(Lab 4) Problem Set 3: Methods, Classes, & APIs

P3 Solutions limited in scope to:		
• P1 Concepts • P2 Concepts	• Methods Declarations ○ Passing data in ○ Returning data out	• Method Invocations ○ Use Java API methods ○ Use pre-existing methods

Submission Rules:

1. Submissions must be zipped into a **handin.zip** file. Each problem must be implemented in its own class file. Use the name of the problem as the class name.

2. You must use standard input and standard output for ALL your problems. It means that the input should be entered from the keyboard while the output will be displayed on the screen.

3. Your source code files should include a comment at the beginning including your name and that problem number/name.

4. The output of your solutions must be formatted exactly as the sample output to receive full credit for that submission.

5. Compile & test your solutions before submitting.

6. Each problem is worth up to 10 points total. The breakdown is as follows: 2 points for compiling, 3 points for correct output with sample inputs, 5 points for additional inputs.

7. This lab is worth a max total of: 40 points. You can complete as many problems as you like, but cannot receive more than 40 points towards the lab grade. All points in excess of that are for bragging rights. (Check the scoreboard to see how you did!)

8. Submission:

- You have unlimited submission attempts until the deadline passes
- You'll receive your lab grade immediately after submitting
- **IMPORTANT:** if your grade is lower than 70% when the deadline passes, then you must attend a recitation session & get TA signoff to receive full credit for that lab challenge.

Problem 1: Geometry (10 points) Make API

(API design) Java is an extensible language, which means you can expand the programming language with new functionality by adding new classes. You are tasked to implement a Geometry class for Java that includes the following API *(Application Programming Interface)*:

Geometry Method API:

Modifier and Type	Method and Description
`static double`	**`getAreaRectangle`**`(double width, double length)` Returns the area of a rectangle, $\text{area} = \text{length} * \text{width}$
`static double`	**`getAreaCircle`**`(double radius)` Returns the area of a circle, $\text{area} = \boldsymbol{\pi}(\text{radius}^2)$
`static double`	**`getAreaTriangle`**`(double base, double height)` Returns the area of a triangle, $\text{area} = ½(\text{base} * \text{height})$
`static double`	**`getPerimeterRectangle`**`(double width, double length)` Returns the perimeter of a Rectangle, $\text{perimeter} = 2(\text{length} + \text{width})$
`static double`	**`getPerimeterCircle`**`(double radius)` Returns the perimeter of a Circle, $\text{perimeter} = 2\boldsymbol{\pi}(\text{radius})$
`static double`	**`getPerimeterTriangle`**`(double side1, double side2, double side3)` Returns the perimeter of a triangle, $\text{perimeter} = \text{s1} + \text{s2} + \text{s3}$

Facts

- Java **`Math`** class contains an approximation for **`PI`**, i.e. **`Math.PI`**
- Your **`Geometry`** class implementation should ***not*** have a **`main`** method.
- <u>NO</u> **`Scanner`** for input & <u>NO</u> **`System.out`** for output!

Input

The **`Geometry`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`Geometry`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***
`getAreaRectangle(1,1);`	`1.0`
`getAreaCircle(1);`	`3.141592653589793`
`getAreaTriangle(1,1);`	`0.5`
`getPerimeterRectangle(1,1);`	`4.0`
`getPerimeterCircle(1);`	`6.283185307179586`
`getPerimeterTriangle(1,1,1);`	`3.0`

Problem 2: Time (10 points) Make API

(*API design*) Java is an extensible language, which means you can expand the programming language with new functionality by adding new classes. You're tasked to implement a Time class for Java that includes the following API *(Application Programming Interface)*:

Time Method API:

