PA1 OOP Review

Submit Assignment

Due Friday by 11:59pm **Points** 100 **Submitting** a file upload

Programming Assignment 1: OOP Review

1 Purpose

The primary purpose of this assignment is to help you review (and demonstrate that you have acquired) the knowledge and skills required to program in Java. From a language and algorithms perspective, there is nothing new in this assignment. There is also nothing new from a software engineering perspective.

This assignment will also help you gain some experience with the integrated development environment that you will be using this semester in this course. Some of the tools you will be using this semester will be new to some of you, and we will discuss them as the semester progresses. Regardless of whether you have used them in the past, at this point, you should be able to figure out everything that you need for this assignment.

2 Overview

SagaciousMedia is a (fictitious) company that develops educational hardware, software, and content for both the formal and informal education markets. SagaciousMedia's products are designed to excite and educate students, and to inspire and assist teachers/instructors.

They are in the process of developing a suite of products for working with grading assignments of various kinds (e.g., exams, homework assignments, labs, programming assignments). They have hired you to construct several interfaces/classes that will, ultimately, become part of these products.

These classes/interfaces might be used by the first application they are creating (called **Gradient**) to calculate the numerical grade for a course that has 6 programming assignments (with the lowest dropped) that account for 40% of the course grade, 5 homework assignments that account for 10% of the course grade, one midterm exam that accounts for 20% of the course grade, and a final exam that accounts for 30% of the course grade. For example, assuming the the programming assignments and homework assignments are each graded on a 20-point scale and the exams are each graded on a 100-point scale, the following grades

```
20.0 18.0 5.0 15.0 20.0 20.0 20.0 5.0 0.0 10.0 15.0 80.0 75.0
```

should result in a PA grade of 20.0+18.0+15.0+20.0+20.0=93.0, a homework grade of 20.0+5.0+0.0+10.0+15.0=50.0 and a course grade of 0.4·93.0+0.1·50.0+0.2·80.0+0.3·75.0=37.2+5.0+16.0+22.5=80.7.

3 Preparatory Tasks

Before you do anything else you should:

1. Install the latest version of Eclipse on your computer. (Using other IDEs should be OK for this assignment. But since later in the semester we will practice using Eclipse for debugging, static analysis, and working with Git, it would be better to start using it now.)
2. Read all policies related to homework assignments in the **syllabus**.

4 Documents

Sagacious Media uses a software process called Scrum. As a result, the documents they have created are organized in a particular way.

The terms that Sagacious Media uses when discussing Gradient and its functionality are described in the following document.

- **Product Domain Glossary**

The requirements of the first product they are building, **Gradient**, have been organized into what are called *stories*. *Epics* are abstract stories that might take several releases (developed over the course of many months) to completely realize. Each release has its own set of *sprintable stories* that describe its features. Both the epics and the sprintable stories (which are what you are concerned with for this assignment) for Gradient have been collected in the following document.

- **Stories for Gradient**

The existing team at Sagacious Media used the *sprintable* stories to create a set of tasks for this assignment. They completed some of the tasks and have left others for you to complete. The complete set of tasks is contained in the following document.

- **Tasks for Gradient**

The tasks that are "checked" have already been completed by other team members, the evidence for which is contained in the following documents (which are listed in alphabetical order).

- Engineering Design for Gradient
- Implementation of the `Gradient` class
- Specification of the `DropFilter` class
- Specification of the the `Grade` class
- Specification of the `Missing` class
- Specification of the `SizeException` class
- Specification of the `TotalStrategy` class
- Specification of the `WeightedTotalStrategy` class
- System/Integration Tests for Gradient

You must complete the tasks that have not been "checked-off".

5 A Recommended Process

The tasks that were identified by the team at Sagacious Media are organized by story. Hence, though they are numbered so that they can be referred to in documents and conversations, the numbers should not, in any way, influence the order in which you complete them. I would suggest you sequence your activities as follows.

