CS 2102 D-Term 2025 Ahrens

Lab 3 (10 Points)
& Homework 3 - Stateful Data Processing
Template Method Pattern with Inheritance (60 Points)

Lab Due: Wednesday, April 9, 2025 @ 11:59 pm EDT
Lab Late: Friday, April 11, 2025 @ 11:59 pm EDT (No Penalty)
HW Due: Tuesday, April 15, 2025 @ 11:59 pm EDT
HW Late: Thursday, April 17, 2025 @ 11:59 pm EDT (Late Pass or Penalty)

You must do this **lab** individually but may collaborate on ideas and tests with your lab section.
You may do this **homework** individually or with a partner for extra credit.

Recommended Background knowledge (from lecture)

- Inheritance for code reuse
- Domain shifting via helper methods and using intermediate data structures
- Access and Mutation of fields
- Inheritance by design using Abstract classes

Assignment Goals (To practice...)

- Supertype-subtype relationships using inheritance (“is a” relationships)
- Defining client code that depends on an abstract class (supertype)
- Transforming data from one form to another to
    - Eliminate unwanted values (cleaning)
    - Make query logic easier (parsing)
- Have multiple implementations of the same operation to
    - Process data as answers are queried (batch processing)
    - Process data as it is entered (real-time processing)
- Structuring code ahead of time via using a class hierarchy, helper methods, and method overriding (Template Method Pattern)

The Assignment

Preliminaries

A. Read the **ENTIRE** document. There are some design problems when doing the HW.
B. Using IntelliJ is recommended.
Using another IDE (Eclipse, VSCode) is up to you.
C. Use of JDK version 17 (Sept 2021) is recommended.
Versions newer than 17 may not work on the autograder.
D. You are encouraged to make a new project **for each** lab+homework assignment, but for this one specifically you can build on HW1 if you like (reuse Examples.java, MAV, etc.)

Overview

You are helping develop a small, *Internet of Things (IoT)*-like library to process data coming in from a combo temperature and humidity sensor unit for a smart greenhouse. The Solarpunk farmers hiring you want to be able to deploy different implementations for different parts of the greenhouse. Some sensor deployments **collect frequently** but get **queried infrequently** while others are the opposite, so we want to be able to optimize with Batch and Real-Time Processing Strategies (BP vs RTP). Lastly, some real-time sensor deployments are only valid for a particular date and some batch-processing deployments are on error-prone hardware.

Data is collected in the form of a String with the following format:

