Ape Intelligence

General information

Only those who passed the challenge program Entry Test can participate in the Challenge Program.

Your submission for all the parts of the exercise should be in Java. All the challenge exercises must be done individually. Feel free to send an e-mail to course2ip90+CP@gmail.com if you have any questions.

Assignment description

The Earth is ruled by apes. Humanity is submitted and suffer (see here for the details.) However, there is a hope. Uprising is underway. We need to know what the enemy plans are. The ape's military command encodes its orders to the troops. Fortunately for us, the ape military intelligence is not very advanced. We deciphered their code.

Not wanting to waste time interpreting the ape military commands , it was decided we make a program that automatically interprets commands and transforms them to English. You have been chosen to perform this task. Do it well, or we might end up getting wiped out...

Your submission for all three parts is to be a single Java program. The first line of any input will contain either "Part 1", "Part 2" or "Part 3", which tells your program which part of the exercise the input is for.

Exercise

The input will consist of a sequence of integers between -1 and 10000. An integer value of -1 signifies the end of the input. Your job is to read this input and output the corresponding English text, depending on the format the sequence is in (which is different for the three parts). For example, the sequence "0 0 4 -1" should be translated to " Attack with all your mounted orangutans." in Part 1, and as " Build a wooden cage." in Part 2. Note that some sequences may result in orders that are nonsense just as English has sentences which are syntactically correct but make no sense at all, such as "colorless blue concepts snore intelligently". We don't think the apes will send such weird commands. But your program should be able to handle them as well.

You have learned about functions this week, and therefore this exercise will be all about them. Your job is to identify parts of the message and implement these in functions, and calling suitable other functions so that all these parts of the order are only implemented once. For example, on multiple occasions you will have to interpret an integer as a quantity, which means you should have a single function for that job, and call it whenever the language demands it. If you have multiple pieces of code doing the same thing, you will lose points, so take extra care in this respect. However, the goal of this exercise is not to remove any and all code duplication (to do this properly, you need to know more than you have been taught anyway), so don't start a crusade to remove all of this: just ensure that parts of the language that are the same are implemented only once, and try to cluster parts of the language that logically belong together into functions as well.

We strongly recommend you complete Part 1 before moving on to Part 2 and Part 3. Also, carefully read the *additional restriction* below. Furthermore, in order to understand following part(s), you need to have read the previous one(s).

Part of this week's challenge is to interpret the inputs in a single scan, that is, you never 'go back' in the input and look at a number that you've already looked at in the past. If you follow this, your code should become very clean and very simple. If you are tempted to look back in the input, you're doing it inefficiently.

To make this assignment more interesting, your code must follow an **additional restriction**. You are **not allowed to use loop statements** such as **'while' and 'for'** this week, as there is no need for them. If you have implemented Part 1, you will hopefully have partitioned the language into several functions that each handle some part of the language. If you do the exact same thing for this parts 2 and 3, this same approach will work without problems (unless your code has bugs): you can simply let a function call itself at the proper moment, which should be quite clear from the language definition. Java is perfectly happy if a function calls itself. However, **using recursion to implicitly implement a loop is not allowed either**.

Part 1: Apes attack!

The enemy have three kinds of commands they may give in battle: a command *to attack* with some troops ("resources"), *to search* for enemies (that's some of us!) or *to retreat*. The first integer of the input represents which command they give: a **0** means **attacking** with resources, a **1** means **searching** for humans and a **2** means an order to **retreat** to a certain place.

A **resource request** looks like "Attack with all your mounted orangutans." A resource request consists of two more integers: the first constitutes a quantity and the second the type of resources requested. If the quantity is 0, this means they want everything they have of that type.

The types of resources are:

[1] "orangutans",
[2] "chimps",
[3] "gorillas",
[4] "mounted chimps",
[5] "mounted orangutans".

A **search request** consists of one mandatory part and one optional part. The mandatory part looks like " Search West" or " Search 42 caves" and the optional part looks like " and look for male humans" and is inserted right before the period. The mandatory part either starts with a 0 or with a number greater than 0. If the number is 0, it is followed by a quantity and a type of a place or a direction to search.

