Project 2: Stanley's Storage, Part 2

1 Task

Stanley Storage is remodeling and reconfiguring their storage locations to meet customer demands. And they are introducing a new pricing model for all storage units. Update the previous work to reflect these changes.

2 Changes

Storage locations offer the same type of storage units, but they are configured differently. The first seven rows are standard units, with ten units in each row; the next three, humidity-controlled units, eight in a row; the next two, temperature-controlled units, six per row.

Pricing has changed significantly. Each storage location has established a base unit price, based on demand at that location (a price that may later change). Standard units add an additional flat rate of $75 on top of the base price. Humidity-controlled units add $5.00 per square foot of floor space in the unit. Temperature-controlled units add $1.00 per cubic foot of space in the unit.

There are now customer-selectable options for special unit types. These options may also affect pricing.

- For humidity-controlled units, customers specify the humidity level, from 20 to 60% (whole numbers). Customers who want humidity in the 20-29% range pay an extra $20 a month.
- For temperature-controlled units, temperatures can be similarly specified, in the range 45° to 70° (whole numbers). Customers who want temperatures in the 45-49° or 65-70° range pay an extra $30 a month.

Also, multi-unit renters get a 5% discount on their monthly rent. This doesn't change their *rental price*; it only changes the amount they get charged at *billing* time. Each discounted charge is rounded to the nearest penny.

Because of these pricing changes, no rental lock-in prices are offered; standard prices are *the* prices.

3 Design Documentation

Update your design documentation and obtain feedback before proceeding. After receiving feedback, incorporate that feedback into the design. Use design documents during the rest of the development process, then submit them along with your final project.

4 Code Implementation

4.1 Additional Building Blocks You'll Need

- Inheritance
- Polymorphism
- Abstract classes
- Abstract methods
- Arrays: 2D Jagged

4.2 Requirements

In addition to what was stipulated in Stanley's Part 1, here are some additional requirements:

- Remove the enumerated type. The temptation to use it is usually poor OOD.
- Use a single 2D array to hold *all* units. The type of the subarrays is up to you (think).
- Ensure that when unit details are displayed, each unit describes itself fully, e.g., it should say it's a humidity-controlled unit. It should show the price, as well, along with all other pertinent information (including customer name). Each unit's output should fit on one attractively-formatted line.
- When using inheritance, push as much work to superclasses as possible; subclasses should be relatively small if you design cleverly and implement correctly.
- Do not make superclasses aware of specific subclasses; that goes against good OOD. That means that you should not have any logic in a superclass that says, "If the subclass is of type *x*, then do this."
- If you choose to use getClass() or instanceof, use them sparingly and carefully; *never* use it in cases where inheritance and polymorphism would solve the problem; that's poor OOD.
- Storage Location's method that retrieves units of a specific type will need some thought; bring your ideas for us to discuss.

4.3 Additional Requirements

Important: there will be additional requirements that we'll determine together after you have some time to ponder potential design approaches to this project. These are a required part of your design, even though they aren't written in this document. If you miss this discussion, please follow up.

4.4 Style

Follow the Course Style Guide. You'll lose points on every assignment if you fail to do so.

5 Testing

Update tests, creating new test classes and updating the existing ones as needed. You can't instantiate abstract classes to test them; you'll need to test methods in derived *concrete* classes, instead.

6 Hints

- Copy and paste the entire Proj01 folder and make changes; don't "inherit" classes to create Proj02.
- A complete object diagram will be of great help when navigating project drill-downs.
- Before starting this project, incorporate feedback from the previous project; you should start building on a firm foundation.
- Most classes will be affected by these changes. Test methods will need tweaking, as well.
- Some methods will need to be tweaked to accommodate these changes; carefully review your methods, esp. set methods which may now need to be removed to present client code from setting arbitrary values when they are fixed for a certain type of unit. Some constructors may change, too, either gaining or losing parameters.

7 Extra Credit

7.1 Unit Map

Write a method to generate a unit map (a string) for the storage location. Use a one-letter type designation, followed by underscores for empty units. Follow this by an asterisk for rented standard units. For humidity and temperature units, show the selected setting instead of the asterisk, on rented units. The resulting string might contain something like this:

```
-------------------------------------------------------
            Unit Map for Location WA23Issaquah
-------------------------------------------------------

     0    1    2    3    4    5    6    7    8    9

00:  S*   S__  S__  S*   S__  S__  S__  S*   S__  S__
01:  S__  S__  S__  S__  S__  S__  S__  S__  S__  S__
02:  S__  S__  S__  S__  S__  S__  S__  S__  S__  S__
03:  S__  S__  S__  S*   S__  S__  S__  S__  S__  S__
04:  S__  S__  S__  S__  S__  S__  S__  S__  S__  S__
05:  S__  S__  S__  S__  S__  S*   S*   S__  S__  S__
06:  S__  S__  S__  S__  S__  S__  S__  S__  S__  S__
07:  H__  H50  H__  H__  H__  H__  H20  H60
08:  H__  H__  H__  H35  H__  H__  H__  H__
09:  H__  H__  H__  H__  H__  H__  H__  H__
10:  T45  T__  T__  T__  T50  T__
11:  T__  T__  T62  T__  T__  T70
```

7.2 Booking Date

When a unit is rented, instead of requiring a date parameter, use the system date. Hint: research the LocalDate and Calendar classes. Pay attention to the way the Calendar class is used; that information may come in handy on a later project.

8 Submitting Your Work

Follow the course's Submission Guide when submitting your project.

(project handout continues…)

9 Grading Matrix and Points Values

Area	Value	Evaluation
Design docs	10%	Did you submit initial design documents, and were they in reasonable shape to begin coding? Did you submit final design documents, and were they a good representation of the final version of the project?
Class updates and pricing model	25%	Were the classes updated to the current project's spec? Was the pricing model implemented as discussed in class?
Inheritance	15%	Was inheritance used, and used successfully and properly?
Polymorphism	15%	Was polymorphism used successfully and properly?
Abstract classes	10%	Were abstract classes used successfully and properly?
Abstract methods	5%	Were abstract methods used properly and correctly?
Style/internal documentation	10%	Did you use JavaDoc notation, and use it properly? Were other elements of style (including the Style Guide) followed?
JUnit tests	10%	Were test classes created for each production class? Were all possible methods tested? Was all state tested?
EC: unit map	2%	Did you produce a correct and attractive unit map?
EC: rental date	1%	Was the current date automatically added at rental time?
Total	**103%**	

Code that does not compile or run won't be graded