**PROBLEM 1 :**

**UNIVERSAL TECHNICIANS :**

Universal Technicians is one of the famous software trainers' institutes. They wanted to calculate the salary and rating of the employees. Help them as a software developer to do this task.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type (Class)** | **Attribute** | **Methods** | **Responsibilities** |
| **Employee** | String employeeId  String employeeName  String employeeType  double creditScore | The getters and setters method for all the attributes are provided as a part of the code skeleton. | Include a parameterized constructor of four arguments in the following order- *employeeId, employeeName, employeeType, creditScore*   Declare the abstract methods in this class.  abstract public double findEmployeeSalary() |
| **PermanentEmployee** | int experience | The getter and setter method for the attribute are provided in the code skeleton. | Include a public 5 argument constructor in the order- *employeeId, employeeName, employeeType, creditScore, and experience.* |
| **PermanentEmployee** |  | **findEmployeeSalary** | The method finds the employee's salary based on experience.   |  |  | | --- | --- | | **experience**  **(inclusive)** | **salary** | | 1-5 | 30000 | | 6-10 | 60000 | | 11-15 | 120000 | | 16-20 | 240000 | | Above 20 | 500000 |     **Constraints:**   * When the experience doesn't match any of the mentioned above, the method should return -1. * The found salary is returned as double. |
| **TemporaryEmployee** | int dailyHrsOfWork | The getter and setter method for the attribute are provided in the code skeleton. | Include a public 5 argument constructor in the order- *employeeId, employeeName, employeeType, creditScore, and dailyHrsOfWork.* |
| **TemporaryEmployee** |  | **findEmployeeSalary** | The method finds the employee's salary based on dailyHrsOfWork.   |  |  | | --- | --- | | **dailyHrsOfWork**  **(inclusive)** | **salary** | | 4-6 | 25000 | | 7-9 | 35000 | | 10-12 | 45000 | | 13-15 | 55000 | | Above 15 | 70000 |     **Constraints:**   * When the dailyHrsOfWork doesn't match any of the mentioned above, the method should return -1. * The found salary is returned as double. |

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

**Note:**

* The **employeeType** can be either "Permanent" or "Temporary" (case-sensitive).
* The **PermanentEmployee** and **TemporaryEmployee** implements abstract methods of the Employee class.
* Edit only the **PermanentEmployee** and **TemporaryEmployee** 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 AND OUTPUT 1:**

Enter Employee id

**EMP123**

Enter Employee name

**Jenny**

Enter Employee type

**Permanent**

Enter Employee credit score

**25**

Enter Employee experience

**7**

Permanent Employee

Name : Jenny

Salary : 60000.0

**SAMPLE INPUT AND OUTPUT 2:**

Enter Employee id

**EMP124**

Enter Employee name

**John**

Enter Employee type

**full**

full is an invalid employee type

**PROBLEM 2 :**

**BOT HANDLER :**

In the world of digital assistants, the Bot Handler serves as the gatekeeper, ensuring that each bot's details are accurate and valid before they're put to work. Develop an application to verify the bot details as per the requirements.

Functional Requirements:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Req.# | Requirements Description | Type (Class) | Method Name | Parameters | Description |
| 1 | Include a four-argument constructor in Bot class. | Bot | The getter-setter methods for all the attributes and no-argument constructor are provided as part of code skeleton | String botName, String purpose, String activeStatus, int numberOfUsers | Include a public four-argument constructor in Bot class by following the order mentioned : *botName, purpose, activeStatus, and numberOfUsers* |
| 2 | Include a one-argument constructor in InvalidBotException class | InvalidBotException | - | String message | Include a public one argument constructor to set the message string to the super class. |
| 3 | Extract the details of the Bot , verify the details, and create an object for the Bot class. | Bot | verifyBotDetails | String botDetails | This method is tasked with validating and parsing a string representing bot details. It takes in the string details, splits this string into an array of strings using the colon*( : )* as a delimiter.  It proceeds to check whether each part of the bot details conforms to certain patterns:   * The botName must match the pattern "*BOT/*" followed by exactly *5 digits*. * The purpose must match one of the predefined categories: "*Assistance*", "*Entertainment*", or "*Automation*". * The activeStatus must match either "*active*" or "*inactive*". * The numberOfUsers must be a greater than zero.   If all parts of the details pass the validation, it constructs a Bot object using the parsed details and returns it. However, if any part of the details fails the validation, it throws an *InvalidBotException* with an error message "*Invalid bot details*" indicating that the bot details are invalid.  *Constraints*   * The metod should return the Object of type Bot. * *purspose* and *activeStatus* is case-sensitive. |

