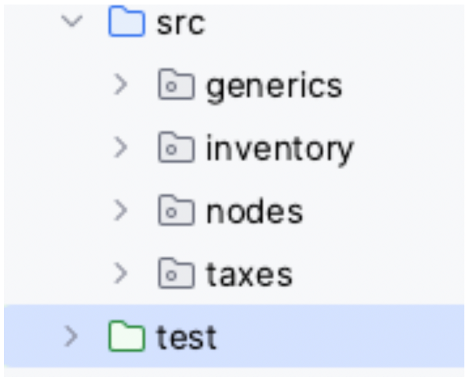
CS5004 2024 Fall MIDTERM

Remember: Your code must be zipped and follow the following structure:



<u>General Notes:</u>

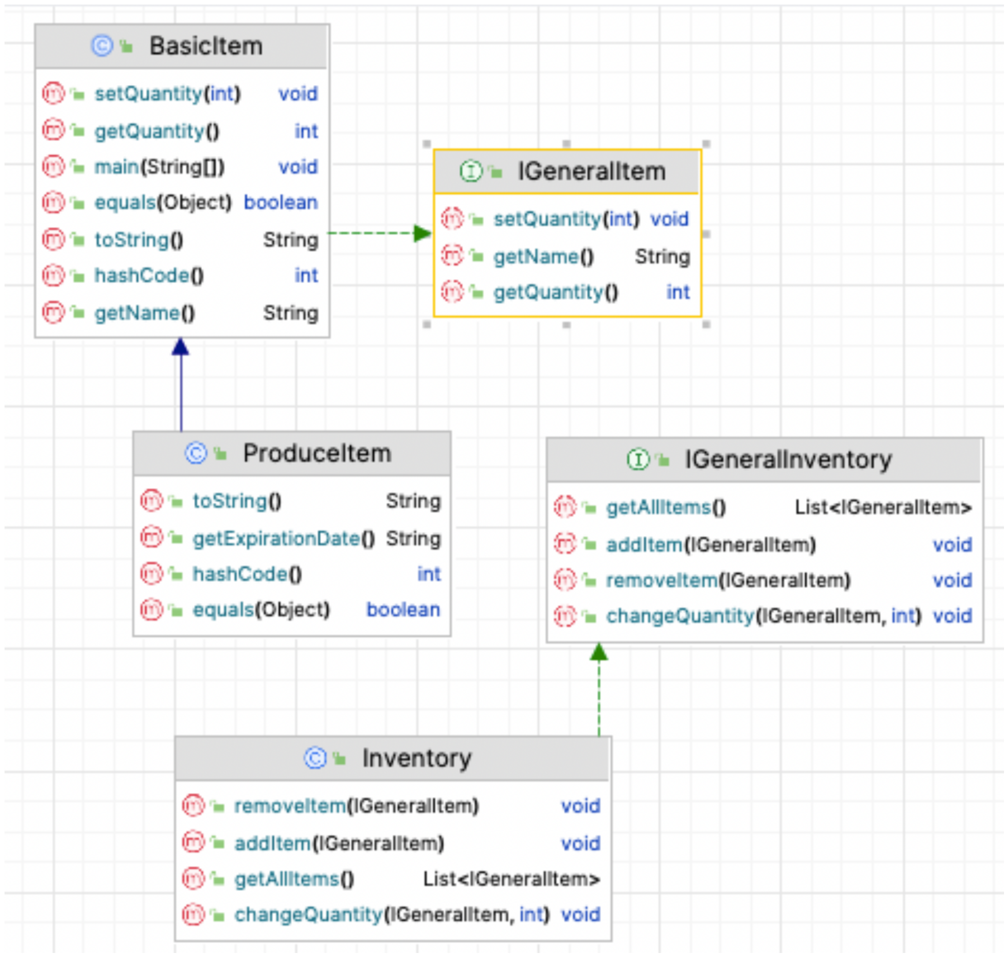
I have provided several "starter interfaces" and some existing "starter code" for implementation. Do not modify, extend, or otherwise alter these interfaces or the code provided.