1. Read all of the documents carefully. (This is utterly important, which is often ignored, unfortunately.)
2. Review the source code for the `Gradient` class so that you understand how the classes you are implementing will be used.
3. Implement the `Grade` class.
4. Test and debug the `Grade` class.
5. Implement the `Missing` class.
6. Test and debug the `Missing` class.
7. Implement the `SizeException` class.
8. Implement the `GradingStrategy` interface.
9. Think about how the `WeightedTotalStrategy` and `TotalStrategy` classes should be related (if at all).
10. Implement the `WeightedTotalStrategy` or `TotalStrategy` class (whichever one you think should be implemented first).
11. Test and debug the class you just implemented.
12. Implement the `WeightedTotalStrategy` or `TotalStrategy` class (whichever one you think should be implemented second).
13. Test and debug the class you just implemented.
14. Implement the `Filter` interface.
15. Implement the `DropFilter` class.
16. Test and debug the `DropFilter` class.
17. Test and debug the complete system.

Note that, you will need to have a stubbed-out version of all of the classes for your code to compile. This will, hopefully, encourage you to create and use stubs (i.e., pieces of code that stand in for the code that would be needed to provide full functionality) as part of your development process.

6 Getting Help

If you need help with UML syntax, please refer to the following ebook chapters:

- Object and Classes (http://proquest.safaribooksonline.com/book/certification/9780128097830/chapter-16dot-behavior-activity-diagrams/st0020_html_9?query=((activity+diagram))#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAxMjgwOTc4MzAlMkZjaHAwMDZ0aXRsX2h0bWwmcXVlcnk9KChhY3Rpdml0eSUyMGRp
- Packages and Namespaces (http://proquest.safaribooksonline.com/book/certification/9780128097830/chapter-16dot-behavior-activity-diagrams/st0020_html_9?query=((activity+diagram))#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAxMjgwOTc4MzAlMkZjaHAwMDh0aXRsX2h0bWwmcXVlcnk9KChhY3Rpdml0eSUyMGRp
- The Static Model (http://proquest.safaribooksonline.com/book/certification/9780128097830/chapter-16dot-behavior-activity-diagrams/st0020_html_9?query=((activity+diagram))#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAxMjgwOTc4MzAlMkZjaHAwMTB0aXRsX2h0bWwmcXVlcnk9KChhY3Rpdml0eSUyMGRp

7 Submission

Please create a zip file including all your source code, and submit the zip file through Canvas. By default, your latest submission will be graded.