Modifier and Type	Method and Description
`static double`	**`secondsToMinutes`**`(int seconds)` Returns number of minutes from seconds, 1 minute = 60 seconds
`static double`	**`secondsToHours`**`(int seconds)` Returns number of hours from seconds, 1 hour = 60 minutes
`static double`	**`secondsToDays`**`(int seconds)` Returns number of days from seconds, 1 day = 24 hours
`static double`	**`secondsToYears`**`(int seconds)` Returns number of years from seconds, 1 year = 365 days
`static double`	**`minutesToSeconds`**`(double minutes)` Returns number of seconds from minutes, 1 minute = 60 seconds
`static double`	**`hoursToSeconds`**`(double hours)` Returns number of seconds from hours, 1 hour = 60 minutes
`static double`	**`daysToSeconds`**`(double days)` Returns number of seconds from days, 1 day = 24 hours
`static double`	**`yearsToSeconds`**`(double years)` Returns number of seconds from hours, 1 year = 365 days

Facts

- Use **`double`** literals in your conversion calculations
- Your **`Time`** class implementation should ***not*** have a **`main`** method.
- NO **`Scanner`** for input & NO **`System.out`** for output!

Input

The **`Time`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`Time`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns *(Not Printouts)*
`secondsToMinutes(1);`	`0.016666666666666666`
`secondsToHours(1);`	`2.777777777777778E-4`
`secondsToDays(1);`	`1.1574074074074073E-5`
`secondsToYears(1);`	`3.1709791983764586E-8`
`minutesToSeconds(1.0);`	`60.0`
`hoursToSeconds(1.0);`	`3600.0`
`daysToSeconds(1.0);`	`86400.0`
`yearsToSeconds(1.0);`	`3.1536E7`

Problem 3: Cooking (10 points) Make API

(API design) Java is an extensible language, which means you can expand the programming language with new functionality by adding new classes. You're tasked to implement a Cooking class for Java that includes the following API *(Application Programming Interface)*:

Cooking Method API:

Modifier and Type	Method and Description
`static double`	**`teaspoonsToTablespoons`**`(double teaspoons)` Returns number of TBSPs from TSP, 1 tablespoon = 3 teaspoons
`static double`	**`tablespoonsToTeaspoons`**`(double tablespoons)` Returns number of TSPs from TBSP, 1 tablespoon = 3 teaspoons
`static double`	**`tablespoonsToCups`**`(double tablespoons)` Returns number of CUPs from TBSP, 1 cup = 16 tablespoons
`static double`	**`cupsToTablespoons`**`(double cups)` Returns number of TBSPs from CUPs, 1 cup = 16 tablespoons
`static double`	**`ouncesToCups`**`(double ounces)` Returns number of CUPs from OUNCEs, 1 cup = 8 ounces
`static double`	**`cupsToOunces`**`(double cups)` Returns number of OUNCEs from CUPs, 1 cup = 8 ounces
`static double`	**`cupsToPints`**`(double cups)` Returns number of PINTs from CUPs, 1 pint = 2 cups
`static double`	**`pintsToCups`**`(double pints)` Returns number of CUPs from PINTs, 1 pint = 2 cups

Facts

- Your **`Cooking`** class implementation should ***not*** have a **`main`** method.
- NO **`Scanner`** for input & NO **`System.out`** for output!

Input

The **`Cooking`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`Cooking`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns *(Not Printouts)*
`teaspoonsToTablespoons(1.0);`	`0.3333333333333333`
`tablespoonsToTeaspoons(1.0);`	`3.0`
`tablespoonsToCups(1.0);`	`0.0625`
`cupsToTablespoons(1.0);`	`16.0`
`ouncesToCups(1.0);`	`0.125`
`cupsToOunces(1.0);`	`8.0`
`pintsToCups(1.0);`	`2.0`
`cupsToPints(1.0);`	`0.5`

Problem 4: Temperature (10 points) Make API

(API design) Java is an extensible language, which means you can expand the programming language with new functionality by adding new classes. You're tasked to implement a Temperature class for Java that includes the following API *(Application Programming Interface)*:

Temperature Method API:

