ESS 201 Programming in Java
T1 2020-21
Lab 4
2 Sept 2020

Part A (to be worked on during the lab class)
These exercises are on the topic of inheritance. Complete the following exercises to familiarize yourself with different aspects of inheritance and definition of derived classes.These need not be submitted, but show your work to the TA before leaving the class.

1. *Composition vs Inheritance:*
   A circle is defined by its radius. A cylinder has a circular base (of a certain radius) and is defined by this radius and height. We can compute the volume and surface area of the solid cylinder from the area and circumference of its base circle.
   So, assume we have a class Circle as in the attached file Lab41.java. Fill in the methods here. Now define a class Cylinder whose outline is also given in that directory. We would like to reuse the Circle class in the implementation of Cylinder class. That is, the class Cylinder should not have code for directly computing the area/circumference of the base circle, but should instead invoke the appropriate methods of Circle.
   a. Implement using *composition*. That is, the class Cylinder should contain an instance of class Circle, and then uses this circle's methods to get its area/circumference
   b. Can this be implemented as an *inheritance* - Cylinder as a derived class of Circle? What would this look like and does it make sense?

2. *Upcasting and Downcasting*
   Consider the classes File, TextFile and ImageFile in file Lab42.java
   Fill in the code to get each of these classes to compile (and hopefully work!)
   Make changes to the main, one statement at a time, to ensure it compiles and runs without errors. Use casts only where needed. You may need to delete or comment out some lines to make this even compile.

3.

Part B (to be submitted in one week)

A hospital has a set of diagnostic labs such as Biochemistry, Pathology, Virology etc. Each lab does different kinds of tests, and their internal processes are different. However, to simplify things, they share a common Reception, Billing and Dispatch divisions.

Since Billing and Dispatch don't need to know about the specific type of lab for their work, we can model a base class Lab that has features and functionality common to all labs, and then specific sub-types such BioChemLab, PathLab, ViroLab etc

While the tests done by the labs are different, they have some features in common:

- a Patient ID (String) generated by Reception
- a Test ID (String) generated by the individual labs
- a Report (String) generated by the individual labs
- a Price (int) computed by the individual lab

Beyond this, each lab may add its own information for internal processing.

LabTest should never be called explicitly

To model this, we have a base class LabTest that abstracts the above information. We can create sub-types of LabTest such as PathTest, BioChemTest, VirologyTest, etc, which are all derived from LabTest. Each sub-type of lab should handle only a specific type of lab tests. That is, a PathLab processes only PathTest, BioChemLab processes only BioChemTest etc.
For now, we will model only 2 lab types and test types: PathLab and BioChemLab, and PathTest and BioChemTest.

The flow is as follows:

- The main plays the role of users/patients.
    - It reads the input, where each line specifies the test needed. For now, we will assume there is only one kind of test for each type of lab.
    - It send each test to Reception (using addRequest)
    - When all input requests are over, it gets the list of bills from Billing and prints out the total for the day
    - It then gets the list of all results from Dispatch and prints out each result on a separate line
    - It gets the list of tests from each lab and prints out the total tests per lab
- Reception:
    - Gets requests.
    - Generates a new ID for each test. These are sequential integers starting at 1 each day
    - Looks at the type (String - either Bio, Path) and creates an instance of an appropriate subtype of the test
    - sends it to billing for the information to be recorded

    - sends the test (subtype instance) to the appropriate lab (It should be able to get the reference to each type of lab from the Lab4B class through static methods)
- Billing
    - Is given tests to be billed
    - Keeps track of the tests that have been billed
- Dispatch
    - Is given tests to be sent to customers
    - Keeps track of tests that have been given to it
- Labs
    - Each subtype of lab generates its own TestID which is a string consisting of “B”, “P”,, depending on the BioChem, or Path lab, followed by a unique integer. Each lab starts with integer 1, and increments it for each new test. So, the Pathology IDs would be “P1”, “P2”, etc.

The file Lab4B.java explains the overall structure and key methods of all the classes involved. Modify and expand this as needed.

To keep the design modular, we want Reception, Billing, Dispatch, and base classes Lab and LabTests to be in one package, and the derived classes of Lab and LabTest in another package.

Note that to submit to Domjudge, you will need to place all the files in one level, so you can comment out the “package xxx;” and import statements at the beginning of your file before submission

To simplify the code, we will assume:

- all PathLab tests cost 1000, all BioChem tests cost 400
- PathLab results are the string “benign” or “malignant”.
- BioChem test results have a single number
- PathLab results are “benign”, “benign”, “malignant” for the every consecutive 3 tests, and this pattern repeats
- BioChem results are 100 for the 1st test, 101 for the second test, and so on

Input format:
1st line has an integer n, the number of input requests
Next n lines have a single word each - Bio or Path

Sample input
```
8
Bio
Bio
```

```
Path
Path
Path
Bio
Path
Bio
```

Sample output

```
Total billed: nnnnnnn
Reports:
<pid1> <testid1> <result 1>
…. one line for each test
Total tests:
BioChem: mmm
Path: dddd
```