Bulls and Cows

Stage 1/7: Game log

Description

Let's start by reminding ourselves how the game goes. There are two players: the first writes a secret 4-digit code using different digits, and the second has to guess it. At each turn, the second player writes a 4-digit number that they think might be the correct answer. Then, the first player grades that answer using bulls and cows as a notation. If a digit in the given answer matches a digit and its position in the code, it's called a "bull." If a given digit appears in the code but its position doesn't match, then it's called a "cow." The first player reveals how many bulls and cows there are. The information is general; in other words, it isn't bound to any particular digit. For example:

The code is 4931.
The answer is 1234.
The grade is 1 bull and 2 cows.

Here, 3 is a bull, 1 and 4 are cows. If all the digits are bulls, the secret code has been guessed and the game ends. If not, the game continues and the second player tries again.
This may sound rather complicated, but don't worry, we will take it very gradually. In this stage, you will not even have to implement the gameplay: all you need to do now is output the text that could be in the game. In other words, you don't have to worry about handling any requests or processing data: your goal is to display an example text of the game.

Objectives

In this stage, your program should:

- Print a predefined game log that contains at least two turns.
- The output to be graded is a 4-digit code consisting of unique digits.
- The last line of your output contains a secret number.

Your program should not take any input data. You only have to print a game log similar to the examples below.

Examples

Example 1

```
The secret code is prepared: ****.

Turn 1. Answer:
1234
Grade: 1 cow.

Turn 2. Answer:
5678
Grade: 1 cow.

Turn 3. Answer:
9012
Grade: 1 bull and 1 cow.

Turn 4. Answer:
9087
Grade: 1 bull and 1 cow.

Turn 5. Answer:
1087
Grade: 1 cow.

Turn 6. Answer:
9205
Grade: 3 bulls.

Turn 7. Answer:
9305
Grade: 4 bulls.
Congrats! The secret code is 9305.
```

Example 2

```
The secret code is prepared: ****.

Turn 1. Answer:
1234
Grade: None.

Turn 2. Answer:
9876
Grade: 4 bulls.
Congrats! The secret code is 9876.
```

Stage 2/7: Grader

Description

Let's add some interactivity to our program. The program should create a 4-digit secret code, and the player should try to guess it on the first try. The program should give a grade to evaluate how accurate the player was.

As you remember, a correctly guessed digit is a cow, and if its position is also correct, then it is a bull. After the player tries to guess the secret code, the program should grade the attempt and finish the execution.

For example, if the secret code is `9305`, then:

1. The number `9305` contains 4 bulls and 0 cows since all 4 digits are correct and their positions are correct as well. It's the only number that can contain 4 bulls, so it's also the secret code itself.
2. The numbers `3059`, `3590`, `5930`, `5039` contain 0 bulls and 4 cows since all 4 digits are correct but their positions don't match the positions in the secret code.
3. The numbers `9306`, `9385`, `9505`, `1305` contain 3 bulls.
4. The numbers `9350`, `9035`, `5309`, `3905` contain 2 bulls and 2 cows.
5. The numbers `1293`, `5012`, `3512`, `5129` contain 0 bulls and 2 cows.
6. The numbers `1246`, `7184`, `4862`, `2718` contain 0 bulls and 0 cows.

Note that guesses can contain repetitive digits, so:

1. The number `1111` contains 0 bulls and 0 cows.
2. The number `9999` contains 1 bull and 3 cows.
3. The number `9955` contains 2 bulls and 2 cows.

Objectives

In this stage, your goal is to write the core part of the game: the grader.

1. Your program should take a 4-digit number as an input.
2. Use a predefined 4-digit code and grade the answer that was input. You can do it digit by digit.

The grade is considered correct if it contains number-and-word pairs (like `X bulls and Y cows`) that give the correct information. If the answer doesn't contain any bulls and

cows, you should output `None`.

Examples

The greater-than symbol followed by a space `> ` represents the user input. Note that it's not part of the input.

Example 1

```
> 1234
Grade: 1 cow(s). The secret code is 9305.
```

Example 2

```
> 9087
Grade: 1 bull(s) and 1 cow(s). The secret code is 9305.
```

Example 3

```
> 1267
Grade: None. The secret code is 9305.
```

Stage 3/7: Secret code

Description

Using a predefined secret code doesn't make a fun game. Let's implement the ability to generate a pseudo-random secret number of a given length. If the length is greater than 10, print a warning message and do not generate a number.
We suggest you use the following algorithm to generate the numbers:

```
long pseudoRandomNumber = System.nanoTime();
```

This code saves the nanoseconds elapsed since a certain time to the `pseudoRandomNumber` variable. We can assume that this is a random number. You can generate a secret code by iterating over the `pseudoRandomNumber` in the reverse order and adding unique digits. If the `pseudoRandomNumber` lacks the required number of

unique digits, call `System.nanoTime()` again and try to generate the secret code again until you get a satisfactory result.

You can use the `Character.getNumericValue(char a)` method to get an integer representation of a number!

Objective

In this stage, your program should generate a pseudo-random number of a given length with unique digits and print it. If the length is greater than 10, the program should print a message containing the word `Error`.

The secret code may contain any digits from 0 to 9 but only once. The secret code shouldn't start with a digit 0: for the first digit of the secret code, use digits from 1 to 9.

Don't delete your previous work, just move your code to a separate method. You will need it in the future stages.

Examples

The greater-than symbol followed by a space `> ` represents the user input. Note that it's not part of the input.