Modifier and Type	**Method and Description**
`static double`	**`celsiusToFahrenheit`**`(double celsius)` Returns degrees Fahrenheit from degrees Celsius, $F=\frac{9}{5}C+32$
`static double`	**`celsiusToKelvin`**`(double celsius)` Returns degrees Kelvin from degrees Celsius, $K=C+273.15$
`static double`	**`fahrenheitToCelsius`**`(double fahrenheit)` Returns degrees Celsius from degrees Fahrenheit, $C=\frac{5}{9}(F-32)$
`static double`	**`fahrenheitToKelvin`**`(double fahrenheit)` Returns degrees Kelvin from degrees Fahrenheit, $K=\frac{5}{9}(F+459.67)$
`static double`	**`kelvinToFahrenheit`**`(double kelvin)` Returns degrees Fahrenheit from degrees Kelvin, $F=\frac{9}{5}K-459.67$
`static double`	**`kelvinToCelsius`**`(double kelvin)` Returns degrees Celsius from degrees Kelvin, $C=K-273.15$

Facts

- Your **`Temperature`** class implementation should ***not*** have a **`main`** method.
- NO **`Scanner`** for input & NO **`System.out`** for output!

Input

The **`Temperature`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`Temperature`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	**Sample Method Returns** ***(Not Printouts)***
`celsiusToFahrenheit(1.0);`	`33.8`
`celsiusToKelvin(1.0);`	`274.15`
`fahrenheitToCelsius(1.0);`	`-17.22222222222222`
`fahrenheitToKelvin(1.0);`	`255.92777777777778`
`kelvinToFahrenheit(1.0);`	`-457.87`
`kelvinToCelsius(1.0);`	`-272.15`

Problem 5: Autocorrect (10 points) Use API

(Text Processing) You've just been hired by Apple to implement a new autocorrect feature in their text messenger app for the iPhone XX . The innovative idea behind this improved autocorrect feature is that Apple will use each individual's personal data to construct a custom dictionary for each and every user. Your autocorrect software must scan a line of user text and then use that custom dictionary to look for certain misspellings and replace them with the suggested spelling. So given a line of text, your software must replace keywords with new replacement words. Your autocorrect program is provided with this custom dictionary, which is given as a sequence of keywords and associated values, where the first word is the misspelling and the second term is the proper spelling.

Facts

- Java **`String`** class has a **`replace`** method in its API
  - https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
- Your solution should have a **`main`** method.
- **Use** **`Scanner`** for input & **Use** **`System.out`** for output!

Input

The first line of input is the number of test cases. Each test case consists of two lines of input. The first line of each test case represents the user's inputted text message as a String. The second line represents the custom dictionary consisting of a String of multiple key-value pairs separated by spaces.

Output

The software should print the new message replacing the key terms with the appropriate replacement terms.

Sample Input	Sample Output

`3`
`Helo Wolrd`
`Helo Hello Wolrd World`
`yuDodat?`
`y Why u You dat That`
`h3110 70 411`
`3 e 1 l 0 o 4 a 7 t`
	`Hello World`
	`WhyYouDoThat?`
	`hello to all`

Problem 6: Log It (10 points) Use API

(Data Structure Algorithms) High-Low is a simple number guessing game where one player thinks of a random integer number between 0 to some maximum value and another player attempts to guess that number. With each turn, player one tells player two whether their guess was correct, too high, or too low. The optimal strategy for playing this game is to select the middle value between the largest and smallest possible numbers. In Computer Science, this strategy is called a Binary Search Algorithm. Binary search algorithms eliminate half the possible search space with each pass. Your task is to determine the maximum number of tries it takes to guess the number correctly using the Binary Search algorithm. This can be determined by taking the log2 value of the maximum number. Since this result represents the max number of guesses round any fractional result up to the next highest integer.

Facts

- Log2(number) is calculated using $guesses = \frac{\log_{\square}(max\ number)}{\log(2)}$
- Java **`Math`** class contains ***`log`*** and ***`ceil`*** methods
  - https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html
- Your solution should have a **`main`** method.
- **Use** **`Scanner`** for input & **Use** **`System.out`** for output!

