

Due: Monday, September 19, 2016 by the end of lab

* Reading for this week is the same as 4A along with the additional material in the class notes.

Goals:

By the end of this activity you should be able to do the following:

- Understand the basic syntax of the Java programming language
- Have a better understanding of classes and methods
- Have a better understanding of how to use the viewer canvas in jGRASP

Description:

In this activity you will create two classes. First, you will create a class called Grade which will represent your grade in COMP 1210. The second class, which will be called GradeGenerator, is a driver program that will calculate your course average. Words underlined and highlighted in **blue** represent concepts and terminology that will be important for Exam 1 on Wednesday. **Boxed-in text does not have to be read in lab, but explains the highlighted concepts in context as an exam review.** See pages 3 - 5 for important information related to all future projects (highlighted in red).

Directions:

Part 1: Grade.java (20%) – skeleton code (minimal code required to compile the class and methods with the specified method signatures)

- Create your Grade **class** and **declare** the following **instance variables** using the private **access** (or **visibility**) **modifier**:
 - exam1, exam2, and finalExam: **double** values to hold exam grades
 - activityAvg: **double** that holds the activity average
 - quizAvg: **double** that holds the quiz average
 - projectAvg: **double** that holds the project average
 - studentName: **String** representing student name

Read the boxed-in portions after lab to help study for Exam 1!

Instance variables are declared inside of the class, but not inside of a method. Instance variables are available to any instance method in the class. They should be private to avoid violating encapsulation (see class notes / textbook). When an object is created, space is set aside for the instance variables, and the variables are assigned initial values.

An access (or visibility) modifier specifies whether a member of a class can be accessed and modified from outside of the class (**public**) or inside of the class only (**private**). The **protected** visibility modifier will be further discussed in chapter 9 with respect to inheritance.

Which of your instance variables are **reference types, which are **primitive types**, and what is the difference between the two?**

- You will now declare three constants representing the three possible exams. Declare the three **constants** with public visibility.

```
public static final int EXAM_1 = 1, EXAM_2 = 2, FINAL = 3;
```

Constants are fields in a class that contain a value that never changes. Constant names are capitalized and are declared as **static** (they exist even when an object has not been created and can be accessed using the class name, similar to static methods) and **final** (the value cannot be changed within the program). An example of a public constant: `Math.PI`

Why do public instance variables violate encapsulation while public constants do not?

- Now create private constants that represent the weight of each portion of your grade:

```
private static final _____ EXAM_WEIGHT = 0.15,
                        FINAL_WEIGHT = 0.30, ACT_WEIGHT = 0.05,
                        QUIZ_WEIGHT = 0.10, PROJ_WEIGHT = 0.25;
```
- Add skeleton code (or stub) for a constructor to the Grade class that accepts the student name as a parameter.

```
public _____(_____studentNameIn) {

}
```

A constructor is invoked when an object is created using the new operator and is used to initialize fields using parameter values or default values. Constructors have no return type and always have the same name as the class. Think about why a constructor would have public visibility rather than private visibility.

The formal parameter nameIn is a variable that has local scope and represents input being sent from the calling method. Because it is a local variable, it can only be accessed from within the constructor and will no longer exist when the end of the constructor is reached (space is no longer reserved in memory; see garbage collection).

- Set up **method skeleton code (or stubs)** for the following methods:
 - setLabAverages: no return value; takes 2 double parameters activityAvg and quizAvg. The projectAvg will be set separately below.

```
public void setLabAverages(double activityAvgIn, double quizAvgIn) {

}
```
 - setProjectAvg: no return; takes a double called projectAvgIn as a parameter.

<Do this one on your own>

- setExamScore: no return; takes an int parameter examType and a double examScore.

```
public _____ setExamScore(____ examType, _____ examScoreIn) {

}
```
- calculateGrade: returns a double representing your grade; no parameters.

```
public double calculateGrade() {
    return 0.0;
}
```
- toString: String return and no parameters.

```
public String toString() {
    return "";
}
```

Compile your program. If this were a project, this is the point where you would submit to the “Skeleton Code (Ungraded)” Web-CAT assignment to check the correctness of your method headers.

Part 2: Grade.java – Completing the constructor and instance methods

Completing the Constructor (10%)

- The formal parameter String studentNameIn is considered to be a local variable that will not be accessible after the constructor ends. You need to assign the studentNameIn to the instance variable studentName using an assignment statement within the constructor.

```
studentName = _____;
```

Examples of declaration, assignment, and initialization.

