Exceptions and Assertions

Overview

In this lab, you will take an existing application and add exception-handling capabilities to it. If time permits, you will also use assertions to unit-test the application functionality.

Source folders

Student project: `StudentExceptionsAssertions`
Solution project: `SolutionExceptionsAssertions`

Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Throwing and catching runtime exceptions
2. Defining a custom exception class
3. Throwing and catching checked exceptions
4. Using assertions

Familiarization

Open your student project and take a look at the code in `Employee.java`:

- Each employee has a (unique) ID, name, and salary.
- Each employee also has a "retired" flag; the system will keep track of employees even after they have retired.
- Employees can have a pay rise.

Now take a look at the code in `Company.java`:

- A company has a collection of employees, indexed by employee id.
- The company can hire, fire, and retire employees. The company doesn't forget its employees when they retire.
- The company allows employees to have a pay rise.
- The company allows you to get a list of all employees, or just those that are still working, or just those that have retired.

Finally take a look at the code in `UsingCompany.java`. This is the main entry point for the application, and displays a menu of options for the user. The code in this class is almost entirely complete.

Exercise 1: Throwing and catching runtime exceptions

In `Employee.java`, locate the `payRise()` method. Add code to throw runtime exceptions, as indicated by the TODO comments. Then in `UseCompany.java`, add `try/catch` logic to deal with these exceptions.

Run the application, and make sure all the options work as expected. For example, ensure the following behaviour is exhibited:

- You can't give an employee a negative pay rise, not even in the current economic climate!
- You can't give a retired employee a pay rise.
- You can give a non-retired employee a positive pay rise ☺.

Exercise 2: Defining a custom exception class

In `CompanyException.java`, define a custom exception class named `CompanyException`. The exception class should cater for the following information about a company-related problem:

- An error message
- The id of the employee that was being processed when the exception occurred
- An "inner exception" cause. This will enable a `CompanyException` object to encapsulate a lower-level technical exception object.

Run the application, and make sure all the options work as expected.

Exercise 3: Throwing and catching checked exceptions

In `Employee.java`, add code to throw `CompanyException` exceptions, as indicated by the TODO comments in the following methods:

- `hireEmployee()`
- `fireEmployee()`
- `retireEmployee()`
- `giveEmployeePayRise()`

Then modify `UsingCompany.java`, to `catch` these exceptions.

Exercise 4 (If time permits): Using assertions

Open `TestCompanyAndEmployee.java`, and add code to test the various capabilities of your `Employee` class. Hint: Call various methods, and use `assert` to ensure you get the expected results; the comments give you some suggestion.

Don't forget to enable assertions before you run the application - ask your instructor if you need help to do this.