Exercises: Enumerations and Attributes

This document defines the exercises for "C# OOP Basics" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

Problem 1. Card Suit

Create an enumeration type that has as its constants the four suits of a deck of playing cards (Clubs, Hearts, Diamonds, Spades). Iterate over the values of the enumeration type and print all ordinal values and names.

Examples

Input	Output

`Card Suits`	`Card Suits:`
	`Ordinal value: 0; Name value: Clubs`
	`Ordinal value: 1; Name value: Diamonds`
	`Ordinal value: 2; Name value: Hearts`
	`Ordinal value: 3; Name value: Spades`

Problem 2. Card Rank

Create an enumeration type that has as its constants the fourteen ranks of a deck of playing cards (Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King). Iterate over the values of the enumeration type and print all ordinal values and names.

Examples

Input	Output

`Card Ranks`	`Card Ranks:`
	`Ordinal value: 0; Name value: Ace`
	`Ordinal value: 1; Name value: Two`
	`Ordinal value: 2; Name value: Three`
	`Ordinal value: 3; Name value: Four`
	`Ordinal value: 4; Name value: Five`
	`Ordinal value: 5; Name value: Six`
	`Ordinal value: 6; Name value: Seven`
	`Ordinal value: 7; Name value: Eight`
	`Ordinal value: 8; Name value: Nine`
	`Ordinal value: 9; Name value: Ten`
	`Ordinal value: 10; Name value: Jack`
	`Ordinal value: 11; Name value: Queen`
	`Ordinal value: 12; Name value: King`

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

Problem 3. Card Power

Create **class Card** that holds **Rank** and **Suit**. Create a program that generates a deck of cards which have a power. The power of a card is calculated by adding the power of its rank plus the power of its suit.

Rank powers are as follows: (Ace - 14, Two - 2, Three - 3, Four - 4, Five - 5, Six - 6, Seven - 7, Eight - 8, Nine - 9, Ten - 10, Jack - 11, Queen - 12, King - 13).

Suit powers are as follows: (Clubs - 0, Diamonds - 13, Hearts - 26, Spades - 39).

You will get a command consisting of two lines. On the first line you will receive the Rank of the card and on the second line you will get the suit of the card.

Print the output in the format "Card name: ACE of SPADES; Card power: 53".

Note

Try using the enumeration types you have created in the previous problems but extending them with constructors and methods. To get the card power cast to integer Rank and Suit and add them together.

Examples

Input	Output
Two Clubs	Card name: Two of Clubs; Card power: 2
Ace Spades	Card name: Ace of Spades; Card power: 53

Problem 4. Card ToString()

If you haven't done it already override the **ToString()** of your **Card** class you've created earlier. Make it so it returns the same information as before e.g. in format:

"Card name: {Rank} of {Suit}; Card power: {Card power}"

Examples

Input	Output
Two Clubs	Card name: Two of Clubs; Card power: 2
Ace Spades	Card name: Ace of Spades; Card power: 53

Problem 5. Card CompareTo()

As your cards have power you can safely add a functionality for comparing them. Try using the already available interface **IComparable<T>** and override the **CompareTo()** method.

Read two cards from the console and print the greater of the two. In the given format:

"Card name: {Rank} of {Suit}; Card power: {Card power}"

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

Examples

Input	Output
Two Clubs Ace Spades	Card name: Ace of Spades; Card power: 53

Problem 6. Custom Enum Attribute

Create a custom attribute **TypeAttribute** that can be applied to classes and can be accessed at runtime. The **TypeAttribute** elements should contain **type, category** and **description** as parameters. Apply the attribute to both enumeration types you have created for the previous problems (Rank and Suit). Provide them these exact values:

Rank:

- type = "Enumeration"
- category = "Rank"
- description = "Provides rank constants for a Card class."

Suit:

- type = "Enumeration"
- category = "Suit"
- description = "Provides suit constants for a Card class."

Create a program which gets the description of an enumeration type by a given rank.

Note

Try using the **typeof(TypeAttribute).GetCustomAttributes()** method.

Examples

Input	Output
Rank	Type = Enumeration, Description = Provides rank constants for a Card class.