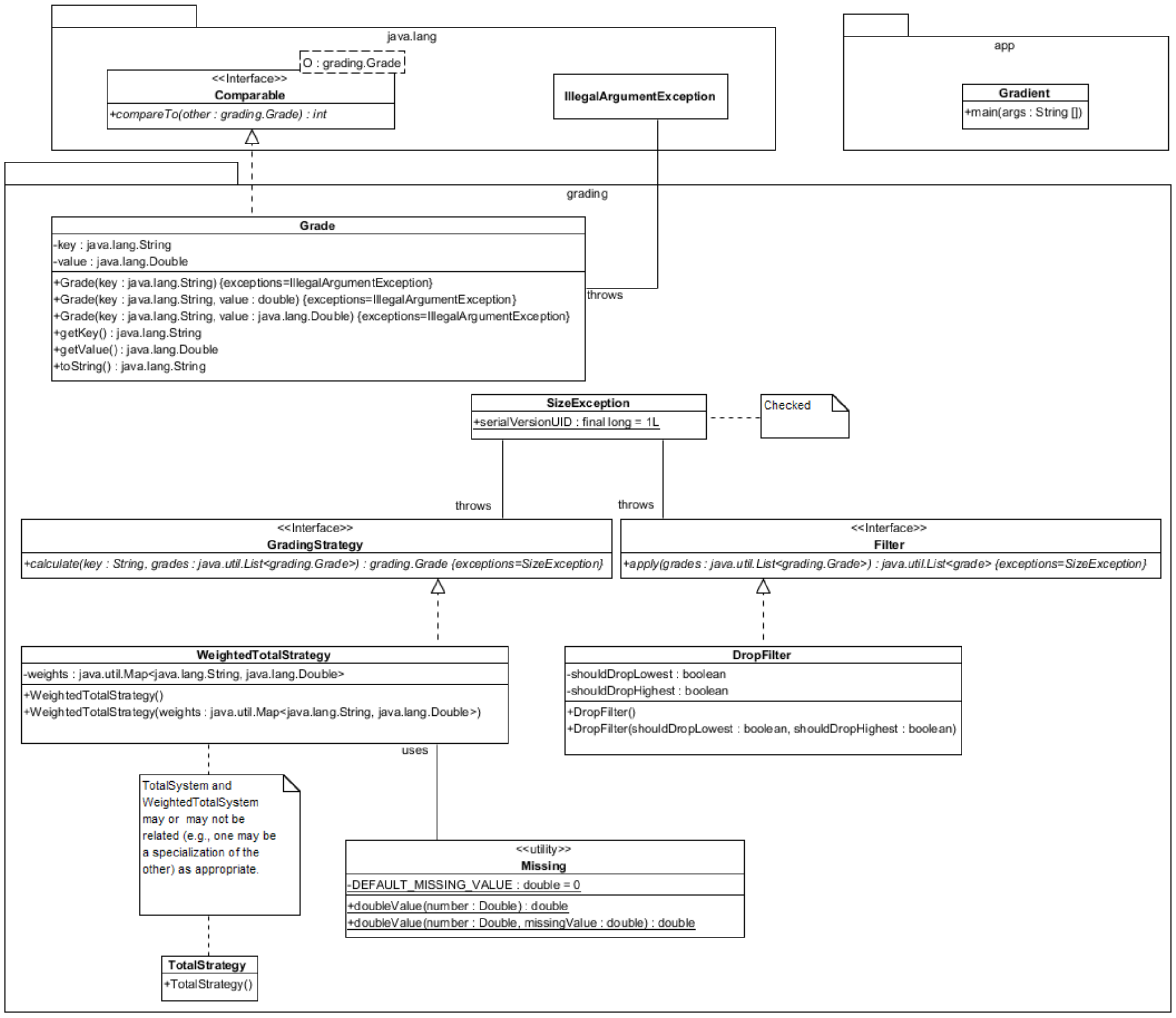
8 Grading

Partial credit will be awarded. Your grade will be based on the percentage of the system tests (worth 7*10 = 70 points) that your classes pass and on the quality of your code (30 points), as determined by our grader.

Since we have not discussed unit testing yet you will not be penalized **now** if your classes do not conform to the specifications (as long as your system passes the system tests). However, you will be penalized for such defects in the future. Hence, you should strive to write code that conforms to the specifications (and test it as best you can). Also, you should strive to write code that is clear, elegant, and well-documented.

Rule of Thumb: Start Early, Read, Think.

Engineering Design


java.lang
O : grading.Grade
<<Interface>>
Comparable
+compareTo(other : grading.Grade) : int
IllegalArgumentException
app
Gradient
+main(args : String [])
grading
Grade
-key : java.lang.String
-value : java.lang.Double
+Grade(key : java.lang.String) {exceptions=IllegalArgumentException}
+Grade(key : java.lang.String, value : double) {exceptions=IllegalArgumentException}
+Grade(key : java.lang.String, value : java.lang.Double) {exceptions=IllegalArgumentException}
+getKey() : java.lang.String
+getValue() : java.lang.Double
+toString() : java.lang.String
throws
SizeException
+serialVersionUID : final long = 1L
Checked
throws
throws
<<Interface>>
GradingStrategy
+calculate(key : String, grades : java.util.List<grading.Grade>) : grading.Grade {exceptions=SizeException}
<<Interface>>
Filter
+apply(grades : java.util.List<grading.Grade>) : java.util.List<grade> {exceptions=SizeException}
WeightedTotalStrategy
-weights : java.util.Map<java.lang.String, java.lang.Double>
+WeightedTotalStrategy()
+WeightedTotalStrategy(weights : java.util.Map<java.lang.String, java.lang.Double>)
DropFilter
-shouldDropLowest : boolean
-shouldDropHighest : boolean
+DropFilter()
+DropFilter(shouldDropLowest : boolean, shouldDropHighest : boolean)
uses
TotalSystem and WeightedTotalSystem may or may not be related (e.g., one may be a specialization of the other) as appropriate.
<<utility>>
Missing
-DEFAULT_MISSING_VALUE : double = 0
+doubleValue(number : Double) : double
+doubleValue(number : Double, missingValue : double) : double
TotalStrategy
+TotalStrategy()

Specifications: `Grade`

In addition to the obvious specifications illustrated in the UML class diagram, the `Grade` class must satisfy the following specifications.

