GMU Spring 2023 – CS 211 – Exercise 2

Due Date: Saturday, February 25th, 11:59 pm

Changelog

- Evidence first constructor:
  Use **byte** instead of **String** for the type of the parameter "type"):
  public **Evidence** (String **description**, byte **type**, byte **unambiguity**, byte **credibility**, byte **completeness**, byte **conclusiveness**)
- In the Evidence and Hypothesis classes, the method computeProbability signature is:
  public byte **computeProbability**()
- The probability displayed in the **printFullDescription**() example is 3 instead of 5. Use type casting from float to byte when computing the probability using the 4 factors (unambiguity, credibility, completeness, conclusiveness).
- Examples of tests provided at the end of the document.
- Other minor typos fixed. See text in red.

Description

Now that we've seen the syntax for creating our own classes, in this exercise we'll practice creating some simple classes with associated fields and methods. Let's assume we're developing a simple evidence-based reasoning program. You will need to model Problems, Hypotheses, Relevance, items of Evidence, and implement a simple program to combine those elements to make logical Reasoning.

There is no template provided for this exercise, so make sure that you read the instructions carefully to determine how your code should be structured. Briefly, you will be making five classes in their respective .java files. Remember: source files in java should be named exactly the same as the class they represent and have the .java extension. Make sure to include all the fields and methods asked for, paying particular attention to the access modifiers, return types, capitalization, and parameter order and type. In this exercise, these elements will be provided to you. However, in future assignments, you may be required to deduce some or all of them from the context of the problem.

Overview

1. Create the Java classes Problem, Hypothesis, Evidence, Relevance. The class Reasoning is provided to you.
2. Include all the fields and methods described below.
3. Test your code for correctness.
4. Prepare the assignment for submission and submit the java files through Gradescope.

Rules

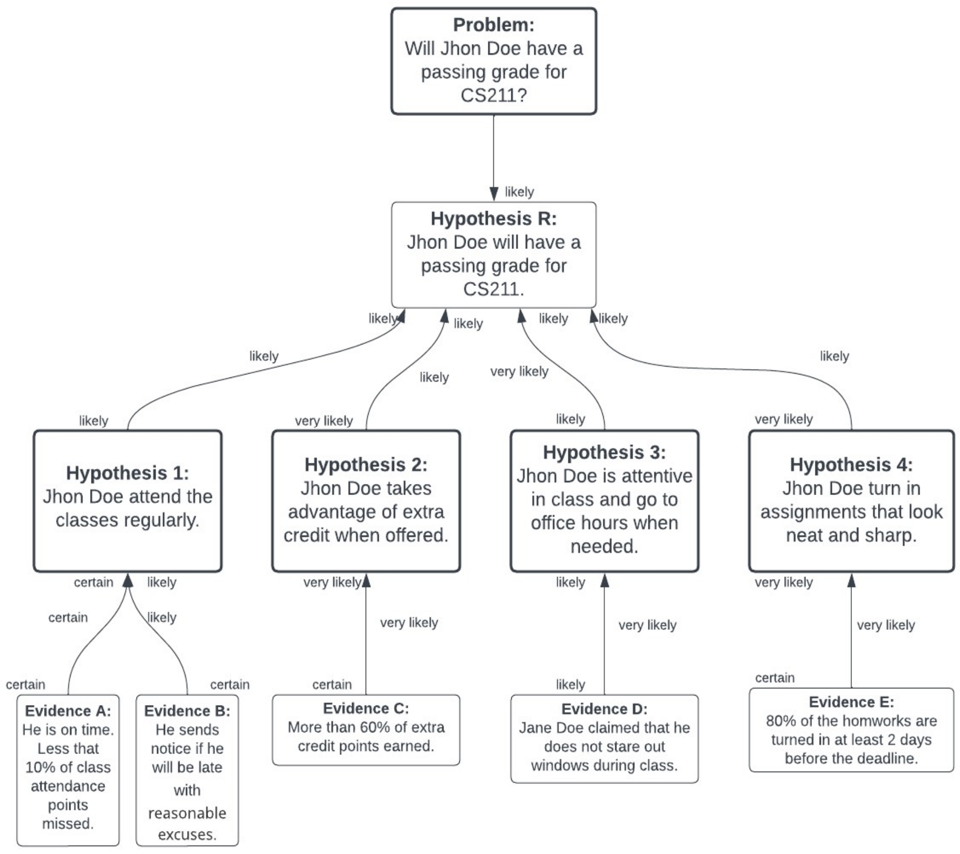
1. You may not import any extra functionality besides the default. For example, System and Math are imported by default and thus may be used, whereas something like ArrayList must be explicitly imported so it is disallowed.
2. The main method will not be tested; you may use it any way you want.
3. Comment your code, especially the parts where it is not obvious what you're doing.