Problem 7. Deck of Cards

Create a program that generates all cards of a card playing deck. First print the clubs, starting from the Ace, ending with a King. Continue with the same cards from Hearts, Diamonds and Spades. Print them in the format given below.

Note

Try using the enumeration types you have created in the previous problems.

Examples

Input	Output

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

Card Deck	Ace of Clubs Two of Clubs Three of Clubs Four of Clubs Five of Clubs ... ... King of Spades

Problem 8. Card Game

Simulate a card game in which you have two players. Each player has a hand of five cards. The winning player is the player which holds the highest powered card in his hand.

Rank powers are as follows: (Ace - 14, Two - 2, Three - 3, Four - 4, Five - 5, Six - 6, Seven - 7, Eight - 8, Nine - 9, Ten - 10, Jack - 11, Queen - 12, King - 13).

Suit powers are as follows: (Clubs - 0, Diamonds - 13, Hearts - 26, Spades - 39).

Input

On the first two lines you will get the names of the players.

On the next lines, you should **read cards** from the console in the format **{Ace of Clubs}** for a certain player until he has **exactly 5 cards in his hand**. If he receives a card that is not in the deck, you should print "Card is not in the deck.". If he receives an invalid card name, for example "spades of Ace", print "No such card exists.".

Output

Print the name of the winner and his winning card in the format "`{Player name} wins with {Card name}`.".

Examples

Input	Output	Comments
Ivo Gosho Queen of Diamonds King of Diamonds **Ace of Hearts** Ace of Hearts spades of Ace Two of Hearts Three of Hearts Four of Hearts Five of Hearts Six of Hearts Seven of Hearts Eight of Hearts	Card is not in the deck. No such card exists. Ivo wins with Ace of Hearts.	Player Ivo receives cards (in orange) from the deck, until he has exactly five of them. When he is given ACE of HEARTS for a second time, error message is printed and his hand stays the same size. When a card with invalid name is given, error message is printed and his hand stays the same size. When Ivo's hand has 5 cards, Gosho starts receiving cards from the deck. When Gosho has 5 cards, the hands

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

		are evaluated and one of the players wins.

Problem 9. Traffic Lights

Implement a simple state machine in the form of a traffic light. Every traffic light has three possible signals - red, green and yellow. Each traffic light can be updated, which changes the color of its signal (e.g. if it is currently red, it changes to green, if it is green it changes to yellow). The order of signals is red -> green -> yellow -> red and so on.

On the first line you will be given multiple traffic light signals in the format "Red Green Yellow". You need to make as many traffic lights as there are signals in the input.

On the second line, you will receive the **n** number of times you need to change each traffic light's signal.

Your output should consist of **n** number of lines, including each updated traffic light's signal. To better understand the problem, see the example below.

Examples

Input	Output
Green Red Yellow 4	Yellow Green Red Red Yellow Green Green Red Yellow Yellow Green Red

Problem 10. *Inferno Infinity

If you've been involved with the creation of Inferno III last year, you may be informed of the disastrous critic reception it has received. Nevertheless, your company is determined to satisfy its fan base, so a sequel is coming and yeah, you will develop the crafting module of the game using the latest OOP trends.

You have three different weapons (Axe, Sword and Knife) which have base stats and a name. The base stats are min damage, max damage and number of sockets (sockets are basically holes, in which you can insert gems). Below are the base stats for the three weapon types:

- Axe (5-10 damage, 4 sockets)
- Sword (4-6 damage, 3 sockets)
- Knife (3-4 damage, 2 sockets)

What's more, every weapon comes with a different level of rarity (how rare it is to come across such an item). Depending on its rarity, a weapon's maximum and minimum damage can be modified.

- Common (increases damage x1)
- Uncommon (increases damage x2)
- Rare (increases damage x3)
- Epic (increases damage x5)

So a Common Axe would have its damage modified in the following way: minimum damage = 5 *1, maximum damage = 10 *1. Whereas an Epic Axe would look like this: minimum damage = 5 * 5, maximum damage = 10 * 5.

© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.