1. `Grade` objects must be immutable.
2. If a constructor is passed a `key` that is `null` or empty (i.e., "") then the constructor must throw an `IllegalArgumentException`.
3. The `Grade(String key)` constructor must construct a `Grade` object with a `value` attribute of `0.0`.
4. The `compareTo(Grade other)` method must return the result of comparing `this.value` and `other.value` accounting for missing (i.e., `null`) values appropriately.
   - 4.1. If `this.value` is `null` and `other.value` is non-`null` then it must return `-1`.
   - 4.2. If `this.value` is `null` and `other.value` is `null` then it must return `0`.
   - 4.3. If `this.value` is non-`null` and `other.value` is `null` then it must retun `1`.
   - 4.4. If both `this.value` and `other.value` are non-`null` then it must return the result of calling `compareTo()` on `this.value` and passing it `other.value` (though it need not be implemented this way).
5. The `toString()` method must return a `String` representation of the `Grade` object.
   - 5.1. If the `value` attribute is not `null` then the `String` must contain the `key` attribute, followed by the `String` literal "`:`", followed by a single space, followed by the `value` attribute (in a field of width 5 with 1 digit to the right of the decimal point).
   - 5.2. If the `value` attribute is `null` then the `String` must contain the `key` attribute, followed by by the `String` literal "`:`", followed by a single space, followed by the `String` literal "`   NA`" (which is also a field of width 5).

Note that, while `null` `key` attributes are invalid (i.e., every `Grade` object must have a non-`null`, non-empty `key` attribute), `null` `value` attributes are valid (and are used to indicate that the `Grade` is missing).

Specifications: **`Missing`**

The `Missing` class is a utility class. In addition to the obvious specifications illustrated in the UML class diagram, the `Missing` class must satisfy the following specifications.

1. The default `double` value of a missing value must be `0.0`.
2. The method `doubleValue(Double number)` must return the `double` value of the `Double` parameter unless it is `null`, in which case it must return the default `double` value of a missing value.
3. The method `doubleValue(Double number, double missingValue)` must return the `double` value of the `Double` parameter unless it is `null`, in which case it must return `missingValue`.

Specifications: **`SizeException`**

In addition to the obvious specifications illustrated in the UML class diagram, the `SizeException` class must satisfy the following specifications.

1. The `SizeException` class must be a checked exception.

Specifications: **`TotalStrategy`**

In addition to the obvious specifications illustrated in the UML class diagram and the specifications for the parent class and interface, the `TotalStrategy` class must satisfy the following specifications.

1. `public` methods must not have any side effects. That is, they must not change the parameters that they are passed in any way (e.g., the `List` that is passed to the `calculate()` method must not be changed in any way) and they must not change attributes that are not "owned" (i.e., attributes that are aliases) in any way.
2. The `calculate()` method must calculate the total of the `List` of `Grade` objects it is passed.
   - 2.1. You may assume that the `calculate()` method is passed a `List` that does not contain any `null` elements.
   - 2.2. If the `List` is `null` then it must throw a `SizeException`.
   - 2.3. If the `List` is empty then it must throw a `SizeException`.
   - 2.4. Otherwise, it must return a `Grade` object with the given key and a value equal to the total of the `Grade` objects in the `List`.
     - 2.4.1. If the value of a particular `Grade` is missing (i.e., `null`) then a value of `0.0` must be used. Note: The `Missing` class has a method that can be used to accomplish this.

