1. Write a complete Java abstract class `VotingGroup`. The purpose of the class is to determine how many votes it takes to decide something in an organization.

   (a) Each instance of the class stores the total number of members in the group as an integer.

   (b) The class’s constructor has this total as a parameter.

   (c) There is an unimplemented method `votesNeeded` that takes no parameters and returns an integer (count of people who must vote Yea).

   (d) There is a static method that, given a number of people, returns the minimum number that would constitute a simple majority. You can determine the minimum integer $m$ such that $2 \times m > n$ using only integer arithmetic in Java. Consider even and odd values of $n$. Set up the method so that it can only be called by `VotingGroup` and its descendants.

   (e) Finally there is a public method `passed` that takes the number of Yea votes on some resolution and returns `true` if and only if the number of Yea votes is enough to pass the resolution.
2. If you write a subclass `Club` that extends `VotingGroup`, what is/are the condition(s) that must be met so that the `Club` class can be instantiated (i.e., using `new`)?

3. (T/F) In UML class diagrams, the same kind of arrow is used when a class refers to an interface it implements, and when it refers to another class the original class extends.

4. Class `B` extends class `A`. Check off the statements below that are true in this setup.

   • `B` may redefine (re-implement) some of the methods in `A`.
   • `B`’s methods have access to fields and methods declared `private` in `A`.
   • `B` may declare totally new methods beyond those declared in `A`.
   • `B` must redeclare the fields declared in `A` if it wants access to them. (We’re not talking public/private issues here.)
   • `B` may declare totally new fields beyond those declared in `A`.
   • A class can only extend another class if the other class is abstract.

5. You may have noticed that an abstract class with no declared fields and no non-abstract methods looks pretty much identical to an interface. Why have both?
6. What, if anything, is wrong with the following code snippet that is supposed to work with a
generic ArrayList of Strings called names?

```java
ArrayList names = new ArrayList();
names.add('a');
names.add(10);
names.add("string");
String str = names.get(2);
System.out.println("Third element: " + str);
```