needs a system to manage the rental of various appliances. Each appliance has a unique ID, a name, and a status to indicate whether it is currently rented or not. Sally needs a system to:

- Add new appliances to her inventory.
- Rent appliances if they are found by their ID.
- Remove appliances from her inventory by their ID.
- Track how many appliances are still available for rent after each operation.

Task:

Implement an ApplianceRentalSystem class using an ArrayList to manage the list of appliances. Each appliance should be represented by an Appliance class. The Appliance class should have the following attributes:

- id (int): A unique identifier for the appliance, starting from 301 and increasing sequentially for each new appliance added.
- name (String): The name of the appliance.
- rented (boolean): Status indicating whether the appliance is currently rented (default = false).

The ApplianceRentalSystem class should include the following methods:

Add Appliance:

Implement the method public void addAppliance(String name) to add a new appliance to the inventory list.

- The appliance's ID should be assigned automatically, starting from 301 and incrementing for each new appliance.
- This method should take the appliance name as input and create a new Appliance object with a unique ID and the given name, then add it to the list of appliances.

Rent Appliance:

Implement the method public void rentAppliance(int id) to rent an appliance by its ID.

- If the appliance is found, mark it as rented and print a message indicating that the appliance with the given ID is now rented.

- If the appliance is not found, print a message indicating that the appliance with the given ID is not found.

Remove Appliance:

Implement the method public void removeAppliance(int id) to remove an appliance from the inventory by its ID.

- Search through the list to find the appliance with the matching ID and remove it.
- Print whether the appliance was successfully removed or if it was not found.

Track Available Appliances:

Implement the method public int countNonRentedAppliances() to count and return the number of appliances that are not rented.

- This method will be used to display how many appliances are still available for rent after performing the rent and remove operations.

Input Format:

- First line: An integer n representing the number of appliances to add to the inventory.
- Next n lines: Each line contains the name of an appliance to be added.
- The next line: An integer representing the ID of the appliance to be rented.
- The last line: An integer representing the ID of the appliance to be removed from the inventory.

Output Format:

Current Inventory:

- Print "Appliances in the Inventory:".
- For each appliance in the inventory, print the appliance details in the format: Appliance{id=<id>, name='<name>', rented=<rented>}.

Appliance Rental Status:

- If the appliance is rented, print: "Appliance with ID <id> is rented."
- If the appliance is not found, print: "Appliance with ID <id> not found."

Appliance Removal Status:

- If the appliance is removed, print: "Appliance with ID <id> removed successfully."
- If the appliance is not found, print: "Appliance with ID <id> not found."

Updated Inventory:

- Print "Updated Inventory:".
- For each appliance in the updated inventory, print the appliance details in the format: Appliance{id=<id>, name='<name>', rented=<rented>}.

Total Non-Rented Appliances:

- Print "Total non-rented appliances: <number>", where <number> is the total count of appliances that are not rented.

Sample Input and Output:

Sample 1:

Input:

3

Fridge

Oven

Microwave

302

301

Output:

Appliances in the Inventory:

Appliance{id=301, name='Fridge', rented=false}

Appliance{id=302, name='Oven', rented=false}

Appliance{id=303, name='Microwave', rented=false}

Appliance with ID 302 is rented.

Appliance with ID 301 removed successfully.

Updated Inventory:

Appliance{id=302, name='Oven', rented=true}

Appliance{id=303, name='Microwave', rented=false}

Total non-rented appliances: 1

Sample 2:

Input:

2

Air Conditioner

Washing Machine

999

305

Output:

Appliances in the Inventory:

Appliance{id=301, name='Air Conditioner', rented=false}

Appliance{id=302, name='Washing Machine', rented=false}

Appliance with ID 999 not found.

Appliance with ID 305 not found.

Updated Inventory:

Appliance{id=301, name='Air Conditioner', rented=false}

Appliance{id=302, name='Washing Machine', rented=false}

Total non-rented appliances: 2

Sample 3:

Input:

1

Television

301

301

Output:

Appliances in the Inventory:

Appliance{id=301, name='Television', rented=false}

Appliance with ID 301 is rented.

