Exercise 00:

First, start by the smallest cog in this vast bureaucratic machine: the Bureaucrat.
A Bureaucrat must have:
• A constant name.
• And a grade that ranges from 1 (highest possible grade) to 150 (lowest possible grade).
Any attempt to instantiate a Bureaucrat using an invalid grade must throw an exception:
either a Bureaucrat::GradeTooHighException or a Bureaucrat::GradeTooLowException.
You will provide getters for both these attributes: getName() and getGrade(). Implement also two member functions to increment or decrement the bureaucrat grade. If the grade is out of range, both of them will throw the same exceptions as the constructor.
You will implement an overload of the insertion («) operator to print something like (without the angle brackets):
<name>, bureaucrat grade <grade>.
As usual, turn in some tests to prove everything works as expected.

Exercise 01:

Now that you have bureaucrats, let’s give them something to do. What better activity could there be than the one of filling out a stack of forms?
Then, let’s make a Form class. It has:
• A constant name.
• A boolean indicating whether it is signed (at construction, it’s not).
• A constant grade required to sign it.
• And a constant grade required to execute it.
All these attributes are private, not protected.
The grades of the Form follow the same rules that apply to the Bureaucrat. Thus, the following exceptions will be thrown if a form grade is out of bounds: Form::GradeTooHighException and Form::GradeTooLowException.
Same as before, write getters for all attributes and an overload of the insertion («) operator that prints all the form’s informations.
Add also a beSigned() member function to the Form that takes a Bureaucrat as parameter. It changes the form status to signed if the bureaucrat’s grade is high enough (higher or egal to the required one). Remember, grade 1 is higher than grade 2.
If the grade is too low, throw a Form::GradeTooLowException.
Lastly, add a signForm() member function to the Bureaucrat. If the form got signed, it will print something like:
<bureaucrat> signed <form>
Otherwise, it will print something like:
<bureaucrat> couldn’t sign <form> because <reason>.
Implement and turn in some tests to ensure everything works as expected.

Exercise 02:

Since you now have basic forms, it’s time to make a few more that actually do something.
In all cases, the base class Form must be an abstract class, and therefore should be renamed AForm. Keep in mind the form’s attributes need to remain private and that they are in the base class.
Add the following concrete classes:
• ShrubberyCreationForm: Required grades: sign 145, exec 137
Create a file <target>_shrubbery in the working directory, and writes ASCII trees inside it.
• RobotomyRequestForm: Required grades: sign 72, exec 45
Makes some drilling noises. Then, informs that <target> has been robotomized successfully 50% of the time. Otherwise, informs that the robotomy failed.
• PresidentialPardonForm: Required grades: sign 25, exec 5
Informs that <target> has been pardoned by Zaphod Beeblebrox.
All of them take only one parameter in their constructor: the target of the form. For example, "home" if you want to plant shrubbery at home.
Now, add the execute(Bureaucrat const & executor) const member function to the base form and implement a function to execute the form’s action of the concrete classes. You have to check that the form is signed and that the grade of the bureaucrat attempting to execute the form is high enough. Otherwise, throw an appropriate exception.
Whether you want to check the requirements in every concrete class or in the base class (then call another function to execute the form) is up to you. However, one way is prettier than the other one.
Lastly, add the executeForm(Form const & form) member function to the Bureaucrat. It must attempt to execute the form. If it’s successful, print something like:
<bureaucrat> executed <form>
If not, print an explicit error message.
Implement and turn in some tests to ensure everything works as expected.

Exercise 03:

Because filling out forms is annoying enough, it would be cruel to ask our bureaucrats to do this all day long. Fortunately, interns exist. In this exercise, you have to implement the Intern class. The intern has no name, no grade, no unique characteristics. The only thing the bureaucrats care about is that they do their job.
However, the intern has one important capacity: the makeForm() function. It takes two strings. The ﬁrst one is the name of a form and the second one is the target of the form. It return a pointer to a Form object (whose name is the one passed as parameter) whose target will be initialized to the second parameter.
It will print something like:
Intern creates <form>
If the form name passed as parameter doesn’t exist, print an explicit error message.
You must avoid unreadable and ugly solutions like using a if/elseif/else forest. This kind of things won’t be accepted during the evaluation process. You’re not in Piscine (pool) anymore. As usual, you have to test that everything works as expected.