1.Write a program to print the area of a rectangle by creating a class named 'Area' having two methods. First method named as 'setDim' takes length and breadth of rectangle as parameters and the second method named as 'getArea' returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.
****

2. Create a class named 'Student' with String variable 'name' and integer variable 'roll_no'. Assign the value of roll_no as '2' and that of name as "John" by creating an object of the class Student.
****

3.Assign and print the roll number, phone number and address of two students having names "Sam" and "John" respectively by creating two objects of class 'Student'.
****

4.Write a program to print the area and perimeter of a triangle having sides of 3, 4 and 5 units by creating a class named 'Triangle' without any parameter in its constructor.
****

5.Write a program to print the area and perimeter of a triangle having sides of 3, 4 and 5 units by creating a class named 'Triangle' with constructor having the three sides as its parameters.
****

6.Write a program to print the area of two rectangles having sides (4,5) and (5,8) respectively by creating a class named 'Rectangle' with a method named 'Area' which returns the area and length and breadth passed as parameters to its constructor.
****

7.Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered

through keyboard.
****

8.Print the average of three numbers entered by user by creating a class named 'Average' having a method to calculate and print the average.
****

9.Write a Java class Clock for dealing with the day time represented by hours, minutes, and seconds. Your class must have the following features:

- Three instance variables for the hours (range 0 - 23), minutes (range 0 - 59), and seconds (range 0 - 59).
- Three constructors:
  - default (with no parameters passed; is should initialize the represented time to 12:0:0)
  - a constructor with three parameters: hours, minutes, and seconds.
  - a constructor with one parameter: the value of time in seconds since midnight (it should be converted into the time value in hours, minutes, and seconds)
- Instance methods:
  - a set-method method setClock() with one parameter seconds since midnight (to be converted into the time value in hours, minutes, and seconds as above).
  - get-methods getHours(), getMinutes(), getSeconds() with no parameters that return the corresponding values.
  - set-methods setHours(), setMinutes(), setSeconds() with one parameter each that set up the corresponding instance variables.
  - method tick() with no parameters that increments the time stored in a Clock object by one second.
  - method addClock() accepting an object of type Clock as a parameter. The method should add the time represented by the parameter class to the time represented in the current

class.

- Add an instance method toString() with no parameters to your class. toString() must return a String representation of the Clock object in the form "(hh:mm:ss)", for example "(03:02:34)".
- Add an instance method tickDown() which decrements the time stored in a Clock object by one second.
- Add an instance method subtractClock() that takes one Clock parameter and returns the difference between the time represented in the current Clock object and the one represented by the Clock parameter. Difference of time should be returned as an clock object.

Write a separate class ClockDemo with a main() method. The program should:

- instantiate a Clock object firstClock using one integer seconds since midnight obtained from the keyboard.
- tick the clock ten times by applying its tick() method and print out the time after each tick.
- Extend your code by appending to it instructions instantiating a Clock object secondClock by using three integers (hours, minutes, seconds) read from the keyboard.
- Then tick the clock ten times, printing the time after each tick.
- Add the secondClock time in firstClock by calling method addClock.
- Print both clock objects calling toString method

Create a reference thirdClock that should reference to object of difference of firstClock and secondClock by calling the method subtractClock().

****

10.Write a Java class Complex for dealing with complex number. Your class must have the following features:

- Instance variables :
    - realPart for the real part of type double
    - imaginaryPart for imaginary part of type double.
- Constructor:
    - public Complex (): A default constructor, it should initialize the number to 0, 0)
    - public Complex (double realPart, double imaginaryPart): A constructor with parameters, it creates the complex object by setting the two fields to the passed values.
- Instance methods:
    - public Complex add (Complex otherNumber): This method will find the sum of the current complex number and the passed complex number. The methods returns a new Complex number which is the sum of the two.
    - public Complex subtract (Complex otherNumber): This method will find the difference of the current complex number and the passed complex number. The methods returns a new Complex number which is the difference of the two.
    - public Complex multiply (Complex otherNumber): This method will find the product of the current complex number and the passed complex number. The methods returns a new Complex number which is the product of the two.
    - public Complex divide (Complex otherNumber): This method will find the ... of the current complex number and the passed complex number. The methods returns a new Complex number which is the ... of the two.
    - public void setRealPart (double realPart): Used to set the real part of this complex number.
    - public void setImaginaryPart (double realPart): Used to set the imaginary part of this complex number.