Appliance with ID 301 removed successfully.

Updated Inventory:

Total non-rented appliances: 0

QUESTION 2:

Antique Bookstore System

An antique bookstore tracks unique book items. Each book has a unique identifier (a serial number), a title, an author's name, and a decade of publication. The store needs to perform various operations on these book items using an ArrayList.

Operations Required:

- Add New Books: Add book items to the collection.
- Retrieve Book Items: Retrieve and display details of a book by its unique identifier.
- Check for Books by Decade: Check if there are any books from a specific decade in the collection.
- Sum of Decades for Book Count: Calculate the sum of the publication years (grouped by decade) for all books from a specific decade.

Attributes for Book Items:

- serialNumber (int): A unique identifier for the book item (e.g., 5001, 5002).
- title (String): The title of the book.
- author (String): The author of the book.
- decade (int): The decade of publication of the book (e.g., 1920, 1930).

Task:

Implement a BookstoreSystem class using ArrayList to manage the list of book items. Each book item should be represented by a Book class with the attributes described above. The BookstoreSystem class should include the following methods:

Add Book Item

- Method: public void addBookItem(int serialNumber, String title, String author, int decade)
- Description: Adds a new book item to the collection.

Retrieve Book Item

- Method: public void retrieveBookItem(int serialNumber)
- Description: Retrieves and prints the details of a book item by its unique identifier. If the item is not found, print a message indicating that the book with the given serial number does not exist.

Check for Books by Decade

- Method: public void checkForBooksByDecade(int decade)
- Description: Checks if there are any books from the specified decade in the collection. Print whether books from the decade are present or not.

Sum of Decades for Book Count

- Method: public int sumOfDecadesForBooks(int decade)
- Description: Calculates and returns the sum of the publication years (grouped by decade) of all items from the specified decade.

Input Format

1. First Line: An integer n representing the number of book items to add to the collection. The value of n must be between 1 and 5, inclusive.
2. Next n Sets of Lines: Each set of lines contains four lines:
    - First Line: An integer representing the serialNumber of the book item.
    - Second Line: A string representing the title of the book item.
    - Third Line: A string representing the author of the book item.
    - Fourth Line: An integer representing the decade of the book item.
3. Next Line: An integer representing the serialNumber of the book item to retrieve.
4. Next Line: An integer representing the decade to check for books.
5. Last Line: An integer representing the decade to calculate the sum of decades.

Output Format

Book Collection:

- Print "Book Collection:".
- For each book item in the collection, print the item details with each attribute on a new line in the format:

BookItem Serial Number: <serialNumber>

Title: <title>

Author: <author>

Decade: <decade>

Retrieve Book Item Status:

- If the item is found, print:

Book Item with Serial Number <serialNumber>:

BookItem Serial Number: <serialNumber>

Title: <title>

Author: <author>

- Decade: <decade>
  If the item is not found, print: Book Item with Serial Number <serialNumber> does not exist.

Check for Books by Decade:

- If there are books from the decade, print: Books from the decade <decade> are in stock.
- If there are no books from the decade, print: No books from the decade <decade> in stock.

Sum of Decades for Book Count:

- Print: Sum of Decades for books from the decade <decade>: <sum>

Sample input:

3

5001

The Great Gatsby

F. Scott Fitzgerald

1920

5002

To Kill a Mockingbird

Harper Lee

1960

5003

1984

George Orwell

1940

5001

1920

1960

Sample 1 Output

Book Collection:

BookItem Serial Number: 5001

Title: The Great Gatsby

Author: F. Scott Fitzgerald

Decade: 1920

BookItem Serial Number: 5002

Title: To Kill a Mockingbird

Author: Harper Lee

Decade: 1960

BookItem Serial Number: 5003

Title: 1984

Author: George Orwell

Decade: 1940

Book Item with Serial Number 5001:

BookItem Serial Number: 5001

Title: The Great Gatsby

Author: F. Scott Fitzgerald

Decade: 1920

Books from the decade 1920 are in stock.