Argumentation concepts

The figure below shows an example of a simplified version of a Wigmorean probabilistic inference network used to model an argumentation. This type of network has features required by models developed in the 3rd wave of AI: transparency, explanations, and trust.
To read more:

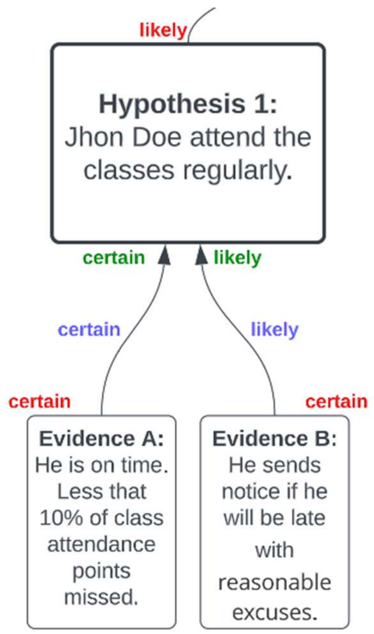
- The evidence-based reasoning: Tecuci, Gheorghe, et al. "Toward a Computational Theory of Evidence-Based Reasoning for Instructable Cognitive Agents." arXiv preprint arXiv:1910.03990 (2019). (https://arxiv.org/pdf/1910.03990)

- Wigmore Chart: https://en.wikipedia.org/wiki/Wigmore_chart


Problem:
Will Jhon Doe have a passing grade for CS211?
likely
Hypothesis R:
Jhon Doe will have a passing grade for CS211.
likely
likely
likely
likely
likely
likely
very likely
likely
likely
very likely
likely
very likely
Hypothesis 1:
Jhon Doe attend the classes regularly.
Hypothesis 2:
Jhon Doe takes advantage of extra credit when offered.
Hypothesis 3:
Jhon Doe is attentive in class and go to office hours when needed.
Hypothesis 4:
Jhon Doe turn in assignments that look neat and sharp.
certain
likely
very likely
likely
very likely
certain
likely
very likely
very likely
very likely
certain
certain
certain
likely
certain
Evidence A:
He is on time. Less that 10% of class attendance points missed.
Evidence B:
He sends notice if he will be late with reasonable excuses.
Evidence C:
More than 60% of extra credit points earned.
Evidence D:
Jane Doe claimed that he does not stare out windows during class.
Evidence E:
80% of the homworks are turned in at least 2 days before the deadline.

Probability: for this model, we will use a probability scale from 0 to 5. This scale is also used to quantify the relevance level and the inferential force.

- 0: Extremely Unlikely
- 1: Very Unlikely
- 2: Unlikely
- 3: Likely
- 4: Very Likely
- 5: Certain


likely
Hypothesis 1:
Jhon Doe attend the classes regularly.
certain
likely
certain
likely
certain
certain
Evidence A:
He is on time. Less that 10% of class attendance points missed.
Evidence B:
He sends notice if he will be late with reasonable excuses.

This will be used to quantify the probability scale of the:

- Hypothesis or Evidence (in **red**)
- Relevance (in **blue**)
- Inferential Force (in **green**)

The interpretation is the following:
Evidence A probability comes from the user. The user set the probability of Evidence A to **certain.** It makes sense since he gets attendance data from a reliable measurement system. So, he estimates that the probability of this Evidence being True is certain. He could assign a lower probability level if he had some doubts about the accuracy of those data.
Evidence B probability is also set to **certain** by the user.