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 Bot and InvalidBotException 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:  *<botName>: <purpose>: <activeStatus>: <numberOfUsers>*

Sample Input / Output 1

Enter the Bot Details

BOT/45452:Automation:active:3245

Bot Details

Bot Name: BOT/45452

Purpose: Automation

Active Status: active

Number of Users: 3245

Sample Input / Output 2

Enter the Bot Details

BOT/35452:Technology:inactive:3498

Invalid bot details

**PROBLEM 3 :**

**VIDEO GAME ORGANISER**

In the realm of gaming, video game organisers take charge, ensuring that each video game's information meets the mark before it hits the shelves. Develop an application to verify the details of video games according to set criteria before they're released to players.

**Functional Requirements:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req.#** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Description** |
| 1 | Include a four-argument constructor in the VideoGame class. | **VideoGame** | The getter-setter methods for all the attributes and no-argument constructor are provided as part of the code skeleton. | String title, String genre, String platform, int year | Include a public four-argument constructor in the **VideoGame**class by following the order mentioned: *title, genre,* *platform, and year* |
| 2 | Include a one-argument constructor in InvalidVideoGameException  class | **InvalidVideoGameException** | - | String message | Include a public argument constructor to set the message string to the super class. |
| 3 | Extract the details of the VideoGame, verify the details, and create an object for the VideoGame class. | **VideoGame** | **verifyVideoGameDetails** | String videogameDetails | This method is responsible for validating and parsing a string representing video game details. It takes in the string videogameDetails, splits this string into an array of strings using the colon**(**as a delimiter.  It proceeds to check whether each part of the video game details conforms to certain patterns:   * The genre must match one of the predefined categories: "***Action***", "***Adventure***", or "". * The platform must match one of the predefined categories: "***PS***", "***XBOX***", or "". * The year must be a four-digit integer, representing the release year of the video game.   If all parts of the details pass the validation, it constructs a VideoGame object using the parsed details and returns it. However, if any part of the details fails the validation, it throws an **InvalidVideoGameException**with an error message "**Invalid Video Game Details**" indicating that the video game details are invalid.  ***Constrains***   * The method should return an object of type**V*ideoGame*.** * ***genre***and are case-sensitive. * Assume title is always valid |

**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 **VideoGame**and **InvalidVideoGameException** 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:**  *<title>:<genre>:<platform>:<year>*

**Sample Input / Output 1**

Enter the video game details

**GTA:Action:PC:2013**

Video Game Details

Title: GTA

Genre: Action

Platform: PC

Year: 2013

**Sample Input / Output 2**

Enter the video game Details

**GTA:Action:P3S:2013**

Invalid Video Game Details

**PROBLEM 4 :**

**YOUR NAME IS MINE**

The population of the planet has topped 8 billion users, based on new UN Population Division data. Archer was requested to create a plan in an attempt to control population growth. He is concentrating on marriages this time. Being as smart as he is, Archer devised the following strategy. Write a java program to decide if a couple can be married or not.

**Constraint:**

* Names of the couples should be in the alphabet otherwise, display "**<name 1> is an invalid name**" or "**<name 2> is an invalid name**".
* If both names include a number or other special characters or white space, then display "**Both <name 1> and <name 2> are invalid names**"
* If <name 1> is a subsequence of <name 2> or vice versa, display "**<name 1> and <name 2> are made for each other**".
* If <name 1> is not a sub-sequence of <name 2> or vice versa, display "**<name 1> and <name 2> are not made for each other**".

**Note:**

* Both name 1 and name 2 are case-sensitive.
* 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/Output 1**

Enter the man's name

**john**

Enter the woman's name

**johnan**

john and johnan are made for each other

**Sample Input/Output 2**

Enter the man's name

**surya sivakumar**

Enter the woman's name

**jothik@**

Both surya sivakumar and jothik@ are invalid names

**PROBLEM 5 :**

**TASTY TRAVERSE**

Create the Tasty Traverse application to help organise and publish recipes. It ensures recipes meet specific criteria, empowering users to maintain high-quality culinary content.