Sum of Decades for books from the decade 1960: 1960

Sample 2 Input

2

6001

Pride and Prejudice

Jane Austen

1810

6002

Moby Dick

Herman Melville

1850

9999

1830

1820

Sample output 2:

Book Collection:

BookItem Serial Number: 6001

Title: Pride and Prejudice

Author: Jane Austen

Decade: 1810

BookItem Serial Number: 6002

Title: Moby Dick

Author: Herman Melville

Decade: 1850

Book Item with Serial Number 9999 does not exist.

No books from the decade 1830 in stock.

Sum of Decades for books from the decade 1820: 0

QUESTIONS 3

Help Desk Ticket Management System Using ArrayList

You are developing a software application for a Help Desk to manage and track user support tickets. The application allows users to input multiple help tickets and automatically sorts them based on the total time spent on each ticket. Each ticket consists of a Ticket ID, a problem description, and the number of hours spent resolving the issue, which must be entered on separate lines.

The application must validate the input to ensure that:

- The Ticket ID is a positive integer between 1 and 10 and must be unique across all tickets.
- The number of hours spent is an integer between 1 and 100.

If the input does not meet the expected format or contains invalid data, appropriate error messages should be displayed.

After collecting all ticket details, the program should sort the tickets based on the number of hours spent (in descending order) and also calculate the average number of hours spent across all tickets.

Input Format

Input all ticket records in a single batch, with each ticket's details on separate lines:

- **First Line**: Ticket ID (unique integer between 1 and 10)
- **Second Line**: Problem Description (string)
- **Third Line**: Number of Hours Spent (integer between 1 and 100)

Enter "stop" to signal that the input is complete.

Output Format

Each ticket should be displayed in the following format:

Ticket ID: [ticket ID]

Problem Description: [problem description]

Hours Spent: [hours spent]

Each ticket is displayed on a separate line, with a blank line between tickets.

At the end, display the average number of hours spent across all tickets in the following format:

Average Hours Spent: [average number of hours spent]

where [average number of hours spent] is a double formatted to two decimal places.

Error Handling:

- **Number Format Exception**:
    - Error Message for Non-Integer Input: "Please enter a valid integer."
- **Number of Hours Spent**:
    - Error Message for Out-of-Range Values: "Hours spent must be between 1 and 100."
- **ID Validation**:
    - Error Message for Invalid ID: "Ticket ID must be a positive integer between 1 and 10 and must be unique."

Sample 1 Input:

1

Software Installation

10

2

Password Reset

5

3

Network Issue

20

4

System Crash

18

5

Backup Restore

15

Stop

Sample 1 Output:

Ticket ID: 3

Problem Description: Network Issue

Hours Spent: 20

Ticket ID: 4

Problem Description: System Crash

Hours Spent: 18

Ticket ID: 5

Problem Description: Backup Restore

Hours Spent: 15

Ticket ID: 1

Problem Description: Software Installation

Hours Spent: 10

Ticket ID: 2

Problem Description: Password Reset

Hours Spent: 5

Average Hours Spent: 13.60

Sample 2 Input:

one

Hard Drive Failure

12

2

Printer Error

7

3

System Downtime

30

10

Data Corruption

40

12

Connection Error

25

Stop

Sample 2 Output:

Please enter a valid integer.

Ticket ID must be a positive integer between 1 and 10 and must be unique.

Sample 3 Input:

1

Account Setup

15

2

Email Setup

10

3

Software Crash

30

5

Printer Jam

60

10

Data Loss Recovery

105

Stop

Sample 3 Output:

Hours spent must be between 1 and 100.

QUESTION 4:

Caregiver Shift Management System Using HashMap

You are developing a system to manage caregiver duty schedules at an assisted living facility. The system should allow users to input the names of caregivers and the number of days they have worked during a month. This information should be stored in a HashMap. The system must perform the following operations to manage and query caregiver duty records:

Operations to be performed:

1. **Add Caregiver Records**: Input the number of caregivers and their respective names along with the number of days worked.
2. **Retrieve Caregiver's Working Days**: Look up how many days a specific caregiver has worked by their name.
3. **Remove Caregiver Records**: Remove a caregiver's record from the schedule.
4. **Check if a Caregiver Exists**: Verify if a specific caregiver is in the system based on their name.
5. **Count Total Days Worked**: Calculate and display the total number of days worked by all caregivers.