Specifications: **WeightedTotalStrategy**

In addition to the obvious specifications illustrated in the UML class diagram and the specifications for the parent class and interface, the `WeightedTotalStrategy` class must satisfy the following specifications.

1. `public` methods must not have any side effects. That is, they must not change the parameters that they are passed in any way (e.g., the `List` that is passed to the `calculate()` method must not be changed in any way) and they must not change attributes that are not "owned" (i.e., attributes that are aliases) in any way (e.g., the `Map` that is passed to the constructor must not be changed in any way).
2. The `calculate()` method must calculate the weighted total of the `List` of `Grade` objects it is passed.
   - 2.1. You may assume that the `calculate()` method is passed a `List` that does not contain any `null` elements.
   - 2.2. If the `List` is `null` then it must throw a `SizeException`.
   - 2.3. If the `List` is empty then it must throw a `SizeException`.
   - 2.4. Otherwise, it must return a Grade object with the given key and a value equal to the weighted total of the `Grade` objects in the `List`.
     - 2.4.1. The weight for each element must be obtained from the `Map` using the key for that element.
       - 2.4.1.1. If the `weights Map` is `null` than a weight of `1.0` must be used.
       - 2.4.1.2. If the weight for a particular `Grade` is unspecified (i.e., `null)` then a weight of `1.0` must be used. Note: The `Missing` class has a method that can be used to accomplish this.
       - 2.4.1.3. If the weight for a particular `Grade` is less than `0.0` then a weight of `0.0` must be used.
     - 2.4.2. If the value of a particular `Grade` is missing (i.e., `null`) then a value of `0.0` must be used. Note: The `Missing` class has a method that can be used to accomplish this.
3. The default constructor must (directly or indirectly) initialize the `weights Map` to `null`.

Specifications: `DropFilter`

In addition to the obvious specifications illustrated in the UML class diagram, the `DropFilter` class must satisfy the following specifications.

1. `public` methods must not have any side effects. That is, they must not change the parameters that they are passed in any way (e.g., the `List` that is passed to the `apply()` method must not be changed in any way) and they must not change attributes that are not "owned" (i.e., attributes that are aliases) in any way.
2. You may assume that the `apply()` method is passed a `List` that does not contain any `null` elements.
3. The default constructor must construct a `DropFilter` object that drops the lowest and highest element.
4. The `apply()` method must construct a new `List` that is a subset of the `List` it is passed.
    - 4.1. If the `apply()` method is passed a `List` that has an inappropriate size then it must throw a `SizeException`.
        - 4.1.1. If the `apply()` method is passed a `null List` then it must throw a `SizeException`.
        - 4.1.2. If the `apply()` method is passed a `List` that contains fewer elements than are to be dropped then it must throw a `SizeException`.
        - 4.1.3. If the `apply()` method is passed a `List` that contains the same number of elements as are to be dropped then it must throw a `SizeException`.
    - 4.2. If the `apply()` method is passed a list that has an appropriate size then it must return a new `List`.
        - 4.2.1. The elements of the new `List` must be aliases for (**not** copies of) the `Grade` objects in the `List` it is passed.
        - 4.2.2. Because each `Grade` object in the `List` has a key that can be used to identify it, the new `List` need not be in the same order as the `List` it is passed.
        - 4.2.3. The elements in (and size) of the returned List must be based on the values of the parameters that were passed to the constructor when the object was constructed.
            - 4.2.3.1. If `shouldDropLowest` was `true` then it must drop exactly one of the elements with the lowest value in the original `List`.
            - 4.2.3.2. If `shouldDropHighest` was `true` then it must drop exactly one of the elements with the highest value in the original `List`.
            - 4.2.3.3. When dropping the highest and lowest, two elements must always be dropped, even if the highest and lowest have the same value.
            - 4.2.3.4. When determining the highest and/or lowest values it must account for missing (i.e., `null`) values as in the `compareTo()` method of the `Grade` class (i.e., missing values have smaller magnitude than non-missing values and one missing value has the same magnitude as another missing value).

Epics

E1. As a user I want to be able to enter the numeric grades for the assignments in a course and have the system calculate my numeric grade for the course so that I don't have to perform the calculations by hand.

