420371 - Programming with Java EE

Academic Year 2022-2023 Semester 1

Programming Examination

Time Slot: 8.50-11.20am, 21 October 2022
Venue: Lab 430, Jishi Building, Tongji University

Notice:

1. There are three problems in this examination.
2. For each problem, compress all source code folders into one ZIP file, and submit it via the *Canvas* system. No binary file (such as .class/.jar file) or IDE-related file should be submitted.
3. When it reaches 11.20am, you must immediately stop your programming work and initiate the submission process. You will be given 15 minutes for uploading your code and solving technical issues encountered. The *Canvas* submission portal will be automatically closed at 11.35am.
4. When solving the problems, please also pay attention to the design, readability and maintainability of your source code.
5. Please write your matriculation number and full name (as comments) in the beginning of each source code file submitted.
6. It is not allowed to use any third-party class library. Use JDK classes only.
7. During grading, JDK 17.0.5 will be used for testing your code.

Problem A: Resume Filter

(25 Marks)

A company plans to recruit 20 software developers this year, but the HR department has received more than 23,300 resumes now. In order to save time on dealing with the resumes, the company requires you to develop a simple resume filter. Each resume contains the following fields:

Field	Type	Validation Rule
Name	String	Not null, not blank
Age	int	Older than 18
School of Graduation	String	Length must be larger than 4
Education Background	Enumeration (*Bachelor*, *Master* or *Doctor*)	Not null
Internship or Working Experience	boolean	True or false

You must firstly design and implement a ***Resume*** class, and then create an inner class named ***ResumeValidator*** to tell whether the resume is valid (i.e. all validation rules have been met).

As for the filter, your program must support at least three rules as follows:

a) A candidate whose resume is not valid will be rejected;

b) A candidate who is older than 35 and has no internship or working experience will be rejected;

c) If a candidate's school of graduation does not contain the string "*Tongji*" and the education background is not "*Doctor*", he/she will be rejected randomly (with a possibility of 50%).

Implement the filter with a set of reasonably designed classes. You must also consider the potential extension of more rules in the future. In the main method, use an appropriate array/collection to hold some instances of resumes, and you may initialize it with arbitrarily created test cases. Call the filter and output the information of candidates who are not rejected after filtering.

Problem B: Tax Calculation Engine

(35 Marks)

Tax calculation plays an essential role in product sales transactions. To simplify the scenario, each transaction contains only one product, as well as one or several tax codes applied. Please read the following concepts in tax calculation:

- **Tax Code**: the core of tax calculation, which indicates a tax calculation rule represented as a combination of tax rate and tax calculation origin.
- **Tax Rate**: the percentage at which a transaction is taxed.
- **Tax Calculation Origin**: the base amount for tax calculation. The program must support at least two types of origins, with consideration on potential extension in the future:
  - **By net amount**: the tax amount is calculated as a percentage of the origin amount (i.e. the net amount);
  - **By gross amount**: the tax amount is calculated as a percentage of the gross amount, which is the transaction's net amount plus all tax amounts without applying the tax code by gross amount. Each transaction may have at most one tax code by gross amount.

Examples of a transaction's tax codes are illustrated as below:

Tax Code	Tax Rate	Tax Calculation Origin
SalesTax1	10%	By net amount
SalesTax2	5%	By net amount
SalesSP1	2%	By gross amount

The following table demonstrates two examples on how tax calculation is performed according to

the tax codes applied.

Product Sales Transaction	**Net Amount**	**Tax Codes Applied**	**Calculation of Tax Amount**
PlayStation 5 (Ultra HD Blu-ray)	2000	SalesTax1, SalesTax2	2000 * 10% + 2000 * 5% = 300
Estee Lauder Advanced Night Repair Eye Synchronized Complex	1000	SalesTax1, SalesSP1	1000 * 10% + (1000 + 1000 * 10%) * 2% = 122

Design a ***TaxCode*** class and a ***Transaction*** class, as well as other necessary classes, for implementing the above concepts. The following functionalities must be supported:

- User-friendly constructor(s) for the TaxCode class
- At least two user-friendly constructors for the Transaction class
- toString(): outputting important properties of a transaction
- calcTax(): returning the tax amount for a single transaction, which is defined in an interface named ***Taxable*** and implemented in the Transaction class

In the main method, use an appropriate array/collection to hold some instances and test the above functionalities.

Problem C: Auction Simulation

(40 Marks)

Design and implement a Java program to simulate the auction process. Your program firstly reads a text file named ***auction-items.txt*** to obtain the names and starting prices of a list of items to be auctioned, and then start to auction the items one by one. During the auction process for each item, several threads are created representing a number of bidders. Each bidder is initialized with an expected maximum price and a bid increment value (both can be initialized randomly). The auction proceeds as follows:

- For each bidder, if the sum of the current price and the bid increment value is less than or equal to the expected maximum price, bid and ask the auctioneer to update the current price as the sum, and then sleep for 300-2,000ms randomly; otherwise, output "*Bidder [Bidder ID] terminates*" (e.g. "*Bidder 1 terminates*") and exit.
- The auctioneer continuously controls the auction process. If there is only one active bidder left, treat him/her as the winner. If all bidders have exited, treat the last one who bids as the winner. The auctioneer outputs the final price and name of the winner once determined.

Each line of the auction-items.txt file contains the name (String) and the starting price (float) of a single item to be auctioned, separated by '#'. An example is illustrated as follows:

```
Diamond Crown#3500.00
Sunflowers by Vincent van Gogh#10000.00
Gundam RX-78 MG#980.00
Nendoroid Eyjafjalla GSC2021#120.00
Book-Thinking in Java#65.00
```

Implement a public class to start the auction. Use a thread pool to manage all bidders, and use concurrency programming techniques (e.g. *synchronized*, *AtomicInteger*, *Lock*, etc.) to ensure thread safety when the auctioneer is being invoked by several bidders. All parameters except for the name and starting price of each item can be generated randomly when running.

[THE END]