Input Format:

1. **Adding Caregivers to the System**:
    - The first line contains an integer n representing the number of caregivers on duty (between 1 and 12).

- For each caregiver, input two lines:
  - The first line contains the caregiver's name (String).
  - The second line contains the number of days worked (an integer between 1 and 30).

2. **Perform Operations**:
   - A line containing the name of a caregiver to look up how many days they worked.
   - A line containing the name of a caregiver to remove from the system.
   - A line containing the name of a caregiver to check if they exist in the system.

Output Format:

1. **For retrieving the number of days a caregiver worked**:
   - If the caregiver is found, print: [Caregiver name] worked for [days worked] days.
   - If the caregiver is not found, print: Caregiver [Caregiver name] not found.
2. **For removing a caregiver**:
   - If the caregiver is found and removed, print: Caregiver [Caregiver name] has been removed from the schedule.
   - If the caregiver is not found, print: Caregiver [Caregiver name] not found for removal.
3. **For checking if a caregiver exists**:
   - If the caregiver exists, print: Caregiver [Caregiver name] exists in the system.
   - If the caregiver does not exist, print: Caregiver [Caregiver name] does not exist in the system.
4. **For counting total days worked**:
   - Print the total number of days worked by all caregivers; Total Days Worked: [total days worked].

Error Handling:

Ensure that the number of days worked is between 1 and 30. If invalid, output: Days worked must be between 1 and 30.

Sample 1 Input:

3

Laura

12

John

20

Megan

10

John

Megan

Laura

Sample 1 Output:

John worked for 20 days.

Caregiver Megan has been removed from the schedule.

Caregiver Laura exists in the system.

Total Days Worked: 32

Sample 2 Input:

2

Elena

25

Noah

18

Sophia

Liam

Isabella

Sample 2 Output:

Caregiver Sophia not found.

Caregiver Liam not found for removal.

Caregiver Isabella does not exist in the system.

Total Days Worked: 43

Sample 3 Input:

3

Michael

10

Emma

50

Olivia

7

Lucas

Olivia

Lucas

Sample 3 Output:

Days worked must be between 1 and 30.

Caregiver Olivia has been removed from the schedule.

Caregiver Lucas not found.

Total Days Worked: 10

QUESTION 5:

Farmland Harvest Tracking Using HashMap

You are developing a system to manage and track harvest yields for different plots on a farm. The system should allow users to input plot names and their respective harvest yields (in tons) and store this information using a HashMap. The system should also allow users to query and manage the harvest yield records by performing the following operations:

Operations to be Performed:

1. **Add Plot Harvest Records**:
    - Input plot names and their harvest yields (in tons) and store them in the system.
2. **Retrieve Harvest Yield by Plot**:
    - Look up and display the harvest yield for a specific plot using its name.
3. **Check if a Certain Harvest Yield Exists**:
    - Verify if any plot has a harvest yield of a specified value (in tons).
4. **Check the Total Number of Plots**:
    - Calculate and display the total number of plots stored in the system in ascending order of harvest yields (in tons).
5. **Display All Plot Names and Yields**:
    - Output all the plots with their corresponding harvest yields.

Input Format:

1. **Adding Plot Harvest Records**:
    - The first line contains an integer n representing the number of plots to be recorded (between 1 and 20).
    - For each plot, input two lines:
        - The first line contains the plot name (String).
        - The second line contains the harvest yield in tons (a positive number).
2. **Performing Operations**:
    - A line containing the name of a plot to retrieve its harvest yield.
    - A line containing a harvest yield value (in tons) to check if any plot has this yield.

Output Format:

1. **For retrieving harvest yield by plot**:
    - If the plot is found, print: [Plot name] has a harvest yield of [yield] tons.
    - If the plot is not found, print: Plot [Plot name] not found.
2. **For checking if a certain harvest yield exists**:

- If the harvest yield exists, print: A plot with a harvest yield of [yield] tons exists.
- If the harvest yield does not exist, print: No plot has a harvest yield of [yield] tons.

