**C PROGRAMS :**

**PROBLEM 1 :**

One day, Bunny and his friends were playing hide and seek in the forest. Tweety went along to hide. She finds an abandoned bag containing a board(8x8) game in a cave. She gets excited and starts playing the game only to realizes that it's magical. She has three chances to roll the dice. Each turn's outcome will lead into a new surrounding and some would be dangerous. If the outcome of rolling dices is in multiples of 3 then she will be caught in a dangerous phase. Write a C program to find whether Tweety finishes the game and returns safely or not.

If the outcome of rolling dices is either 0 or less than that you should tell tweety as **"Invalid Turn"**

**Sample Input and Output 1:**

Enter value of turn 1

**10**

Enter value of turn 2

**5**

Enter value of turn 3

**2**

Tweety is safe

**Sample Input and Output 2:**

Enter value of turn 1

**3**

Enter value of turn 2

**9**

Enter value of turn 3

**12**

Tweety is in danger

**Sample Input and Output 3:**

Enter value of turn 1

**0**

Enter value of turn 2

**9**

Enter value of turn 3

**12**

Invalid Turn

**PROBLEM 2 :**

Suppose you are given a matrix that represents the grades of students in a class. Each row in the matrix represents a student, and each column represents a subject. Write a C program to find the highest grade in given row and the highest grade in given column.

**Input Format:**

The input consists of (m\*n+2) integers. The first integer corresponds to m, the number of rows in the matrix and the second integer corresponds to n, the number of columns in the matrix. The remaining integers correspond to the elements in the matrix. The elements are read in row wise order, first row first, then second row and so on. Assume that the maximum value of m and n is 10. If number of rows or columns is less than or equal to zero display message as **"Invalid Input"**.

**Output Format:**

Refer sample output for details.

**Sample Input and Output 1:**

Enter number of rows

**3**

Enter number of columns

**2**

Enter matrix values

**4 5**

**6 9**

**0 3**

Enter a row number

**2**

Maximum value in row 2 is 9

Enter a column number

**1**

Maximum value in column 1 is 6

**Sample Input and Output 2:**

Enter number of rows

**0**

Invalid Input

**Sample Input and Output 3:**

Enter number of rows

**3**

Enter number of columns

**-1**

Invalid Input

**Sample Input and Output 4:**

Enter number of rows

**11**

Invalid Input

**Sample Input and Output 5:**

Enter number of rows

**3**

Enter number of columns

**2**

Enter matrix values

**84 55**

**99 67**

**10 93**

Enter a row number

**2**

Maximum value in row 2 is 99

Enter a column number

**3**

Invalid Input

**Sample Input and Output 6:**

Enter number of rows

**3**

Enter number of columns

**2**

Enter matrix values

**84 55**

**99 67**

**10 93**

Enter a row number

**0**

Invalid Input

**JAVA :**

**PROBLEM 1 :**

Arun's teacher gives him an assignment in strings. Write a Java program to divide the 26 letters of the alphabet into 13 pairs and find out the pair of each alphabet. Help Arun complete his assignment using a Java program.

**Constraint:**

* The first half will cover letters a to m, while the second half will cover letters z to n. So, for 'a', the pair is 'z', for 'b', it is 'y', and so on.
* Assume that characters are in lowercase.
* The input string cannot contain any digits or special characters; otherwise, print the message **"<input> is an** **invalid input"** and terminate the program.

**Note:**

* Do not edit the existing code template.
* 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.
* Implement the business requirements within the main method. Please do not change the class name.
* **Please do not use System.exit(0); to terminate the program.**

**Sample input 1**

Enter the string

**zebra**

**Sample output 1**

avyiz

**Sample input 2**

Enter the string

**5rar!**

**Sample output 2**

5rar! is an invalid input

**PROBLEM 2:**

**SERENITY BAY :**

The Serenity Bay Wellness Retreat offers seasonal yoga classes as part of its wellness packages. To automate the calculation of total charges based on the session type and the number of participants, the retreat is hiring a software developer to build this system.