Additionally, every weapon provides a bonus to three magical stats - strength, agility and vitality. At first the bonus of every magical stat is zero and can be increased with gems which are inserted into the weapon.

Every gem provides a bonus to all three of the magical stats. There are three different kind of gems:

- Ruby (+7 strength, +2 agility, +5 vitality)
- Emerald (+1strength, +4 agility, +9 vitality)
- Amethyst (+2 strength, +8 agility, +4 vitality)

Every point of strength adds +2 to min damage and +3 to max damage. Every point of agility adds +1 to min damage and +4 to max damage. Vitality does not add damage.

Furthermore, every gem comes in different levels of clarity (basically level of quality). Depending on its level of clarity, a gem's stats can be modified in the following manner:

- Chipped (increases each stat by 1)
- Regular (increases each stat by 2)
- Perfect (increases each stat by 5)
- Flawless (increases each stat by 10)

So a Chipped Amethyst will have its stats modified like this: strength = 2 + 1, agility = 8 + 1, vitality = 4 + 1. Whilst a Perfect Emerald would look like this: strength = 1 + 5, agility = 4 + 5, vitality = 9 + 5.

Your job is to implement the functionality to read some weapons from the console and optionally to insert or remove gems at different socket indexes until you receive the END command.

Also, upon the **Print** command, in order to print correct final stats for a given weapon, first calculate the weapon's **base stats** taking into account **its type and rarity**. Afterwards, calculate the stats of each of its gems based on their **clarity** and finally add everything together. For the specific format of printing refer to the Output section.

Note

If you add gem on top of another, just overwrite it. If you add a gem to an invalid index, nothing happens. If you try to remove a gem from an empty socket or from invalid index, nothing happens. Upon receiving the END command print the weapons in order of their appearance in the format provided below.

Input

Each line consists of three types of commands in which the tokens are separated by ";".

Command types:

- Create;{weapon type};{weapon name}
- Add;{weapon name};{socket index};{gem type}
- Remove;{weapon name};{socket index}
- Print;{weapon name}

Output

Print weapons in the given format:

"{weapon's name}: {min damage}-{max damage} Damage, +{points} Strength, +{points} Agility, +{points} Vitality"

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

Examples

Input	Output
Create;Common Axe;Axe of Misfortune Add;Axe of Misfortune;0;Chipped Ruby Print;Axe of Misfortune END	Axe of Misfortune: 24-46 Damage, +8 Strength, +3 Agility, +6 Vitality
Create;Common Axe;Axe of Misfortune Add;Axe of Misfortune;0;Flawless Ruby Remove;Axe of Misfortune;0 Print;Axe of Misfortune END	Axe of Misfortune: 5-10 Damage, +0 Strength, +0 Agility, +0 Vitality

Problem 11. Create Custom Class Attribute

Create a custom attribute that can be applied to classes and can be accessed at runtime. The attribute type elements it should contain are author, revision, description and reviewers. Apply the attribute to the Weapon class you have created for the Inferno Infinity problem. Provide these **exact** values:

- author = "Pesho"
- revision = 3
- description = "Used for C# OOP Advanced Course - Enumerations and Attributes."
- reviewers = "Pesho", "Svetlio"

Implement additional commands for extracting different annotation values:

- Author - prints the author of the class
- Revision - prints the revision of the class
- Description - prints the class description
- Reviewers - prints the reviewers of the class

Examples

Input
Author Revision Description Reviewers END

Problem 12. **Refactoring - Bonus

Refactor your Inferno Infinity problem code according to all HQC standards.

- Think about the proper naming of all your variables, methods, classes and interfaces.
- Review all of your methods and make sure they are doing only one highly concrete thing.
- Review your class hierarchy and make sure you have no duplicating code.

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

- Consider making your classes less dependent of each other. If you have the **new** keyword anywhere inside the body of a non-factory or main class, think about how to remove it. Read about **dependency injection.**
- Consider adding independent classes for reading input and writing output.
- Create repository class that stores all weapon data.
- Create an engine, weapon creator and so on. Try using design patterns like command and factory.
- Make you classes **highly cohesive** and **loosely coupled**.

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us: