Specifications

Problem Description

Given the attached class named `Item` with fields for holding the item's name and description, extend `Item` with a class named `SaleItem`. The `SaleItem` class should have `double` fields for price and shipping cost, and an `int` field for the number of items in stock. Write a copy constructor and a constructor that takes data to instantiate all fields. Write a `boolean` method `purchase` that takes a quantity as a parameter and returns `true` if the purchase can be completed (if there's enough stock), and `false` otherwise. The `purchase` method should appropriately deduct from the field that holds the number of items in stock. Write a `restock` method that takes a quantity and adds it to the stock. Write accessors for price and shipping cost, and write a `toString()` method for `SaleItem`. Write exactly the methods described here, and no others. Include exactly the fields described here, and no others.

A `SaleItem` object's `toString` should produce a string that looks like this example:

```
Lovely shirt
Blue shirt that is lovely.
Price:     $9.99
Shipping:  $4.95
```

Write another class called `AuctionItem` that extends the `Item` class. `AuctionItem` should have one constant, accessible from outside the class – `NONE`, of type String, representing the high bidder when there are no bids in the auction. It should have three private fields: one for the account name of the high bidder (a `String`) and one for the high bid (a `double`). It should also have a field for the ending time of the auction, which will be an instance of the `Instant` class. (See the appendix at the end of this assignment for help with the `Instant` class.)

The `AuctionItem` class should have two constructors – a copy constructor and an argument constructor. In the copy constructor, all fields should be copied from the object passed in. In the argument constructor, the arguments passed in will be those needed to instantiate the superclass object and one other -- an integer `duration`, which is the number of hours that the auction should last. The high bid field will be initialized to 0, the high bidder field will be initialized to `NONE`, and the ending time field will be initialized to a time that is `duration` hours from the current time.

The `AuctionItem` class should have a method `isActive`, which returns `true` if the auction is still active and `false` otherwise. An auction is active if it hasn't yet ended. Whether it has ended depends on the current time and the ending time.

It should have a `bid` method, which takes an `amount` (`double`) and the `bidder` (`String`) as parameters and returns a `boolean` – `true` if the bid was successful (auction still active and bid higher than high bid), and `false` otherwise.

It should have two accessors – `getTimeLeft`, which returns the amount of time left in seconds, and `getHighBidder`. It should have a `toString` method.

An `AuctionItem` object's `toString` should produce a `String` that looks like this example:

```
Disney pin
Cute stylized Pluto pin!
High bid:       $1.00
High Bidder:    Princess4Evar
Time Left: 23 hours, 59 minutes
Ends at:        2018-01-14T00:22:58.662903800Z
```

Focus on inheritance first and getting the parts of the class that use the `Instant` object working second, after you're confident you've written a correct inheritance hierarchy.

You've been provided with an `ItemClient` java program that should work with your classes, if your classes are implemented to specifications. The output of this program should be self explanatory. Do not edit this program.

Please create a class diagram showing the three classes and the inheritance relationship.

Design constraints

Use only techniques that are covered in chapters 1-8 and 10 of Gaddis/Muganda or covered in this course. Output to the console should be from the `ItemClient` program only (your classes should not interact with the end user in any way). Do not, under any circumstances, use System.exit, or any other code (e.g. return, break) that artificially breaks out of a logically controlled block of code (conditional statement or loop). Do not submit code that includes these constructs. Submit whatever you can write that does not include these constructs.

Turn in

Please turn in all four java files (`Item`, `ItemClient`, `SaleItem`, and `AuctionItem`), zipped together with the class diagram in a folder. (Please do not submit your entire Eclipse project.) Use the zip compression format. Please remove all package specifications from the top of your files.

Please include, in a Javadoc comment at the top of the `SaleItem.java` file, a description of any errors in your program.

Rubric

An exceptional-quality assignment will meet the following standards:

- Program submitted according to instructions, 10 points
  The assigned was submitted under the correct assignments link in Blackboard and packaged according to the *Turn In* section above.
- Meeting functional and design specifications, 75 points
  The JUnit test cases run successfully. The classes are designed according to specifications. The programmer has used only the allowed programming techniques. If the program misses specifications or does not function correctly, errors are acknowledged in the comment at the top of the program.

- Communicating with identifiers and white space, 5 points
  The program makes appropriate use of variables. Variables, constants, classes, and methods are named according to convention and are named for understandability and purpose. White space, both vertical and horizontal, is correctly used for readability and meets programming conventions.
- Communicating through documentation, 10 points
  The Java program contains comments including the programmer's name and date. There are block comments (as many as necessary) for each distinct block of code which accurately describe what the block is accomplishing by relating the code to the problem being solved. Javadoc is included and meets the javadoc standards. (5 points) The class diagram is a UML diagram that correctly shows the inheritance relationship and accurately shows all fields, methods, parameters, return types, and visibility of the classes (5 points).