As the developer, your task is to write a Java program to calculate the yoga session cost.

| Req.# | Requirement Description | Type (Class) | Attributes | Method Name | Description |
| --- | --- | --- | --- | --- | --- |
| 1 | Include a four-argument constructor | YogaSessionBooking | String bookingId, Date sessionDate, String sessionType, int numberOfParticipant |  | Include a public 4-argument constructor in the order: bookingId, sessionDate, sessionType, numberOfParticipant.  The getters and setters method for all the attributes is provided as a part of the code skeleton. |
| 2 | Calculate yoga session cost | YogaSessionBooking |  | calculateYogaSessionCost() | This method calculates total yoga session cost based on session type and number of participants.   |  |  | | --- | --- | | sessionType | Cost (per person) | | basic | 500 | | intermediate | 1500 | | advanced | 4500 |     Constraints:   * The session type must be "basic", "intermediate", or "" (case-insensitive). Otherwise, return 0. * The number of participants must be between 1 and 10 (inclusive). Otherwise, return 0. * The method calculateYogaSessionCost() must return a double. |

You are provided with the main method in the UserInterface class as code template, and it is excluded from evaluation.

Note:

* Edit only the YogaSessionBooking class to implement the four-argument constructor and the business requirements.
* The methods and the constructor should be public, and the attributes of the class should be private.
* 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 that the names for classes, attributes, and methods are provided as specified in the question description.
* Please do not use System.exit(0); to terminate the program.

Sample Input And Output 1:

Enter the booking id:

YGA1010

Enter the session date:

15-08-2024

Enter the session type:

advanced

Enter the number of participant:

2

Booking Id:YGA1010

Session Date:15-08-2024

Session Type:advanced

Number Of Participant:2

Total Yoga Cost: 9000.0

Sample Input And Output 2:

Enter the booking id:

YGA8945

Enter the session date:

14-07-2024

Enter the session type:

expert

Enter the number of participant:

12

Invalid Booking details

**PROBLEM 3 :**

**VIRTUAL FITNESS TRAINER :**

The Flex Core Gym is a fitness and wellness center. The management has decided to compute a virtual personalized fitness score for each session based on the exercise type, repetitions, and duration. As a programmer, help the trainer evaluate these sessions effectively.

**Functional Requirements:**

| **Req. #** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Responsibilities** |
| --- | --- | --- | --- | --- | --- |
| **1.** | Extract the details of the workout session and create an object for the WorkoutSession class. | **WorkoutUtility** | **extractSessionDetails** | String sessionDetails | This method accepts workout session details as a colon-separated string in the format (sessionId:exerciseType:reps:duration). It parses the values, validates them, and returns a populated WorkoutSession object. |
| **2.** | Calculate the fitness score of the workout. | **WorkoutSession** | **calculateFitnessScore** |  | This method calculates and returns the fitness score of a workout session. The fitness score is calculated by multiplying **the number of repetitions (reps) by the duration of the workout in minutes (duration) and by an intensity factor** that depends on the type of exercise performed.  The intensityFactor is determined based on the exerciseType of the session, using the following mapping:   | **Exercise Type(case-sensitive)** | **Intensity Factor** | | --- | --- | | pushup | 1.2 | | squat | 1.0 | | plank | 1.5 | | burpee | 1.8 | | Unkown excercise type | 0.5 |   ***Constraints:***   * The calculated fitness score should be returned as double. * If the **duration**is less than 10, return -1 and terminate. |

**You are provided with the main method in the UserInterface class as code template, and it is excluded from evaluation.**

**Note:**

* Edit only the **WorkoutSession**and **WorkoutUtility** classes to implement the business requirements.
* The methods and the constructor should be public, and the attributes of the class should be private.
* 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 that the names for classes, attributes, and methods are provided as specified in the question description.
* **Please do not use System.exit(0); to terminate the program.**

**Input Format:  <**sessionId>:**<**exerciseType>:**<**reps>:**<**duration>

**Sample Input / Output 1**

Enter workout details:

**S1234:plank:30:20**

Workout Summary:

Session Id : S1234

Exercise Type : plank

Reps : 30

Duration (min) : 20

Fitness Score : 900.00

**Sample Input / Output 2**

Enter workout details:

**S1245:Aerobic:30:8**

Invalid session details

**PROBLEM 4 :**

**TIME CODE LOCK :**

You're helping design a futuristic digital lock system used in secure facilities. This lock accepts only 4 digits and uses them to generate a valid 24-hour format time (HH:MM). It must use all four digits exactly once, and it always unlocks only at the greatest possible time that can be formed from them.

Your task is to write a program that takes:

* 4 digits as input from the user (one at a time),
* Calculate the greatest valid 24-hour time (00:00 to 23:59) that can be formed by rearranging these digits.

**Requirement**:

* Prompt the user to enter exactly four single-digit integers (from 0 to 9), one at a time.
* Generate all possible combinations of the four digits to explore every arrangement.
* For each combination, form a time in HH:MM format using the first two digits as the hour and the last two as the minute.
* Check whether the time is valid (i.e., hour is between 00-23 and minute is between 00-59).
* Among all valid times, identify the greatest valid time that can be formed.
* If at least one valid time exists, display it in the format: **Greatest valid time: HH:MM**(Ensure the output includes leading zeros; for example, display '01:30' instead of '1:30' for the greatest valid time.)
* If no valid time can be formed, display: **No valid 24-hour time can be formed**

**Note**:

* Assume that all single digits should be a **positive integer.**
* 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.
* **Do not use System.exit(0) to terminate the code.**

**Sample input and output 1:**

Enter the 4 digits:

**2 3 4 5**

Greatest valid time: 23:54

**Explanation**:

The program tries all possible ways to arrange the digits 2, 3, 4, 5 into a valid 24-hour time in HH:MM format.

Some valid combinations include:

23:45(valid)

23:54(valid)

24:35(invalid)

25:34(invalid)

Among all valid times, 23:54 is the Greatest. So, the program prints: Greatest valid time: 23:54

**Sample input and output 2:**

Enter the 4 digits:

4 4 4 4

No valid 24-hour time can be formed

**PROBLEM 5 :**

**BLUE BANK :**

Blue Bank wants to streamline its loan approval process by building a system that automatically checks if an applicant is eligible for a loan based on specific financial criteria. As a programmer, you are responsible for implementing the logic to assess eligibility and handle applicant information.

**Functional Requirements**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. #** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Responsibilities** |
| 1 | Extract the details of the applicant and create an object for the LoanApplicant class. | **LoanUtility** | **extractDetails** | String applicantDetails | This method accepts applicantDetails separated by a colon **:**as an argument, extracts the properties of the LoanApplicant, sets the values, and returns a LoanApplicant object. |
| 2 | Check the eligibility of the applicant to receive a loan. | **LoanApplicant** | **isEligible** |  | This method validates the eligibility of a loan application based on the applicant's age, income, credit score, and requested loan amount.  **Constraints:**   * Age must be between 21 and 60, inclusive. If this check fails, return ***"<applicantId> - Age not eligible"***. * Income must be greater than or equal to 20,000. If this check fails, return **"<applicantId> - Insufficient income".** * Credit score must be greater than or equal to 650. If this check fails, return **"<applicantId> - Low credit score".** * Loan amount must be less than or equal to 10 times the income. If this check fails, return **"<applicantId> - Loan amount too high".** * If all checks pass, return**"<applicantId> - Approved".** |

**You are provided with the main method in the UserInterface class as a code template, and it is excluded from evaluation.**

**Note:**

* Edit only the **LoanApplicant**and **LoanUtility**classes to implement the business requirements.
* The methods and the constructor should be public, and the attributes of the class should be private.
* 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 that the names for classes, attributes, and methods are provided as specified in the question description.
* Please do not use System.exit(0); to terminate the program.

***Input Format:****<applicantId>:<name>:<age>:<income>:<creditScore>:<loanAmount>*

**Sample Input / Output 1**

Enter the applicant details:

**L001:Alice:30:30000:700:200000**

====Applicant Details====

Applicant Id: L001

Name: Alice

Age: 30

Monthly Income: 30000.0

Credit Score: 700

Requested Loan Amount: 200000.0

Eligible for loan:  L001 - Approved

**Sample Input / Output 2**

Enter the applicant details:

**L002:Bob:19:15000:680:100000**

====Applicant Details====

Applicant Id: L002

Name: Bob

Age: 19

Monthly Income: 15000.0

Credit Score: 680

Requested Loan Amount: 100000.0

Eligible for loan: L002- Age not eligible

**PROBLEM 6 :**

**REVIEW HIGHLIGHTS :**

You are working on a backend system for a large e-commerce platform that collects user-submitted review feedback for each product. This feedback reflects common sentiments such as:

*"Too Expensive", "Great Value", "Durable", "Poor Quality", "Fast Delivery"*

Your task is to analyse an array of such feedback and identify those that represent a dominant perception — that is, feedback that appears more than [N / 3] times (using integer division).

