Feature Accessibility, Overloading, and Overriding

Purpose:
The purpose of this lab is to help you understand feature accessibility, method overloading, and method overriding.

Setup:
Create a directories named Lab26/accessibility0 and Lab26/accessibility1 in your Java directory. Copy the files from ~labCourse/Labs/Lab26/accessibility0 and ~labCourse/Labs/Lab26/accessibility1 to your accessibility0 and accessibility1 directories.

Feature accessibility – public, restricted, private:
You can get the full story from the text. We give an abbreviated version here.
First, if a class is not marked public, then the class and its features are accessible only from within its own package.
Accessibility to the features (both methods and components) of a class can be restricted in the following ways.
If a feature is labeled public, e.g.,

    public void someMethod ()

then access is not restricted.
If a feature is not labeled, e.g.,

    void someMethod ()

then access is restricted to the package.
If a feature is labeled private, e.g.,

    private void someMethod ()

then access is restricted to the class containing the feature definition.

• Examine the definition of the class Lab26/accessibility1.AClass. Note how the methods are specified.
• Examine the definition of the class Lab26/accessibility1.BClass. Which method invocations are not valid?
• Compile BClass.java and confirm your observations.
• Examine the definition of the class Lab26.accessibility1.AXClass. Which method invocations are not valid?
• Compile AXClass.java and confirm your observations.
• Comment out the invalid method invocations in `Lab26.accessibility1.BClass` and `Lab26.accessibility1.AXClass`. Test with `Lab26.accessibility0.Lab6Test1`.

**Feature accessibility – protected:**

If a feature is defined as protected, then it is accessible from any class in the same package as the class in which the feature is defined.

A subclass inherits protected features from its parent superclass, whether it is in the same package as its superclass or not. The inherited features are accessible from the subclass.

Furthermore, a class can access a protected feature in a subclass instance, as the subclass inherits the feature through the class. For example, suppose class `C` has protected member `p`. From class `C`, the member `p` of any `C` instance, or any subclass of `C` instance, can be accessed.

Note that `Lab26.accessibility0.P4Class` extends `Lab26.accessibility0.P3Class` extends `Lab26.accessibility1.P2Class` extends `Lab26.accessibility0.P1Class`. All but `P2Class` are in the same package.

![Class Diagram]

`P2Class` defines a protected feature that is inherited by `P3Class`, and `P4Class`. We are going to try to access the protected feature inherited by a `P3Class` instance from each class in the hierarchy.

• Look at the definitions of the classes, including `Lab6Test2`. Which invocations of `protectedMethod` do you think are invalid?

• Compile `Lab6Test2` and confirm your observations.

• Move `P4Class` to the package `Lab26.accessibility0`. (Don’t forget to edit the package and import statements!) Remove the `class` files from your `accessibility0` and `accessibility1` directories, and recompile `Lab6Test2`.

• What do the results of this compilation tell you?
• Do the same for P1Class. (Don’t forget to remove the class files.)

Overloading:

A class can have several methods with the same name, as long as all have different signatures: that is, different number and/or types of parameters. For instance, a class could contain both these methods

```
public void sayHi ()
public void sayHi (String message)
```

since the first has no parameters and the second has one. But a class could not contain the following two methods, even though the return types are different:

```
public char grade (Student s)
public double grade (Student s)
```

Having different methods in the same class with the same name is referred to as overloading.

• Look at the definition of the class Lab26.accessibility1.CClass. Note that there are three distinct methods with the same name.

• Look at Lab26.accessibility0/Lab6Test3. What do you expect the output from this test to be?

• Compile and run Lab26.accessibility0/Lab6Test3 and verify your conclusions.

Overriding:

If a class extends another, it inherits all the methods of its parent class. The subclass, however, can redefine some or all of the methods it inherits. This is called overriding.

• Look at the definition of the class Lab26.accessibility1.CXClass. This class inherits the three sayHi methods from CClass, but overrides one of them.

• Look at Lab26.accessibility1.Lab6Test4. What do you think the output from this will be?

• Compile and run Lab26.accessibility1.Lab6Test4. Be sure to note the way in which the CXClass constructor invokes its parent class constructor. Also notice that the CXClass instance always behaves like a CXClass instance even when its accessed through a CClass variable (aC in Lab6Test4), or when it executes an inherited, non-overridden method (sayHi(3)).

Suppose that you declare the following:

```
CClassList list = new CClassList();
```

where CClassList is a list of CClass instances, and add several CClass and CXClass instances to the list:

```
list.add(new CClass());
...
list.add(new CXClass());
...
```

• Explain why it is legal to add either type instance to the CClassList.
Now suppose we write for some legal value of $i$:

    list.get(i).sayHi();

- Explain what will be written.

A class can decide to behave like its parent by using the keyword \textit{super}.

- Look at the definition of \textit{sayHiTwice} in \textit{CXClass}. The first invocation of the method \textit{sayHi} will be done “as its parent would do it.” Compile and run \textit{Lab26.accessibility1.Lab6Test5}, and note the results.

Finally, it is important to note the difference between overloading and overriding. With overloading, a class has several different methods with the same name. With overriding, a method is given different implementations in a class and its parent.

The particular method denoted by a method invocation (overload resolution) is determined at compile time by the \textit{static syntactic type} or the object reference, and the \textit{static syntactic types} of the arguments.

- Look at \textit{Lab26.accessibility1.DClass}, \textit{Lab26.accessibility1.DXClass}, and \textit{Lab26.accessibility0.Lab6Test6}. What do you think the output from \textit{Lab6Test6} will be?

- Compile and run \textit{Lab6Test6} to verify your observations.

\textbf{Post-lab:}

Submit the following, as directed by your lab instructor:

- answers to the questions from this lab;
- listings of your modified \textit{BClass} and \textit{AXClass}. 