Write your midterm solution code using the interfaces provided. Write your solution code to pass the JUnit tests provided. If ever in doubt, the JUnit tests provided are the final arbiter and have the final word of what is expected with respect to solution behavior. Barring a formal announcement stating otherwise, the unit tests are the "Supreme Court" of this exam. Any confusion that may arise from the provided English specifications is settled via the Junit provided. **Again: Write your code to pass the tests. Full stop.**

When you are required to write JUnit tests to submit, the question will explicitly tell you to do so. If not explicitly directed to do so, you are not required to write JUnit tests for your solutions.

Question 1: Inventory System (20 points)

Note: Implement the following solution in Java. The UML class diagram gives a high-level overview; More specifications follow. This class diagram was generated by IntelliJ and has some mistakes in notation (the basic elements are correct, however, so you can use this as a visual guide). You will be asked to identify the mistakes IntelliJ made with its UML syntax below.



Place your solution in a package named inventory.

I have provided the `IGeneralItem` and `IGeneral` Interfaces for your use and "code centric" documentation.

We are building a portion of a simple inventory management system. Our system manages two different types of items: `BasicItem` and `ProduceItem`. Both types of items implement the IGeneralItem interface and have a name (String) and a quantity (integer). Additionally, Produceitems maintain an expiration date (String). There is no specific format for the expiration date, but the constraint is that the String cannot be null, and it cannot be an empty string. Throw an `IllegalArgumentException` if either of these cases occurs.

BasicItem and ProduceItem are concrete classes – our system can instantiate objects of either type. ProduceItem is-a-kind-of BasicItem.

See the IGeneralItem interface as it lists all features a BasicItem, supports. In addition, the constructor for BasicItems take this form:
`public BasicItem(String name, int quantity)`

The constructor for the ProduceItem takes the form:
`public ProduceItem(String name, int quantity, String expirationDate)`

Additionally, you should plan to create the typical “boilerplate” methods: toString, equals(), hashCode() and a copy constructor for both classes. The expected return for toString is of the form: `ProductName:Quantity` for General Items and, `ProductName:Quantity:ExpDate` for ProduceItems.

Remember: The constructors and setQuantity() methods throw an `IllegalArgumentException` if the quantity is less than zero.

ProduceItem adds this behavior, not included in the GeneralItem interface:
`public String getExpirationDate()`

Inventory implements the IGeneralInventory interface. You should provide at least the “default (no parameter) constructor for this class. See the interface for specifics, but take note of these important details:

```
public Inventory() // creates an empty inventory
public List<GenericItem> getAllItems() // return unmodifiable list
…
public void changeQuantity(GenericItem item, int newQuantity ) //
updates item quantity. Defensively “clamp” values and ignore // any
request
// for quantities < 0 (less than zero)
```

* For BasicItems, “equality” or “a match” are items that simply match in name (caseinsensitive). For ProduceItems, “equal” items must match in name and expiration date. Your equals() method for both classes must be able to handle anything that is-a-kind of IGeneralItem. If the two classes being compared are not the same, the rules for the BasicItem apply (simply match by name alone).

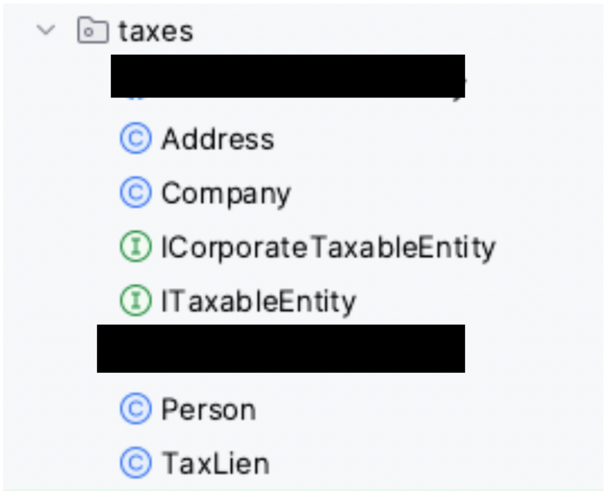
* getAllItems() returns an unmodifiable list. You may wish to check the Java documentation (or Google) to ensure you return a list that cannot be modified.