3. **For getting the total number of plots**:
    - Print the total number of plots and display them in the following format: Total number of plots: [number of plots].
4. **For displaying all plot names and yields**:
    - Output each plot name and its corresponding harvest yield in ascending order as follows: Plot: [Plot name], Yield: [yield] tons

Error Handling:

- If the harvest yield is a negative value or zero, output: "Harvest yield must be a positive number." and stop the process.

Refer to the sample output for formatting specifications.

Sample 1 Input:

2

NorthPlot

350

SouthPlot

400

NorthPlot

400

Sample 1 Output:

NorthPlot has a harvest yield of 350 tons.

A plot with a harvest yield of 400 tons exists.

Total number of plots: 2

Plot: NorthPlot, Yield: 350 tons

Plot: SouthPlot, Yield: 400 tons

Sample 2 Input:

2

PlotX

120

PlotY

180

PlotZ

250

Sample 2 Output:

Plot PlotZ not found.

No plot has a harvest yield of 250 tons.

Total number of plots: 2

Plot: PlotX, Yield: 120 tons

Plot: PlotY, Yield: 180 tons

QUESTION 6

Environmental Sustainability Program Management System

You are developing an Environmental Sustainability Program Management System for an organization focused on promoting sustainability initiatives. The system should manage different sustainability programs for various regions. Users can add new programs, search for a specific program, display all current programs, and get the total number of active programs. After all operations, the system should also identify and display the program with the longest name.

Input Format:

1. The first line contains an integer n, representing the number of sustainability programs.
2. The next 2 * n lines provide the details of each program:
    - A line containing the program ID (String).
    - A line containing the program name (String).
3. The last line of input contains a String representing the program ID to search for.

Output Format:

Program Search Result:

- If the program ID is found, print: Program with ID [program ID] is named: [program name]
- If the program ID is not found, print: No program found with ID: [program ID]

Total Number of Programs:

- Print: Total number of programs: [number]

Display All Programs:

- Display all programs with the heading: Sustainability programs: followed by each program in the format:
  Program ID: [program ID], Name: [program name]

Program with the Longest Name:

- Print: Program with the longest name is: [program name]

Sample 1 Input:

3

S101

Green Energy Initiative

S202

Zero Waste Campaign

S303

Clean Oceans Movement

S202

Sample 1 Output:

Program with ID S202 is named: Zero Waste Campaign

Total number of programs: 3

Sustainability programs:

Program ID: S202, Name: Zero Waste Campaign

Program ID: S303, Name: Clean Oceans Movement

Program ID: S101, Name: Green Energy Initiative

Program with the longest name is: Clean Oceans Movement

Sample 2 Input:

3

S101

Protect Our Wildlife

S202

Plastic-Free Future

S303

Global Climate Action Plan

S404

Sample 2 Output:

No program found with ID: S404

Total number of programs: 3

Sustainability programs:

Program ID: S202, Name: Plastic-Free Future

Program ID: S303, Name: Global Climate Action Plan

Program ID: S101, Name: Protect Our Wildlife

Program with the longest name is: Global Climate Action Plan

QUESTION 7:

Factory Machine Management System

You are developing a **Factory Machine Management System** to manage unique machine IDs in a factory. The system uses a **'HashSet'** to store machine IDs and ensure that all IDs are unique. Your task is to implement the following operations:

Operations to be Performed:

1. **Add Machine IDs:**
   Input a number of unique machine IDs and store them in the HashSet.
2. **Check for a Machine ID:**
   After storing the IDs, check if a specific machine ID exists in the HashSet and report whether it is present or not.

3. **Delete a Machine ID:**
   If the machine ID exists, it should be removed from the HashSet. If the ID is not found, indicate that it is not present.
4. **Display the Updated List of Machine IDs:**
   After attempting the deletion, display the updated list of machine IDs.
5. **Count Odd Machine IDs:**
   Count how many machine IDs are odd numbers and display the count.

Input Format:

- The first line contains an integer n, representing the number of machine IDs.
- The next n lines each contain a unique integer representing a machine ID.
- The following line contains an integer representing the machine ID to be checked.
- The final line contains an integer representing the machine ID to be removed.