Declaration:

```
int aNum;
```

Assignment:

```
aNum = 1;
```

Declaration and
initialization:

```
int bNum = 10;
```

Completing the Method: toString (10%)

- Complete the toString method so that it returns a String representation of a Grade object:

```
return "Name: " + _____ + "\n"
      + "Course Grade: " + _____;
```

- Test your constructor and toString in the interactions pane:

```
▶ Grade grade = new Grade("Pat");
```

```
▶ grade
```

```
[ Name: Pat
  Course Grade: 0.0
```

```
▶
```

Invokes the toString method and prints the return.

Completing the Set Methods: setLabAverages, setExamScore, setProjectAverage (30%)

- In your setLabAverages method, add the following code to store your activity and quiz average.

```
_____ = activityAvgIn;
_____ = quizAvgIn;
```

- In your setProjectAverage, set the value of the instance variable representing project average.

```
_____ = projectAvgIn;
```

- Your setExamScore method should use the public constants to decide which of the exam score instance variables to set when the method is called: exam1, exam2, or finalExam.

```
if (examType == EXAM_1) {  
    exam1 = examScoreIn;  
}  
else if (examType == EXAM_2) {  
    exam2 = examScoreIn;  
}  
else if (examType == FINAL) {  
    finalExam = examScoreIn;  
}
```

Typically, this set method would include a boolean return and would return false if examType was not 1, 2, or 3; however, for brevity in this activity the method return type is void.

Completing the Method: calculateGrade (10%)

- Add code to calculate your grade. The weight constants are not necessarily useful to users of the Grade class, so they are private rather than public.

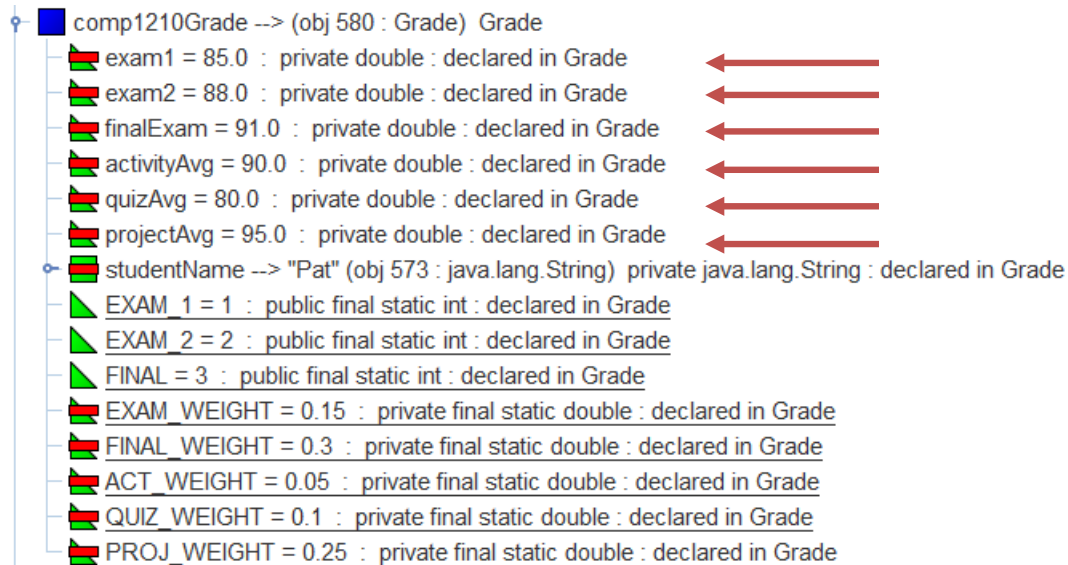
```
double grade = exam1 * EXAM_WEIGHT + exam2 * EXAM_WEIGHT  
              + finalExam * FINAL_WEIGHT  
              + activityAvg * ACT_WEIGHT  
              + quizAvg * QUIZ_WEIGHT  
              + projectAvg * PROJ_WEIGHT;  
return grade;
```


Note that the grade you are calculating is not an instance variable. In general you should try to keep as many variables local as possible to avoid method dependencies and logic errors. This is especially true for values that are calculated from field values. It is usually preferable to calculate a value when needed rather than attempt to update it each time any of several fields it depends on is updated.