If the starting number is a 0, it is a place, which are:

[0] "hills",
[1] "marshes",

[2] "caves",
[3] "woods"

If the starting number is greater than 0, it is a direction (without “the”!), which are:

[1] "North",
[2] "East",
[3] "West",
[4] "South",
[5] "South-East",
[6] "South-West",
[7] "North-East",
[8] "North-West"

The mandatory part may optionally be followed by an integer, which indicates searched objects. They are us:

[0] "humans",
[1] "human males",
[2] "human females",
[3] "human children",
[4] "**mutated humans**"

A **retreat request** consists of one mandatory part and one optional part. The mandatory part looks like "Retreat to Forbidden Zone.". The optional part looks like " and move 23 mounted chimps to South-West " and is inserted right before the period. The mandatory part consists of an integer which is the intended place to retreat:

[0] "Ape City",
[1] "Forbidden Zone",
[2] "Rocky Mountains"

The optional part consists of a quantity, a resource and a direction.

Examples

input:

```
Part 1

0 0 4 -1
```

output:

```
Attack with all your mounted chimps.
```

input:

```
Part 1
```

```
      1 0 42 2 2 -1
```

output:

```
      Search 42 caves and look for human females.
```

input:

```
      Part 1

      1 2 -1
```

output:

```
      Search East.
```

input:

```
      Part 1

      2 1 -1
```

output:

```
      Retreat to Forbidden Zone.
```

input:

```
      Part 1

      2 2 23 3 5 -1
```

output:

```
Retreat to Rocky Mountains and move 23 gorillas to South-East.
```

Part 2: Ape war logistics

The ape military command also gives orders to send different supplies to their troops. They make items such as steel helmets or wooden shields. They love to make containers putting one item inside another. This kind of order will specify what kind of a container they order to make and to what direction send it.

An item has a type:

0. **"wooden",**
1. **"steel",**
2. **"stone",**
3. **"cotton"**

An item can be:

0. **"cage",**

1. **"net",**
2. **"helmet",**
3. **"shield"**

A container specification starts with a type followed by an item name. It is then followed either by a 4, in which case the container contains nothing, or followed by a 5, in which case two more container specifications follow this 5, or by a container specification which is contained in the container.

For example, "0 0 4" is a container (a wooden cage),

and "0 0 5 1 1 4 2 2 4" is a container (a wooden cage containing a steel net and a stone helmet).

A request starts with a container specification, and is optionally followed by a destination. A destination is optionally followed by a destination. Hence, the request "0 0 4 1 2 -1" would translate to "Pack a wooden cage and send one to North and send one to East".

Examples

input:

```
Part 2

0 2 4 -1
```

output:

```
Pack a wooden helmet.
```

input:

```
Part 2

2 0 1 3 5 0 2 4 3 1 4 1 3 -1
```

output:

```
Pack a stone cage containing a steel shield containing a wooden helmet
and a cotton net and send one to North and send one to West.
```

Part 3. Ape fortifications

The apes also give orders to build fortifications. They build them as towers according to the specifications they send as orders. They want their towers to consist of 1 by 1 by 1 meter boxes and 2 by 2 by 2 meter boxes, stacked on top of each other so that the 1x1x1 boxes are at the top and the 2x2x2 boxes are at the bottom. These are then placed on a concrete base. For some reason (perhaps to scare us humans), they want the 1x1x1 boxes to have images of their troops painted on them, while the 2x2x2 boxes are not allowed to be painted.

A box specification is either just the number 4, which means that the top of the tower is reached (and is therefore not really a box at all), or the type of the item (see Part 2), followed by the box that is on top of this box, optionally followed by a resource name (see Part 1). If the box is followed by a resource name, it is a 1x1x1 box, and otherwise (except of course if the box is just the integer 4) it is a 2x2x2 box.

Upon a 2x2x2 box, either a 2x2x2 box or a 1x1x1 box may be placed, but upon a 1x1x1 box only a 1x1x1 will be placed. So, the request "0 1 2 3 4 1 2 -1" should be interpreted as:

"Build a concrete base on top of which is a 2x2x2 wooden cube on top of which is a 2x2x2 steel cube on top of which is a 1x1x1 stone cube on top of which is a 1x1x1 cotton cube with chimps painted on the sides with gorillas painted on the sides."

Examples

input:

```
    Part 3

    0 1 2 3 4 1 2 -1
```

output:

```
Build a concrete base on top of which is a 2x2x2 wooden cube on top of
which is a 2x2x2 steel cube on top of which is a 1x1x1 stone cube on
top of which is a 1x1x1 cotton cube with orangutans painted on the
sides with chimps painted on the sides.
```

input:

```
    Part 3

    4 -1
```

output:

```
    Build a concrete base.
```

Handing in the assignment

Submit into Canvas a single file named ApeIntelligence.java.