**Constraints:**

* Feedback is case-insensitive.
* If N is 0 or negative, then display: ***Invalid input: N - Number of feedback entries must be greater than 0***
* If any feedback contains special characters display:***Invalid input: <original feedback array>***
* If no feedback entry occurs more than N/3 times, then display: ***No feedback appears more than N/3 times in: <original feedback array>***
* If dominant feedback is found, display it in a comma-separated.

**Note:**

* In the sample input/output provided, the highlighted text in bold corresponds to the user's input, and the rest of the text represents the output.
* Adhere to the code template, if provided.
* **Please do not use System.exit(0) to terminate the program.**

**Sample Input 1**

Enter the number of feedback entries:

**8**

Enter the feedback:  

**Durable**

**Durable**

**Great Value**

**Too Expensive**

**durable**

**Poor Quality**

**Great Value**

**Great value**

**Sample Output 1**

Durable, Great Value

**Explanation: N = 8**, so the threshold is **N / 3 = 2**. After converting to lowercase, "durable" appears **3 times**", great value" appears **3 times.**Both exceed the threshold of 2 → they are **dominant feedbacks**.

**Sample Input 2**

Enter the number of feedback entries:

**4**

Enter the feedback:

**Poor Quality**

**Durable**

**Too Expensive**

**Great Value**

**Sample Output 2**  
No feedback appears more than N/3 times in: Poor Quality, Durable, Too Expensive, Great Value

**Explanation:**N = 4, so the threshold is N / 3 = 1. Each of the four feedback phrases appears only once, which is not more than 1. Therefore, no entry is dominant.

**Sample Input 3**

Enter the number of feedback entries:

**0**

**Sample Output 3**

Invalid input: 0 - Number of feedback entries must be greater than 0

**Explanation:**Entry count is 0 → invalid.

**Sample Input 4**

Enter the number of feedback entries:

**5**

Enter the feedback:

**@#$%**

**!!!@@**

**???**

**---**

**\*\*&^**

**@@**

**Sample Output 4**

Invalid input: @#$%, !!!@@, ???, ---, \*\*&^, @@

**Explanation**: All feedbacks contain **only special characters**, **no letters**.

**PROBLEM 7 :**

**RED READY :**

Red Ready wants to create a system to identify whether a person is eligible to donate blood based on certain medical criteria. As a programmer, you are tasked with building a program that checks eligibility and stores donor information.

**Functional Requirements**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. #** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Responsibilities** |
| 1 | Extract the details of the Donor and create an object for the Donor class. | **DonorUtility** | **extractDetails** | String donorDetails | This method accepts donorDetails separated by a colon **:** as an argument, extracts the properties of the Donor, sets the values, and returns a Donor object. |
| 2 | Check the eligibility of the donor to donate blood. | **Donor** | **checkEligibility** | double haemoglobinLevel | This method validates the eligibility of a blood donor based on the donor's age, gender, weight, blood group, and haemoglobin level.  **Constraints:**   * Age must be between 18 and 65, inclusive. If this check fails, return "***<donorID> - Age not eligible"***. * Gender must be 'f'/'F', 'm'/'M', or 't'/'T'. If this check fails, return***"<donorID> - Invalid gender".*** * Weight must be greater than or equal to 50 kg. If this check fails, return ***"<donorID> - Weight too low"***. * Blood group must be one of the following (case-sensitive): A+, A-, B+, B-, O+, O-, AB+, AB-. If this check fails, return***"<donorID> - Invalid blood group".*** * Haemoglobin level must be greater than or equal to 12.5 g/dL. If this check fails, return ***"<donorID> - Haemoglobin too low".***   If all checks pass, return ***"<donorID> - Eligible to donate blood".*** |

**You are provided with the main method in the UserInterface class as a code template, and it is excluded from evaluation.**

**Note:**

* Edit only the **Donor**and **DonorUtility**classes to implement the business requirements.
* The methods and the constructor should be public, and the attributes of the class should be private.
* 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 that the names for classes, attributes, and methods are provided as specified in the question description.
* Please do not use System.exit(0); to terminate the program.