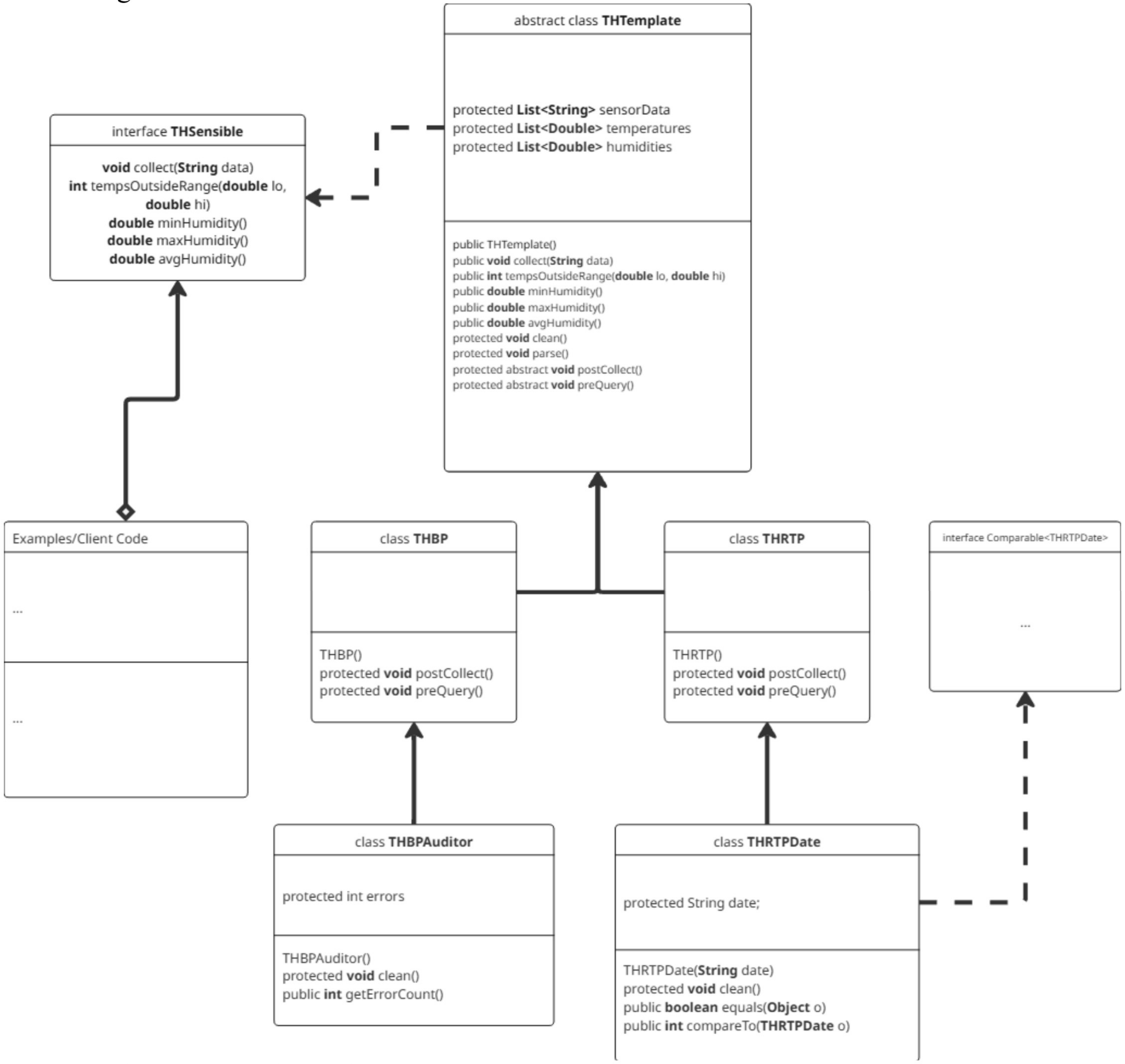
- The string starts with a yyyymmdd date followed by a space
- The string then has 0 or more space-separated entries which is one of:
    - "T" followed by a space and a decimal number
      in degrees Fahrenheit typically in the range of –100 to +200
    - "H" followed by a space and a decimal % of humidity
      in the range 0.0% - 100.0%
    - "Err" meaning a sensor had an error
- The sensors are distributed and asynchronous, so the dates are not guaranteed to come in in order.
- The code may collect data from multiple sensors multiple times before being queried for answers (collect() may be called multiple times).
- E.g., `"20250407 T 68.0 H 30.0 T 71.0 Err H 45.0 T 81.5 H 50.0 Err"`

The public methods available to the farmers' code will be provided by an interface. Those public methods will be implemented in an abstract class using the Template Method Design Pattern, with the deployment customization provided by subclasses and method overriding.

The structure can be seen via the UML and most of the methods that need to be tested are provided by the following THSensible interface (with incomplete documentation comments)

```java
public interface THSensible {
    /** Split and store sensor data */
    void collect(String data);
    /** @return counts temps outside range exclusively */
    int tempsOutsideRange(double lo, double hi);
    /** @return the lowest humidity % (or 0 if none) */
    double minHumidity();
    /** @return the largest humidity % (or 0 if none) */
    double maxHumidity();
    /** @return the avg humidity % (or 0 if none) */
    double avgHumidity();
}
```

UML Arrow Meanings:
Aggregation/Composition (Diamond arrows) is a “has a” relationship (fields)
Implementation (dotted) & Inheritance (solid) arrows are “is a” relationships (super/subtypes)
UML Diagram:


abstract class THTemplate
protected List<String> sensorData
protected List<Double> temperatures
protected List<Double> humidities
public THTemplate()
public void collect(String data)
public int tempsOutsideRange(double lo, double hi)
public double minHumidity()
public double maxHumidity()
public double avgHumidity()
protected void clean()
protected void parse()
protected abstract void postCollect()
protected abstract void preQuery()
interface THSensible
void collect(String data)
int tempsOutsideRange(double lo, double hi)
double minHumidity()
double maxHumidity()
double avgHumidity()
Examples/Client Code
...
...
class THBP
THBP()
protected void postCollect()
protected void preQuery()
class THRTP
THRTP()
protected void postCollect()
protected void preQuery()
interface Comparable<THRTPDate>
...
class THBPAuditor
protected int errors
THBPAuditor()
protected void clean()
public int getErrorCount()
class THRTPDate
protected String date;
THRTPDate(String date)
protected void clean()
public boolean equals(Object o)
public int compareTo(THRTPDate o)

TH means Temperature and Humidity
BP means Batch Processing
RTP means Real-Time Processing
THBPAuditor’s getErrorCount() extends beyond these methods to count the times ”Err” occurs.
THRTPDate only keeps data, while cleaning, for a given date.