**Functional Requirements:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req.#** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Description** |
| 1 | Include a four-argument constructor in the Recipe class. | **Recipe** | The getter-setter methods for all the attributes and no-argument constructor are provided as part of the code skeleton | String recipeName, String cuisineType, int cookingTime, int servings | Include a public four-argument constructor in the **Recipe**class by following the order mentioned: recipeName*,*cuisineType*,* cookingTime*, and servings* |
| 2 | Include a one-argument constructor in InvalidRecipeException  class | **InvalidRecipeException** | - | String message | Include a public one-argument constructor to set the message string to the super class. |
| 3 | Extract the details of the Recipe, verify the details, and create an object for the Recipe class. | **Recipe** | verifyRecipeDetails | String recipeDetails | This method is tasked with validating and parsing a string representing recipe details. It takes inthe string details, splits this string into an array of strings using the colon***(*** as a delimiter.  It proceeds to check whether each part of the recipe details conforms to certain patterns:   * CuisineType must be "***Indian***", "***Italian***", or "". * CookingTime must be greater than 0 and less than 60 minutes. * Servings must be greater than 0 and less than 10 persons.   If all parts of the details pass the validation, it constructs a Recipe object using the parsed details and returns it. However, if any part of the details fails the validation, it throws an InvalidRecipeException with an appropriate error message "***Invalid recipe details*** ".  ***Constraints***   * The method should return an object of type ***Recipe*** * ***cuisineType****is*  case-sensitive. * Assume recipeName is always valid |

**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 **Recipe**and **InvalidRecipeException** 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:**  *<recipeName>:<cuisineType>:<cookingTime>:<servings>*

**Sample Input / Output 1**

Enter the recipe details

**Pizza:Italian:30:5**

Recipe Details

Recipe Name: Pizza

Cuisine Type: Italian

Cooking Time: 30

Servings: 5

**Sample Input / Output 2**

Enter the recipe details

**Pulav:Indian:70:5**

Invalid Recipe Details

**PROBLEM 6 :**

**MAGIC CARDS :**

While spending their summer vacation, Scott and his sister Jerlin decided to create a fun number pattern using magic cards. Scott handed Jerlin green cards with positive numbers and red cards with negative numbers. However, there was a twist: they had to alternate between green and red cards, and any remaining cards were to be added at the end.

Create a Java Program to help Jerlin in creating this new number pattern.

**Constraints:**

* If the total number of green or red cards is zero or less, please print **"<number> is an invalid number.**
* If the embedded green card number is negative or zero, print **"Oops Green card should not contain <number>"**
* If the embedded red card number is positive or zero, print **"Oops Red card should not contain <number>**
* Finally, print the newly formed pattern as**"The new number pattern is <newPattern>"**

**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 and output 1**

Enter the total number of Green cards

**3**

Enter the embedded numbers in the Green Card

**4**

**6**

**5**

Enter the total number of Red cards

**3**

Enter the embedded numbers in the Red Card

**-4**

**-5**

**-6**

The new number pattern is 4 -4 6 -5 5 -6

**Sample input and output 2**

Enter the total number of Green cards

**-1**

-1 is an invalid number

**PROBLEM 7 :**

**HIDDEN WORD :**

The professor gave the students some strings containing both letters and numbers. To find the hidden word, the students need to remove all the numbers from string and then find the hidden word. Michael wants to write a program to solve this problem more efficiently. Can you help him?

**Constraints:**

* Input should be a single String.
* If the input contains more than one word, then print "***Please provide a single word string"***and terminate the program***.***
* The input should contain only alphabets and numbers; otherwise, print "***<String> is an invalid input***" and terminate the program.
* If the input contains only numbers, then print "***No hidden word found***" 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 a string

**Q456u82a0l8i54f1y**

**Sample Output 1**

The resultant string is Qualify

**Sample Input 2**

Enter a string

**J7u43s90t 6B45el8i4ev3e**

**Sample Output 2**

Please provide a single word string

**PROBLEM 8 :**

**ADDRESS ALCHEMY :**

Design an application called Address Alchemy, to verify addresses that becomes effortless and precise, empowering business owners, delivery drivers, and anyone who values accurate location data to confirm addresses with ease and confidence.

