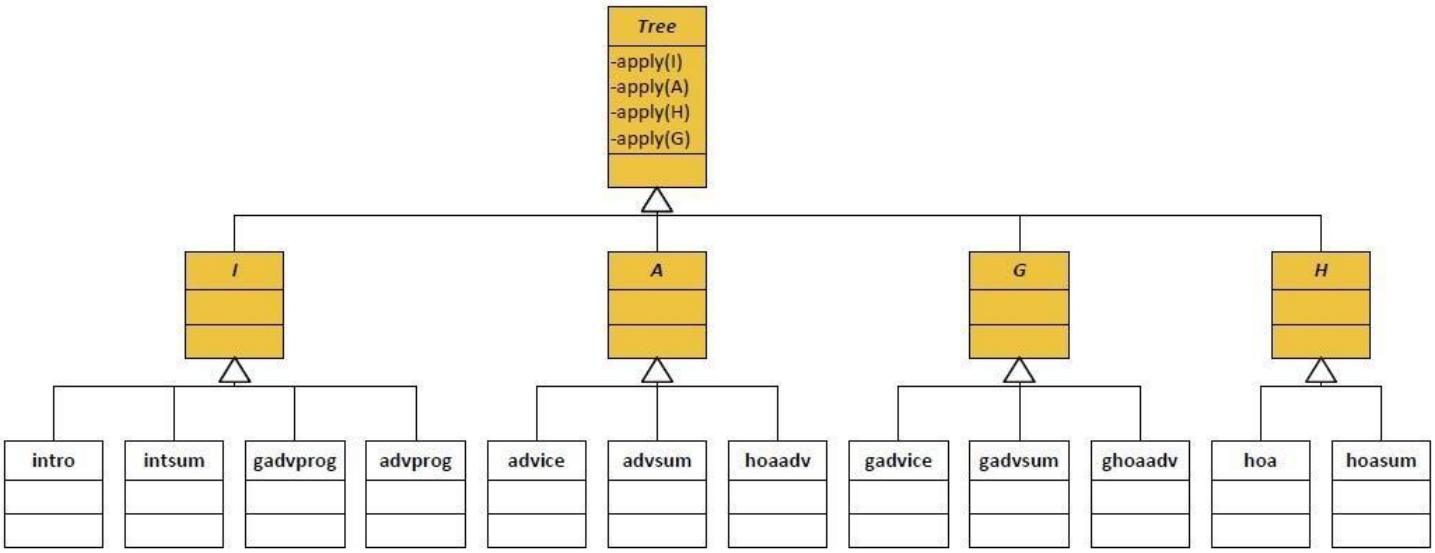
Provided Files:

- 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 an empty 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 directory file (**two**) 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 directory file (**three**) 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 directory file (**four**) 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 directory file (**five**) containing all your current Java files (i.e., twenty .java files).