Input

The first line of input represents the number of test cases. Each case contains one positive integer number that represents the maximum numbered value in the High-Low game.

Output

Print the integer that represents the maximum number of guesses it would take to find the secret number using binary search algorithm.

Sample Input	Sample Output
4 10 100 1000 10000	4 7 10 14

Problem 7: Distance (10 points) Use API

(Data Science) Data Science is an emergent field from Computer Science with applications in almost every domain including finances, medical research, entertainment, retail, advertising, and insurance. The

role of a data analyst is to aggregate data points and make conclusions based on how that data clusters together. Fundamentally, this is how Pandora selects what song to play next, how Netflix determines what shows to recommend next, how Amazon chooses which products to promote together, how banks determine whether to issue a loan, and how hedge companies decide which stocks to invest into. One critical component that empowers data analysts to make such predictions is to determine how close or far two data points are in relation to one another. This is accomplished using Trigonometry, a field of study in mathematics which observes the relationships of the sides and angles of triangles. By mapping the data points onto a 2d chart, it is then possible to use the distance formula (*pythagorean theorem*) to calculate the geometric distance between these two points. Your task in this problem is to calculate the distance between two points.

Facts

- Distance formula: $distance=\sqrt{\square}$
- Java **`Math`** class contains ***`sqrt`*** and ***`pow`*** methods
  - https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html
- Your solution should have a **`main`** method.
- **Use** **`Scanner`** for input & **Use** **`System.out`** for output!

Input

The first line of input is the number of test cases. Each line afterwards contains four double values. The first two values represent the first point's coordinates as (x_1,y_1). The second set of values represent the second point's coordinates as (x_2,y_2).

Output

Print the distance between the two points as a double type. Use *`System.out.println()`* on your result.

Sample Input	Sample Output

`3`
`0 0 2 2`
`-1 -1 1 1`
`0 0 0.5 0.5`
	`2.8284271247461903`
	`2.8284271247461903`
	`0.7071067811865476`

Problem 8: Enemy Move (10 points) Use API

(Game Development) You're leading the development of the highly anticipated game, Z3ldar 3. You've been challenged to implement the enemy movement within the game. Each game update, the enemy tries to move toward the player. Enemies movements are limited by their speed. If the enemy speed is greater than the distance from the player, then the enemy moves to the player's coordinates and attacks. However, if the enemy speed is less than the distance to the player, then the enemy moves in shortest

path toward the player. The enemy's location is tracked by (x_1,y_1) values and the player's location is tracked by (x_2,y_2) values. Given the current position of the enemy, the player position, and the enemy speed you must determine the new position of the enemy.

Facts

- Distance formula: $distance = \sqrt{\square}$
- Enemy's new x coord formula: $dx = x_1 + \frac{speed}{distance}(x_2 - x_1)$,only if enemy cannot reach player.
- Enemy's new y coord formula: $dy = y_1 + \frac{speed}{distance}(y_2 - y_1)$,only if enemy cannot reach player.
- Java **`Math`** class contains ***`sqrt`*** and ***`pow`*** methods
    - https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html
- Your solution should have a **`main`** method.
- **Use** **`Scanner`** for input & **Use** **`System.out`** for output!

Input

The first input is a positive integer representing the number of test cases. Each test case contains five non-negative double values. The first two inputs represents the enemy (x_1,y_1) position *(start)*. The third number represents the enemy's speed. The last two inputs represent the player (x_2,y_2) position *(end)*.

Output

The program must display the final enemy x,y position after moving. The x,y coords should be displayed with one precision point. Terminate each set of coordinates with a new line (*n*) character.

Sample Input	Sample Output

`3`
`0 0 4 10 10`
`-1 -1 2 1 1`
`0.0 0.0 10 1.0 1.0`
	`x=2.8, y=2.8`
	`x=0.4, y=0.4`
	`x=1.0, y=1.0`