(b) Written: What are the issues/errors with the IntelliJ-generated diagram I’ve included (there are at least 2 syntax problems)? In your submission, include a text file named “Written.txt” and provide no more than 1-2 sentences describing the errors.

Question 2: Taxes (20 points)

This question is based on some of the code we developed in lecture a few weeks ago. There are some minor updates for this exam, so be sure you use the code included for the exam, rather than the partially finished code we created in lecture.

Place your solution assets in a package named taxes:



taxes
[redacted]
Address
Company
ICorporateTaxableEntity
ITaxableEntity
[redacted]
Person
TaxLien

For this question, **develop the** `Person` **and** `Company` **classes.**
Write your code to pass the JUnit tests provided!

Person is-a-kind-of ITaxableEntity and Company is-a-kind-of ICorporateTaxableEntity.

Taxable Entities have one or more Addresses. Each address has a TaxLien. Tax Liens may be paid in either full or half-year increments. A Person may have a mixture of payment plans for their addresses. For example, George Jetson may own 3 properties and pay 2 of them in full-year increments, and 1 in half-year increments.

Review the provided interface specifications for the other specifics.

You must provide the following constructors for the concrete classes:

```
public Person(String name, String taxID, Address address, int dependents)
public Company(String name, String taxID, Address address)
```

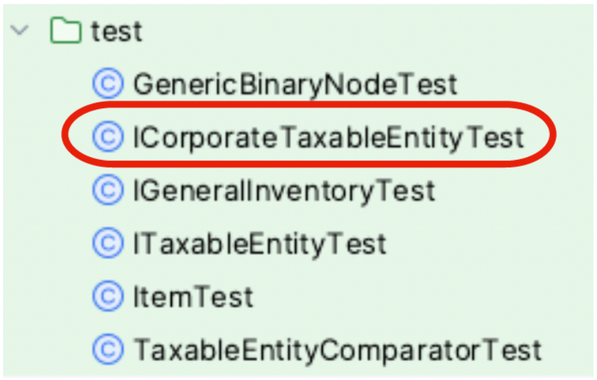
Note for EQUALITY: any taxable entity (of either type provided to you) is equal-to another taxable entity (of either type provided to you) if their name is the same (case-insensitive) and their taxID is the same (case-insensitive).

Note Variations in tax calculation for sub-types:

** For PERSONS and other Taxable entities that are not CorporateTaxableEntity: There is the*
** concept of DEPENDENTS. A non-corporate taxable entity may have a maximum of 10 dependents*
** (minimum 0). Non-corporate taxable entities have a reduced tax liability by dividing the*
** total tax liability for the period by the number of dependents (zero dependents is treated as 1).*
** For example: if George Jetson owes 100 and has 4 dependents, his liability is 25. If George has*
** 2 dependents, his liability is 50. If George has zero or 1 dependent, his liability is 100.*
***
** For COMPANIES and other CorporateTaxableEntities, there are no such deductions so the total*
** tax liability is never altered from the raw summation of taxes from addresses.*

Question 3: Tax JUnit (20 points)

Note: For this question, create your test assets in your test directory, in the **default package**. Do NOT place your tests in the taxes package.



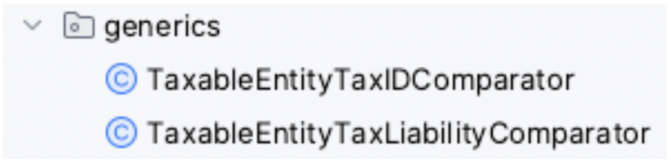
Create JUnit5 tests to validate solutions that implement the ICorporateTaxableEntity interface. Your test class MUST be named `ICorporateTaxableEntityTest` (as shown above).

You've written the concrete class for Company, so your tests should validate your solution for that. We've also created a "buggy" version and a (we think???) correct version of that class too. We'll run your JUnit against both of our solutions.

Your score for this question will be based on your tests validating your own code AND our solution code (both the buggy and correct versions).