**Functional Requirements:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req.#** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Description** |
| 1 | Include a four-argument constructor in the Address class. | **Address** | The getter-setter methods for all the attributes and no-argument constructor are provided as part of the code skeleton | String street, String city, String state, int zip | Include a public four-argument constructor in the **Address**class by following the order mentioned: *street, city,  state, and zip*. |
| 2 | Include a one-argument constructor in InvalidAddressException  class | **InvalidAddressException** | - | String message | Include a public one-argument constructor to set the message string to the superclass. |
| 3 | Extract the details of the Address, verify the details, and create an object for the Address class. | **Address** | verifyAddressDetails | String addressDetails | This method is responsible for validating and parsing a string representing address details. It takes in the string addressDetails, and splits this string into an array of strings using the comma **( , )**as a delimiter.  It proceeds to check whether each part of the address details conforms to certain patterns:   * The city must match one of the city mentioned "**Coimbatore**", "**Chennai**", "" or "**Trichy**". * The state must be "***Tamilnadu***". * The zip should be 6 digit number where, it should start with either 60,  61, 62, 63, or 64.   If all parts of the details pass the validation, it constructs an Address object using the parsed details and returns it. However, if any part of the details fails the validation, it throws an **InvalidAddressException**with the error message "***Invalid Address Details***"indicating that the address details are invalid.  ***Constraints:***   * The method should return an object of type ***Address*.** * ***city****and****state****are*case-sensitive. * Assume the street is always valid. |

**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 **Address**and **InvalidAddressException** 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:**  *<street>,<city>,<state>,<zip>*

**Sample Input / Output 1**

Enter the address details

**4/5 3rd street,Madurai,Tamilnadu,600032**

Address Details

Street: 4/5 3rd street

City: Madurai

State: Tamilnadu

Zip: 600032

**Sample Input / Output 2**

Enter the address details

**3rd street,Madurai,Karanatka,600032**

Invalid Address Details

**PROBLEM 9 :**

**CONVERGENCE CALCULATOR**

John's teacher gives him an assignment on the topic "array." He got two arrays of the same size as the input, and he had to add the common elements of both arrays.

Help John complete his assignment.

**Constraints:**

* If the first or second array size is less than or equal to 0, the program should print "**Array size should be positive**".
* Both array size should be same, otherwise, program should print "**Arrays size should be same**"
* If there are no common elements in both arrays, the program should print "**There are no common elements in these arrays**".
* Finally, the output should print "**Sum of common elements is <ans>**".

**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 first array size

**4**

Enter the first array elements

**6**

**8**

**2**

**3**

Enter the second array size

**4**

Enter the second array elements

**4**

**7**

**6**

**2**

**Sample output 1:**

Sum of common elements is 8

**Sample input 2:**

Enter the first array size

**-5**

**Sample output 2:**

Array size should be positive

**PROBLEM 10 :**

**TIME PIECE**

Design the application Timepiece to simplify the process of verifying watch details, providing a hassle-free solution for watch enthusiasts, collectors, and sellers.

**Functional Requirements:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req.#** | **Requirements Description** | **Type (Class)** | **Method Name** | **Parameters** | **Description** |
| 1 | Include a four-argument constructor in the Watch class. | **Watch** | The getter-setter methods for all the attributes and no-argument constructor are provided as part of the code skeleton | String brand, String strapType, String displayType, String isSmart | Include a public four-argument constructor in the **Watch**class by following the order mentioned: *brand, strapType, displayType, and isSmart*. |
| 2 | Include a one-argument constructor in InvalidWatchException  class | **InvalidWatch Exception** | - | String message | Include a public one-argument constructor to set the message string to the super class. |
| 3 | Extract the details of the Watch, verify the details, and create an object for the Watch class. | **Watch** | verifyWatchDetails | String watchDetails | This method is tasked with validating and parsing a string representing watch details. It takes in the string details, splits this string into an array of strings using the comma **(,)** as a delimiter.  It proceeds to check whether each part of the watch details conforms to certain patterns:   * The ***strapType***must match one of the predefined type: **Leather**, , or **Plastic** * *The****displayType****must match either****Analog****or .* * ***isSmart***must be **Yes** or .   If all parts of the details pass the validation, it constructs a Watch object using the parsed details and returns it. However, if any part of the details fails the validation, it throws an **InvalidWatchException**with an appropriate error message indicating **"Invalid watch details** ".  ***Constraints:***   * The method should return an object of type ***Watch* .** * ***strapType, displayType****and****isSmart****are*case-sensitive. * Assume the brand is always valid. |

**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 **Watch**and **InvalidWatchException** 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:**  *<brand>:<strapType>:<displayType>:<isSmart>*

**Sample Input / Output 1**

Enter the watch details

**Apple,Metal,Digital,Yes**

Watch Details

Brand: Apple

Strap type: Metal

Display type: Digital

Is smart: Yes

**Sample Input / Output 2**

Enter the watch details

**Apple,Silicon,Digital,No**

Invalid watch details