Peer evaluations will work the same as in the weekly labs, except you are grading on four criteria for a total of 100 points. If there are errors that the programmer acknowledges in the comment at the top of the `SaleItem.java` file, you may take off fewer points than if you found the error and they didn't acknowledge it.

```java
/**
 * Item is an abstract base class for all items for sale.
 *
 *@author Maggie
**/

public class Item {

	//Fields:
	private String name;					//the name of the item
	private String description;				//the person's address

	//Methods:
	//CONSTRUCTORS
	/**
	 * Item, constructor that takes arguments for all fields
	 * @param name -- String, name of the item
	 * @param description -- String, description of the item
	 */
	public Item(String name, String description){
		this.name = name;
		this.description = description;
	}

	/**
	 * Item, copy constructor
	 * @param toCopy -- an Item object to copy into the current object
	 */
	public Item(Item toCopy){
		this.name = toCopy.name;
		this.description = toCopy.description;
	}

	//ACCESSORS

	/**
	 * getName
	 * @return name of the item, String
	 */
	public String getName(){
		return name;
	}

	/**
	 * getDescription
	 * @return description of the item, String
	 */
	public String getDescription(){
		return description;
	}

	/**
	 * toString
	 * @return a String representing the data of the object.
	 */
	public String toString(){
		return name + "\n" + description;
	}
}
```

Appendix: The Instant class

The Oracle documentation on `Instant` can be found here:
https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html

The `Instant` class is in the `time` package. You can therefore use either one of the following import statements at the top of your program:

```
import java.time.*
import java.time.Instant
```

Within your program, assign `Instant.now()` to a variable of type `Instant` (e.g. `Instant timestamp = Instant.now();`)

You can convert an `Instant` to the number of seconds since the Java Epoch:

> `long getEpochSecond()`
> Gets the number of seconds from the Java epoch of 1970-01-01T00:00:00Z.

And you can add and subtract:

> `Instant minusSeconds(long secondsToSubtract)`
> Returns a copy of this instant with the specified duration in seconds subtracted.
>
> `Instant plusSeconds(long secondsToAdd)`
> Returns a copy of this instant with the specified duration in seconds added.

When converted to a `String`, you will get a date-time string such as "`2017-04-19T10:36:45.176Z`" which represents the time that the object was created. You can invoke toString on the object directly to obtain a String:

> `String toString()`
> A string representation of this instant using ISO-8601 representation.

I recommend you practice with the Instant class, using a simple program such as the following. Make changes to it and predict the results until you're certain you know how to use the class and the methods shown here.

```
import java.time.Instant;
import java.util.Scanner;
/**
    Instant practice
*/
public class InstantPractice {
    public static void main(String[] args) {
           // Create an instant
        Instant now = Instant.now();

           // Delay a bit -- press enter after
           // a number of seconds has passed
           Scanner keyboard = new Scanner (System.in);
           System.out.print("Enter your name: ");
           String delay = keyboard.nextLine();
```

```
            // Make another instant, now that you've delayed
            Instant later = Instant.now();

            // Print the two times
            System.out.println(now.toString());
            System.out.println(later.toString());

            // Subtract the first time (as seconds)
            // from the second time to get the difference
            long nowSeconds = now.getEpochSecond();
            Instant difference = later.minusSeconds(nowSeconds);
            System.out.println(difference.getEpochSecond());

    }
}
```

You might also want to use the Duration class to determine how much time is between two Instant objects. Information about how to do that can be found in this tutorial on Oracle's website:

https://docs.oracle.com/javase/tutorial/datetime/iso/period.html

To use Duration, import as follows:

```
import java.time.Duration;
```

Assignment 2 Rubric

Component		Quality		
		Exceptional	**Acceptable**	**Amateur**
	Run-time specifications 50%	50 pts: The program meets all of the run-time specifications, with no additional unspecified functionality.*	40 pts: There is additional unspecified functionality or the program produces incorrect results in no more than 5% of the customer's tests.	25 pts: The program produces incorrect results in no more than 10% of the customer's tests.
	Design specifications 25%	25 pts: The program meets design specifications. No fields or methods are added or removed. Hierarchy is correct.	20 pts: Classes mostly meet specifications but are off by less than 10% (e.g. additional or missing field or method, incorrect type field or method).	12 pts: Classes mostly meet specifications but ar off by less than 20%.
	Diagramming techniques 12.5%	12.5 pts: Diagramming techniques are used appropriately and accurately reflect the static state of the program.	10 pts: Diagramming techniques are used appropriately but with minor errors that don't affect readability and accurately reflect the static state of the program or have minor errors that don't affect comprehension of the system state.	6.25 pts: Diagramming techniques are used appropriately but with several errors that somewhat affect readability or have minor errors that affect comprehension of the system state in less than 20% of the execution portrayed.
	Documentation 12.5%	12.5 pts: The program contains comments including the programmer's name and date. Javadoc comments are included as shown in the text for all classes. There are block comments (as many as necessary) for each distinct block of code which accurately describe what the block is accomplishing.	10 pts: The header comment is incomplete but contains name and date, and/or the block comment(s) aren't clear. Javadoc comments are missing components less than 10% of the time.	6.25 pts: The documentation partially meets the exceptional guidelines with several poorly written comments and/or missing comments or missing components less than 25% of the time.

*If you want to change the functional specifications of the program in any way, you must clear it with your customer (the instructor) in writing prior to making the changes. Include documentation of the specification changes when you submit your program.