Due: Monday, September 17, 2012 by the end of lab
* Reading for this week is the same as 4A along with the additional material on the lecture slides.

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

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 final 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%)
- Create your Grade **class** and **declare** the following **instance variables** using the private **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!

Declaring an instance variable sets aside space in memory to hold a value when an object is created, but does not assign a specific value. 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 book / slides).

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.

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

- You will now create two constants representing the two possible types of exam (regular exam = 0, final exam = 1). Declare the two **constants** with public visibility.

```
public static final int EXAM_1 = 0, EXAM_2 = 1, FINAL = 3;
```

Cosntants are fields in a class that contain a value that never changes. Contant names are capitalized and are declared as **static** (they exist even when an object has not be 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 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 = 0.3, LAB_WEIGHT = 0.2,  PROJ_WEIGHT = 0.2;
```

- Add a **constructor** that accepts the student name as a **parameter**.

```
public _______(_______ nameIn)
{

}
```

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 represent 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 stubs** for the following methods:
  - setLabAverages: no return; takes 2 double parameters activityAvg and quizAvg. The projectAvg will be set separately below.

```
public void setLabAverages(double activityAvgIn,
        double quizAverageIn) {
}
```

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

```
public _______ setExamScore(_____ examType,
        _______ examScore) {
}
```

  - setProjectAverage: no return; takes a double called average as a parameter.

    <Do this one on your own>

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

```
public _______ calculateGrade() {
        return 0;
}
```

  - toString: String return and no parameters.

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

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

Completing the Constructor (10%)

- Because nameIn is a **local variable** that will not be accessible at the end of the constructor, you will need to have the instance variable name reference the nameIn String object 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: " + __________ + "\r\n"
     + "Final Grade: " + calculateGrade();
```

- Test your constructor and toString in the interactions pane:

```
Grade grade = new Grade("Lauren");
grade
Name: Lauren
Final 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;
__________ = quizAverageIn;
```

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

```
__________ = average;
```

- The setExamScore will use the public constants to decide which exam's score to set:

```
if (examType == EXAM_1) {
    exam1 = examScore;
}
else if (examType == EXAM_2) {
    __________ = examScore;
}
else if (examType == FINAL) {
    __________ = examScore;
}
```

Project note: Use constant fields rather than literal values in your project code for full credit.

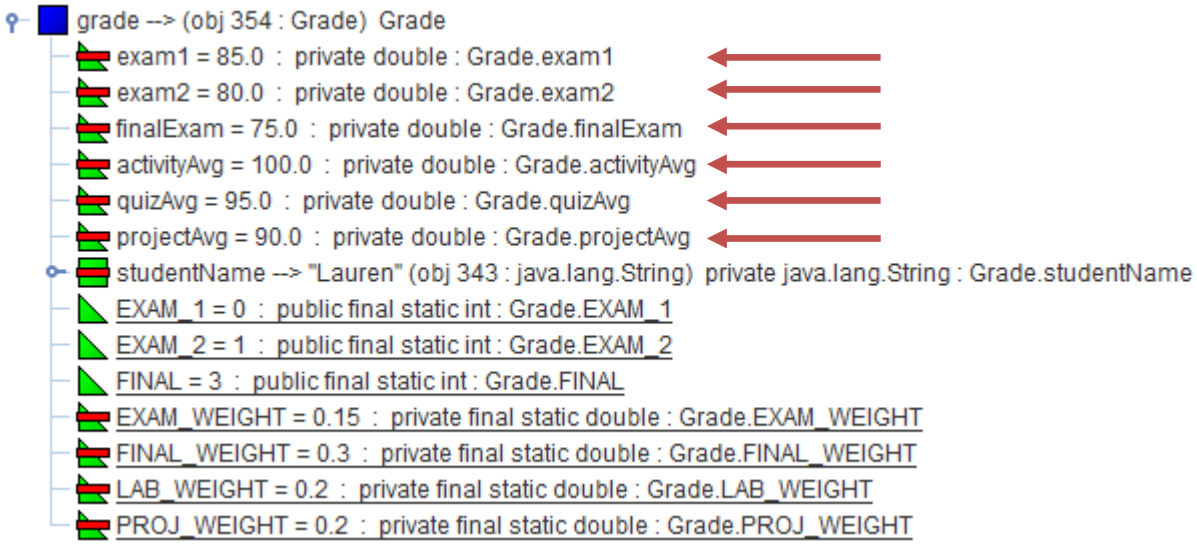
Typically, this set method would include a boolean return and would return false if examType was not 1-3.

- Test your set methods in the interactions pane:

```
Grade grade = new Grade("Lauren");
grade.setLabAverages(100, 95);
grade.setProjectAverage(90);
grade.setExamScore(Grade.EXAM_1, 85);
grade.setExamScore(Grade.EXAM_2, 80);
grade.setExamScore(Grade.FINAL, 75);
```

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 grade object and make sure that your instance variables have been set correctly.



"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.

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 average 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 one of several fields it depends on is updated. However, this is a design decision that could go either way.

- Test the getGrade method in the interactions pane:

```
Grade grade = new Grade("Lauren");
grade.calculateGrade()
0.0
grade.setLabAverages(100, 95);
grade.setProjectAverage(90);
grade.setExamScore(Grade.EXAM_1, 85);
grade.setExamScore(Grade.EXAM_2, 80);
grade.setExamScore(Grade.FINAL, 75);
grade.calculateGrade()
84.75
```

Project note: Do not use or invoke the toString method to test your other methods; for example, to test calculateGrade you should invoke calculateGrade only before displaying the toString output.

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 other instances to prevent unwanted **integer division**.

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 doubledNum = aNum * 2;
double result = doubledNum / 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 wouldn't have to necessarily know or use this value, but the class itself would have to use it to calculate the score).

Part 2: Driver Program (20%)

- Download and complete the driver program called GradeGenerator. Do NOT use literal values to set exam grades.

```
Bad:  aGradeObj.setExamScore(1, 85);
Good: aGradeObj.setExamScore(Grade.EXAM_1, 85);
```

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