***Input Format:****<donorId>:<name>:<age>:<gender>:<weight>:<bloodGroup>*

**Sample Input / Output 1**

Enter the donor details:

**D003:Jane:28:T:53:O+**

Enter haemoglobin level:

**13.2**

====Donor Details====

Donor Id: D003

Name: Jane

Age: 28

Gender: T

Weight: 53.0

Blood Group: O+

Haemoglobin Level: 13.2

D003 - Eligible to donate blood

**Sample Input / Output 2**

Enter the donor details:

**D005:Trump:35:M:83:L+**

Enter the haemoglobin level

**12.8**

====Donor Details====

Donor Id: D005

Name: Trump

Age: 35

Gender: M

Weight: 83.0

Blood Group: L+

Haemoglobin Level: 12.8

D005 - Invalid blood group

**PROBLEM 8 :**

**PIXEL WRITER :**

A digital text-rendering system writes strings character by character, where each character consumes a specific number of pixels. Each line has a fixed pixel width limit. Your task is to write a Java program that accepts a lowercase word and a corresponding array of pixel widths (for each character in the word). The program should calculate and display how many lines the word wraps across and the total pixel width of the last line.

**Requirements**:

1. Prompt the user to enter a string containing only lowercase English letters.
2. Prompt the user to enter the pixel widths (positive integers) for each character, space-separated.
3. **Validate the inputs as follows**:
   * If the string contains characters other than lowercase English letters, print: **"Invalid string:  <Input String> - Only lowercase English letters are allowed."**
   * If the number of pixel widths does not match the number of characters in the string, print: **"Invalid input: You entered <pixel Widths length> widths for a <word length> -character string."**
4. Using a maximum line width of **50 pixels**, determine:
   * How many lines are required to render the string.
   * The total pixel width of the last line.
5. Print the result in the format: **"Output: [<number of lines>, <last line width>]"**

**Example**:

**Input**:  **String**=helloworld    **Pixel Widths**= [5 5 10 10 10 5 10 5 5 5]

**Output**: [2, 25]

**Explanation:**

* Characters: h(5), e(5), l(10), l(10), o(10), w(5), o(10), r(5), l(5), d(5)
* Line 1: h(5) + e(5) + l(10) + l(10) + o(10) = 40
* Add w(5) = 45 → still fits
* Add o(10) = 55 → exceeds 50 → move to line 2
* Line 2: o(10), r(5), l(5), d(5) = 25

**Note**:

* Assume that all pixel width should be a **positive integer.**
* 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.
* **Do not use System.exit(0) to terminate the code.**

**Sample input/output 1:**

Enter the string:

**computer**

Enter the pixel widths:

**10 5 6 14 7 8 9 15**

Output: [2, 24]

**Sample input/output 2:**

Enter the string:

**Orange**

Invalid string: Orange - Only lowercase English letters are allowed.

**Sample input/output 3:**

Enter the string:

**mobile**

Enter the pixel widths:

**1 3 6 8**

Invalid input: You entered 4 widths for a 6-character string.

**PROBLEM 9 :**

**CHARACTER FREQUENCY :**

Arya is playing a word game with her friend Mira. She gives Mira two words and asks her to find out which characters from the first word also appear in the second word. But there's a twist, she should only do this if both words are of equal length. For every matching character, Mira must count how many times it appears in the both the words and print the character along with its frequency.

Constraints:

* If the first word and second word lengths are not equal, then display "<word1> and <word2> have different word lengths".
* If the first word and second word does not have any common character then display, "<word1> and <word2> have no matching character frequencies"
* Both the words are case-sensitive.

Note:

* In the sample input/output provided, the highlighted text in bold corresponds to the user's input, and the rest of the text represents the output.
* Adhere to the code template, if provided.
* Please do not use System.exit(0) to terminate the program.

Sample Input and Output 1:

Enter first word:

apple

Enter second word:

happy

Matching characters and their frequency:

a - 1

p - 2

Sample Input and Output 2:

Enter first word:

apple

Enter second word:

happen

apple and happen have different word lengths

**PROBLEM 10 :**

**SMART PAYMENT PROCESSOR :**

A digital wallet service wants to automate transaction fee calculations based on the user's payment method-Credit Card or UPI. Design a system that validates inputs and computes the total amount deducted, applying dynamic fee rates based on card types or bank names.