Example 1

```
> 5
The random secret number is 48379.
```

Example 2

```
> 4
The random secret number is 5213.
```

Example 3

```
> 11
Error: can't generate a secret number with a length of 11 because there aren't enough
 unique digits.
```

Stage 4/7: Game time!

Description

In this stage, you should combine all the previous parts into a simple playable version of the "Bulls and Cows" game. First, prompt the player to input the length of the secret code. The length will determine the difficulty level for the current game session. The program should generate a secret code of the given length. Remember that it should consist of unique numbers.

Then, the game starts and the program prompts the player to guess the code. When the player inputs a number, the program should grade it in bulls and cows. The game goes on until the code is guessed, that is, until the number of bulls is equal to the number of digits in the code. When the game ends, the program should finish its execution.

Objectives

In this stage, your program should:

1. Ask for the length of the secret code and then generate the code.
2. Wait for the user input.
3. Grade the guessing attempt in bulls and cows.
4. If the secret code has been guessed, the program stops; otherwise, return to the second step.

Example

The greater-than symbol followed by a space `> ` represents the user input. Note that it's not part of the input.

```
Please, enter the secret code's length:
> 4
Okay, let's start a game!
Turn 1:
> 1234
Grade: 1 bull and 1 cow
Turn 2:
> 7354
Grade: 2 bulls and 1 cow
Turn 3:
> 9374
Grade: 4 bulls
Congratulations! You guessed the secret code.
```

Stage 5/7: Improve the code generator

Description

The algorithm suggested for generating the secret code in the previous stage was really a “reinvention of the wheel”. Java already has the tools for generating random numbers! Research some common pseudo-random generation methods such as `Math.random()` and other methods from the `Random` class. Choose the method you like and use it to rewrite the secret code generation.

Nothing else is supposed to change at this stage: the program asks for the length, generates a secret code, and then receives and grades the attempts until the code is guessed. Your task here is to rewrite the code generator without breaking the existing code.

Objective

In this stage, rewrite the secret code generator using a suitable Java method.

Example

The greater-than symbol followed by a space `> ` represents the user input. Note that it's not part of the input.

```
Please, enter the secret code's length:
> 4
Okay, let's start a game!
Turn 1:
> 1234
Grade: 1 bull and 1 cow
Turn 2:
> 7354
Grade: 2 bulls and 1 cow
Turn 3:
> 9374
Grade: 4 bulls
Congratulations! You guessed the secret code.
```

Stage 6/7: New level

Description

Some players need a challenge, so let's make the secret code in the game harder to guess. Add support for more than 10 symbols by adding letters. Now, the secret code can contain the numbers `0-9` and the lowercase Latin characters `a-z`.

The new maximum length for the code is 36. Note that the length of the secret word may not match the number of possible characters in the secret code, so you should request input twice: once for the secret code length and once for the number of possible characters.

Also, since a secret code is not a number anymore, the symbol `0` should be allowed as the first character in a secret code.

Objectives

In this step, your program should:

1. Ask for the length of the secret code.
2. Ask for the range of possible characters in the secret code.
3. Generate a secret code using numbers and characters. This time, you should also print the secret code using `*` characters and print which characters were used to generate the secret code.
4. Function as a fully playable game.

Example

The greater-than symbol followed by a space `> ` represents the user input. Note that it's not part of the input.

```
Input the length of the secret code:
> 4
Input the number of possible symbols in the code:
> 16
The secret is prepared: **** (0-9, a-f).
Okay, let's start a game!
Turn 1:
> 1a34
Grade: 1 bull and 1 cow
Turn 2:
> b354
Grade: 2 bulls and 1 cow
Turn 3:
> 93b4
Grade: 4 bulls
Congratulations! You guessed the secret code.
```

Stage 7/7: Error!

Description

There are a lot of error possibilities. What if someone enters an answer of the wrong length? Or the number of possible symbols is less than the length of the code? What if the answer contains invalid symbols? The game may crash before the secret number was guessed!

Let's handle errors like this. At this point, you can implement this without the `try catch` construction. Use the following rule of thumb: if you can avoid the exception-based logic, avoid it! If you use exceptions in normal situations, how would you deal with unusual (exceptional) situations? Now it may not seem that important, but when you need to find errors in more complex programs, this makes a difference.

Objectives

In this stage, your program should:

1. Handle incorrect input.
2. Print an error message that contains the word `error`. After that, don't ask for the numbers again, just finish the program.

Examples

The greater-than symbol followed by a space `> ` represents the user input. Note that it's not part of the input.

Example 1

```
Input the length of the secret code:
> 6
Input the number of possible symbols in the code:
> 5
Error: it's not possible to generate a code with a length of 6 with 5 unique symbols.
```

Example 2

```
Input the length of the secret code:
> abc 0 -7
Error: "abc 0 -7" isn't a valid number.
```

Example 3

```
Input the length of the secret code:
> 6
```

```
Input the number of possible symbols in the code:
> 37
Error: maximum number of possible symbols in the code is 36 (0-9, a-z).
```

Example 4

```
Input the length of the secret code:
> 4
Input the number of possible symbols in the code:
> 12
The secret is prepared: **** (0-9, a-b).
Okay, let's start a game!
Turn 1:
> a234
Grade: 1 bull and 1 cow
Turn 2:
> 73b4
Grade: 2 bulls and 1 cow
Turn 3:
> 9374
Grade: 4 bulls
Congratulations! You guessed the secret code.
```