Now the **Relevance** of *Evidence B to Hypothesis 1* is set by the user to **likely**. This value reflects how relevant the item of evidence is to the hypothesis being estimated. For instance, it is not certain that "The Student who sends a notice when he will be late would attend class regularly". But the user thinks that it is **likely** that this student is likely to be present regularly.
However, the **Relevance** of *Evidence A to Hypothesis 1* is set by the user to **certain**. Assuming that the item of evidence is true, it would be **certain** that this student attends the class regularly.

The **Inferential Force** is computed by the formula provided in this document. It reflects the probability of **Hypothesis 1** to be true if it depended only on a given item of evidence (or a Sub-Hypothesis). For instance, Hypothesis 1 would be **likely** to be true if it depended only upon Evidence B. Also, Hypothesis 1 would be **certain** to be true if it depended only upon Evidence A.

Consequently, the **Probability of Hypothesis 1** is **likely**. This value is computed by the formula provided in this document. It depends on the 2 **inferential forces** computed earlier.

Instructions

1. Implement the five java classes described below in five separate, appropriately named, .java files.
2. You must do your own testing. Feel free to create a main method in the driver class to test your code.
3. Submit the files on Gradescope. Make sure that you submit the correct files.

The Evidence class:

This class represents an item of evidence that can be used to support or disfavor a Hypothesis. Assume that we will use only favoring pieces of evidence and hypotheses. To evaluate the probability of an item of evidence being true, we need to know how ambiguous it is and how credible its source is. Out of the 5 well-known characteristics of an item of evidence, we will use 4 of them for simplicity. This relation depends on the type of evidence. The table below shows how this value should be estimated a priori:

/ Feature Type	Unambiguity {0, ..., 5}	Credibility {0, …, 5}	Completeness {0, …, 5}	Conclusiveness {0, …, 5}
1-Real Evidence	0.4	0.3	0.2	0.1
2-Testimonial Statement	0.2	0.4	0.2	0.2
3-Demonstrative Evidence	0.3	0.2	0.4	0.1
4-Documentary Evidence	0.2	0.6	0.1	0.1
5-Not Specified	0.25	0.25	0.25	0.25

For instance, if the evidence type is 1 (i.e Real Evidence), its probability of being true would be: 0.4(unambiguity) + 0.3(credibility) + 0.2(completeness) + 0.1(conclusiveness). The result will be a float value. You will use type casting to convert float to byte. This will truncate the float value.

You need to implement constructors and methods to instantiate the fields of an Evidence.

- ➔ **Fields**:
    - ◆ **description**: a **String** that represents the description of the evidence.
    - ◆ **type**: a **byte** that represents the type of evidence. The default value is 5.
    - ◆ **unambiguity**: a **byte** that represents the unambiguity of evidence.
    - ◆ **credibility**: a **byte** that represents the credibility of evidence.
    - ◆ **completeness**: a **byte** that represents the completeness of evidence.
    - ◆ **conclusiveness**: a **byte** that represents the conclusiveness of evidence.
    - ◆ **probability**: a **byte** that represents the probability of this evidence being true.

All fields are **private**.

➔ **Getters** and **Setters**:

The getters will be public and will have the following signature:

- **String getDescription()**
- **byte getType()**
- **byte getUnambiguity()**
- **byte getCredibility()**
- **byte getCompleteness()**
- **byte getConclusiveness()**
- **byte getProbability()**

The setters will be public and will have the following signature:

- **void setDescription(String description)**
- **void setType(byte type):**
    - *You must check if the value type is between 1 and 5. Otherwise set the value to 5.*
- **void setUnambiguity(byte unambiguity)**
- **void setCredibility(byte credibility)**
- **void setCompleteness(byte completeness)**
- **void setConclusiveness(byte conclusiveness)**

N.B- The value of the attributes unambiguity, credibility, completeness, and conclusiveness must be between 0 and 5. **If the user tries to set the value to an invalid number, choose the valid value that is closer to the argument.**

➔ **Constructors**: Create two constructors for this class. See the signature of the constructors below:

public **Evidence** (String **description**, byte **type**, byte **unambiguity**, byte **credibility**, byte **completeness**, byte **conclusiveness**)

public **Evidence** (String **description**, byte **unambiguity**, byte **credibility**, byte **completeness**, byte **conclusiveness**)

The constructors should instantiate the variables using the provided arguments and will compute the probability based on the weights and factors in the table above. For the second constructor, set the **type** to its default value and compute the probability of the evidence.