**Functional Requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type (Class)** | **Attribute** | **Methods** | **Responsibilities** |
| **PaymentProcessor** | String transactionId  String payerName  double amount  String currency | The getters, and setters methods for all the attributes, are provided as a part of the code skeleton. | Include a parameterized constructor of four arguments in the following PaymentProcessor class -*transactionId, payerName, amount and currency.*  Include the abstract method in the **PaymentProcessor** class:  *public abstract double calculateTotalAmountDeducted()* |
| **CreditCardProcessor** | String cardNumber  String cardType | The getters, and setters methods for all the attributes, are provided as a part of the code skeleton. | Include a public 6 argument constructor in the order-*transactionId, payerName, amount, currency,cardNumber and cardType.* |
| **CreditCardProcessor** |  | **calculateTotalAmountDeducted** | The method calculates the total amount deducted for a credit card transaction by applying a fee percentage based on the type of card used.  **Constraints:**   * The fee percentage is applied on the transaction amount based on the card type:  |  |  | | --- | --- | | **Card Type** | **Fee %** | | VISA | 2% | | MASTERCARD | 1.80% | | AMEX | 2.50% | | Other card types | 2.20% |  * If the amount is **less than or equal to 0**, or if the cardNumber length is not **exactly 16 digits**, return -1 . * The method returns the final amount as the sum of the original amount and the calculated fee. * The result should be returned as a **double**. |
| **UPIProcessor** | String upiId  String bankName | The getters, and setters methods for all the attributes, also no argument constructor are provided as a part of the code skeleton. | Include a public 6 argument constructor in the order-*transactionId, payerName, amount, currency,upiId and bankName.* |
| **UPIProcessor** |  | **calculateTotalAmountDeducted** | The method calculates the total amount deducted for a UPI-based transaction by applying a dynamic fee rate based on the bank name associated with the UPI ID.  **Constraints:**   * The fee percentage is determined by the bank name as follows:  |  |  | | --- | --- | | **Bank Name** | **Fee %** | | SBI | 1% | | HDFC | 1.2% | | ICICI | 1.1% | | Other banks | 1.5% |  * If the amount is **less than or equal to** 0, or if the upiId does not contain "**@**", return -1 . * The total amount deducted is calculated by adding the fee to the original amount. * The result should be returned as a **double**. |

**You are provided with the main method in the UserInterface class as a code template, and it is excluded from evaluation.**

**Note:**

* The platform namecan be either "**creditcard**" or "**upi**" (case-insensitive).
* The **CreditCardProcessor**and **UPIProcessor**class implements abstract methods of the **PaymentProcessor**class.
* Edit only the **PaymentProcessor**, **CreditCardProcessor**and **UPIProcessor**classes to implement the business requirements.
* Assume that all the input values are valid.
* The methods and the constructor should be public, and the attributes of the class should be private.
* 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 that the names for classes, attributes, and methods are provided as specified in the question description.
* **Please do not use System.exit(0); to terminate the program.**

**Sample Input/Output 1:**

Enter Transaction ID:

**TXN3001**

Enter Payer Name:

**Neha**

Enter Amount:

**20000**

Enter Currency:

**INR**

Enter platform (creditcard / upi):

**creditcard**

Enter Card Number:

**1111222233334444**

Enter Card Type (VISA / MasterCard / AMEX):

**VISA**

=== Credit Card Payment ===

Transaction ID: TXN3001

Payer Name: Neha

Card Type: VISA

Card Number: 1111222233334444

Amount: 20000.0

Total amount Deducted including transaction fee: 20400.0

**Sample Input/Output 2:**

Enter Transaction ID:

**TXN2332**

Enter Payer Name:

**Rohan**

Enter Amount:

**15000**

Enter Currency:

**INR**

Enter platform (creditcard / upi):

**upi**

Enter UPI ID:

**rohan-hdfc**

Enter Bank Name:

**HDFC**

UPI Payment Failed: Invalid UPI details

**Sample Input/Output 3:**

Enter Transaction ID:

**TXN1004**

Enter Payer Name:

**Krish**

Enter Amount:

**12000**

Enter Currency:

**INR**

Enter platform (creditcard / upi):

**Paytm**

Invalid platform name.