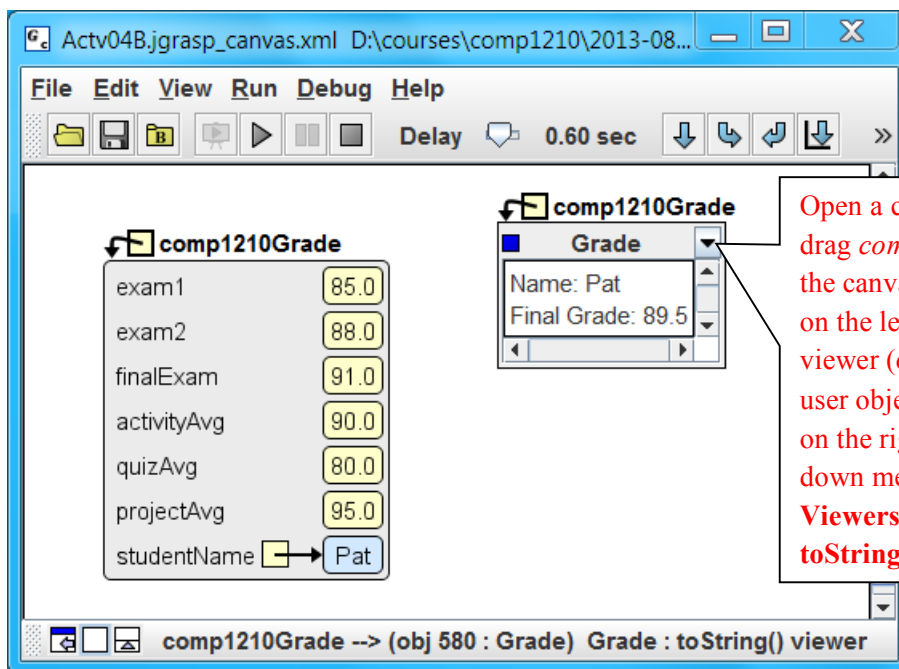
- Test your set methods in the interactions pane:
 - ▶ comp1210Grade.setLabAverages(90, 80);
 - ▶ comp1210Grade.setProjectAvg(95);
 - ▶ comp1210Grade.setExamScore(Grade.EXAM_1, 85);
 - ▶ comp1210Grade.setExamScore(Grade.EXAM_2, 88);
 - ▶ comp1210Grade.setExamScore(Grade.FINAL, 91);
 - ▶ comp1210Grade.calculateGrade()
89.5

Project note:
Always use public constants in your driver programs rather than literal values whenever possible. Points will be deducted for using literal values.

In the workbench to the top left-hand side of jGRASP, open the myGrade object and make sure that your instance variables have been set correctly.



You can also open a canvas window (click Open Canvas button  on the Workbench tab) and then drag comp1210grade onto the canvas. Be sure to change the viewer to “Basic” if this is not already selected. Note the Basic view does not display the static variables/constants but rather only the instance variables. In the canvas below, comp1210Grade has been dragged onto the canvas a second time. The first one is set to the **Basic viewer** and the second one set to the **toString() viewer**, which displays the toString value to the object.



“Set” methods are [mutator methods](#) and modify your object's attributes. “Get” methods are [accessor methods](#) that return the specified attribute.

Project note: As stated in Activity 4A, your *get* methods should not modify an object's instance data, but should return their values.

Project note: Do not depend on the toString method to test your other methods; for example, to test calculateGrade you should invoke calculateGrade directly.

There may be instances in which you want to convert a number from one type to another, which can be done via [assignment conversion](#), [promotion](#), or [casting](#). All of the numbers were doubles in the above example, but conversion may be needed in when an expression contains integers and [integer division](#) is unwanted.

Question: Suppose that aNum is an integer. Modify the following code 3 different times to using assignment conversion, promotion, and casting to produce a correct output (assume that integer division is not desired).

```
int aNumTimes2 = aNum * 2;
double result = aNumTimes2 / 3;
```

**** Important project note:** to score full points on the projects, you should use constants in your code rather than just the value itself (this avoids the use of “magic numbers” and is considered good programming practice). Constant identifiers should always be capitalized. For example, if you had an integer of value 1 that is used represent the color red, you should not use the value 1 throughout your code. Instead, you should have a constant named RED set to the value of 1 and refer to the constant throughout your code.

