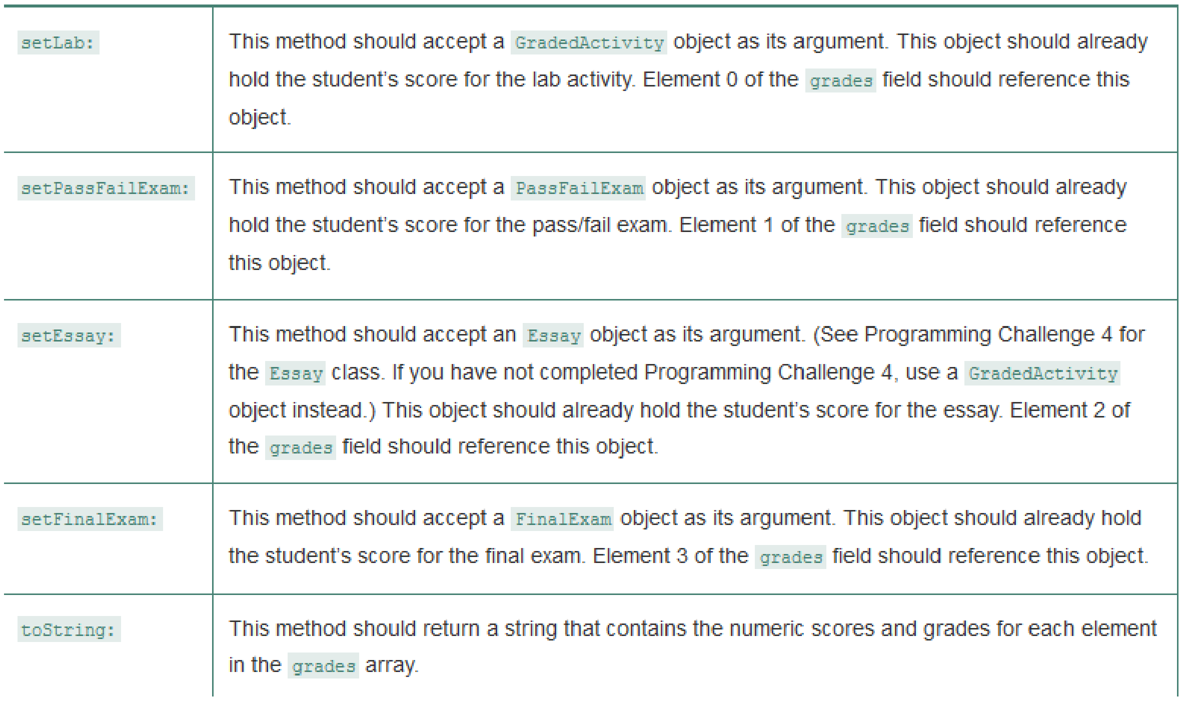
5. Course Grades (Ch 10)

In a course, a teacher gives the following tests and assignments:

A lab activity that is observed by the teacher and assigned a numeric score.
A pass/fail exam that has 10 questions. The minimum passing score is 70.
An essay that is assigned a numeric score.
A final exam that has 50 questions.

Write a class named CourseGrades. The class should have a GradedActivity array named grades as a field. The array should have four elements, one for each of the assignments previously described. The class should have the following methods:



`setLab:`	This method should accept a `GradedActivity` object as its argument. This object should already hold the student's score for the lab activity. Element 0 of the `grades` field should reference this object.
`setPassFailExam:`	This method should accept a `PassFailExam` object as its argument. This object should already hold the student's score for the pass/fail exam. Element 1 of the `grades` field should reference this object.
`setEssay:`	This method should accept an `Essay` object as its argument. (See Programming Challenge 4 for the `Essay` class. If you have not completed Programming Challenge 4, use a `GradedActivity` object instead.) This object should already hold the student's score for the essay. Element 2 of the `grades` field should reference this object.
`setFinalExam:`	This method should accept a `FinalExam` object as its argument. This object should already hold the student's score for the final exam. Element 3 of the `grades` field should reference this object.
`toString:`	This method should return a string that contains the numeric scores and grades for each element in the `grades` array.

Demonstrate the class in a program.

9. BankAccount and SavingsAccount Classes (Ch 10)

Design an abstract class named BankAccount to hold the following data for a bank account:

- Balance
- Number of deposits this month
- Number of withdrawals

- Annual interest rate
- Monthly service charges

The class should have the following methods:

Constructor:	The constructor should accept arguments for the balance and annual interest rate.
`deposit:`	A method that accepts an argument for the amount of the deposit. The method should add the argument to the account balance. It should also increment the variable holding the number of deposits.
`withdraw:`	A method that accepts an argument for the amount of the withdrawal. The method should subtract the argument from the balance. It should also increment the variable holding the number of withdrawals.
`calcInterest:`	A method that updates the balance by calculating the monthly interest earned by the account, and adding this interest to the balance. This is performed by the following formulas: *Monthly Interest Rate* = (*Annual Interest Rate* / 12) *Monthly Interest* = *Balance* * *Monthly Interest Rate* *Balance* = *Balance* + *Monthly Interest*
`monthlyProcess:`	A method that subtracts the monthly service charges from the balance, calls the `calcInterest` method, and then sets the variables that hold the number of withdrawals, number of deposits, and monthly service charges to zero.

Next, design a SavingsAccount class that extends the BankAccount class. The SavingsAccount class should have a status field to represent an active or inactive account. If the balance of a savings account falls below $25, it becomes inactive. (The status field could be a boolean variable.) No more withdrawals may be made until the balance is raised above $25, at which time the account becomes active again. The savings account class should have the following methods:

`withdraw:`	A method that determines whether the account is inactive before a withdrawal is made. (No withdrawal will be allowed if the account is not active.) A withdrawal is then made by calling the superclass version of the method.
`deposit:`	A method that determines whether the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $25, the account becomes active again. A deposit is then made by calling the superclass version of the method.
`monthlyProcess:`	Before the superclass method is called, this method checks the number of withdrawals. If the number of withdrawals for the month is more than 4, a service charge of $1 for each withdrawal above 4 is added to the superclass field that holds the monthly service charges. (Don't forget to check the account balance after the service charge is taken. If the balance falls below $25, the account becomes inactive.)