Lab 3
Summary:

1. Using the UML diagram, **stub out** and set up the implements/extends relationships between: THSensible, THTemplate, THBP, THRTP
    a. THTemplate initializes its fields to empty lists.
2. Test the interface methods (all queries) on a THBP object

```
@Test
    public void testBPtor(){
       THSensible th = new THBP();
       th.collect("???");
       th.collect("???");
       assertEquals(???, th.tempsOutsideRange(???, ???));
}
```

3. Test the timing properties of THBP vs THRTP for **both** collect() and one query method. Note: You may want to make your test data large-ish so that data processing takes a non-trivial amount of time.

```
@Test
public void testCollectTiming(){
   THSensible bp = new THBP();
   THSensible rtp = new THRTP();

   long preBP = System.nanoTime();
   bp.collect("???");
   long postBP = System.nanoTime();
   long deltaBP = postBP - preBP;

   long preRTP = System.nanoTime();
   rtp.collect("???");
   long postRTP = System.nanoTime();
   long deltaRTP = postRTP - preRTP;

   assertTrue(deltaBP ??? deltaRTP);
}
```

4. [Extra Credit] **Stub** and **Test** THRTPByDate's and THBPAuditor's methods
    a. Only keeps data for a particular date set at construction

To turn in:

- **Submit** only your **Examples.java** unit tests to the Gradescope autograder

HW3

The listed order is recommended to implement the Template Method Pattern.

1. **Implement** the THSensible methods on THTemplate
    - Collect adds the data to the List<String> field and then calls postCollect()
        - Consider using: the string split() method, the Arrays.asList() static method, and the List addAll() method.
    - The query methods call preQuery() and then perform their operation using the temperatures and humidities fields, respectively. The query methods assume the temperatures and humidities fields are already sorted.
2. **Implement** clean() and parse() on THTemplate
    - clean() removes dates and errors from the List<String> field
    - parse() moves the data from the List<String> field to the List<Double> fields, sorts the List<Double> fields, and clears the List<String> field.
3. **Implement** the abstract methods, postCollect() and preQuery(), on THBP and THRTP
    - These methods should only call clean() and parse() when appropriate for that deplyoment strategy.
4. **Override** clean() or preQuery() on THBPAuditor
    - Counts the "Err" in the List<String> field in overridden method.
    - Calls super.clean() or super.preQuery() from overridden method
    - Calls preQuery() from getErrorCount()
5. **Override** clean() on THRTPDate
    - Deletes data following a date other than the date field from the List<String> field.
    - Calls super.clean().
6. **Override** equals(Object o) **and implement** Comparable<THRTPDate> on THRTPDate
    - Class signature:
      public class THRTPDate extends THRTP implements Comparable<THRTPDate> { ... }
    - Two objects should be equal if they have the same date
    - compareTo(THRTPDate other) should return what compareTo() on their date fields returns (a negative int if this date is earlier, 0 if this date is equal, positive int if this date is later).

Rubric:

[40 points] Autograder tests (only some are visible on submission)

[15 points] Template Method Design Pattern (maximum code reuse via inheritance)

- THTemplate does most of the computation for reuse
    - collect() and the query methods call the abstract methods correctly to stage stateful computation.
    - **None** of THTemplate's methods call clean() or parse() directly
- THBP and THRTP **only** implement the abstract methods to call clean() and parse() at the correct time (No need to define constructors or the existing methods)
- THRTPDate and THBPAuditor do **not** do more work than necessary when overriding clean()
    - They **only** do the logic for their specific cleaning logic and call super.clean() otherwise.
- If something is simply inherited, without overriding, do **not** redundantly define it on the subclasses. Let the subclass inherit it implicitly.

[5 points] Documentation and style

- Each class is documented with a javadoc description.
- Public methods are documented with javadocs (unless documentation is defined on a supertype)

What to submit to HW3 Gradescope

- THTemplate.java
- THBP.java
- THRTP.java
- THBPAuditor.java
- THRTPDate.java
- Examples.java

If working with a **partner:** add your partner's name to the submission using **"view or edit group"** in the top right of the submission.

Academic Integrity:

Do not upload any part of this assignment prompt to an LLM like ChatGPT and do not use CoPilot or IntelliJ's AI assistants for writing code for this course. If you reference any resources other than those provided by the course that may make your code look suspicious to graders, you should **cite its use** in a comment at the beginning of your Examples.java file's javadocs.