    - public double getRealPart(): This method returns the real part of the complex number
    - public double getImaginaryPart(): This method returns the imaginary part of the complex number
    - public String toString(): This method allows the complex number to be easily printed out to the screen

Write a separate class ComplexDemo with a main() method and test the Complex class methods.

****

11. Write a Java class Author with following features:

- Instance variables :
    - firstName for the author's first name of type String.
    - lastName for the author's last name of type String.
- Constructor:
    - public Author (String firstName, String lastName): A constructor with parameters, it creates the Author object by setting the two fields to the passed values.
- Instance methods:
    - public void setFirstName (String firstName): Used to set the first name of author.
    - public void setLastName (String lastName): Used to set the last name of author.
    - public double getFirstName(): This method returns the first name of the author.
    - public double getLastName(): This method returns the last name of the author.
    - public String toString(): This method printed out author's name to the screen

Write a Java class Book with following features:

- Instance variables :
    - title for the title of book of type String.

  - author for the author's name of type String.
  - price for the book price of type double.
- Constructor:
  - public Book (String title, Author name, double price): A constructor with parameters, it creates the Author object by setting the the fields to the passed values.
- Instance methods:
  - public void setTitle(String title): Used to set the title of book.
  - public void setAuthor(String author): Used to set the name of author of book.
  - public void setPrice(double price): Used to set the price of book.
  - public double getTitle(): This method returns the title of book.
  - public double getAuthor(): This method returns the author's name of book.
  - public String toString(): This method printed out book's details to the screen

Write a separate class BookDemo with a main() method creates a Book titled "Developing Java Software" with authors Russel Winderand price 79.75. Prints the Book's string representation to standard output (using System.out.println).

****

Create a class called Wrestler for representing a wrestler in a wrestling simulation program, which simulates the result of a wrestling match with two wrestler based on their wrestling attributes. A wrestler has the following attributes.

Attribute Name	Description	Value
Name	The name of the wrestler	a text
Power(pow)	Wrestlers cannot fight when their power reach 0.	a number less than Max. power
Max. power	The power at the starting of a match	100
Strength (str)	The fighting strength of the wrestler	an integer from 1 to 20
Toughness (tou)	The ability to endure damage	an integer from 1 to 20
Stamina (sta)	A wrestler with high stamina can fight longer.	an integer from 1 to 20
Speed (spd)	The agility of the wrestler	an integer from 1 to 20
Luck (lck)	Luck determines how lucky the wrestler is.	an integer from 1 to 20
Courage (cou)	A wrestler with high courage tends to become boosted more often.	an integer from 1 to 20
Skill (skl)	A wrestler with high skill can turn defense to attack more frequently.	an integer from 1 to 20

Wrestler must have a constructor that sets the variable **`name`** to a given *String*. All data members of the class must be **`private`**.

Also provide appropriate mutator and accessor methods. All accessor methods are **`public`**. However, for **`str`**, **`sta`**, **`tou`**, **`spd`**, **`lck`**, **`cou`**, and **`skl`**, make their mutator methods **`private`** and have each of them check whether its input value are in the valid range (1-20). Mutator methods whose inputs are not in the valid range must return **`false`**. Otherwise, they must return **`true`**.

Write a public method *rest()*, which resets **`pow`** to the value of max. power.

Write a public method *reducePow()* that reduce the current **`pow`** by a **`double`** value taken as its only input argument.