Business rules:

- The value of the above-mentioned factor needs to be between 0 and 5.
- The value of probability needs to be between 0 and 5 inclusively.
- If the argument is outside of this range, choose the valid value that is closer to the argument.

➔ **Other methods**: You need to define the following methods:

- public byte **computeProbability**(): This method can be used to compute the probability of an item of evidence.

- public static String **probability2String**(byte **probability**): This method can be used to return the String representation of the probability level. Eg. For input = 1, the function will return the String “Very Unlikely”. For 0 or invalid input, the function returns “Extremely Unlikely”.

- public String **toString**(): This method can be called to return the object in a human-friendly manner. It needs to return the following information:

```
Evidence: This is the description.
** Type of evidence: Real Evidence
** Probability: 3 -> likely
```

- public String **printFullDescription**(): This method can be called to return the object in a human-friendly manner. It re-uses the toString() method described above and needs to return the following information:

```
Evidence: This is the description.
** Type of evidence: Real Evidence
** Probability: 3 -> likely
** Evaluated based on those characteristics:
** >> Unambiguity: 5
** >> Credibility: 3
** >> Completeness: 4
** >> Conclusiveness: 2
```

The Relevance class:

This class will represent the relevance of an Evidence or another Hypothesis (Sub-Hypothesis) to a specific Hypothesis of interest. Here is the detailed specification:

➔ **Fields**:

- **evidence**: an **Evidence** object representing the item of evidence whose relevance is represented by the current object.
- **subHypothesis**: a **Hypothesis** object representing the sub-hypothesis whose relevance is represented by the current object.

- **level**: a **byte** representing the probability of the hypothesis being true if the underlying sub-hypothesis or the evidence was certain. This value must be between 0 and 5.
- **inferentialForce**: a **byte** representing the probability of the Hypothesis of interest to be true considering only the underlying evidence or sub-hypothesis.

All fields are **private**.

➔ **Getters and Setters:**

The getters will be public and will have the following signature:

- **Evidence getEvidence()**
- **Hypothesis getSubHypothesis()**
- **byte getLevel()**
- **byte getInferentialForce()**

The setters will be public and will have the following signature:

- **void setEvidence(Evidence evidence)**
  - This setter method will also set the subHypothesis attribute to **null**. And will set the inferential force calling the function **setInferentialForce()** described below.
- **void setSubHypothesis(Hypothesis subHypothesis)**
  - This setter method will also set the evidence attribute to **null**. And will set the inferential force calling the function **setInferentialForce()** described below.
- **void setLevel(byte level):**
  - The level must be between 0 and 5. Otherwise, use the valid value that is closer to the argument provided. This method will also set the inferential force by calling the function **setInferentialForce()** described below.
- **void setInferentialForce():**
  - This function takes no argument. It will use the following formula to compute the inferential force:
    - If the hypothesis of interest is supported by an item of evidence:
    inferentialForce = Min(evidence.probability, relevance.level)
    - If the hypothesis of interest is supported by a sub-hypothesis:
    inferentialForce = Min(subhypothesis.probability, relevance.level)

➔ **Constructors**: Create two constructors for this class as follows. The value of the inferentialForce will be computed based on the formula provided.

public **Relevance** (**Evidence** evidence, **byte** level)

public **Relevance** (**Hypothesis** subHypothesis, **byte** level)

<u>*Business rules:*</u>

- For any update in the relevance level, the program needs to make sure that the value of the inferential force is correct.