- **Constants should be public if** they are useful outside of the class (for example, to allow a user to change an exam category without having to memorize that regular exam = 1 final = 2).
- **Constants should be private if** they are only useful inside of the class (for example, in a bowling class the total number of pins is 10; the user would not have to necessarily know or use this value, but the class itself would have to use it to calculate the score).

Part 3: Driver Program (20%)



- Download GradeGenerator.java and GradeGenerator.jgrasp_canvas.xml from the lab web page and save them in the same folder as your Grade.java.
- Complete the main method in GradeGenerator by invoking the methods required set the fields for grades in the Grade object that is created assigned to the variable *comp1210Grade*.

When setting the scores for exam1, exam1 and finalExam, do NOT use literal values to set the exam grades. Instead use the appropriate constants.

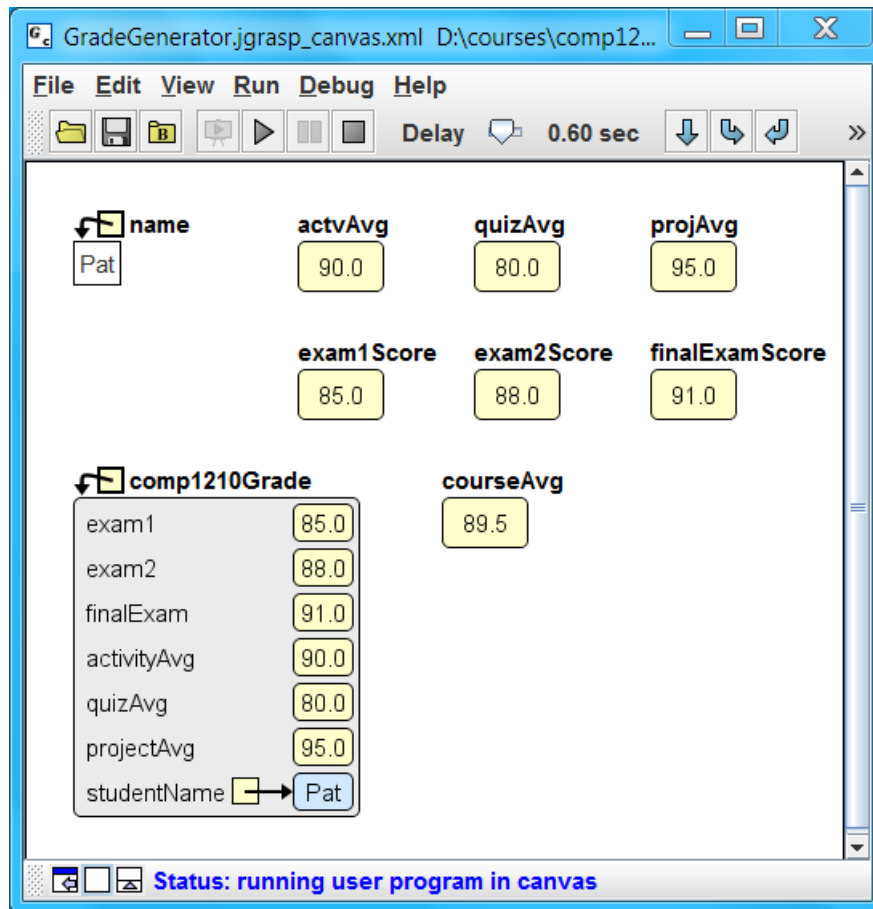
Bad: `comp1210Grade.setExamScore(1, exam1Score);`

Good: `comp1210Grade.setExamScore(Grade.EXAM_1, exam1Score);`

After completing the main method GradeGenerator.java, compile and run the program to test it.

- Run the program using the canvas file you downloaded (GradeGenerator.jgrasp_canvas.xml) by clicking the Run in Canvas button  on the desktop toolbar. After the canvas window opens, click the play button  to begin stepping in the program. Note that the variables on the canvas were dragged from the Debug tab during a previous run. Below is an example of the canvas

window for “Pat Smith-Jones” prior to setting any of the grades in *comp1210Grade*. You may need to reposition and/or resize one or more of individual viewers on the canvas depending on your screen’s pixel density. Be sure to demo this for your TA.



Disclaimer: This activity is not comprehensive in its presentation of Exam 1 concepts.