Question 4: Generic Comparators 10 points

Note: For this question, create your solution assets in a **package** called `generics`

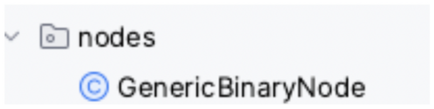

generics
TaxableEntityTaxIDComparator
TaxableEntityTaxLiabilityComparator

Create two generic comparators that allow us to sort Taxable Entities in ascending order by value. The first comparator: `TaxableEntityTaxIDComparator` sorts by comparing the entity's taxID values.

The second comoparator: `TaxableEntityTaxLiabilityComparator` sorts by comparing the entity's current tax liability value.

Question 5: Nodes 15 points

For this question, place your solution in a package named nodes.



Create a generic class named GenericBinaryNode, which can hold ANY kind of data (T). Binary Nodes also form the basis of Binary Trees. For this question, we are NOT asking you to create a binary tree. Rather, you are creating the node class that can be used as the basis for a binary tree.

As such, your node should be able to hold a data element of any type. Your node should also be able to get and set values for its left and right children.

You must provide the following constructors:

```
public GenericBinaryNode(T data)
public GenericBinaryNode(T data, GenericBinaryNode<T> left,
GenericBinaryNode<T> right)
```

And you must provide the following methods:

```
public void addLeft(GenericBinaryNode<T> node)
public void addRight(GenericBinaryNode<T> node)
public GenericBinaryNode<T> getLeft()
public GenericBinaryNode<T> getRight()
```

Also provide a toString() method. No other "boilerplate" methods are required for this question. This method must return the node data as a String, and delegate to its children to do the same (effectively returning a string containing the entire node structure). For example, if I create a root node with data 1 and then add a left node with 2 and a right node with 3 to that root, my toString() from root will include "1, 2, 3" (or some variation of that e.g. "2,1,3")

You have flexibility to choose *any* ordering you want for this method, and *you are* free to embellish the string as long as the unit test can identify that you have the data in your string results.

For example, when I create a node with data 5 and a child node with data 6 connected to it, my toString() returns
Node with data: 5
--> Node with data: 6

Write your code to pass the tests provided.

Question 6: UML 15 points

Note: For this UML question, you may hand-draw your solution and take a picture or use a drawing program like Lucidchart. Using IntelliJ to "reverse engineer" code for your diagram is disallowed and will result in a zero (0) for this question.

For the **UML class diagram, you are required to use proper UML notation** (UML class boxes with compartments, etc.) Include multiplicity and role names for full credit.

For the object diagram (snapshots) you are required to use proper UML object diagram notation (UML instance boxes).

Ensure that you remember to include your diagrams (png, jpeg, pdf) in your exam submission. Place it in your .zip file before you submit!

(a) [5 pts] Design a well-structured UML class diagram for the following relationships:

A Person plays the role of an employee and can work for only one Company at a time; a Person may be unemployed and not work for a Company. Conversely, a Company has at least one Person as an employee. A Company may have an unlimited number of Persons who are employees.

A Person plays the role of a stockholder. Persons who are stockholders can hold stock in one or more Companies. Conversely, a Company may or may not have stockholders. For our worldview, if a Company has stockholders, it can have an unlimited number of stockholders.

(b) [10 pts} Draw a set of object snapshots for the following four scenarios. Along with each drawing, indicate "Yes" if the diagram conforms to the rules established by the class diagram in part (a) of this question, or indicate "No" if it does not.
To earn full credit for this part of the question, you must draw the object diagram AND include either "yes" or "no" to indicate if the situation drawn adheres to the class diagram.

1. Herc is a Person. IBM is a Company. There is no relation between Herc and IBM.
2. Herc is a Person. NU is a Company. Herc is employed by NU (Herc is an employee of NU).
3. Herc is a Person. John is a Person. NU is a Company. IBM is a Company. Herc is employed by NU (Herc is an employee of NU). Herc is a stockholder of IBM. John is an employee of IBM. John is a stockholder of IBM.
4. Herc is a Person. IBM is a Company. NU is a Company. Disney is a Company. Herc is employed by NU. Herc is employed by IBM. Herc is a stockholder for Disney.