Problem One: Hiking

Write a class named Hike and a class named HikeDriver.

The class Hike has

1. Fields. Three fields which are the name of the hiker, the name of the trail, and the distance hiked.

2. Constructors.
   (a) One constructor that has three parameters to initialize the fields.
   (b) One constructor that has no parameters and initializes the fields to default values.

   (a) a getter (accessor) method for each field that just returns the field
   (b) a setter (mutator) method for each field that sets the field to the value of the parameter
   (c) a howFar method that displays on the console the value of the hiker name field and then a phrase as specified below. For example, if the value of the name field is “Jill”, then the output is
      i. Jill, that is just a walk!
      if distance is strictly less to 3 miles.
      ii. Jill, that is a good hike!
      if distance is greater than or equal to 3 miles and strictly less than 10 miles.
      iii. Jill, you must be tired!
      if distance is greater than or equal to 10 miles.
   (d) a nickname method that prints (that is, displays on the console) a nickname for the hike. If the hiker’s name is at least five characters long, then the nickname is the first five characters of the hiker’s name followed by the first three characters of the trail’s name. If the hiker’s name is less than five characters long, then instead display on the console an error message saying “Hiker name is too short.”.
The class `HikeDriver` has a public, static, void method named `main` that

1. calls each constructor and method at least once
2. if a method returns something, then display that output on the console
3. call the `howFar` method at least three times and so that each of the three cases of output is tested
4. call the `nickname` method at least two times and so that each of the two cases of output is tested

Remember to submit both `Hike.java` and `HikeDriver.java`. 
Problem Two: Balanced Parentheses

Write a program that determines whether a given line of input has a balanced number of parentheses or not. The input may contain more symbols than parentheses, but you are only concerned about the parentheses.

Here is the definition of balanced parentheses:

1. As you move from left to right through the input string, you should never encounter more right parentheses than left parentheses.

2. The total number of left parentheses must equal the total number of right parentheses.

Here are some examples of strings with balanced parentheses:

```
abcedfg
(abc)(def)
(abcdefghekjsl)
()
{(abc(ef)(gh)(ij)(hjklk)(jkladk))((fkjdf)(kdjfd))}
()
()
()
()
()
```

Here are some examples of strings that do not have balanced parentheses:

```
(abscd
(dkjdjd)
(fdadf))
(ka)ad)(jk(dj)
```

Your program must execute like the two examples below. Your program prompts for input and then reads in the line of input. Your program then states whether or not that line has a balanced set of parentheses or not using the wording below.

Example Run:

Please enter a string: (abc)(def)
The string (abc)(def) has balanced parentheses.

Example Run:

Please enter a string: {fdadf})
The string {fdadf}) does not have balanced parentheses.
Problem Three: University

Write a University class and a UniversityDriver class.

The University class models the progression of students through four years at a university. Your University class should have the following constructors, methods, and fields:

- **Fields**
  - a String field to hold the university’s name
  - a field called students that holds the numbers of freshmen, sophomores, juniors, and seniors. The field can either be an ArrayList of Integers or a 1-dimensional int array.

- **Constructors**
  - one constructor with a String parameter and an int parameter. It assigns the value of the String to the name field for the university. It assigns the value of the int parameter to every element in the array field using a loop.

- **Methods**
  - a fallEnroll method with one int parameter. The fallEnroll method has the effect of promoting everyone up to the next level (Freshman to Sophomore, and so on), with some built-in attrition. It works as follows (it must use a loop) (the examples below assume you are using a one-dimensional array; you can use an ArrayList instead):
    * it takes the value in students[2], multiplies it by 0.9, and stores it in students[3] (hint: 0.9 is a double value, so you will have to cast the product to put it into an int variable; alternatively, you can just multiply by the integer 9 and divide by the integer 10)
    * it does the same thing for students[1] and students[0]
    * the parameter becomes the new value for students[0]
  - a private total helper method that takes no argument and returns the total enrollment of the university. This is used in the printUniv method below.
  - A printUniv method that prints out a description of the university, to include its name, its freshman, sophomore, junior, and senior enrollment, and the total enrollment

Write a UniversityDriver class that has a main method that does the following.

- It constructs two University objects and establishes them with 2000 and 2500 freshmen, respectively (you may choose their names).
- It then prints to the monitor (using the printUniv method) the initial enrollment figures of each university.
- It then uses a loop to do the following:
  - It invokes the fallEnroll method on each object, assume that freshman enrollment increases by 100 each year for the first object and decreases by 100 each year for the second object.
Thus, the first time through the loop, the first university has enrollment of 2100, the second university has enrollment of 2400.

The second time through the loop, the first university has enrollment of 2200, the second university has enrollment of 2300, and so on.

It invokes the printUniv method on each object to describe the current enrollment figures for each university. The loop should model 3 years of enrollment (i.e., it should iterate 3 times).

Here is what the output of executing the main method of the UniversityDriver should look like:

Here are the enrollment figures for Western:
Total enrollment: 8000

Here are the enrollment figures for StateU:
Freshmen: 2500 Sophomores: 2500 Juniors: 2500 Seniors: 2500
Total enrollment: 10000

Here are the enrollment figures for Western:
Freshmen: 2100 Sophomores: 1800 Juniors: 1800 Seniors: 1800
Total enrollment: 7500

Here are the enrollment figures for StateU:
Freshmen: 2400 Sophomores: 2250 Juniors: 2250 Seniors: 2250
Total enrollment: 9150

Here are the enrollment figures for Western:
Freshmen: 2200 Sophomores: 1890 Juniors: 1620 Seniors: 1620
Total enrollment: 7330

Here are the enrollment figures for StateU:
Freshmen: 2300 Sophomores: 2160 Juniors: 2025 Seniors: 2025
Total enrollment: 8510

Here are the enrollment figures for Western:
Total enrollment: 7439

Here are the enrollment figures for StateU:
Freshmen: 2200 Sophomores: 2070 Juniors: 1944 Seniors: 1822
Total enrollment: 8036

Remember to submit both University.java and UniversityDriver.java.
Problem Four: Alternate

Write a Java class named Alternate and a Java class named AlternateDriver. The class Alternate that has a method that has the method header

```java
public String genAlternate(int numZeroes, int numRows)
```

that returns a string representing numRows rows of alternating ones and zeroes starting with a one and ending with a one. The parameter numZeroes indicates the number of zeroes and so the number of ones is one more. Below is an example of what the string your method generates must look assuming that the parameter numZeroes has the value of 4 and the parameter numRows has the value of 2.

```
101010101
101010101
```

The class AlternateDriver has a public, static, void method named main that in order does

1. creates an instance of class Alternate and a variable to hold a reference to that instance.

2. calls the genAlternate method of that instance of class Alternate with the parameters 4 and 2 for numZeroes and numRows, respectively. Display the string returned from the method call on the console.

3. calls the genAlternate method of that instance of class Alternate with the parameters 6 and 3 for numZeroes and numRows, respectively. Display the string returned from the method call on the console.

Remember to submit both Alternate.java and AlternateDriver.java.