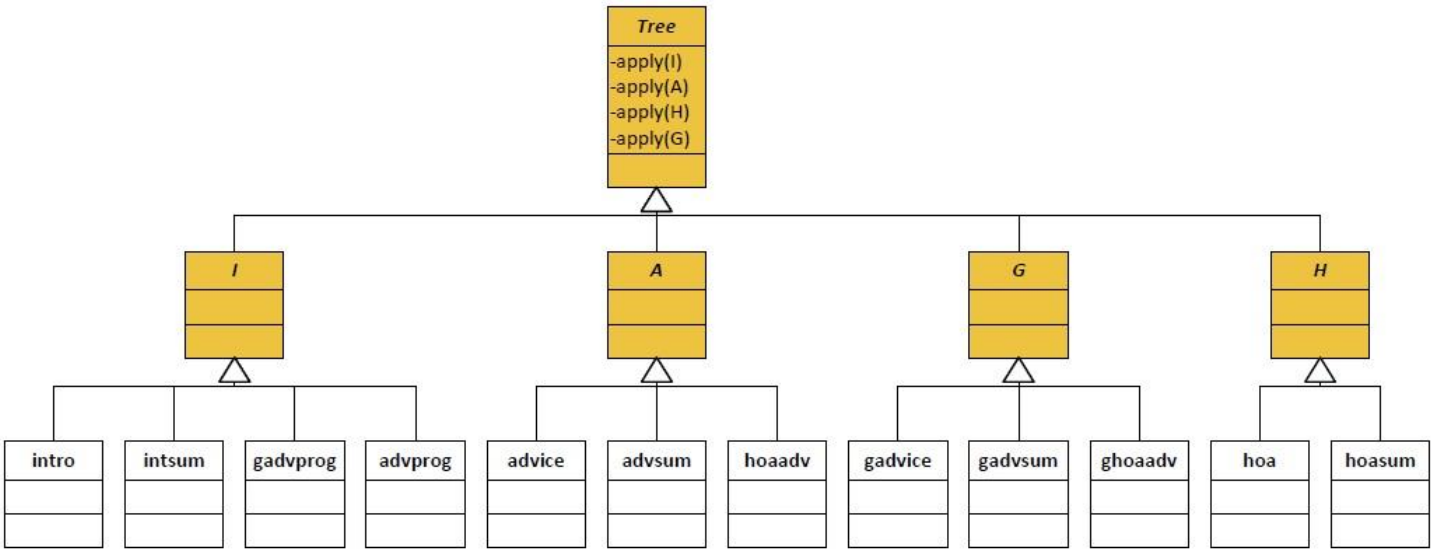
CSC 402-02 Assignment #3

```
Original Due: 2:00 PM, Tuesday, November 15
    Extended: 2:00 PM, Tuesday, November 22
```

You must complete this assignment by yourself. You cannot work with anyone else in the class or with someone outside of the class. You are not allowed to copy solutions from the world wide web. The code you write must be your own.

Provided Files:

- A.java
- advice.java
- advprog.java
- advsum.java
- G.java
- gadvice.java
- gadvprog.java
- gadvsum.java
- ghoaadv.java
- H.java
- hoa.java
- hoaadv.java
- hoasum.java
- I.java
- intro.java
- intsum.java
- Main.java
- SwingApp.java
- Tree.java
- Visitor.java



Description: You are given a simple Java GUI program called Quark. A class diagram of the Quark is shown at the first page. The root is an abstract class called `Tree`, which has 4 methods (there are more methods, but only these are shown):

```
Tree apply(I)
Tree apply(A)
Tree apply(H)
Tree apply(G)
```

All classes in this hierarchy implement or inherit these methods. The semantics (meaning) of these methods is not significant for this assignment, only to the extent that one of these methods will be involved with a Visitor pattern. For this assignment, your task is to make a Visitor design pattern for method `apply(I)` following each of the **steps #1-5** below.

Step #1

1. Nothing. I have already created a Visitor class for you.

Step #2

1. Retrofit a Singleton design pattern into class `Visitor` (in Visitor.java).
2. Make sure that your refactored code compiles and runs.
3. Create a zip file (**two.zip**) containing all your current Java files (i.e., twenty .java files).

Step #3

1. Add a `Visitor`-type parameter to all `apply(I)` methods and use the `Visitor`-type singleton instance as a default value.
2. Make sure that your refactored code compiles and runs.
3. Create a zip file (**three.zip**) containing all your current Java files (i.e., twenty .java files).

Step #4

1. Perform the Move-Instance-Method-with-Leaving-a-Delegate-Behind refactoring on all concrete (not abstract/interface) `apply(I)` methods.
2. Make sure that your refactored code compiles and runs.
3. Create a zip file (**four.zip**) containing all your current Java files (i.e., twenty .java files).

Step #5

1. Rename all delegate methods (created by **Step #4**) to "`accept`" without breaking run-time polymorphism.
2. Rename all `apply(I)` methods in class `Visitor` to "`visit`"
3. Make sure that your refactored code compiles and runs.
4. Create a zip file (**five.zip**) containing all your current Java files (i.e., twenty .java files).

You must use the shell files for this assignment.

Submission: your files named **two.zip**, **three.zip**, **four.zip**, **five.zip**

General Programming Assignment Requirements:

- Classes must be in the default (no package statement) unless otherwise specified. You will lose all points if you put a package statement in your program.

- If your program does not compile or does not run, you will lose all points.
- If you submit the wrong file, you will lose all points.
- You must fill in the header for every file you submit. Otherwise, you will lose all points.

Checklist: Did you remember to:

- work on the programming assignment individually?
- fill in the header in your `Visitor.java`?
- ensure your program does not suffer a compile error or runtime error?
- ensure your program creates the correct output and that it matches the expected output exactly?
- properly indent your source code so that your indenting is readable and consistent?
- use good names for variables to make your program easy to understand?
- turn in your Java source code in files named **two.zip**, **three.zip**, **four.zip**, **five.zip** through D2L?