Output Format:

Check Machine ID:

- Print: "The machine ID [ID] is present in the HashSet." if the ID is found.
- Print: "The machine ID [ID] is not present in the HashSet." if the ID is not found.

Delete Machine ID:

- Print: "The machine ID [ID] was removed from the HashSet." if the ID was found and removed.
- Print: "The machine ID [ID] was not found in the HashSet." if the ID was not found.

Display Updated List:

- Print: "Updated list of machine IDs:" followed by each ID in the HashSet.

Count Odd Machine IDs:

- Print: "Number of odd machine IDs: [count]" where [count] is the number of odd machine IDs.

Sample 1 Input:

4

150

301

502

703

301

502

Sample 1 Output:

The machine ID 301 is present in the HashSet.

The machine ID 502 was removed from the HashSet.

Updated list of machine IDs:

Machine ID: 150

Machine ID: 703

Machine ID: 301

Number of odd machine IDs: 2

Sample 2 Input:

3

22

44

66

30

100

Sample 2 Output:

The machine ID 30 is not present in the HashSet.

The machine ID 100 was not found in the HashSet.

Updated list of machine IDs:

Machine ID: 22

Machine ID: 44

Machine ID: 66

Number of odd machine IDs: 0

QUESTION 8:

Advertising Campaign Budget Management System

You are developing an **Advertising Campaign Budget Management System** to manage unique advertising campaigns using their IDs and corresponding budgets. The system uses a **'HashSet'** to track the campaigns. Each campaign has a unique ID and an associated budget.

Operations to be Performed:

1. **Add Campaigns:**
   Input a number of unique campaign IDs and their budgets, and store them in a HashSet.
2. **Check for a Campaign:**
   After storing the campaigns, check if a specific campaign ID exists in the HashSet and report whether it is present or not.
3. **Remove a Campaign:**
   If the campaign ID exists, it should be removed from the HashSet. If it is not found, indicate that the ID is not present.
4. **Display the Updated List of Campaigns:**
   After attempting the deletion, display the updated list of campaign IDs and their associated budgets.
5. **Calculate Average Budget:**
   Calculate and display the average budget of all campaigns.

Input Format:

- The first line contains an integer n, representing the number of campaigns.
- The next 2 * n lines contain alternating entries of a campaign ID (string) and its budget (integer).
- The following line contains a string representing the campaign ID to be checked.
- The final line contains a string representing the campaign ID to be removed.

Output Format:

Check Campaign ID:

- Print: "The campaign ID [ID] is present in the HashSet." if the ID is found.
- Print: "The campaign ID [ID] is not present in the HashSet." if the ID is not found.

Remove Campaign ID:

- Print: "The campaign ID [ID] was removed from the HashSet." if the ID was found and removed.
- Print: "The campaign ID [ID] was not found in the HashSet." if the ID was not found.

Display Updated List:

- Print: "Updated list of campaigns:" followed by each ID and its associated budget.

Calculate Average Budget:

- Print: "Average Budget of All Campaigns: [average]" where [average] is the average budget of all campaigns rounded to two decimal places.

Sample 1 Input:

4

AdCamp1

5000

AdCamp2

7000

AdCamp3

4500

AdCamp4

6000

AdCamp3

AdCamp2

Sample 1 Output:

The campaign ID AdCamp3 is present in the HashSet.

The campaign ID AdCamp2 was removed from the HashSet.

Updated list of campaigns:

Campaign ID: AdCamp1, Budget: 5000

Campaign ID: AdCamp3, Budget: 4500

Campaign ID: AdCamp4, Budget: 6000

Average Budget of All Campaigns: 5166.67

Sample 2 Input:

3

Promo1

1000

Promo2

2000

Promo3

3000

Promo4

Promo5

Sample 2 Output:

The campaign ID Promo4 is not present in the HashSet.

The campaign ID Promo5 was not found in the HashSet.

Updated list of campaigns:

Campaign ID: Promo1, Budget: 1000

Campaign ID: Promo2, Budget: 2000

Campaign ID: Promo3, Budget: 3000

Average Budget of All Campaigns: 2000.00