➔ **Other methods**: You need to define the following methods:

- public byte **computeInferentialForce**(): This method can be used to re-compute and update the value of the inferential force based on the formula provided earlier. It returns the computed value.

- public String **toString**(): This method can be called to return the object in a human-friendly manner. It needs to return the following information:

```
Relevance:
** of: Desc. of the sub-hypothesis or Evidence.
** relevance level: 5 -> certain
** inferential force: 3 -> likely
```

The Hypothesis class:

This class will represent a Hypothesis. Here is the detailed specification:

➔ **Fields**:

- **description**: a **String** containing the description of the Hypothesis.
- **supportingItems**: an **array of Relevance**.
- **probability**: a **byte** representing the probability of the hypothesis being true.

Those attributes need to be **private**.

➔ **Getters and Setters:**

The getters will be public and will have the following signature:

- **String getDescription()**
- **Relevance[] getSupportingItems()**
- **byte getProbability()**

The setters will be public and will have the following signature:

- **void setDescription(String)**
- **void setSupportingItems(Relevance[])**
- **void setProbability(byte)**: Set the probability to the specified value. If provided argument is not between 0 and 5, set it to the closest valid value. For instance, if the provided value is -1, set the value to 0.
- **void setProbability()**:

  For simplicity, this probability will be computed based on the inferential force of each underlying Hypothesis or item of Evidence (use the attribute supportingItems for this).

  probability = Min(all the supportingItems **inferentialForces**)

➔ **Constructors**: Create one constructor for this class as follows.

public **Hypothesis** (String **description**, Relevance[] **supportingItems**)
public **Hypothesis** (String **description**)
When the second constructor is invocated, the value of the probability must be set to 0.

Business rules:

- For any update in the array of supporting items, i.e. when the methods setSupportingItems() or addSupportingItem() are called, the program needs to make sure that the value of the **probability** is correct by calling the method **setProbability**().

➔ **Other methods**: You need to define the following methods:

- void **addSupportingItem**(Relevance): This method adds a relevance item to the array of Relevance. Add this element to the last position i.e. after the previous elements.

- public byte **computeProbability**(): This method can be used to compute and update the probability of a hypothesis being true. It returns the probability.

- public String **toString**(): This method can be called to return the object in a human-friendly manner. It needs to display the following information:

```
Hypothesis:
** Description: Description of the hypothesis.
** Probability: 3 -> likely
```

The Problem class:

The Problem is where everything starts. It is the question we want to answer. In the example described in this document, the question is: “Will John Doe have a passing grade for CS211?”. The class representing a Problem will be as follow:

- **Fields**:
  - **question**: a **String** containing the question we are trying to answer.
  - **rootHypothesis**: a hypothesis formed from what we believe should be the most probable answer to the question. For simplicity, we will consider situations where only one root hypothesis per problem is necessary for the analysis.

  Those fields are public.

- **Constructors**: Create two constructors for this class as follows.

  public **Problem** (String **question**, Hypothesis **rootHypothesis**)
  public **Problem** (String **question**)

- **Method**:
  - **public String generateArgumentation**():

    This method needs to return the report following strictly this template:

```
Auto-generated report:
	Question: this.question
	It is VERY UNLIKELY to observe rootHypothesis.description
	This conclusion is based on the following hypothesis or evidence:
		It is VERY UNLIKELY that hypothesis1.description
		It is VERY LIKELY that hypothesis2.description
		It is CERTAIN that evidence1.description
```

    Use tab (\t) for the indentations. And no extra space at the end of each line.

    If rootHypothesis cannot be dereferenced,
    Or if there is not supportingItems fo the rootHypothesis,
    return an empty String: “”

The Reasoning class:

This is the driver class. It contains the main method that you need to execute in order to perform the analysis. This class is provided. Feel free to edit it to test your code.

Testing

Below is an example of how those classes could be used, you must do more testing. Feel free to use the provided Reasoning.java class for this purpose.