E2. As a salesperson I want the system to be able to handle courses with arbitrary structures so that we can sell the system to everyone.

E3. As an administrator I want to be able to enter information that describes the structure of a course so that it can be used for different courses.

Sprintable Stories

ID	Story	Acceptance Criteria
S1	As a salesperson I want the system to work for a course with 6 programming assignments where the lowest is dropped (40% of the course grade), 5 homework assignments (10% of the course grade), one mid-term exam (20% of the course grade), and one final exam (30% of the course grade) so that the system can be used right away in several courses offered by one of our perspective clients.	☐ Calculation of correct result for three integration tests with no missing values.
S2	As a salesperson I want the grade for a category to be the total of the grades on the assignments in that category so that the system can be used right away in several courses offered by one of our perspective clients.	☐ Calculation of correct result for three integration tests with no missing values.
S3	As a user I want the system to be able handle missing grades so that I can use it while a course is underway.	☐ Calculation of correct result for one integration test with one missing value in one category. ☐ Calculation of correct result for one integration test with one missing value in each category.

		☐ Calculation of correct result for one integration test with multiple missing values in each category. ☐ Calculation of correct result for one integration test with all missing values.
S4	As a user I want to be able to enter all of the grades for a course at the command line so that the system is easy to use.	☐ Calculation of correct result for three system tests with no missing values.
S5	As a user I want to be able to enter missing grades as “NA” so that they can be easily distinguished from actual grades.	☐ Calculation of correct result for one system test with one missing value in one category entered as “NA” from the command line. ☐ Calculation of correct result for one system test with one missing value in each category entered as “NA” from the command line. ☐ Calculation of correct result for one system test with multiple missing values in each category entered as “NA” from the command line. ☐ Calculation of correct result for one system test with all missing values.
S6	As a user I want the system to display the course grade after I enter all of the grades at the command line course so that I don’t have to perform the calculations by hand.	☐ Display of correct output for three system tests with no missing values. ☐ Display of correct output for one system test with one missing value in one category entered as “NA” from the command line. ☐ Display of correct output for one system test with one missing value in each category entered as “NA” from the command line. ☐ Display of correct output for one system test with multiple missing values in each category entered as “NA” from the command line.

☐ Display of correct result for one system test with all missing values.

Tests

Course Structure

The course grade is calculated as a weighted total of a programming assignment grade (40%), a homework assignment grade (10%), a mid-term exam grade (20%), and a final exam grade (30%).

> The programming assignment grade is the total of 6 programming assignments each of which is graded on a 20-point scale but the lowest grade is dropped.
>
> The homework assignment grade is the total of 5 programming assignments each of which is graded on a 20-point scale.
>
> The mid-term exam is graded on a 100-point scale.
>
> The final exam is graded on a 100-point scale.

Complete 01

PA1: 20
PA2: 18
PA3: 5
PA4: 15
PA5: 20
PA6: 20
Programming Assignments: 20 + 18 + 15 + 20 + 20 = 93

H1: 20
H2: 0
H3: 5
H4: 10
H5: 15
Homework: 20 + 0 + 5 + 10 + 15 = 50

Midterm Exams: 80

Final Exams: 75

Course: 93·0.4 + 50·0.1 + 80·0.2 + 75·0.3 = 37.2 + 5.0 + 16.0 + 22.5 = 80.7

Complete 02

PA1: 20
PA2: 20
PA3: 20
PA4: 20
PA5: 20
PA6: 20
Programming Assignments: 20 + 20 + 20 + 20 + 20 = 100

H1: 20
H2: 20
H3: 20
H4: 20
H5: 20
Homework: 20 + 20 + 20 + 20 + 20 = 100

Midterm Exams: 100

Final Exams: 100

Course: 100·0.4 + 100·0.1 + 100·0.2 + 100·0.3 = 40 + 10 + 20 + 30 = 100

Complete 03

PA1: 10
PA2: 10
PA3: 5
PA4: 15
PA5: 5
PA6: 20
Programming Assignments: 10 + 10 + 15 + 5 + 20 = 60