Furthermore, provide a public method called *setAttr()* that takes in the 7 **`int`** values to be assigned to **`str`**, **`tou`**, **`sta`**, **`spd`**, **`lck`**, **`cou`**, and **`skl`**, and makes use of all private mutator methods to set the values appropriately. This method must return `false` if any one of the private mutators return `false`. Otherwise, it must return `true`.

11. Add *toString()* to Wrestler in the previous problem so that it returns a *String* in the following format: `Wrestler: [name]`. For example, `System.out.print(x)` would print `Wrestler: Hulk Hogan` if `x` refers to a *Wrestler* object with the instance variable `name` being "Hulk Hogan".

12. The fighting between two wrestlers in a wrestling match in our wrestling simulator is simulated in turns. Each turn follows the following steps.

    1) Determine who initiates the assault.
    Each turn starts with determining which wrestler does the attacking. For each wrestler, a random `double` number *d* is uniformly drawn from 0 to (the wrestler's `spd`)+(the wrestler's `lck`+the wrestler's `cou`)/8.0. The wrestler with the bigger *d* takes the *attack chance*. Redo if the two wrestlers have got the same values of *d*. (A tie in the value of two randomly picked `double` is rare.)

2) Counterattack.
Before the actual attack takes place, the attacker might loose his *attack chance* if the defender successfully counterattacks. For each wrestler, a random `double` number *c* is uniformly drawn from 0 to (the wrestler's `skl`+(the wrestler's `lck`)/8.0). A successful counterattack takes place if the defender has a bigger *c* than the attacker. When the counterattack takes place, the original defender takes the *attack chance* from the attacker.

3) Deal the damage.
The wrestler with the *attack chance* ($w1$) can now deal some damage to the other wrestler ($w2$). A `double` number *dmg* is calculated from:

$$dmg = (\texttt{str}\text{ of }w1+10)-(\texttt{tou}\text{ of }w2)+e$$

where *e* is a random `double` number uniformly taken from 0 to (`cou` of $w1$)/4.0+(`skl` of $w1$)/4.0-(`skl` of $w2$)/4.0+(`lck` of $w1$)/8.0-(`lck` of $w2$)/8.0

If *dmg* is less than 0, *dmg* is set to 0. Then, *dmg* is subtracted from *w2*'s current `pow`.

4) Account for tiredness.
After the damage dealing step, both wrestlers get some reduction in their `pow` values. The reduction *r* in `pow` is a `double` value and is calculated by:

$$r = (\text{current } \mathtt{pow} / \text{Max. Power}) \times 20 \times (\mathtt{sta}+1)^{-1}$$

5) Check whether the power is depleted.
A wrestling match is over if, after a turn is finished, `pow` values of any one of the two wrestlers are less than or equal to 0. Otherwise, a next turn is executed following similar steps, with the `pow` value of each wrestler remains what it is at the end of the current turn.

If the match is over, a wrestler with positive `pow` is the winner. If there is no winner, the match is called a tie.

Now, create a class called *FightingRule* that contains *non-static* methods needed for commencing a wrestling match according to the steps above. These methods include:

```
int determineAttacker(Wrestler w1, Wrestler w2)
```

The method returns 1 if `w1` takes the attack chance as the result of step 1). It returns 2 if `w2` takes the *attack chance*.

```
boolean canCounter(Wrestler w1, Wrestler 2)
```

The method assumes that `w1` is the attacker and `w2` is the defender. It returns `true` if a counterattack takes place as the result of step 2). Otherwise it returns `false`.

```
double dealDamage(Wrestler w1, Wrestler w2)
```

The method assumes that after step 2) in the current turn `w1` has the *attack chance*. It returns the value of inflicted damage according to the rule explained in step 3). This method also reduce `pow` of `w2` according to the damage inflicted.

```
void exhaust(Wrestler w)
```

****