Test	Input	Output

1	`Problem prob =`
	`  new Problem("Is Jupyter producing Nuclear Bomb?");`
	
	`System.out.println(prob.generateArgumentation());`	

2	`Hypothesis rootHypothesis =`
	`  new Hypothesis("Jupyter is producting Nuclear Boomb.");`
	
	`System.out.println(rootHypothesis);`
		`Hypothesis:`
		`** Description: Jupyter is producting Nuclear Boomb.`
		`** Probability: 0 -> extremely unlikely`

3	`Problem prob =`
	`  new Problem("Is Jupyter producing Nuclear Bomb?");`
	
	`Hypothesis rootHypothesis =`
	`  new Hypothesis("Jupyter is producting Nuclear Boomb.");`
	
	`prob.rootHypothesis = rootHypothesis;`
	
	`System.out.println(prob.generateArgumentation());`	

4	`Evidence claim1 =`
	`  new Evidence("Jupyter does have scientist. (GMU claim)",`
	`(byte)1, (byte)5, (byte)3, (byte)4, (byte)2);`
	
	`System.out.println(claim1);`
		`Evidence: Jupyter does have scientist. (GMU claim)`
		`** Type of evidence: Real Evidence`
		`** Probability: 3 -> likely`

5	`Evidence claim1 =`
	`  new Evidence("Jupyter does have scientist. (GMU claim)",`
	`(byte)1, (byte)5, (byte)3, (byte)4, (byte)2);`
	
	`System.out.println(claim1.printFullDescription());`
		`Evidence: Jupyter does have scientist. (GMU claim)`
		`** Type of evidence: Real Evidence`
		`** Probability: 3 -> likely`
		`** Evaluated based on those characteristics:`
		`** >> Unambiguity: 5`
		`** >> Credibility: 3`
		`** >> Completeness: 4`
		`** >> Conclusiveness: 2`

6

```
Evidence claim1 =
  new Evidence("Jupyter does have scientist. (GMU claim)",
(byte)1, (byte)5, (byte)3, (byte)4, (byte)2);

System.out.println(claim1.printFullDescription());

Relevance relevance1 =
  new Relevance(claim1, (byte)3);

System.out.println(relevance1);
```

```
Evidence: Jupyter does have scientist. (GMU claim)
** Type of evidence: Real Evidence
** Probability: 3 -> likely
** Evaluated based on those characteristics:
** >> Unambiguity: 5
** >> Credibility: 3
** >> Completeness: 4
** >> Conclusiveness: 2
Relevance
** of: Jupyter does have scientist. (GMU claim)
** relevance level: 3 -> likely
** inferential force: 3 -> likely
```

7

```
Problem prob = new Problem("Is Jupyter producing
Nuclear Bomb?");
Hypothesis rootHypothesis = new Hypothesis("Jupyter is
producting Nuclear Boomb.");
prob.rootHypothesis = rootHypothesis;

Evidence claim1 = new Evidence("Jupyter does have
scientist. (GMU claim)", (byte)1, (byte)5, (byte)3,
(byte)4, (byte)2);
Relevance relevance1 = new Relevance(claim1, (byte)3);
rootHypothesis.addSupportingItem(relevance1);

Hypothesis subHypothesis1 = new Hypothesis("Jupyter
has enough space to develop weapons.");
subHypothesis1.setProbability((byte)5);
Relevance relevance2 = new Relevance(subHypothesis1,
(byte)3);
rootHypothesis.addSupportingItem(relevance2);

Hypothesis subHypothesis2 = new Hypothesis("Jupyter
has no legal constraint.");
subHypothesis2.setProbability((byte)5);
Relevance relevance3 = new Relevance(subHypothesis2,
(byte)1);
rootHypothesis.addSupportingItem(relevance3);

rootHypothesis.setProbability();

System.out.println(prob.generateArgumentation());
```

```
Auto-generated report:
        Question: Is Jupyter producing Nuclear Bomb?
        It is VERY UNLIKELY to observe Jupyter is producting Nuclear Boomb.
        This conclusion is based on the following hypothesis or evidence:
                It is LIKELY that Jupyter does have scientist. (GMU claim)
                It is CERTAIN that Jupyter has enough space to develop weapons.
                It is CERTAIN that Jupyter has no legal constraint.
```