H1: 20
H2: 0
H3: 0
H4: 10
H5: 15
Homework: 20 + 0 + 0 + 10 + 15 = 45

Midterm Exams: 60

Final Exams: 45

Course: 60·0.4 + 45·0.1 + 60·0.2 + 45·0.3 = 24 + 4.5 + 12 + 13.5 = 54

Missing All

PA1:
PA2:
PA3:
PA4:
PA5:
PA6:
Programming Assignments: 0 + 0 + 0 + 0 + 0 = 0

H1: 0
H2: 0
H3: 0
H4: 0
H5: 0
Homework: 0 + 0 + 0 + 0 + 0 = 0

Midterm Exams:

Final Exams:

Course: 0·0.4 + 0·0.1 + 0·0.2 + 0·0.3 = 0

Missing One in One Category

PA1: 20
PA2: 18
PA3:   5
PA4: 15
PA5: 20
PA6: 20
Programming Assignments: 20 + 18 + 15 + 20 + 20 = 93

H1: 20
H2:
H3:   5
H4: 10
H5: 15
Homework: 20 + 0 + 5 + 10 + 15 = 50

Midterm Exams: 80

Final Exams: 75

Course: 93·0.4 + 50·0.1 + 80·0.2 + 75·0.3 = 37.2 + 5.0 + 16.0 + 22.5 = 80.7

Missing One in Each Category

PA1: 20
PA2: 18
PA3:   5
PA4: 15
PA5:
PA6: 20
Programming Assignments: 20 + 18 + 5 + 15 + 20 = 78

H1: 20
H2: 5
H3: 0
H4:
H5: 15
Homework: 20 + 5 + 0 + 0 + 15 = 40

Midterm Exams:

Final Exams:

Course: 78·0.4 + 40·0.1 + 0·0.2 + 0·0.3 = 31.2 + 4.0 + 0.0 + 0.0 = 35.2

Missing Multiple in Each Category

PA1:
PA2: 20
PA3:   5
PA4: 15
PA5:
PA6:
Programming Assignments: 20 + 5 + 15 + 0 + 0 = 40

H1: 20
H2:
H3: 0
H4:
H5: 15
Homework: 20 + 0 + 0 + 0 + 15 = 35

Midterm Exams:

Final Exams:

Course: 40·0.4 + 35·0.1 + 0·0.2 + 0·0.3 = 16 + 3.5 + 0 + 0 = 19.5

Product Domain Glossary

Category	A collection of graded assessments of the same type. For example, “Homeworks” is a category in many courses. Another example of a category is a “Quiz”. The grades on the assessments in a category are often used to calculate a summary grade for that category.
Course Grade	The numeric grade for an entire course. A course grade is often calculated by weighting the summary grades for the categories.
Dropped	A grade is said to be dropped when it is not included in subsequent calculations. For example, it is not uncommon to assign 11 homework assignments and then drop the one with the lowest grade.
Grade	A labeled numeric score on an assessment mechanism (e.g., an exam) or group of assessment mechanisms (e.g., the homework assignments for the semester). The label is referred to as the key and the numeric score is referred to as the value.
Missing Grade	A grade that does not have a numeric score.
Unspecified Weight	A weight is said to be unspecified for a particular grade when there is no weight that corresponds to that grade’s label/key.
Weight	A number that will be multiplied by the numeric score of a grade. Weights are identified using the label/key of the grade.
Weighted Average	The weighted average of a collection of grades is defined as: $$a = \frac{\sum_{i=0}^{N-1} w_i \cdot g_i}{\sum_{i=0}^{N-1} w_i}$$ where g_i denotes the value of grade i (properly accounting for missing values) and w_i denotes the weight associated with grade i (properly accounting for unspecified weights).
Weighted Total	The weighted total of a collection of grades is defined as: $$t = \sum_{i=0}^{N-1} w_i \cdot g_i$$

where g_i denotes the value of grade i (properly accounting for missing values) and w_i denotes the weight associated with grade i (properly accounting for unspecified weights). Note that the weighted total and weighted average are identical if the weights are non-negative and sum to 1.